£9.95 | €10.95 | $12.95

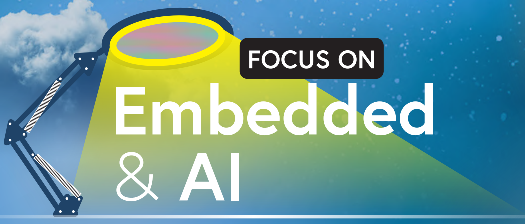
elektor MAG
design > share > earn

520
* SINCE 1961 *
MARCH & APRIL 2023
ELEKTORMAGAZINE.COM

ESP32-Based

Alarm Clock

Connects to Time Server, Controls Radio & TV



FOCUS ON

Embedded & AI

What's Next for AI and Embedded Systems?

Tools, Platforms, and Writer Replacements

RP2040 PIO in Practice

Experiments Using the Programmable I/O

Join us at

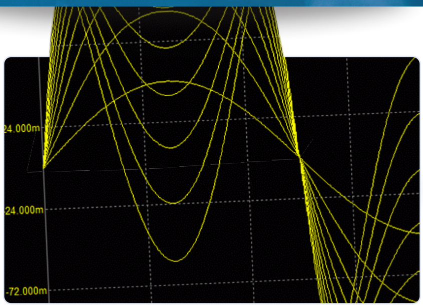
embeddedworld

Exhibition&Conference

... it´s a smarter world

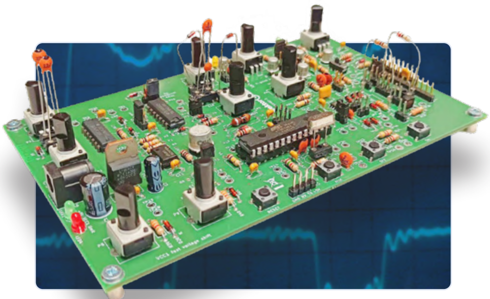
Video Output with Microcontrollers
VGA and DVI Output

p. 92



Circuit Simulation With Micro-Cap First Steps in a Complicated World

p. 48



Poor Man's ChipTweaker
DIY Hardware Attacks

p. 22

MICROCHIP

CURIOSITY NANO

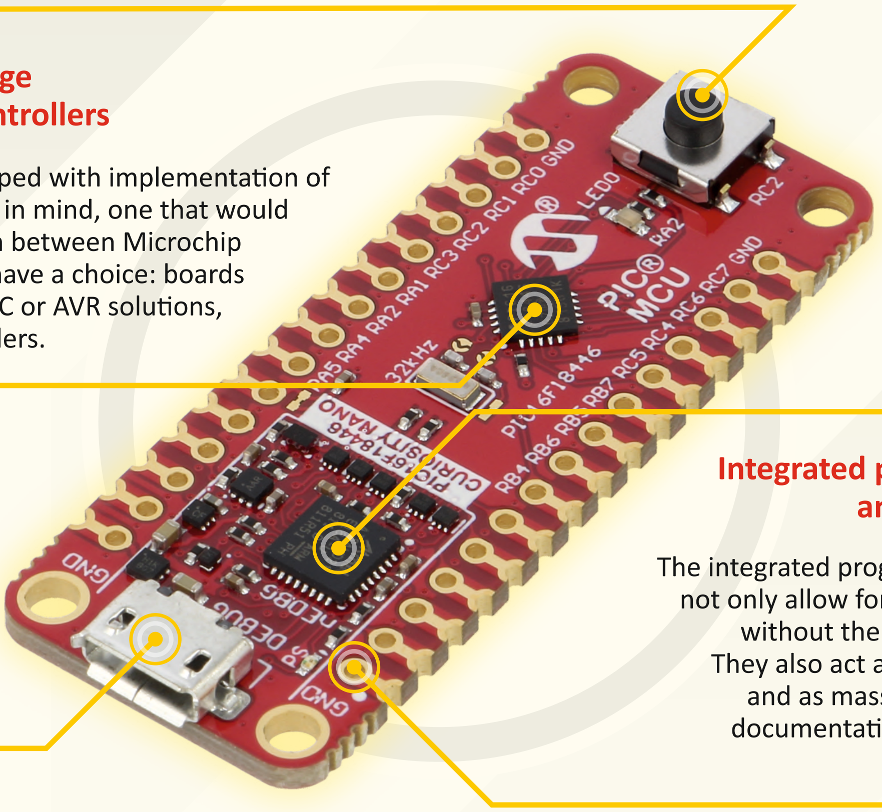
Prototype boards compatible with expansion modules

Basic components

Curiosity Nano boards are equipped with a button, LED indicator, quartz resonator, and programmable voltage regulator. Thanks to that, basic prototyping and starting work with the platform do not require any additional accessories.

Diverse range of microcontrollers

This series was developed with implementation of a normalised platform in mind, one that would facilitate the migration between Microchip products. The clients have a choice: boards equipped with 8-bit PIC or AVR solutions, or 32-bit microcontrollers.



Integrated programmer and debugger

The integrated programmer and debugger not only allow for updating the software without the use of additional tools. They also act as a COM port emulator and as mass storage where the full documentation of the board is kept.

USB interface

In order to develop and launch your code on the MC Nano platform, you just need to connect the board to the USB port of the computer and copy the .hex file onto the device.

Normalised terminals

Independently of the selected model of microprocessor or even its architecture, the basic terminals on the PCB are always in the same place. High level of standardisation limits the costs and time of the prototyping process.

TME®
Electronic Components

EN Łódź, Poland, +48 42 645 54 44, export@tme.eu
PL Łódź, Polska, +48 42 645 55 55, dso@tme.pl
UK Coleshill, Birmingham, United Kingdom, +44 167 579 00 26, office@tme-uk.eu
HU Budapest, Hungary, +36 1 220 67 56, tme@tme.hu
SK Žilina, Slovakia, +421 41 500 20 47, tme@tme.sk
CZ Ostrava, Czech Republic, +420 59 663 31 05, tme@tme.cz
RO Timişoara, Romania, +40 35 646 74 01, tme@tme.ro
DE Leipzig, Germany, +49 341 212 03 40, tme@tme-germany.de
ES Coslada, Madrid, Spain, +34 91 123 47 71, iberica@tme.eu
IT Grassobbio, Bergamo, Italy, +39 035 03 93 111, tme@tme-italia.it
NL Eindhoven, Netherlands, +31 40 737 04 57, tme@tme-benelux.nl
CN Shenzhen, China, +86 755 8666 0094, tme@tme.cn
US Atlanta, USA, +1 (678) 691-2347, +1 (678) 691-5147, tme-us@tme.com

facebook.com/TME.eu
youtube.com/TMElectroniComponent
linkedin.com/company/1350565
instagram.com/tme.eu
twitter.com/tme_eu
tiktok.com/@tme.eu

tme.eu / tme.com

COLOPHON

Elektor Magazine,
English edition

Volume 49, No. 520
March & April 2023

ISSN 1757-0875 (UK / US / ROW distribution)

www.elektor.com
www.elektormagazine.com

Elektor Magazine, English edition
is published 8 times a year by
Elektor International Media

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

Memberships:
E-mail: service@elektor.com
www.elektor.com/memberships

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

www.elektor.com/advertising
Advertising rates and terms available on request.

EDITORIAL

Jens Nickel

International Editor-in-Chief, Elektor Magazine

Thrilling Times

When we started planning this issue several months ago, we could not have known that artificial intelligence (AI) would have so much hype. The magic word here is, of course, ChatGPT, a chatbot that not only answers questions about all kinds of knowledge, but can also churn out stories, song lyrics and much more at the push of a button. Like one of our writers (see page 64), I also tried out ChatGPT. I asked questions about solar island systems and the appropriate electricity storage systems. I was impressed with the results. Not only could the chatbot estimate what the capacity of a LiFePo4 would need to be for a typical two-person weekend home, it also provided me with five suggestions for such a battery, with manufacturer and type, when asked. In addition, ChatGPT was also "aware" of the limitations of its own statements. The bot warned me that capacity calculations depended on energy consumption and module size, and it recommended that I consult a professional before buying.

But, of course, ChatGPT has no awareness or sense of the things he/she/it is outputting. The output is a kind of weighted average of tens of millions of possible statements, all of which can be found somewhere in similar form on the Internet. The linguistic skill is surprising, but almost everything has been heard before in this or a similar way. This "averaging" from a vast body of knowledge often makes ChatGPT's answers seem a bit mushy.

Nevertheless, we at Elektor are fascinated. I can well imagine that the bot could help us and you in the future — for example, in the search for a suitable Elektor article in our archive because, after all, 62 years of concentrated electronics knowledge is stored there. Maybe it could also help in writing boilerplate code, creating parts lists, and designing recurring circuit parts?

At least in the medium term, I don't see it as a danger that ChatGPT will make our jobs — or even your input, dear readers, to our magazine — obsolete. So far, AI doesn't get goosebumps when it discovers a great project or device that could be super useful for its own developments. AI doesn't get an idea in the shower, nor does it write me whimsical emails about how everything used to be better. These emotions are what drive us in the basement lab and when writing articles.

Do you see it differently? What experiences have you already had with AI? And where will it all lead? Write me at editor@elektor.com!

Elektor at embedded world
From March 14 to 16, embedded world will take place in Nuremberg, a must-attend event for all microcontroller and software professionals. As usual, Elektor will be exhibiting. **Visit us at booth 4A-646!**

The Team

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

DEUTSCHE FACHPRESSE

Elektor is a member of VDZ (Association of German Magazine Publishers), which "represents the common interests of 500 German Consumer and B2B publishers."

THIS EDITION

Volume 49, No. 520
March & April 2023

Regulars

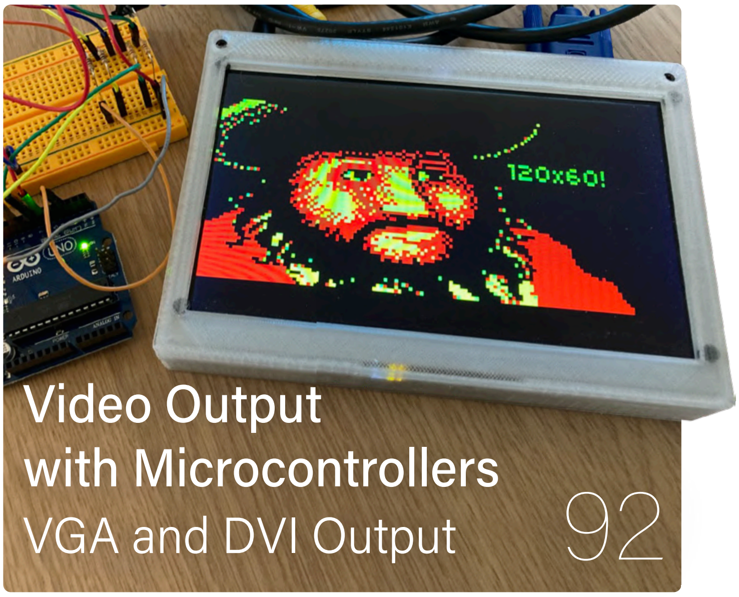
Features

Industry

My First Software-Defined Radio

Built in Less than 15 Minutes

56



Video Output with Microcontrollers

VGA and DVI Output

92

Projects

Next Edition

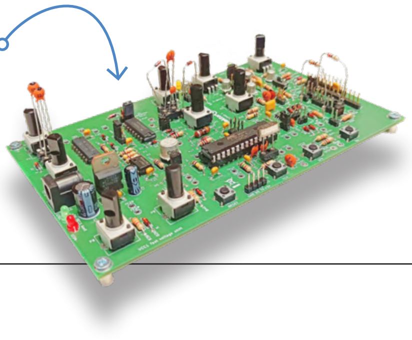
Elektor Magazine Edition 5-6/2023 (May & June 2023)
As usual, we'll have an exciting mix of projects, circuits, fundamentals, and tips and tricks for electronics engineers and makers. This time, we'll focus on Test and Measurement.

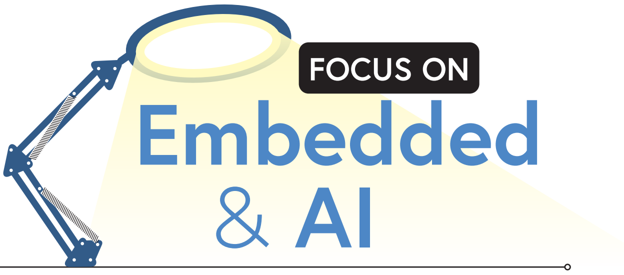
From the Contents:
- Energy Logger
- Oscilloscope Probe for RF
- Adaptive Universal Remote Control with ESP32
- The Android Wi-Fi API
- BL808: RISC-V for AI, Video, and Much More
- eCO$_2$ Telegram Bot
- Basics: Analog Signals and Microcontrollers
- Super Servo Tester
- Programming Voice-Controlled IoT Applications

And Much More!
Elektor Magazine edition 5-6/2023 (**May & June 2023**) will be published around May 11, 2023. The arrival of printed copies for Elektor Gold Members is subject to transport. Contents and article titles are subject to change.

Poor Man's ChipTweaker
DIY Hardware Attacks

22





FOCUS ON

Embedded & AI

Cloc 2.0: The Alarm Clock You've Always Wanted

By Yves Bourdon (France)

Clocks are probably one of the most popular topics in DIY electronics, together with radio receivers. I wouldn't be surprised if every reader of this article has built a clock at least once. I have built a lot of clocks in my life, and many had alarm functions. Because of this, I know what I want from an alarm clock. As my ideal alarm clock is not available commercially, I built it myself. Cloc 2.0, presented here, is the result.

I built my first clock in 1971 using Nixie tubes from an old IBM computer display that I had been given. The board used an impressive number of components, and the precision was about ten seconds per day. In the years that followed, I designed and built many clocks, including the most famous ones from Elektor.

My Nixie clock was also used as an alarm clock, and it's a feature I developed a lot over the years by simplifying the human-machine interface as much as possible. My last alarm clock was based on a PIC32 processor with two displays (**Figure 1**). It had an ESP8266 module to set the time automatically from an online time server over NTP. The PIC32 pressed buttons on a real remote control to turn on my favorite radio station or play music. A rotary encoder and two push-buttons allowed setting the alarm clock. This alarm clock was the basis of the project described below: Cloc 2.0.

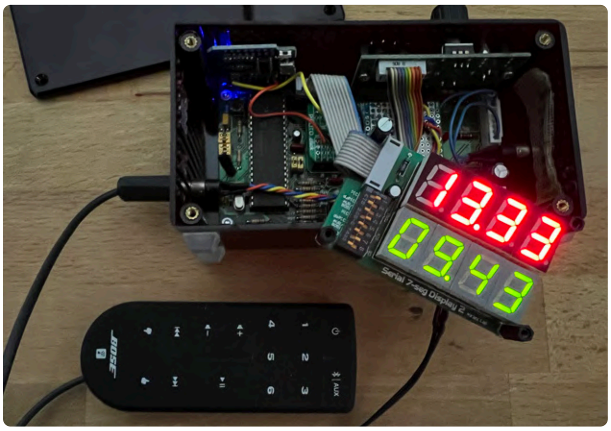
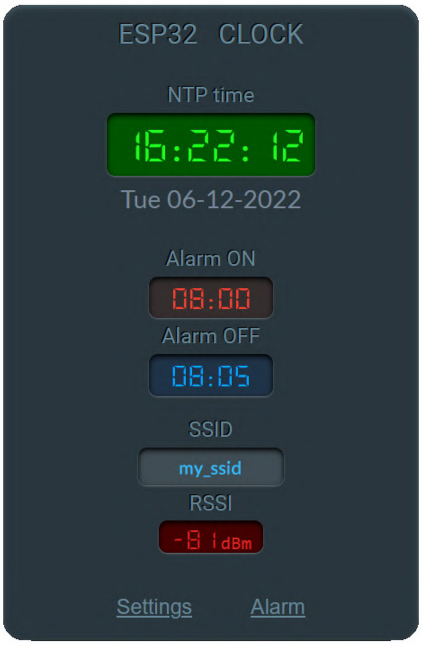


Figure 1: Cloc 1.0, the predecessor of Cloc 2.0. Based on a PIC32, it used a real remote control as IR output.



ESP32 CLOCK

NTP time

16:22:12

Tue 06-12-2022

Alarm ON

08:00

Alarm OFF

08:05

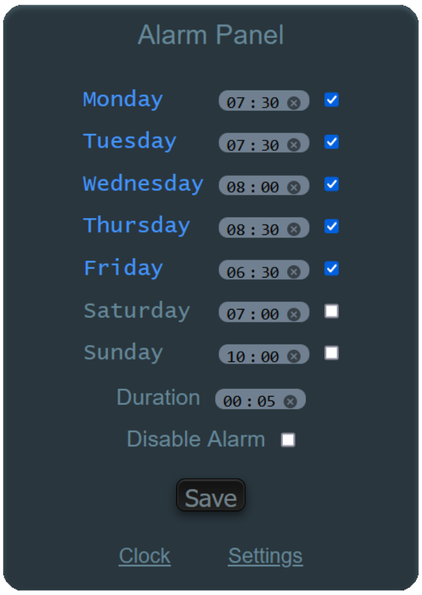
SSID

my_ssid

RSSI

-81 dBm

Figure 2: The Main panel shows besides the current time and date an overview of the most important settings.



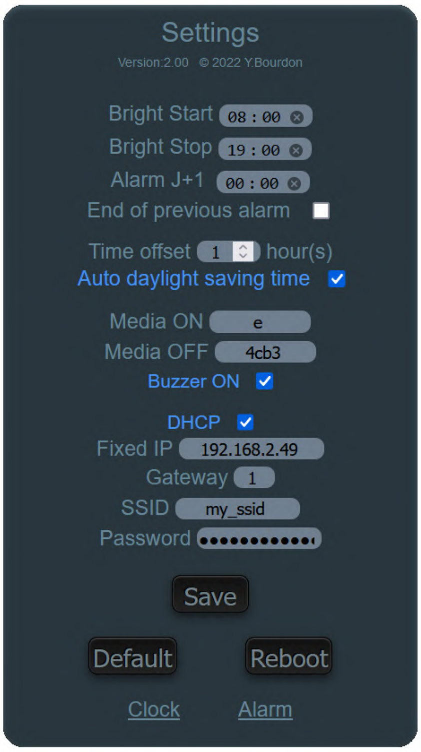
Alarm Panel

Monday	07:30	☑
Tuesday	07:30	☑
Wednesday	08:00	☑
Thursday	08:30	☑
Friday	06:30	☑
Saturday	07:00	□
Sunday	10:00	□

Duration 00:05

Disable Alarm □

Figure 3: The Alarm panel lets you set the alarm time for every day of the week.



Settings

Version:2.00 © 2022 Y.Bourdon

Bright Start 08:00

Bright Stop 19:00

Alarm J+1 00:00

End of previous alarm □

Time offset 1 hour(s)

Auto daylight saving time ☑

Media ON e

Media OFF 4cb3

Buzzer ON ☑

DHCP ☑

Fixed IP 192.168.2.49

Gateway 1

SSID my_ssid

Password ●●●●●●●●●●●●

Figure 4: All the user-configurable parameters are found in the Settings panel.

Beautiful and Easy to Reproduce

I wanted to make an alarm clock that was easy to build, did not use SMD parts, and fit as easily as possible into a standard commercially available enclosure. I also wanted to keep the retro touch of the 7-segment displays, whose variable brightness makes the alarm clock very readable at night without being too disturbing when you like to sleep in the dark.

The Web Interface

Cloc connects to a Wi-Fi network, so it can access a time-server somewhere on the planet. It can connect using DHCP or with a fixed IP address. Once it is connected, you can access its web server to know the time, set the alarms and adjust the clock's parameters. The optimized graphical HTML interface works on most standard browsers and handheld devices. The interface consists of three panels: Main, Alarm and Settings.

Main Panel

The main panel (**Figure 2**) displays the current time and date, and the start and end time of the upcoming alarm. It also shows the SSID of the Wi-Fi network it is connected to with the signal strength RSSI. The color of the latter (green, orange, or red) depends on its value. The main panel is refreshed approximately every second.

Alarm Panel

The alarm panel (**Figure 3**) is used to set the alarm times for each day of the week. Unchecking a weekday disables the alarm for that day. It is also possible to disable all alarms with a single click.

The duration of the alarm is configurable too. At the start of an alarm, the *Media ON* command is sent using the infrared LED; at the end of the alarm, the IR *Media OFF* command is sent. Additionally, the alarm can sound the buzzer (to be enabled in the Settings panel).

Each time an alarm parameter is changed, a short beep is emitted. If the value changed with respect to its previous value, it is saved in the EEPROM.

Settings Panel

The following parameters can be adjusted. See **Figure 4.**

- High-brightness display period (Bright Start and Bright Stop).
- The moment when the upcoming alarm time will be displayed (*Alarm J+1*), e.g., in the evening when you go to bed or immediately after the last alarm (by ticking the *End of previous alarm* box).
- Daylight-saving time, automatic or manual.
- Infrared (IR) *Media ON* and *Media OFF* code to control another device (i.e., a radio).
- Enable/disable the buzzer.
- Wi-Fi network credentials using either DHCP or a fixed IP address.

Push Buttons

Cloc has two push-buttons for interacting with the user: S1 (Disable) and S2 (Stop). In normal operation mode, pressing S1 enables or disables the alarm (the lower display switches on or off). You can also (de)activate the alarm through the web interface. If S1 is pressed down when switching on Cloc's power, the default values as defined in the program (and that you must adapt to your own needs before programming the clock!), are restored. This is the thing to do when you switch on the clock for the very first time.

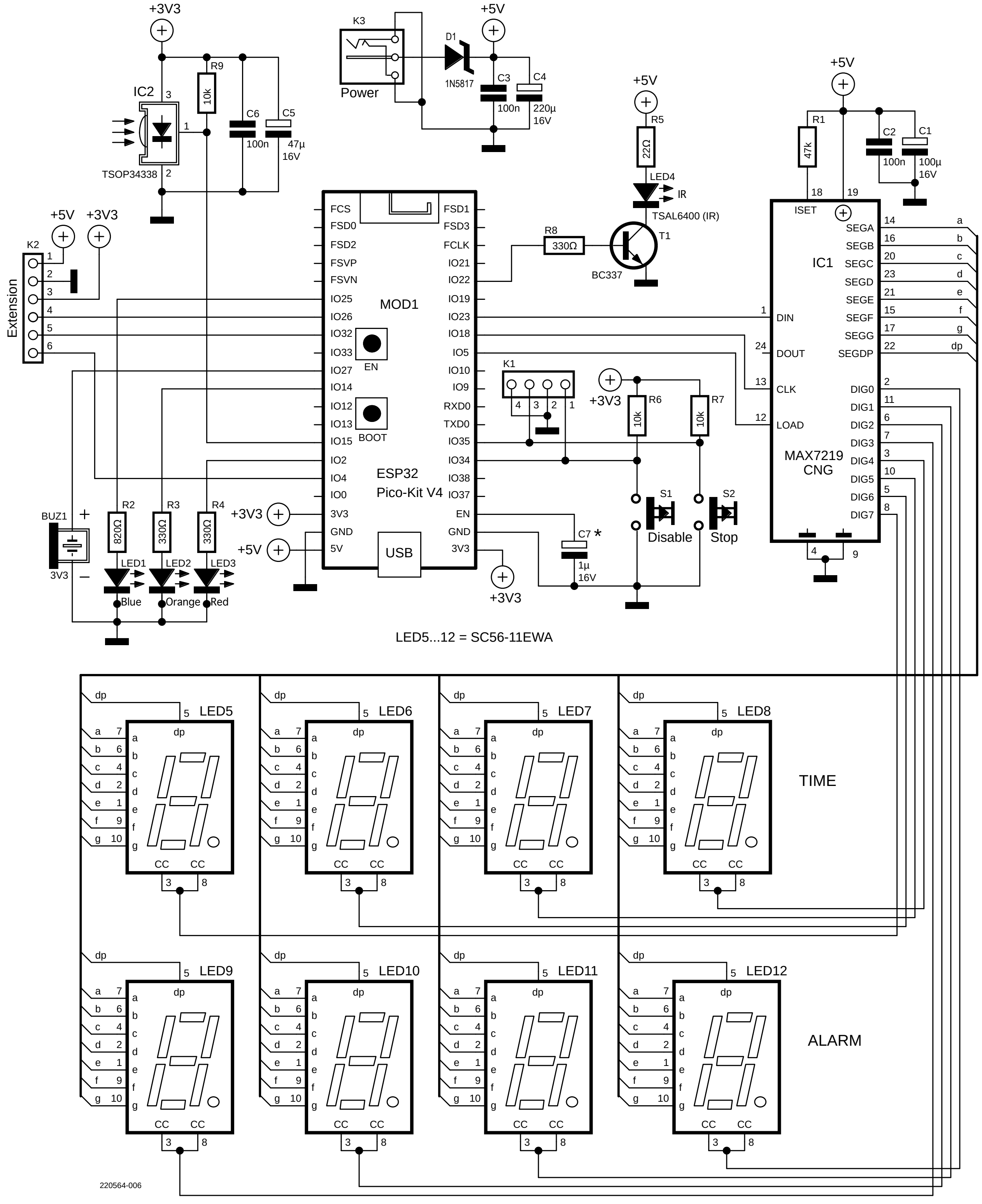


Figure 5: The ESP32-PICO-KIT provides Internet connectivity and the time, while the MAX7219 controls eight 7-segment digits organized in two rows. Note the IR interface around IC2 (input) and LED4 (output).

When an alarm was triggered in normal mode, pressing S2 will immediately stop it and the bottom display stops flashing. Pressing S2 without an alarm sounding sends the IR media ON/OFF code to the device of your choice. This lets you use the clock as a simple remote control.

If S2 is pressed during power-on, the clock will enter Access Point (AP) mode at address 192.168.4.1. You can connect to it using a cell phone, a tablet, or a computer to access the web interface where you can configure every user-programmable parameter of the device.

Circuit Description

The schematic of Cloc 2.0 is shown in **Figure 5**. Almost two thirds of it are taken up by the two 7-segment displays and its driver, IC1. To drive the eight digits, four per display (time & alarm), I used the MAX7219. It has a 2-wire SPI bus (DIN is not used) and it is easy to program thanks to open-source libraries. Resistor R1 allows setting the maximum brightness of the display (the higher its value, the dimmer the displays). Note that high-brightness 7-segment digits are really bright, and maybe too bright for an alarm clock.

The ESP32 module chosen for Cloc is the ESP32-PICO-KIT, which offers many I/O ports, and which has the perfect dimensions for this project. Capacitor C7 is only needed when the ESP32 module stays in reset mode after power-up. Some (older?) ESP32 modules appear to suffer from this problem, but I have never noticed it. If you decide to mount it, mount it lying down and pointing away from R6 and R7.

The alarm buzzer BUZ1 (without integrated oscillator) is driven by a PWM signal on port IO27 of the ESP32.

The ESP32's ports IO34 and IO35 are wired to the push-buttons S1 and S2, and connector K1. In a typical build, S1 and S2 would be mounted on top of the clock instead of on the PCB, which is why K1 is there. Pull-up resistors of 10 kΩ are used (R6 & R7) even though the ESP32 has built-in pull-ups. But as they have values of several hundreds of kilo-ohms, relying only on them makes a button wired this way a bit more sensitive to noise.

Three (debug) LEDs (LED1, LED2 & LED3) indicate the Wi-Fi network connection status (blue), web server interface connection (orange) and IR in- and outbound traffic (red).

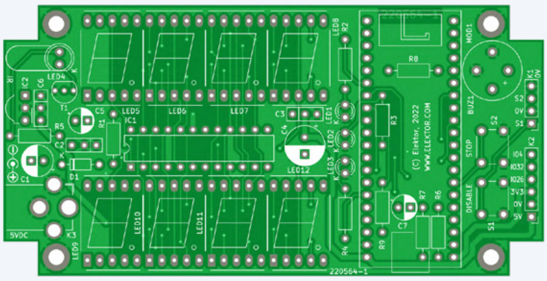
IR Interface

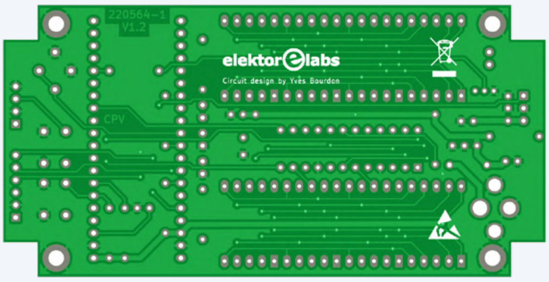
C5 and C6 decouple infrared receiver IC2 to recover an IR signal as clean as possible. The infrared TX LED (LED4) is controlled by transistor T1. The current through it is limited by R5 to about 150 mA. The IR output is powerful enough to control any equipment in its vicinity in the same way as the equipment's original remote control would do.

Power Supply

The external 5 V_{DC} power supply must be able to deliver at least 500 mA. Diode D1 protects the circuit against a polarity inversion. Two electrolytic capacitors (C1 & C4) provide buffering for current peaks. Small 100 nF capacitors in parallel (C2 & C3) are intended for attenuating higher frequency noise.

COMPONENT LIST





Resistors

R1 = 47 kΩ
R2 = 820 Ω
R3, R4, R8 = 330 Ω
R5 = 22 Ω
R6, R7, R9 = 10 kΩ

Capacitors

C1 = 100 µF, 16 V
C2, C3, C6 = 100 nF
C4 = 220 µF, 16 V
C5 = 47 µF, 16 V
C7 = 1 µF, 16 V

Semiconductors

D1 = 1N5817
IC1 = MAX7219
IC2 = TSOP34338
LED1 = blue, 3 mm
LED2 = orange, 3 mm
LED3 = red, 3 mm
LED4 = TSAL6400, IR LED, 5 mm
LED5-12 = 7-segment digit 0.56", CC (e.g. SC56-11EWA)
T1 = BC337

Miscellaneous

BUZ1 = Piezo buzzer without oscillator, 12 mm, 3.3 V
K1 = Pinheader 1x4
K2 = Pinheader 1x6
K3 = Barrel jack
MOD1 = ESP32-PICO-KIT
S1, S2 = Tactile switch 6x6 mm
PCB 220465-1
Enclosure Hammond 1591CTDR (translucent red)
DS3231 real-time clock (RTC) module

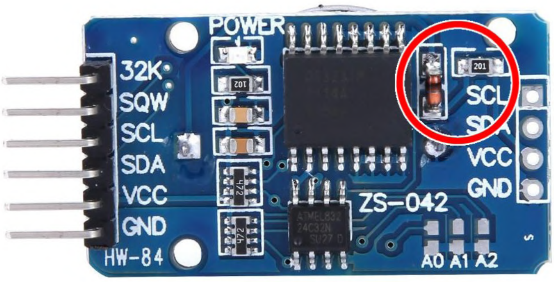


Figure 6: If the (optional) DS3231 real-time clock (RTC) module is used with a non-rechargeable CR2032 button cell, remove the resistor or the diode (or both) in the red circle to avoid damaging the battery.

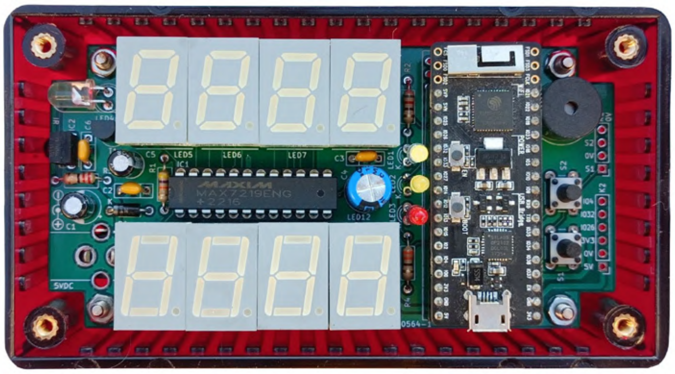


Figure 7: The assembled board fits nicely inside the enclosure. Note that you can also mount it on the cover instead to hide the enclosure's screws. The power supply connector can be mounted on either side of the board.

Extensions

Connector K2 allows extending the clock with, for instance, a real-time clock module (see below). Three I/O ports are available for this: IO4, IO26 and IO32.

Real-Time Clock (RTC) Option

A DS3231 RTC module can be connected to extension bus K2. The RTC takes over if the NTP server is inaccessible for some reason (e.g., Wi-Fi is down, or a problem at the internet provider's end). The RTC module is detected and configured automatically by the software and updated once a day (at 12:00). The DS3231 is a very capable chip featuring crystal drift compensation thanks to an integrated temperature sensor.

The DS3231 RTC module sports an I^{2}C interface and connecting it to K2 requires only four wires: GND, 3.3 V, SDA and SCL. If an RTC module is detected, port IO26 becomes the SDA signal while port IO32 takes care of the SCL signal of the I^{2}C bus.

Danger!

Be careful, these RTC modules can be dangerous! Indeed, the module features a battery charging circuit, but it often comes with a non-rechargeable CR2032 battery installed that might eventually explode. There are two solutions to this problem. The easy but more expensive one is to replace the battery by a rechargeable LIR2032 battery. The second solution is to remove the diode or the 200 Ω resistor (or both) near the SCL pin of the 4-pin connector, see **Figure 6**.

Mechanical Details

A good enclosure for Cloc is the 1591CTDR from Hammond, which measures 120 mm by 63 mm by 38 mm. It is translucent (IR) red and perfect for the time-of-day and alarm time 7-segment displays (red and/or orange). The transmission and reception of IR signals also benefit from such an optical filter. Note that light with little or no red content (like the blue LED) is hardly visible due to this filtering.

And while we're on the subject of LEDs, the ESP32-PICO-KIT has a (way too) bright red power LED that you probably want to hide or remove.

The printed circuit board (PCB) was designed with this enclosure in mind. The KiCad6 project can be downloaded from [1]. The board, mounted on four 16 mm studs, fits perfectly inside the enclosure, as shown in **Figure 7**. Only four 3.2 mm holes need to be drilled in the bottom to fix the board, one 8 mm hole is needed on the side for the power connector, and two 12 mm holes on top allow inserting two nice push buttons.

The optional RTC module can be connected under the main PCB to K2 with its component side up using the 4-pin non-populated header (you may want to remove the 6-pin angled header to prevent short circuits). You can also stick it to the main PCB with double-sided adhesive tape (as I did).

The Software Development Environment

I used the Arduino IDE to develop the software for Cloc 2.0 with the ESP32 boards package added to it. To avoid compilation or memory management errors, you must select the board *ESP32 PICO-D4* with a 240 MHz CPU frequency. For the *Partition Scheme* choose *Default* (i.e., 4 MB with SPIFFS). If you select the right serial port, then compiling and uploading the program should go smoothly!

The following external libraries must be installed in the IDE before compiling the program. I have indicated in the source code where to get them. Most can also be obtained using the IDE's *Library Manager*.

- AsyncElegantOTA
- AsyncTCP
- ESPAsyncWebServer
- IRremote (see line 126 of `Alarm_Clock.ino`).
- LedControl
- RTClib

Important: `LedControl.h` must be modified. Line 30 (`#include <avr/pgmspace.h>`) must be commented out. Also see the comment in the source code.

The download [1] contains a folder named *libraries* which includes all the libraries used, including the pre-modified `LedControl.h`. Copy these libraries into *libraries* subfolder of the Arduino sketchbook folder (you can see where it is by going to the IDE's *Preferences* window).

Refer to [2] to learn how to add the *ESP32 Sketch Data Upload* option to the IDE. You'll need it to load the clock's web interface into the ESP32 module's data memory. Don't forget to upload this data; otherwise the web server will not work.

Over-the-Air (OTA)

To remotely update the firmware of the alarm clock, point a browser to `IP_of_cloc/update` where `IP_of_cloc` represents the IP address of the clock. You can now pick an executable on your computer and upload it to the clock. A detailed explanation of the OTA interface is available at [3].

Program Internals

The program is quite well documented (unfortunately often in French), and after reading the following, you should find your way around easily. You might want to monitor the serial port of the ESP32 because a lot of debug information is sent on it.

After including the libraries, the default values are defined that are used either on the first boot, by holding down the alarm disable key (S1), or from the Settings page after clicking the Defaults button. The order of the constants follows the order of the parameters in the Settings panel (lines 40 to 65 of *Alarm_Clock.ino*). Check in particular the fixed IP address and gateway, as well as Wi-Fi credentials. For the Alarm panel (lines 67 to 84) things are organized the same way.

At power up, the function `setup` is executed first. It is easy to follow the initialization order of the Cloc hardware. The function `initRTC` checks if a DS3231 RTC module is connected. If button S1 is being held down at this time, the default values will be loaded.

If this is the clock's first boot or if the button Stop (S2) is pressed during power up, the clock will enter Access Point (AP) mode. In this case, you can connect with a browser to its web interface at 192.168.4.1, as indicated by the clock.

After retrieving the IP address (DHCP or fixed) and connecting to the Wi-Fi network, we look for the NTP time. If it is found and an RTC is present, they will be synchronized. The clock's IP address is shown briefly before switching to time display. If the display starts flashing 'HH:HH' on the time display and 'AA:AA' on the alarm display, the NTP server could not be reached, and you must reboot the device (if no RTC is present; otherwise it displays the RTC Time).

Initialization ends by starting the web server and the `AsyncOTA` service, which allows doing a remote update of the firmware and SPIFFS.

The function `loop` executes the rest of the program in an endless loop. If the RTC is present (`flagRTC=1`), it reads the RTC time. Without RTC, it tries the NTP time (`NTPflag=1`). Once the clock is connected, the program reads the RTC time, while `strTime` contains the NTP time as "hh:mm:ss" if it is valid or as '0' otherwise.

Then the state of the buttons is checked (`checkButtons`) and it checks if the clock must beep and if an IR code was received. Since the server is asynchronous, we must use flags to signal parameter changes and alarm triggers to the main program.

The screen is refreshed every second (`dispTime`) and the alarms are checked (`checkAlarm`). Also, the program checks if it is in the high-brightness period or not (`checkBrightness`) and if it must synchronize the RTC with the NTP time (`checkSyncRTC`). This happens once a day at 12:00 (`syncRTC=12*60`). The function `checkTimer` checks if an alarm must be activated. For this, the time-of-day and the alarm time are converted to minutes (`computeMinutes`) before comparing them (`testRange`).

As mentioned before, the server is asynchronous. Each time the Settings or Alarm panel is refreshed, the data to be displayed (`ParametresProcessor` and `TimerProcessor`) is returned. For this, the `%value%` tags contained in the HTML page must be replaced by the real values.

For the main panel, it's a bit different because we must display almost in real time the time of day, the date, and the alarm start and stop times. A function written in JavaScript queries approx. every second the main program to retrieve the values to display (`onDataUpdate`). At each query, LED2 flashes briefly to indicate that the server is being accessed (`flashRqstBeacon`). To check that the page is indeed displaying the values in real time, press S1 (Alarm Disable). The alarm time is displayed or turned off simultaneously on both the 7-segment display and on the HTML page.

Regarding the IRremote Library

The `IRremote` library works perfectly fine, but requires some integration rules. Do not move the initialization sequences unless you know what you are doing.

To wake up to music instead of a beep, I use a Bose radio. Therefore, I limited the reception and decoding to Bose remote controls (see line 125 of `Alarm_Clock.ino`). You can remove this line to decode other remotes. However, I recommend limiting infrared reception to the brand of your radio or hi-fi system. Refer to [4] for possible options.

Specifications

- Based on an ESP32 processor.
- Two independent 7-segment displays for the time of day and the alarm time.
- The time is set automatically by connecting to an online time-server.
- Alarm time for each day of the week.
- The alarm output: buzzer and infrared code for e.g. radio, hi-fi system or TV.
- Two push-buttons for interacting with the device.
- Integrated web server for remote configuration via Wi-Fi.
- All settings are saved in EEPROM.
- Over-the-Air (OTA) mode allows updating the firmware remotely.
- Optional DS3231-based real-time clock (RTC) module with back-up battery.

Similarly, IR transmission is limited to the Bose format. Therefore, it is quite likely that you must adapt the code to the brand of the Hi-Fi equipment you use. Replace the line `IrSender.sendBoseWave` in the functions `startMedia` and `stopMedia` in `Alarm_Clock.ino` by a function that corresponds to your brand (again, refer to [4]).

To find out which IR codes to use for your (audio) system, connect the USB port of the ESP32 module to a computer running a serial monitor, for instance the *Serial Monitor* built in the Arduino IDE. Point your remote control to the clock's IR receiver and press a key to display the corresponding code in hexadecimal on the computer. Enter the codes in hexadecimal in the *Media ON* and *Media OFF* fields of the Settings panel.

IR activity is indicated by LED3 flashing briefly. As a result, this LED also flashes when an alarm is active.

A DIY Solution

I've been using this new version of Cloc 2.0 for several months now, and I've never overslept due to it malfunctioning. The ability to set alarm times for each day of the week is very convenient.

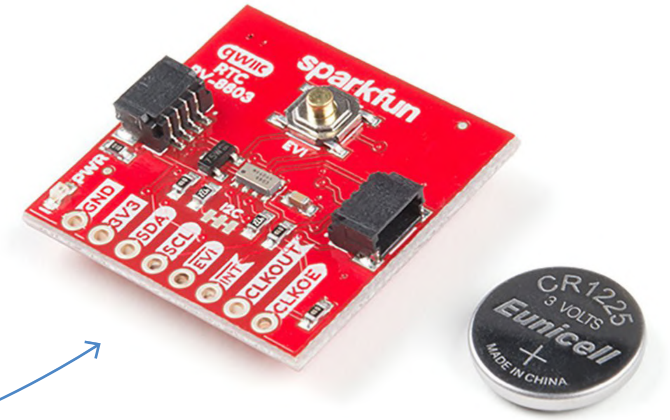
I highly recommend adding the RTC option because if the NTP server does not respond (for instance because your modem got offline for some reason), the alarm clock will lose time at the next NTP time resynchronization moment. You will then have to stop and restart the alarm clock when the NTP time is available again. For even more reliability, I built a small NTP server which receives the time from a GPS [5]. So, even if the Internet connection is cut, my alarm clock can set itself automatically.

I hope that many of you will make this clock. Please don't hesitate to contact me on my private email (yb.electronique@orange.fr) if you encounter any difficulties or if you make any improvements to my favorite alarm clock!

220564-01

Questions or Comments?

Do you have technical questions or comments about this article? Email the author at yb.electronique@orange.fr or contact Elektor at editor@elektor.com.



WEB LINKS

[1] Downloads for Cloc 2.0 at Elektor Labs: https://www.elektormagazine.com/labs/cloc-le-reveil-20
[2] ESP32 SPIFFS data upload: https://randomnerdtutorials.com/install-esp32-filesystem-uploader-arduino-ide/
[3] ESP32 OTA (Over-the-Air) Updates: https://randomnerdtutorials.com/esp32-ota-over-the-air-arduino/
[4] IRremote library: https://github.com/Arduino-IRremote/Arduino-IRremote
[5] NTP Server at Elektor Labs: https://www.elektormagazine.com/labs/ntp-server

PIO in Practice

Experiments Using the RP2040's Programmable I/O

By Prof. Dr. Martin Ossmann (Germany)

The Raspberry Pi Pico is a popular development board that offers high processing power with low power consumption. And it represents excellent value at just €4 a pop. The RP2040 microcontroller used on the board has a (so far) unique feature: two PIO-I/O Blocks with eight state machines. Using working examples, we take a closer look at the benefits these can bring to an application.

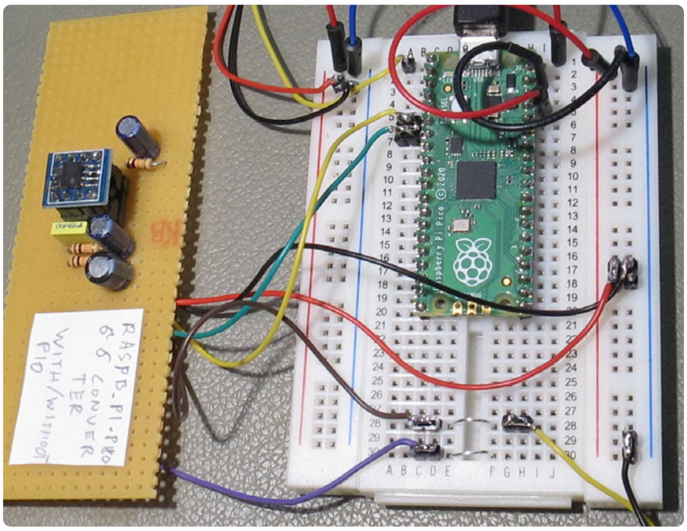


Figure 1: The Raspberry Pi Pico board mounted on a plugboard and the Delta_Sigma PCB.

If you already have some experience programming microcontrollers, you will no doubt appreciate just how useful a built-in programmable I/O unit can be to reduce load on the main processing cores. The big question is whether this unit is unnecessarily complicated to use and just not worth the effort. You will certainly get a better appreciation of what it can do once you've worked through the practical examples described here running on the PIOs built into the RP2040 processor. **Figure 1** shows my experimental hardware setup on which I developed the examples discussed.

The PIO Unit

The flexibility of the programmable input/output unit allows you to use the I/O pins to implement other types of communication protocols in addition to the existing standards such as SPI and I2C. It consists of two PIO blocks, each with four I/O processors. Each single I/O processor integrates with the chip architecture as shown in **Figure 2**.

Altogether there are two PIOs in the RP2040 and each contains four state machines (SMs) and one 32-word instruction memory to control the state machines. Each SM has two 32-bit registers X and Y and a 32-bit input shift register (ISR) and a 32-bit output shift register (OSR). The PIO cores are each connected to the RP2040 CPU using two 32-bit wide by four-word deep FIFOs. The TX FIFO unit is used for sending and the RX FIFO unit for receiving data. If data only flows in one direction the two FIFO can be configured to operate as a single eight-word deep FIFO. Each PIO processor has a 16-integer, eight

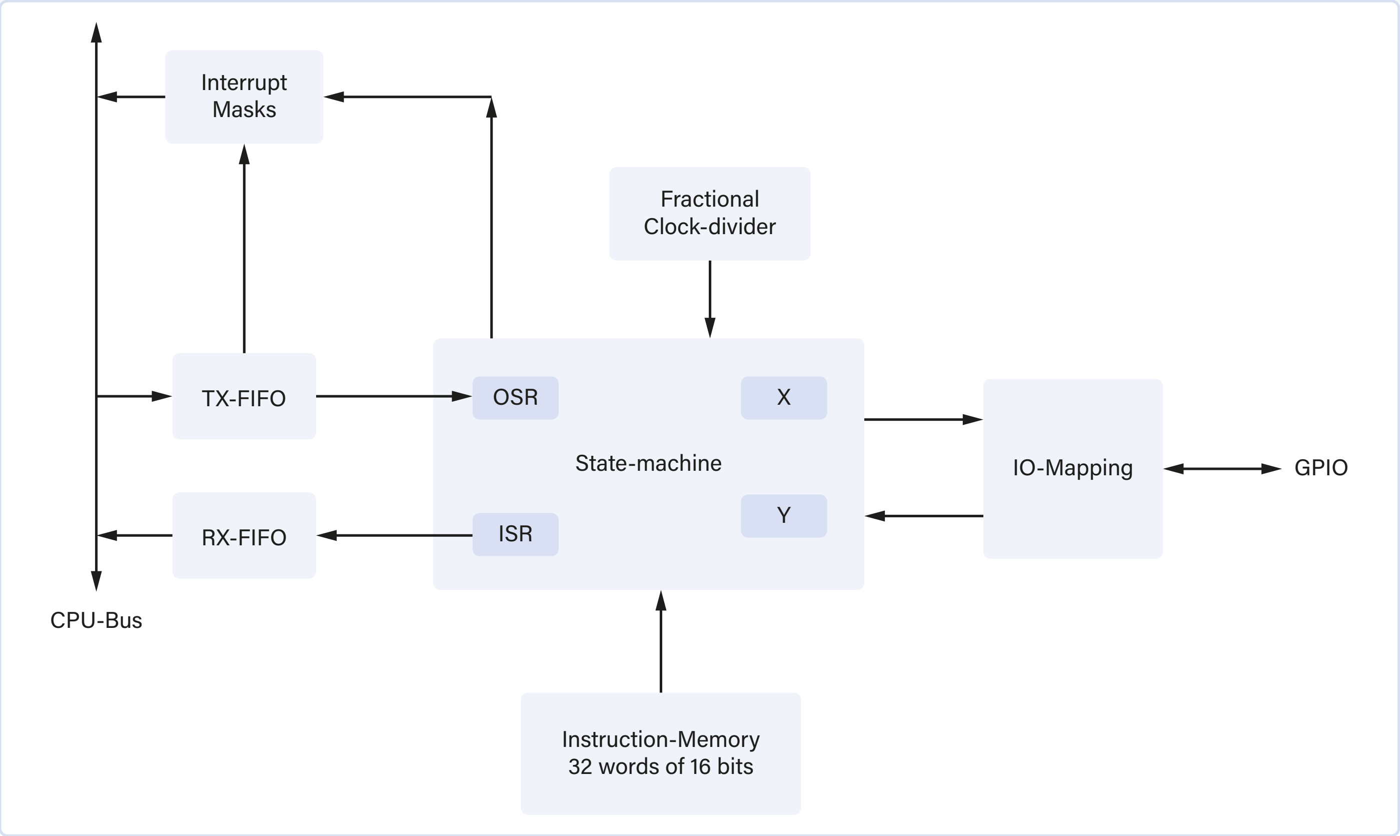


Figure 2: Block diagram of one of the two I/O processors (Source [4]).

fractional bits divider for clock generation (see below). This gives you the flexibility to set the clock rate of each processor at a rate between 2 kHz and 125 MHz.

The PIO processors are only able to understand nine instructions. These instructions are 16-bit wide stored in a 32 × 16-bit instruction memory which control the four SMs in each PIO block. The individual PIO programs are short enough to fit in this small memory. The instructions are stored here by the RP2040 CPU. These small programs are written using a special but fairly basic PIO assembler program. Each instruction takes one clock cycle to execute.

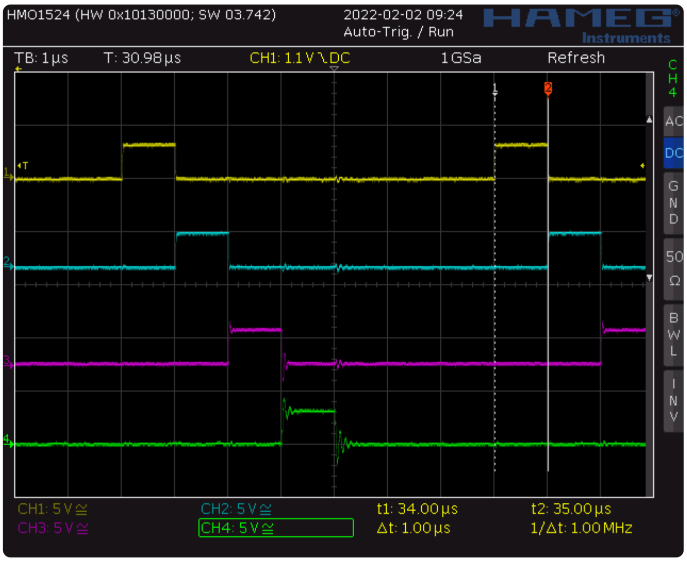


Figure 3: Waveforms from the first example.

A single PIO processor can only address a few GPIO pins, and it uses short addresses to do so. The IO mapping determines which pins are then specifically addressed. There are several programmable constants that determine for the respective area (input, output, side set and set) which is the first addressed pin. This allows the IO pins of a PIO processor to be defined very flexibly. Multiple processors can also access the same pin.

Tools of the Trade

VS Code is used as the development environment here. The installation and use of this tool are well described in [1]. There you will also find information on how to use the project generator. This is used to create a Raspberry Pi project with all the necessary files (source code, Configuration files, Make-Setup, etc.).

You also need to specify whether `printf()` function will output via the USB or UART. Here you also configure which libraries you need to use. In this article, for example, we use the PIO tool. You can also find help for VS Code installation in [2]. There is also a detailed description of which additional tools (Make, Git, etc.) need to be installed.

All software examples can be downloaded from [3].

Let's Get Started

We want this first program to output a 1 µs pulse from four output pins one after another as shown in **Figure 3**. A four-channel oscilloscope is connected to the four output pins to view the resulting output signals.

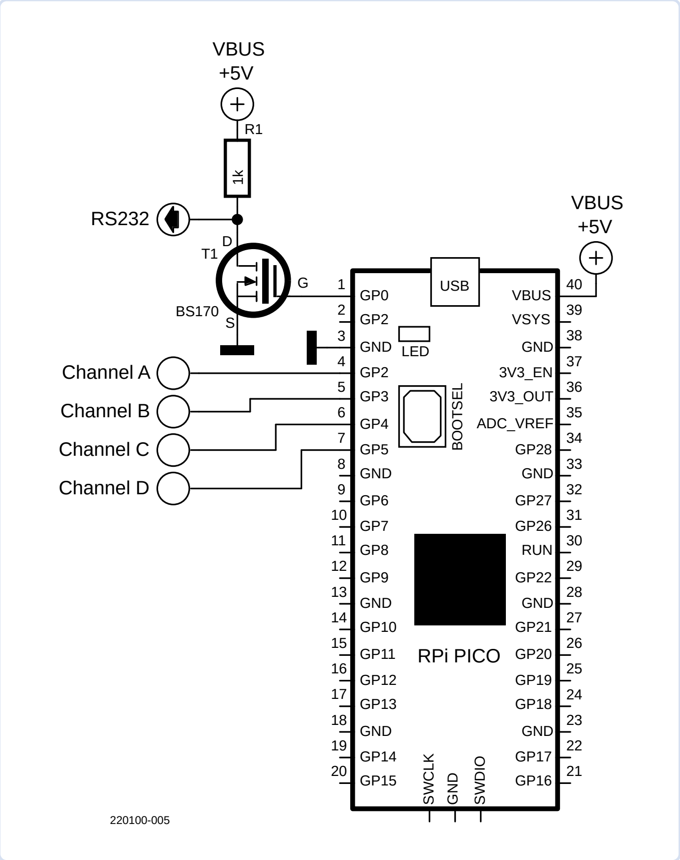


Figure 4: Four-channel scope hookup to the Raspberry Pi Pico to view output waveforms from the first example.

Figure 4 shows the connection of the oscilloscope to the I/O pins of the Raspberry Pi Pico.

The following listing contains the first sample PIO program. After three lines of assembler directives, the first executable instruction is on line 4. This and subsequent lines then generate the four pulses.

Listing 1

```
001 .program pioTest
002 .side_set 2 opt
003 .wrap_target
004   nop           side 0b01 // GP2=1, GP3=0
005   nop           side 0b10 // GP2=0, GP3=1
006   set pins,0b01 side 0b00 // GP2=0, GP3=0,
         GP4=1, GP5=0
007   set pins,0b10       //    GP4=0,      GP5=1
008   set pins,0b00   [2]  //    GP4=0,      GP5=0
009 .wrap
```

Lines 4 and 5 contain the nop instructions, which, as you would suspect, do nothing, but line 4 is extended by the text side 0b01. This indicates that on this line something else is "done on the side". In this case Side Pin 0 is set to a "1" and Side Pin 1 is reset to a "0". Every PIO instruction can be extended in this way. This allows you to easily influence many more pins at the same time. It is important to identify which are the side-set pins by using configuration commands written in C. The definition is done in C in Listing 2 using the sm_config_set_sideset_pins() function in line 5. This indicates that the side set pins start at GP2.

Listing 2

```
001 #define set_base 4   // pins start at GP4
002 #define set_count 2  // number of pins is 2
003 sm_config_set_set_pins (&c,set_base,set_count);
004
005 #define side_set_base 2 // side-pins start at GP2
006 sm_config_set_sideset_pins(&c, side_set_base);
007
008 float div = 125.0;
009 sm_config_set_clkdiv(&c, div);
```

The nop instruction in line 4 of Listing 1 generates a rising edge via side-set at side pin 1 (= GP3). Line 6 contains a set instruction that sets pins 0 and 1 of the set pins. The pulse then reaches pin GP4. Side pins 0 (GP2) and 1 (GP3) are reset via side setting. In the configuration routine in the second listing in line 3 it is determined that the set pins start from pin number GP4. The set instruction can be used to load registers or IO pins with constant values.

The modifier, [2], after the set instruction in line 8 of the first listing causes the instruction to be followed by a two-clock delay. With this specification using square brackets, each instruction can be delayed by 1 to 31 clock periods.

Now you only have to use a suitable clock to ensure that each pulse is exactly 1 µs. To do this, you have to divide the 125 MHz system clock by 125. With the resulting clock rate of 1 MHz, each cycle lasts the desired 1 µs. This is defined in the configuration routine of the second listing above in lines 8 and 9.

The assembler directives .wrap_target and wrap in Listing 1 cause the instructions between them to be repeated endlessly in a loop at runtime. The jump back to the start of the loop (.wrap_target) does not take any time. You can program "zero-cycle-overhead loops." In our program, it takes seven clock cycles (five instructions + two cycles of delay) to go through the loop once.

Clock Generation: A Fractional Divider

As already mentioned, the PIO processor clock is derived from the 125 MHz system clock with the help of a "fractional divider". The 16-bit divider can count up to 2^{16} = 65,536 and has eight binary digits after the zero point. The accuracy is thus 1/256. The "fractional divider" is

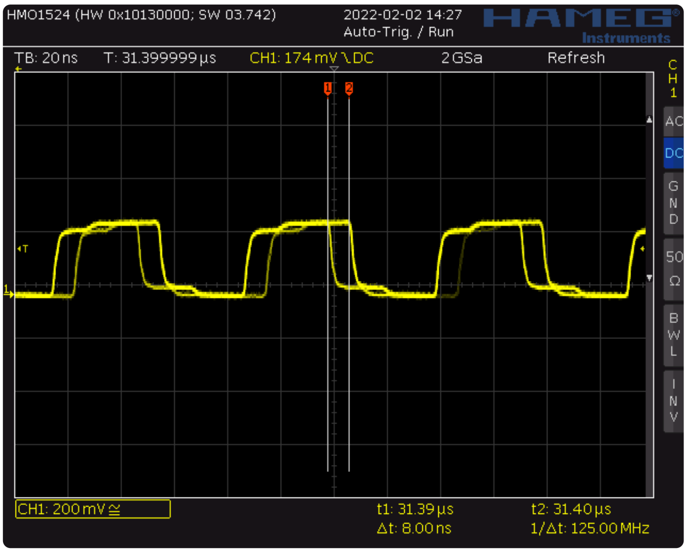


Figure 5: The generated clock signal exhibits jitter.

implemented in the same way as in DDS signal generators. The fraction behind the decimal point is added continuously and a clock edge is generated with each carry event. The following listing demonstrates setting a fractional clock rate.

Listing 3

```
float clockFrequency=28e6;
float div=125e6/clockFrequency;
sm_config_set_clkdiv(&c,div);
```

Due to the way the clock is generated, the clock period fluctuates around one system clock period and produces jitter in the clock waveform. In the example, we want a clock rate of 28 MHz but since 28 does not evenly divide into 125, the clock divider factor is not a whole number but has some remainder value. The clock division factor is 125/28 = 4.4642857...

The third listing shows how this divider is used for configuration. As a demonstration, a simple square-wave signal is generated with the PIO program in the fourth listing below.

Listing 4

```
.program theProgram
.side_set 1 opt
   nop   side 0
   nop   side 1
```

The oscilloscope shows clock jitter very clearly in the waveform in **Figure 5**. With higher clock divider values the jitter becomes proportionally smaller and much less important, so that it can often be neglected. With the "Fractional Divider" you can then generate e.g. baud rates such as the typical 115,200 bit/s with sufficient accuracy.

Serial Data Transmission: A TX-UART

The next example is a bit more practical: It is about sending data serially using an RS232 interface. The interface is connected to pin GP4 so that GP0 on the built-in UART remains free for debugging purposes. Since the interface is inverting, you have to generate an inverted signal (idle state = high). When the ASCII character is sent, the pulse shape at pin GP4 then looks like that in **Figure 6**.

In addition to the TXD signal, the UART should output a 1 on a busy line while sending the eight data bits. GP5 (side pin 0) should be used as the busy line. The associated PIO program can be seen in Listing 5 below.

Listing 5

```
001 .program theProgram
002 .side_set 1 opt
003     pull              [1]
004     set pins,0 side 1 [2] // startBit=0
005     set x,7               // loop for 7+1 times
006 bit:
007     out pins,1        [2] // LSB first
008     jmp x-- bit
009     set pins,1 side 0 [2] // stopBit=1
```

The `pull` instruction in line 3 takes a 32-bit word from the TX FIFO unit and stores it in the OSR. If no word is currently stored in the TX FIFO, the instruction just waits for a word and then inserts it directly into the OSR. The output pin is set to 0 to act as the start bit by the set instruction in line 4.

The eight least significant bits of the word in the OSR are now output in a loop. For this it's necessary to initialize the loop counter. The X register serves here as a loop counter. The set instruction on line 5 initializes it with the value 7.

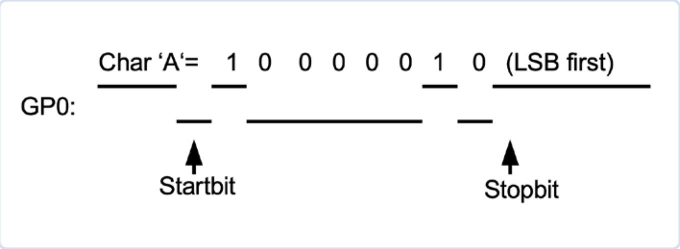


Figure 6: Pulse sequence to send the character "A".

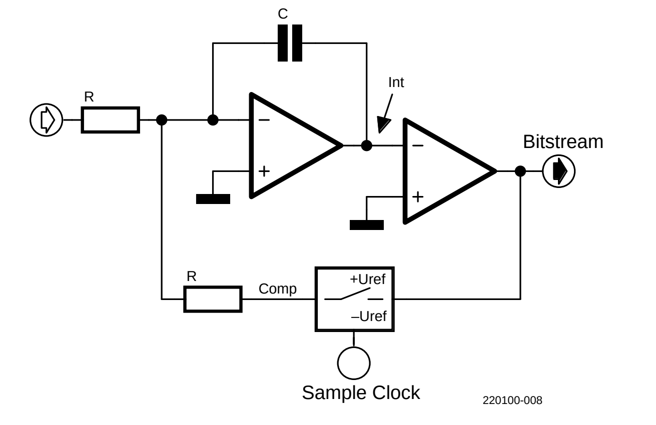


Figure 7: Delta-Sigma-ADC functional diagram.

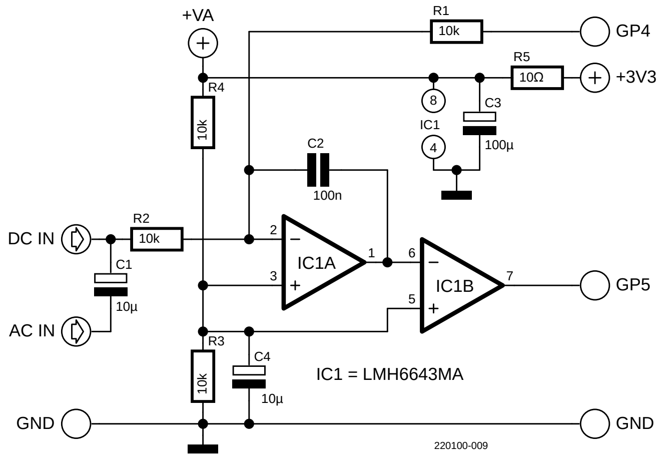


Figure 8: Delta-Sigma-ADC practical circuit according to Figure 7.

The label `bit` in line 6 marks the beginning of the loop. The first and only instruction in the loop is the `out` instruction on line 7. This outputs the LSB from the OSR at Out-Pin 0 and shifts the bits in the OSR one position to the right. In addition to the set pins and the side set pins, the out pins form the third method of using the GPIO pins. Pin GP4 is also set as an out pin.

The conditional jump in line 8 with the option `x--` decrements the X register by 1 and then jumps to the `bit` label(start of the loop) when `x` is greater than 0 . In this way, the loop runs through eight times. The last instruction of the TX transmit routine is the `set` instruction on line 9, which creates the stop bit by setting set pin 0 to a 1.

With the two side extensions the Busy signal is generated by side pin 0 (GP5).

The delay extensions in the square brackets cause each bit to be exactly four clock periods long. So the baud rate equals the clock rate divided by four. Setting `clkDiv` to 125,000,000 / (4 * 115,200) consequently results in the desired baud rate of 115,200 bit/s.

The `pio_sm_put_blocking(...)` function is used to transfer characters with a C program to the PIO-UART, as shown in the `PIOprint()` function in the next listing. Here it passes a character to the TX FIFO unit. If the unit is full, it blocks and waits for more space.

Listing 6

```
void PIOprint(const char *s) {
   while (*s)
      pio_sm_put_blocking(pio, sm, *s++);
   }
```

The TX-UART routine neatly shows that you can achieve meaningful functionality with just six instructions, with side-set and delay instruction additions playing an important role.

A Delta Sigma ADC

In the following section, a delta-sigma A/D converter is to be implemented using a PIO. The principle of such a converter can be seen in **Figure 7**. An attempt is made to compensate for the input voltage with a switchable voltage source. An integrator at the input integrates the error signal and a subsequent comparator determines the sign of the compensation voltage in the next time interval. A bit sequence is created at the output, which follows the input voltage average. **Figure 8** shows a practical circuit following this principle.

This circuit connects to GP4 and GP5 of the Raspberry Pi Pico. The bit sequence is determined via PIO and eight bits are stored together in one byte in the RX FIFO unit. The microcontroller then takes the bytes from this unit and processes them further. The associated PIO program can be seen in the following listing.

Listing 7

```
001 .program hello
002 Loop1:
003    set   x,7
004 inLoop:
005    in    pins,1
006    mov   osr,~isr
007    out   pins, 1
008    jmp   x-- go1    [20]
009    push  noblock
010    jmp   Loop1      [23]
011 go1:
012    jmp   inLoop     [26]
```

The outer loop with the label `Loop1` repeats endlessly. The inner loop always makes eight iterations. Each iteration begins with the instruction after the `inLoop` label. The `in` instruction on line 5 takes one bit from the Input-Pin and writes it to the ISR. In line 6 the ISR is copied inverted into the OSR. In line 7 the least significant bit is output at the output pin, which generates the next compensation step in the integrator.

The bit read is the next output bit of the delta-sigma converter. Eight bits are collected in the ISR. The `jmp` instruction on line 8 controls the inner loop. When it has completed 8 loops the `push` instruction on line 9 is executed. This instruction stores the ISR in the RX FIFO unit which the MCU can then unload. The delay instruction extensions in square brackets ensure that each delta-sigma step lasts 50 clock pulses. The clock divider was set to 25. The sampling rate of the delta-sigma converter is therefore 125 MHz / (25*50) = 100 kHz.

For the demonstration, a 50 Hz sine wave was converted by the delta-sigma converter. **Figure 9** shows the resulting spectrum. The distance between the useful signal and the noise is around 60 dB. That's not shabby at all for such a simple circuit!

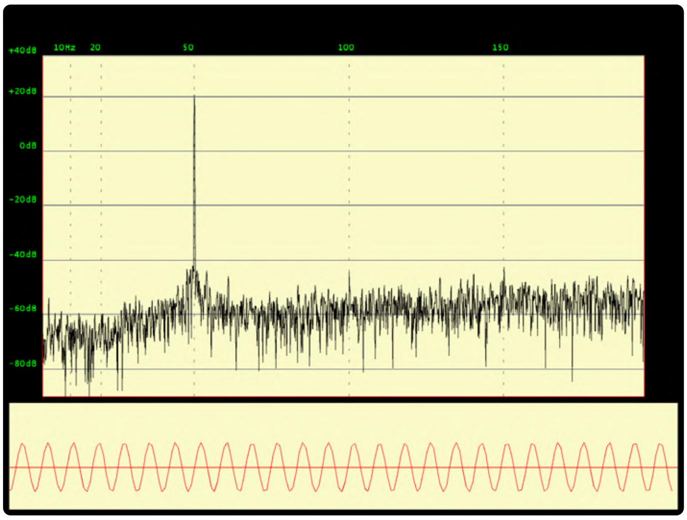


Figure 9: Spectral content of a 50 Hz sine wave following Delta-Sigma conversion.

Build a Signal Generator

In this example we build a simple R2R-DAC using a ladder arrangement of resistors according to **Figure 10**. This is connected it to the Raspberry Pi Pico I/O pins to produce analogue output signals. The generator will need to generate a continuous periodic output signal. To achieve this, data values need to be repetitively transported from a table in memory to the D/A converter. This works particularly well using DMA (**D**irect **M**emory **A**ccess). The DMA controller of the Raspberry Pi Pico transfers the data to the TX FIFO unit. The PIO program consists of a single executable instruction, as shown below in **Listing 8**.

Listing 8

```
.program pio_serialiser

.wrap_target
    out pins,8 // use autopull
.wrap
```

The out instruction brings the eight least significant bits from the OSR to the eight out pins GP0 to GP7. The OSR is then automatically reloaded from the TX FIFO unit because the *auto-pull* PIO feature was activated in the configuration routine. The next eight bits are then output with the next instruction. So the output loop is exactly one PIO clock pulse long. The data must therefore get from the main memory to the TX FIFO unit as quickly as possible. To do this, two DMA controllers of the RP2040 are connected in series using a data and a control DMA. The control DMA controller ensures that the data DMA controller is restarted quickly. With this technique, a new byte can be output every four system clocks (125 MHz / 4). However, the simple D/A converter shown in **Figure 10** is not suitable for such a high output speed.

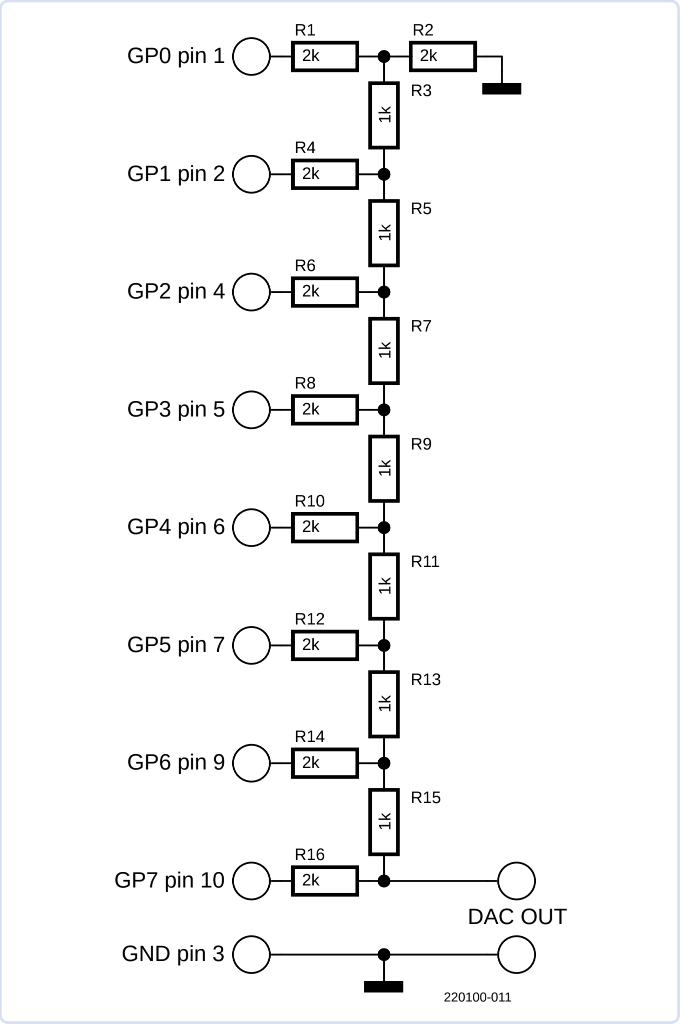


Figure 10: R2R-DAC circuit.

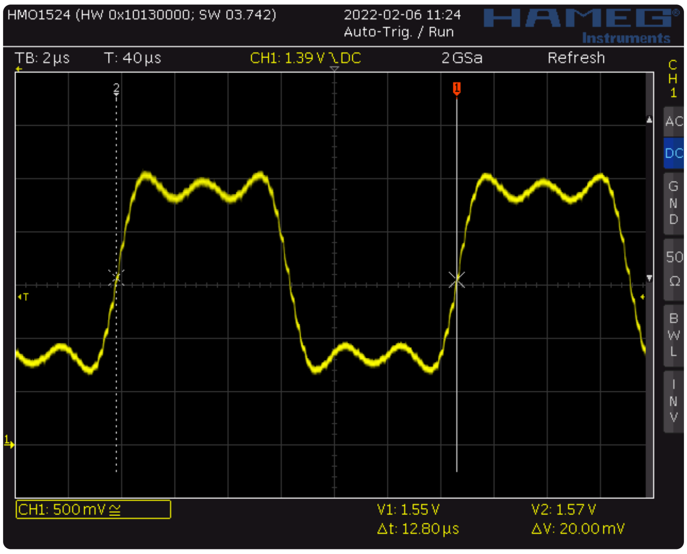


Figure 11: Base frequency plus two harmonics give a square wave.

In **Figure 11** *clkDiv* was set to 25. The resulting DAC clock is 125 MHz / 25 = 5 MHz. Since one period of the output signal is 64 samples long, the generated frequency is calculated as 5 MHz / 64 = 78.125 kHz. The Fourier approximation of a square wave made up of a fundamental wave plus two harmonics was used to produce the output waveform.

Register Initialization

The set instruction can be used to initialize registers X and Y with fixed values. Since instructions are encoded in 16 bits, the set command can only process constants in the range between 0 to 31. To initialize X or Y with larger constants, you can use a trick.

By calling the `pio_sm_exec(pio, sm, instruction)` function from a C program, you interrupt the instruction sequence of the PIO block `pio` and the state machine `sm` and insert the command `instruction`. The following function sequence can be used to initialize register X with the value 500,000.

Listing 9

```
pio_sm_put_blocking(pio, sm, 500000);
pio_sm_exec(pio, sm,
      pio_encode_pull(false, false));
pio_sm_exec(pio, sm,
      pio_encode_mov(pio_x, pio_osr));
```

First you put the 500,000 into the TX FIFO unit, and then you execute the `pull` instruction on the PIO controller, which brings the 500,000 from the FIFO unit into the OSR. Finally, the instruction `mov X,OSR` puts the value into the X-register as desired. Now you can start the actual PIO. The program in **Listing 10**, which flashes an LED, serves as an example.

Listing 10

```
.program blink
.side_set 1 opt
    mov y,x    side 1
yLoop1:
    jmp y--    yLoop1
    mov y,x    side 0
yLoop2:
    jmp y--    yLoop2
```

MicroPython

The Raspberry Pi Pico is a platform well suited to the development of programs written in MicroPython. There is a corresponding Python interface for PIO programming. This is not programmed using the PIO assembler, but by using appropriate Python syntax, which is strongly reminiscent of an assembler programming language. One advantage of using MicroPython is that you can save complete projects in one file. The associated sample program can be found in the **Listing MicroPython**.

The actual PIO program is coded as Python function `pio_prog()`. It consists of the code contained in the loop between `wrap_target()` and `wrap()`. The program toggles the Side Set pin 0 – which is the LED pin GP25 here. The on/off time is governed by a Y wait loop. The initial value of the Y register is in the X register. The value in the X register is controlled externally via the TX FIFO unit. This is executed by the `pull(noblock)` function. If a word is in the TX FIFO unit it is pulled into the OSR register. The `noblock` parameter causes the contents of the X register to be placed in the OSR when the FIFO unit is empty and the command sequence continues.

Closing Remarks

At this point, the overview of the PIO programming of the RP2040 MCU ends. It was demonstrated that powerful interfaces can be programmed using just a few simple instructions. The PIO programming facility of these MCUs makes them more versatile so that you may not even need to consider the use of FPGAs to expand the processor's capabilities.

This article can only give a rough overview of PIO programming. There is so much more to explore. If you want a good practical guide take a look at the literature [1][2][4][5][6][7], which also includes numerous other examples.

220100-01

Translated to English by Martin Cooke.

Listing MicroPython

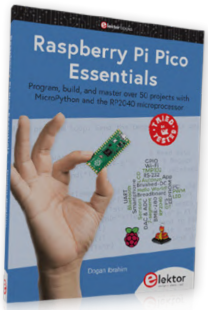
```
from machine import Pin
from rp2 import PIO, StateMachine, asm_pio
from time import sleep

@asm_pio( sideset_init=(PIO.OUT_LOW))
def pio_prog():
     wrap_target()
     pull(noblock)
     out(x, 32)

     mov(y, x) .side(1)
     label("Loop1")
     jmp(y_dec, "Loop1")

     mov(y, x) .side(0)
     label("Loop2")
     jmp(y_dec, "Loop2")
     wrap()

class PIOAPP:
     def __init__(self, sm_id, pinA, periodLength, count_freq):
        self._sm = StateMachine(sm_id, pio_prog, freq=count_freq, sideset_base=Pin(pinA)
        self._sm.active(1) # start PIO execution

     def set(self, value): self._sm.put(value) # put val into TX-FIFO

pio1 = PIOAPP(0, pinA=25, periodLength=1, count_freq=1_000_000)
print("ready1")
while True:
     pio1.set(200000) # 200 ms On/Off time
     sleep(1) # for one second
     pio1.set(100000) # 100 ms On/Off time
     sleep(1) # for one second
     print("idle")
     print("idle")
```

Questions or Comments?

To answer any technical questions relating to this article please contact the author at ossmann@fh-aachen.de or get in touch with the elektor team at editor@elektor.com.

Related Products

- Raspberry Pi Pico RP2040 (SKU 19562)
  https://elektor.com/19562
- Dogan Ibrahim, *Raspberry Pi Pico Essentials* (SKU 19673)
  https://elektor.com/19673



WEB LINKS

[1] Getting Started with Raspberry Pi Pico: https://datasheets.raspberrypi.com/pico/getting-started-with-pico.pdf
[2] Manual to set up the toolchain by Shawn Hymel: https://tinyurl.com/yde5dd8a
[3] Software download: https://elektormagazine.com/220100-01
[4] Data sheet for the RP2040: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf
[5] Raspberry Pi Pico SDK: https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf
[6] SDK Documentation: https://raspberrypi.github.io/pico-sdk-doxygen/index.html
[7] RP2040 MicroPython: https://docs.micropython.org/en/latest/library/rp2.html

Poor Man's ChipTweaker

We Have (Low-Budget) Ways of Making You Talk

By Luka Matic (Croatia)

Hardware Security Modules (HSMs) are electronic devices using various hardware protection schemes against unauthorized reading of the secret data they hold. They usually perform actions like encryption or digital signing but may perform other tasks as well. Bank cards or access cards are typical examples of HSMs. They must keep PINs and private keys secret as long as possible in case the card is stolen or lost. Furthermore, many general-purpose MCUs also sport some memory-protection features intended to protect against illegal copying of proprietary firmware. The Poor Man's ChipTweaker presented here is a tool to learn the basics of non-invasive attacks against HSMs. With it you can try to unlock a memory-protected microcontroller.

As explained in a previous article [1], our poor low-budget spies Alice and Bob can't rely on building their own HSMs as this requires expensive specialized equipment and knowledge they probably won't have. For their critical operations, they use their own hardware built with cheap general-purpose components they can trust. This works well for them, as long as proper Operations Security (OpSec) procedures are followed. On the other hand, they also can't avoid using HSMs like bank cards, and they may stumble upon an HSM whose secrets they may want to pry out. The Poor Man's ChipTweaker (PMCT) therefore will be quite useful for them as it allows them to learn the basics of non-invasive attacks against HSMs.

Hardware Attacks

Attacks against HSMs can be divided into three main types:

1. **Non-invasive:** The HSM is not opened or decapsulated. The attack is performed using various electrical signals on the HSM's regular communication port and power supply pins. This is what the PMCT is designed to do. A cheap attack.
2. **Semi-invasive:** The top layer of the HSM's silicon die is removed. The electrical contacts on the silicon die are not exposed. The attack is performed by illuminating certain parts of the HSM microcircuits with pulses of light. A moderately expensive attack.
3. **Invasive:** The silicon die is fully exposed, so microprobes can be connected to the contacts on the die. The attack is performed by injecting electrical signals into the HSM microcircuits. This is an expensive attack that requires highly specialized equipment.

Powerful digital ChipWhisperers are available, of course [2]. But what I wanted to do was to make something based on older technology, similar to what Dr. Skorobogatov did [3], with all the analog and digital signals fully accessible, which may be a better learning platform for understanding the principles of non-invasive attacks.

Specifications

Non-invasive attacks are based on various fault injection methods like sending badly formatted data or sending signals of incorrect amplitude and frequency to any of the HSM's pins (including the power supply pins). These can make the DUT (Device Under Test, or rather Attack) perform many uncontrolled actions and hopefully reveal its secrets.

Non-invasive attacks require a lot of data processing, tweaking and hunting in the dark for every particular DUT, but if it is possible to crack it non-invasively, the cheap and simple PMCT can help you do it. It has the following features:

1. **Clock glitch generator:** Inserting signals that are too fast to the HSM's clock input (CLK) can make its CPU skip a machine

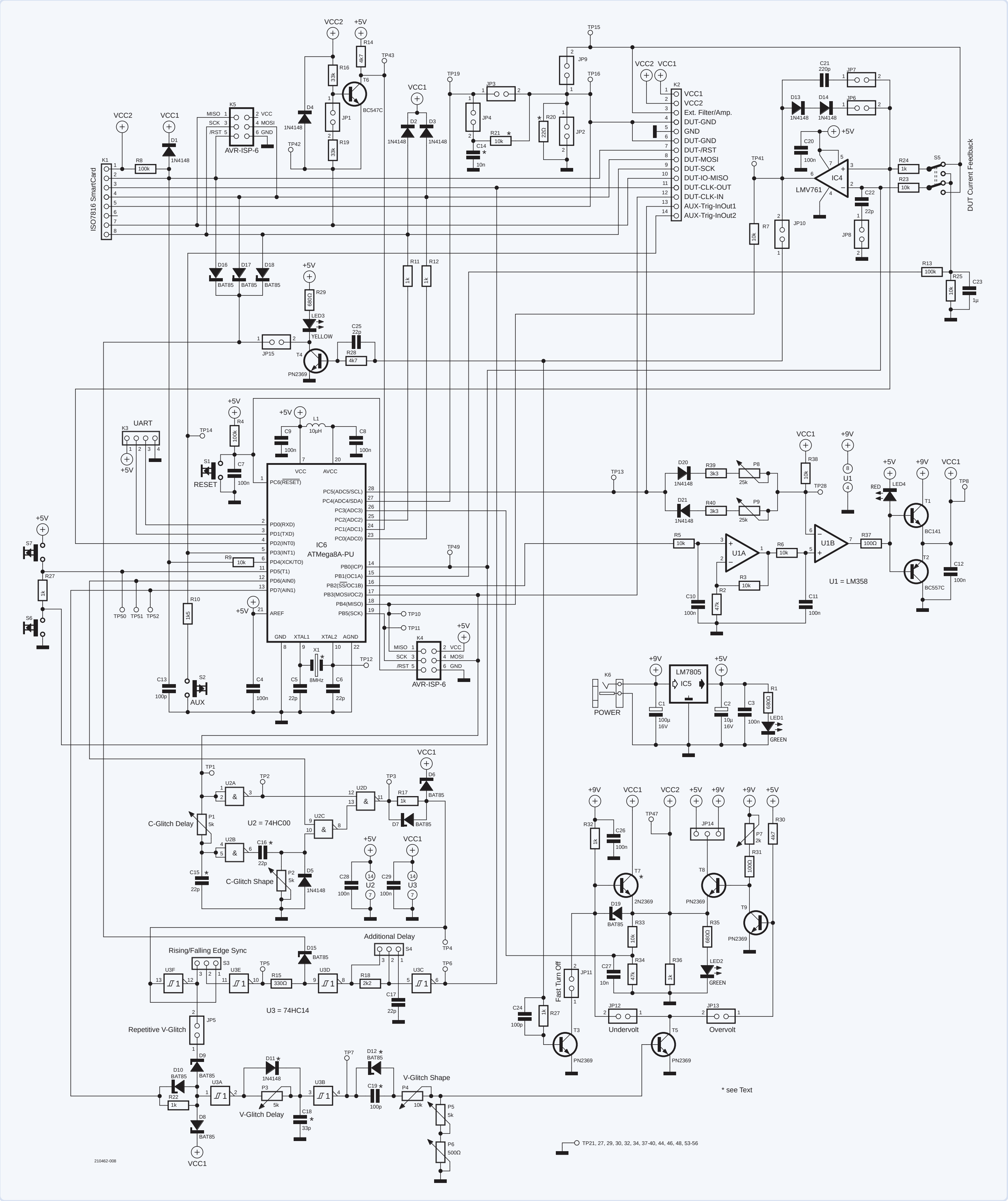


Figure 1: The schematic of the Poor Man's ChipTweaker. Note the use of fast transistors of type 2369. For heat dissipation reasons T7 should be a TO-18 type (metal can, 2N2369). The plastic PN2396 types used for the others might be cheaper and/or easier to find. Other parts marked with a '' are supposed to be mounted in a socket to allow for easy experimenting with different values.*

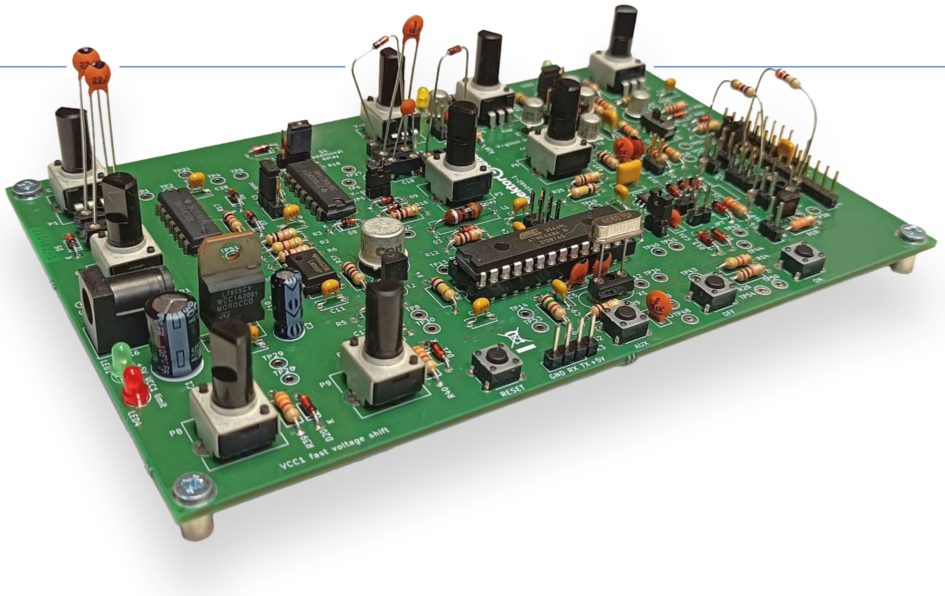


Figure 2: The Poor Man's ChipTweaker built on a comfortable PCB. The parts mounted in sockets are marked with a ''.*

instruction or perform it incorrectly, and thus e.g. skip a PIN check.

2. **Voltage glitch generator:** Changing the HSM's power supply voltage outside its rated operating range can also cause uncontrolled operation. The PMCT can perform slow, medium and fast voltage glitches, both undervoltage and overvoltage.
3. **Variable 1.0 V to 6.0 V DUT power supply:** Controlled by the PMCT's microcontroller.
4. **Fast DUT-turn-off comparator:** Above or below a certain power supply current threshold (for e.g. an EEPROM write sabotage attack), armed-disarmed or directly triggered by the main MCU.
5. **Bit-banged SPI and bidirectional single-pin UART interface** for communication with the DUT (including all types of SmartCards), with bidirectional voltage level shifters covering the range of 1.5 V to 6.0 V.

Besides aforementioned attacks, the PMCT can perform all sorts of timing and power-analysis attacks in combination with a digital scope (like a LabNation SmartScope), a 100-200 MHz analog storage scope (e.g., Tektronix 466) and a PC with software like MATLAB for offline data analysis.

Unlike the fully digital ChipWhisperers, all the analog and digital signals are accessible for studying and analysis, and the glitch timing can be even more precisely adjusted thanks to continuous analog adjustment with potentiometers without quantization steps. This is why a slow ATmega8 can coordinate the whole attack procedure, and no FPGA is needed. Glitches with a duration of less than 10 ns can be generated, which will induce errors in CPUs designed for operation up to 50 MHz and higher. Many SmartCards and MCUs operate at up to 20 MHz only.

Description of the Hardware

In the middle of the schematic (**Figure 1**), we find the microcontroller unit (MCU, IC6, ATmega8) that coordinates the attack procedures. The circuit around IC1 is a variable voltage source for the DUT. Its output voltage VCC1 is controlled using PWM by IC6's timer output OC1B. This allows the generation of slow voltage glitches. MCU pin PC5 can have one of three states — L, H and high-Z — allowing it to quickly change the amplification of IC1B and hence produce three different values of VCC1, set by P8 and P9. This is the way to generate a medium speed voltage glitch (measured in microseconds). The red LED4 will light as a warning and limit VCC1 to approximately 6 V, which is not too high for a 5 V device.

Generating Glitches

The DUT is normally powered by VCC2 through transistor T7 (a 2N2369 in a TO-18 package). T5 can quickly pull its base low (place jumper JP12), which results in a fast undervoltage glitch (in the order of 10 ns). If JP13 is placed, T8 will quickly pull the VCC2 high, creating a fast overvoltage glitch. Fast voltage glitches are shaped by C19, P4, P5 and P6. The delay of the voltage glitch with respect to the CLK pulse is adjusted with P3. When MCU output PD7 is set high, a single voltage glitch is fired. If it remains high, and if JP5 is shorted, voltage glitches will fire on each CLK pulse.

IC2 is used to generate a clock glitch on the DUT. The MCU's PWM output OC2 generates the CLK pulse for the DUT. If port PD6 is high, IC2C will pass a short glitch to IC2D on each CLK pulse. The polarity of the glitch can be selected with a jumper or switch on S3.

IC4 is a fast comparator with positive feedback (enhanced by D13, D14 and C21) that can turn off the DUT in a matter of nanoseconds. This can also be achieved with a command from the MCU (PD5 or PD2 to turn on, PB0 turn off the DUT) or by pushing buttons S6 and S7.

Current Consumption Attacks

IC4 can also be configured to turn off the DUT if the DUT's current consumption (measured with resistor R20) is higher or lower (selected by S5) than a threshold voltage (at C23, controlled by PWM on the OC1A pin). This is typically used to prevent the DUT from writing to its internal EEPROM (which consumes extra current). The output of IC4 activates T3, which turns off T7. It also activates transistor T4 (auxiliary turn-off) to pull all the SmartCard lines low and avoid any possible phantom back powering.

Analog input ADC4 monitors the DUT current. As this is a slow ADC, the signal can be filtered by R21/C14, or it can be additionally processed by a faster external filter/amplifier (at connector K2) if needed. For a timing attack or power-analysis attack, this signal is typically recorded by an oscilloscope for offline processing and analyzing with a tool like MATLAB.

As the MCU runs from 5 V while the DUT is powered from the (usually) lower voltage VCC2, the bidirectional level shifter (built around T6) is needed for the DUT's I/O pin 7 on connector K1 (a bidirectional UART port, as most SmartCards use nowadays). Pins 4 and 8 are used for SmartCards with SPI interface (e.g., the FunCard). Since they are unidirectional, their level shifters are simpler (diodes D2 and D3).

Printed Circuit Board

To facilitate building your own Poor Man's ChipTweaker a Euro-card-sized printed circuit board (PCB) was designed for it (**Figure 2**). (Refer to [4] for the KiCad project and Gerber files.) All the parts are through-hole type components except for IC4, which has a SOIC-8 package. Assembling the board should not pose any problems, but there are some things to be aware of:

- 3-pin jumpers 'Sxx' may be more practical as SPDT slide or toggle switches.
- You may prefer using trimmers instead of potentiometers. Trimmers are easier to find and offer a wider range of values.
- T7 must be a 2N2369 in a TO-18 metal can for heat dissipation reasons. The TO-92 PN2396 transistors may be replaced by 2N2369 types, whichever is more convenient to source.
- Parts marked with '*' (e.g., D11 & C18, etc., except for T7) are supposed to be mounted in a socket to allow for easy experimenting with different values and types.

The board has many test points with a ground connection always nearby. Don't try to understand their numbering; there is no logic to them.

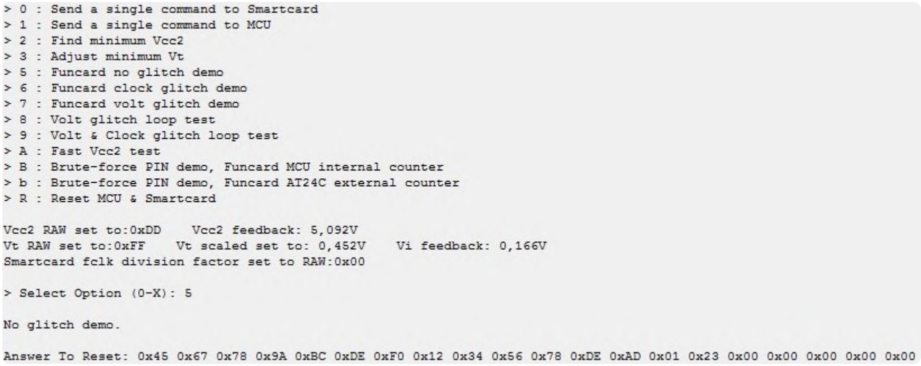
Prepare for Attack

The DUT for the demonstration applications is a standard-issue FunCard (containing a Microchip AVR AT90S8515 MCU together with an AT24C EEPROM). Without any serious protections, it's relatively easy to attack, so it's your first step if you want to advance in this area. Load the FunCard with the firmware found at [4] by means of ISP connector K5. Then use one of the two ATmega8 firmware variants (for attacking a FunCard, also at [4]) to program the ChipTweaker's MCU IC6 through ISP connector K4 (and not K5). You can now try out some basic actions.

Connect a serial terminal to the UART port K3 and configure it for 9600-8-N-1 (a.k.a. 9600n81). After pressing the reset button S1, you should get a screen like the one shown in **Figure 3**. Options 0 and 1 are for sending simple commands to the MCU and to the FunCard. All the commands are 4 bytes long, where the last byte is always equal to 0x0d (CR). Refer to **Table 1** and **Table 2**.

Clock-Glitch Attack Demo

This is a demo of a basic clock-glitch attack. The FunCard performs several 16-bit operations on two input operands, located in its MCU's internal EEPROM at addresses 0x00, 0x01, 0x02 and 0x03 in big-endian form. Glitching a clock pulse (sending a pulse which is too fast for the DUT) at various stages of the calculations produces different errors.



```
> 0 : Send a single command to Smartcard
> 1 : Send a single command to MCU
> 2 : Find minimum Vcc2
> 3 : Adjust minimum Vt
> 5 : Funcard no glitch demo
> 6 : Funcard clock glitch demo
> 7 : Funcard volt glitch demo
> 8 : Volt glitch loop test
> 9 : Volt & Clock glitch loop test
> A : Fast Vcc2 test
> B : Brute-force PIN demo, Funcard MCU internal counter
> b : Brute-force PIN demo, Funcard AT24C external counter
> R : Reset MCU & Smartcard

Vcc2 RAW set to:0xDD    Vcc2 feedback: 5,092V
Vt RAW set to:0xFF    Vt scaled set to: 0,452V    Vi feedback: 0,166V
Smartcard fclk division factor set to RAW:0x00

> Select Option (0-X): 5

No glitch demo.

Answer To Reset: 0x45 0x67 0x78 0x9A 0xBC 0xDE 0xF0 0x12 0x34 0x56 0x78 0xDE 0xAD 0x01 0x23 0x00 0x00 0x00 0x00 0x00
```

Figure 3: Choose the attack you want to execute from a menu.

Command	Description
V2x	'x' is an 8-bit number defining the PWM ratio for OC1B to set the VCC2 voltage.
Vtx	'x' is an 8-bit number defining the PWM ratio for OC1A to set the Vt voltage, a threshold at C23 to activate IC4.
Fxy	'x' is a raw frequency division factor, for generating a CLK pulse for DUT at OC2. If set at 0, the OC2 will run at half of the MCU crystal frequency; 'y' is ignored.
GLx	The lowest 3 bits of 'x' set the MCU pins PD7, PD6 & PC5; for testing.
onx	Turn on the DUT; 'x' is ignored.
ofx	Turn off the DUT; 'x' is ignored.
Sxy	Save the current settings to the MCU EEPROM; 'x' and 'y' are ignored.

Table 1: Commands to the MCU. Terminate every command with 0x0d (CR).

Command	Description
Rxy	Read a byte from address 'x' of the internal EEPROM of the AT90S; 'y' is ignored.
Wxy	Write byte 'y' to address 'x' of the internal EEPROM of the AT90S.
Pxy	Test a PIN number 'xy' (in BCD format). The PIN is stored in the internal EEPROM of the AT90S.
rxy	Read a byte from address 'x' of the AT24C EEPROM; 'y' is ignored.
wxy	Write byte 'y' to address 'x' of the AT24C EEPROM.
pxy	Test a PIN number 'xy' (in BCD format). The PIN is stored in AT24C EEPROM.

Table 2: Commands to SmartCard. Terminate every command with 0x0d (CR).

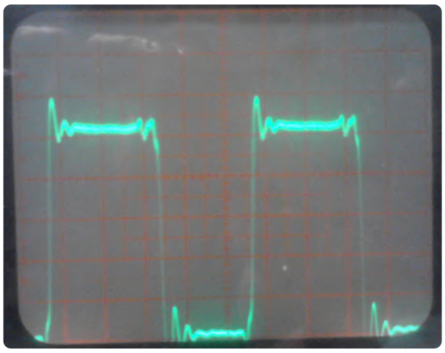


Figure 4: A clean clock signal without glitches. The oscilloscope was set to 50 ns/div on the horizontal axis and 1 V/div on the vertical axis.

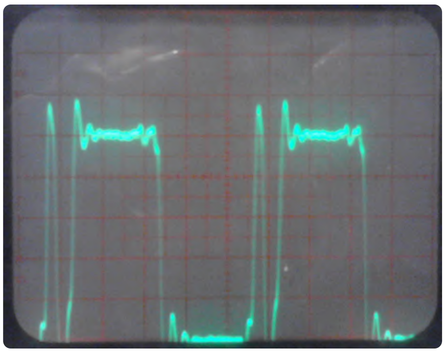


Figure 5: A glitched clock signal (50 ns/div horizontal, 1 V/div vertical axis). Adjust potentiometers P1 and P2 to create a glitch as shown here.

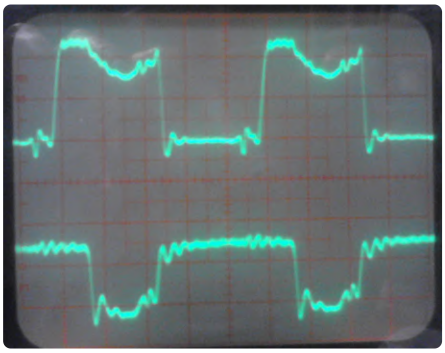


Figure 7: The voltage glitch fired 30 to 40 ns after the rising edge of the clock pulse. The upper trace is the clock signal, the lower is the power supply voltage VCC2 (50 ns/div horizontal; 1 V/div vertical).

Maybe the math operation result will be wrong, or simply one machine instruction will be skipped.

Refer to **Figure 4**, **Figure 5**, and **Figure 6** for the signals and the results. An effective glitch for the FunCard is an active low pulse with a duration between 10 ns and 20 ns fired about 10 ns to 20 ns after the rising edge of the CPU clock (see **Figure 5**).

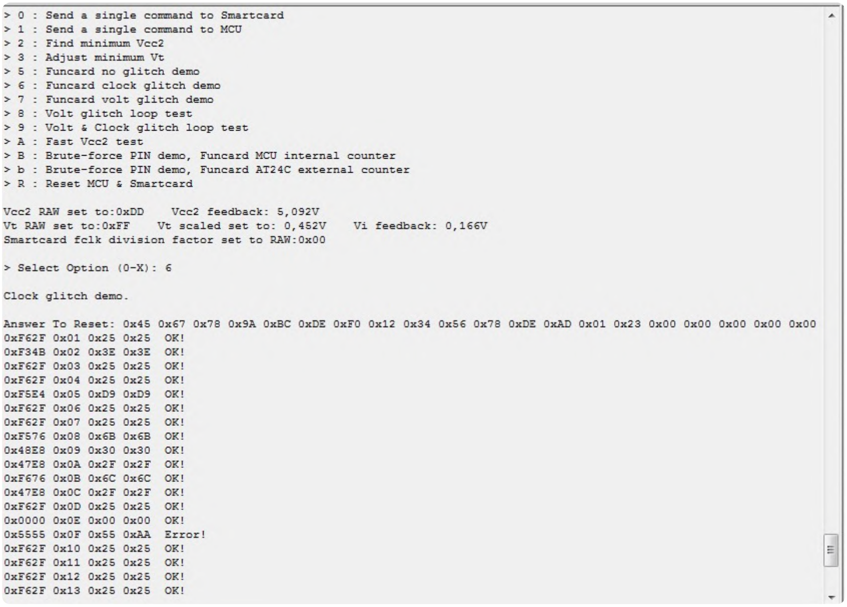
Voltage-Glitch Attack Demo

This is a demo of a basic voltage-glitch attack. The FunCard performs the same operations as in the previous example. An effective undervoltage glitch has a duration of at least 50 ns and produces good results with the timing shown in **Figure 7**. Here the glitch fired 30 to 40 ns after the rising edge of the clock pulse. The power supply voltage for the FunCard was set to 2.24 V. The voltage glitch was about 1.5 V deep.

Adjust the shape of the voltage glitch with P4, P5, and P6. Use P3, S3 and S4 to adjust glitch delay and sync. The results are displayed in the same way as in the previous example (**Figure 8**).

Cracking the Memory Protection to Extract the Firmware

Every protected MCU or a SmartCard requires a different attack procedure. It requires a lot of trial-and error and hunting in the dark to find possible weak spots. The procedure that works on many Microchip AVR microcontrollers is a slow undervoltage glitch.



```
> 0 : Send a single command to Smartcard
> 1 : Send a single command to MCU
> 2 : Find minimum Vcc2
> 3 : Adjust minimum Vt
> 5 : Funcard no glitch demo
> 6 : Funcard clock glitch demo
> 7 : Funcard volt glitch demo
> 8 : Volt glitch loop test
> 9 : Volt & Clock glitch loop test
> A : Fast Vcc2 test
> B : Brute-force PIN demo, Funcard MCU internal counter
> b : Brute-force PIN demo, Funcard AT24C external counter
> R : Reset MCU & Smartcard

Vcc2 RAW set to:0xDD    Vcc2 feedback: 5,092V
Vt RAW set to:0xFF    Vt scaled set to: 0,452V    Vi feedback: 0,166V
Smartcard fclk division factor set to RAW:0x00

> Select Option (0-X): 6

Clock glitch demo.

Answer To Reset: 0x45 0x67 0x78 0x9A 0xBC 0xDE 0xF0 0x12 0x34 0x56 0x78 0xDE 0xAD 0x01 0x23 0x00 0x00 0x00 0x00 0x00
0xF62F 0x01 0x25 0x25  OK!
0xF34B 0x02 0x3E 0x3E  OK!
0xF62F 0x03 0x25 0x25  OK!
0xF62F 0x04 0x25 0x25  OK!
0xF5E4 0x05 0xD9 0xD9  OK!
0xF62F 0x06 0x25 0x25  OK!
0xF62F 0x07 0x25 0x25  OK!
0xF576 0x08 0x6B 0x6B  OK!
0x48E8 0x09 0x30 0x30  OK!
0x47E8 0x0A 0x2F 0x2F  OK!
0xF676 0x0B 0x6C 0x6C  OK!
0x47E8 0x0C 0x2F 0x2F  OK!
0xF62F 0x0D 0x25 0x25  OK!
0x0000 0x0E 0x00 0x00  OK!
0x5555 0x0F 0x55 0xAA  Error!
0xF62F 0x10 0x25 0x25  OK!
0xF62F 0x11 0x25 0x25  OK!
0xF62F 0x12 0x25 0x25  OK!
0xF62F 0x13 0x25 0x25  OK!
```

Figure 6: Terminal window showing the clock-glitch attack results. ***Column 1*** *is the 16-bit result of the math operation as executed by the DUT. It will be wrong if the glitch was successful.* ***Column 2:*** *glitch delay. Glitching on different moments gives different results.* ***Column 3:*** *checksum as the sum of the two bytes of the 16-bit result as calculated by the FunCard.* ***Column 4:*** *checksum calculated by the PMCT's MCU.* ***Column 5:*** *'Error!' if columns 3 and 4 don't match.*

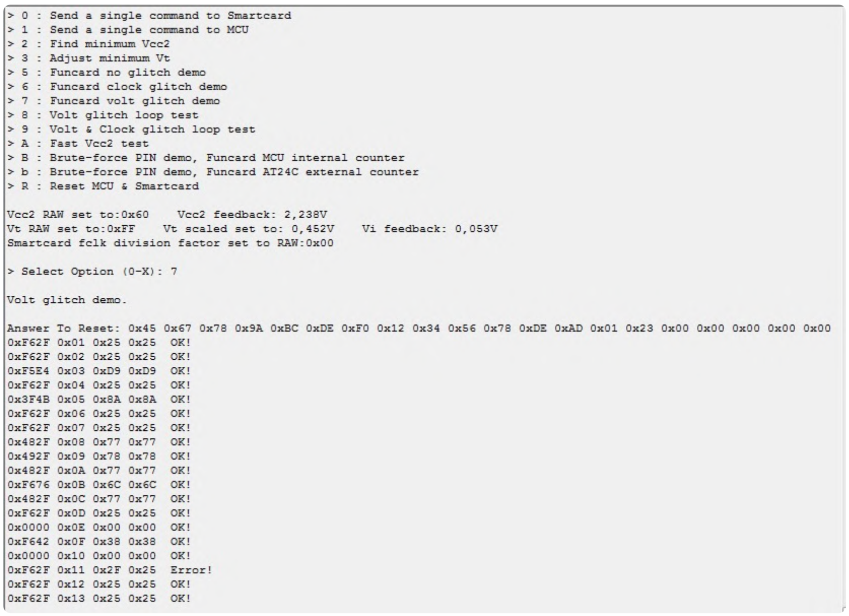
It works like this. When the FunCard MCU (an AT90S8515) is protected, both memory lock bits are programmed to 0. The flash memory then can't be read, only erased. In serial programming mode (its reset input pulled down to 0 V) the *Chip Erase* command will first erase the flash memory (revert all the flash bytes to 0xFF) and then erase the lock bits (deactivate them by reverting them to 1).

The design flaw present on many AVR MCUs (not all) allows us to crack the protection in the following way. If the power supply voltage is lowered below the rated minimum (2.7 V), down to 1.6 V to 1.7 V, there won't be enough power to erase the flash memory, but erasing the lock bits will still be possible. The attack is started at 1.1 V and the voltage is gradually increased. The lock bits were successfully removed at 1.62 V without erasing the flash memory!

When you try to read the firmware or the device signature from a protected AVR microcontroller, the response will be "0x00, 0x01, 0x02, 0x03" and so you know the memory is protected (**Figure 9**). Once you get the correct signature ("0x1E, 0x93, 0x01" for the AT90S8515) the memory lock bits have been removed and the program memory can now be read using any ISP programmer.

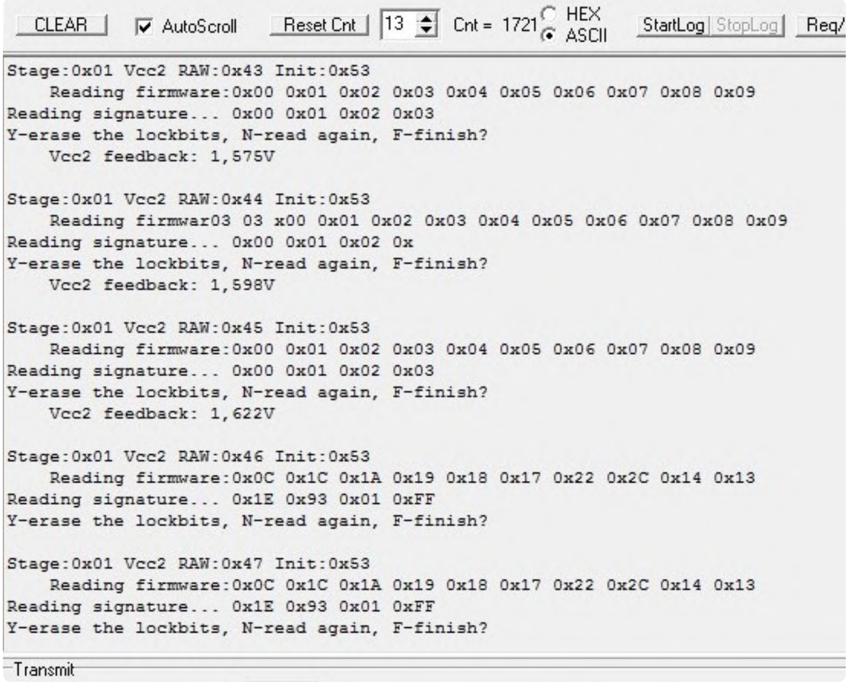
Power-Analysis Attack on a Bank Card

Bank cards are becoming more sophisticated and use multiple methods to physically protect critical memory areas. They also try to mask the power supply current, so critical operations can't be easily detected. If certain procedures, like checking the entered PIN, can be precisely pinpointed in time, then we know



```
> 0 : Send a single command to Smartcard
> 1 : Send a single command to MCU
> 2 : Find minimum Vcc2
> 3 : Adjust minimum Vt
> 5 : Funcard no glitch demo
> 6 : Funcard clock glitch demo
> 7 : Funcard volt glitch demo
> 8 : Volt glitch loop test
> 9 : Volt & Clock glitch loop test
> A : Fast Vcc2 test
> B : Brute-force PIN demo, Funcard MCU internal counter
> b : Brute-force PIN demo, Funcard AT24C external counter
> R : Reset MCU & Smartcard

Vcc2 RAW set to:0x60    Vcc2 feedback: 2,238V
Vt RAW set to:0xFF    Vt scaled set to: 0,452V    Vi feedback: 0,053V
Smartcard fclk division factor set to RAW:0x00

> Select Option (0-X): 7

Volt glitch demo.

Answer To Reset: 0x45 0x67 0x78 0x9A 0xBC 0xDE 0xF0 0x12 0x34 0x56 0x78 0xDE 0xAD 0x01 0x23 0x00 0x00 0x00 0x00 0x00
0xF62F 0x01 0x25 0x25  OK!
0xF62F 0x02 0x25 0x25  OK!
0xF5E4 0x03 0xD9 0xD9  OK!
0xF62F 0x04 0x25 0x25  OK!
0x3F4B 0x05 0x8A 0x8A  OK!
0xF62F 0x06 0x25 0x25  OK!
0xF62F 0x07 0x25 0x25  OK!
0x482F 0x08 0x77 0x77  OK!
0x492F 0x09 0x78 0x78  OK!
0x482F 0x0A 0x77 0x77  OK!
0xF676 0x0B 0x6C 0x6C  OK!
0x482F 0x0C 0x77 0x77  OK!
0xF62F 0x0D 0x25 0x25  OK!
0x0000 0x0E 0x00 0x00  OK!
0xF642 0x0F 0x38 0x38  OK!
0x0000 0x10 0x00 0x00  OK!
0xF62F 0x11 0x2F 0x25  Error!
0xF62F 0x12 0x25 0x25  OK!
0xF62F 0x13 0x25 0x25  OK!
```

Figure 8: Terminal window showing the voltage-glitch attack results. The columns are formatted in the same way as for the clock glitch attack (see Figure 6).



```
Stage:0x01 Vcc2 RAW:0x43 Init:0x53
    Reading firmware:0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
Reading signature... 0x00 0x01 0x02 0x03
Y-erase the lockbits, N-read again, F-finish?
    Vcc2 feedback: 1,575V

Stage:0x01 Vcc2 RAW:0x44 Init:0x53
    Reading firmwar03 03 x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
Reading signature... 0x00 0x01 0x02 0x
Y-erase the lockbits, N-read again, F-finish?
    Vcc2 feedback: 1,598V

Stage:0x01 Vcc2 RAW:0x45 Init:0x53
    Reading firmware:0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
Reading signature... 0x00 0x01 0x02 0x03
Y-erase the lockbits, N-read again, F-finish?
    Vcc2 feedback: 1,622V

Stage:0x01 Vcc2 RAW:0x46 Init:0x53
    Reading firmware:0x0C 0x1C 0x1A 0x19 0x18 0x17 0x22 0x2C 0x14 0x13
Reading signature... 0x1E 0x93 0x01 0xFF
Y-erase the lockbits, N-read again, F-finish?

Stage:0x01 Vcc2 RAW:0x47 Init:0x53
    Reading firmware:0x0C 0x1C 0x1A 0x19 0x18 0x17 0x22 0x2C 0x14 0x13
Reading signature... 0x1E 0x93 0x01 0xFF
Y-erase the lockbits, N-read again, F-finish?
```

Figure 9: Cracking the flash memory protection.

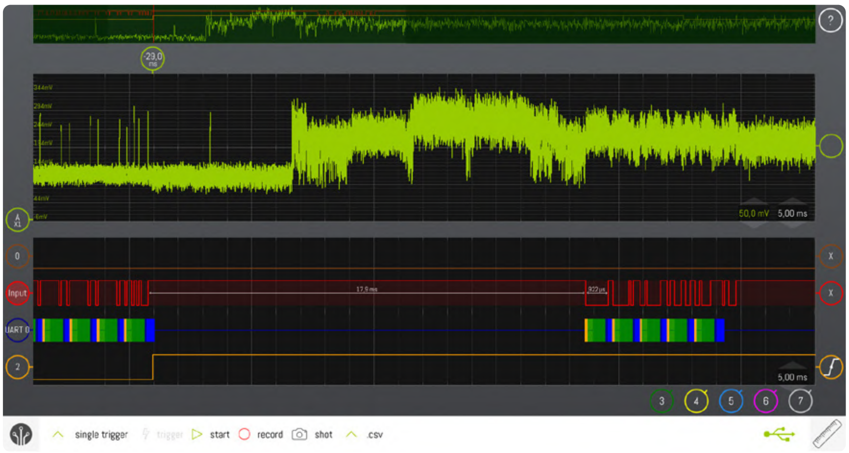


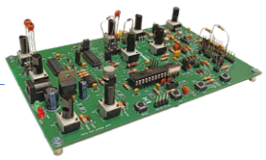
Figure 10: Power supply current (yellow trace) of a bank card when checking an incorrect PIN.

when glitches can be fired to try to bypass the protections.

Some of the known bank card protection methods are:

1. Using an internal RC clock for security-critical operations, switching to external CLK only for precise timing, e.g. when communicating through the UART. This prevents clock-glitch attacks.
2. Using very fast charge pumps on small internal (pF range) power supply capacitors to maintain a stable power supply voltage. This prevents voltage-glitch attacks.
3. Changing the half-periods of the internal RC clock randomly (based on an internal True Random Number Generator, TRNG). This way, the critical operations (like checking the PIN) won't always happen at the same time, which will make attacks more difficult.
4. Adding random noise to the power supply current. This prevents power-analysis attacks (**Figure 10**).
5. When checking a PIN, first decrease the PIN try counter in the SmartCard's EEPROM, then check the PIN and increase the PIN try counter if the PIN is OK. Old cards used to only write to EEPROM to decrease the counter on a wrong PIN entry. This enabled brute force attacks to extract the PIN by quickly cutting the power supply after each wrong try.
6. Using very carefully crafted functions for security-critical operations (assembler programming required here!), which always spend the same amount of CPU clock cycles and (possibly) the same amount of power, regardless of their input variables. This prevents timing and power-analysis attacks.
7. Extending a security-critical operation, which normally takes less than 1 ms to for instance 200 ms. A useful 1 ms signal is thus hidden inside 199 ms of garbage data processing.

Defeating a modern bank card non-invasively (if at all possible) requires extensive analysis of recorded data and long and painstaking work. Read the expert book *Power Analysis Attacks* [5] for further comprehensive and detailed information on power-analysis attacks.



Test and Learn

The Poor Man's ChipTweaker PMCT is not a hi-tech rig, but it can be used for testing and learning non-invasive attacks on HSMs. I hope you found this article interesting, at least as a good starting point. Designing and attacking single-chip HSMs nowadays is very hard work for everyone involved. This is exactly why this area remains open for further research.

Happy spying!

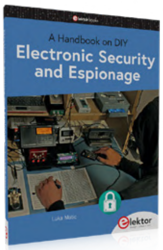
210462-01

Questions or Comments?

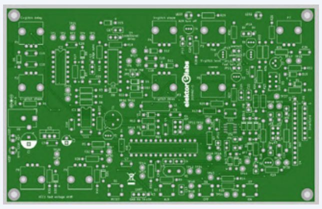
Do you have technical questions or comments about this article? Email the author at luka.matic@fer.hr or contact Elektor at editor@elektor.com.

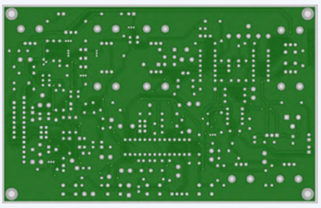
Related Products

- **LabNation SmartScope USB Oscilloscope (SKU 17169)** www.elektor.com/17169
- **Luka Matic, *Electronic Security and Espionage,* Elektor 2021 (SKU 19903)** www.elektor.com/19903



COMPONENT LIST





Resistors

R1,R29,R35 = 680 Ω
R2,R34 = 47 kΩ
R3,R5,R6,R7,R9,R23,R25,R33,R38 = 10 kΩ
R4,R8,R13 = 100 kΩ
R10 = 1.5 kΩ
R11,R12,R17,R22,R24,R26,R27,R32,R36 = 1 kΩ
R14,R28,R30 = 4.7 kΩ
R15 = 330 Ω
R16,R19 = 33 kΩ
R18 = 2.2 kΩ
R20* = 22 Ω*
R21* = 10 kΩ*
R31,R37 = 100 Ω
R39,R40 = 3.3 kΩ
P1,P2,P3,P5 = Potentiometer, 5 kΩ, vertical
P4 = Potentiometer, 10 kΩ,vertical
P6 = Potentiometer, 500 Ω, vertical
P7 = Potentiometer, 2 kΩ, vertical
P8,P9 = Potentiometer, 25 kΩ,vertical

Capacitors

C1 = 100 μF 16V, 3.5 mm pitch
C2 = 10 μF 16V, 2.5 mm pitch
C3,C4,C7,C8,C9,C10,C11,C12,C20,C26,C28,C29 = 100 nF, 5 mm pitch
C5,C6,C17,C22,C25 = 22 pF, 2.5 mm pitch
C13,C24 = 100 pF, 2.5 mm pitch
C14* = 10 nF, 5 mm pitch*
C27 = 10 nF, 5 mm pitch
C15*,C16* = 22 pF, 5 mm pitch*
C18* = 33 pF, 5 mm pitch*
C19* = 100 pF, 5 mm pitch*
C21 = 220 pF, 2.5 mm pitch
C23 = 1 μF, 5 mm pitch

Inductors

L1 = 10 μH

Semiconductors

D1,D2,D3,D4,D5,D13,D14,D20,D21 = 1N4148
D11* = 1N4148*
D6,D7,D8,D9,D10,D15,D16,D17,D18,D19 = BAT85
D12* = BAT85*
IC1 = LM358
IC2 = 74HC00
IC3 = 74HC14
IC4 = LMV761, SOIC8
IC5 = 7805, TO220
IC6* = ATmega8A-PU*
LED1,LED2 = LED, 3 mm, green
LED3 = LED, 3 mm, yellow
LED4 = LED, 3 mm, red
T1 = BC141, TO39
T2 = BC557C
T3,T4,T5,T8,T9 = PN2369, TO92
T6 = BC547C
T7 = 2N2369, TO18

Miscellaneous

JP1-JP13,JP15 = 2-way pinheader, 2.54 mm pitch
JP14,S3,S4 = 3-way pinheader, 2.54 mm pitch
K1 = 8-way pinheader, 2.54 mm pitch
K2 = 14-way pinheader, 2.54 mm pitch
K3 = 4-way pinheader, 2.54 mm pitch
K4,K5 = 2-row 6-way pinheader, 2.54 mm pitch
K6 = Barrel jack
S5 = Slide switch (C&K JS202011CQN)
X1* = Crystal 8 MHz*

* mount with socket for easy experimenting

Funcard. (Source: www.cellularcenter.it)

WEB LINKS

[1] Luka Matic, "Tamper-Evident Box", Elektor 5/ 2020: https://www.elektormagazine.com/magazine/elektor-146/58537
[2] ChipWhisperers from NewAE Technology Inc.: https://www.newae.com/chipwhisperer
[3] Sergei P. Skorobogatov, "Copy Protection in Modern Microcontrollers": https://www.cl.cam.ac.uk/~sps32/mcu_lock.html
[4] Downloads for this article at Elektor Labs: https://www.elektormagazine.com/labs/poor-mans-chipwhisperer-or-a-smartcard-tweaker
[5] S. Mangard, E. Oswald, T. Popp, Power Analysis Attacks: Revealing the Secrets of Smart Cards, Springer, 2007: https://amzn.to/3RWBtuy

elektor TV

Elektor TV Shows

Elektor **Engineering Insights**

Elektor Industry Insights is a go-to resource for busy engineers and maker pros who want to stay informed about the world of electronics. During each episode, Stuart Cording (Editor, Elektor) will discuss real engineering challenges and solutions with electronics industry experts.
www.elektormagazine.com/elektor-engineering-insights

Elektor LabTalk

Are you passionate about DIY electronics, embedded programming, or engineering theory? Join Elektor Lab team engineers and editors as they share engineering tips, plan future electronics projects, discuss Elektor Mag and answer community questions.
www.elektormagazine.com/elektor-lab-talk

Debugging Techniques for Arduino

Approaches to finding bugs and fixing your code.

elektor academy

Do you want to enhance your electronics skills? Turn to Elektor Academy for resources to level up your engineering capabilities. Our Expert Stuart Cording will guide you through the Elektor academy courses
www.elektormagazine.com/elektor-academy

Stay informed and join our
YouTube channel Elektor TV
www.youtube.com/c/ElektorIM

elektor
design > share > earn

USB True Random Number Generator

Two PICs for the Price of One AVR

By Matthias Wolf (Germany)

A good True Random Number Generator (TRNG) is a useful thing to have. Secure data encryption, for instance, is almost impossible without one. Gaming and gambling applications also require top-class TRNGs. Elektor published an analog TRNG in 2017; in this update, we add USB to it.

The True Random Number Generator (TRNG) based on inexpensive components by Luka Matic [1] used an SD card to store the random sequence it generated. In these pages we present an adaptation of this work where the memory card slot is replaced by a USB interface.

The new circuit (**Figure 1**) has two Microchip Technology PIC microcontrollers instead of a single ATtiny2313A: one for the analog signal processing part (PIC16F19156) and one for the USB interface (PIC18F25K50). Both MCUs communicate with each other via a SPI bus, galvanically isolated with high-speed optocouplers to prevent digital USB circuit noise from disturbing the analog signal.

Filter Calibration

A small PC application was written for calibrating the analog filters in real time in an easy way (**Figure 2**).

My filter settings for the analog filters are as follows:

- S5: C37,C38,C39,C40,C41 ON with C39 at 109 pF
- S6: C45,C46,C47 ON with C47 at 25 pF
- S7: Switch1 ON and P6 at 409 Ω

These settings depend on the components used to build the TRNG, you may need different ones.

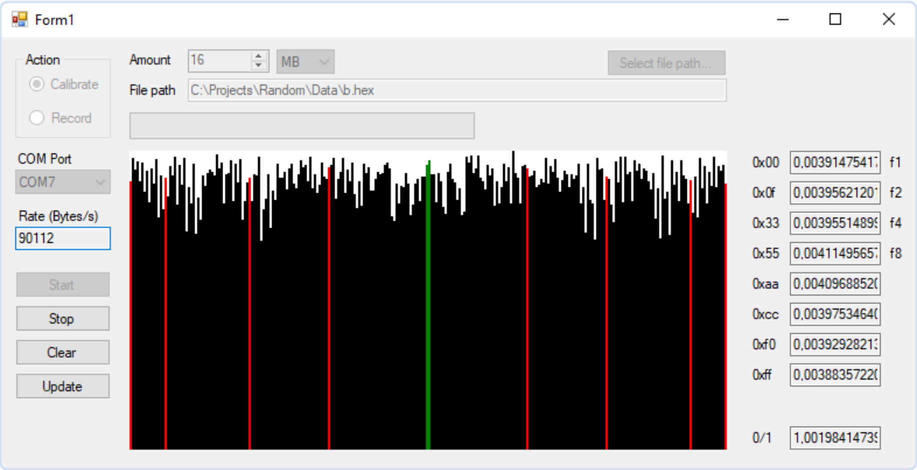
The type of the Zener diodes used is also important. I first used the BZX384C12-E3-08 from Vishay, but its noise was bad.

Finally, compared to the original TRNG from Luka, the PIC16 uses a slightly higher sampling rate (800 kHz, 1.25 µs per sample).

Go Get the Files!

All the project files can be downloaded from [2]. They include the schematics (three pages), the Target 3001 CAD project, firmware for the PIC16 (analog sampling, MPLAB X IDE with PCM C compiler from CCS), firmware for the PIC18 (USB handling, MPLAB X IDE with XC8), and the desktop application (C#, Visual Studio 2017 commmunity edition).

190235-01



○ Calibrate
○ Record

Amount: 16 MB

File path: C:\Projects\Random\Data\b.hex

COM Port: COM7

Rate (Bytes/s): 90112

0x00	0,0039147541	f1
0x0f	0,0039562120	f2
0x33	0,0039551489	f4
0x55	0,0041149565	f8
0xaa	0,0040968852	
0xcc	0,0039753464	
0xf0	0,0039292821	
0xff	0,0038835722	
0/1	1,0019841473	

Figure 2: Screenshot of the PC desktop application for calibrating the analog filters and recording random data.

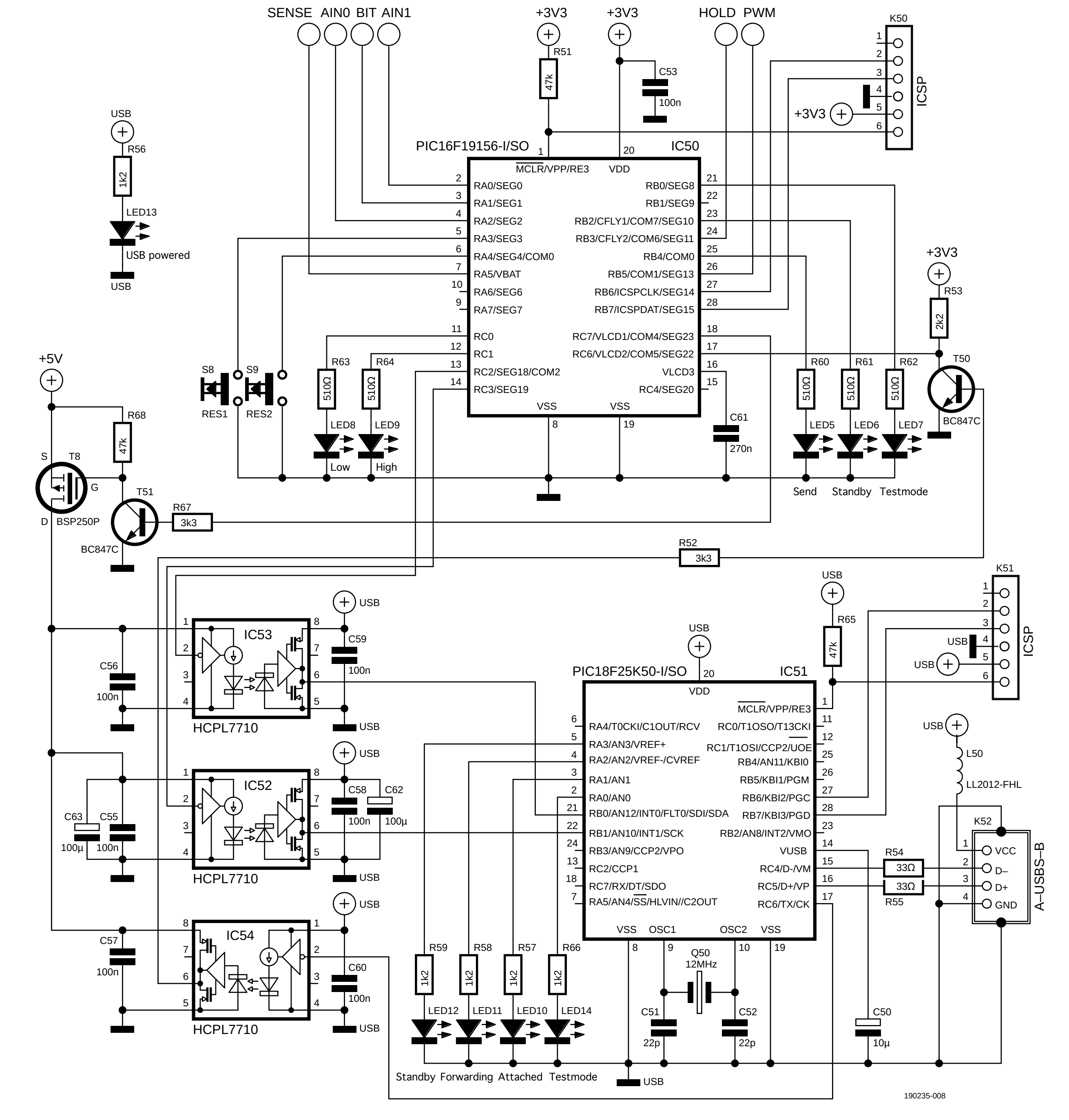


Figure 1: The digital part of the original TRNG from 2017 with its ATtiny2313A MCU has been replaced by two PIC microcontrollers. The USB port replaces the SD card slot.

Questions or Comments?

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

Related Products

- **L. Matic, *Electronic Security and Espionage* (Elektor 2021, SKU 19903)**
www.elektor.com/electronic-security-and-espionage

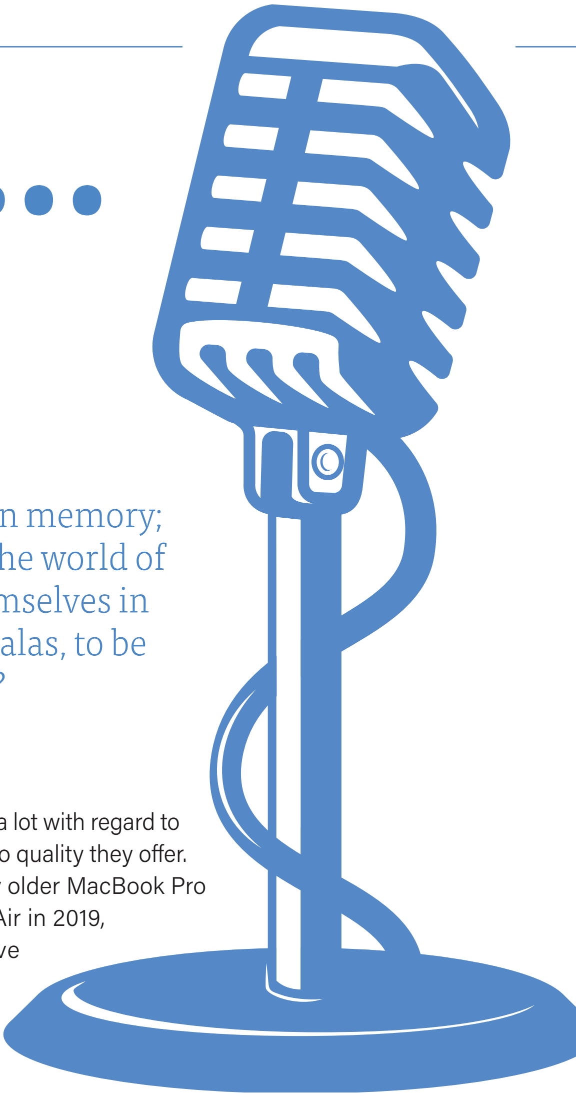
WEB LINKS

[1] L. Matic, "Truly Random-Number Generator," Elektor 3/2017: www.elektormagazine.com/matic-trng-2017
[2] This project at Elektor Labs: www.elektormagazine.com/labs/usb-random-number-generator

Pimp My Mic...

Self-Designed Level Booster

By **Dr. Thomas Scherer** (Germany)



'Corona' will not be the only reason that the last years remain in memory; it will also be because a lot of us were suddenly introduced to the world of online meetings - and not only those who suddenly found themselves in home offices. Poor quality of video during online-meetings is, alas, to be expected, but does this also have to apply to the sound quality?

Now that *Skype* is no longer a foreign term, and *Zoom* and *Teams* have entered the vernacular, ever more of us have been subjected to the poor experience and frustration that these video-conferencing services present. In addition to the operating errors of some participants, the limited bandwidth of the Internet connections, the poor quality of the video images (due to bad cameras and/or bad lighting), it is the poor quality of the audio transmission which adds to the list of annoyances. This is especially true for those who travel with a laptop. Unlike with a desktop PC where you can choose your hardware, you're likely to be stuck with a poor quality webcam.

Laptop frustrations

Due to many of my friends, acquaintances and relatives being scattered all over the world, I have been using 'video telephony' quite frequently for a long time. Since I also use portable computers when 'on the road,' I have already cursed a lot with regard to the quality of the video quality they offer. After I exchanged my older MacBook Pro for a new MacBook Air in 2019, I could hardly believe what an unbelievably noisy and underexposed image the camera provided; Apple dares to sell this for a lot of money. I could have just bought an extra webcam and plugged it in via USB, but you buy something portable precisely because you *don't* want to carry a load of extra equipment around with you, right? In addition, the MacBook Air only has USB-C interfaces. These may be very advanced but webcams with USB-C are scarce and having to carrying around an extra USB-C/USB-A adapter for regular webcams exceeded my capacity to suffer. Meanwhile, the new laptop was sold and I bought a MacBook Air with an M2 CPU, which is a little better.

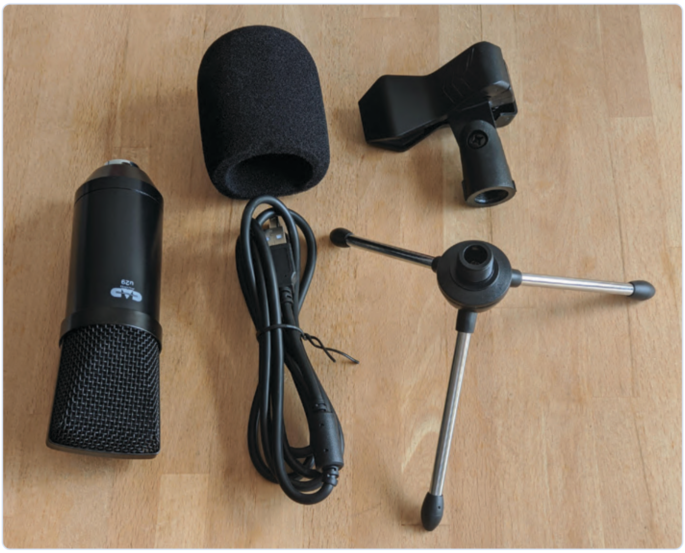


Figure 1: Included in the package are the actual microphone, a foam mic cover, a clamp (including tripod) and a USB cable.

The problem of inferior cameras (and microphones) in folding computers is not limited to Apple by the way. Even Windows 10/11 users can be affected. I know at least one (expensive) Samsung notebook where the picture noise is obvious and the sound is terrible. But there are also portable computers with mediocre video quality. It's a pity that exactly these aspects are neglected in the tests of relevant magazines. By now the relevance of the cameras should have become clear to the manufacturers. By the way, it is difficult to comprehend why every cheap smartphone has a much better front camera and a halfway usable microphone, but laptops still don't!

Remedy

The easiest way to solve all these image and sound problems is to buy a webcam. This is,of course, obligatory for a PC but it is also worth considering after the purchase of a laptop for your home office. It is unlikely that anyone will be satisfied with an extremely inexpensive, no-name webcam from a Chinese supplier. However, both Logitech,

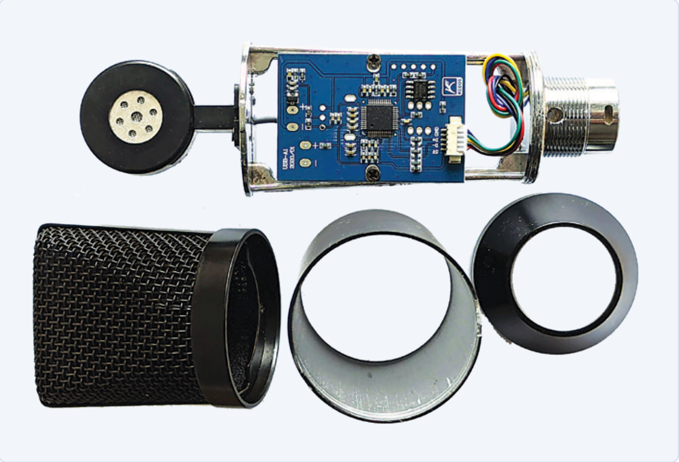


Figure 2: The new microphone disassembled into its individual parts.

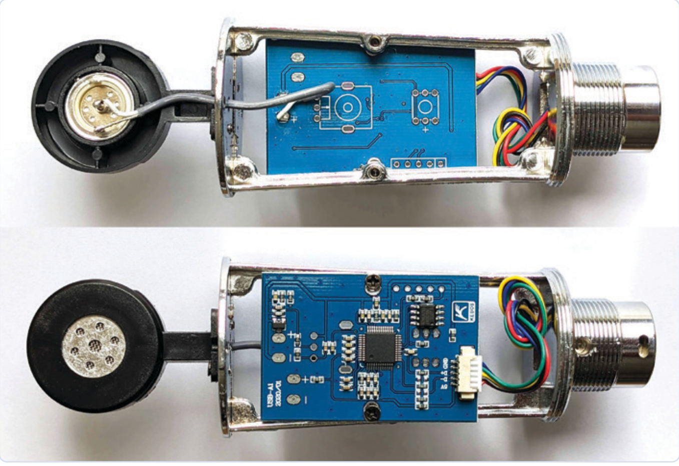


Figure 3: The inside of the microphone from the front and the back.

Microsoft and other known-brand manufacturers are offering webcams starting from as little as 35 €. The built-in microphones are not perfect, but they are good enough and usually sufficient for voice transmission. Should the quality of the camera be sufficient, but the sound quality not, you might consider buying an external microphone. The industry has recognised this market and offers several models with USB instead of the more typical jack plug. Thanks to the integrated A/D converter and the integration of the necessary drivers in all common operating systems, the connection of such devices to a PC running Windows, MacOS or Linux is completely problem-free. Of course you could also use 'normal' microphones with an analog output - at least on most desktop PCs. However, modern laptops are increasingly removing the microphone input. As a replacement, a cheap USB audio converter would be a good choice that offers such an input and, usually, an audio output too. But this leaves you with another small part to lose... not the solution for me.

USB Microphone

Having never really been satisfied with various inexpensive microphones over recent years, I recently ordered a large diaphragm microphone with a USB plug. These condenser microphones are also used in recording studios and should, therefore, be quite useful. Wikipedia provides insights on the characteristics of microphones under the keyword "Mikrofonierung" [1] (English: to mic up), the situation-specific selection and placement of microphones for recording sound sources.

Furthermore, due to the enormous growth of social media, many (younger generation) people produce podcasts or actively run their own video channel. This requires not only a good video camera but also a good microphone. It is precisely for such purposes that 'podcast kits', consisting of a large diaphragm microphone (sometimes even with a spider), stand, boom mount, and pop filter are on offer today.

So, I wanted to try my luck in this sector. Two phenomena are particularly apparent in audio technology: firstly, not every argument put forward is necessarily rational and, secondly, quality quickly becomes really expensive. Unfortunately, the price/performance ratio is not a gently rising straight line but, instead, has an exponential curve. And, since a better sound was only worth a limited amount of money to me, I Googled inexpensive large-diaphragm USB microphones...and found what I was looking for.

CAD Audio U29

This microphone comes with a tripod and pop filter, and costs only 34 €. I could not do much wrong in purchasing it, I thought. So I ordered it and, two days later and, after unpacking it, I had the view shown in **Figure 1**. Visually, the microphone makes a good impression: no plastic - all metal.

I immediately connected it to my PC, started the freeware Audacity [2] and made a recording for comparison with the microphone of my webcam (Logitech C525). I was not disappointed. I could clearly hear the sound improvement and, so clear was it, that further measurement was not deemed necessary. Not bad!

I was curious if my Skype conversation partners would also notice a difference. And, yes, the difference was noticed here, too. Two test subjects could hear quite clearly which of the two microphones was active when I switched. The difference? The CAD Audio microphone was not quite as bass-heavy and had a much clearer treble.

Levels and implications

The story could have ended here. However, during the recordings I noticed that the level of the CAD Audio microphone was more than 5 dB lower than that of the webcam, so the microphone was quieter. The video conferencing programs offer a level increase, but then the slider would have had to be 'turned up to 11.'

It wasn't the first time I had encountered low levels in microphones. But, being an electronics engineer we know how to fix things, which is why I have already retrofitted amplifiers in microphones or increased the gain of existing electronics by changing components. This should also be possible with this microphone, right?

With the though barely formed in my mind, the microphone was already disassembled as **Figure 2** proves. Above, you can see the

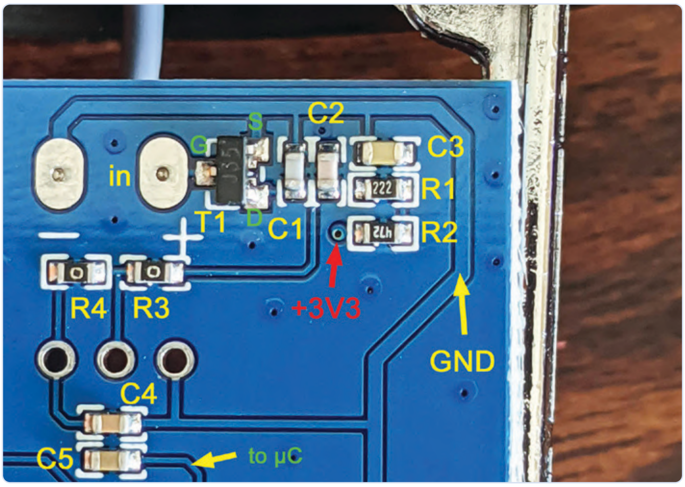


Figure 4: Annotated detail view of the circuit of the impedance converter.

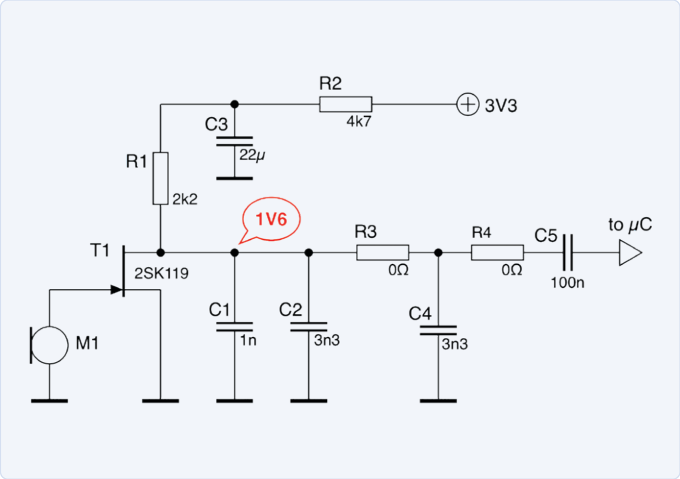


Figure 5: The circuit of the impedance converter.

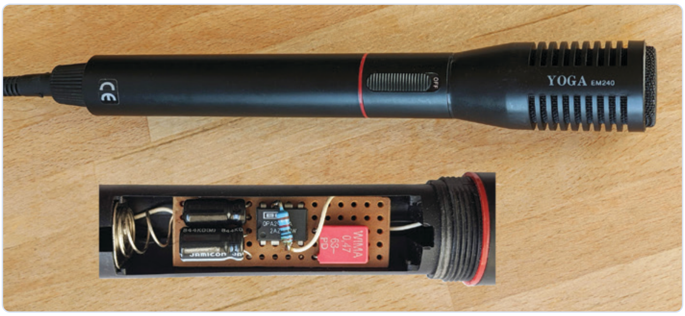


Figure 6: The cheap Yoga EM240 microphone with built-in preamplifier.

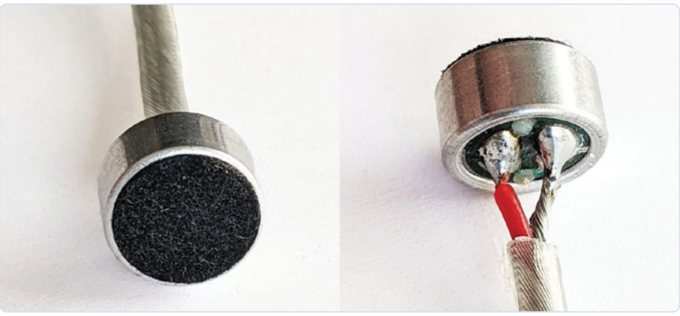


Figure 7: A small electret capsule with a 5 mm membrane.

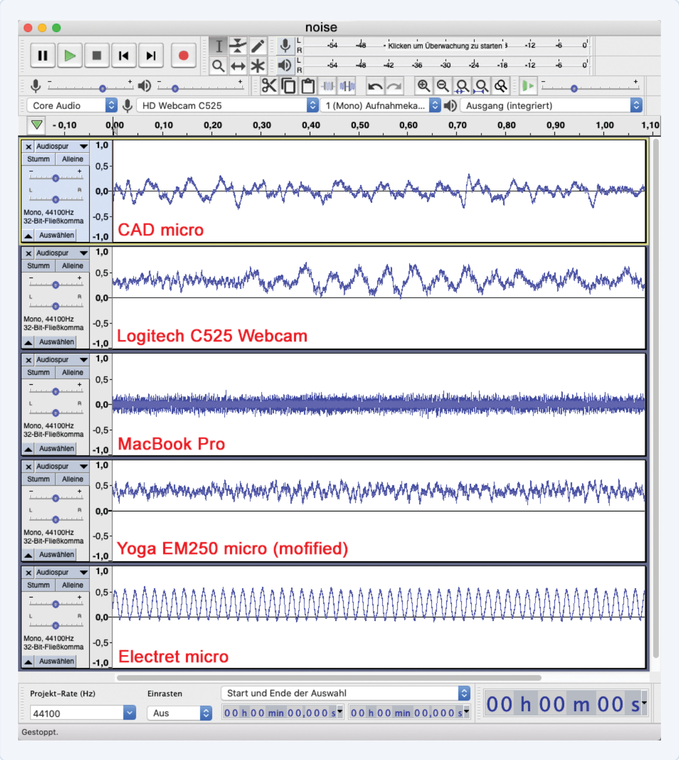


Figure 8: Different noise levels of the different microphones.

actual microphone capsule whose diaphragm, with a diameter of about 1/2" , is a bit small for my understanding of the term 'large diaphragm'. **Figure 3** shows the capsule, which is also too small, also from behind. Of more interest is the circuit board. The chip in the middle is an SoC with audio input and USB interface. In contrast to the serial EEPROM on the bottom right, the SoC is probably not labelled due to secrecy - a pity! The little three-legged black SMD device at the top is the actual preamplifier.

A first measurement on the two pads "-" and "+" in the upper middle showed that exactly 0.0 V is applied to the microphone membrane. So, no bias voltage. Thus it must be an electret condenser microphone. I was a bit disappointed, despite the good sound. The SMD device turned out to be a 'J35', an N-channel JFET of type 2SK1109 with a drain current of about 200 µA with gate shorted.

Impedance converter

Figure 4 shows a section of the circuit board around this JFET. An impedance conversion is necessary because the capacitance of the microphone membrane is very low and its impedance for audio signals is very high. Connected to the source input, T1 provides amplification - the foundation for a higher output level. In **Figure 5** you can see the circuit of the components from **Figure 4**. R2 is used together with C3 to filter the supply voltage. R1 and R2 together drop about 1.7 V; the drain current is therefore about 245 µA.

What is noticeable is that you can simply swap R1 and R2 without endangering the DC current setting and the gain doubles (≈ +6 dB). This doesn't cause any issues since the low-pass filter formed by the 2.2 kΩ resistor and 22 µF capacitor lies at a respectable 3.2 Hz. However, the low-pass formed by R1 = 4.7 kΩ and C1 + C2 + C4 then lies under 5 kHz. This can be changed by removing C2 or C4.

Results

After modification the level was now slightly higher than my webcam and the sound was unchanged. So, the modification went well.

Now I was interested in how the modified CAD microphone sounded compared to other microphones. So, I made recordings of it, my webcam, my MacBook Pro, as well as of an old cheap microphone (**Figure 6**) and a cheap, naked electret microphone capsule (**Figure 7**). Not surprisingly, the MacBook pro (subjectively) sounded the worst. The lack of bass turned my sonorous voice into a rather tinny one. Years ago I built a low-noise preamp into a 'Yoga' microphone because, in its original state, it was too noisy despite its low output level.

To provide you with an impression of the results, I have generated short MP3 files which can be downloaded from the Elektor website for this article [3]. Each file contains a short sentence together with about one second of noise. These are all set to the same level so only the sound differs. **Figure 8** shows the five different noise levels, each amplified by 30-40 dB. Only the microphones from CAD Audio, Logitech and Yoga show 'real' noise. With the Logitech you can also hear a superimposed buzzing, with the MacBook Pro high-frequency noise, and the electret capsule a mains hum.

Conclusion

Am I satisfied with the sound of the CAD Audio microphone? My answer could be from Radio Yerevan [4]: In principle yes, but...

On the one hand, I was sold a small membrane microphone as a large one. On the other, it was cheap and actually better than the webcam's microphone, and that's something.

By the way, such a microphone is not spoken into from the front but, because of its directional characteristic, from the side as shown in **Figure 9**.

200434-02

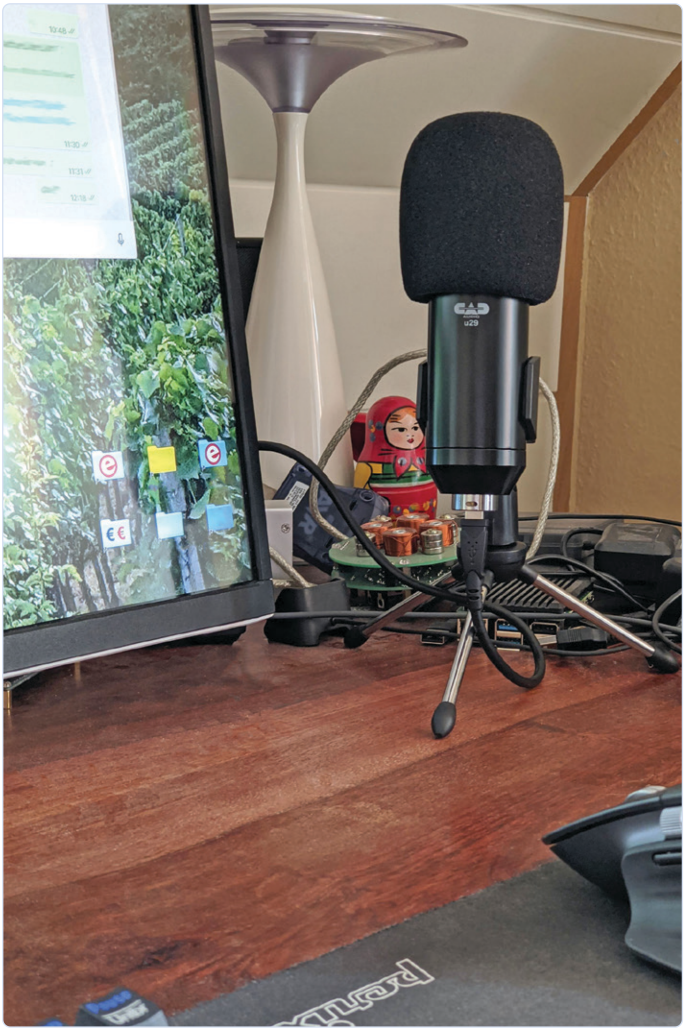


Figure 9: This is how the CAD Audio microphone should be positioned: to the right (or left) of the monitor.

Questions or Comments?

Do you have questions or comments about this article? Then email Elektor at editor@elektor.com.

Related Products

- **Book: Robert Sontheimer, *Audioschaltungstechnik* (German, PDF)**
  www.elektor.de/18458
- **Simple Preamp for Pro Microphones (Bare) (SKU 18096)**
  www.elektor.com/18096
  www.elektormagazine.com/magazine/elektor-201705/40325

WEB LINKS

[1] Wikipedia: Mikrofonierung: https://de.wikipedia.org/wiki/Mikrofonierung

[2] Audacity: www.audacityteam.org

[3] Audio recording files: www.elektormagazine.com/200434-02

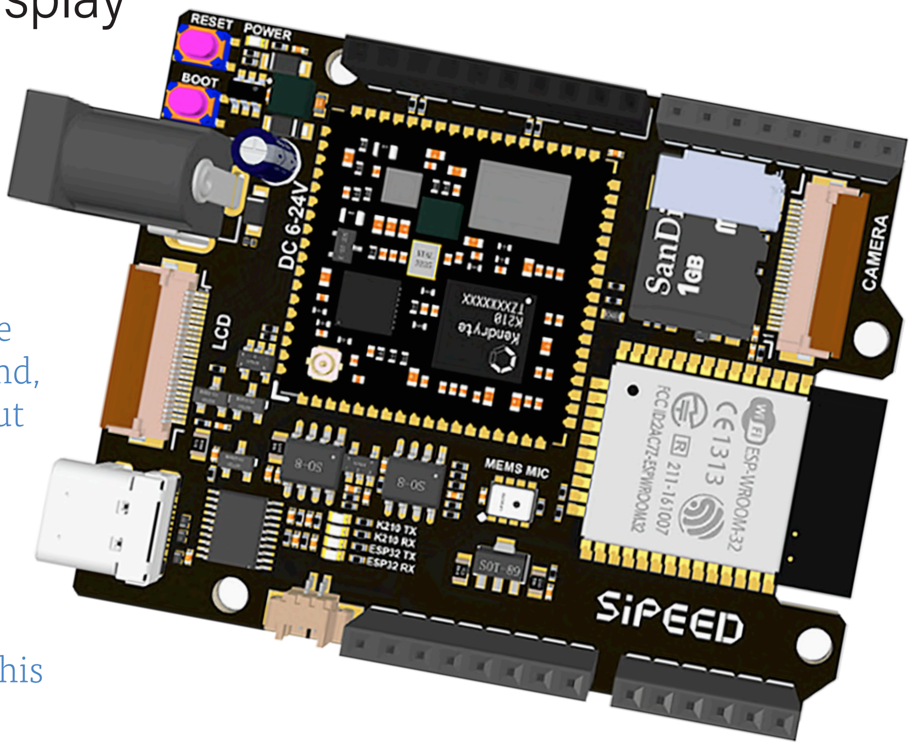
[4] Radio Yerevan Jokes: https://en.wikipedia.org/wiki/Radio_Yerevan_jokes

FFT with a Maixduino

Frequency Spectrum Display

By Tam Hanna (Slovakia)

Fast Fourier transforms not only allow you to perform tasks in the field of audio. They allow anyone to analyze the noise of a turbine or the voltage levels of vehicle electronics and, in many cases, draw conclusions about rotational speeds and the like. The Maixduino board with its K210-SoC, available from Elektor, is not only suitable for experimentation with neural networks but can also be used for fast, real-time FFT, as we show in this article.



The continuous improvements in both software and hardware have led to the fact that technologies that were once only available to governments are now also in reach of standard developers.

Semiconductor company Sipeed, until now known primarily for RISC-V processor experiments, is now offering a new development board with their Maixduino. In addition to an ESP32, which handles network communication, the board also includes an SoC known as the *K210 Kendryte*. This is intended by the manufacturer for use in AI applications. The mention of the name Kendryte is no surreptitious advertisement by Elektor. Rather, we mention it because a lot of information on this device can only be found in Google by using the search term "kendryte [your search term]."

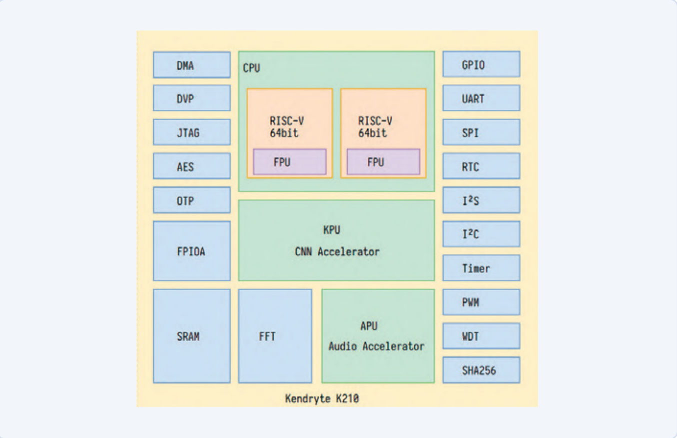


Figure 1: The K210 incorporates some powerful coprocessors. (Source: Canaan Inc.)

If you want to know more about the SoC used you should take a look at the, sometimes sluggish, website [1]. The K210 data sheet mainly contains a description of the pins, but the architecture diagram in **Figure 1** is also interesting.

The main processor (CPU) is based on the open source RISC-V architecture (with options I, M, A, and C). These cores are flanked by two helper processors. First is the KPU, behind which is a Neural Network Processor (without further information in the data sheet) that implements a neural network implemented in hardware. Similar to the DPO Tristar processor, once used by Danaher in the TDS 5xxD and TDS 7xxD, this is also a one-trick pony.

The other extension is the audio processor, called APU. It processes audio information at up to 192 kHz, stores this information via DMA directly in memory, and can also automatically apply FFT processes. We will use this APU here to implement a simple FFT.

For the sake of completeness, it should be noted that the device also includes the usual range of microcontroller peripherals, something that we will not dwell on here.

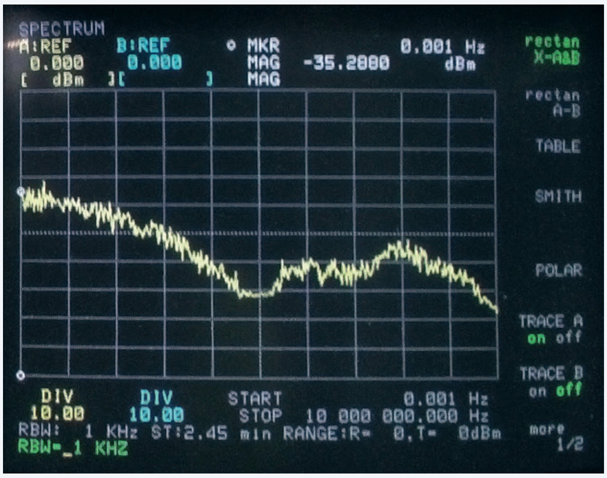


Figure 2: Narrow frequency bands produce higher resolution but require more sweep time (ST)...

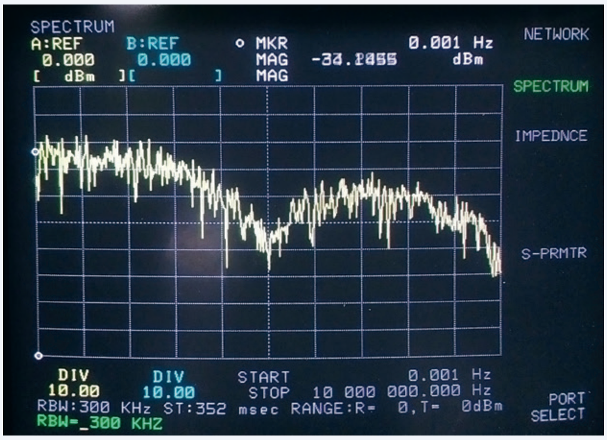


Figure 3: ...while faster sweeps and wider bands show less detail.

The Tools

Of immediate interest for us are the available development tools. At the lowest level is the standalone SDK available for download under [2] - a C library that allows us to interact directly with processor primitives.

One level above, there is ArduinoCore [3], a framework for the Arduino IDE that can also be used with Platform.IO. Last, but not least, is MaixPy. This is a programming environment derived from MicroPython that is intended for all those who have a lot of experience with Python (please read the text box first!).

To demonstrate the capabilities of the FFT, we will use some audio data. Luckily, the Maixduino comes with a MEMS microphone connected to the processor via the I2S bus. This is a nice 'low-hanging fruit' and the example application published in [4] may have been one of the first applications developed. Let's start with the integration of some of the required header files:

```
#include <Sipeed_ST7789.h>
#include “i2s.h”
#include “fft.h”
```

The next step is to consider how to construct the buffers for processing. In the datasheet mentioned in the introduction, we discover that the Kendryte cannot work with arbitrary buffer sizes. If you want to use hardware acceleration, a choice between the buffer sizes of 64, 128, 256 or 512 must be made.

Even if some mathematicians may whine about it, it is reasonable to always have a classic spectrum analyzer in the back of your mind (or better on the lab bench) when working with the FFT. In our case, the bin size is relevant as it describes the frequency resolution of the window.

Figure 2 and **Figure 3** are taken from the author's HP4195A and illustrate the procedure in principle. While our analog spectrum analyzer shows 'unsteady' bands due to noise in both cases, the actual curve is more even with the FFT. The frequency resolution is given by the formula Sampling Rate/FFT Size; if 1024 samples are taken at 8192 Hz, the resolution bandwidth (RBW) is 8 Hz.

As is often the case with the FFT, maximization is not everything. In principle, the correlation between the number of frequency bands and the number of incoming samples is a factor of 2; however, as a small complication, FFT algorithms quickly become very complex as the number of samples increases. This leads to a growth in complexity, such as that shown in **Figure 4**, where the accuracy of our FFT operation is limited by the realities of real-world implementation constraints.

In the next step, we define a group of constants. It is recommended to define the FFT length using a `#define` statement, allowing us later to make quick adjustments between accuracy and speed:

```
uint32_t rx_buf[1024];
#define FFT_N                512U
#define FFT_FORWARD_SHIFT    0x0U
#define SAMPLE_RATE          38640
#define FRAME_LEN            512
```

The storage of the data as used by the DMA requires the definition of a group of memory locations:

```
int16_t   real[FFT_N];
int16_t   imag[FFT_N];
int       hard_power[FFT_N];
uint64_t  fft_out_data[FFT_N / 2];
uint64_t  buffer_input[FFT_N];
uint64_t  buffer_output[FFT_N];
fft_data_t *output_data;
fft_data_t *input_data;
complex_hard_t data_hard[FFT_N] = {0};
```

Initializing the Hardware

As mentioned, the manufacturer of the Maixduino has provided an MSM261S4030H0 MEMS microphone. To understand how to use such small microphones, we refer to **Figure 5**. The pin assignment clearly shows that the module undertakes all of the signal conditioning and amplification itself, delivering the resultant digital audio data via a serial bus.

The first challenge is that the software does not automatically initialize the data lines required for communication with the microphone. This can easily be remedied at the beginning of our setup function:

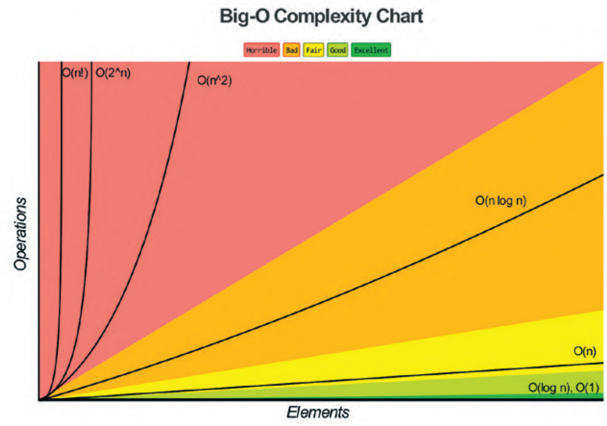
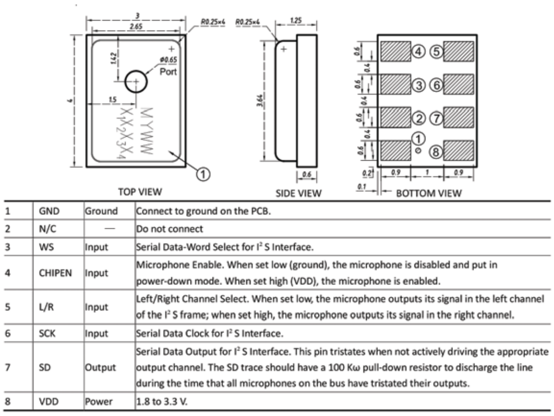


Figure 4: Ever smaller frequency bands require lots of processing power. (Image: https://www.bigocheatsheet.com/)



1	GND	Ground	Connect to ground on the PCB.
2	N/C	—	Do not connect
3	WS	Input	Serial Data-Word Select for I^2 S Interface.
4	CHIPEN	Input	Microphone Enable. When set low (ground), the microphone is disabled and put in power-down mode. When set high (VDD), the microphone is enabled.
5	L/R	Input	Left/Right Channel Select. When set low, the microphone outputs its signal in the left channel of the I^2 S frame; when set high, the microphone outputs its signal in the right channel.
6	SCK	Input	Serial Data Clock for I^2 S Interface.
7	SD	Output	Serial Data Output for I^2 S Interface. This pin tristates when not actively driving the appropriate output channel. The SD trace should have a 100 Kω pull-down resistor to discharge the line during the time that all microphones on the bus have tristated their outputs.
8	VDD	Power	1.8 to 3.3 V.

Figure 5: MEMS microphones and their digital outputs save the user a lot of work. (Image: MEMSensing Microsystems)

```
void setup()
{
    lcd.begin(15000000, COLOR_RED);
    fpioa_set_function(20, FUNC_I2S0_IN_D0);
    fpioa_set_function(18, FUNC_I2S0_SCLK);
    fpioa_set_function(19, FUNC_I2S0_WS);
```

The next step is to configure the I2S hardware. The code is mostly predetermined by the board's hardware implementation, but an important setting is the sample rate that is configured using the `i2s_set_sample_rate()` function. Since we have comparatively few options given the size of the FFT, the sample rate is one of the few means of controlling the frequency bandwidth:

```
   i2s_init(I2S_DEVICE_0, I2S_RECEIVER, 0x3);
   i2s_set_sample_rate(I2S_DEVICE_0, SAMPLE_RATE );
   i2s_rx_channel_config(I2S_DEVICE_0, I2S_CHANNEL_0,
                   RESOLUTION_16_BIT, SCLK_CYCLES_32,
                   TRIGGER_LEVEL_4, STANDARD_MODE);
```

All that remains are a few initializations:

```
    dmac_init();
    sysctl_enable_irq();
}
```

Our next task is to carry out the actual FFT operation. In the first step, we ensure that the data received via I2S is in memory:

```
void loop()
{
  int i, sampleID;

  i2s_receive_data_dma(I2S_DEVICE_0, &rx_buf[0],
   FRAME_LEN * 2, DMAC_CHANNEL3);
  dmac_wait_idle(DMAC_CHANNEL3);
```

By calling `dmac_wait_idle()`, we instruct the processor to wait until the DMA channel becomes free. This ensures that each pass through loop operates on a new and consistent set of data. If you remove (temporarily, to experiment) the call to `dmac_wait_idle()`, you will experience strange aliasing artefacts on the screen. This is probably due to the fact that, according to the datasheet, the FFT module can only be used via the DMA engine. Conflicts due to unplanned accesses could cause disaster!

When working with the FFT engines and other fixed-function units, you must always be aware that the hardware expects the information to be delivered in a specific format. In the case of `tag_fft_data` it is declared in several places in the SDK and follows this structure:

```
typedef struct tag_fft_data
{
    int16_t  I1;
    int16_t  R1;
    int16_t  I2;
    int16_t  R2;
} fft_data_t
```

At this point, observant readers will wonder why two components named R and I are expected. To explain this, we must first clarify that R stands for real and I for imaginary.

The Fourier transformation is defined in the classical area of mathematics as an operation based upon complex numbers. FFT algorithms therefore expect the supply of both real and imaginary components, at least in the generic implementations that hardware developers like to create in silicon in the interest of maximum flexibility. However, since our sound information consists exclusively of real elements, we always set the value of I to zero at this point:

```
      for ( i = 0; i < FFT_N / 2; ++i)
      {
          input_data = (fft_data_t *)&buffer_input[i];
          input_data->R1 = rx_buf[2*i];
          input_data->I1 = 0;
          input_data->R2 = rx_buf[2*i+1];
          input_data->I2 = 0;
      }
```

This is where a review of the FFT process comes in handy. The classic "Numerical Recipies in C" [5], which is available online as a PDF free of charge and sometimes in print at a low price, offers, in the twelfth chapter (of the second edition), some interesting formulas for this purpose. Reward for the reader's efforts would be an FFT that processes two real functions at the same time, whereupon the developer can then separate the output information with very little effort.

In practice, the FFT is one of those areas that you can play with as an electronics engineer ad infinitum, a process that is often more effective in practice than a purely mathematical learning effort. As a next step we want to execute the FFT commands to obtain the results:

```
fft_complex_uint16_dma(DMAC_CHANNEL0, DMAC_CHANNEL1,
   FFT_FORWARD_SHIFT, FFT_DIR_FORWARD, buffer_input,
   FFT_N, buffer_output);
```

Note here the passing of the parameter `FFT_DIR_FORWARD` that informs the hardware engine that we are changing from the time domain to the frequency domain. In practice, there are also situations in which you want to transfer a signal that is in the frequency domain back into the time domain. To do this, the constant `FFT_DIR_BACKWARD` would be passed here instead. The next step is to collect the data as follows:

```
    for ( i = 0; i < FFT_N / 2; i++)
    {
        output_data = (fft_data_t*)&buffer_output[i];
            data_hard[2 * i].imag = output_data->I1 ;
            data_hard[2 * i].real = output_data->R1 ;
        data_hard[2 * i + 1].imag = output_data->I2 ;
        data_hard[2 * i + 1].real = output_data->R2 ;
    }
```

Note that the results returned have both real and imaginary parts. To understand this problem, let's go back to mathematics again, albeit briefly. The formula:

$S_f(t) = a_0/2 \; \Sigma \; [a_n \cos(n\omega t) + b_n \sin(n\omega t)]$

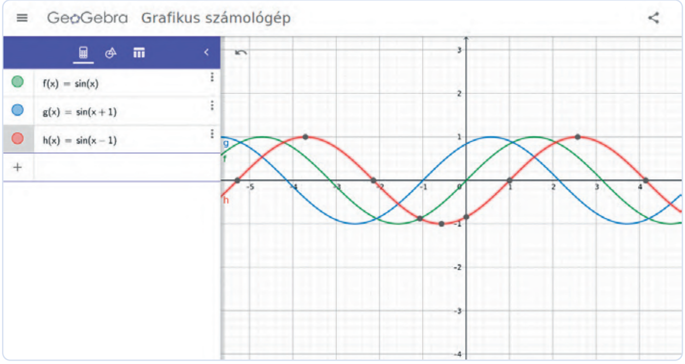
shows one of many representations of the basic Fourier series. The use of I in our formula shows us that we are working with complex units, but since the FFT is basically only a numerical representation of this classical Fourier transform, it must follow the same rules.

Why Complex?

The sampling operation we have just performed delivers only real (R) data, but this does not make the mathematician happy. One reason for this is that there is no 'defined' trigger or starting point for the signal. Although the sine wave signals as shown in **Figure 6** have the same frequency, from a mathematician's point of view they have different phases. Our FFT operation expresses this in the returned complex number values that express both the power and the phase shift of the respective components. As a result, the three sine waves signals shown would deliver different R and I values although they all have the same frequency.

The electronics engineer is usually not interested in the complex components, so we eliminate them by calculating the absolute values:

```
float pmax=10;
for (i = 0; i < FFT_N; i++)
{
    hard_power[i] = sqrt(data_hard[i].real * data_
    hard[i].real + data_hard[i].imag * data_hard[i].
    imag);
```



f(x) = sin(x)
g(x) = sin(x + 1)
h(x) = sin(x − 1)

Figure 6: Despite having identical frequency and amplitude, these waveforms are not identical.

Logarithms and square roots invite simplification in order that the calculations can be mathematically optimized. Simpler, however, doesn't mean that the procedures would be easier to understand — quite the opposite, in fact. Therefore, in the interest of improved clarity, the simplification will be omitted here.

Whoever performs the above mentioned combination of two different functions in one FFT operation would have to perform an extraction at this point and separate the two results (using an algorithm that is simple compared to the FFT). We will stick with our version and, in the next step, use the logarithm function to obtain dB-like power:

```
      hard_power[i] = 20*log10(2*hard_power[i]/FFT_N);
      if( hard_power[i]>pmax && i>5)
          pmax = hard_power[i];
 }
```

The next task is to show the results on the display. A challenge here is that the Arduino's display API suffers poor performance when it is passed 'invalid' coordinate values:

```
    lcd.setRotation(0);
    lcd.fillScreen(COLOR_WHITE);
    for (int i=0;i<256;i++)
    {
      int dval = 240 - 240*(hard_power[i]/pmax);
      if (dval<2)dval=1;
      if (dval>238)dval=238;
      lcd.drawLine( i,  240, i, dval  , COLOR_RED);
    }
    delay(50);
}
```

We are now ready for a first test run that will produce the result shown in **Figure 7**. You can try using test tones with the system at this point. Changing the test tone frequency will shift the lines that represent it on the screen.

The FFT is Mirrored

The FFT is one of those subjects that you can spend many hours investigating and gaining new insights. As the test tone frequency changes in our test setup, the peaks on the left and right of the display move towards and away from each other, mirrored around the centre.

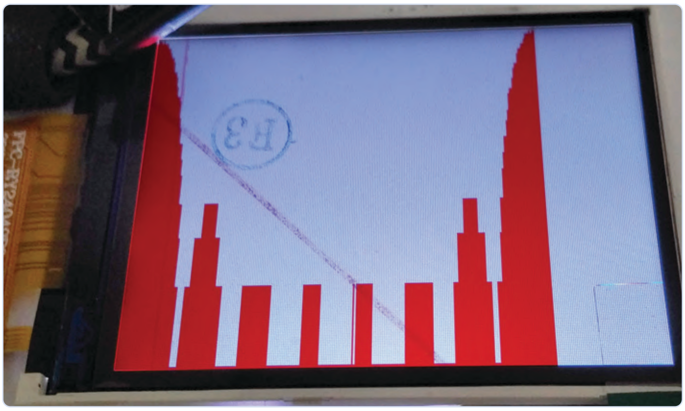


Figure 7: The ‘symmetry’ displayed is confusing.

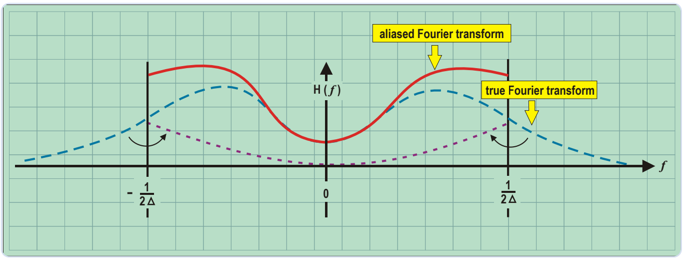


Figure 8: Spectral elements eliminated by a Nyquist window appear as artefacts.

Our FFT repeats itself already in the 128th field that, according to the theorems just stated, should only happen in the 256th field. The source of this problem is the input samples supplied via the I2S bus – a topic that we will not delve into too deeply here. A modified version [6] mitigates the problem to some extent.

The emphasis here is on the term ‘mitigates’ because, in the case of FFTs, a reflection always occurs with purely real input values. The values from zero to N/2 correspond to those from N/2 + 1 to N. The reason for this is that the complex parts of the FFT must cancel each other out.

Further fun with the FFT can be had with, for example, the introduction of a window function to limit the bandwidth on the input signal. The purpose for trying this is to suppress the signal that falls outside the Nyquist window that is currently being mirrored inward on both sides. A recommended document for this is the document “Some Windows with Very Good Sidelobe Behavior” [7] published in 1981 by Albert H. Nuttall.

Conclusion

The hardware FFT engine enables simple experimentation with FFT algorithms and applications. This is a capability that is needed time and again in both the civil and military sectors. Should you purchase such a board and play around with it a little, you will certainly not regret the time invested, nor the financial outlay. ◀

200297-02

SEVERAL TYPES OF PYTHON

In the field of Python for microcontrollers, two concepts are fighting for dominance. One concept is MicroPython that implements a fully-fledged runtime on the MCU. The target system contains a complete Python interpreter that even loads libraries from the Internet, analogous to a PC development platform. The advantage of this approach is a higher flexibility, while the disadvantage is that problems can occur due to the limited available resources (RAM).

The other approach is currently only represented by Zerynth and is not available for Maixduino. The PC development platform is used to create a monolith (single block of code) that is burned into the target system. The reduced flexibility is balanced by a higher resulting stability - if the libraries are located on a workstation, resource-intensive processes can be handled away from the microcontroller.

WEB LINKS

[1] Kendryte K210 Datasheet: https://bit.ly/3fptbHu
[2] Standalone SDK: https://github.com/kendryte/kendryte-standalone-sdk
[3] ArduinoCore-k210: https://github.com/Seeed-Studio/ArduinoCore-k210
[4] fft_lcd Project: https://github.com/MrJBSwe/fft_lcd/blob/master/main.c
[5] Teukolsky, Press, Vetterling und Flannery: Numerical Recipes in C: www.numerical.recipes/
[6] stft_lcd Project: https://github.com/jpiat/stft_lcd
[7] Albert H. Nuttall: Some Windows with Very Good Sidelobe Behavior: https://bit.ly/3fTkCWb

LISTING 1: THE COMPLETE MAIXDUINO LISTING.

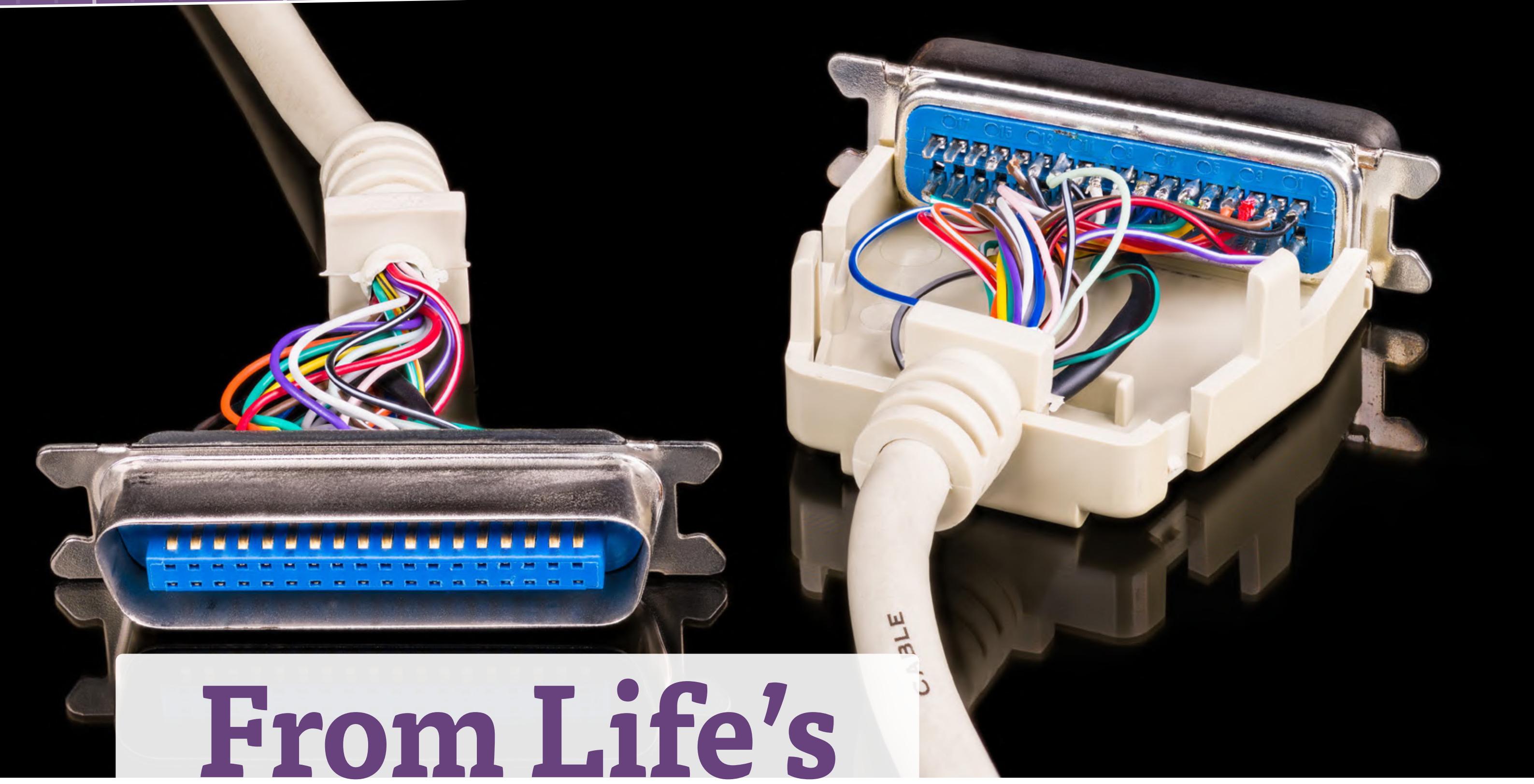
```
#include <Sipeed_ST7789.h>
#include "i2s.h"
#include "fft.h"
SPIClass spi_(SPI0); // MUST be SPI0 for Maix series on board LCD
Sipeed_ST7789 lcd(320, 240, spi_);
uint32_t rx_buf[1024];
#define FFT_N               512U
#define FFT_FORWARD_SHIFT   0x0U
#define SAMPLE_RATE         38640
int16_t  real[FFT_N];
int16_t  imag[FFT_N];
int    hard_power[FFT_N];
uint64_t fft_out_data[FFT_N / 2];
uint64_t buffer_input[FFT_N];
uint64_t buffer_output[FFT_N];
fft_data_t *output_data;
fft_data_t *input_data;
complex_hard_t data_hard[FFT_N] = ;
#define FRAME_LEN           512

void setup()
 {
     lcd.begin(15000000, COLOR_RED);
```

continued overleaf

elektor e-zine
Your dose of electronics
elektor e-zine
Your dose of electronics
December 2, 2022 | #515
LiDAR Precision Gauge: Measure Up to 12 m
LiDARs are great devices for detecting obstacles and measuring distances. Here we use one for taking precise distance readings up to 12 meters with a resolution of 1 cm. Let's take a look.
more...
elektor e-zine
Your dose of electronics
December 2, 2022 | #515
LiDAR Precision Gauge: Measure Up to 12 m
elektor
design > share > earn

```
     fpioa_set_function(20, FUNC_I2S0_IN_D0);
     fpioa_set_function(18, FUNC_I2S0_SCLK);
     fpioa_set_function(19, FUNC_I2S0_WS);

    i2s_init(I2S_DEVICE_0, I2S_RECEIVER, 0x3);
    i2s_set_sample_rate(I2S_DEVICE_0, SAMPLE_RATE );
    i2s_rx_channel_config(I2S_DEVICE_0, I2S_CHANNEL_0,
                           RESOLUTION_16_BIT, SCLK_CYCLES_32,
                           TRIGGER_LEVEL_4, STANDARD_MODE);

     dmac_init();
     sysctl_enable_irq();
}
void loop()
 {
   int i, sampleID;

   i2s_receive_data_dma(I2S_DEVICE_0, &rx_buf[0], FRAME_LEN * 2, DMAC_CHANNEL3);
    dmac_wait_idle(DMAC_CHANNEL3);//wait to finish recv

     for ( i = 0; i < FFT_N / 2; ++i)
     {
         input_data = (fft_data_t *)&buffer_input[i];
         input_data->R1 = rx_buf[2*i];   // data_hard[2 * i].real;
         input_data->I1 = 0;             // data_hard[2 * i].imag;
         input_data->R2 = rx_buf[2*i+1]; // data_hard[2 * i + 1].real;
         input_data->I2 = 0;             // data_hard[2 * i + 1].imag;
     }

    fft_complex_uint16_dma(DMAC_CHANNEL0, DMAC_CHANNEL1, FFT_FORWARD_SHIFT, FFT_DIR_FORWARD,
                                                            buffer_input, FFT_N, buffer_output);
    for ( i = 0; i < FFT_N / 2; i++)
     {
         output_data = (fft_data_t*)&buffer_output[i];
         data_hard[2 * i].imag = output_data->I1 ;
         data_hard[2 * i].real = output_data->R1 ;
         data_hard[2 * i + 1].imag = output_data->I2 ;
         data_hard[2 * i + 1].real = output_data->R2 ;
     }
    float pmax=10;
     for (i = 0; i < FFT_N; i++)
     {
         hard_power[i] = sqrt(data_hard[i].real * data_hard[i].real +
                                              data_hard[i].imag * data_hard[i].imag);
        //Convert to dBFS
         hard_power[i] = 20*log10(2*hard_power[i]/FFT_N);
        if( hard_power[i]>pmax && i>5)
             pmax = hard_power[i];
     }

     lcd.setRotation(0);
     lcd.fillScreen(COLOR_WHITE);
     for (int i=0;i<256;i++)
     {
       int dval = 240 - 240*(hard_power[i]/pmax);
       if (dval<2)dval=1;
       if (dval>238)dval=238;
       lcd.drawLine( i,  240, i, dval  , COLOR_RED);
     }
     delay(50);
}
```

UNITED



Amphenol 36-way connector, selected in 1970 for printers. Source: Shutterstock / KPixMining

From Life's Experience

Design Logic (or Non-Logic)

By Ilse Joostens (Belgium)

When I was a teenager, I built a whole lot of electronic circuits like a chicken with its head cut off. I didn't care much about the details of how a circuit worked; getting a result as quickly as possible was the main thing that drove me. Properly finishing a project also often left something to be desired, since I was perfectly happy if what I built more or less worked.

In the course of time I moved on to designing circuits myself, initially just as a hobby but in the end as a business. Particularly in the latter case, I had to get away from the idea of always designing the best possible circuit. Financial considerations, the availability of components in the required quantities, and what you currently have on hand are also significant factors for commercial projects, and as a result, you sometimes make design decisions that may not appear logical at first sight.

The Centronics Fiasco

At a certain point in time, I needed a large number of bright RGB LEDs for one of my earlier projects, and I ordered them at low cost from a German supplier. The LEDs were perfectly okay, and the supplier was even so kind as to supply a heap of resistors together with the LEDs, with the idea that they could be used as series resistors. I actually didn't need these resistors; most of them had a resistance of 510 Ω, which was not so common. So, I wanted to get rid of them, and in the end I simply used them in the design of my VFD-tube shield for Arduino (R1 in the schematic diagram) [1]. The value of that resistor is not especially important, but I quickly became a victim of my own success. The kits sold like hotcakes, and before I knew I had run out of my stock of 510 Ω resistors and had to order more. I needed to find a more continuous source for these resistors, but that wasn't all — at the time they were a lot more expensive than, for example, 470 Ω of 560 Ω resistors. Talk about shooting yourself in the foot! The kit was so enormously popular that modified versions started appearing online, including a design with six tubes. It turned out that everyone had copied the 510 Ω resistor willy-nilly.

Guglielmo Marconi. Source: Shutterstock / Morphart Creation.

Now I am ruefully reminded of the situation with the Centronics interface [2]. In 1970 Centronics developed a low-cost dot matrix printer (Model 101) [3]. By chance, their US parent company at the time, Wang Laboratories, had a surplus stock of 20,000 36-way Amphenol flat cable connectors on hand, originally intended for one of their early calculators. The decision was quickly made to use these connectors for the parallel interface of the new printer. That connector — in my opinion rather clunky — soon became an industry standard, and who knows how many more Centronics had to order afterwards.

Cost-Effectiveness and 'Muntzing'

I while ago, I ran into criticism of my design for the VFD-tube clock with an ESP32 [4] on a German online forum. It all started with someone who did something wrong while building the project and then tried to blame me as the designer. Then another forum contributor remarked that using a 47 µH inductor in the boost converter resulted in large peak currents and that the whole thing was not very efficient considering that the gate of the MOSFET was driven directly by the output of the 7555. This criticism was not entirely groundless, but as it happens, I am more or less drowning in 47 µH inductors because of another project, so I am inclined to use them wherever possible. In total, the clock consumes around 2.25 W, of which the boost converter accounts for a bit more than 550 mW. For that amount of power I don't really worry about efficiency, and I certainly don't intend to develop an expensive state-of-the-art converter. Where efficiency is really so important, I use an LC display instead of VFD tubes. Unfortunately, an LCD doesn't have the same aura. I may sometimes indulge in a bit of overengineering, but the financial bottom line has to be right. I don't want to go as far as the flamboyant businessman Earl 'Madman' Muntz [5][6], who reportedly always carried a pair of side cutters in his breast pocket so that he could cut 'unnecessary' components out of the designs of his engineers to keep the cost down. To be accused of greed despite everything, with a snide reference to 'greedy investment bankers' on the same forum because my kits are claimed to be a bit pricey, is pretty low. Maybe those people have never heard of taxes or labor costs.

Intuition Versus Reason

What mattered most at school was calculation, argumentation, and — where necessary — proof with the emphasis on theory, along with a certain disdain for practical matters, and above all for empirical methods. However, your boss will certainly not appreciate it if you spend hours on calculations for a simple circuit, and for me as a self-employed person the saying 'time is money' is all too true. Of course, I sometimes resort to calculations, but I'm not afraid of designing based on intuition, especially for relatively simple things. Some circuits simply need an empirical approach, especially when you're working with obscure vintage parts. You should also have the courage to reuse existing designs instead of always wanting to reinvent the wheel.

For the development and refinement of the magnetic radio detector, nicknamed 'Maggie' [7], Guglielmo Marconi also took an empirical approach, based on the work of Ernest Rutherford and Harry Shoemaker. The magnetic radio detector served well until the advent of detectors using vacuum tubes, and a magnetic radio detector was also on board the Titanic. It took twenty years until scientists were able to provide an adequate explanation of how the device worked. That's a testimonial to Marconi's ability as a practical engineer, with the emphasis on 'practical.' ◀

220671-01

Disclaimer: All unnecessary text has been carefully removed from this column with the Delete button that I always carry in my handbag.

Translated to English by Kenneth Cox

WEB LINKS

[1] Ilse Joostens, "VFD Shield for Arduino," ElektorMag 9-10/2015: https://elektormagazine.com/magazine/elektor-201509/28071
[2] Parallel port: https://en.wikipedia.org/wiki/Parallel_port
[3] Manuals Library – Centronics 101 Operator's Manual: https://manualslib.com/manual/1438231/Centronics-101.html
[4] "VFD-tube Clock with ESP32," ElektorMag 5-6/2018: https://elektormagazine.com/magazine/elektor-201805/41547
[5] Bob Pease – "What's All This Muntzing Stuff, Anyhow?" : – Electronic Design (1992): https://elektor.link/MuntzingStuff
[6] Documentary – Madman Muntz: https://youtu.be/deFlB2G0mH8
[7] Magnetic detector: https://en.wikipedia.org/wiki/Magnetic_detector

UCN5804 Stepper Motor Driver

Peculiar Parts, the Series

By David Ashton (Australia)

A stepper motor driver? What's peculiar about that? There are probably hundreds of them! Well, the peculiarity of this one lies in its simplicity. It's just a single IC with Step and Direction inputs to directly drive a stepper motor – no other logic or ICs required.

The UCN5804 is a stepper motor driver IC made by Allegro Microsystems. It comes in a 16-pin (DIP or SOIC) package, with the center pins on each side (4/5 and 12/13) tied together and to ground. These provide heat sinking in addition to grounding. It's a great little IC, capable of driving four windings of a 4-phase unipolar stepper motor with only *Step* and *Direction* inputs. Its output capabilities (1.5 A output at up to 35 V) are also impressive, so it can drive most small stepper motors directly. There are also single-phase and half-step inputs for a bit more versatility. Nice and straightforward, and easy to use.

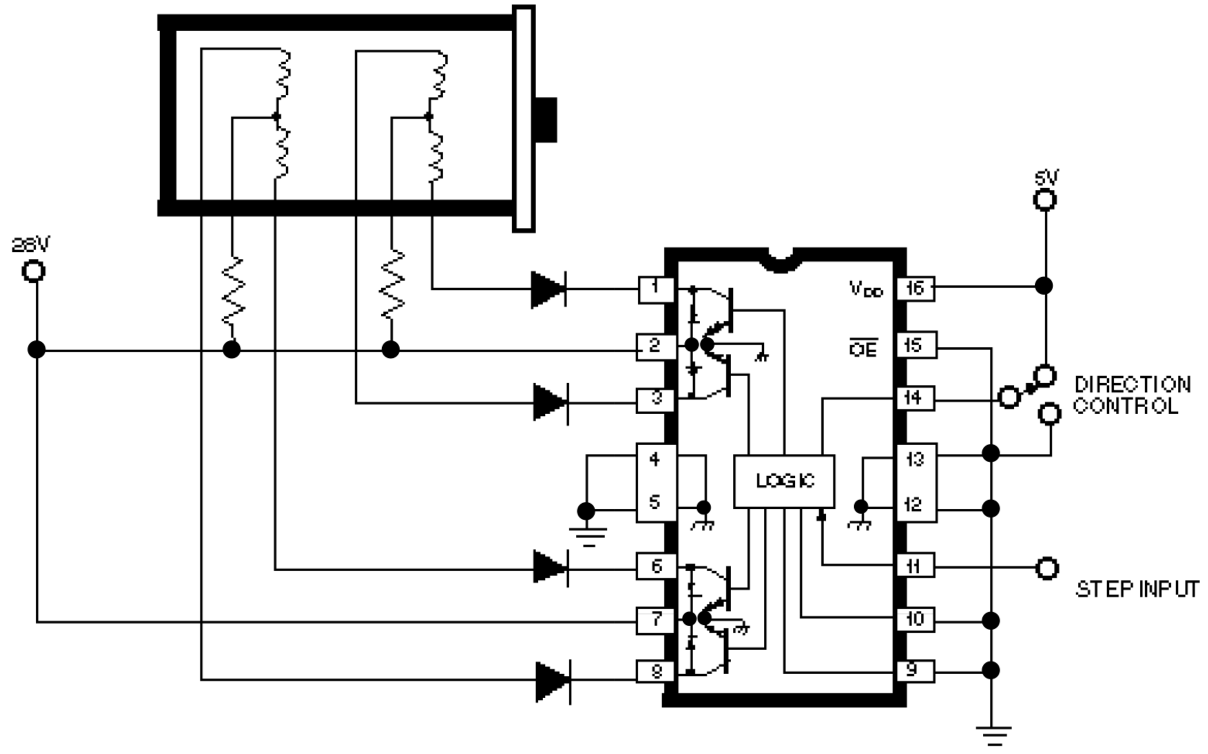


Figure 1: UCN5804 circuit. Source: Allegro Microsystems datasheet.

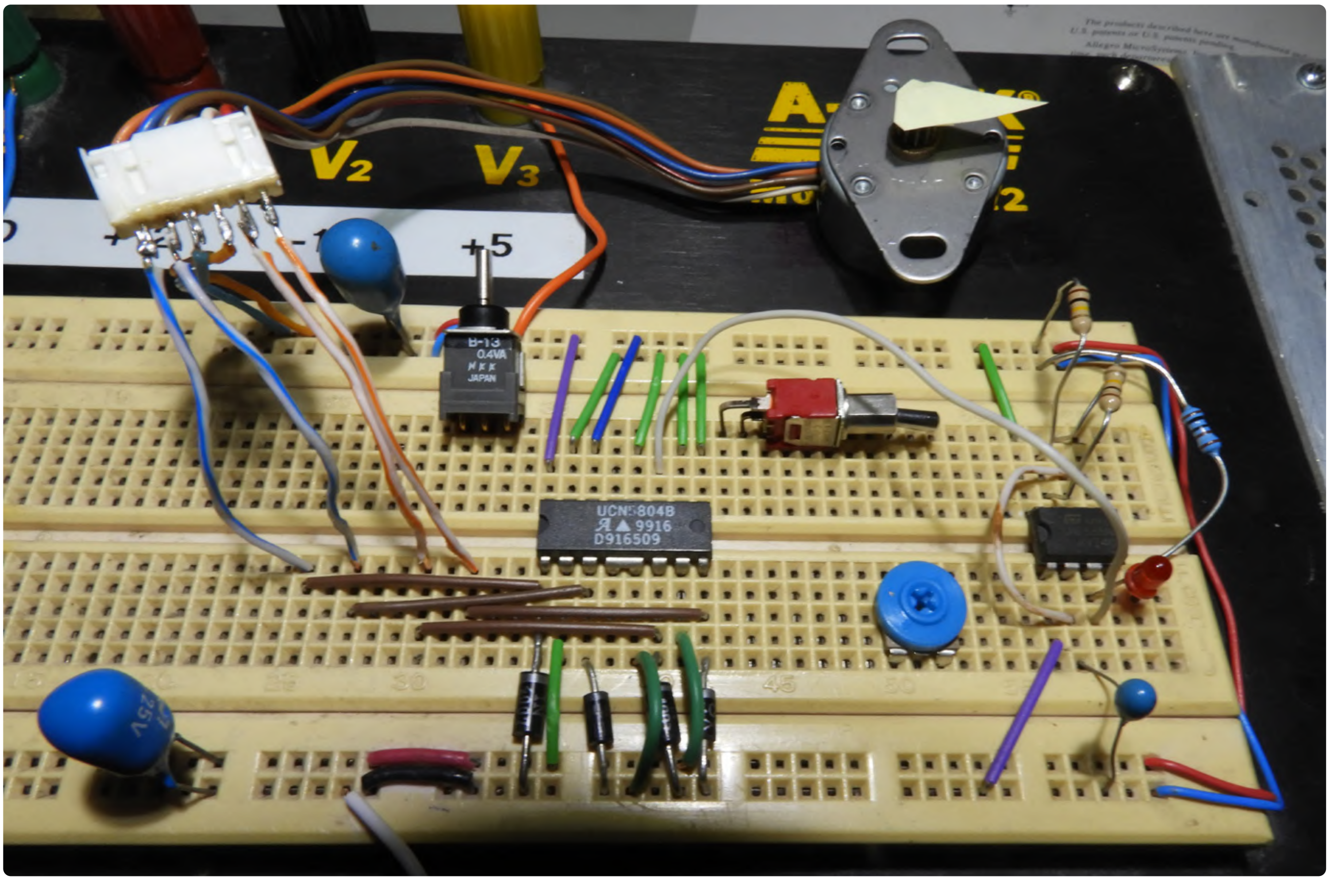


Figure 2: A breadboard circuit using a recovered UCN5804, a 555 for step timing, and some switches for control.

Unfortunately, it's been discontinued, although Allegro still provides the datasheet on its site [1], which is commendable — many companies withdraw datasheets of discontinued components.

A typical circuit is shown in **Figure 1**. As can be seen, all you need to drive a stepper motor is a handful of diodes. The direction control is implemented with a switch, but this could just as easily be a signal from a microcontroller or other logic circuit.

So, why did Allegro discontinue such a great little stepper motor driver? Probably because it's an older bipolar IC, not MOS, and has none of the power consumption reduction features that manufacturers today require for making their system electrically efficient. The discontinued notice on the datasheet recommends Allegro's A3967 and A3977, which are not the same thing at all, as these are made for driving bipolar stepper motors (which, it's true, are more common these days) with H-bridge drivers. They also have many more bells and whistles, such as current sensing, PWM drive, and half-, quarter-, and eighth-step modes. Another vital omission, especially to hobbyists like me who still tinker with stripboard, is that these two ICs are only available in 24- and 28-pin SOIC packages, respectively.

There are several other stepper motor driver solutions, but they all fall short of the UCN5804 in one respect or another. The well-known L297/298 combination is pretty versatile, but that's a two-device solution. Other devices purporting to be stepper motor drivers are merely half H-bridge circuits and without the nice and simple *Step* and *Direction* inputs. That leaves it up to you, the plucky developer, to generate the control waveforms with another IC or your microcontroller.

It's a great pity that Allegro discontinued this IC, as it is ideal for hobbyists (yes, I know, hobbyists are a tiny market). In **Figure 2**, you'll find a breadboard circuit using the UCN5804 together with a 555 timer and some switches. Luckily, I still have a few recovered from junked tape backup drives. However, if you look around, you can still pick them up on the internet for around $5 - $10 a piece.

220530-01

Questions or Comments?

Do you have technical questions or comments about this article? Email Elektor at editor@elektor.com.

WEB LINKS

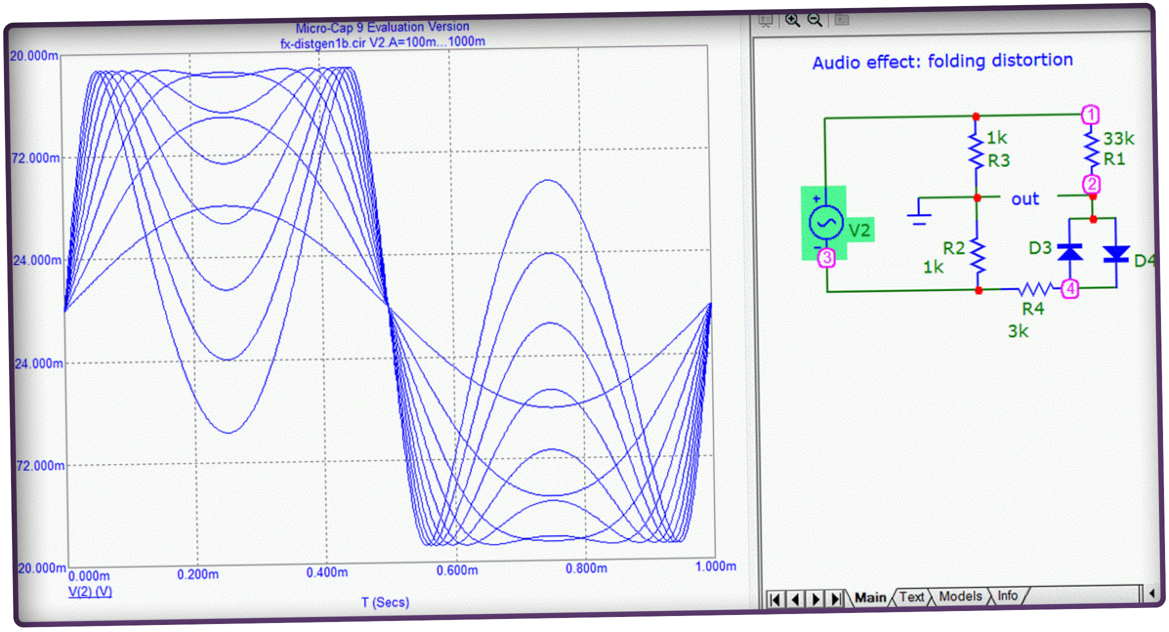
[1] Allegro Microsystems - UCN5804 datasheet: https://elektor.link/ucn5804

developer's zone

Tips & Tricks, Best Practices and Other Useful Information

Circuit Simulation With Micro-Cap

First Steps in a Complicated World



By Raymond Schouten (The Netherlands)

A circuit simulator should be part of the toolbox of every electronics enthusiast, just like a power supply, a multimeter and an oscilloscope. And like every other tool, to get the most out of a circuit simulator, you must spend some time learning how to use it. In this article, we show you how to get started using the freeware simulator, Micro-Cap.

Why Simulate?

Use of a circuit simulator can save you time on testing ideas or on defining components. It also is a good learning and debugging tool, showing you what goes on inside a circuit.

Circuit simulation software like LTspice (Analog Devices), Tina (Texas Instruments) and Micro-Cap use the text-input-based SPICE code developed at the University of California, Berkeley (1973). Modern software mainly adds a graphic interface to that. A significant part of this article is therefore valid for all three, especially when it comes to tips on avoiding simulation mistakes.

The simulator introduced in this article is Micro-Cap [1]. It is a professional-quality circuit simulator ($4,500) that was released as freeware after product development stopped. Compared to other free simulators, it features quite advanced functions. The inconvenience is that it might get outdated eventually or even disappear because it is no longer being worked on.

Micro-Cap lets you draw circuits and probe the signals in it while 'turning knobs' by stepping parameters. It comes with a large library of component models, even including vacuum tubes. Design tools for active and passive filters are built in. Nice for audio applications is that it allows you to listen to the waveform produced by your simulated circuit.

How to Simulate?

Doing a circuit simulation is like building something on a breadboard. You place the components, interconnect them to make a circuit, add supply voltages and (optionally)

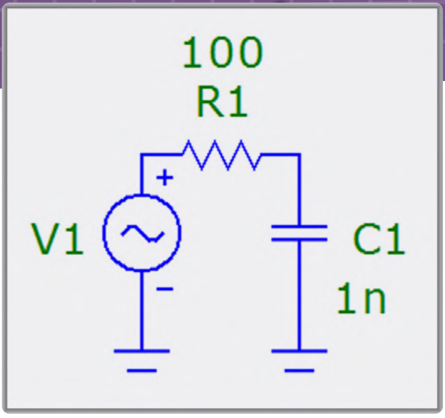


Figure 1: My first circuit, a simple RC low-pass filter.

connect a signal source. Next, you choose to either measure signals with an oscilloscope (in the time domain) or with a spectrum analyzer (in the frequency domain). More details on this can be found in the inset, **Analysis Types**.

Drawing My First Circuit

After launching Micro-Cap, we begin by creating a simple RC low-pass filter and simulate its frequency response as shown in **Figure 1**. There are several ways and shortcuts to add components to the circuit (see inset). For now, we use the component bar at the top of the screen (**Figure 2**).

Left-click briefly on the resistor symbol and move the mouse pointer into your empty worksheet (optionally, click on the little arrow or triangle right to the symbol button to select another orientation). A resistor symbol will appear. Left-click again to place the resistor at the desired location in the worksheet. In the window that now opens, enter a value of 100 in the *Value* box at the top. Close the window with the *OK* button or by pressing the *Enter* key.

Repeat this procedure for a capacitor; set its value to *1n*. If you place the capacitor in such a way that one of its wires touches one of the resistor's wires, they will be connected automatically.

Next, we add a sine wave generator or source by clicking *Component* on the main menu and then selecting *Analog Primitives Waveform Sources Sine Source*. Pick *GENERAL* from the list on the right of the source's properties window that opened, and then click *OK*.

Finish the circuit by adding the ground symbols from the component bar to the capacitor and the signal source.

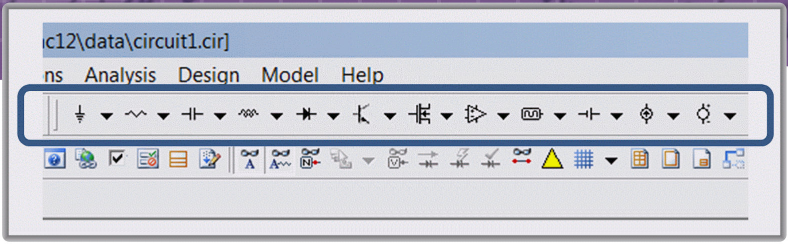


Figure 2: Micro-Cap comes with a Component Bar.

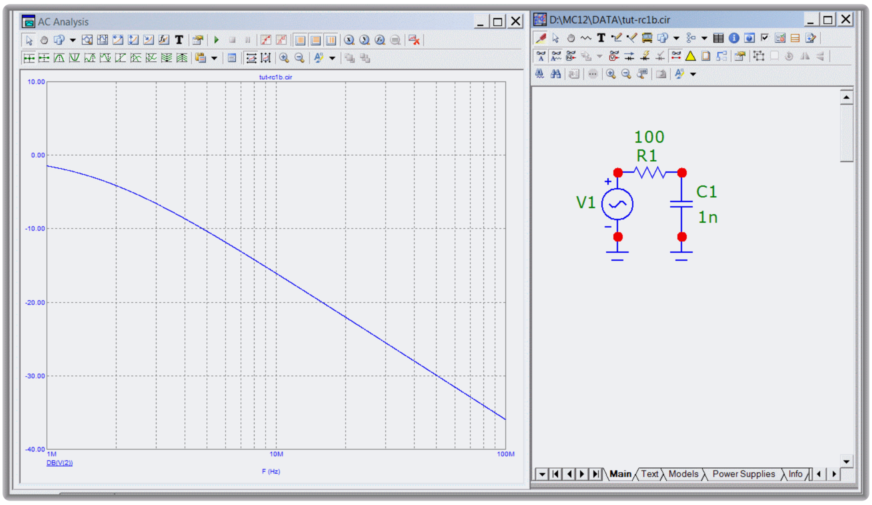


Figure 3: Clicking the RC node will display the transfer curve of the filter.

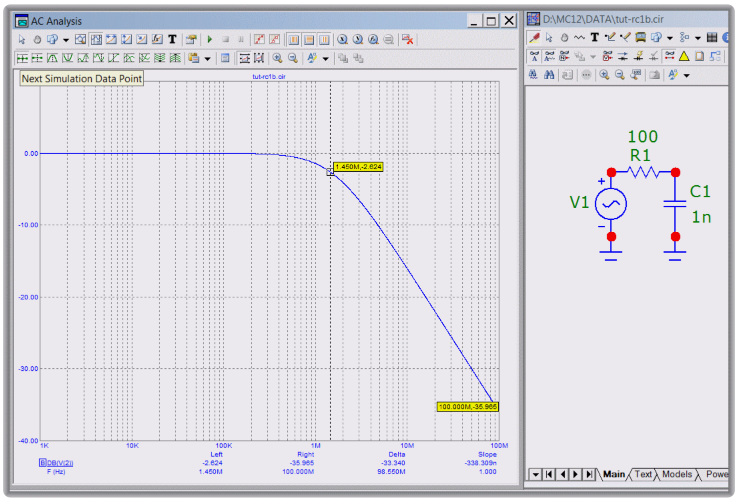


Figure 4: Adjust the graph's limits to see what you want to see.

If you placed the components away from each other, you must wire them together. The *Wire Mode* button or pressing Ctrl-W allows you to do this. To draw a wire, click the wire's starting point and, while keeping the left mouse button down, drag the pointer to the wire's destination.

Simulating It

We're now ready to run a simulation. From the *Analysis* menu, select *Probe AC*... to take you to analysis mode, ready to 'measure.' In this mode, you cannot change the circuit drawing anymore, only the component values. The red dots in the circuit are the nodes. Clicking a dot will produce a graph in the *AC Analysis* window similar to the one shown in **Figure 3**.

This is a plot with the default frequency settings, but we want a graph starting at a lower frequency. To achieve this, select *Limits*... from the *Probe* menu. Change the *Frequency Range* to '100Meg, 1k' (the format is 'high, low') and click *Close* to exit. The graph changes to the one shown in **Figure 4**. We can now see both the pass band of the filter (DC

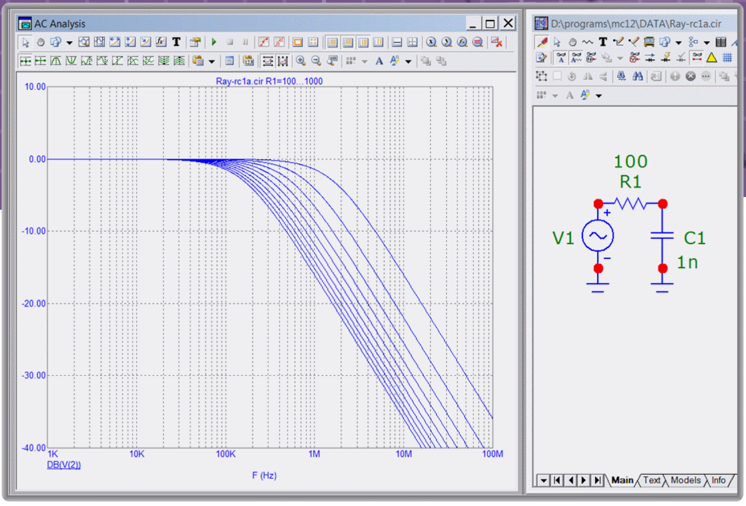


Figure 5: Stepping a parameter (here the value of R1) allows visualizing its influence on the circuit's transfer function.

How Much is 'M'?

In SPICE language M means milli (10^{-3}) so if you set a resistor value to 1 Mohm, it will be 0.001 Ω... You can use MEG for mega or 10^6.

up to 1 MHz) and its stop band (everything above 1 MHz).

You can read the data points in the graph by using a cursor. Click on the top-left button *Next Simulation Data Point* above the graph and a cursor pops up that you can move around.

Simulation Vs. Reality

According to the graphs in Figures 3 and 4, our filter damps everything above 1 MHz? This would mean that this filter suppresses all frequencies up to infinity. We know that this is not true in reality. Then why is it so in the simulation?

The reason is that a simulator uses ideal components. If you want to get closer to the real-world behavior of a circuit, you must add the imperfections of both your components and the real world by adding stray parasitic elements. Refer to the inset, **Component Model Limitations**, for details about making your simulation more like a real-world circuit.

Changing a Component Value in Analysis Mode

When in analysis mode, you can still change component values. To do so, you must switch to *Select Mode* by clicking the button with the mouse arrow icon on it at the top-left of the circuit window (or press Ctrl-E). Now, you can click on a component (value) and change it in the same ways as before. The simulation results are automatically updated. If you want to probe somewhere else in the circuit, you must revert to *Probe* mode by clicking the *Probe* button.

Stepping a Parameter

A powerful feature of a simulator is that it can step the value of one or more parameters and display the results as multiple graphs. As an example, let's step the value of resistor R1 as if it were a potentiometer to see what the tuning range is for a certain potentiometer sweep.

We will sweep the filter's cutoff frequency by stepping R1 from 100 Ω up to 1000 Ω in steps of 100 Ω. To do so, select *Stepping*... from the *Probe* menu. Select R1 in the *Step What* box. Enter values for the *From*, *To*, and *Step Value* boxes and do not forget to set *Step It* to *Yes*. Click *OK* and then click the *Run* (F2) button (with the green arrow or triangle) in the AC Analysis window to display the graph as a function of the resistor value (**Figure 5**).

In this example, the steps were of equal size (linear) but you can also step exponentially. It is possible to step multiple component values at the same time or in nested sweeps. You can also step component model values, e.g., sweep the gain of a transistor.

Transient Analysis

You need to choose the right analysis mode to see non-linear behavior (see inset **Analysis Types**). To get some non-linear behavior, let's make a simple audio distortion effect, a diode clipper as shown in **Figure 6**. We can then use transient analysis to observe distortion as if we were using an oscilloscope.

Set the frequency of the sine source, V1, to 1 kHz (in the '*F*' box in the bottom-right corner of the properties window) and choose *Probe Transient...* from the *Analysis* menu. From the *Probe* menu, choose *Limits...* and set *Maximum Run Time* to *2m* and the *Maximum Time Step* to *2u*. Good practice is to set the time step to 1/1000 of the run time range. Now probe the circuit, both at the input (showing the sine wave), and at the output (showing the clipped waveform).

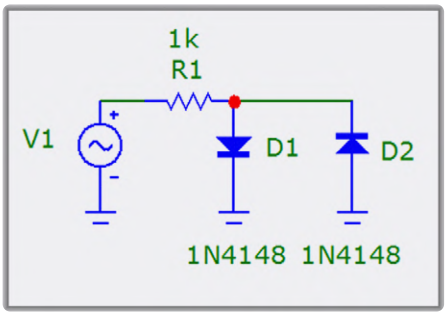


Figure 6: This simple circuit exhibits non-linear behavior.

Analysis Types

After drawing the circuit, you want to take measurements. For this, you have to select an analysis mode. In SPICE language, measuring with an oscilloscope is called 'transient analysis' while measuring with a spectrum analyzer is called 'AC analysis.' The component interconnect points are called nodes. In analysis mode, you can probe the voltages on these nodes and the currents that flow through them.

For visualizing non-linear behavior (e.g., distortion, clipping) you need to use the transient analysis. The AC analysis always assumes linear behavior and gives you the small-signal frequency response of circuits like filters and amplifiers.

In both analysis types, you can also see the DC voltages and currents. More types are possible, like noise and impedance analysis.

When the circuit has been drawn correctly (no floating nodes, for example), an analysis type is selected. A calculation is then done over a user-defined time period or frequency range with a selected resolution. Setting this resolution is a trade-off between curve smoothness and calculation time. When the calculation is ready, you can start probing the circuit.

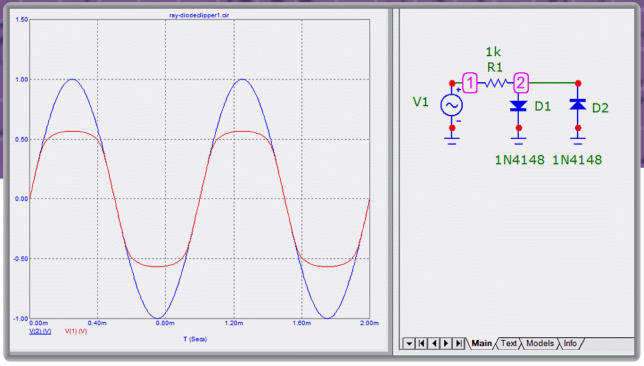


Figure 7: A transient analysis of the circuit from Figure 6 produces these signals.

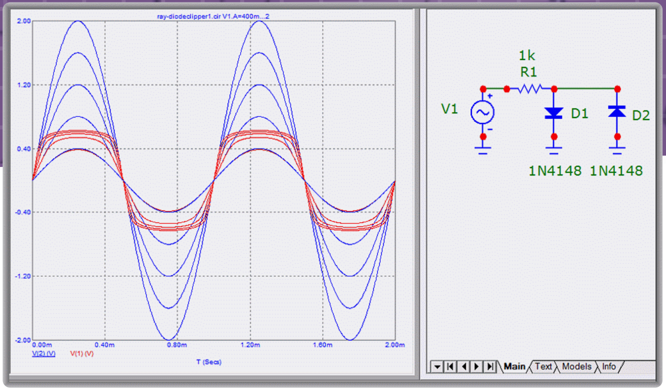


Figure 8: The output saturation gradually increases as we step the input signal's amplitude.

Sweep the Amplitude

As explained before, you can step a component value or a model parameter. Let's step the sine source's amplitude. Select *Stepping*... from the *Probe* menu. Choose V1 amplitude 'A' and step from *400m* to *2V* in 400 mV steps (don't forget to set *Step It* to *Yes*). Figure 8 shows the results.

As the circuit of Figure 6 is meant to be an audio effect, it would be useful to hear what it sounds like. Micro-Cap offers the ability to play the waveform on your sound card or store it as a WAV or CSV file.

The duration of the sound is set by the probe limits. Change *Maximum Run Time* to *500m* (and *Maximum Time Step* to *50u*) for half a second of sound. Next, double-click the trace window to open its properties and select the *Save Curves* tab. Select the wave you want to hear from the *Curves* list. If you enabled stepping (such as stepping the amplitude of the input signal), you can listen to each step by selecting it in the *What To Save* box and then pressing *Play.*

Simulation Resolution

Incorrect resolution settings can affect or even remove details from the simulation results. For example, a notch filter has a sharp dip at the frequency where it operates. Not having enough frequency resolution makes the dip appear less deep than it is. **Figure 9** and **Figure 10** show the results of the same Twin-T

Where to Find Components?

All components can be found via the Components tab and in the expandable library list box (the 'Panel') but there are also some shortcuts. There is the component bar at the top of the screen, but even faster are keys like 'R' for a resistor and 'C' for a capacitor. In addition, the program stores the recently used components in the start window of the Components tab.

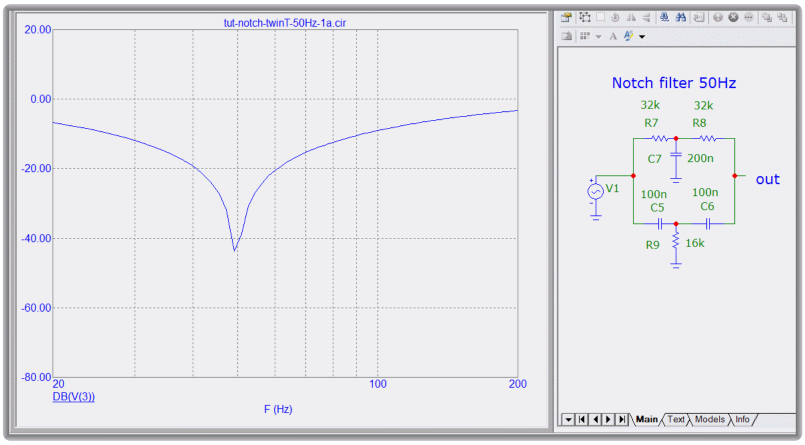


Figure 9: Simulation resolution matters. Here, ***Probe Limits*** *was set to 100 points.*

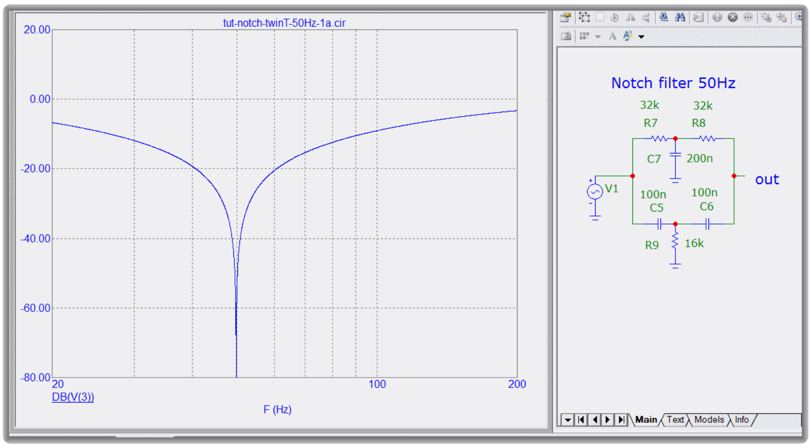


Figure 10: The same simulation as in Figure 9, but now with ***Probe Limits*** *set to 10,000 points.*

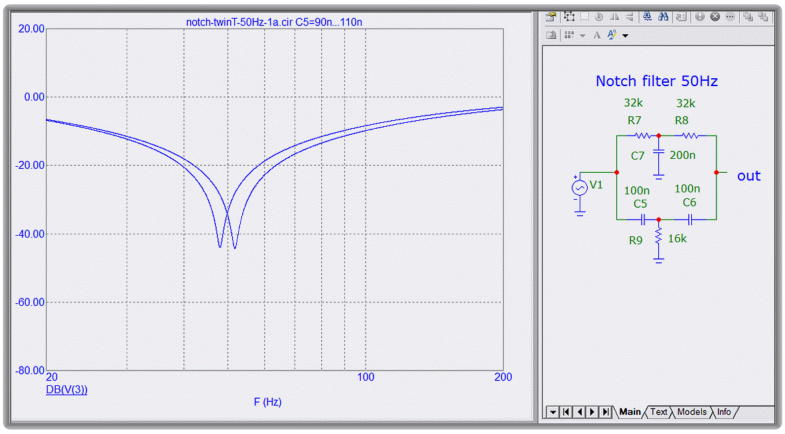


Figure 11: The same circuit as in Figure 9, but this time C5 is not identical to C6.

notch filter simulated twice, but with different resolution settings.

Ideal Components

This filter has an infinitely deep notch at 50 Hz with the given component values. However, before you build this ‘perfect’ filter, think a bit about component values. In a simulator, the two 100 nF capacitors will be identical and exactly 100 nF, and you obtain an infinitely deep 50 Hz notch filter as shown here.

Changing the value of e.g. C5 by plus or minus 10% has a large influence (**Figure 11**). The dip is now only 40 dB deep and the filter also detunes, resulting in just 35 dB suppression at 50 Hz. This shows how a simulator can help you to predict the way tolerances of real-world components work out in your circuit before building it.

For even more realistic simulations, you must also take other parasitic parameters, such as lead- and wire lengths, into account (see inset).

That’s Not All, Folks

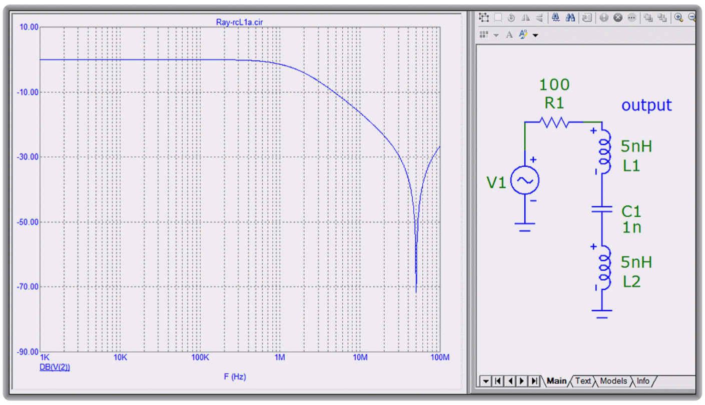
This article only scratched the surface of a complicated subject. As stated at the beginning of this article, a circuit simulator should be part of your toolbox, just like a multimeter and an oscilloscope. However, always keep in mind that a circuit simulator shows a simplified version of reality. Therefore, obtaining correct results with a simulator, Micro-Cap or not, still requires some insight in practical aspects of how to build a circuit. Knowing how to set it up and what its limitations are is a prerequisite. Combining simulation with building a real prototype is still a good idea. So, do not dispose of your breadboard with its bad contacts yet (how to model these?).

220336-01

Component Model Limitations

A simulation is only as good as the models it uses. All components in our filter are ideal. In practice, this is not true. For our simple low-pass filter, the main limitation is the series inductance of the capacitor leads. Each millimeter of wire adds roughly 1 nH of inductance that increases the impedance of the capacitor above a certain frequency, reducing the filtering effect.

Let’s assume your capacitor has 5 mm leads, and simulate this by adding two 5 nH inductors in series with the capacitor connections (which is the same as adding one 10 nH series inductor).



We find that the filter has a sharp dip around 50 MHz and then starts to ‘leak’ from 50 MHz up. This is caused by the resonant LC circuit. If you want to filter RF interference up to 1 GHz this is a problem, but in special cases you can make use of this effect. For instance, if you wish to filter a sampling clock frequency at 50 MHz, then this is an improvement, but you do have to tweak the inductance value.

The above is an example of how a simulator can help you to build up a circuit the right way.

Question: if wires add inductance, does using SMD capacitors then make a filter without inductance? No, but it comes closer. The capacitor still has a finite length (2 mm is 2 nH) and its PCB pad connection to ground adds inductance too. Vias can add 0.3 nH to 1 nH depending on how you define and place them.

Questions or Comments?

Do you have technical questions or comments about his article? Email the author at rs.elc.projects@gmail.com or Elektor at editor@elektor.com.

WEB LINK

[1] Micro-Cap website: https://www.spectrum-soft.com/download/download.shtm

PAUL Award 2022

Young Technical Talents and Their Creative Solutions

By Florian Schäffer on behalf of Elektor

Promoting young electronics talent is an important mission, and this also applies to the German Electronics Design and Manufacturing Association (FED). As part of this year's FED conference in Potsdam, Germany, the PAUL Award, worth 6,000 euro, was awarded again in 2022. Anyone between the ages of 15 and 25 was entitled to submit a creative electronics solution as an entry. Three winning teams are: Fisego (First), Manuel Wilke (Second), and BMK (Third).

Group photo with the nominees and winners of the PAUL Award 2022 in Potsdam (copyright FED e.V.).

Power Strip with Fire Protection

The winning team *Fisego* developed a power strip with built-in temperature protection based on a microcontroller to prevent fires caused by overload. The jury honored the idea, the implementation, and the documentation with the first prize of 3,000 euro. In addition, Sophia Reiter (a 22-year-old student of electrical engineering) and Fabian Goedert (a 25-year-old studying civil engineering) happily received the media prize, which was awarded independently by Elektor and presented to them at the ceremony in Potsdam.

Jury member Jürgen Deutschmann presents the first prize for the fire-safe multi-purpose socket to Sophia Reiter and Johannes Steube (Copyright FED e.V.).

In the 3D-printed prototype socket, a circuit with an ATmega328, an ACS712 Hall Effect sensor and a DS18B20 temperature sensor monitor the power consumption and temperature in the housing. In a three-stage warning phase, the parameters are displayed on an OLED display. When a limit value is exceeded, a warning with instructions for action is issued first, before the socket automatically switches off when the next limit value is reached. In contrast to sockets with thermal fuses or automatic circuit breakers, this one allows the user to intervene before total shutdown by, for example, switching off an individual load.

According to Sophia Reiter, the self-imposed goal of the two students was a multiple socket for the Internet of Things (IoT) with a built-in fire protection system, which is not only to be used in the private sector, but might also be of use to industry users and can monitor any type of machine or plant. The recently founded company has already taken a further step toward this goal by manufacturing prototypes of the socket using extrusion moulding.

FED Recognizes Lack of Young Talent and Launches Idea Promotion Program

The PAUL Award was held for the first time in 2020, but then had to take a break due to COVID, so that the second award ceremony could only take place this year on September 28. As Jürgen Braunsteiner, newly elected board member of the FED, emphasized in his introductory speech in front of around 100 guests, the industry is facing a generational change, with small and medium-sized companies in particular struggling with problems in recruiting young talent.

Promoting young people with creative ideas is therefore an important goal: They should be motivated by the PAUL Award to develop their projects and be given a chance to present them and establish contacts with industry.

As a positive example, Braunsteiner mentioned the young Australian Cynthia Sin Nga Lam, whose *H2prO* project aims to clean wastewater using photocatalysis and generate electricity at the same time. Her device, which has the potential to change the world, caused quite a stir in the media in 2014. The project demonstrated that new ideas can be found everywhere if you only dare to work on realizing them yourself with simple means.

Energy Harvesting with Peltier Element

The small field of five candidates also had to meet the additional challenge of energy harvesting: The projects were required to work without an external energy source — a task that was solved, for example, by

Team Fisego with the special prize from Elektor (copyright FED e.V.).

Second place went to Manuel Wilke for his battery management system (copyright FED e.V.).

the team *Drink Safe*: They developed a cup coaster that, while strongly reminiscent of similar gadgets that keep cups warm via USB, has a completely different task: The coaster measures the temperature of the cup and signals via an LED when the drink has cooled down enough to be enjoyed without burning the drinker. Considering warnings such as "Caution, hot drink" on disposable cups, this is not too far-fetched an idea.

In order to function, the project makes use of the effect of a Peltier element, which generates a current flow when there is a temperature difference between its two sides. Because the current generated in this process is very small, it requires an integrated step-up control to drive the LED. This eliminates the need for an additional power source such as a battery. The idea and implementation earned the team third place.

Weather Station and Battery Management System

Another team's idea of installing a water wheel in a gutter downpipe and using it to operate a self-sufficient IoT weather station without any further power supply was, unfortunately, unable to convince the jury, which may also have been due to the fact that the project was still struggling with teething troubles. After the event, the two students were unable to say whether they would let this stop them from continuing their work or whether they would pursue a new idea.

Second prize (2000 euro) was awarded to Manuel Wilke, a fifth-semester student at the Berlin University of Applied Sciences (BHT), who developed the *nandomBMS* battery management system (BMS). The design is based on an XC4000 Cortex-M4 microcontroller and an LTC6813 load balancer with BMS from Analog Devices and can monitor up to 18 cells connected in series. The microcontroller can communicate with the outside world using CAN. The board is to be used primarily in an in-house E-Buggy and features galvanic isolation between the high-voltage and low-voltage sides.

Next Round of the FED PAUL Award in 2023

Next year, the organizers would particularly like to see more participants, which is to be achieved through more public relations work, for example. In addition, there will no longer be any additional technical requirements. Only the costs are required to stay within pocket money limits, which means that the focus will probably be more on school groups and the maker scene. Those who want to participate individually or as a team can submit their application now and then work on it until September 30, 2023. All information can be found on the PAUL Award website [1]. ◀

220599-01

Translated to English by Jörg Starkmuth.

The winning PAUL Award projects were exhibited at the FED conference: a new prototype of the multi-purpose socket, the unpopulated board of the battery management system, and the Drink Safe device.

Team BMK, which won an award for its Drink Safe project, came in third place (copyright FED e.V.).

WEB LINKS

[1] PAUL Award: https://www.paul-award.de

My First Software-Defined Radio

Built in Less Than 15 Minutes

By Clemens Valens (Elektor Labs)

Did you know that you can create an FM radio receiver simply by drawing a few blocks and lines on a virtual canvas? If painting isn't your passion yet, it will be after reading this article.

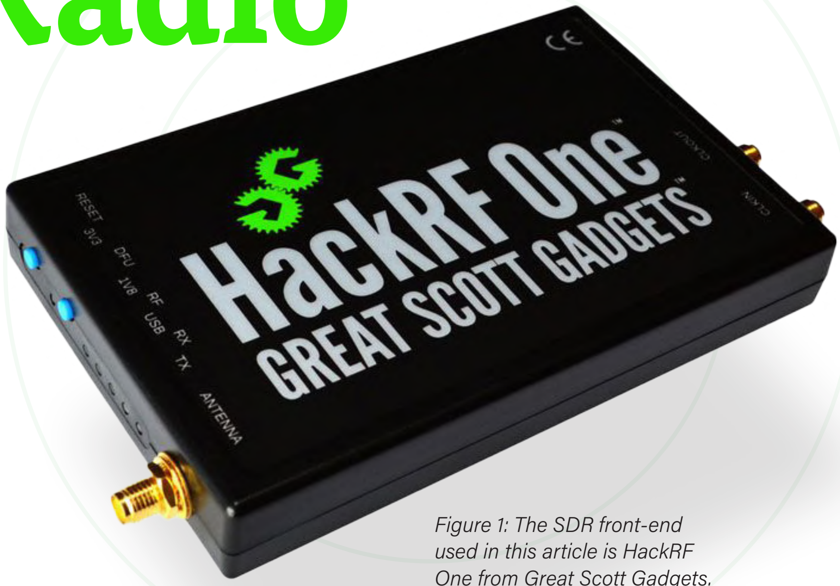


Figure 1: The SDR front-end used in this article is HackRF One from Great Scott Gadgets.

Software-Defined Radio, also known as SDR, is about modulating and demodulating radio signals in software. Instead of having special electronic circuitry to do this, like a mixer or demodulator, SDR uses software. Of course, there is some hardware involved in turning analog radio signals into digital and vice versa, but that is all you need.

Please note that to keep things simple, in this article we will limit ourselves to receiving radio signals only, but most of the presented material is also valid for transmitting. SDR can work both ways.

SDR Can Do Everything

SDR offers many advantages over analog radio. First, SDR understands every modulation technique you can think of, and more. This means that with SDR you can not only listen to AM and FM radio, but you can also do Wi-Fi or Bluetooth or decode digital TV or DAB+ or do HAM radio stuff and more.

SDR can do this because it implements the modulation or demodulation technique in software, and software is easy to change. Because everything is done in software, SDR can also do things that are very difficult or even impossible to do with electronic components only.

One Size Fits All

A second advantage of SDR is that you need only one piece of hardware to do all of this. You don't need a separate FM receiver and a Wi-Fi receiver and a digital TV receiver and what not. SDR uses the same front-end for everything.

This front-end is an analog-to-digital converter (ADC) with an antenna that turns analog radio signals into digital signals that can be processed with a computer. For transmitting, a digital-to-analog converter (DAC) is needed. In reality, this device is a bit more than just an ADC and/or DAC. It also does some mixing to bring the RF signal of interest into the range it can sample (and the other way around when transmitting), but its basic function remains the same.

SDR front-ends come in many flavors, and you must choose one that fits your budget and desires. The more expensive the converter, the more you can do with it. Here we use the HackRF One from Great Scott Gadgets [1] (**Figure 1**). This is not the cheapest SDR front-end, but it supports receiving and transmitting, and it works up to 6 GHz. Also, not unimportant, it is supported by many SDR software platforms like GnuRadio [2], SDR Sharp (SDR#) [3] and SDR++ [4]. For the project described below, other (cheaper) SDR front-ends will probably work equally well as long as they are capable of working in the FM radio band (87.5...108 MHz).

Open Source and Hardware

Another nice thing of SDR is that a lot of it is open-source and open hardware, created by a large community of enthusiasts. They design and publish schematics for front-ends and develop software tools and algorithms for SDR applications. HackRF One is open-source too, and the design files are available on GitHub [5].

Prerequisites

As mentioned before, for this SDR project I used HackRF One as the front-end with an 88 cm telescopic antenna. We will use GNU Radio for the software part. Actually, we will use GNU Radio Companion a.k.a. GRC, which is the graphical interface for GNU Radio. GNU Radio is available for Linux, Windows and macOS and so chances are that your computer can run it too. It even runs on a Raspberry Pi.

Drag 'n' Drop Programming

GRC is a graphical programming tool which lets you create a radio application without doing any real programming. You just drag and drop blocks on a canvas that you interconnect with virtual wires. The programming remains limited to configuring the blocks. GRC comes with a large library of blocks for creating all sorts of modulation and demodulation schemes.

We start with a blank canvas. It is not completely blank, as it contains already two blocks: an *Options* block and a *Variable* block. A double left mouse click on a block opens it, so you can change its parameters. The *Options* block lets you define some low-level options, and you can set a title for the canvas, see **Figure 2**. We will leave everything at the default values.

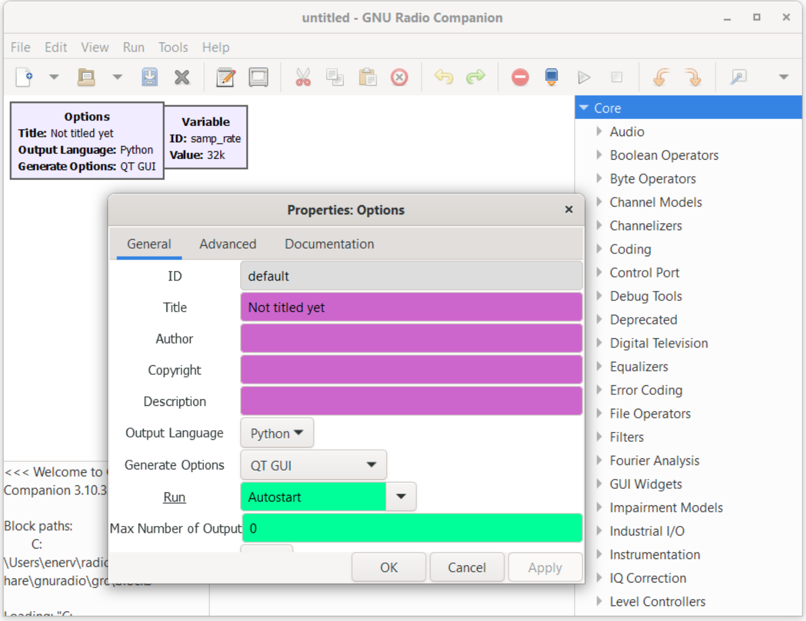
Global Sample Rate

The *Variable* block defines a global sample rate variable named `samp_rate`. This is used in other blocks. Strictly speaking, you don't need it, but it is practical, and so we keep it. However, its value is too low for sampling FM signals, and so we change it to 10 million. The maximum value for HackRF One is 20 million. If you set it too high, your computer may find it hard to keep up.

Add an RF Source

Now we need an RF input device. In GRC, this is called a source. There is no special HackRF One source block. The *Soapy* section has a Soapy HackRF Source, but that didn't work for me. The *osmocom* source did. The OsmoSDR front-end no longer exists, but the block supports other SDR hardware as well, including HackRF One.

Double-clicking on it opens its properties, and we see that the sample rate is set to `samp_rate`, the name of the variable block GRC gave us. We only have one parameter to change here, and that is the RF gain of channel 0, which is too high for FM radio signals that tend to be rather strong. We can set it to zero.



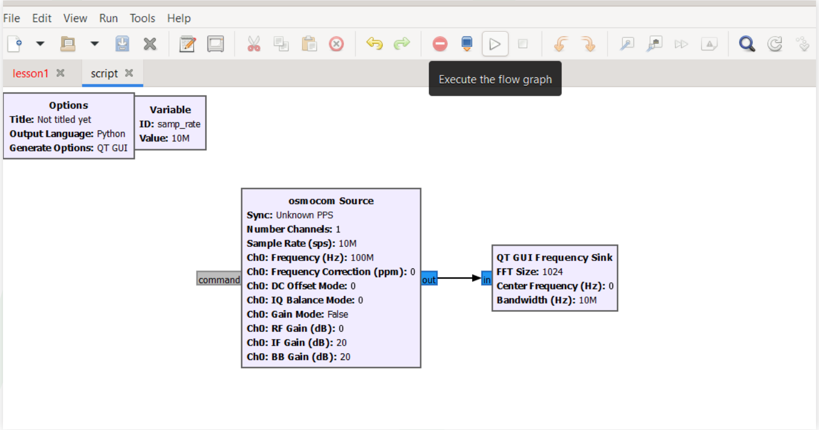
Options
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 32k

Properties: Options

ID: default
Title: Not titled yet
Author:
Copyright:
Description:
Output Language: Python
Generate Options: QT GUI
Run: Autostart
Max Number of Output: 0

Figure 2: A new project in GNU Radio Companion (GRC) comes with two blocks. Double-clicking a block opens its properties.



Options
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 10M

osmocom Source
Sync: Unknown PPS
Number Channels: 1
Sample Rate (sps): 10M
Ch0: Frequency (Hz): 100M
Ch0: Frequency Correction (ppm): 0
Ch0: DC Offset Mode: 0
Ch0: IQ Balance Mode: 0
Ch0: Gain Mode: False
Ch0: RF Gain (dB): 0
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 10M

▲

Figure 3: Click the play button to execute your first flow graph.

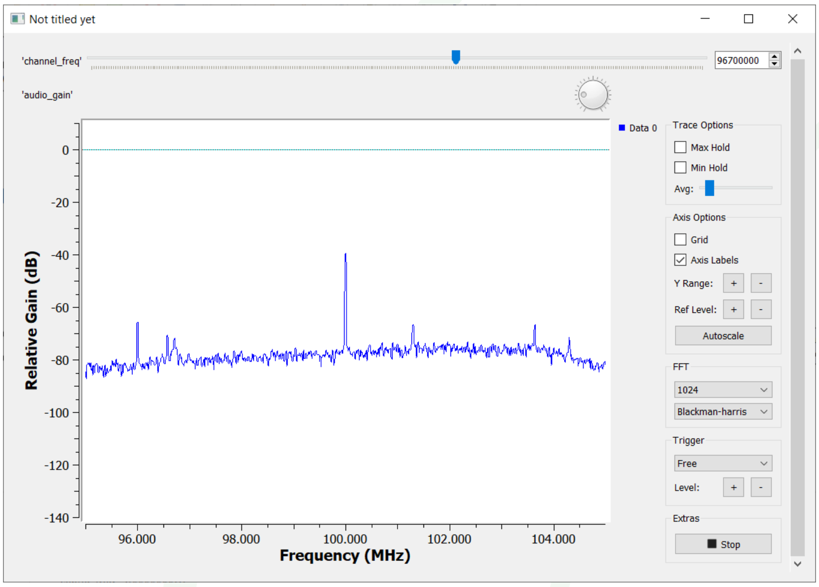
Spectrum Analyzer

To quickly see if things are working, we can add a spectrum analyzer block to the canvas and connect it to the output of the RF source block. It is called the QT GUI Frequency Sink and is found in the *Instrumentation* section of GRC. The way these blocks are named can be a bit confusing, and it makes them sometimes hard to find.

To connect the frequency sink to the RF source, simply click its input and then click the source's output. The other way around works too, and so does drawing with the left mouse button pressed down. Note that you can only connect inputs and outputs of the same color, blue in this case.

First Try

You can now click the *Play* button with the little triangle on it to execute the flow graph (**Figure 3**). If you didn't



'channel_freq' 96700000

'audio_gain'

□ Max Hold
□ Min Hold
Avg:

□ Grid
☑ Axis Labels
Y Range: + -
Ref Level: + -

FFT
1024
Blackman-harris

Trigger
Free
Level: + -

▲

Figure 4: Enabling the Control Panel on the QT GUI Frequency Sink provides access to several controls.

do so already, you must save your canvas before you can continue. If all is well, a frequency plot will open. Fiddling with the antenna should produce slightly different results, but they may be difficult to see. To improve things a bit, we must configure the frequency plot.

Click the *Stop* button with the little square on it to kill the flow graph. Double-click the frequency sink block to open its properties and set averaging to medium. Set the center frequency to the same value as the Channel 0 frequency of the osmocom source, which is 100 MHz in our case. Next, click the *Config* tab and set *Control Panel* to *Yes*. Close the properties window and click *Play* again. Now the frequency plot has a control panel, and you can check the *Max Hold* box (**Figure 4**). Fiddling with the antenna should become more visible and if it does, you know that you can receive RF signals.

It is All Plug 'n' Play!

A nice thing to notice is that we had nothing to do to make the flow graph talk to HackRF One. There are no drivers to install, no ports to select, it is all plug and play.

Variables Are Practical

We just had to set the center frequency of the frequency sink to the same value as that of the RF source, meaning that it is needed in at least two places. Therefore, if we replace it by a global variable, like GRC did for the sample rate, we can set it in one place.

Copy the *sample rate* variable block, open its properties, and change the ID to `center_freq`. Set the value to 100 MHz, set the Channel 0 frequency of the osmocom source to `center_freq`, and set the center frequency of the frequency sink to `center_freq`. The two parameters are now connected. If we change the value of the `center_freq` variable, it will also change in all the other blocks using it.

Mixing

To achieve something with SDR you do, of course, need some knowledge about how radios work. The algorithms in SDR do the same things as the electronic circuits in an analog radio. So, to receive an FM radio station we must tune to it, demodulate and filter it and make it audible.

Tuning can be done with a so-called mixer or frequency multiplier. When done with quadrature or complex signals, it allows shifting frequencies up and down. In GRC, the blue inputs and outputs correspond to complex floating-point signals that are suitable for mixing.

The goal is to shift the frequency of interest to the center of the frequency plot, from which it is easier to extract. For tuning to a radio station, we therefore need a multiplier and another signal to control the frequency shift. A *Multiply* block is available in the *Math Operators* section. For the tune signal, we can use the *Signal Source* block from the *Waveform Generators* section.

Tune to a Station

FM radio stations have a known frequency, which we will call the channel frequency. As we want to be able to change this frequency easily, we create a global variable for it that we call `channel_freq`. Where I live,

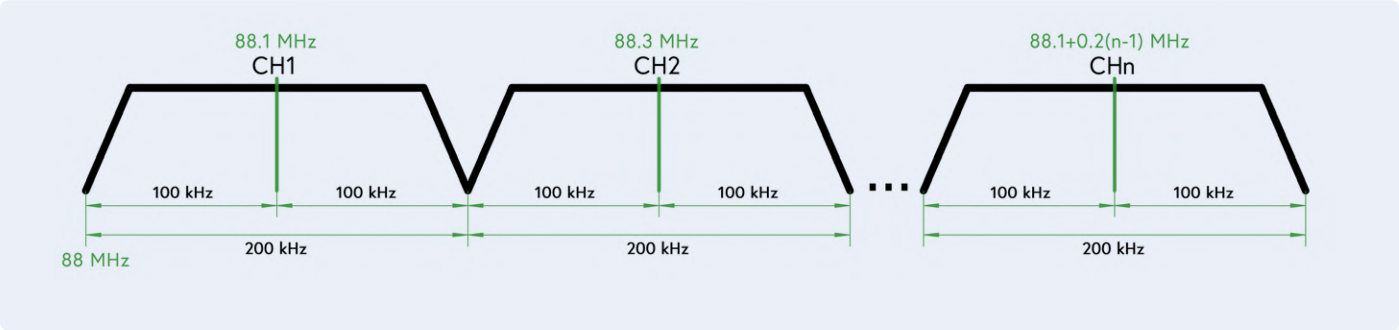
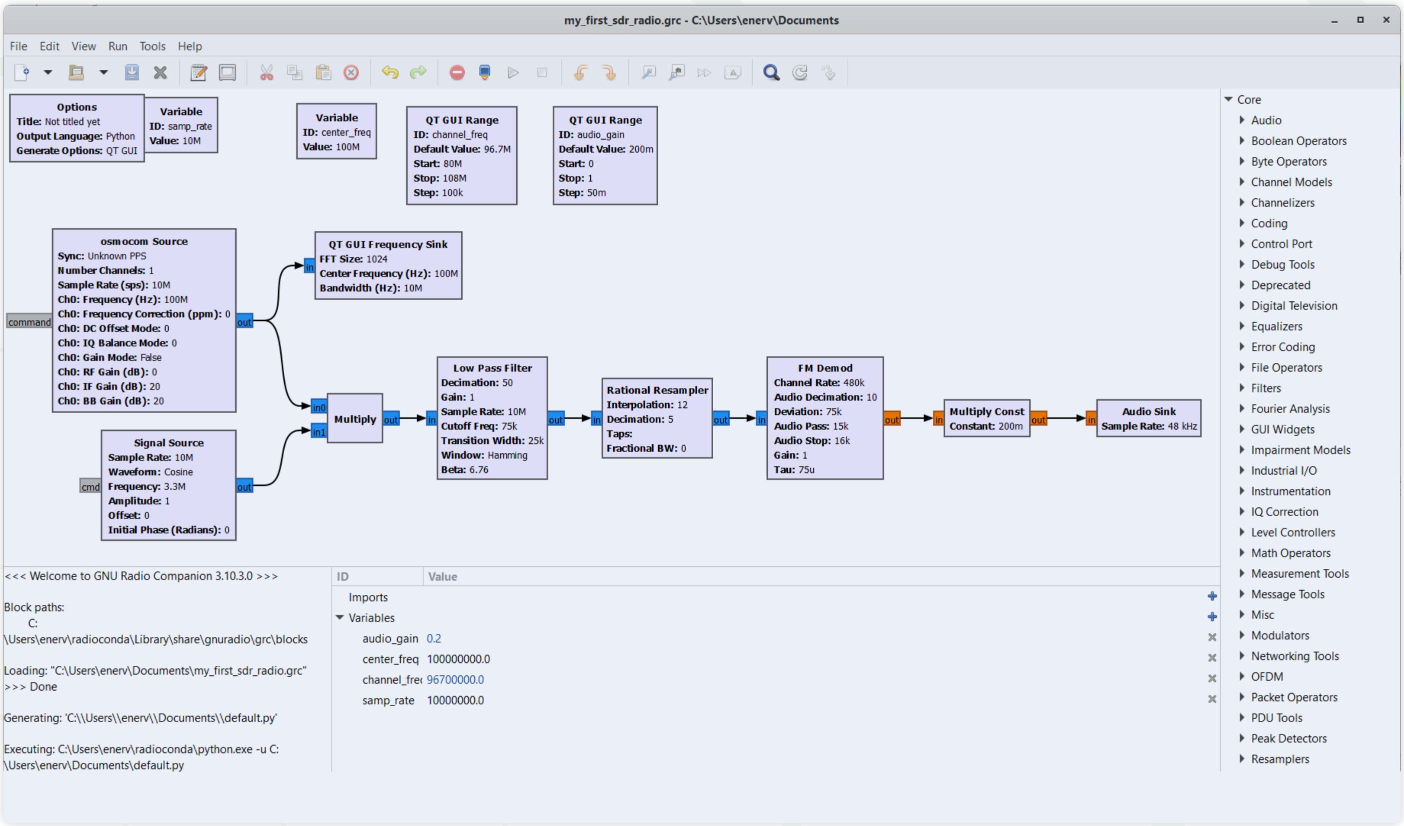


Figure 5: The FM radio frequency band is divided in 200 kHz-wide channels.



Options
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 10M

Variable
ID: center_freq
Value: 100M

QT GUI Range
ID: channel_freq
Default Value: 96.7M
Start: 80M
Stop: 108M
Step: 100k

QT GUI Range
ID: audio_gain
Default Value: 200m
Start: 0
Stop: 1
Step: 50m

osmocom Source
Sync: Unknown PPS
Number Channels: 1
Sample Rate (sps): 10M
Ch0: Frequency (Hz): 100M
Ch0: Frequency Correction (ppm): 0
Ch0: DC Offset Mode: 0
Ch0: IQ Balance Mode: 0
Ch0: Gain Mode: False
Ch0: RF Gain (dB): 0
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 100M
Bandwidth (Hz): 10M

Signal Source
Sample Rate: 10M
Waveform: Cosine
Frequency: 3.3M
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

Multiply

Low Pass Filter
Decimation: 50
Gain: 1
Sample Rate: 10M
Cutoff Freq: 75k
Transition Width: 25k
Window: Hamming
Beta: 6.76

Rational Resampler
Interpolation: 12
Decimation: 5
Taps:
Fractional BW: 0

FM Demod
Channel Rate: 480k
Audio Decimation: 10
Deviation: 75k
Audio Pass: 15k
Audio Stop: 16k
Gain: 1
Tau: 75u

Multiply Const
Constant: 200m

Audio Sink
Sample Rate: 48 kHz

```
<<< Welcome to GNU Radio Companion 3.10.3.0 >>>

Block paths:
	C:
\Users\enerv\radioconda\Library\share\gnuradio\grc\blocks

Loading: "C:\Users\enerv\Documents\my_first_sdr_radio.grc"
>>> Done

Generating: 'C:\\Users\\enerv\\Documents\\default.py'

Executing: C:\Users\enerv\radioconda\python.exe -u C:
\Users\enerv\Documents\default.py
```

ID	Value
Imports	
Variables	
audio_gain	0.2
center_freq	100000000.0
channel_fre	96700000.0
samp_rate	10000000.0

▲

Figure 6: The completed FM radio receiver flow graph with tune and volume controls. When you run it, it will look like Figure 4.

there is a radio station at 96.7 MHz, so I put that value in `channel_freq`.

The mixer must shift the channel frequency to the center frequency, so the frequency of the signal source must be the difference of the two. The difference between 100 MHz and 96.7 MHz is 3.3 MHz and that is indeed the frequency shown in the signal source block.

Down-Converting

With the station at the center frequency, we must extract its signal and mix it down so that it can be demodulated. Extracting means filtering and down-converting in SDR corresponds to reducing or decimating the sample rate. The *Low Pass Filter* from the *Filters* section combines both operations in a single block.

The FM band is divided in channels. The width varies per country, but is typically around 200 kHz. A radio station sits in the middle of a channel, which means that it has 100 kHz bandwidth on either side (**Figure 5**). The minimum sample rate required for a 100 kHz signal is 200 kHz (Nyquist-Shannon theorem). Our sample rate is 10 MHz, and therefore we set the decimation value of the filter block to 10 MHz / 200 kHz, which is 50. To extract the signal of interest, we don't need a super-steep filter. A cut-off frequency of 75 kHz with a transition width of 25 kHz is fine for a total bandwidth of 100 kHz.

Demodulating

We now have an FM-modulated quadrature signal with a sample rate of 200 kHz that we want to demodulate. We can use the *FM Demod* block from the *Modulators* section for this. The output of the demodulator is audio that we can feed into an audio sink, the computer's sound card in our case.

Sample Rate Issues

The problem we now run into is sample-rate matching. The audio sink supports several sample rates that depend on your sound card. On my computer, the ratio of the possible audio sample rates to the 200 kHz input sample rate is a none-integer value in all cases. But the FM demodulator can only decimate by an integer value. So, how can we get this right?

The solution is to insert a sample rate converter that can do fractional conversion. GRC has two options for this: a fractional resampler and a rational resampler. Since our input and output rates are integer values, we can use the rational resampler from the *Resamplers* section. A fractional resampler can be used too, but it consumes much more precious computing resources.

For the best audio quality, we set the audio sample rate to 48 kHz. Then, if we make the demodulator work at, say, 10 times that rate, at 480 kHz, we need a sample rate conversion of 200 : 480. This ratio can be simplified by finding the greatest common divisor of both values, which is 40. 200 : 40 = 5, and 480 : 40 = 12, so if we first upsample or interpolate by 12 and then downsample or decimate by five, a rate of 200 kHz is turned into a rate of 480 kHz. We must set this rate in the *FM Demod* block, together with a decimation factor of ten, to ensure that the output rate matches the input rate of the audio sink.

Add User Controls

If you execute this flow graph with the channel frequency set to an existing FM radio station nearby, you should now hear the radio program. Isn't that cool?

We can improve our radio a bit by adding a tune control, so you can easily tune in to another radio station. For this, we can use a *QT GUI Range* block from the *GUI Widgets* section. Set its ID to `channel_freq`. Fill in the parameters while making sure that they are all in range. Note that there is no point in setting a (very) small step value, as most FM stations are on MHz frequencies with only one decimal; 100 kHz will probably be fine.

You can choose a shape for the control, but Counter + Slider is the most practical as it allows you to see the frequency as a numerical value. Also, you can type a frequency directly. Rename or delete the old `channel_freq` variable block, as it is no longer needed and conflicts with the slider. You can now tune the radio like any other radio.

Signal Colors

Another improvement is adding a volume control. This is similar to the tune control, and you can copy it and adjust its parameters. Rename the ID to `audio_gain`. Pick a *Multiply Const* from the *Math Operators* section and insert it between the *FM Demod* block and the audio sink. Note how the wires are red. This is because the input and output of the multiplier are blue instead of orange. They are not of the same type.

Open the multiplier's properties and set the *IO Type* to *float*. Also set the field labeled *Constant* to `audio_gain` to connect it to the slider. Close the block and notice how the wires have turned black, as all the audio in- and outputs are now orange.

Output Options

By default, the output of GRC is a Python script with GUI as defined by the Generate Options in the Options block. But there are other possibilities, like C++. If you remove all the GUI blocks from the flow graph and set Generate Options to No GUI, the script can be run outside GRC. This lets you build stand-alone SDR applications.

That's it. Execute the flow graph and enjoy your software-defined FM radio receiver! To get deeper into the fascinating world of SDR, I highly recommend watching the excellent video course by Michael Ossmann [6] that inspired this article.

220659-01

Questions or Comments?

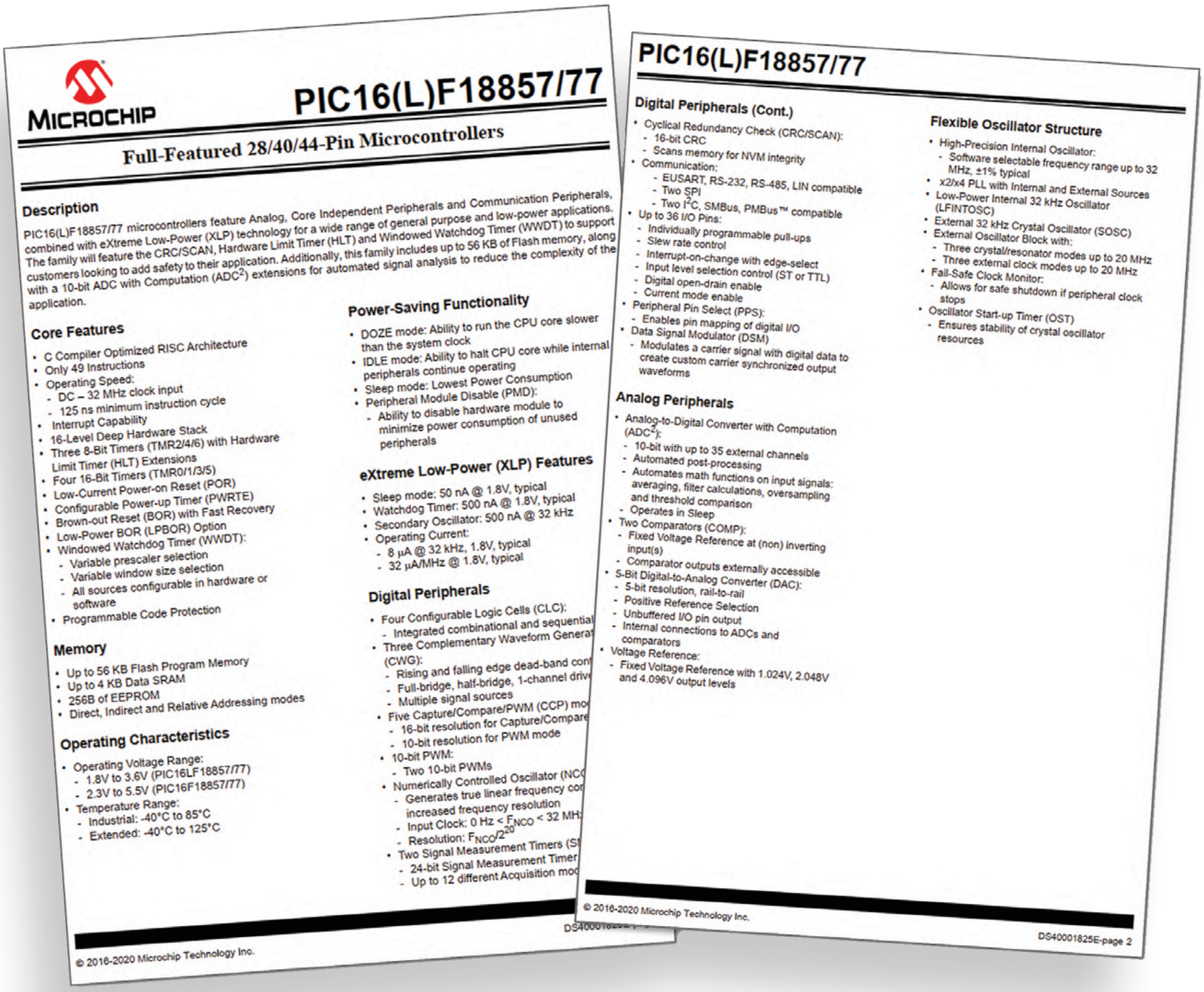
Do you have technical questions or comments about this article? Email the author at clemens.valens@elektor.com or contact Elektor at editor@elektor.com.

WEB LINKS

[1] HackRF One at Great Scott Gadgets: https://greatscottgadgets.com/hackrf/
[2] GNU Radio and GRC: https://www.gnuradio.org/
[3] SDR# a.k.a. SDR Sharp: https://airspy.com/download/
[4] SDR++: https://www.sdrpp.org/
[5] HackRF One at GitHub: https://github.com/greatscottgadgets/hackrf
[6] Video course on SDR by Michael Ossmann: https://greatscottgadgets.com/sdr/

Microcontroller Documentation Explained

(Part 1) Datasheet Structure



MICROCHIP

PIC16(L)F18857/77

Full-Featured 28/40/44-Pin Microcontrollers

Description

PIC16(L)F18857/77 microcontrollers feature Analog, Core Independent Peripherals and Communication Peripherals, combined with eXtreme Low-Power (XLP) technology for a wide range of general purpose and low-power applications. The family will feature the CRC/SCAN, Hardware Limit Timer (HLT) and Windowed Watchdog Timer (WWDT) to support customers looking to add safety to their application. Additionally, this family includes up to 56 KB of Flash memory, along with a 10-bit ADC with Computation (ADC^2) extensions for automated signal analysis to reduce the complexity of the application.

Core Features

- C Compiler Optimized RISC Architecture
- Only 49 Instructions
- Operating Speed:
  - DC – 32 MHz clock input
  - 125 ns minimum instruction cycle
- Interrupt Capability
- 16-Level Deep Hardware Stack
- Three 8-Bit Timers (TMR2/4/6) with Hardware Limit Timer (HLT) Extensions
- Four 16-Bit Timers (TMR0/1/3/5)
- Low-Current Power-on Reset (POR)
- Configurable Power-up Timer (PWRTE)
- Brown-out Reset (BOR) with Fast Recovery
- Low-Power BOR (LPBOR) Option
- Windowed Watchdog Timer (WWDT):
  - Variable prescaler selection
  - Variable window size selection
  - All sources configurable in hardware or software
- Programmable Code Protection

Memory

- Up to 56 KB Flash Program Memory
- Up to 4 KB Data SRAM
- 256B of EEPROM
- Direct, Indirect and Relative Addressing modes

Operating Characteristics

- Operating Voltage Range:
  - 1.8V to 3.6V (PIC16LF18857/77)
  - 2.3V to 5.5V (PIC16F18857/77)
- Temperature Range:
  - Industrial: -40°C to 85°C
  - Extended: -40°C to 125°C

Power-Saving Functionality

- DOZE mode: Ability to run the CPU core slower than the system clock
- IDLE mode: Ability to halt CPU core while internal peripherals continue operating
- Sleep mode: Lowest Power Consumption
- Peripheral Module Disable (PMD):
  - Ability to disable hardware module to minimize power consumption of unused peripherals

eXtreme Low-Power (XLP) Features

- Sleep mode: 50 nA @ 1.8V, typical
- Watchdog Timer: 500 nA @ 1.8V, typical
- Secondary Oscillator: 500 nA @ 32 kHz
- Operating Current:
  - 8 µA @ 32 kHz, 1.8V, typical
  - 32 µA/MHz @ 1.8V, typical

Digital Peripherals

- Four Configurable Logic Cells (CLC):
  - Integrated combinational and sequential
- Three Complementary Waveform Generat (CWG):
  - Rising and falling edge dead-band con
  - Full-bridge, half-bridge, 1-channel driv
  - Multiple signal sources
- Five Capture/Compare/PWM (CCP) mo
  - 16-bit resolution for Capture/Compare
  - 10-bit resolution for PWM mode
- 10-bit PWM:
  - Two 10-bit PWMs
- Numerically Controlled Oscillator (NC
  - Generates true linear frequency co increased frequency resolution
  - Input Clock: 0 Hz < F_{NCO} < 32 MH
  - Resolution: $F_{NCO}/2^{20}$
- Two Signal Measurement Timers (S
  - 24-bit Signal Measurement Timer
  - Up to 12 different Acquisition mo

© 2016-2020 Microchip Technology Inc.

DS400018

PIC16(L)F18857/77

Digital Peripherals (Cont.)

- Cyclical Redundancy Check (CRC/SCAN):
  - 16-bit CRC
  - Scans memory for NVM integrity
- Communication:
  - EUSART, RS-232, RS-485, LIN compatible
  - Two SPI
  - Two I^{2}C, SMBus, PMBus™ compatible
- Up to 36 I/O Pins:
  - Individually programmable pull-ups
  - Slew rate control
  - Interrupt-on-change with edge-select
  - Input level selection control (ST or TTL)
  - Digital open-drain enable
  - Current mode enable
- Peripheral Pin Select (PPS):
  - Enables pin mapping of digital I/O
- Data Signal Modulator (DSM)
  - Modulates a carrier signal with digital data to create custom carrier synchronized output waveforms

Analog Peripherals

- Analog-to-Digital Converter with Computation (ADC^2):
  - 10-bit with up to 35 external channels
  - Automated post-processing
  - Automates math functions on input signals: averaging, filter calculations, oversampling and threshold comparison
  - Operates in Sleep
- Two Comparators (COMP):
  - Fixed Voltage Reference at (non) inverting input(s)
  - Comparator outputs externally accessible
- 5-Bit Digital-to-Analog Converter (DAC):
  - 5-bit resolution, rail-to-rail
  - Positive Reference Selection
  - Unbuffered I/O pin output
  - Internal connections to ADCs and comparators
- Voltage Reference:
  - Fixed Voltage Reference with 1.024V, 2.048V and 4.096V output levels

Flexible Oscillator Structure

- High-Precision Internal Oscillator:
  - Software selectable frequency range up to 32 MHz, ±1% typical
- x2/x4 PLL with Internal and External Sources
- Low-Power Internal 32 kHz Oscillator (LFINTOSC)
- External 32 kHz Crystal Oscillator (SOSC)
- External Oscillator Block with:
  - Three crystal/resonator modes up to 20 MHz
  - Three external clock modes up to 20 MHz
- Fail-Safe Clock Monitor:
  - Allows for safe shutdown if peripheral clock stops
- Oscillator Start-up Timer (OST)
  - Ensures stability of crystal oscillator resources

© 2016-2020 Microchip Technology Inc.

DS40001825E-page 2

Figure 1: The opening pages list the features integrated into the microcontroller. (Source: Microchip Technology)

By **Stuart Cording** (Elektor)

Documentation — regardless of whether you love it or hate it, you still have to negotiate it. Microcontrollers have a lot of documentation because, in comparison to other, simpler semiconductor devices, they are really complex. In this series of articles we'll take a look at microcontroller documentation, what is in the datasheet, what isn't, and where to find the missing details.

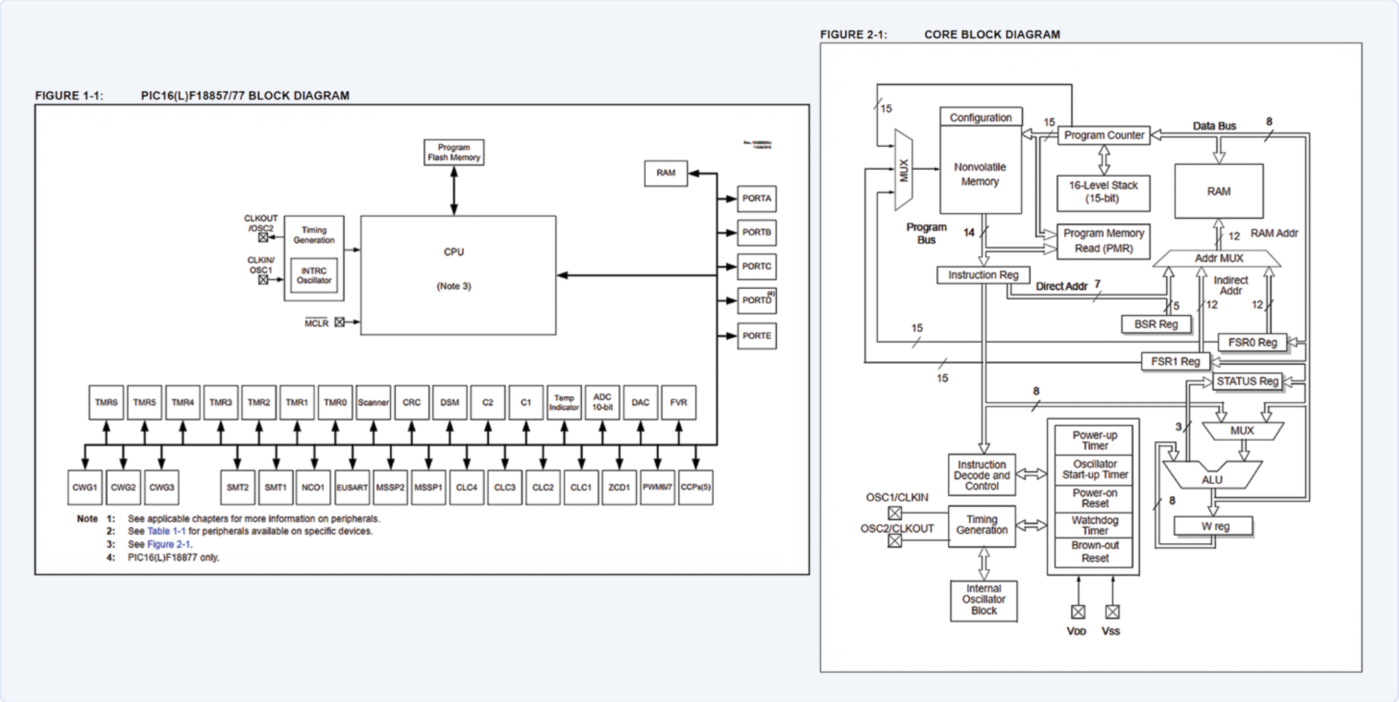


Figure 2: The block diagram is a good starting point to ascertain the overall capabilities of the microcontroller (left) and sometimes the processing core too (right). (Source: Microchip Technology)

Microcontroller datasheets can easily run to more than 600 pages today. Thankfully, Elektor has plenty of resources, such as beginner articles and books, to help you get started. However, at some point, you are going to have to acquaint yourself with the real microcontroller documentation. Although the microcontroller datasheet runs to so many pages, you're unlikely to find all the information you are looking for.

To complement the datasheet you will also have to find documentation for tools that turn source code into firmware, tools for developing code and debugging, and tools for mass-production programming. If you're new to the world of microcontrollers, this guide will help you to understand where things are documented, how to read the hieroglyphics of their contents, and where to find out if there were any mistakes in what you were reading.

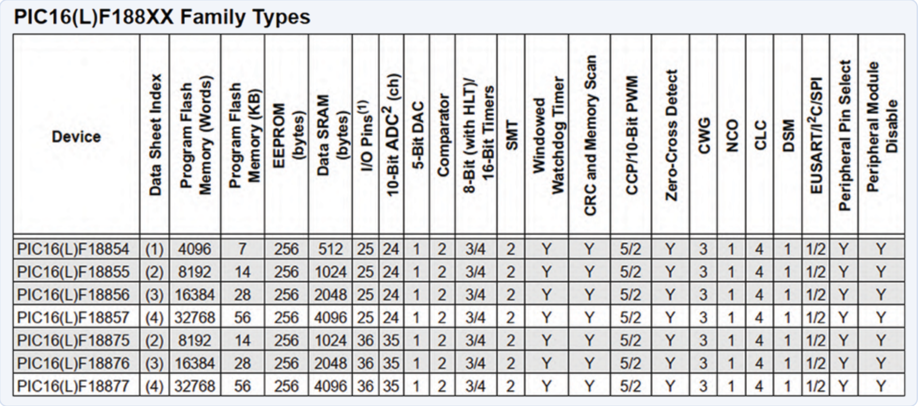
Microcontroller Documentation Example: PIC16F18877

To kick of part one of this series of three articles about microcontroller documentation, let's focus on a simple, 8-bit microcontroller such as the Microchip Technology PIC16F18877, and what documentation is offered. Open the link [1] and then click on "Documents". You'll see that we are offered the datasheet, errata, some "Supporting Collateral" covering some application-specific items and evaluation boards, the programming specifications, and a long list of application notes. Further down there is some source code to accompany some of the application notes, some sales brochures, and a white paper on analog-to-digital converters (ADCs).

What's in the Microcontroller Datasheet?

We're going to start by downloading the datasheet for the PIC16F18877 [2]. Datasheets for microcontrollers can be quite daunting and, perhaps surprisingly, not everything you need to know is included. **Figure 1** shows what you should find as a minimum.

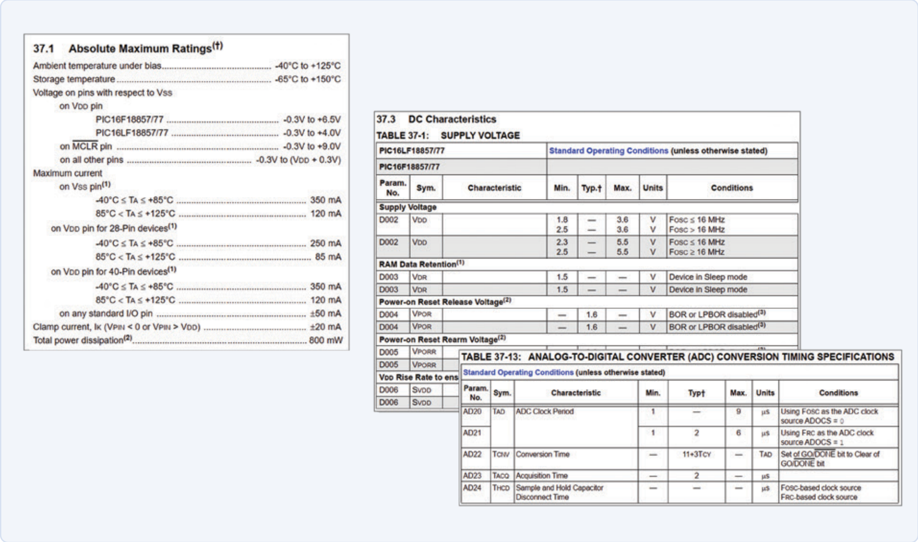
- Detailed information on one or more microcontrollers — Often, several different microcontrollers with various numbers of pins and packages will be created from a single silicon die. Rather than create and maintain documentation for each variant, you're more likely to find several devices covered in a single datasheet. This is the case here, as explained on page 1, with two variants covered: the PIC16F18857 and PIC16F18877.
- Block diagram of the microcontroller — This typically includes the processing core (the level of detail varies; the more complex the core, the simpler its diagrammatic implementation), the memories, buses, and peripherals. From this, you can quickly ascertain the basic capabilities of the microcontroller. The microcontroller block diagram (**Figure 2**) appears on page 18, while a block diagram of the processing core is on page 33.
- Packaging options — These will range from through-hole types (if still available) to a range of surface-mount options. In this example, they start on page 4.
- Memory-size options — The microcontroller may well be offered with differing sizes of RAM, flash, EEPROM, and any other memory types, such as caches. In our example, we get a quick overview on page 3. The table lists memory sizes along with how many of each peripheral are implemented (**Figure 3**).
- On-chip peripheral block diagrams — The functionality of on-chip



PIC16(L)F188XX Family Types

Device	Data Sheet Index	Program Flash Memory (Words)	Program Flash Memory (KB)	EEPROM (bytes)	Data SRAM (bytes)	I/O Pins(1)	10-Bit ADC^2 (ch)	5-Bit DAC	Comparator	8-Bit (with HLT)/ 16-Bit Timers	SMT	Windowed Watchdog Timer	CRC and Memory Scan	CCP/10-Bit PWM	Zero-Cross Detect	CWG	NCO	CLC	DSM	EUSART/I^{2}C/SPI	Peripheral Pin Select	Peripheral Module Disable
PIC16(L)F18854	(1)	4096	7	256	512	25	24	1	2	3/4	2	Y	Y	5/2	Y	3	1	4	1	1/2	Y	Y
PIC16(L)F18855	(2)	8192	14	256	1024	25	24	1	2	3/4	2	Y	Y	5/2	Y	3	1	4	1	1/2	Y	Y
PIC16(L)F18856	(3)	16384	28	256	2048	25	24	1	2	3/4	2	Y	Y	5/2	Y	3	1	4	1	1/2	Y	Y
PIC16(L)F18857	(4)	32768	56	256	4096	25	24	1	2	3/4	2	Y	Y	5/2	Y	3	1	4	1	1/2	Y	Y
PIC16(L)F18875	(2)	8192	14	256	1024	36	35	1	2	3/4	2	Y	Y	5/2	Y	3	1	4	1	1/2	Y	Y
PIC16(L)F18876	(3)	16384	28	256	2048	36	35	1	2	3/4	2	Y	Y	5/2	Y	3	1	4	1	1/2	Y	Y
PIC16(L)F18877	(4)	32768	56	256	4096	36	35	1	2	3/4	2	Y	Y	5/2	Y	3	1	4	1	1/2	Y	Y

Figure 3: Take a look at the memory options for the PIC16F18x7 variants along with the peripherals implemented. (Source: Microchip Technology)



37.1 Absolute Maximum Ratings(†)

Ambient temperature under bias .... -40°C to +125°C
Storage temperature .... -65°C to +150°C
Voltage on pins with respect to Vss
on VDD pin
PIC16F18857/77 .... -0.3V to +6.5V
PIC16LF18857/77 .... -0.3V to +4.0V
on $\overline{\text{MCLR}}$ pin .... -0.3V to +9.0V
on all other pins .... -0.3V to (VDD + 0.3V)
Maximum current
on Vss pin(1)
-40°C ≤ TA ≤ +85°C .... 350 mA
85°C < TA ≤ +125°C .... 120 mA
on VDD pin for 28-Pin devices(1)
-40°C ≤ TA ≤ +85°C .... 250 mA
85°C < TA ≤ +125°C .... 85 mA
on VDD pin for 40-Pin devices(1)
-40°C ≤ TA ≤ +85°C .... 350 mA
85°C < TA ≤ +125°C .... 120 mA
on any standard I/O pin .... ±50 mA
Clamp current, IK (VPIN < 0 or VPIN > VDD) .... ±20 mA
Total power dissipation(2) .... 800 mW

37.3 DC Characteristics

TABLE 37-1: SUPPLY VOLTAGE

PIC16LF18857/77			Standard Operating Conditions (unless otherwise stated)				
PIC16F18857/77							
Param. No.	Sym.	Characteristic	Min.	Typ.†	Max.	Units	Conditions
Supply Voltage							
D002	VDD		1.8 2.5	— —	3.6 3.6	V V	Fosc ≤ 16 MHz Fosc > 16 MHz
D002	VDD		2.3 2.5	— —	5.5 5.5	V V	Fosc ≤ 16 MHz Fosc ≥ 16 MHz
RAM Data Retention(1)							
D003	VDR		1.5	—	—	V	Device in Sleep mode
D003	VDR		1.5	—	—	V	Device in Sleep mode
Power-on Reset Release Voltage(2)							
D004	VPOR		—	1.6	—	V	BOR or LPBOR disabled(3)
D004	VPOR		—	1.6	—	V	BOR or LPBOR disabled(3)
Power-on Reset Rearm Voltage(2)							
D005	VPORR						
D005	VPORR						
VDD Rise Rate to ens							
D006	SVDD						
D006	SVDD						

TABLE 37-13: ANALOG-TO-DIGITAL CONVERTER (ADC) CONVERSION TIMING SPECIFICATIONS

Standard Operating Conditions (unless otherwise stated)							
Param. No.	Sym.	Characteristic	Min.	Typ†	Max.	Units	Conditions
AD20	TAD	ADC Clock Period	1	—	9	µs	Using FOSC as the ADC clock source ADOCS = 0
AD21			1	2	6	µs	Using FRC as the ADC clock source ADOCS = 1
AD22	TCNV	Conversion Time	—	11+3TCY	—	TAD	Set of GO/$\overline{\text{DONE}}$ bit to Clear of GO/$\overline{\text{DONE}}$ bit
AD23	TACQ	Acquisition Time	—	2	—	µs	
AD24	THCD	Sample and Hold Capacitor Disconnect Time	—	—	—	µs	FOSC-based clock source FRC-based clock source

Figure 4: Electrical specifications are provided as absolute limits, as well as minimum, typical, and maximum value for DC and AC values, as well as timing limits. (Source: Microchip Technology)

peripherals is more easily explained in diagrams than in words. Block diagrams provide a vital source of understanding as well as clarity on pin connections and clock sources, where the microcontroller features multiple clock outputs from its oscillator and, if implemented, PLL (phase-locked loop).

We'll take a look at some examples shortly in part 2 of the series.

- Register descriptions — Each peripheral can be configured in a specific manner. For example, a UART (universal asynchronous receive/transmit serial interface) can often be configured for different baud rates, number of bits, and so on. The register description explains how to configure the peripheral, and how to ascertain its state after something happens, such as receiving a byte of data. How these are deciphered is covered later in part two of the series.
- Electrical specifications — These inform the user the limits in voltage and current that can be applied to or drawn from the microcontroller's pins. These are typically defined twice: once as absolute maximum values, and once more as minimum, typical, and maximum values during normal operation. There are also timing characteristics, as shown in **Figure 4**, and can be found starting at page 592 in this datasheet.
- Recommendations for tools – You'll need tools to convert source code to assembler and tools to debug the results. Most datasheets provide some sort of recommendation of the available tools. Here they are covered briefly on page 638.

Looking Ahead to Block Diagrams

With the basic structure of a microcontroller datasheet understood, we will examine, in the next article, how registers are described and how to decipher the block diagrams. We will also be looking in detail at the two most important blocks of any microcontroller — the clock and oscillator — and the implementation of the reset circuitry.

200271-01

Questions or Comments?

Do you have questions or comments about this article? Email the author at stuart.cording@elektor.com.

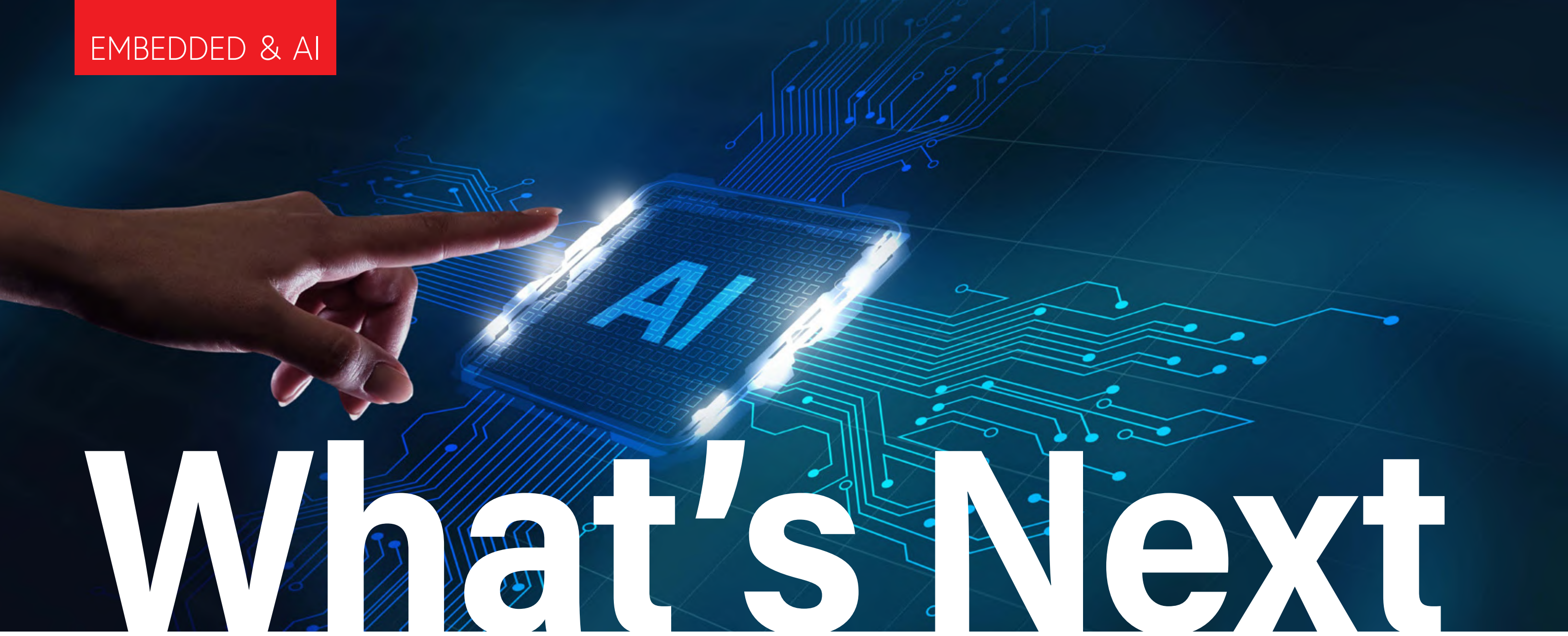
WEB LINKS

[1] PIC16F18877 Product Page: https://bit.ly/2KS1s8C

[2] PIC16F18877 Datasheet: https://bit.ly/3nNwPjn

Related Products

- T. Hanna, *Microcontroller Basics with PIC*, Elektor 2020
www.elektor.com/microcontroller-basics-with-pic
- A. Pratt, *Programming the Finite State Machine*, Elektor 2020.
www.elektor.com/programming-the-finite-state-machine



What's Next for AI and Embedded Systems?

Tools, Platforms, and Writer Replacements

by Stuart Cording (Elektor)

AI is often wielded as a universal problem-solving sword, ready to take on any challenge. Other times it is posited as heralding the end of civilization itself. As engineers, we know that neither case is true. But how can we make better use of AI, or more correctly, machine learning, in the applications we develop? And will the most recent advancements in AI really make us redundant?

Mentioning artificial intelligence (AI) is a surefire way to grab the attention of the mainstream press. Thanks to the Internet and cloud services, coupled with sometimes questionable data sources, new AI-driven services seem to pop up everywhere. After a few mouse clicks, your words can be placed into the mouth of Morgan Freeman or your image integrated into a new work of art. And while this can be amusing and thought-provoking, it isn't easy to find a link back to the world of embedded systems. But that is just what is going on within the industry.

Most embedded systems use rules-based programming approaches to implement their functionality. Given a set of input data, a series of if/else or switch statements determine how to respond. This works well for a limited number of inputs. However, at some point, the number of inputs or the subtlety in their relationships makes it difficult to define clear, programmatic rules.

As an example, imagine a motor in a factory running 24 hours per day, seven days a week. Experience tells you that, over time, wear occurs, and bearing grease thickens. Over time, this changes spin-up time, noise generated, vibration modes, motor operational temperature, and current drawn. Regular maintenance is how this challenge is handled today, but it results in fixed downtime that stops production. Whether this is too often or too infrequent is often hard to say.

Furthermore, it is unlikely to catch a pending failure resulting from a hairline fracture in the shaft, bearings, casing, or fixtures. When correctly trained, AI approaches can determine pending failures using a complex mix of data sources. If such intelligence can be deployed into a microcontroller-based system, you get an affordable monitoring system that saves time and money, and reduces wastage through unnecessary lubrication and part replacement.

Getting Intelligence Into Microcontrollers

At the microcontroller level, we're talking about machine learning (ML) rather than AI. This indicates programming a machine to use rules developed through analysis of available data to make decisions. While microcontrollers are more than powerful enough to execute such ML algorithms, learning from training data remains beyond their means and requires at least a desktop computer, if not a cloud server, to execute. Edge Impulse, founded in 2019, has developed a platform dedicated to ML in embedded systems and has successfully partnered with the world's semiconductor vendors [1] to roll out broad support.

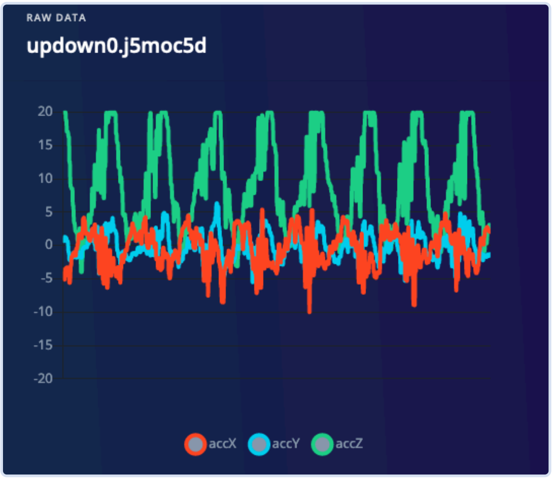
The starting point for any ML application is data. While some applications, like autonomous driving, need terabytes of training data, simple microcontroller-based systems can learn from as little as a few kilobytes of data. Getting the data off the board and onto the Edge Impulse environment is, therefore, the first challenge. The initial thought would be to use an Arduino's serial interface to send it to your PC, then upload it as a text file from there. However, their platform is set up to ingest the data directly.

Data — Food for AI

One tool is the Data Forwarder [2], a command-line (CLI) application that sends data from a development board directly to the Edge Impulse environment. Using your username and password, a link is established between the serial port on your PC and the server. On the microcontroller side, all that is needed is data output over the serial interface in a comma- or tab-delimited format. As long as the sampling rate is relatively low, this is an ideal way to gather representative data directly from your sensors (**Figure 1**). More powerful embedded systems, such as the Raspberry Pi or the NVIDIA Jetson Nano, can use the software development kit (SDK) [3] provided. This also supports sensors such as microphones and cameras generating more significant quantities of data.

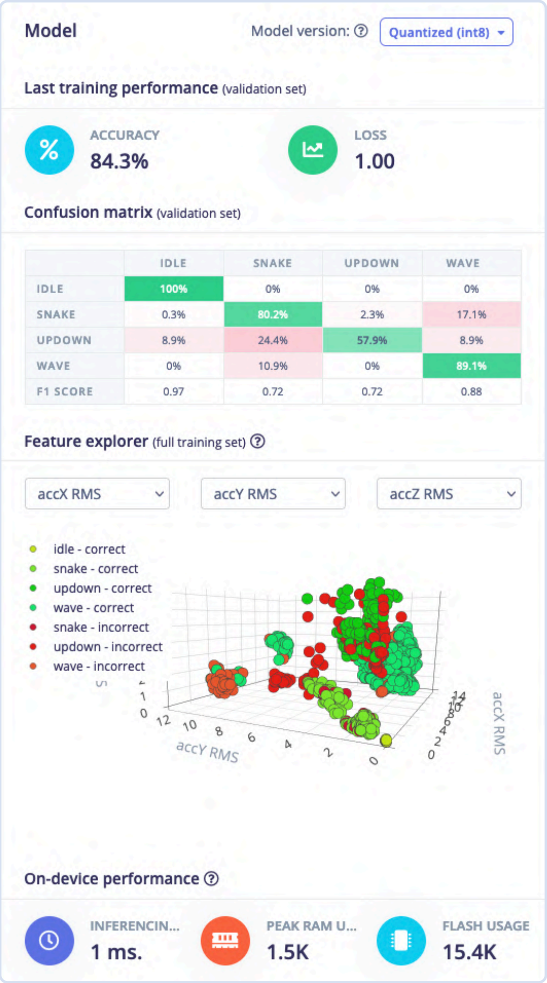
With the data uploaded, the next step is to define an 'impulse.' These consist of two blocks. The first slices the data into smaller chunks and uses signal processing techniques to extract features. This ensures that the available sensor data is transformed into consistent information for the second processing stage. The second block is where the learning and classification take place (**Figure 2**). In an example project, "Continuous Motion Recognition," there is a good explanation of how these blocks are configured to analyze accelerometer data and classify this input as one of four gestures [4].

This is the most critical step in any ML development, often needing lateral thinking and several iterations to determine the best approach. Sometimes, ignoring some sensor inputs is the best solution, while other times, more data is required. You may even find you are overlearning, or the neural network model chosen is a poor fit for the classification you are attempting to make. Another crucial step is anomaly classification. In the example project, there are four defined gestures. However, other motions similar to the learned gestures need to be excluded. Good anomaly detection delivers a more robust ML result.


RAW DATA

updown0.j5moc5d

Figure 1: A three-axis accelerometer delivers motion data to develop an ML-based gesture recognition application. (Source: Edge Impulse)


Model

Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY 84.3%

LOSS 1.00

Confusion matrix (validation set)

	IDLE	SNAKE	UPDOWN	WAVE
IDLE	100%	0%	0%	0%
SNAKE	0.3%	80.2%	2.3%	17.1%
UPDOWN	8.9%	24.4%	57.9%	8.9%
WAVE	0%	10.9%	0%	89.1%
F1 SCORE	0.97	0.72	0.72	0.88

Feature explorer (full training set)

accX RMS | accY RMS | accZ RMS

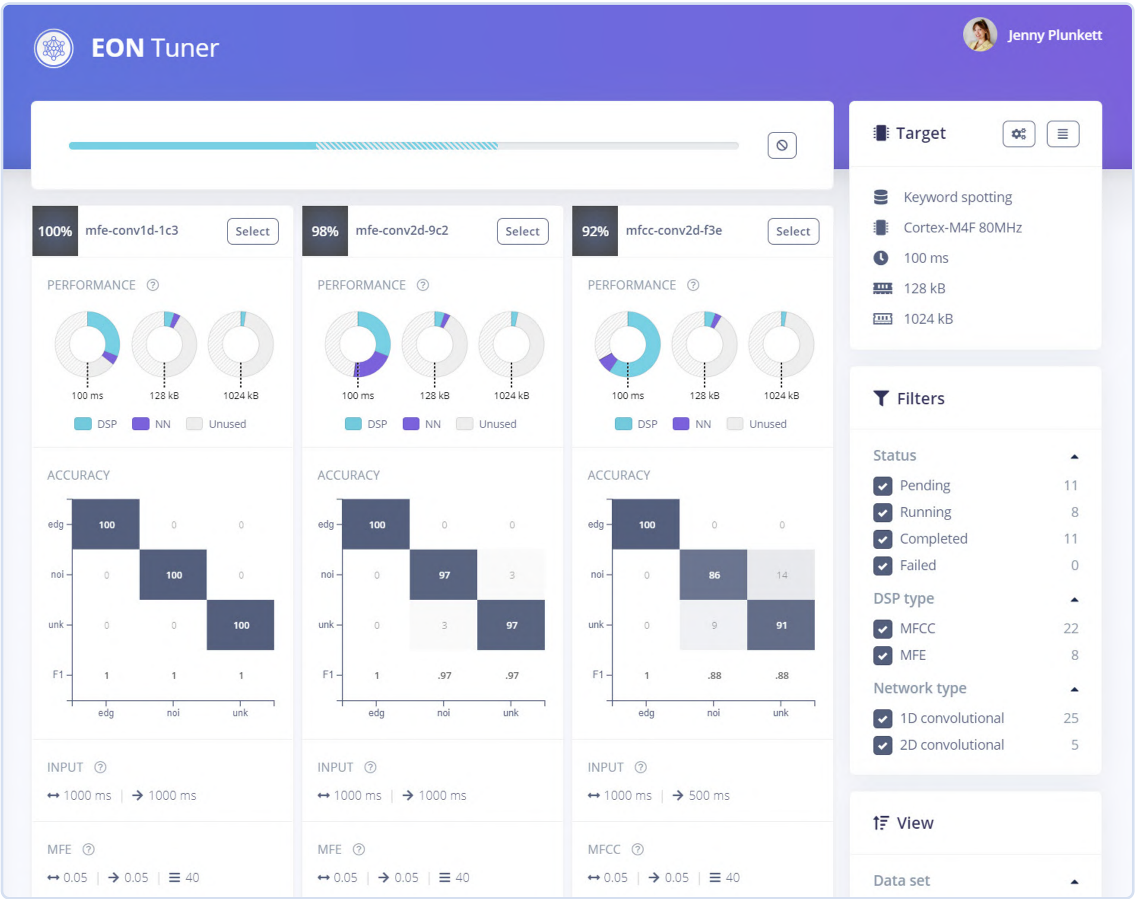
On-device performance

INFERENCIN... 1 ms.

PEAK RAM U... 1.5K

FLASH USAGE 15.4K

Figure 2: The Edge Impulse environment provides feedback on accuracy, speed, and memory usage after an initial training cycle. (Source: Edge Impulse)



EON Tuner

Jenny Plunkett

100% mfe-conv1d-1c3 Select

PERFORMANCE ?

ACCURACY

	edg	noi	unk
edg	100	0	0
noi	0	100	0
unk	0	0	100
F1	1	1	1

INPUT ?
↔ 1000 ms | → 1000 ms

MFE ?
↔ 0.05 | → 0.05 | ≡ 40

98% mfe-conv2d-9c2 Select

PERFORMANCE ?

ACCURACY

	edg	noi	unk
edg	100	0	0
noi	0	97	3
unk	0	3	97
F1	1	.97	.97

INPUT ?
↔ 1000 ms | → 1000 ms

MFE ?
↔ 0.05 | → 0.05 | ≡ 40

92% mfcc-conv2d-f3e Select

PERFORMANCE ?

ACCURACY

	edg	noi	unk
edg	100	0	0
noi	0	86	14
unk	0	9	91
F1	1	.88	.88

INPUT ?
↔ 1000 ms | → 500 ms

MFCC ?
↔ 0.05 | → 0.05 | ≡ 40

Target

Keyword spotting
Cortex-M4F 80MHz
100 ms
128 kB
1024 kB

Filters

Status

☑ Pending	11
☑ Running	8
☑ Completed	11
☑ Failed	0

DSP type

☑ MFCC	22
☑ MFE	8

Network type

☑ 1D convolutional	25
☑ 2D convolutional	5

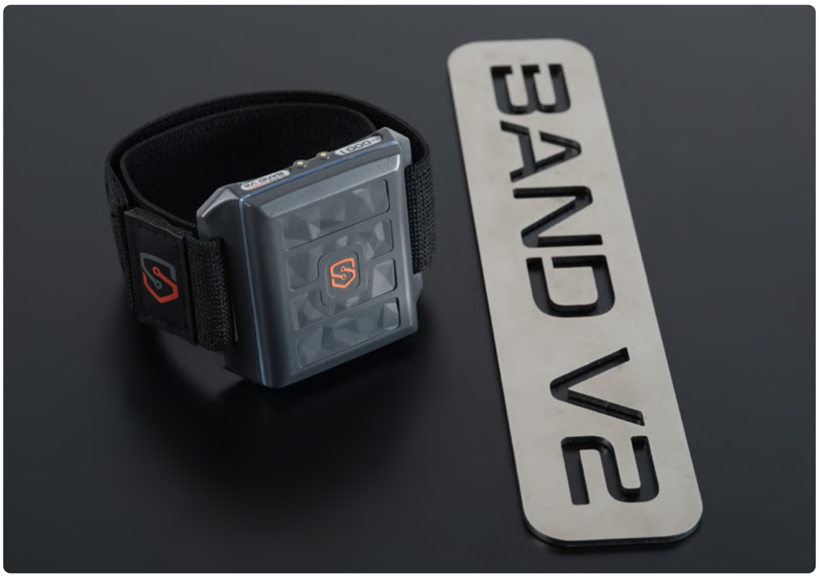
View

Data set

Figure 3: ML algorithms are further optimized for the target microcontroller using EON Tuner, enabling improvements in inference response times and reductions in memory usage. (Source: Edge Impulse)

The final step is deployment. Through the web interface, firmware for your chosen device is downloaded for integration into your application. For Arduino, a library is generated, while for other microcontrollers, you can generate a C++ file. Of course, the performance of microcontrollers varies massively. To ensure the best possible result, Edge Impulse offer their EON Tuner [5]. This tool can further improve detection accuracy, speed up inference, and reduce memory requirements by using information such as target device, memory size, and latency (**Figure 3**).

Figure 4: SlateSafety used Edge Impulse to provide edge ML physiological monitoring and improve BAND V2 safety monitoring. (Source: SafetySlate)



ML in Real Applications

Real applications are using this approach to embed ML capability. SlateSafety's BAND [6] integrates a range of biometric sensors to monitor workers who operate under challenging situations (**Figure 4**). These range from first responders to industrial workers that wear heavy personal protective equipment (PPE), such as firefighters. Their product normally uploads data to the cloud so that colleagues can monitor a user's vital signs.

However, especially in disaster situations, connectivity can be patchy or non-existent. The development team utilized Edge Impulse to integrate edge ML into the existing product, trained on historical biometric data. Using the EON Tuner, the algorithm was optimized to the hardware, then deployed using an over-the-air update. Now the BAND can provide the wearer with a warning when there is a risk of heat exhaustion, even when no wireless connection is present.

Improving Product Development with AI

Of course, AI doesn't have to be integrated into the product. It can also be used for its development. Today, many complex applications are developed using a model-based approach, essentially developing a description of how something would work using software and mathematical equations rooted in physics.

However, even this approach has its limits. This is where Monolith and their self-learning AI platform [6] comes in.

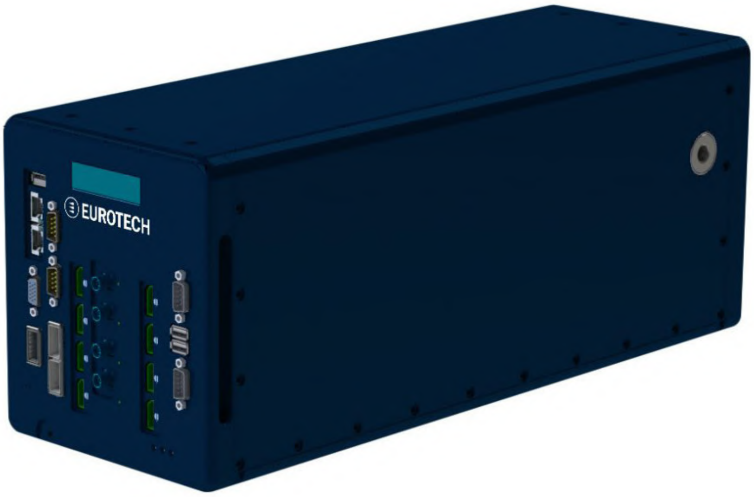
The platform is capable of learning the physical properties of complex systems based on data that has already been collected. For example, vehicles go through a series of tests at a test track, with multiple sensors monitoring yaw and roll along with wheel speed and acceleration. Gathering data for different suspension stiffnesses provides a good overview of how the vehicle responds to a collection of driving situations. Typically, the data would be analyzed, resulting in new settings to apply during the next test run. Monolith can evaluate the data from the first set of test runs and predict the outcome of changes to the suspension with a high degree of accuracy. The results can be used to hone in on the best suspension settings more quickly, reducing the number of additional physical test runs required.

This approach can also be applied to metrology. Gas meters must be exceptionally accurate to ensure correct billing, but this is challenging to achieve when the meter must measure a range of different gases. For one customer, simulating ultrasonic meters had pushed purely mathematical analysis to its limits, leaving repetitive test processes as the only solution for calibration to achieve the certification needed. Luckily, all the testing resulted in a rich collection of data for analysis. Using self-learning AI models, the amount of testing required has fallen by up to 70%, speeding up development significantly.

Serious Compression of AI Compute

Competitions such as the DARPA Grand Challenge [8], where teams built autonomous vehicles that could traverse a winding course, triggered a wave of interest in self-driving cars. Today, almost twenty years later, significant sums of money have been spent, but little has resulted. Tesla garners headlines regularly in this space, often when over-enthusiastic owners place too much faith in their vehicle's capabilities [9]. Otherwise, only Waymo seems to offer true self-driving vehicles as a ride-hailing service [10], but these operate solely in Phoenix and San Francisco in the USA.

One of the issues is that getting a computer to drive a car is exceptionally challenging. Not only must the vehicle continually assess the situation around it, but it also has to predict the actions of other drivers and road users, such as pedestrians and cyclists, who may not stick to the rules of the road.



▲ *Figure 5: AI algorithm development is sped up using in-vehicle high-performance computers, such as this liquid-cooled DYNACOR 40-36. (Source: Eurotech)*

What is happening is that the electrical and electronic (E/E) architecture of vehicles is changing to meet the future needs of autonomous vehicles. With a multitude of sensors delivering vast quantities of data, the industry is moving to automotive Ethernet. Currently, this approach is forming the advanced driver-assistance systems (ADAS) that, through control of braking, acceleration, and steering, can step in when we drivers make a mistake. According to the Society of Automotive Engineers' (SAE) levels of vehicle autonomy, premium vehicles currently meet level 2+, with some touching into level 3 territory. However, complete 'sit back and relax' autonomy is level 5, so we still have some way to go.

Companies such as Eurotech are supporting the industry to speed up the development of the necessary algorithms. Currently, an eight-hour test drive gathers 120 TB of data that must be returned to the lab for processing and analysis. Improvements to AI algorithms can be tested in the laboratory using the gathered data, but little has been available to support testing and algorithm development in the field.

Leveraging their experience in liquid cooling, Eurotech offers a series of edge AI hardware that is up to the task, essentially compact supercomputers that fit in a vehicle's trunk. A unit such as the DYNACOR 40-36 is ruggedized for use in both on and off-road vehicles [11]. Featuring a 16-core Intel Xeon CPU with 64 GB of RAM and up to two NVIDIA GV100 GPUs with 32 GB of RAM, this fanless computer provides 237 TFLOPS to handle deep learning applications (**Figure 5**). Several gigabit Ethernet interfaces support ingesting the mass of sensor data, from radar and cameras to lidar, pouring it into 32 TB of solid-state storage. By undertaking more inference and reinforcement testing during test drives, the path to level 5 autonomy could be accelerated significantly.

Figure 6: OpenAI's DALL·E 2 AI-generated image of a self-driving vehicle powered by the trusty Commodore 64. A work in progress.

Does AI Put My Job at Risk?

An ongoing discussion on social media is whether advances in AI puts jobs in creative industries at risk. OpenAI's launch of DALL·E 2 [12] turns natural language requests into images (**Figure 6**). But, perhaps more impressive is its ability to edit existing images realistically. For example, it can remove objects in the foreground or background. And, given a work by a Dutch painter Vermeer, the AI can expand his "Girl with a Pearl Earring" to include a believable impression of the room she sat in when painted.

However, writers, such as the editorial team of Elektor and other renowned press outlets, have been shocked by the launch of ChatGPT [13]. This AI can interact with users in a conversational manner and in a multitude of languages. So far, discussions on the merits of silicon carbide MOSFETs (SiC) and gallium nitride transistors (GaN), and their benefits over silicon MOSFETs, have resulted in highly accurate responses. So, even niche topics seem well covered.

While exceptionally clever, the tool only knows answers to topics that occurred before it was trained. Because it is not continuously learning, it won't be up-to-date on current affairs or the latest K-pop band dramas (shame). Another niggle is that, after a while, the answers seem a little canned and formulaic. However, those looking for inspiration or a birthday poem in the style of a celebrity won't be disappointed.

To determine whether Elektor's future is reliant on pink fleshy things or computers, I offer you a summary of the topic of this article written by ChatGPT. Until next time ... or perhaps not!

In summary, embedded systems and AI are two technologies that are increasingly being used together to create intelligent, autonomous devices and systems. Embedded systems provide the hardware and software platform for AI algorithms to run on, while AI algorithms enable these systems to sense, analyze, and respond to their environment in a more intelligent and human-like way. As the capabilities of both embedded systems and AI continue to improve, we can expect to see a wide range of exciting new applications in fields such as robotics, healthcare, transportation, and more.

220673-01

Questions or Comments?

Do you have technical questions or comments about this article? Email the author at stuart.cording@elektor.com or contact Elektor at editor@elektor.com.

WEB LINKS

[1] Edge Impulse Partners: http://bit.ly/3vbqRhG
[2] Edge Impulse CLI Installation: http://bit.ly/3BQQt71
[3] Edge Impulse Ingestion SDK: http://bit.ly/3jplhp3
[4] Continuous motion recognition example project: http://bit.ly/3WAV9G5
[5] EON Tuner: http://bit.ly/3PQhFsr
[6] SlateSafety BAND: http://bit.ly/3YVpe5x
[7] Monolith: http://bit.ly/3BWZImh
[8] DARPA Grand Challenge, Wikipedia: http://bit.ly/3VnqWJI
[9] J. Stilgoe, "Tesla crash report blames human error - this is a missed opportunity," Guardian, January 2017: http://bit.ly/3Vjp7gI
[10] DYNACOR 40-36: http://bit.ly/3GdmgBS
[11] Waymo One: http://bit.ly/3FNG8Ku
[12] DALL·E 2: http://bit.ly/3PNPWsD
[13] ChatGPT: http://bit.ly/3PLjZRo

Digitizing Vertical Farming

Contributed by Würth Elektronik eiSos

Vertical farming is a very promising approach to feeding a growing Earth population. Optimized lighting, irrigation, fertilization, and climate control ensure the highest yields and best quality. This article presents an IoT-connected prototype.

A projected 2050 world population of almost 10 billion will require a 50-percent increase in global agricultural production, according to the UN Food and Agriculture Organization (FAO). Still, farmland and other agrarian activities are limited. Agrarian land has declined in recent decades, from nearly 40% of the world's land area in 1991 to just 37% in 2018 [1].

Globally, agricultural land is estimated to have covered 4.7 billion hectares in 2020 — about one-third of the total land area. Of that, roughly one third is cropland (about 1.6 billion hectares) and the other two thirds are permanent pastures and meadows (about 3.2 billion hectares) [2]. The cropland, 1.6 billion hectares, is 12% of the world's land area. These global shares have not changed significantly since 2000, while the world's population has increased from 6.15 billion in 2000 to 7.91 billion in 2021 — an almost 30% increase — and it continues to grow.

Innovative agricultural techniques, such as controlled-environment agriculture (also sometimes referred to as vertical farming), are promising as a means of increasing agricultural production to meet local or regional food needs, as well as helping to bring fresher and more nutritious food to billions of people while minimizing the environmental impact associated with conventional farming. Digital farms can only succeed if several key technologies, including state-of-the-art lighting systems, control, and monitoring, as well as regional acceptance and appropriate regulations, are effectively utilized.

The purpose of this article is to discuss how artificial lighting and remote monitoring and control are important to plant growth in vertical farming installations, and how you can build your own vertical farming facility in a fast and cost-effective way using rapid prototyping tools.

What's Digital Agriculture?

"Digital agriculture is the seamless integration of digital technologies into crop and livestock management and other processes in agriculture. For farmers, digital agriculture offers the opportunity to increase production, save costs in the long term, and eliminate risk" [3].

Not only can this approach increase production from currently available agricultural land, but it can also be successfully applied in dense urban areas where space is at a premium. The main benefits of indoor farming are that with it, you can grow your own crops, control environmental factors, build quality substances, and minimize carbon emissions. The most obvious benefit of indoor vertical farms is space. Vertical farms can dramatically increase productivity by accommodating more crops on the same land footprint.

There are multiple challenges in digital vertical farming. The biggest drawback of vertical farming is the high cost of electricity required to run a large number of LEDs. Instead of full-spectrum plain white light, LEDs that generate only the necessary colors can be used. However, providing optimal lighting conditions at scale and keeping energy consumption under control manually is a huge challenge.

Furthermore, the environmental conditions necessary for optimal growth change depending on the choice of plant and the phase of growth. It is extremely difficult to monitor and control several environmental

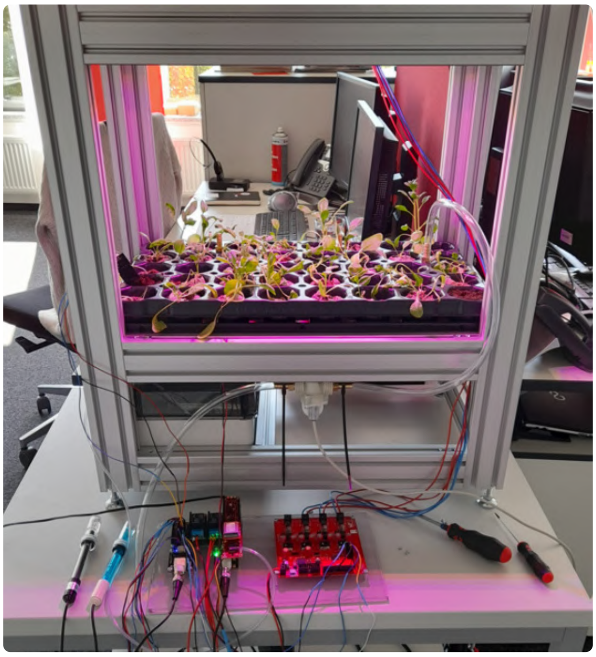


Figure 1: Indoor farming box prototype. (Source of all pictures: Würth Elektronik eiSos)

factors, such as temperature, humidity, soil moisture, and optimize them dynamically.

IoT to Automate Vertical Farming

In this article, an IoT solution to conquer the above challenges and to achieve complete automation is proposed. The Internet of Things (IoT) can be broadly defined as an umbrella term for a range of technologies that enable devices to connect and interact with each other. Interconnected devices generating data enable a range of new applications. Industrial automation, healthcare, smart homes, smart cities, smart grids, and smart farming are some of the areas in which IoT technology provides substantial benefits.

Dubbed the "fourth industrial revolution" or Industry 4.0, the Industrial IoT (IIoT) is the digitization of assets and processes that connects products, machines, services, and locations/sites to workers, managers, suppliers, and partners. Closer networking of the digital world with the physical world of machines helps achieve higher productivity, safety, efficiency, and sustainability.

The IIoT creates a universe of sensors that enables an accelerated deep learning of existing operations, allowing for rapid contextualization and automatic pattern- and trend-detection. This leads to true quantitative capture of qualitative operations, resulting in better quality and efficiency, higher safety and lower costs, among several other benefits.

An IoT-based solution to fully automate and control lighting, irrigation, and other environmental factors for a vertical farm is presented here. Further, this concept is demonstrated with rapid-prototyping tools through a fully automated, connected system with data analysis and cloud connectivity using different connectivity solutions for different environments. **Figure 1** shows a prototype indoor farming box.

"Feather"-Based Implementation

The core task of any IoT solution is to get data from the field to the cloud, where analysis generates the desired value addition for the application. To realize such an application, an open-source hardware and software ecosystem was used, and a

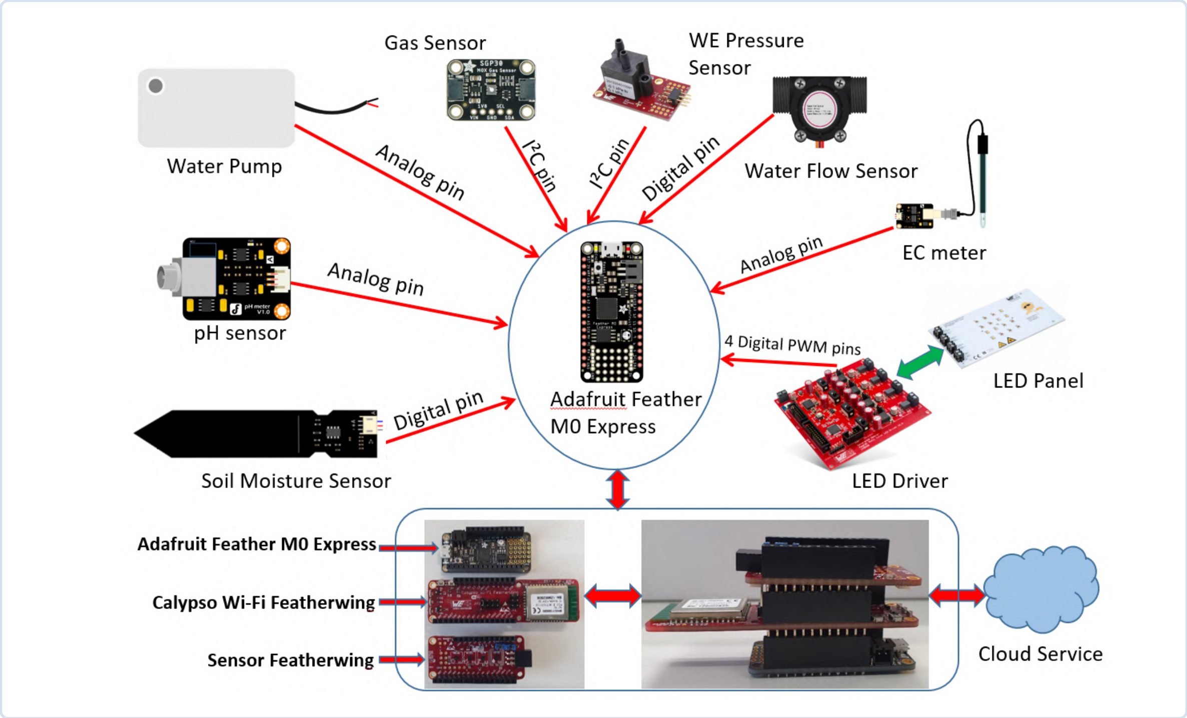


Figure 2: IoT system prototype based on the Feather formfactor for vertical farming.

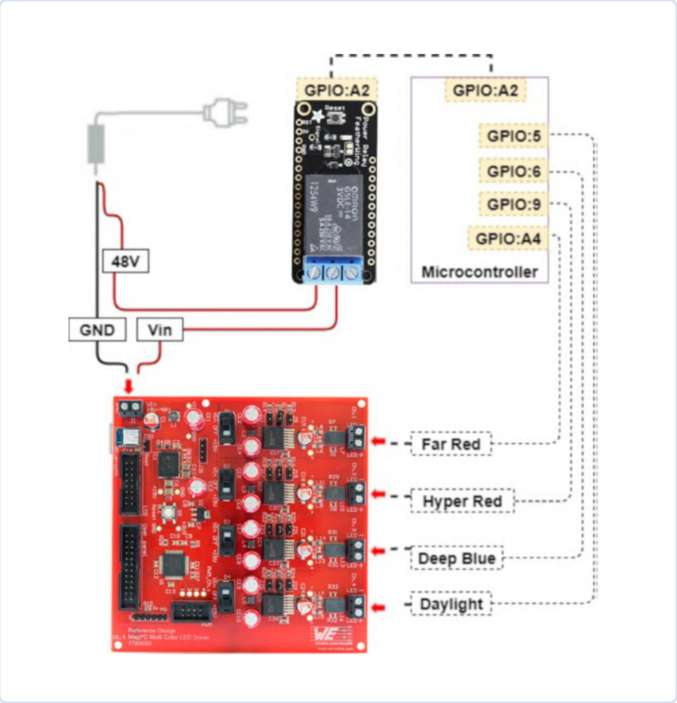


Figure 3: The horticulture lighting system consists of an array of monocolor LEDs and a LED driver.

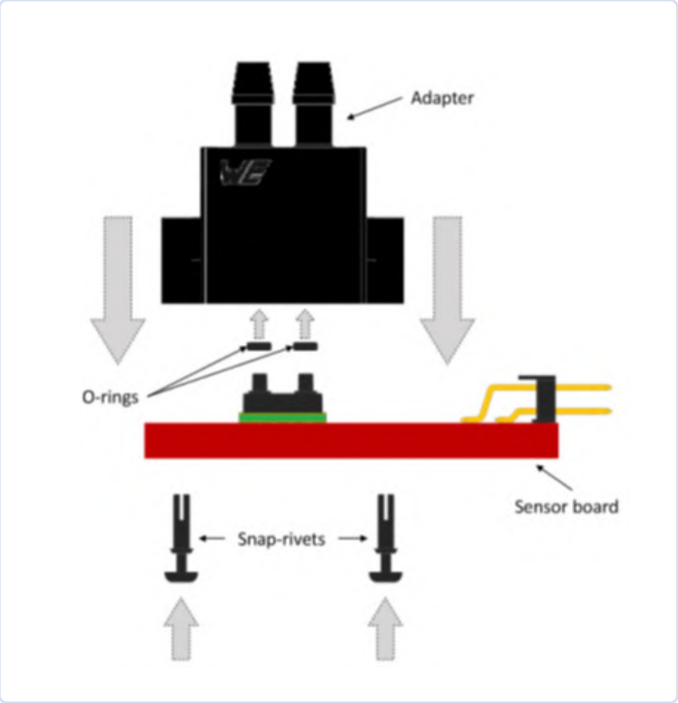


Figure 4: The differential pressure sensor WSEN-PDUS is a very accurate, MEMS-based, piezo-resistive sensor.

complete IoT solution has been created. For that purpose, the Würth Elektronik FeatherWing and Adafruit M0 Express Feather Board have been used along with Würth's LEDs and LED drivers, and multiple other sensors and components (**Figure 2**).

The basic idea was to implement a digital vertical farming system and optimize growth on the one hand, and current consumption and water usage on the other.

Controlled Irrigation and Lighting

A vertical farming system consists of a soil-based vertical farming construction with a 4-channel LED driver and an RGBW LED design kit. This offers an easy solution to mix RGBW color for different lighting situations, amplify the growth of plants with the horticulture panel, or even allow for indoor illumination based on a Human-Centric Lighting concept (HCL). The drop watering system and water tank use recycled water whose pH and EC values are measured, controlled, and plugged into the cloud using different Würth Elektronik connectivity modules.

A soil moisture sensor monitors the soil humidity and, if necessary, the water is pumped into the system using a small pump. The rest of the water is collected, filtered, and returned to a tank. The heart of the system is an Adafruit Feather M0 Express board and the Adafruit power relay. FeatherWings are used as switches. Data is sent to the cloud, where the information is processed, analyzed, and used to control the farm.

Adaptive Lighting System

The lighting system (**Figure 3**) consists of two different parts: a 4-channel Horticulture LED Panel with special plants' monocolor LEDs and the MagI³C multicolor LED driver, both part of the WE Lighting Development Kit [4]. The kit provides an easy solution to amplify the growth of plants. With a MagI³C LED Step Down High Current Module, it is possible to set the intensity and color of each of the four LED strings individually to meet the application needs and set the LED values for the plant profile. The horticultural panel consists of six hyper-red, four far-red, two deep-blue, and four white ceramic LEDs. The system can be controlled using Bluetooth or via the cloud using Wi-Fi or cellular connectivity. This solution shows a more general cloud solution.

Irrigation System

Water tank levels are measured using the WSEN-PDUS differential pressure sensor. It is a very accurate, MEMS-based, piezo-resistive sensor (**Figure 4**). In addition, a 12 V pump pushes the nutrient-rich water into the drip tray via a flow sensor. The water quality is monitored using a Gravity: Analog pH Sensor / Meter Kit and a Gravity: Analog Electrical Conductivity Sensor / Meter from DFRobot.

Most natural waters show a pH value of between 5 and 8. The generally accepted pH value for irrigation water is between 5.5 and 7.5, but the best results have been achieved with water pH values of between 5.5 and 7. Water in this range maintains nutrient balance, provides effective chemical disinfection, and prevents scale formation in irrigation equipment [5].

"Fertigation" is a portmanteau of the words fertilization and irrigation. Due to the low-substrate surface, substrate cultivation requires a lot of watering and fertilizer, which is added to irrigation water. An electrical conductivity (EC) meter is used to detect under- or over-supply of fertilizers in irrigation water. EC is a measure of the concentration of ions in water, and is used to measure water's ability to conduct electric current. The purer the water, the lower its conductivity.

For the soil moisture measurement, the capacitive Adafruit STEMMA Soil Sensor was used. Capacitive measurements use only one probe, they do not have any exposed metal, which can oxidize, and do not introduce any DC currents into plants. The pump is controlled automatically and directly from the cloud. Its operation time

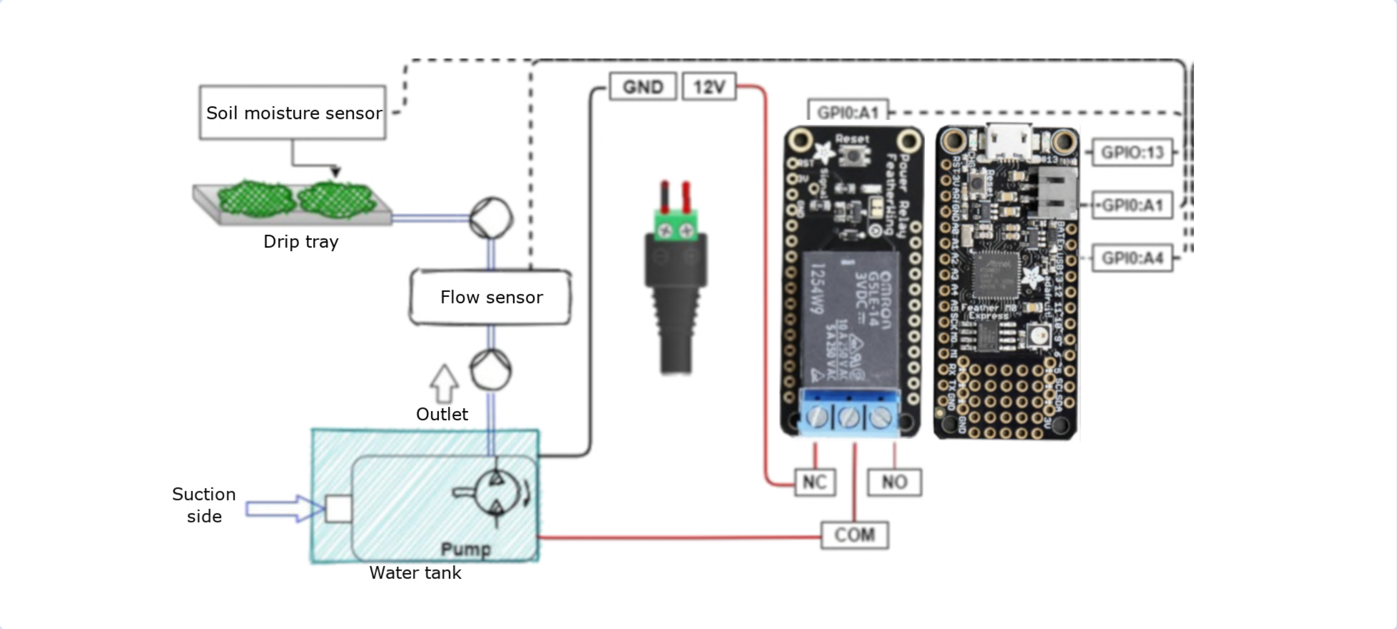


Figure 5: Watering system with sensors, water tank and pump and control boards.

is calculated based on the number of plants, required soil moisture levels and pump flow rate (**Figure 5**).

Environmental Sensing

Sensors measure the state of the environment and interpret the same as analog or digital data. On the other hand, actuators activate a physical change in the measured environment. Advances in the field of electronics in general and semiconductors in particular have led to the availability of a wide range of sensors and actuators that are highly efficient and yet very compact.

Carbon dioxide is utilized extensively in the enrichment and extraction processes. This process not only accelerates plant growth, but can also be used as a fumigant, in most cases. A CO_2 level of 800-1500 ppm is typically used during the enrichment to influence growth at a rate of 20-30%. Increasing overall metabolism helps plants resist the effects of heat. Bigger, healthier, and more robust plants will tolerate environmental extremes better. However, enhanced plant metabolism means extra demands all around. The plants will, apart from needing more water and nutrients, also require extra ventilation.

System CO_2 levels have been measured and monitored online using the Adafruit SPG30 Air Quality Sensor Breakout connected to a Würth Elektronik Sensor FeatherWing. Note that the sensor measures the equivalent CO_2 level. The eCO_2 is calculated based on the H_2 concentration, and is not a 'true' CO_2 sensor for laboratory use. The next step will be to introduce the enrichment.

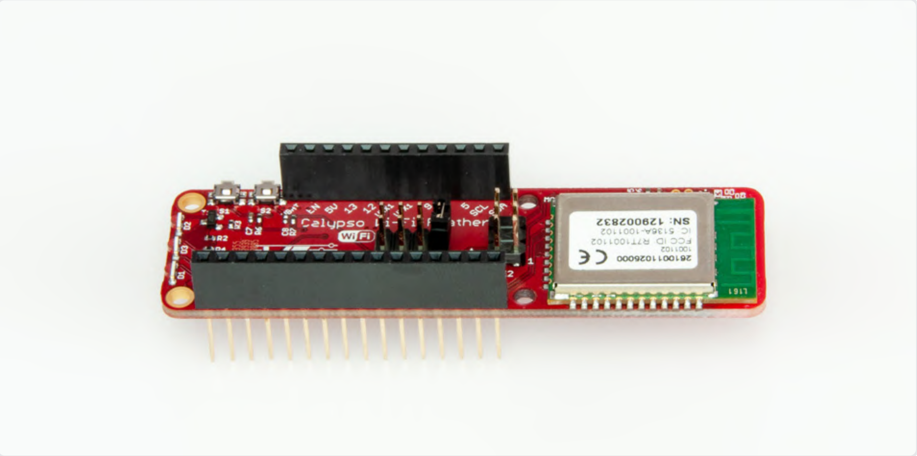


Figure 6: The Calypso FeatherWing provides connectivity in environments with an available Wi-Fi network.

There are multiple plant growth stages and, at each of those stages, a plant needs different lighting and thermal conditions. The majority of the plants can tolerate normal temperature fluctuations. The optimal plant growth thermal conditions can vary not only between the stages, but also during the day. The optimization can be done by measuring the temperature and optimizing daily cycles, and having the temperature a couple of degrees lower when the lights are off.

Atmospheric humidity is expressed as the air moisture percentage. Different plant stages need different humidity levels. Too humid an environment can increase the potential for the spread of diseases. Both humidity and temperature are monitored using the WE Sensor FeatherWing.

IoT Connectivity

Sensors and actuators are typically installed in devices with limited access to the digital world. Wireless connectivity provides — in addition to many other advantages — the reachability necessary for such applications. A wide variety of standards, as well as proprietary wireless connectivity solutions, are available today. Several factors, including range, throughput, spectrum regulation, local statutory requirements, and power budget, determine the choice of a wireless connectivity solution.

Connectivity is implemented using two different approaches: either using a Calypso Wi-Fi FeatherWing (**Figure 6**) for environments with an available Wi-Fi network, or an

Adrastea-I FeatherWing (**Figure 7**) for those without available Wi-Fi. Both are connected to the rest of the system via the Adafruit M0 Express Feather board.

The Calypso is a compact Wi-Fi radio module based on the IEEE 802.11 b/g/n (2.4 GHz) standard. With an on-board TCP/IP stack and MQTT protocol implemented out of the box. The Adrastea-I module is a compact LTE-M/NB-IoT cellular module with integrated GNSS and an ARM Cortex-M4 processor, capable of handling any IoT application.

Both modules can easily and securely be connected to any cloud. In this case, the decision was taken to work with an external M0 processor and Microsoft IoT Central, a platform-as-a-service, because of its simplicity and ease of use. IoT Central has a ready-to-use UI and API built to connect, manage, and operate IoT devices. With its telemetry, properties, and commands, it was used to monitor and control all aspects of the vertical farming system.

AI-Supported Approach

Further AI can be used to detect patterns and create plant profiles in the cloud, which can be used to set the LED values, temperature, humidity, and other parameters automatically to enable optimal conditions.

Vertical farming alone will not feed the world, but it will provide more fresh produce to a higher number of people. With this approach, a miniature version of a vertical-farming system designed for people to have in their kitchens was demonstrated. There was a time when such things were called 'window boxes.' ◀

230077-01

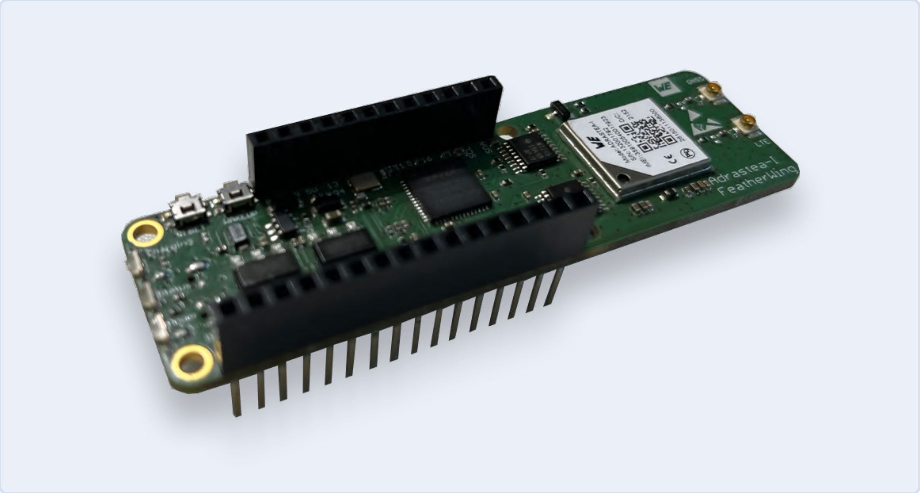


Figure 7: The Adrastea-I module is a compact LTE-M/NB-IoT cellular module with integrated GNSS and an ARM Cortex-M4 processor.

About the Authors

Miroslav Adamov studied Physics and Informatics at the University of Belgrade, Serbia. After that, he continued scientific work at TU-Berlin, WIAS Berlin, FAU Erlangen/Nürnberg and Center of Private Equity Research in Munich. In 2015, after a couple of years in quantitative finance, he joined Würth Elektronik as Senior Business Analyst. He assumed his position as Senior IoT Solution Architect in 2017 with primary focus on conceptualization and implementation of industrial IoT solutions.

Adithya Madanahalli graduated from the Technical University of Munich with an MSc. in Communications Engineering. He then worked as a software engineer for several years, in the field of wireless connectivity and sensors. Since 2022, Adithya has been an IoT Engineer at Würth Elektronik eiSos in the Wireless Connectivity and Sensors business unit. There, he specializes in the design and development of IoT solutions focusing on hardware, embedded software and end-to-end security.

WEB LINKS

[1] "The future of food and agriculture: Trends and challenges," a report by the Food and Agriculture Organization of the United Nations, 2017 [PDF]: https://fao.org/3/a-i6583e.pdf

[2] "Land statistics and indicators – Global, regional and country trends, 2000-2020," Faostat Analytical Brief 48 [PDF]: https://fao.org/3/cc0963en/cc0963en.pdf

[3] E. Ambrose, "Digital agriculture: Why the future is now." : https://ag.purdue.edu/stories/digital-agriculture-why-the-future-is-now/

[4] Würth Elektronik Lighting Development Kit: https://we-online.com/en/components/products/LIGHTING_DEVELOPMENT_KIT

[5] V. Brunton, "Irrigation water quality" [PDF]: https://dpi.nsw.gov.au/__data/assets/pdf_file/0005/433643/Irrigation-water-quality.pdf

Elektor infographic

Embedded and AI Today and Tomorrow

Despite the countless challenges presented by the COVID-19 pandemic and recent geopolitical events, a wide range of exciting business opportunities exist for executives, entrepreneurs, pro engineers, and students focused on both embedded technologies and artificial intelligence-related solutions. Let's look at some facts and figures.

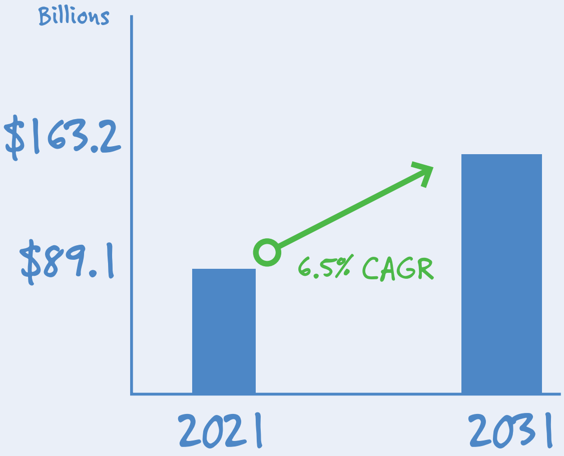
Embedded Systems: Looking Ahead

Wondering about the future of the embedded systems industry? Growth seems to be in the cards. According to a recent Allied Market Research Report, the industry is on track to exceed $163 billion by 2031, with the Asia-Pacific region leading the way. [1]

Companies to Watch

- Advantech Corp.
- Analog Devices
- Infineon Technologies
- Intel Corp.
- Microchip Technology
- NXP Semiconductors
- Qualcomm
- Renesas Electronics
- STMicroelectronics
- Texas Instruments

(Source: Allied Market Research)



Artificial Intelligence in Focus

If the global market intelligence firm IDC is correct, the worldwide artificial intelligence (AI) market — including software, hardware, and services — will post a compound annual growth rate of 18.6% in the 2022-2026 period to hit $900 billion in 2026. [2] Opportunities for companies, professional engineers, entrepreneurs, and even makers abound. Check out these key data points coming from a 2022 report from Stanford University. [3] The growth of AI is pretty clear.

20%

Approximately 1 out of every 5 computer science PhD graduates in 2020 specialized in AI/ML.

$93.5 billion

Private investment in AI hit around $93.5 billion in 2021, which was more than double the amount in 2020.

2x

The number of English-language AI-related publications (e.g., journals, books, thesis papers, conferences) doubled (worldwide) from 162,444 in 2010 to 334,497 in 2021.

North America

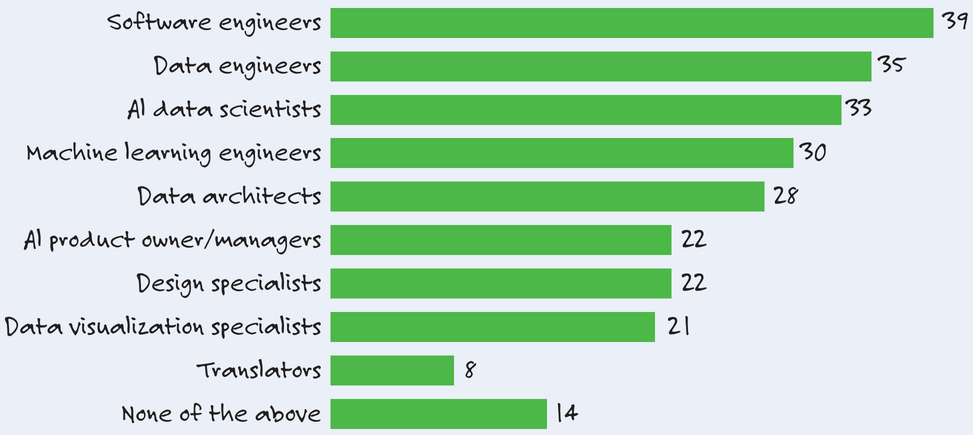
From 2010 to 2021, 56.96% of granted AI patents were filed in North America. 31.09% came from Asia and the Pacific. 11.27% came from Europe and Central Asia.

30x

The number of AI patents filed in 2021 was more than 30 times higher than in 2015. That's a 76.9% annual growth rate.

Where Are the Jobs in AI?

AI-related roles that respondents' organizations hired, past year

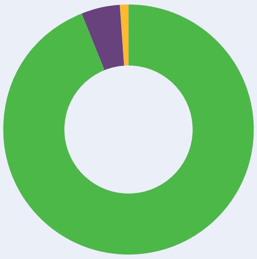


(Source: www.mckinsey.com)

Software engineers have a bright future! According to McKinsey's recent "State of AI" report, software engineers have been leading the pack of AI-focused jobseekers getting hired over the past several months. Data engineers and AI data scientists came in second and third place. [4] This is another clear sign that many organizations have largely shifted from experimenting with AI to actively embedding it in enterprise applications.

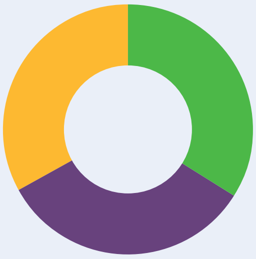
What Do Business Leaders Think of AI?

We all know by now that artificial intelligence = big business. But are you curious about how business leaders are thinking about AI? While interest varies depending on the industry, AI solutions are definitely a topic of discussion among business leaders in conference rooms, at board meetings, and at company outings. The following data from Deloitte offers some insight. [5]



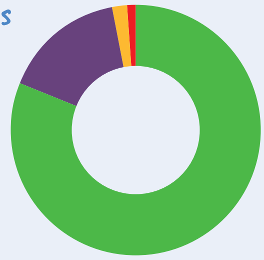
Importance of AI for an org's success

- 🟩 94% Very important or important
- 🟪 5% Somewhat important
- 🟨 1% Not important



What are the top three challenges associated with starting and building projects?

- 🟩 30% Lack of funding for AI tech
- 🟪 29% Selecting the appropriate AI technologies
- 🟨 29% Insufficient technical skills



Will working with AI technology improve job performance and satisfaction?

- 🟩 82% Agree
- 🟪 16% Neither agree nor disagree
- 🟨 2% Disagree
- 🟥 1% Unsure

WEB LINKS

[1] CISION, "Embedded Systems Market to Reach $163.2 Billion by 2031, Globally, by 2031 at 6.5% CAGR," Allied Market Research, 2022: https://bit.ly/embedded-outlook-AMR

[2] IDC, "IDC Forecasts 18.6% Compound Annual Growth for the Artificial Intelligence Market in 2022-2026," 2022: https://bit.ly/IDC-AI-2022-2026

[3] D. Zhang, et al, "The AI Index 2022 Annual Report," HAI, Stanford University, March 2022 https://bit.ly/AI-index-rep-22

[4] McKinsey, "The State of AI in 2022 — And a Half Decade in Review," QuantumBlack, Dec 6, 2022 https://bit.ly/mckinsey-AI-22

[5] Deloitte, "Fueling the AI transformation," October 2022 https://bit.ly/deloitte-ai-state

An Introduction to TinyML

By Mark Patrick (Mouser Electronics)

This article investigates machine-learning concepts using resource-constrained, low-power microcontrollers; collectively termed TinyML. Machine learning continues to make its mark on many aspects of our daily lives, whether at home, in the office, or in-between. While many machine learning applications require significant computational power to crunch complex scientific or financial data, those designed for the Internet of Things (IoT) and other edge-based applications offer only meager compute and connectivity capabilities.

AI and Machine Learning: Already Embedded in Our Lives

Talking to our wristwatches or Star Trek Communicator gadgets were in the minds of science fiction writers not that long ago, yet today many of us routinely do just that. We talk to our smartphone apps, talk to our cars' infotainment systems, and our intelligent 'smart speakers' in our homes.

Artificial Intelligence (AI), the science of creating computers that can think, sense, recognize, and solve problems, has become the foundation of today's computer and data science domains. Machine learning (ML), an application area of AI, focuses on using algorithms to enable computers to learn and improve how they go about a task without being explicitly programmed to achieve it.

Much of the world around us is already shaped by machine learning today, from weather forecasts and finding the best route for your daily commute to the promotional banner advertisements on your favorite social media app. Many areas of scientific research now depend on ML to crunch through petabytes of data to look for for trends.

How Machine Learning Works

Of the machine learning examples highlighted above, we usually only encounter the results, and don't get to see the complexities of finding a new black hole or the number of permutations of past weather events used to determine today's forecast. Many of the complex tasks involve large datasets, multiple algorithm candidates, and significant computing power. Delve further into machine learning, and you'll find different categories of learning, each suiting specific tasks. We won't cover them in-depth here, but they are supervised, unsupervised, semi-supervised, and reinforcement learning.

A neural network is a crucial part of any ML application. Generally speaking, a neural network is an attempt to replicate the function of neurons in the human brain in a mathematical model. The model uses algorithms to infer the probabilities of a result; for example, in an image recognition task, the probability that the picture is a dog may be 95%. There are different types of neural networks, and you'll encounter terms such as convolutional neural networks (CNN) and recurrent neural networks (RNN). Every kind of neural network has a different set of interconnected layers, making it more suited to specific tasks.

Supervised learning involves teaching the neural network lots of training data, so an algorithm can infer a result. For example, for image recognition tasks, a CNN is best. To identify different types of fruit, you need to 'feed' the neural network algorithm with thousands of labelled images of different kinds of fruit taken from different angles and with varying degrees of ripeness. The algorithm then looks for discernible features that help it to identify one type of fruit from another. The training phase is iterative and may involve tweaking the algorithm to achieve the highest probabilities when confronted with a test dataset of images.

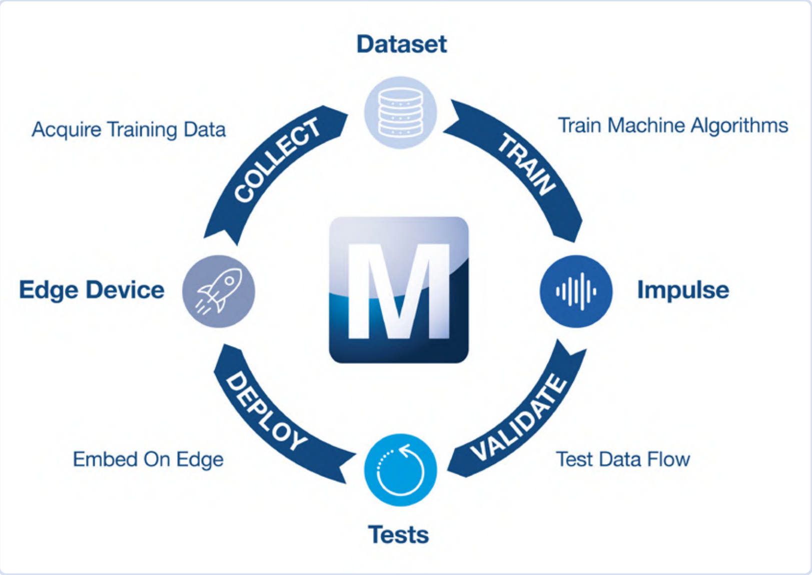
Once the test data achieves the best neural network algorithm performance, the model is ready for deployment. In deployment, also known as inference, the model infers results based on a probability basis.

Our daily interaction with, say, our smart speaker, typically involves a trigger word or phrase such as "Hey Google" to wake it up so that it starts to listen to what comes next. A smart speaker does not have the compute capabilities of a data center, so short audio files are recorded and sent to the cloud for inference in order to determine the nature of our requests. Detecting the trigger word or phrase is an excellent example of simple machine learning; welcome to TinyML!

Machine Learning and the Industrial IoT

As machine learning becomes ubiquitous, the potential list of applications is growing exponentially. Many industrial applications, for example, focus not on 'big data' applications but on how to make production lines more efficient. The costs of an unexpected breakdown in a production process can be high. A motor failure during a chilled food blending process may halt production, resulting in lost raw materials. Many organizations now employ a predictive maintenance regime to schedule planned downtime to prevent such failures. Condition-based monitoring of production assets such as motors and actuators aid in predicting when, for example, a motor bearing starts to show signs of excessive wear. Machine learning algorithms employed in IIoT edge sensor devices can identify when a motor's vibration signature departs from the norm, allowing for sufficient warning.

Within just the industrial domain, the applications for machine learning are huge, but with it come several technical challenges. Unlike the 'Big Data' applications, the compute and memory resources of a simple IIoT edge sensor are a mere fraction of what is available in a data center. A single factory might have hundreds of sensors, so the economy of scale is a factor, as is physical size, availability of a suitable power supply, and wired or wireless connectivity. However, in most cases, the neural network algorithm involved in a vibration sensor is less demanding than deep space research, so embedded developers are exploring ways to run network models on low-power microcontrollers that are battery powered.



▲ *Figure 1: Training and inference steps.*

TinyML: Machine Learning at the Edge

The computing platform used to train a neural network model does not need to be the same as deployment, removing one of the most resource-intensive processes. However, technical challenges are still present. Can the algorithm run in a timely fashion? How will the device communicate to a host system to notify operational staff? Embedded developers are presented with challenges, too. Most are not data scientists, so learning the concepts of machine learning and working with neural networks present a steep learning curve.

230062-01

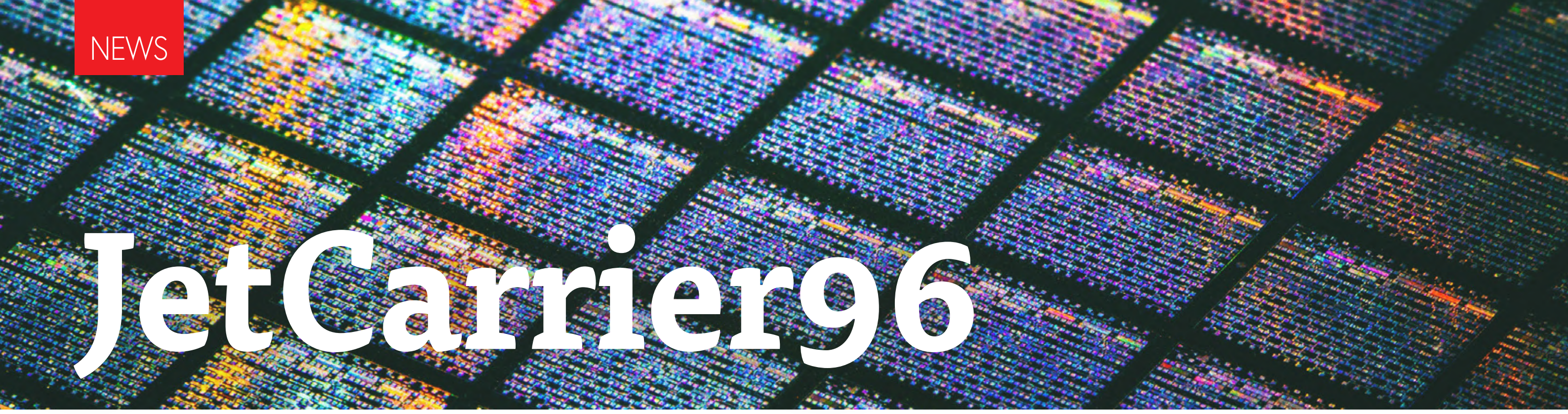
About the Author

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

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

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

NEWS



JetCarrier96

A Versatile NVIDIA Jetson Development System

Global technology solutions provider Arrow Electronics introduced JetCarrier96, a 96Boards Enterprise Edition-compliant carrier board for the popular NVIDIA Jetson family of system-on-modules.

JetCarrier96 enables customers to accelerate custom carrier board designs for applications such as autonomous machines, intelligent vision, diagnostic healthcare imaging, and AIoT.

Arrow is showcasing JetCarrier96 at Embedded World 2023, Hall 3A, Booth 3A-135, Exhibition Center Nuremberg (March 14th-16th).

To support rapid prototyping, the JetCarrier96 carrier board conforms to a standard micro-ATX form factor and will work with readily available power supplies, chassis, and many desktop computer peripherals.

The JetCarrier96 reference design was developed with the latest technologies to speed time to market. JetCarrier96 supports high-speed peripheral expansion via four PCIe card slots, four USB 2.0, four USB 3.2 Gen 2, and a 1 GB Ethernet LAN. JetCarrier96 supports sensor expansion via one 96Boards Mezzanine connector, one Qwiic-compatible connector, one mikroBUS™ socket, and two 15-pin camera connectors.

Featured Supplier Technologies

The JetCarrier96 reference design features

- Analog Devices Silent Switcher high-efficiency, low EMI regulators
- Analog Devices μModule system-in-package power regulators
- Microchip Switchtec PFX Gen 4 Fanout PCIe Switch
- Microchip PolarFire FPGA
- Microchip USB 3.2 Gen 2 SmartHub controller
- Microchip USB 2.0 Hi-Speed Hub controller
- Microchip PIC16F15244 PIC MCU for power supply sequencing
- TE Connectivity — connectivity solutions
- Nexperia — essential semiconductors
- Kemet, a Yageo company — electronic passive components

NVIDIA Jetson Engineering Services

As a preferred member of the NVIDIA Partner Network, eInfochips — an Arrow Electronics company — is the innovator behind the JetCarrier96 reference design.

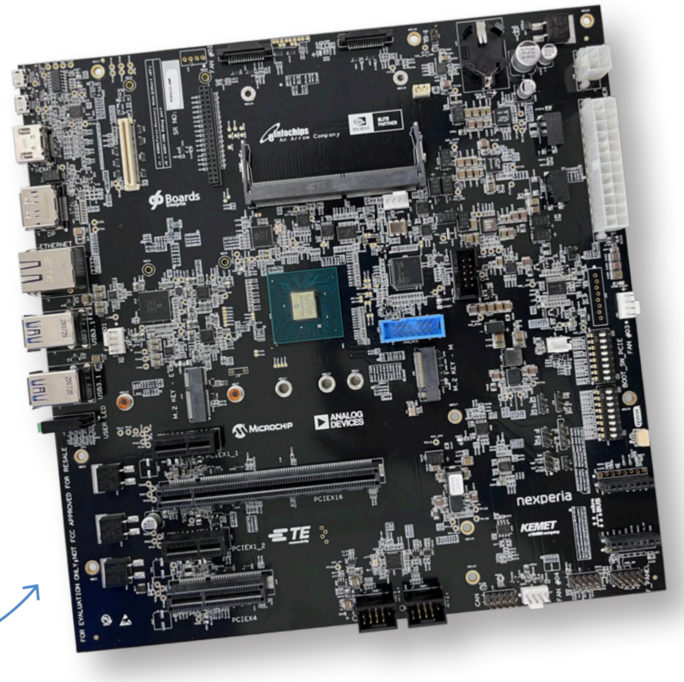
"The megatrend of AI at the edge continues to proliferate and, with it, the engineering challenges placed on customers to bring an edge AI device to market," said Aiden Mitchell, senior vice president, global marketing and engineering, Arrow Electronics. "JetCarrier96 helps to fulfill our mission of delivering innovative reference platforms to customers augmented with eInfochips edge-to-cloud AI product engineering services."

The design objectives of the JetCarrier96 project are twofold. First, to create a versatile development system suitable for proof-of-concept work across a wide range of applications. To meet this objective, JetCarrier96 is specified with ample expansion interfaces of all types. In practice, this means to implement as many NVIDIA Jetson module features as reasonably possible.

Second, to make design documentation and bill-of-materials available as a resource to guide users implementing similar features into custom carrier boards of the developers' own design. It is unlikely that a custom carrier board would require the entire feature set found on JetCarrier96. Therefore, where possible, the JetCarrier96 design team made choices that emphasize modularity over minimizing bill of materials.

JetCarrier96 is a versatile proof-of-concept development system and an NVIDIA Jetson carrier board reference design featuring ample expansion for the professional embedded developer.

230061-01



More info about the Board: https://arrow.com/jetcarrier96

Case Study: Taking **EV Charging Global** with a **Universal RFID Solution**

Contributed by ELATEC

Radio-frequency identification (RFID) enables fast, easy user authentication and access control for EV charging. But, with more than 60 RFID transponder technologies in use globally — along with smartphone-based credentialing systems based on Bluetooth® Low Energy (BLE) or Near-Field Communication (NFC) — it can be challenging for OEMs to serve all markets. AUTEL, a leading provider of DC and AC charging station infrastructure, maximized their market opportunities and simplified inventory management with a universal RFID reader.

RFID for EV Charging: How It Works

The RFID reader allows users to identify themselves at the charging station and gain access to charging services.

- The reader is embedded into the charging infrastructure.
- Users are issued ID cards with an embedded RFID antenna and an integrated circuit that stores a unique data set (e.g., a number) for user identification. Alternatively, they may use a smartphone app that acts as a card emulator.
- When the card or smartphone is held up to the reader, the reader transmits a radio signal to interrogate the tag or app, prompting it to send the unique identifier to the reader.
- The reader connects to backend software to control access levels, connect users to their accounts for payment information or loyalty programs, and track user behavior.

Many Markets, Many Technologies

There are more than 60 RFID transponder technologies in use globally, which are broadly categorized as high-frequency (HF) or low-frequency (LF).

- HF RFID operates in the range of 3 to 300 MHz, with 13.56 MHz being the standard. HF has a good read range (≤30 cm) and excellent data transfer speeds and memory capacity, making it ideal for applications such as EV charging.
- LF RFID operates in the range of 30 to 300 kHz (125 kHz standard) with a read range of ≤10 cm. LF technologies are less likely to be used for EV charging but may still be encountered in some regions.
- Smartphone credentialing systems can use either BLE or NFC. NFC operates at the same frequency as HF RFID (13.56 MHz).

Within these broad categories, there are dozens of different technologies in use, including MIFARE, HID iCLASS, DESFire, and many others. Most RFID readers are able to read only a handful of these.

To reach all their markets, AUTEL had to stock several different versions of their product with readers from different manufacturers. This created significant challenges for inventory control and post-sales support. It also limited their ability to help customers with a diverse user base and prepare for future requirements.

A Universal Reader for Global EV Charging

AUTEL found their answer in the TWN4 MultiTech reader from ELATEC. This universal RFID reader supports 60+ transponder technologies, covering all major RFID standards worldwide for frequency ranges 125 kHz and 134.2 kHz, as well as the 13.56 MHz band, including NFC for mobile (BLE upgrade available). It is also certified for use in 110+ countries worldwide.

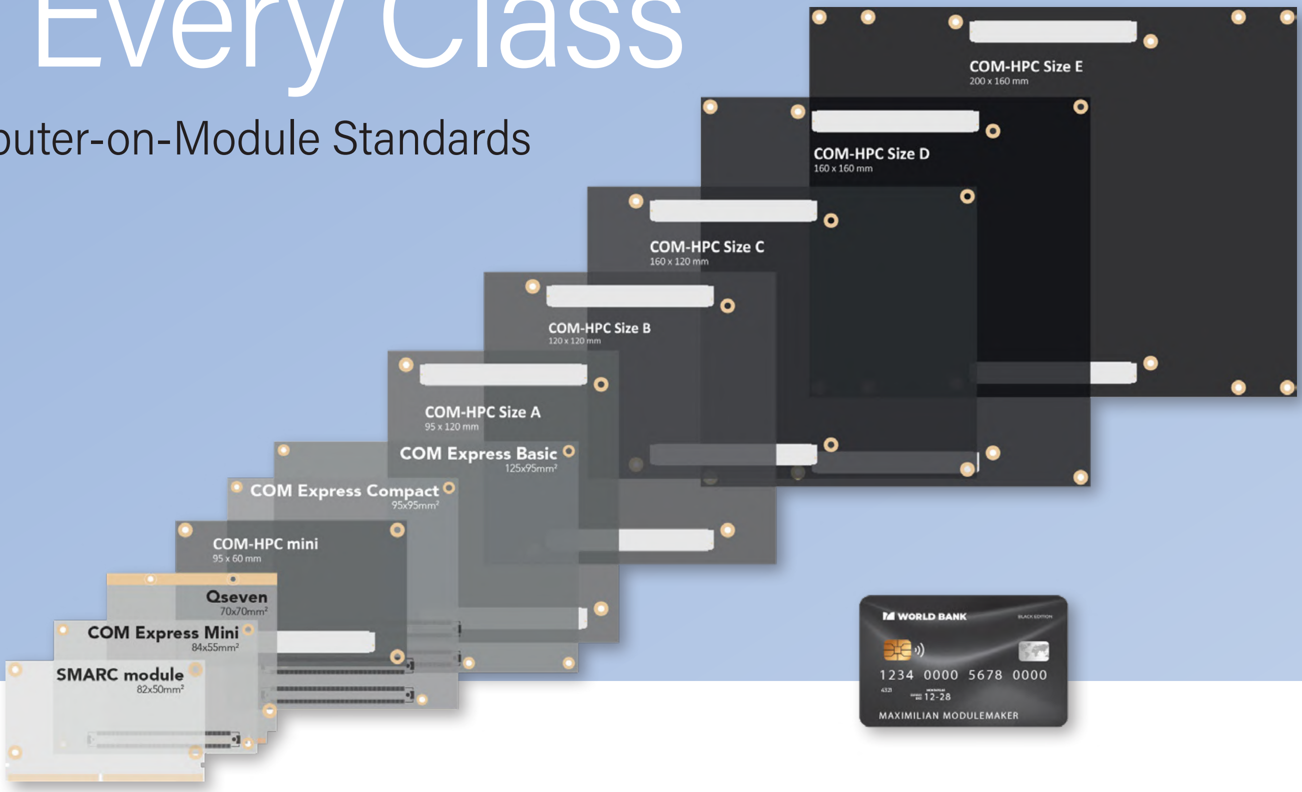
The reader is easy to update via contactless config cards or centralized remote updates. This gives AUTEL — and their customers — the flexibility to add new technologies after installation and update software and firmware to respond to changing security requirements or add new functionality.

With a universal reader, AUTEL has simplified inventory management and after-sales support while maximizing their global market opportunities.

230056-01

High-Performance in Every Class

Computer-on-Module Standards



▲ *Figure 1: Manufacturer-independent computer-on-module standards' different module sizes.*

Contributed by Congatec

A lot is happening in the Computer-on-Module market. With COM Express 3.1, a new version for the most successful Computer-on-Module standard is now available. And COM-HPC, the high-performance embedded computing standard, is also raising many new questions. So, what do OEMs and system designers need to know?

According to market figures from IHS Markit, Computer-on-Modules are the most widely employed embedded design principle — even ahead of classic embedded boards such as Mini-ITX or 3.5-inch single board computers (SBCs). The great popularity of such embedded system designs stems from their successful combination of flexible, customer-specific carrier board design with ready-to-use, easy to integrate modules that include all necessary drivers and firmware. These super components include all key building blocks such as CPU, RAM, high-speed interfaces, and often also the graphics unit in a single, function-validated package. Another advantage is the fact that Computer-on-Modules of the same standard are freely interchangeable, both across processor generations and between manufacturers. This gives OEMs full flexibility when scaling and upgrading their solutions with the latest processor technology even after several years. It also makes it easy to implement multi-vendor strategies, which offers pricing advantages and, above all, guarantees availability. There are two independent committees looking after module standardization: The PICMG, which is hosted in America, and the German SGET, currently maintain a total of four Computer-on-Module standards, most of them providing numerous variants. These standards are COM-HPC and COM Express for the high-end segment, and SMARC and Qseven for the low-power segment.

Figure 2: Congatec offers a comprehensive COM-HPC portfolio from COM-HPC Server and Client to COM-HPC Mini.

COM-HPC: the High-Performance Benchmark

COM-HPC, the most recent Computer-on-Module standard from the PICMG, is aimed at high-performance embedded computer designs, which could not be achieved with previous standards. Its performance ranks above the world's leading COM Express standard. COM-HPC targets new high-speed interfaces such as PCI Express 4.0 and 5.0, as well as 25 Gb Ethernet. It offers two different module versions for this purpose, which were developed by the COM-HPC Subcommittee under its Chairman, Christian Eder, from Congatec: COM-HPC Server and COM-HPC Client. Their main technical differences lie in the footprints, the number and type of supported interfaces, and the memory capacity. Brand new is the COM-HPC Mini specification, which now brings the advantages to a credit-card footprint as well.

COM-HPC Server modules

COM-HPC Server defines the ultra-high-end of embedded computing, addressing the new edge and fog servers in harsh environments that must manage more and more massive workloads. For this purpose, COM-HPC Server specifies two footprint sizes with up to 64 PCIe lanes and up to 256 GB/s, as well as up to 8x Ethernet with 25 Gb/s each. COM-HPC Server is not limited to x86 technology, allowing the use of RISC processors, FPGAs, and GPGUs — which adds new modularization perspectives. In order to meet the requirements of server applications, the modules also offer master-slave modes and remote management. Leveraging the instruction set of the powerful IPMI standard, this makes established server technology also available to Server-on-Modules. With COM-HPC Server modules offering a power budget of up to 300 watts, this standard is suitable for the development of ultra-high-performance embedded edge and fog servers. Today's most powerful COM Express Type 7 Server-on-Modules, by comparison, support a maximum of 100 watts. Another significant difference lies in the number of signal pins: The COM Express connector has 440 pins, while COM-HPC offers almost twice as many with 800.

COM-HPC Client modules

COM-HPC Client modules are designed for high-performance embedded systems with integrated graphics. They provide four graphics outputs via three Digital Display Interfaces (DDI) and 1× embedded DisplayPort (eDP). They host up to four SO-DIMM sockets for up to 128 GB of RAM. For the connection of peripherals, 48 PCIe lanes and 2× USB 4.0 are available; embedded camera modules can also be connected directly via two MIPI-CSI interfaces. COM-HPC Client modules will be available in three different sizes: 120 mm × 160 mm (Size C), 120 mm × 120 mm (Size B) and 120 mm × 95 mm (Size A).

COM-HPC Mini modules

COM-HPC Mini is a new upcoming Computer-on-Module standard currently in development at PICMG. Currently, the technical subcommittee has approved the pinout and footprint of the new credit-card-sized (95 × 60 mm) high-performance Computer-on-Module specification. Providing 400 pins, the new COM-HPC Mini standard is designed to satisfy the rising interface needs of heterogeneous and multi-functional edge computers. Extensions include up to 4× USB 4.0 with full functionality including Thunderbolt and DisplayPort alternate mode, PCIe Gen 4/5 with up to 16 lanes, 2× 10 Gb/s Ethernet port, and much more. Add to that the fact that the COM-HPC Mini connector is qualified for bandwidths of more than 32 Gb/s — enough to support PCIe Gen 5 or even Gen 6 — it is clear that its capabilities go well beyond those of all other credit-card-sized module standards.

COM Express: The Most Successful Module Standard Worldwide

This means the smallest of the COM-HPC Client footprints is almost the same size as COM Express Basic at 125 mm × 95 mm. This illustrates that

Figure 3: Latest processor technology such the high-end 13th Gen Intel Core processors in BGA assembly deliver Congatec on COM-HPC and COM Express modules alike.

COM-HPC Client sits well above COM Express and targets applications that cannot be addressed with COM Express. COM Express was launched in 2005, and, of the Computer-on-Module standards presented here, it has been available the longest. The specification defines a family of different module sizes and pin types. Unlike COM-HPC and small form factor (SFF) module specifications Qseven and SMARC, COM Express focuses solely on x86 processor technology. With the ratification of the 3.1 specification, COM Express now supports the latest high-speed interfaces, such as PCIe 4.0 and USB 4. Despite the improvements, COM Express 3.1 Type 6 modules are fully backward-compatible with 3.0 modules and carrier boards, ensuring that even older designs can be equipped with the latest processor technologies.

COM Express Type 7 Server-on-Modules

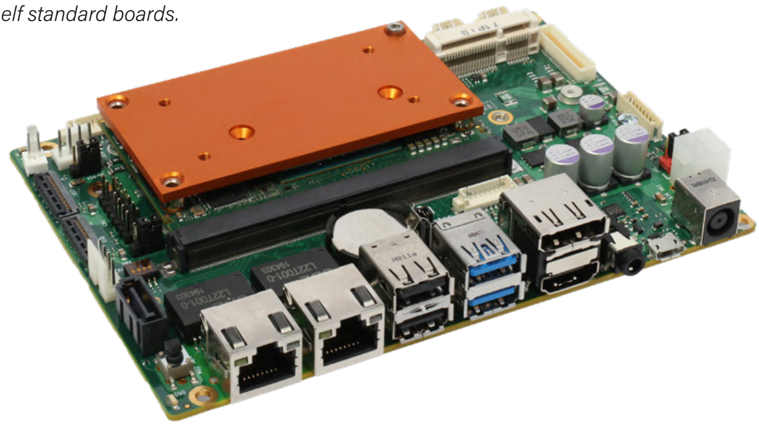
Like COM-HPC, COM Express also offers both server and client modules, which are mainly available in the familiar Type 6 (client) and Type 7 (server) pinouts. As with COM-HPC, the Type 7 server pinout is a headless Server-on-Module without graphics outputs and is likewise designed for embedded edge and fog servers. Notably, it supports up to 4×10 GbE and up to 32× high-speed PCIe Gen 3.0 lanes for interfaces and storage media. These Server-on-Modules are available with Intel® Xeon® D processors or AMD EPYC Embedded 3000 processors. For them, Congatec also offers a 100-watt ecosystem with application-ready cooling solutions to simplify the design-in of the most powerful COM Express Server-on-Modules.

COM Express Type 6 Computer-on-Modules

For classic embedded applications with graphics, PICMG COM Express Type 6 modules are the ideal choice. They feature embedded processors ranging from Intel® Core™, Pentium® and Celeron® to the AMD Embedded R-Series. Available in footprints of 95 mm × 125 mm (Basic) or 95 mm × 95 mm (Compact), they provide 440 pins to the carrier board for a wide range of modern computer interfaces. Supporting up to four independent displays, 24 PCIe lanes, USB 2.0 and USB 3.0 as well as Ethernet, CAN bus, and serial interfaces, they offer everything needed to build powerful PLCs, HMIs, shop-floor systems or SCADA workstations in control rooms. Other application areas include high-end digital signage systems and high-performance medical equipment for diagnostic imaging. With the spec update to 3.1 COM Express Type 6 modules now can optionally provide MIPI-CSI connectors on the module.

Figure 4: SMARC 2.1 carrier board and scalable 3.5-inch single board computer: The conga-SMC1 bridges the gap between module-based designs and highly scalable commercial-off-the-shelf standard boards.

▼



COM Express Type 10 Mini modules

The specification's smallest form factor — COM Express Mini, measuring 55 mm × 84 mm — is addressed by the PICMG Type 10 pinout and completes the set of COM Express specifications for SFF designs. These modules are designed for low-power Intel® Atom™ and Celeron® processors. Here, too, the modules can execute two MIPI CSI connectors with COM Express 3.1. As the same connector technology and design guides are leveraged across the entire PICMG COM Express ecosystem, developers are able to reuse a significant number of functions, which is the main advantage of this mini specification. However, the SGET standards, SMARC and Qseven, are more widely established and used; both of these standards support both x86 and ARM application processors.

SMARC for Embedded Vision

SMARC addresses the high end of SFF applications. This standard has received a major update with revision 2.1. The new revision adds numerous new features, such as SerDes support for extended edge connectivity and two extra interfaces on the module that can be used to connect a total of 4 MIPI-CSI cameras to meet the growing demand for a fusion of embedded computing and embedded vision. The new features are backward-compatible with Rev. 2.0 and all extensions to Rev. 2.0 are optional, so all Congatec SMARC 2.0 modules are automatically compatible with SMARC 2.1.

Another distinguishing feature, besides the two MIPI interfaces on the connector, is that SMARC supports wireless interfaces such as WLAN and Bluetooth directly on the module. Ideal processors for SMARC modules are the latest Intel Atom processors or the entire range of new i.MX 8 application processors.

230071-01

Starting Out in Electronics

Let's Get Active!

By Eric Bogers (Elektor)

As we already mentioned in a previous episode of this series, all components that contain a semiconductor die (such as diodes) are called active components. Actually that's not right if you insist on a very strict definition of active, in which case only components that amplify a current or voltage are truly active. And now – with transistors – we are entering the domain of active components.

The word transistor is a coined term composed from the words 'transfer' and 'resistor'. This means we could playfully call a transistor a 'handover resistor', but fortunately nobody does that. However, you can actually vary the resistance of this component by applying a control voltage. Transistors are what made the current electronics age possible because they are much smaller and less fragile than tubes, and because they consume significantly less power. Transistors are also what made integrated circuits (ICs) possible. ICs contain a very large number of components (active and passive) located on a die or chip. With modern microprocessors, we are talking about many million transistors contained in a single package.

But let's start with transistors as loose components. We'll leave integrated circuits to a later episode. Nowadays, there are ready-made solutions for many applications in the form of ICs, but you often have to dig into your parts' drawer for 'loose' transistors — for example, when you need output transistors for a power amplifier.

Figure 1: Several transistors (source: Shutterstock / Edkida).

Several transistors are shown in **Figure 1**. As a rule of thumb, the amount of power a transistor can handle depends on how big it is and whether it has a metallic package.

Using a Bipolar Transistor as a Switch

Usually, when we talk about a transistor, we mean a bipolar transistor. Two types are available: npn and pnp, where the letters stand for the sequence of the n-doped or p-doped silicon layers of the transistor structure.

Basically (but very much oversimplified), transistors can be used as switches or as amplifiers. First, we'll look at using a transistor as a switch, since it's easier to understand how that works. Afterwards, we will deal with the properties of transistors as amplifiers.

Figure 2 shows an npn transistor on the left and a pnp transistor on the right, in each case represented by a combination of two diodes.

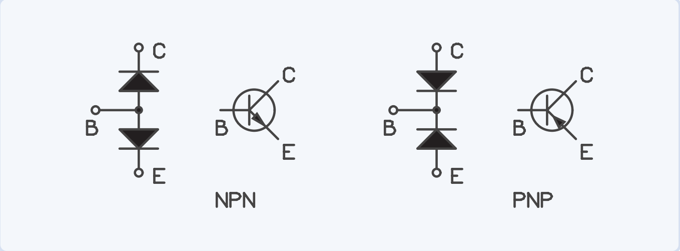


Figure 2: Equivalent circuits of transistors. C = collector, B = base, E = emitter.

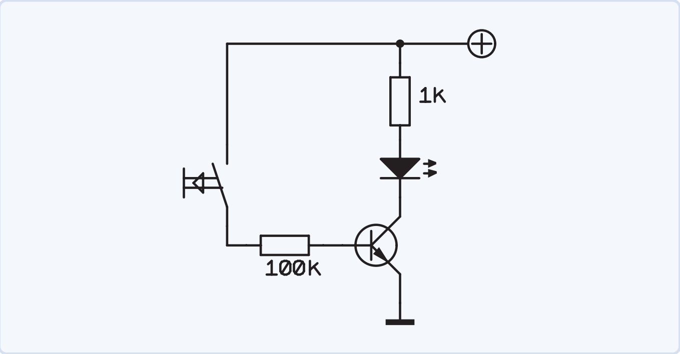


Figure 3: A transistor used as a switch.

In practice a transistor is never used as a simple replacement for two diodes, but when you use a continuity tester to check whether a transistor is an npn type or a pnp type (or whether it still works or is shorted out), it's handy to keep these equivalent circuits in mind.

Figure 3 shows a transistor being used as a switch. By the way, we always use npn transistors in the examples in this episode because they are easier to understand when you have conventional current flow (from positive to negative) in mind. However, each of these examples can easily be converted to a pnp type by simply changing the polarization of all polarized components (diodes, electrolytic capacitors, and of course, the supply voltage).

Let's go back to Figure 3. Here there are two circuits: one for the control current and one for the load. The control current flows through the switch, the series resistor and the base-emitter diode of the transistor. This base-emitter junction is what controls the transistor. When a current flows through this junction, the transistor conducts, and when no current flows through this junction, the transistor is cut off.

The base-emitter junction behaves the same way as an ordinary diode, which means that a current only starts flowing when the voltage on the junction reaches a level of approximately 0.7 V.

The other circuit consists of the series resistor of the LED, the LED, and the collector-emitter junction of the transistor. You might think that no current will flow there because the collector-base diode is reverse biased. In practice, the current does not flow through the collector-base diode, but instead directly from the collector to the emitter, so there isn't any 0.7 V voltage over the collector-emitter junction.

This works because of the special way the different semiconductor layers are arranged; after all, it's not possible to make a transistor from a pair of loose diodes.

You may be wondering (with good reason) why we are using a transistor and an additional resistor in the circuit of Figure 3 instead of simply operating the LED directly with the switch. And you're completely right: that's entirely unnecessary in this specific example.

The current through the switch in this circuit is a factor of 100 less than the current through the LED, although it would be very difficult to find a switch that can't handle the current through the LED. The circuit shown here, however, is also suitable for switching much higher currents (and voltages), as there's no reason why the control circuit and the load circuit have to be powered from the same voltage source. Put simply, the transistor makes it possible to switch a much higher voltage with a control voltage of just a few volts, provided that you use a suitable transistor.

The Bistable Multivibrator (Flip-Flop)

Now let's look at a few examples of multivibrators, which means circuits that switch between two defined states. **Figure 4** shows a bistable multivibrator, commonly known as a flip-flop.

This circuit is called bistable because it has two stable states — either the LED on the left is lit or the LED on the right is lit. The circuit will never change state unless you press one of the two pushbuttons.

Let's suppose the left-hand transistor is conducting, which means that the LED on the left is lit. The voltage between the collector and emitter of this transistor is therefore low (approximately 0.1 V), which is not enough to switch on the right-hand transistor (at least 0.7 V is necessary for that). The right-hand transistor is therefore cut off and its collector-emitter voltage is nearly the same as the supply voltage U_V. This means that the left-hand transistor always remains switched on by the current through the 100 kΩ resistor on the right-hand side. This is therefore a stable state.

However, if you press the left-hand button, the base-emitter voltage of the left-hand transistor drops to zero. As a result, this transistor switches off and its collector-emitter voltage rises to approximately U_V. The left-hand LED goes dark, and the right-hand transistor is driven into conduction by the current through the 100 kΩ resistor

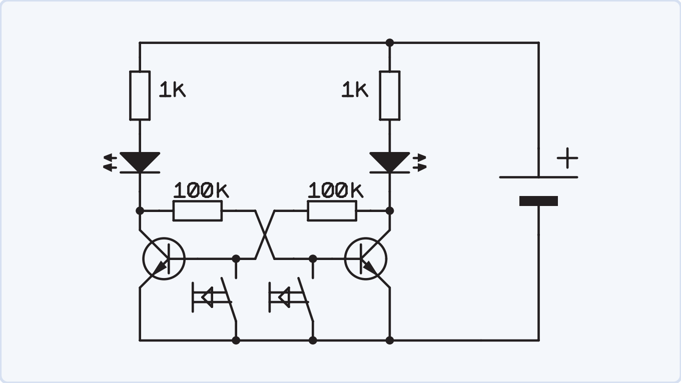


Figure 4: A bistable multivibrator.

on the right-hand side. Its connector-emitter voltage drops to nearly 0 V, and the right-hand LED lights up. Now, the left-hand transistor can no longer be switched on, even when you release the pushbutton. This is again a stable state, but the opposite of the initial state.

The circuit will remain in this stable state until someone presses the right-hand button, causing the circuit to switch back to the other stable state.

Now the only question that needs to be answered is which transistor switches on first after the supply voltage is switched on. That is impossible to predict from the schematic diagram. If both halves of the circuit were to be built with identical components, it would be purely a matter of chance. In other words, it would depend on the unpredictable component noise. In the real world, however, there aren't any exactly identical components. The resistors have manufacturing tolerances, and the current gains of the transistors are not the same (and besides, the current gain depends on the temperature, and therefore on which transistor was recently conducting and is probably warmer than the other transistor).

It's better to turn the question around: how can you ensure that the same transistor always switches on first when the supply voltage is switched on? The answer is simple: reduce the resistance of the base resistor of the transistor concerned to half the value stated on the schematic diagram.

That's it for this episode. In the next episode, we will examine two other multivibrator circuits, and then look at using transistors as amplifiers.

Briefly Revisiting the Diac

In last year's November/December edition [1] we briefly mentioned the diac as a type of diode (now more or less obsolete). In that article, we wrote:

'From your favorite electronics-by-mail store (sadly the "electronics shop around the corner" has vanished) you can buy a component with the simple yet exotic name of "diac" which contains two Zener diodes connected in this way.'

Two Elektor readers (Pablo Medina and Maurizio Spagni), independently of each other, informed us that this is actually not correct. And both of them are absolutely right: a diac is a rather complex component with a negative resistance characteristic, making it suitable for triggering thyristors and triacs [2].

Comparing a diac to two Zener diodes connected in reverse series is therefore a gross oversimplification. Nevertheless, we think that novice electronic hobbyists can better understand how a diac works by regarding it as two Zener diodes instead of a negative resistance. By the time they have understood what is meant by a negative resistance characteristic and how a semiconductor device can have this sort of characteristic, they will have probably thrown their soldering iron out of the window from frustration.

But as we already mentioned, both letter writers are absolutely right, and in future we will choose our words more carefully.

220670-01

Translated to English by Kenneth Cox.

Editor's Note: The 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 by email at editor@elektor.com.

WEB LINKS

[1] Eric Bogers, Michael Ebner: "Starting Out in Electronics... Cheerfully Continues Zenering," ElektorMag 11-12/2022: https://www.elektormagazine.com/220384-01

[2] Diac entry in Wikipedia: https://en.wikipedia.org/wiki/DIAC

I²C Communication Using Node.js and a Raspberry Pi

See Your Sensor Data in a Browser

By Franck Bigrat (France)

The I²C bus has been around since the 80s, and is well-supported on today's popular platforms, including the Raspberry Pi series. Node.js makes it possible to program a web server in JavaScript, serving data to a local network or the internet. This article shows you how easy it is to hook up an I²C sensor to a Raspberry Pi and display the results in a web browser.

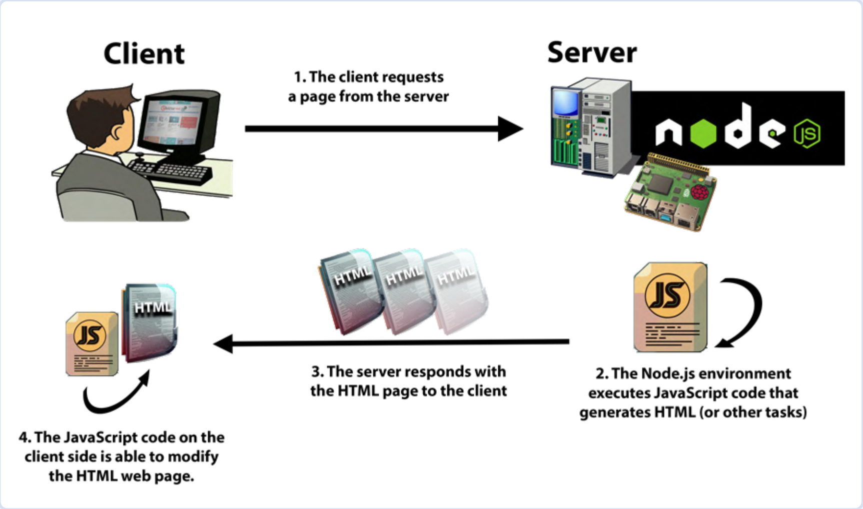

Client
Server
1. The client requests a page from the server
node JS
HTML
JS
3. The server responds with the HTML page to the client
2. The Node.js environment executes JavaScript code that generates HTML (or other tasks)
4. The JavaScript code on the client side is able to modify the HTML web page.

Figure 1: Node.js server communication flow. (Source: sdz.tdct.org)

In this project, we use a simple Raspberry Pi Zero and the DS1621 I²C temperature sensor. The measurement result is displayed on a web page and is visible on a tablet or smartphone connected to the same Wi-Fi network.

Node.js has modules such as *http* and *express*, which allow the programmer to create web servers very simply, so we don't need the ubiquitous Apache server.

Although there is a module designed to manage the DS1621 sensor, it is more interesting to use the Node.js *i2c-bus* module, which enables us to extend the project to a wide variety of other I²C devices.

What is Node.js?

Explaining Node.js in a few words is not an easy task.

Many books and websites are dedicated to Node.js, so simply understand that Node.js is *JavaScript executed by the web server, so it's not executed by a web browser running on a client* (however, the browser can, of course, also execute JavaScript code if need be). It is built on Google Chrome's JavaScript engine (V8). Simply put, the programmer writes JavaScript applications, but these run on the server. The overview is shown in **Figure 1**.

To learn (almost) everything about Node.js, I strongly advise the reader to visit the websites at [1] and [2].

The reader may also refer to these books:

- *Node.js in Practice* by Alex Young and Marc Harter (Manning)
- *Beginning Node.js* by Basarat Ali Sayed (Apress)
- *The Node Beginner Book* by Manuel Kiessling
- *Instant Node.js Starter* by Pedro Teixeira (Packt Publishing)

I²C and SMBus

In this section, I will assume that the reader is somewhat familiar with the I²C (Inter-Integrated Circuit) protocol. Most important to know is that I²C was developed by Philips during the 80s to minimize the number of wires between a microprocessor and several other integrated circuits.

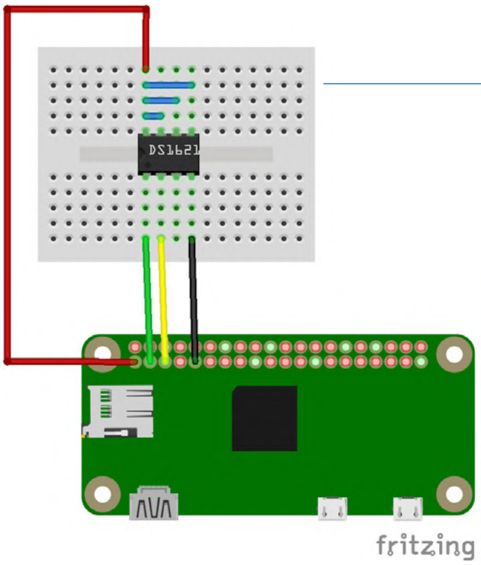


Figure 2: Raspberry Pi Zero W to DS1621 hookup diagram.

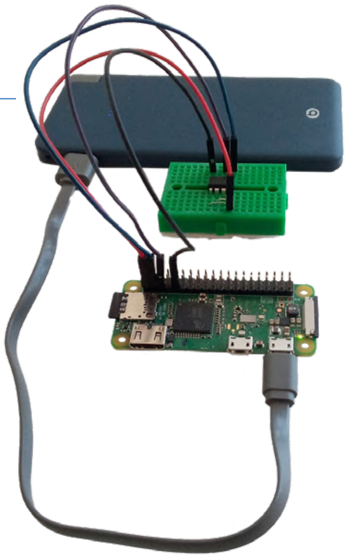


Figure 3: The circuit wired up, complete with battery pack.

It's a serial, two-wire (Data and Clock lines) synchronous data transmission protocol. Several circuits can be connected to the same data and clock lines simultaneously.

Details of the *i2c-bus* Node.js module's functions can be found at [3].

If you read carefully, you can see that some functions have an *smbus* prefix or are enclosed in the SMBus section. SMBus allows communication between the processor of a computer and different peripherals, such as energy-management devices, temperature sensors, or fans.

The Raspberry Pi supports this protocol in hardware: The System Management Bus is derived from the I²C bus, and the two protocols are quite compatible. But, some differences exist between SMBus and I²C; these are explained in the Texas Instruments document at [4].

Connecting and Configuring the Components

Well, it's time to hook the Raspberry Pi up to the DS1621 temperature sensor according to the layout in **Figure 2**.

Don't forget to connect the DS1621's Pins 5, 6, and 7 (A0, A1, A2) to V_{CC} to set the I²C address of the device to 4F (hexadecimal). The wired-up circuit is pictured in **Figure 3**.

Configuring the Raspberry Pi

1. Download and install Raspberry Pi Imager from: *https://raspberrypi.com/software*

2. Launch Raspberry Pi Imager (**Figure 4a**), click on *CHOOSE OS* and then click on *Raspberry Pi OS (Other)* from the list that pops up (**Figure 4b**), and then *Raspberry Pi OS Lite (32-bit)* from the submenu.

3. Click on *CHOOSE STORAGE* and then select your microSD card from the *Storage* dialog (**Figure 4c**).

4. Click on the Settings icon (cog) on the right side of the application and set the Raspberry Pi initial configuration options:

- Check *Set hostname:* and then enter the hostname you want; *raspberry* will do fine

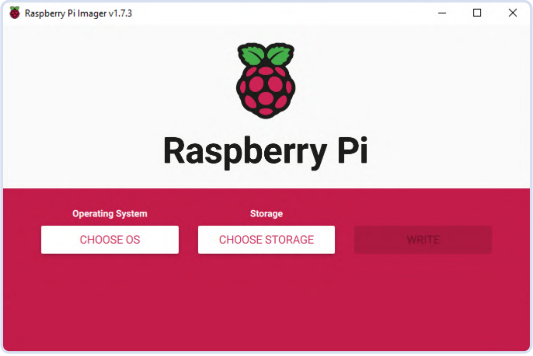
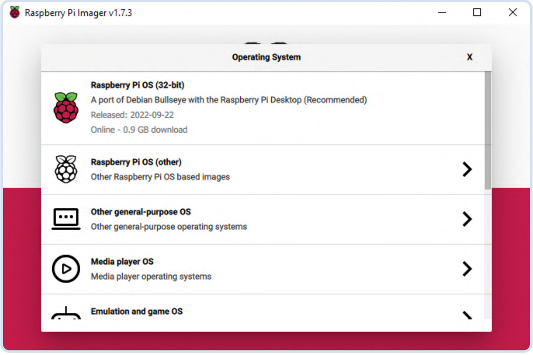


Figure 4a: Raspberry Pi Imager.



Raspberry Pi OS (32-bit)
A port of Debian Bullseye with the Raspberry Pi Desktop (Recommended)
Released: 2022-09-22
Online - 0.9 GB download

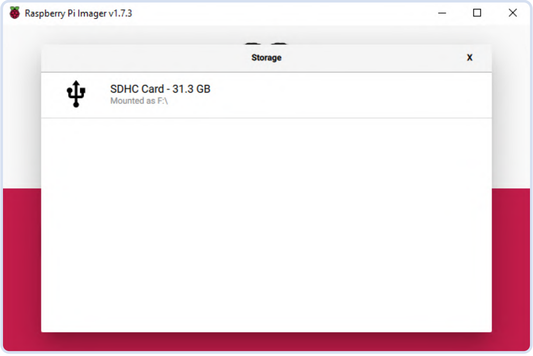
Raspberry Pi OS (other)
Other Raspberry Pi OS based images

Other general-purpose OS
Other general-purpose operating systems

Media player OS
Media player operating systems

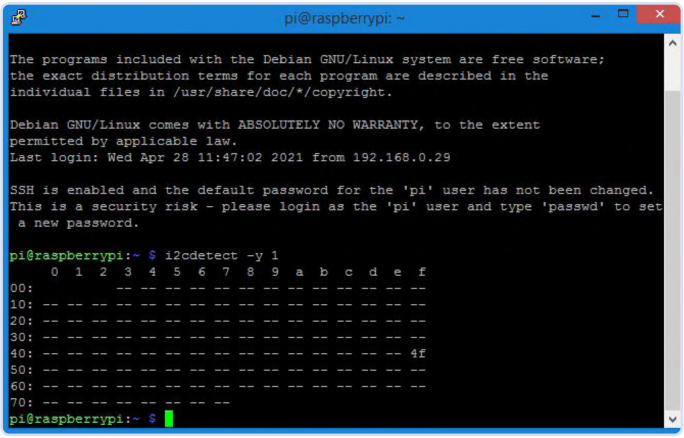
Emulation and game OS

Figure 4b: Selecting the OS.



SDHC Card - 31.3 GB
Mounted as F:\

Figure 4c: Selecting the storage medium.



```
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Apr 28 11:47:02 2021 from 192.168.0.29

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
 a new password.

pi@raspberrypi:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- 4f
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi:~ $
```

Figure 5: Output of the ***i2cdetect*** *program.*

- Check *Enable SSH* and select the *Use password authentication* radio button
- Check *Set username and password*, and enter the *Username* (*pi* is fine) and a *Password* (choose one easy to remember, e.g. *raspberry*)
- Check *Configure wireless LAN* and enter your Wi-Fi network's SSID and password
- Check *Set locale settings* and choose your *Time zone* and *Keyboard layout* as needed, or leave the defaults as they are.
- Click on the *SAVE* button.

5. Install the OS by clicking on *WRITE*, then confirm with *YES*.

6. When it's done writing, remove the microSD card.

7. Insert the card into the Raspberry Pi's microSD slot and power up the Pi.

8. Find the IP address of the Raspberry Pi on the network either using a tool such as Advanced IP Scanner, or, even better, get it from the DHCP lease settings in your Wi-Fi router.

9. Use an SSH client such as Putty to connect to the Raspberry Pi using its IP address and username/password (*pi* and *raspberry*, respectively, in our example).

10. To make use of the full storage of the microSD card, extend the file system:
- Open the r*aspi-config* tool: `sudo raspi-config`
- Choose *Advanced Options* and then choose option *Expand Filesystem.*
- Confirm with the Enter key.
- Tab to <*Finish*> and confirm with Enter, then confirm reboot by tabbing to <*Yes*> and hitting Enter.

11. To activate the I²C bus:
- Open *raspi-config* tool: `sudo raspi-config`
- Select *Interfacing Options* and then *I2C* (*Enable/disable automatic loading of I2C kernel module*).
- Confirm using the Enter key.
- Confirm again by tabbing to the <*Yes*> button and pressing Enter again.
- Finish up with <*OK*> and then <*Finish*>.

12. Connect the Raspberry Pi to the DS1621 according to our hookup diagram in Figure 2.

13. Check for I²C communication:
- First, do a global update: `sudo apt update`
- Install the I²C diagnostic tools: `sudo apt install i2c-tools` (you might need to reboot the Raspberry Pi first).
- Run the command: `i2cdetect -y 1`
- In the list that appears, the sensor should show up with the address we hardwired, *4f* (**Figure 5**).

14. Now, install Node.js and NPM (Node Package Manager): `sudo apt install -y nodejs npm` (this may take a while).

15. Check the versions using `node -v` followed by `npm -v`.

16. Install the *express*, *i2c-bus*, and *sleep* modules:
`npm install express`
`npm install i2c-bus`
`npm install sleep` (could generate warnings)

17. Create a directory named *Documents* and a subdirectory named *NodeJS.*

18. Using an FTP client such as FileZilla, copy the files *DS1621_V3.js* and *DS1621_V4.html* to the *NodeJS* folder.

OK, Let's Try It!

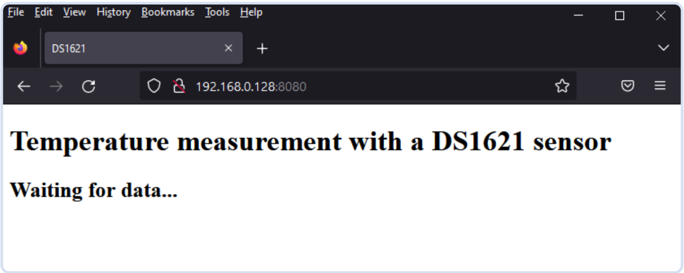
Once all elements are connected, the Raspberry Pi configured, Node.js and the different modules installed, and the *.js* and *.html* files copied onto the Raspberry Pi, open directory *Documents/NodeJS* and execute the script using `node DS1621_V3.js`.

Simply connect a PC, a tablet, or a smartphone to your Wi-Fi network, open your favorite browser and enter the address:

[Your_Raspberry_Pi_IP_address]:8080

If the previous steps were done correctly, it should like **Figure 6**.

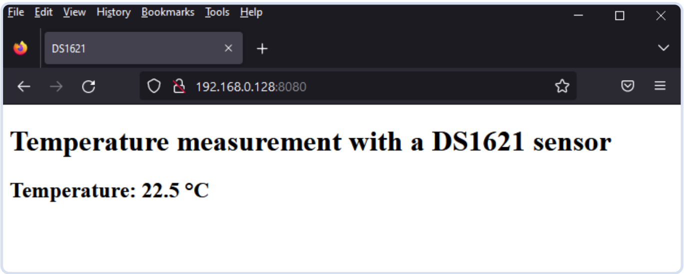
After a few seconds, the temperature will appear (**Figure 7**). From then on, the temperature will refresh every five seconds.



Temperature measurement with a DS1621 sensor

Waiting for data...

Figure 6: Waiting for data.



Temperature measurement with a DS1621 sensor

Temperature: 22.5 °C

Figure 7: Temperature measured and displayed.

Explaining the Node.js and HTML Scripts

What is important, but difficult, with Node.js, is understanding the script's execution. In Node.js, the execution of a script is not sequential. See **Figure 8**: The instructions in the function `Server.get('/',...)` are not executed before the instructions in the function `Server.get('/temp',...)`, which are not executed before the instructions at the bottom.

The instructions in the function `Server.get('/',...)` are executed only if the server receives the *[Rpi_IP_Address]:8080* request from the client. The instructions in the function `Server.get('/temp',...)` are executed only if the server receives the *[Rpi_IP_Address]:8080/temp* request from the client. This is called "Event-driven programming."

In the first five lines of the Node.js script (left side of Figure 8), we import the necessary Node.js modules and create different objects:

- We create an `I2C` object using the `i2c-bus` module:

```
const I2C = require('i2c-bus');
```

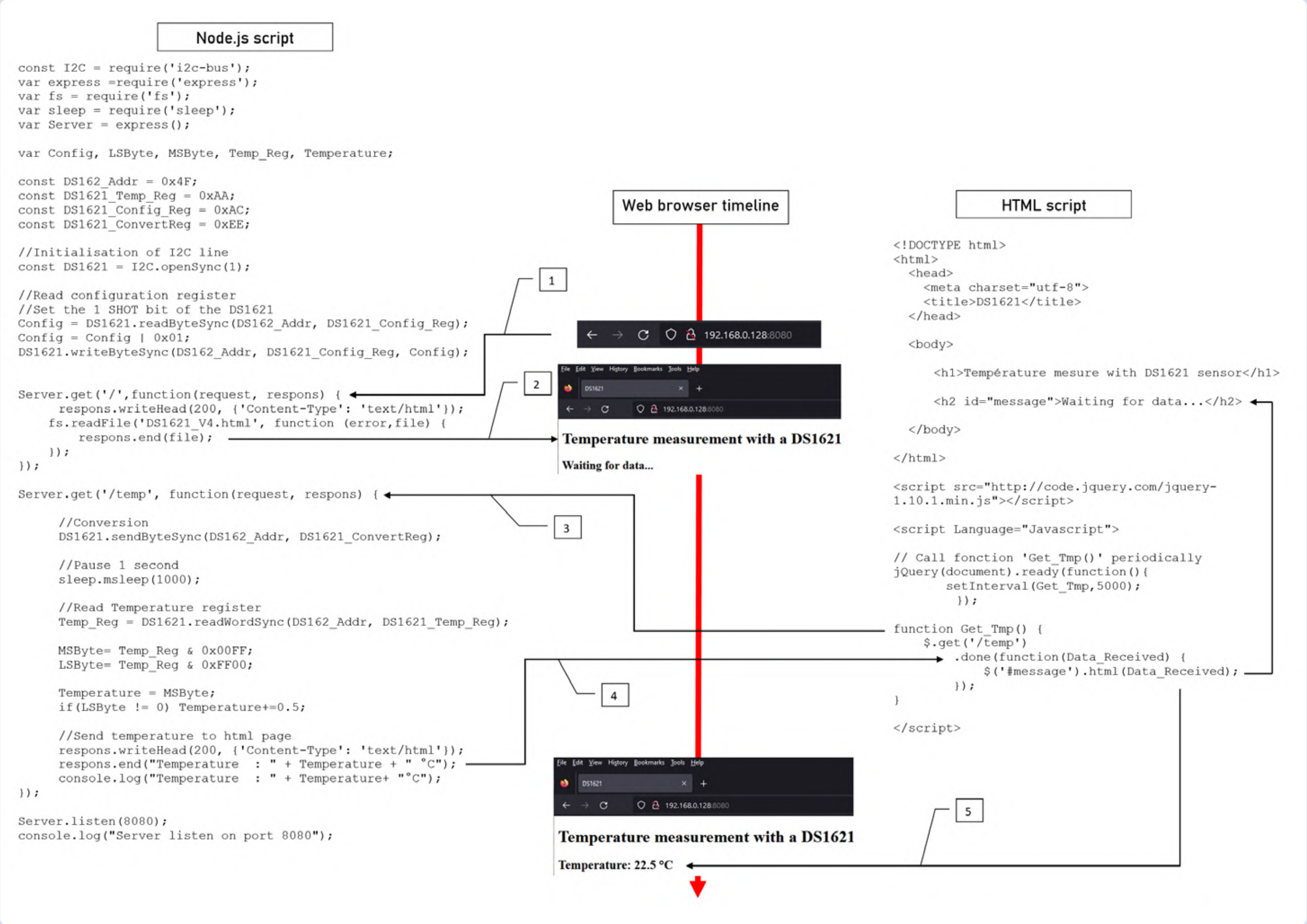
- We create an `express` object using the `express` module. This module is a framework designed for building web applications with Node.js:

```
var express = require('express');
```

- We create an `fs` object using the `fs` (file system) module. This allows us to read and write files on the microSD card:

```
var fs = require('fs');
```

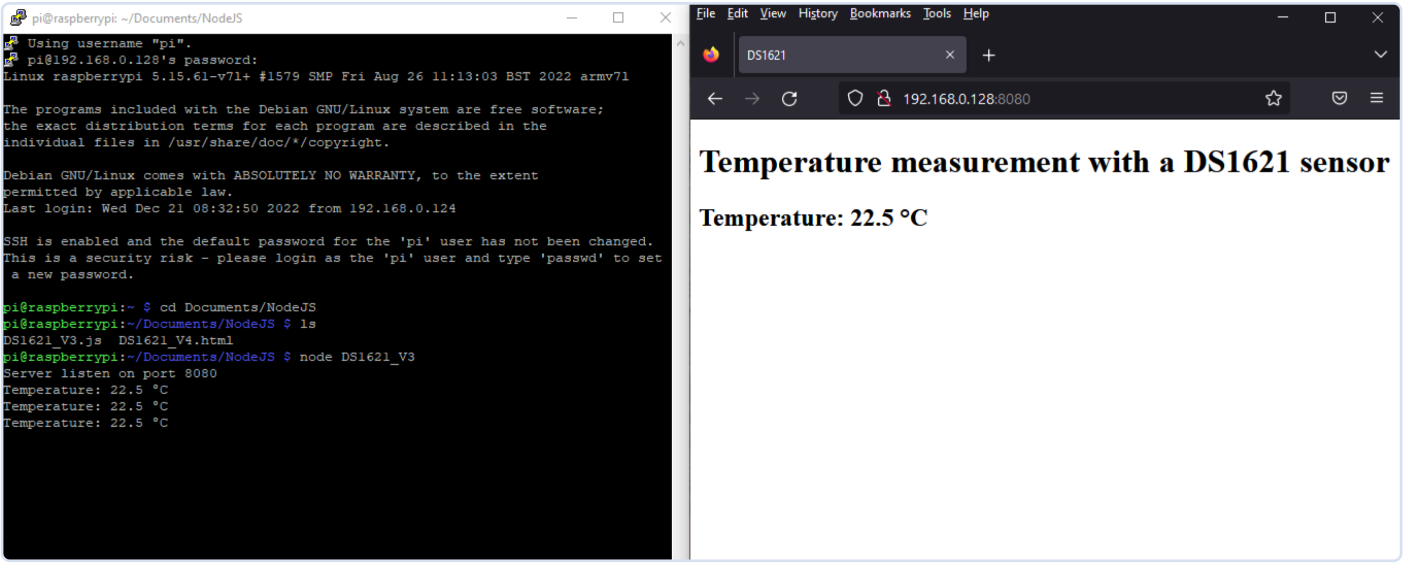
- We create a `sleep` object using the `sleep` module. This module lets us create delays:


Node.js script
Web browser timeline
HTML script
1
2
3
4
5
192.168.0.128:8080
File Edit View History Bookmarks Tools Help
DS1621
Temperature measurement with a DS1621
Waiting for data...
Temperature measurement with a DS1621
Temperature: 22.5 °C

```
const I2C = require('i2c-bus');
var express =require('express');
var fs = require('fs');
var sleep = require('sleep');
var Server = express();

var Config, LSByte, MSByte, Temp_Reg, Temperature;

const DS162_Addr = 0x4F;
const DS1621_Temp_Reg = 0xAA;
const DS1621_Config_Reg = 0xAC;
const DS1621_ConvertReg = 0xEE;

//Initialisation of I2C line
const DS1621 = I2C.openSync(1);

//Read configuration register
//Set the 1 SHOT bit of the DS1621
Config = DS1621.readByteSync(DS162_Addr, DS1621_Config_Reg);
Config = Config | 0x01;
DS1621.writeByteSync(DS162_Addr, DS1621_Config_Reg, Config);


Server.get('/',function(request, respons) {
      respons.writeHead(200, {'Content-Type': 'text/html'});
    fs.readFile('DS1621_V4.html', function (error,file) {
        respons.end(file);
    });
});

Server.get('/temp', function(request, respons) {

      //Conversion
      DS1621.sendByteSync(DS162_Addr, DS1621_ConvertReg);

      //Pause 1 second
      sleep.msleep(1000);

      //Read Temperature register
      Temp_Reg = DS1621.readWordSync(DS162_Addr, DS1621_Temp_Reg);

      MSByte= Temp_Reg & 0x00FF;
      LSByte= Temp_Reg & 0xFF00;

      Temperature = MSByte;
      if(LSByte != 0) Temperature+=0.5;

      //Send temperature to html page
      respons.writeHead(200, {'Content-Type': 'text/html'});
      respons.end("Temperature  : " + Temperature + " °C");
      console.log("Temperature  : " + Temperature+ "°C");
});

Server.listen(8080);
console.log("Server listen on port 8080");
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>DS1621</title>
  </head>

  <body>

     <h1>Température mesure with DS1621 sensor</h1>

     <h2 id="message">Waiting for data...</h2>

  </body>

</html>

<script src="http://code.jquery.com/jquery-
1.10.1.min.js"></script>

<script Language="Javascript">

// Call fonction 'Get_Tmp()' periodically
jQuery(document).ready(function(){
       setInterval(Get_Tmp,5000);
        });

function Get_Tmp() {
    $.get('/temp')
       .done(function(Data_Received) {
           $('#message').html(Data_Received);
       });
}

</script>
```

Figure 8: Timeline of events.


pi@raspberrypi: ~/Documents/NodeJS
File Edit View History Bookmarks Tools Help
DS1621
192.168.0.128:8080

```
Using username "pi".
pi@192.168.0.128's password:
Linux raspberrypi 5.15.61-v7l+ #1579 SMP Fri Aug 26 11:13:03 BST 2022 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Dec 21 08:32:50 2022 from 192.168.0.124

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
 a new password.

pi@raspberrypi:~ $ cd Documents/NodeJS
pi@raspberrypi:~/Documents/NodeJS $ ls
DS1621_V3.js  DS1621_V4.html
pi@raspberrypi:~/Documents/NodeJS $ node DS1621_V3
Server listen on port 8080
Temperature: 22.5 °C
Temperature: 22.5 °C
Temperature: 22.5 °C
```

Temperature measurement with a DS1621 sensor

Temperature: 22.5 °C

Figure 9: The Linux terminal and the web browser.

```
  var sleep = require('sleep');
```

- Finally, we create a `Server` object:

```
  var Server = express();
```

In the next five lines, we define 5 variables and 4 constants. These constants contain the DS1621 registers' respective addresses.

```
var Config, LSByte, MSByte, Temp_Reg, Temperature;

const DS162_Addr = 0x4F;
const DS1621_Temp_Reg = 0xAA;
const DS1621_Config_Reg = 0xAC;
const DS1621_ConvertReg = 0xEE;
```

Next, we open the I²C bus and create an object called `DS1621`. This object will be used to communicate with the sensor through I²C.

```
const DS1621 = I2C.openSync(1);
```

The next three lines set the DS1621's *1SHOT* bit. So, the temperature is measured and converted once (and only once) when a conversion is required by the I²C Master (the Raspberry Pi).

```
Config = DS1621.readByteSync(DS162_Addr,
         DS1621_Config_Reg);
Config = Config | 0x01;
DS1621.writeByteSync(DS162_Addr,
         DS1621_Config_Reg, Config);
```

Now, let's see what happens when the browser makes a request to the server (**Figure 9**).

When the Node.js script is executed, the message "Server listen on port 8080" appears in the Linux terminal after a few seconds. Open a web browser on your PC or tablet connected to the Wi-Fi network. In the address bar, enter *http://* and the IP address of the Raspberry Pi and the port on which the server is listening; for example *http://192.168.0.128:8080.* Port 8080 was chosen to be different to the default used by the Apache web server (Port 80).

1. When you submit the IP address, the browser sends a *get* HTTP request to the server. What happens?
   a. The server detects the request with `Server.get('/'...` and executes the anonymous callback function, `function(request, respons)`.
   b. The `request` object contains the request made by the browser.
   c. In the `respons` object, we send two headers as parameters:
      i. A *200* http code, meaning the request was successful
      ii. The string "'Content-Type': 'text/html'", meaning that the browser must be prepared to receive text data of type HTML.
   d. The `respons.writeHead()` function call returns the parameter data to the browser.

2. The `fs.readFile()` function reads the *DS1621_V4.html* file. The anonymous callback function, `function(error, file)`, is executed and, in the event that it's successful (we'll assume so), the file is red an placed into the `file` object. The `respons.end(file)` function call sends the HTML to the browser, which displays the web page. In the event of an error, an error code is placed in the `error` object.

So, the browser now has the web page in HTML format. What happens next? Let's have a look at the HTML script (right side of Figure 8).

The first thing we see is the line `<h2 id="message">Waiting for data...</h2>`. This is a header placeholder where the temperature will be written. We'll see how later.

The second thing we see is the block that follows. It's a script written in JavaScript, but it's client-side, meaning it will be executed by the browser:

```
<script src="http://code.jquery.com/jquery-1.10.1.min.
js"></script>

<script Language="Javascript">
  // Call function Get_Tmp() periodically:
```

```
  jQuery(document).ready(function() {
         setInterval(Get_Tmp, 5000);
          });

  function Get_Tmp() {
      $.get('/temp')
          .done(function(Data_Received) {
              $('#message').html(Data_Received);
          });
  }
</script>
```

First, from the web, we import the jQuery library, which provides us with some interesting functions. Thanks to jQuery, we can configure a timer that will call a function named `Get_Temp()` every 5 seconds:

```
  jQuery(document).ready(function(){
       setInterval(Get_Tmp, 5000);
       });
```

Finally, we have the `Get_Tmp()` function:

```
function Get_Tmp() {
$.get('/temp')
     .done(function(Data_Received) {
         $('#message').html(Data_Received);
     });
}
```

And that's the interesting part of the script!

Let's see how it interacts with the Node.js script on the server side (see timeline center of Figure 8):

1. When called, function `Get_Tmp()` makes a *get* request to the server. What happens then?
   a. The server detects the request using `Server.get('/temp')` and executes the anonymous callback function, `function(request, respons)`.
   b. The I^2C master (Raspberry Pi) communicates with the DS1621 via the I^2C bus, asks for a data conversion, reads the data and converts the data into °C.
   c. The `request` object contains the request made by the browser.
   d. In the `respons` object, we store our two parameters, again:
      i. The value *200*, a code meaning that the request is successful.
      ii. The string "'Content-Type': 'text/html'" meaning the browser will receive text to be interpreted as HTML.
   e. The function `respons.writeHead()` returns this data to the browser.
2. The `respons.end("Temperature: " + Temperature + " °C")` function call sends the message "Temperature: XX °C" (which is an ASCII string) to the browser. XX is the temperature measured by the DS1621.
3. This string is placed in the `Data_Received` object.
4. Remember the header section `<h2 id="message">Waiting for data...</h2>`? We see that this header tag has the `id` of `message`, so when the string is received, thanks to the line `$('#message').html(Data_Received);`, the string sent by the server is placed in the header section and the web page displays the temperature.

To end with, a few words on the two last lines of the Node.js script :

```
    Server.listen(8080);
    console.log("Server listen on port 8080");
```

The line `Server.listen(8080);` launches the server, and the line `console.log("Server listen on port 8080");` writes the message "Server listen on port 8080" to the Linux console (shell).

Take a closer look at Figure 8, as I think it's better to understand using a timeline.

210367-01

Questions or comments?

Do you have technical questions or comments about this article? Contact Elektor's editorial team at editor@elektor.com.

Related Products

- **Vincent Himpe, *Mastering the I^2C Bus* (SKU 15385)**
  www.elektor.com/15385
- **Raspberry Pi Zero W Starter Kit (SKU 18328)**
  www.elektor.com/18328

WEB LINKS

[1] Introduction to Node.js: A Beginner's Guide to Node.js and NPM: https://elektor.link/udaynode

[2] What is Node.js: A Comprehensive Guide: https://simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs

[3] 'i2c-bus' module: https://npmjs.com/package/i2c-bus

[4] SMBus Compatibility With an I^2C Device: https://ti.com/lit/an/sloa132/sloa132.pdf

Video Output with Microcontrollers (2)

VGA and DVI Output

By Mathias Claussen (Elektor Labs)

Microcontrollers are capable of sending pixels to a monitor via a VGA or DVI interface. Whether with the ESP32 or the Raspberry Pi Pico's RP2040, significantly more than 256 colors are possible. Thanks to sprites, tiles, and a few tricks, today's microcontrollers can generate graphics at least as sophisticated as classic game consoles of the 1990s.

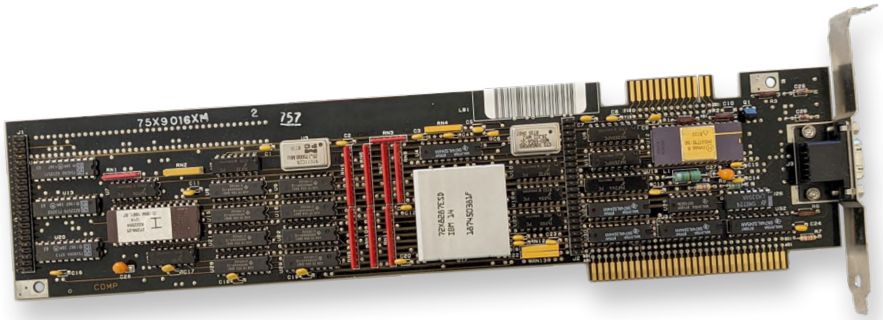


Figure 1: IBM VGA card (source: http://nerdlypleasures.blogspot.com/).

The first part of this article covered the output of composite video signals in PAL (**P**hase **A**lternating **L**ine) or NTSC (**N**ational **T**elevision **S**ystems **C**ommittee) format. While the output of monochrome pictures is even possible with an Arduino UNO, pictures in color make higher demands. With a few tricks, however, you can convince an ESP32 or even an ATmega to produce color output. In principle, however, the picture quality with composite video is mediocre. Mainly, sharpness is missing, and text is thus not well readable.

The first home computers, such as the Sinclair ZX81 and the Commodore 64, generated composite video. For professional applications, however, the sharpest possible text display was needed, and, in some cases, even color graphics. Thus, IBM offered something new and better for its PCs in 1987 with its VGA (**V**ideo **G**raphics **A**rray) interface. Microcontrollers are also capable of generating these analog VGA signals to display a sharp and colorful image. Some microcontrollers can even output an image completely digitally via DVI (**D**igital **V**isual **I**nterface). So, our second part in this series is about VGA and DVI and the tricks used for graphics output and effects.

VGA: Video for the IBM PC

VGA was the graphics card (**Figure 1**) for PS/2 PCs introduced by IBM in 1987. This graphics card made the then-new, 3-row D-sub DE-15 connector (**Figure 2**) popular, although VGA not only specifies the physical connector, but also the necessary signals and their timing.

The color signals in VGA are analog and are transmitted via three signal lines (red, green, and blue). The horizontal and vertical synchronization are done digitally using separate TTL signals. The first VGA card from IBM could output 256 colors from an RGB palette of 3 × 6-bit (= 262,144 colors) at a resolution of 320×200 pixels and still managed to generate 640×480 pixels with 16 colors. The first IBM monitors suitable for this could only handle 640×480 or 640×400 pixels at 60 or 70 Hz refresh rates. IBM's VGA cards were soon cloned by many other manufacturers, making VGA a de facto standard for video output on what were called IBM-compatible PCs.

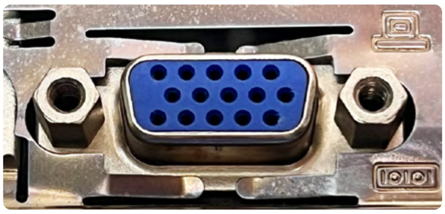


Figure 2: VGA socket on a PC (source: Dr. Thomas Scherer).

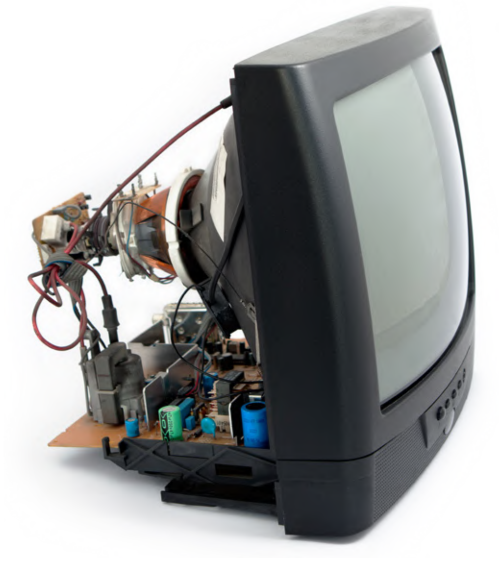


Figure 3: Opened TV with cathode ray tube (source: Shutterstock / Serhiy Stakhnyk).

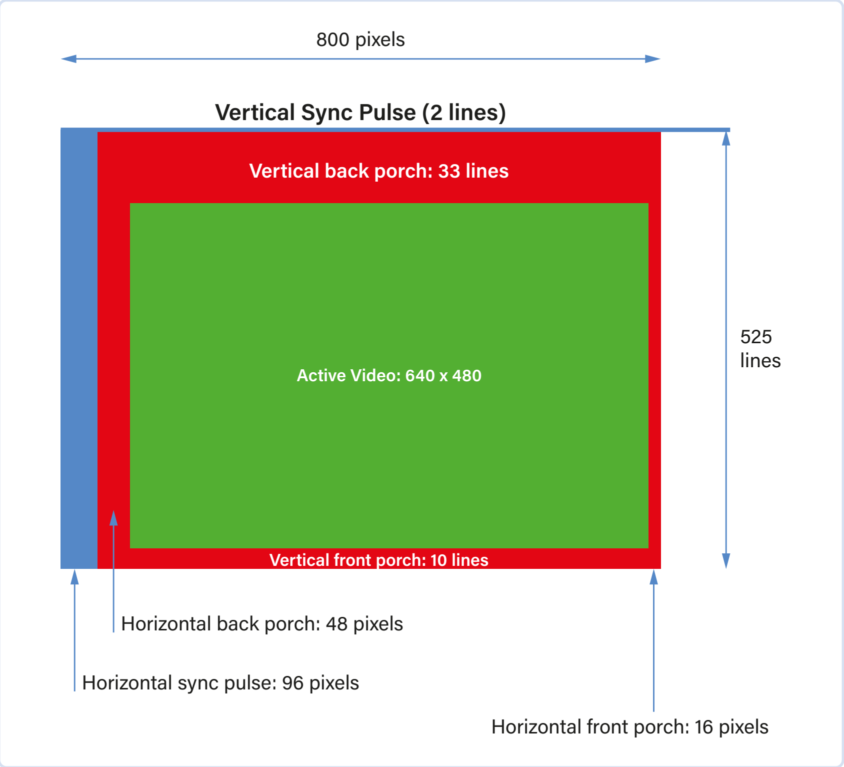

800 pixels
Vertical Sync Pulse (2 lines)
Vertical back porch: 33 lines
Active Video: 640 x 480
525 lines
Vertical front porch: 10 lines
Horizontal back porch: 48 pixels
Horizontal sync pulse: 96 pixels
Horizontal front porch: 16 pixels

Figure 4: Timing of a VGA signal with 640×480 visible pixels (source: https://elektor.link/VGAtiming).

Visible and Invisible Pixels

A video signal with 640×480 visible pixels at a refresh rate of 60 Hz operates with a pixel frequency of 25.175 MHz. However, multiplying the pixels of an image by the frame rate results in only 18.432 MHz. Where does this difference come from?

Just like PAL and NTSC composite signals, VGA also has non-visible signal areas, as this standard, too, was designed for monitors based on cathode ray tubes (**Figure 3**). Basically, similar timing requirements apply here as for controlling the picture tube of a television set.

Figure 4 shows the timing for the 640×480-pixel mode with a pixel clock of 25.175 MHz. An image starts with a vertical synchronization pulse of 63.5551142 µs (= two image lines of 800 pixels each). The three electron beams are blanked out during this time, as they move from the end of the last complete image on the lower-right to the beginning of the upper-left of the new image. The lines start with a horizontal synchronization pulse lasting 69 pixels (≈ 3.81 µs). In the next 33 lines (lines 3 to 35), the horizontal synchronization pulse is followed by what is known as the "vertical back porch" of 704 pixels in this case (≈ 27.96 µs). It is intended to give the electron beams or the magnetic deflection enough time to return to the starting position for the new image. This is followed by 480 lines of actual image information, beginning with the horizontal sync pulse of 96 pixels and then the "horizontal back porch" of 48 pixels (≈ 1.9 µs), plus the 640 image pixels of one line (≈ 25.42 µs) and the final "horizontal front porch" of 16 pixels (≈ 0.64 µs). **Figure 5** shows the structure of such an image line. Finally, there are 10 lines, each with a horizontal sync pulse followed by the "vertical front porch" (again consisting of 704 pixels each). The transmitted image of 640×480 visible pixels thus actually consists of 800×525 pixels as far as timing is concerned.

The electron beams are only active during the actual image and remain blanked out otherwise. Even though this sounds complex, it is actually one of the simplest image signals that can be generated by a microcontroller. Just so that you are not surprised: On the internet, you may find slightly different timing specifications for the VGA frame rate. And, if you divide the pixel clock of 25.175 MHz by the 800×525 pixels, the result is not exactly

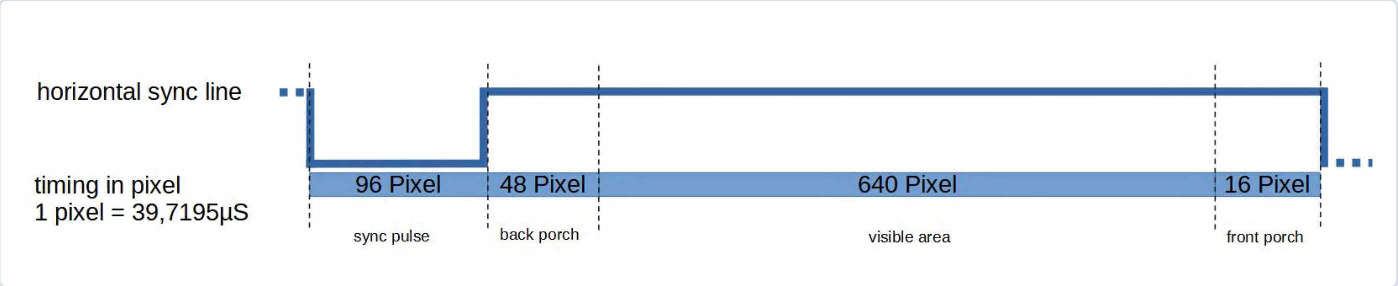


Figure 5: Timing of a line in VGA.

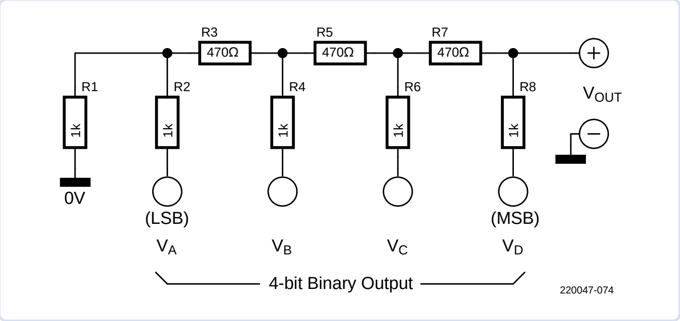


Figure 6: DAC with R2R ladder.

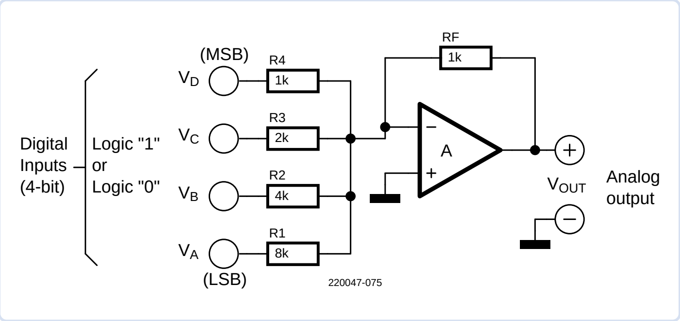


Figure 7: Binary-weighted DAC with op-amp (source: https://elektor.link/BinaryWeightedDAC).

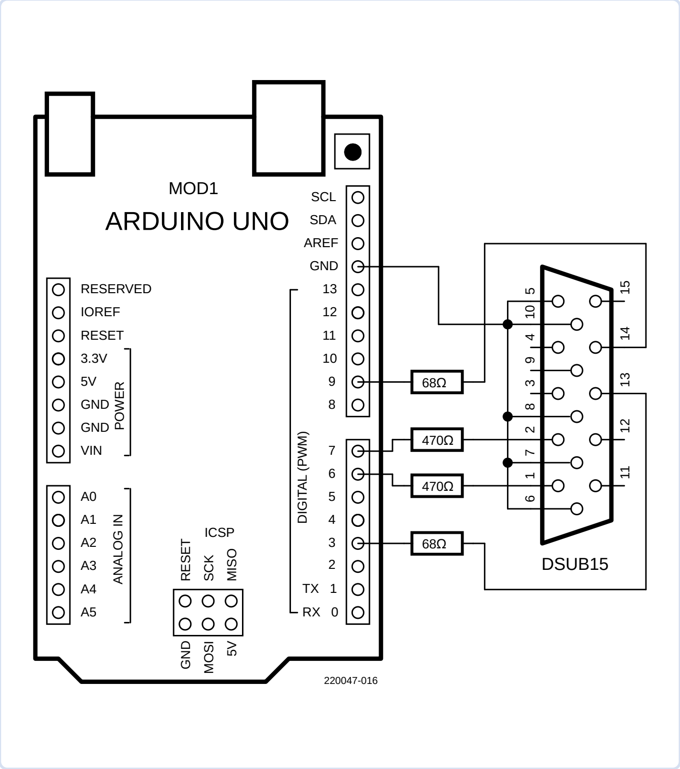


Figure 8: Wiring for VGA output on an Arduino UNO.

60 Hz for the frame rate, but ≈ 59.94 Hz. However, the analog electronics of the associated monitors are not too demanding in this respect and would still synchronize cleanly even with larger deviations from 60 Hz.

VGA Signal Levels

According to the VGA specification, the voltage range of the analog RGB image signals is 0 V to 0.7 V, where 0.7 V corresponds to the maximum brightness. The horizontal and vertical synchronization signals have TTL levels (0 V and 5 V).

Microcontrollers with operating voltages of 2.5 V to 5 V can output the signals for vertical and horizontal synchronization directly through their I/O pins, since 2.5 V is reliably interpreted as a TTL “high” level. To output the varying brightness information for each color, however, voltage dividers are required.

Since the internal DACs of most microcontrollers are not fast enough, the GPIO pins are usually used as simple DACs for generating the three analog signals using resistors. When calculating the necessary resistor values, it must be considered that the analog RGB inputs of the monitor are terminated at 75 Ω. The DACs are usually implemented using R2R networks (**Figure 6**) or binary-weighted resistors (**Figure 7**).

VGA on the ATmega

VGA signals can even be generated with ATmega microcontrollers, so an Arduino UNO can be equipped with VGA graphics. With four resistors and a 15-pin SUB-D connector, we can get an Arduino UNO to display 120×60 pixels in four colors. The corresponding Arduino library is *VGAX* [1] and can be downloaded from the project’s GitHub page.

If an Arduino UNO is wired as shown in **Figure 8**, it is even possible to output graphics in four colors, as shown in **Figure 9**. The limiting factor here is the amount of memory that an Arduino UNO can provide. Since the color signals are generated by the GPIO pins, we must be able to supply them with new values quickly enough.

The timing for signal generation is solved by interrupts in this case. A side effect of this is that interrupts of other peripherals could possibly collide with those for signal generation. Timer1 and Timer2 generate the

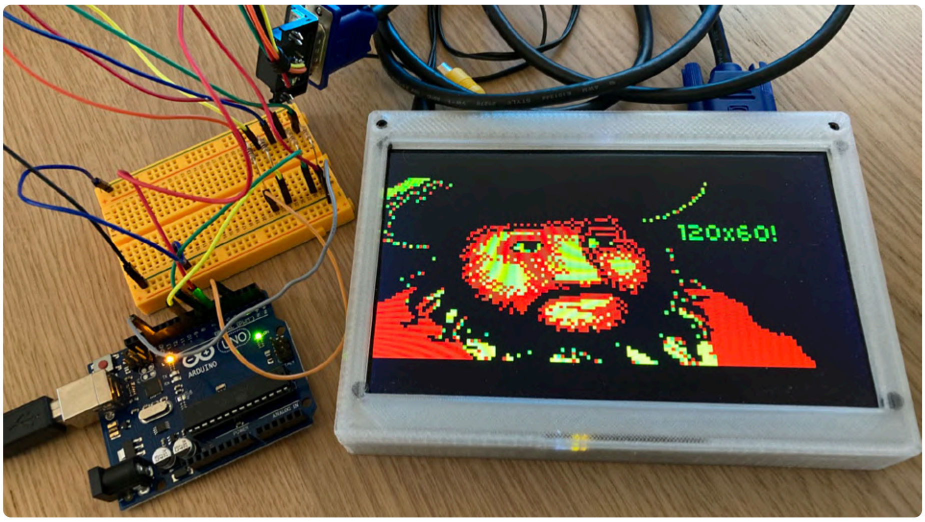


Figure 9: VGA output on a screen connected to the Arduino UNO.

horizontal and vertical synchronization pulses here. Timer0 is used to bypass the jitter in the interrupt calls.

The *VGAXUA* library [2] for the Arduino UNO and Arduino Mega can output monochrome images from 192×080 to 200×240 pixels. To save computing time, the USART is used here to generate the image. Each byte in the (image) memory then corresponds to 8 pixels. Older solutions for VGA output with AVR chips either overclocked the microcontroller up to 32 MHz [3] or used external memory for video output [4].

VGA on the ESP32

An ESP32 can do VGA as well. Ideal conditions are provided by its clock of 240 MHz and the flexible I/O matrix, which allows digital signals to be output at up to 80 MHz. The internal memory of the EPS32 with 520 KB is sufficient for an output of 640×480 pixels with 256 colors (= 307,200 bytes), still leaving enough RAM for other software.

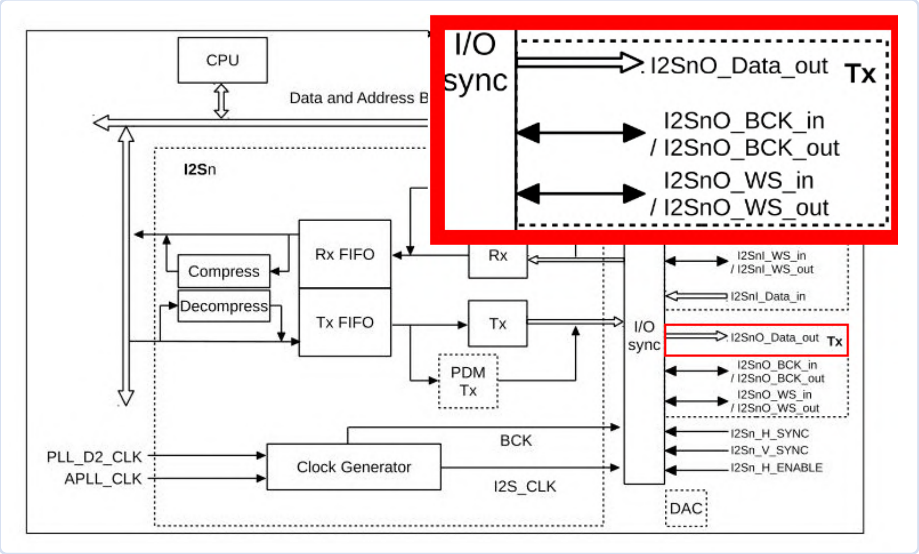


Figure 10: ESP32's I^{2}S controller (source: Espressif — https://elektor.link/ESP32TechMan).

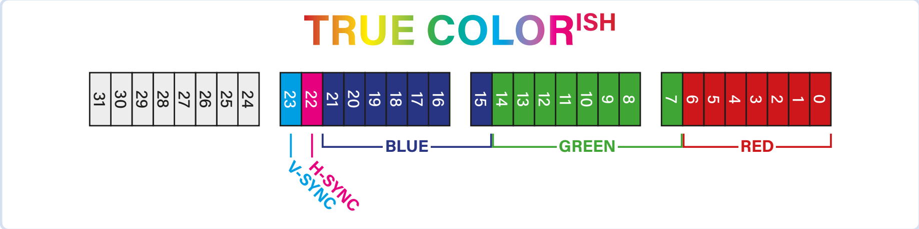


Figure 11: 22-bit video output with H- and V-sync on ESP32 (source: YouTube — https://elektor.link/YTESP32VGA).

The timing specification for a VGA signal at 640 480 pixels and 60 Hz can be found in [5]. The project ESP32 VGA [6] shows an interesting way to generate a VGA signal using the internal I^{2}S controller (**I**nter-**I**C **S**ound) [6]. The I^{2}S controller of the ESP32 can do more than just output audio data [7]: It offers additional operating modes as an interface for a camera or LCD (**Figure 10** shows the corresponding part of the block circuit).

The I^{2}S controller's LCD interface can provide a bus width of up to 24 bits. This would even be enough for TrueColor (16,777,216 colors) if the complete 24 bits could be used for the output of color. However, a VGA signal requires horizontal and vertical synchronization in addition to the RGB signals, so only 22 bits are available for color (**Figure 11**).

There are still two hurdles to overcome: An image of 640×480 pixels occupies 921,600 bytes of RAM at 24-bit color – that's more than an ESP32 offers. In addition, a pixel clock of 25.175 MHz must be achieved. For the output of 24 bits, the I^{2}S controller must read 32 bits per pixel from memory. It seems that the I^{2}S controller in the ESP32 can output the 32-bit words at a maximum of 10 MHz, which is not sufficient. The serial output of the pixels is therefore limited to 10 MHz. The maximum resolution, on the other hand, is limited by the internal memory.

See [8] for a list of the pixel clock rates vs. VGA resolution. In the mode with 800×600 pixels at 60 Hz, the pixel clock of 40 MHz is exactly four times what an ESP32 can output. Thus, 200×600 pixels at 60 Hz and 22-bit color would be mathematically possible in terms of timing and RAM, although these pixels would no longer be rectangular, but horizontally stretched. For rectangular pixels, the vertical resolution must be quartered, and each image line must be output four times. The result is 200×150 pixels in TrueColor. An image with a theoretical 800×600 pixels (**Figure 12a**) is thus displayed rather coarsely, but nicely colored (**Figure 12b**).

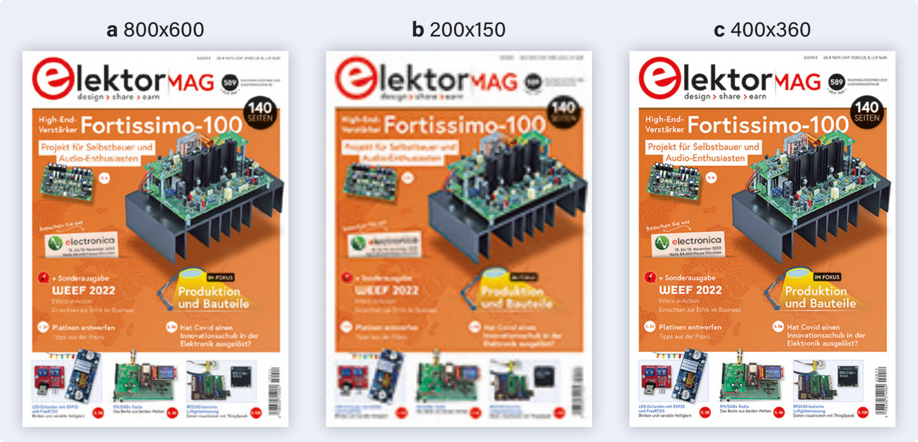

a 800x600 **b** 200x150 **c** 400x360

Figure 12: 800×600 (a) becomes 200×150 (b) and 400×360 pixels (c). (b) and (c) are scaled to the same size as (a) a for better comparison.

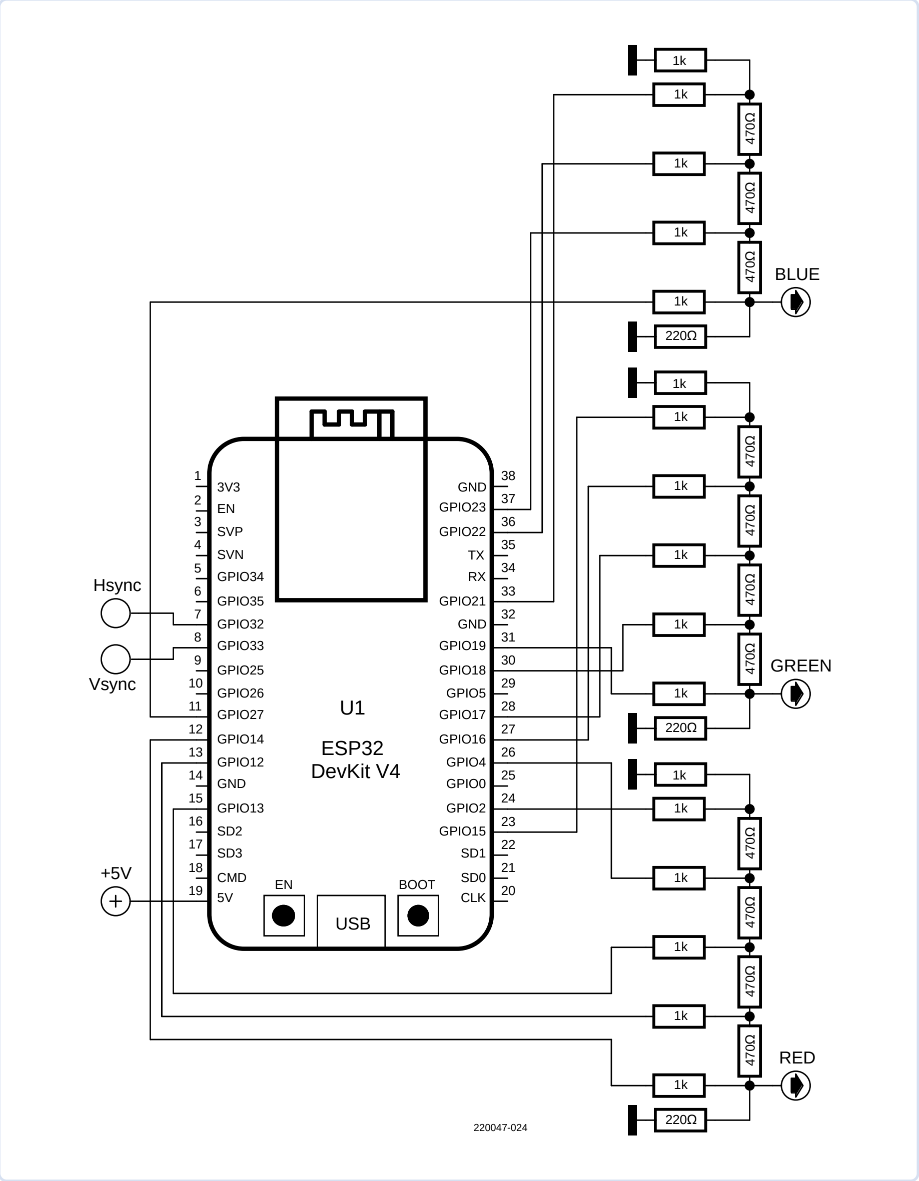
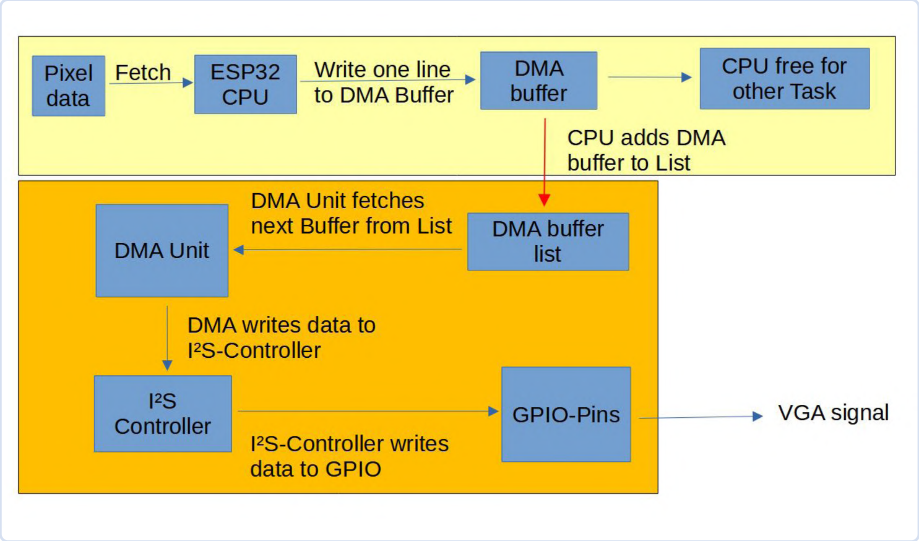


Figure 13: VGA signal generation via R2R DAC on ESP32.



Pixel data
Fetch
ESP32 CPU
Write one line to DMA Buffer
DMA buffer
CPU free for other Task
CPU adds DMA buffer to List
DMA Unit fetches next Buffer from List
DMA Unit
DMA buffer list
DMA writes data to I²S-Controller
I²S Controller
I²S-Controller writes data to GPIO
GPIO-Pins
VGA signal

Figure 14: Pixel output with ESP32.

But, there is a useful compromise, because the I²S controller can also process 16-bit words. Of the 16 bits, 14 bits are then available for the image information and 2 bits for the synchronization signals, which makes for 16,384 displayable colors. For a stable VGA signal, the APLL of the ESP32 should be used. This provides a stable time base for the VGA timing. In this way, images in 14-bit color with resolutions of 320×240, 360×400, or 480×400 pixels can be realized.

To output the RGB signals, the 14-bit color information must be D/A-converted. An R2R DAC is a simple and effective solution for this purpose. The circuit of **Figure 13** shows how such an approach can be implemented. The graphic of Figure 12a looks much better with a resolution of 360×400 pixels (**Figure 12c**).

To fetch and display the data from the RAM, there are a few tricks: To transport the data to the I²S controller, the DMA controller (Direct Memory Access) of the EPS32 is used to reduce the load on the CPU. It can transport data from one place to another without involving the CPU. The source, destination, data quantity, transfer type and the transfer start condition are sufficient for this purpose.

The ESP32 prepares the data for each line and stores it in a line buffer. In addition to the visible pixels, the information for the horizontal and vertical synchronization is also added. When a line has been prepared, the DMA controller transfers the data from the buffer to the I²S controller. The CPUs of the ESP32 can take over other tasks during this time. A rough sequence of the process is shown in **Figure 14**.

This procedure also allows the output of graphics without keeping the complete image in memory. For this, however, the ESP32's CPU must be able to calculate the pixels for the next line during the output time of the current image line.

The VGA library from bitluni for the ESP32 Arduino framework is available on GitHub [9]. But, the ESP32Lib library can do much more than just VGA output. It also contains functions for sprites [10] and a 3D renderer. Furthermore, the FabGL library [11] not only allows VGA output with the ESP32, but provides ingredients for a complete system of graphics and sound. There are even video tutorials for FabGL on YouTube [12].

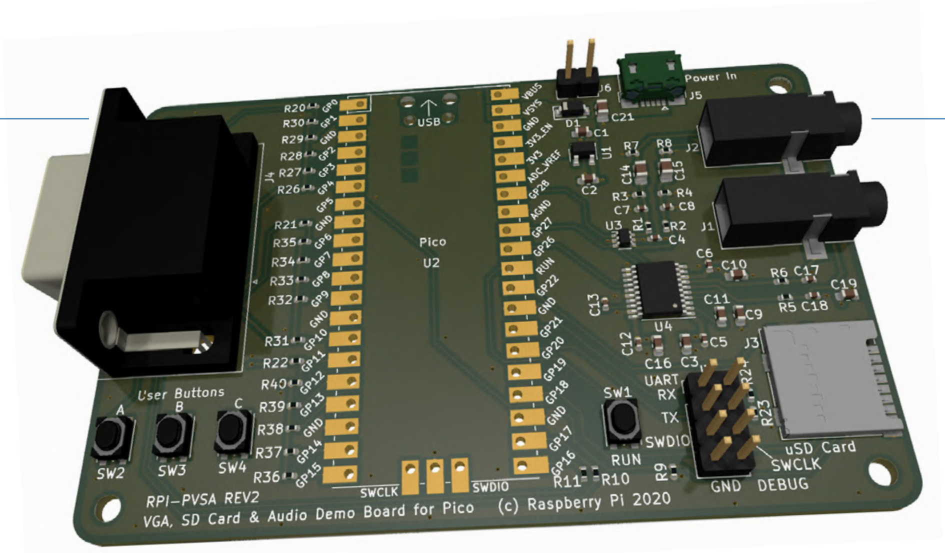


Figure 15: VGA, SD Card & Audio Demo Board for Pico.

VGA on the Raspberry Pi Pico

Its RP2040 SoC makes the Raspberry Pi Pico not only small, but very versatile. With two CPU cores, 264 KB of internal RAM plus external flash memory, this board is inexpensive, yet suitable for many projects. Even at the release of the Raspberry Pi Pico, the generation of VGA signals was presented with a suitable demo board (**Figure 15**). Meanwhile, the board has been updated so that a Raspberry Pi Pico W can also be used. The necessary files for the board [13] are available as a KiCad project, allowing everyone to build it.

The RP2040 offers enough memory to store a 320×240 pixel image at 16 bits (= 65,536 colors) completely in RAM. The PIO state machines of the RP2040 are ideally suited for this purpose.

For the VGA signal, values for the RGB levels as well as for the horizontal and vertical synchronization are necessary. For the generation of these signals with the RP2040's PIO state machines, only some software is necessary. Unfortunately, the RP2040 does not come with a D/A converter, so an external circuit is necessary.

Either a simple R2R DAC or a binary-weighted DAC is suitable for this purpose. For an R2R DAC, only two resistor values are needed, which simplifies things a lot. For a binary-weighted DAC, a different resistor value is needed for each bit. The circuit for the demo board with VGA output shown in **Figure 16** is taken from a PDF file titled "Hardware design with RP2040" [14] by Raspberry Pi Ltd. For RGB565-specified colors, 5 GPIO pins each are used for the colors red and blue, whereas green requires six pins for the resulting 65,536 colors. The resistor values result from the structure of the DAC, the necessary output voltage, and the monitor's terminating resistors.

The values of the analog RGB signals are between 0 V and 0.7 V. A monitor's RGB inputs

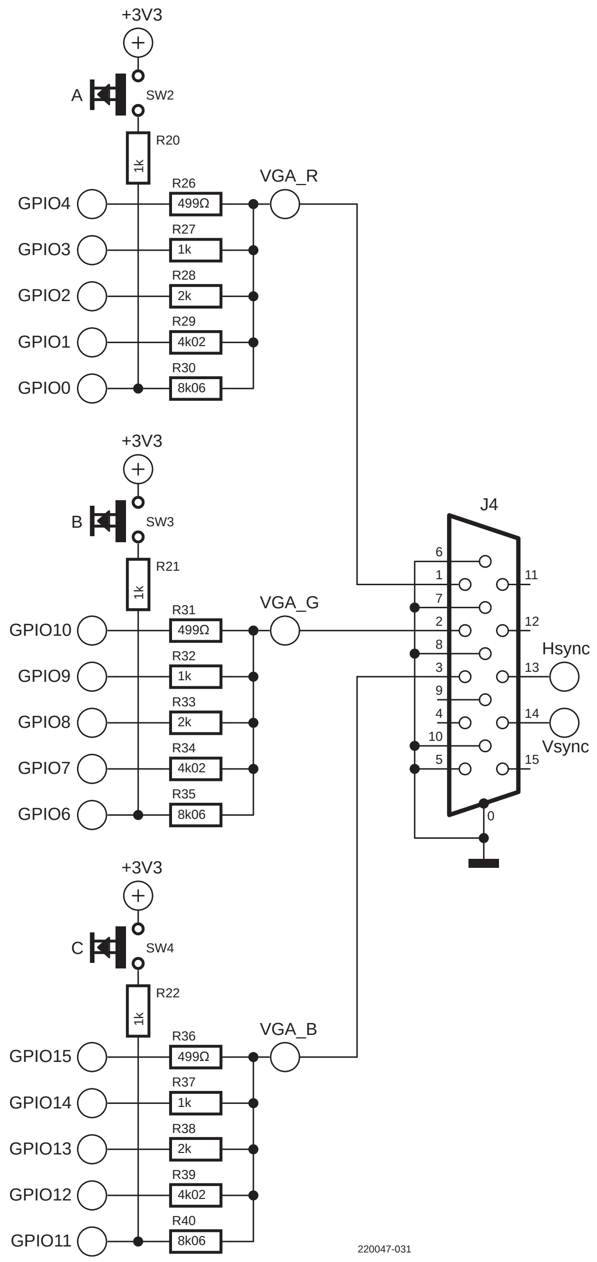


Figure 16: Part of the VGA Base circuit for the Raspberry Pi Pico.

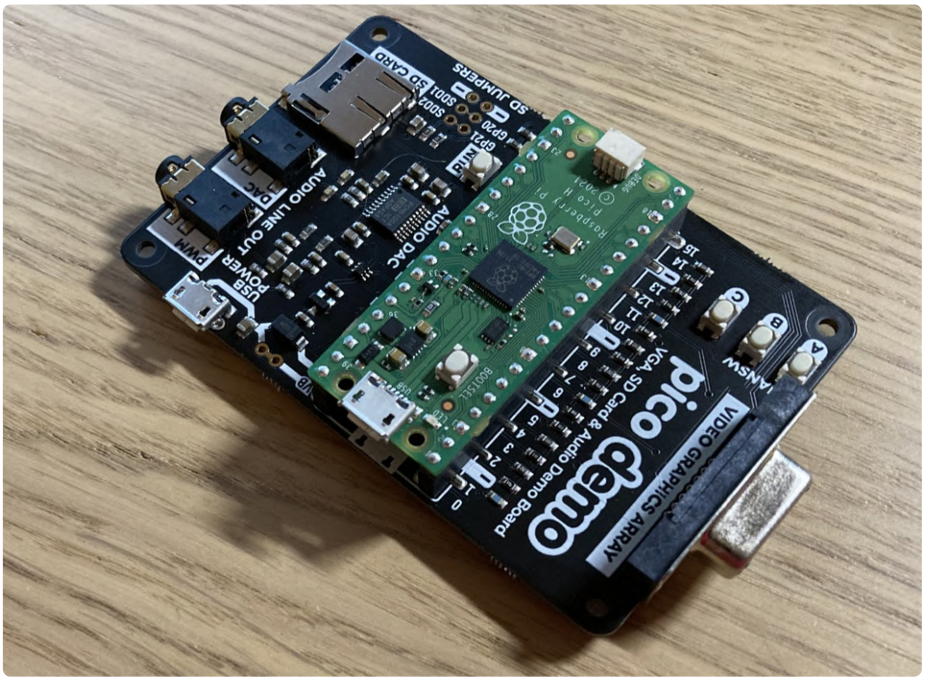


Figure 17: Pimoroni VGA Base for the Raspberry Pi Pico.

are terminated with 75 Ω to ground. Therefore, the voltage dividers must generate a maximum of 0.7 V for each of the three colors using the GPIOs' 3.3 V. This results in resistor ratios of 1:2:4:8:16. To get 0.7 V from 3.3 V, you need a voltage divider with a ratio of 3.7:1. Five or six resistors per color are connected in parallel. The parallel circuits must result in 278 Ω to get to a maximum of 0.7 V at the 75 Ω terminating resistor with a supply of 3.3 V.

With resistors from the E-96 series, values of 499, 1000, 2000, 4020 and 8060 Ω are available with 1% tolerance. Connected in parallel, this results in 258 Ω, which leads to sufficiently accurate 0.74 V at the terminating resistor.

If you want a fully assembled board, you can buy the Pimoroni Pico VGA Demo Base (**Figure 17**) — see also the Related products box. In addition to the VGA output, an audio codec and an SD card slot are provided.

Pixel output with the RP2040

In order for a VGA signal to be output, the image data must be generated and stored. The RAM of the RP2040 is sufficient to store an image of 320×240 pixels at 16-bit color. The *pico-playground* GitHub repo [16] contains suitable demos.

For a simple output, 150 KB of memory is reserved, in which the image data is stored as RGB565 values. The image is read from memory line by line, and then the appropriate timing for the image output is generated for each line. **Figure 18** shows a simplified sequence.

This kind of image output from RAM ensures that both CPUs remain free for other tasks. Only a new image line with suitable timing must be provided. The *pico_scanvideo* library responsible for this can be found at [17]. It allows for the preparation of more than one image line in advance. The number of image lines is configurable, as is the scaling of the screen resolution. So, for example, you can output 320×200 pixels with a resolution of 640×480 pixels. Buffering the image lines also allows other tricks to be achieved, such as generating images "on the fly." This is more or less a race against time or against the electron beam.

Raspberry Pi Pico as a Drawing Artist

The Raspberry Pi Pico and other boards based on the RP2040 can be true drawing artists. In **Figure 19**, several sprites move across the screen. The output is 640×480 pixels at 16-bit color. This is more than would actually fit in the RAM of the Raspberry Pi Pico. For this reason, all pixels are redrawn for each new image to be output.

Figure 19: Moving sprites.

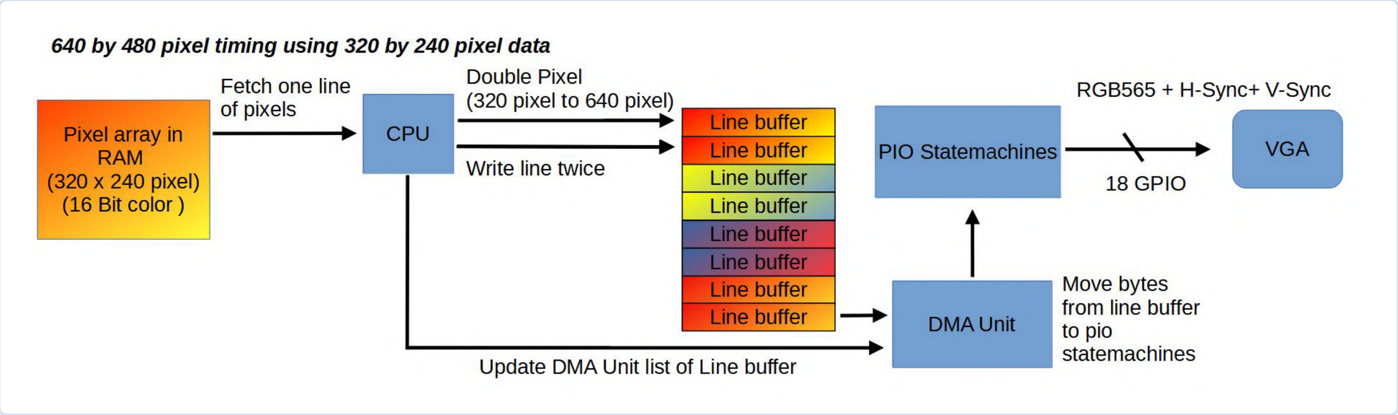

640 by 480 pixel timing using 320 by 240 pixel data
Pixel array in RAM (320 x 240 pixel) (16 Bit color)
Fetch one line of pixels
CPU
Double Pixel (320 pixel to 640 pixel)
Write line twice
Line buffer
Line buffer
Line buffer
Line buffer
Line buffer
Line buffer
Line buffer
Line buffer
PIO Statemachines
RGB565 + H-Sync+ V-Sync
18 GPIO
VGA
DMA Unit
Move bytes from line buffer to pio statemachines
Update DMA Unit list of Line buffer

Figure 18: VGA output sequence with Raspberry Pi RP2040.

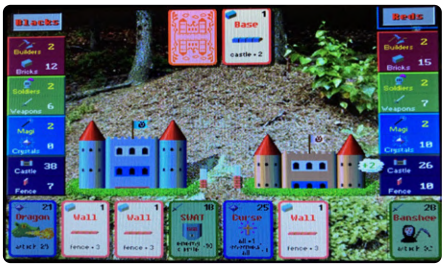


Figure 20: *Ants* on a Raspberry Pi Pico.

Figure 21: *Pac-Man* on a Raspberry Pi Pico.

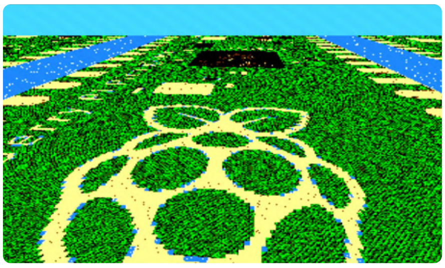


Figure 22: Pseudo-3D with a Raspberry Pi Pico.

While this demo of basic functions can draw 2D graphics on the screen, Miroslav Nemecek's *PicoVGA* library goes a few steps further. The graphics output is limited to 8 bits, which leaves more free IO pins and requires less RAM for a complete image (75 instead of 150 KB). The library's GitHub repo contains several demos and small games [18]. **Figure 20** shows the game *Ants* running on the Raspberry Pi Pico. **Figure 21** shows the classic *Pac-Man*. A short description of the library can be found on the German MagPi page [19].

The Raspberry Pi Pico contains functional units that even allow pseudo-3D effects (**Figure 22**). This type of display was once made famous by the SNES (Super Nintendo Entertainment System) with games such as *Super Mario Kart* (**Figure 23**) or *F-Zero* as Mode 7 graphics [20]. The corresponding library is available on GitHub at [21].

DVI

DVI is an interface introduced in 1999 [22] that delivers image data digitally to a monitor without quality loss. A typical white socket for two-channel DVI-D can be seen in **Figure 24**. The exact DVI specifications can be found in a PDF file at [23].

When the first LCD monitors became affordable, many graphics cards still output the image information via the VGA interface. But at Full HD (1920×1080 pixels), the image was not really sharp due to the analog signal interface. With DVI, there was soon a widespread digital interface that could supply an LCD monitor directly, without conversion losses, with high-resolution images in 24-bit color.

8b/10b, TMDS and 1:10 Clock

Whereas with VGA three of the five signals were analog, with DVI, everything is digital. Four differential line pairs (±Data 0, ±Data 1, ±Data 2 and ±Clock) represent the minimum for single-channel image transmission using DVI.

The RGB image data are transmitted via Data 0, Data 1, and Data 2. For an image in 24-bit color, each color is transmitted using 8 bits. In addition, there is a clock signal that helps to combine the colors to complete pixels on the receiver side.

Digital transmission uses the TMDS (Transition-Minimized Differential Signaling) method [24], which ensures robust signal transport and low electromagnetic radiation due to its data encoding.

The coding method used is 8b/10b [25], where only 460 of 1024 possible combinations are used to encode the 8-bit information for the three colors,red, green, and blue. Several 8-bit values can thus be represented by two combinations. Four combinations are used for signaling C0 and C1 for horizontal and vertical synchronization. When outputting the pixels for a line, the number of zeros and ones in the data stream is balanced to obtain DC-neutral data streams, which is made possible by the double representation of some 8-bit values.

Even if the image information is transmitted completely digitally, horizontal and vertical synchronization signals are still present. However, these are no longer dedicated data lines; instead, they are encoded in the Data 0 stream with the help of C0 and C1. The DVI data stream is designed to be easily converted to an analog VGA signal.

The clock signal in DVI does not correspond to the bit rate for red, green, and blue, but has a ratio of 10:1 and corresponds to the pixel clock due to the 8b/10b coding. For the lowest DVI resolution of 640×480 pixels at 60 Hz, a pixel clock of 25.175 MHz or a bit rate of 251.75 MHz is required for the color information, just as with VGA. So, the VGA origins are still obvious in DVI.

Figure 23: *Super Mario Kart* with pseudo-3D effect.

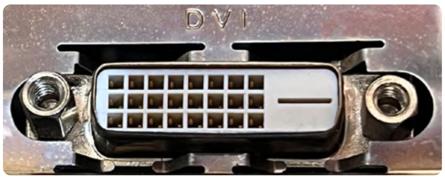


Figure 24: DVI-D socket on a PC (source: Dr. Thomas Scherer).

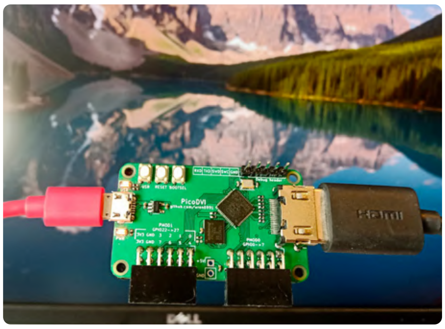


Figure 25: PicoDVI — RP2040 with DVI video output (source: Wren6991 / https://elektor.link/picodviimg).

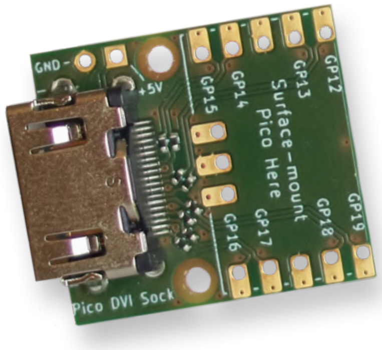


Figure 26: Pico DVI Sock for Raspberry Pi Pico.

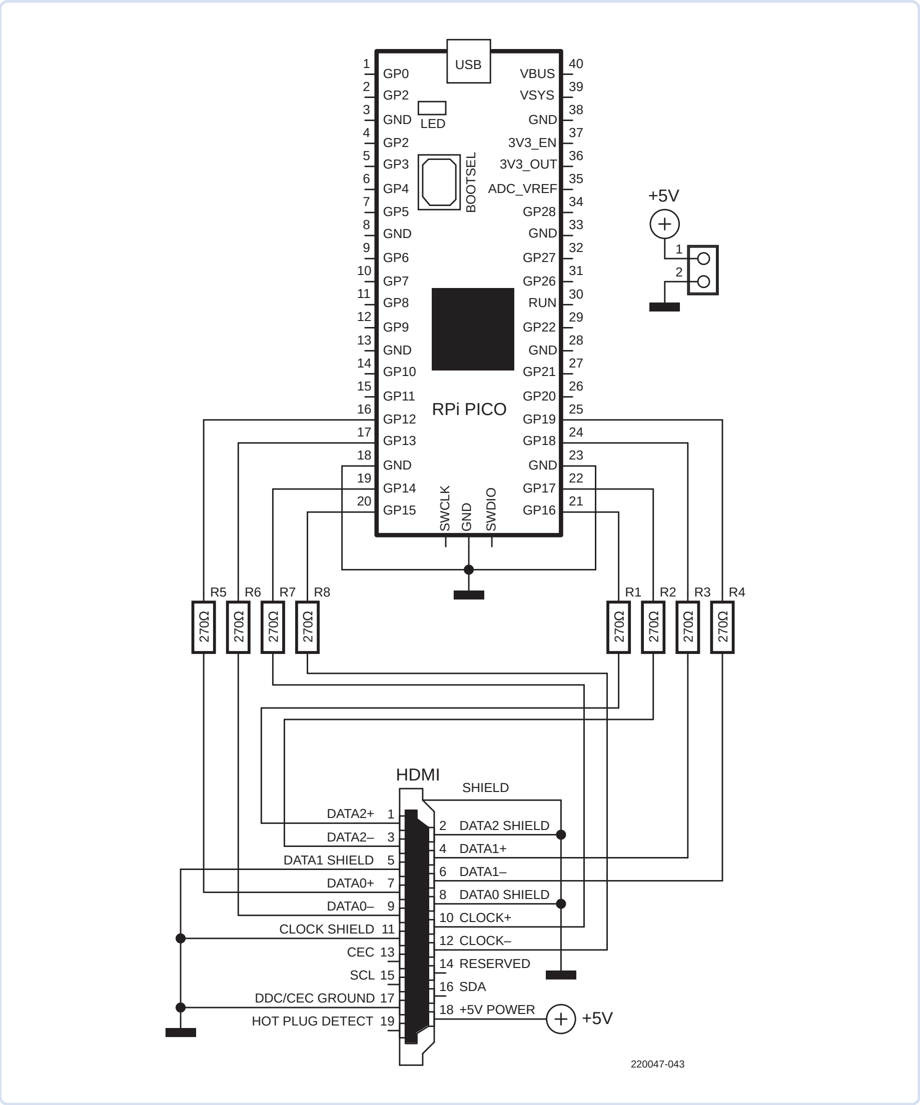


Figure 27: Pico DVI Sock schematic.

DVI with Raspberry Pi Pico

Assuming the previously specified lowest-possible resolution, a Raspberry Pi Pico has to output three data streams at 251.75 Mbps. This is simply not possible with its maximum clock of 133 MHz, because almost 2 bits would have to be output per clock step for each of the three color data streams.

So, the Raspberry Pi Pico could already fail at this hurdle. However, the RP2040 is extremely overclockable. A clock of 251.75 MHz for both cores is possible without noticeable malfunctions at room temperature. However, the RP2040's GPIO pins must also keep up with this enormous speed. In addition, the image data must still be encoded as a TMDS stream and the data must therefore be suitably prepared. Luke Wren was able to show [26] that an RP2040 is capable of doing this (see **Figure 25**); for more information, read the interview at [27].

One of the two cores of the RP2040 is about 60% busy with TMDS encoding. In addition, three of the eight PIO state machines are used to output the data with the necessary 251.75 MHz. Some functions of the DMA controller relieve the CPU cores from data transport. This way, one core remains completely free and is fully available for your own software.

One DVI or HDMI socket (**Figure 26**) plus eight resistors is sufficient to electrically connect the Raspberry Pi Pico to a monitor's DVI or HDMI input. The required wiring (**Figure 27**) is quite straightforward.

In the PicoDVI's GitHub repo [26], Luke Wren describes how the RP2040's interpolator can be used to speed up TMDS encoding, which saves computing time. The interpolator was originally intended to generate pseudo-3D graphics, as in or *Pilotwings* (**Figure 28**). However, it is flexible enough to use it to relieve the CPU cores when generating the three TMDS data streams from the RGB data.

So, how do the pixels get from the RAM to the monitor? For output with a resolution of 640×480 pixels, an image of 320×240 pixels is scaled up. Therefore, only 240 lines have to be TMDS-encoded 60 times per second.

Figure 28: Pseudo-3D in the game *Pilotwings*.

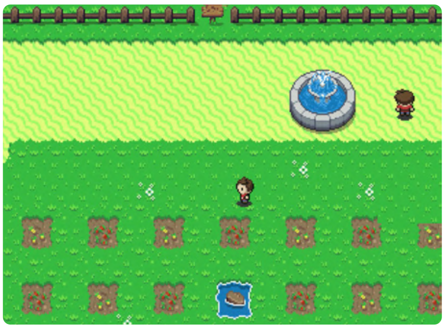


Figure 29: *Walker* demo for the Raspberry Pi Pico.

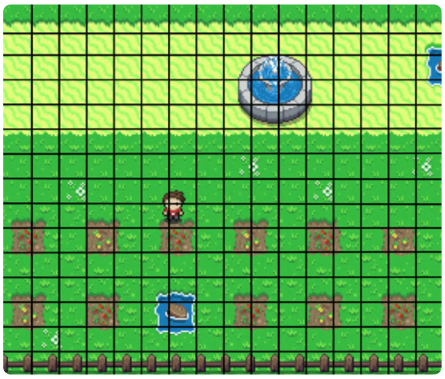


Figure 30: Tiles in the *Walker* demo.

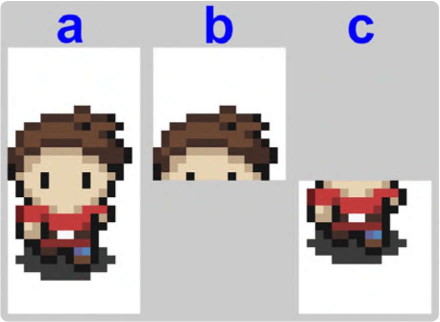


Figure 31: Game character with transparent background (a) as well as upper (b) and lower (c) sprite of the game character.

Each line is output twice, and correspondingly the pixels are twice as wide in the vertical direction.

To achieve this, the raw data is provided line by line, as in the output of a VGA signal. The lines can come from a frame buffer in the Raspberry Pi Pico or can be provided "on the fly." As with VGA, interesting projects are also possible with DVI output. Here, only one of the two CPU cores, together with its interpolator, is busy generating the DVI signal. The *Walker* demo (**Figure 29**) shows impressively that a Raspberry Pi Pico still has enough reserves to conjure up some magic on the screen.

Sprites, Tiles, and Tile Maps with the RP2040

For both VGA and DVI output, there is a graphics library that supports sprites. In the *Walker* demo, the finished image was composed of many small elements called tiles (see **Figure 30**). Each tile has a size of 16×16 pixels. These tiles are intended as background graphics and usually have no transparency information — in contrast to the player character (**Figure 31a**), which is composed of two sprites (**Figure 31b** and **Figure 31c**). Both the background and the sprites can be placed individually. With the appropriate selection, nice backgrounds can be assembled as in **Figure 32**.

The tile maps form the basis for the graphic output here (**Figure 33**), from which the 16×16 pixel tiles are fetched and then composed by the Raspberry Pi Pico to form an image. But, why would one do something like this? Because the nice thing about sprites, tiles, and tile maps is the comparatively low memory consumption and the flexibility of their use.

In addition to the tile map, the only thing that needs to be stored in memory is the position at which the tiles or sprites are to be drawn. Animations or scrolling are also possible

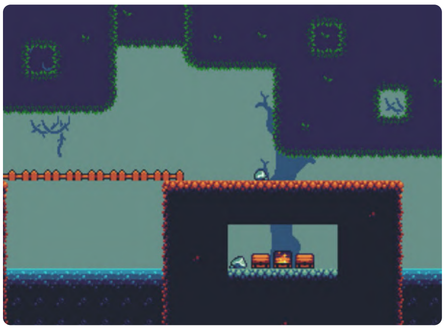


Figure 32: 2D platform demo.

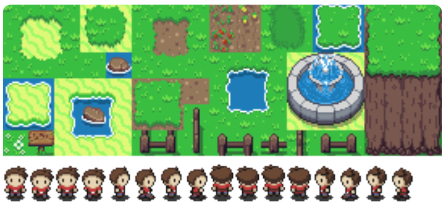


Figure 33: Tile maps (source: ArMM1998 / https://elektor.link/ZeldaTiles).

in this way through comparatively simple updates of sprite positions.

For the demos' source code, please refer to the corresponding GitHub repository [28] by Luke Wren. The images shown here are from the apps *tiles_and_sprites*, *tiles_parallax* and *tiles*. If you have a suitable Raspberry Pi Pico with DVI output, you can try the apps out yourself and play around with the source code.

A Microcontroller as a Drawing Artist?

So far, only demos have been presented — no finished games or applications. However, the demos provide enough information to add graphics to your own projects. Both the ESP32 and the Raspberry Pi Pico are inexpensive boards that allow surprisingly good graphics output. Whether *Pac-Man* or *Tetris* via DVI, and whether VGA or composite video, with each generation, the capabilities and raw performance of MCUs increase. Their hardware allows great effects, such as 3D rendering or moving color images with sprites. Just like back then with the home computers of the 80s, with a lot of creativity and a few tricks, interesting images can be squeezed out of this limited hardware. The microcontrollers just need to be choreographed appropriately when drawing, as are all the ingredients described in this article. It's now up to you to use it to create exciting projects. Post your creations on Elektor Labs [29], and then you can discuss technical problems and features with other Elektor members and get helpful feedback.

220614-01

Questions or Comments?

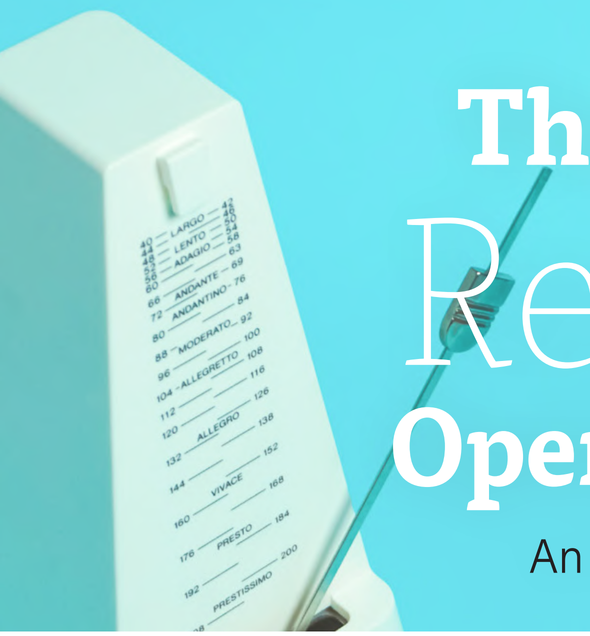
Do you have technical questions or comments about this article? Send an e-mail to the author at mathias.claussen@elektor.com or contact Elektor at editor@elektor.com.

Related Products

- **Pimoroni Raspberry Pi Pico VGA Demo Base (SKU 19769)**
  https://elektor.com/19769
- **Raspberry Pi Pico (SKU 19562)**
  https://elektor.com/19562
- **DVI Sock for Raspberry Pi Pico (SKU 19925)**
  https://elektor.com/19925
- **ESP32-PICO-Kit V4 (SKU 18423)**
  https://elektor.com/18423

WEB LINKS

[1] vgax library: https://github.com/smaffer/vgax
[2] vgaxua library: https://github.com/smaffer/vgaxua
[3] VGA-PacMan [German]: https://niklas-rother.de/artikel/vga-pacman
[4] Atmel ATmega Video generator with SDRAM: http://tinyvga.com/avr-sdram-vga
[5] VGA timing tables: http://tinyvga.com/vga-timing/640x480@60Hz
[6] ESP32 VGA: https://bitluni.net/esp32-vga
[7] ESP32 Technical Reference Manual [PDF]: https://elektor.link/ESP32TechMan
[8] VGA Signal Timing: http://tinyvga.com/vga-timing
[9] ESP32Lib from bitluni: https://github.com/bitluni/ESP32Lib
[10] Sprite (computer graphics) – Wikipedia: https://en.wikipedia.org/wiki/Sprite_(computer_graphics)
[11] FabGL library: http://fabglib.org
[12] FabGL video tutorials [YouTube]: https://elektor.link/YTFabGLTuts
[13] RPI-PVSA VGA, SD Card & Audio Demo Board for Pico – KiCad project [Zip]: https://elektor.link/RP2040VGAKiCad
[14] Hardware design with RP2040 [PDF]: https://elektor.link/RP2040HardwareDesign
[15] Pimoroni Raspberry Pi Pico VGA Demo Base: https://elektor.com/pimoroni-raspberry-pi-pico-vga-demo-base
[16] pico-playground: https://github.com/raspberrypi/pico-playground
[17] pico_scanvideo library: https://elektor.link/pico_scanvideoLib
[18] PicoVGA library: https://github.com/Panda381/PicoVGA
[19] MagPi: VGA Graphics Library for the Raspberry Pi Pico [German]: https://elektor.link/MPPicoVGALib
[20] Mode 7 – Wikipedia: https://en.wikipedia.org/wiki/Mode_7
[21] Interpolator demo – GitHub: https://elektor.link/GHInterperlatorDemo
[22] DVI – Wikipedia: https://elektor.link/WPDVI
[23] DVI Specification [PDF]: https://elektor.link/DVISpec
[24] Transition-minimized differential signaling – Wikipedia: https://elektor.link/WPTMDS
[25] 8b/10b encoding – Wikipedia: https://en.wikipedia.org/wiki/8b/10b_encoding
[26] Luke Wren's PicoDVD: https://github.com/Wren6991/PicoDVI
[27] Mathias Claussen interviews Luke Wren, "DVI with the RP2040," Elektor 3-4/2023: https://elektormagazine.com/220575-01
[28] PicoDVI demos, including tiles_and_sprites "walker" – GitHub: https://elektor.link/GHPicoDVIDemos
[29] Elektor Labs: https://elektormagazine.com/labs



The Metronom Real-Time Operating System

An RTOS for AVR Processors

By Dieter Profos, Dr. sc. techn. ETH (Switzerland)

For many tasks — such as processing continuous signals — microcontrollers have to perform tasks in exact time intervals. The real-time operating system presented here is (also) suitable for AVR controllers with little memory. You have to accept some limitations, such as pure assembler programming, which is still a good compromise for projects where speed and real-time capability are important.

Why Yet Another Operating System?

With the appearance of small and very small processors or controllers, processes became automatable, for which the use of a "real" computer would never have been justified in the past. These microcontrollers do not need to control any peripherals (keyboard, mouse, screen, disk, etc.), so the operating systems can be reduced to the bare essentials of organizing the processing of user programs.

Most operating systems are designed to run as many programs as possible as efficiently as possible and (from the user's point of view) simultaneously. The situation is different, however, where continuous, time-bound signals are to be processed: This requires processes that run at exact intervals. For example, the `delay()` function in Arduino is no longer sufficiently accurate in this case, since it only generates waiting times, but does not take into account the runtimes required for processing, which are clearly noticeable with sampling times of 1 ms or even shorter.

The following two problems must therefore be solved:

- Certain tasks should run exactly at predefined times, others only when there is time left over for them.
- Each interruptible task needs its own stack for buffering the register contents. However, with small controllers the memory space is quite limited: for example, 750 bytes with the ATtiny25 or 1 K with the ATmega8.

The *Metronom* operating system presented here can be downloaded from the Elektor website [1] as open-source software under the BSD-2 license; a release via GitHub is also planned in the coming months.

Cyclic Tasks

Metronom is designed precisely for the execution of what is called *cyclic* tasks at precisely specified time intervals (with up to 8 different cycle times). There is exactly one task per cycle time; if several activities with independent content are to be executed in the same cycle, they are to be combined in the same task.

The cycle times are generated as follows:

- The base cycle time (e.g., 1 ms) is established using the processor's hardware (crystal or internal RC oscillator, hardware- and interrupt-controlled software counter), and
- the other cycle times are generated by a chain of counters, so

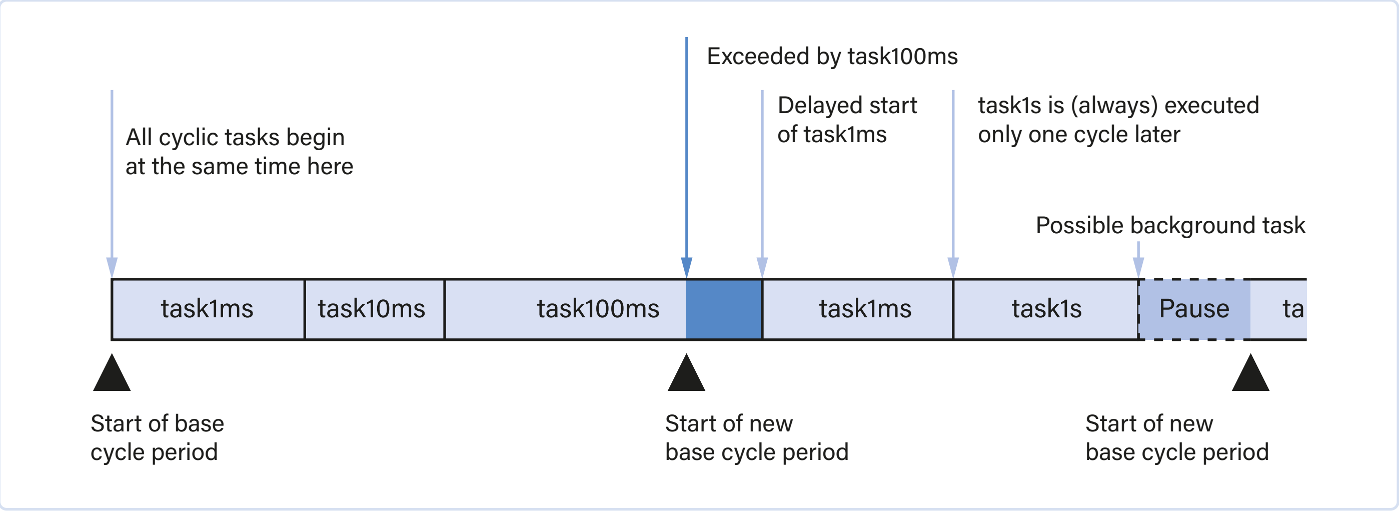

Exceeded by task100ms
Delayed start of task1ms
task1s is (always) executed only one cycle later
All cyclic tasks begin at the same time here
Possible background task
task1ms
task10ms
task100ms
task1ms
task1s
Pause
ta
Start of base cycle period
Start of new base cycle period
Start of new base cycle period

Figure 1: Sequence of several cyclic tasks (extreme case).

that each cycle time is a multiple of the previous one (e.g., the default setting is 1 ms → 10 ms → 100 ms → 1 s).

An important feature of the operating system presented here is that cyclic tasks cannot interrupt each other (*non-preemptive*). On the one hand, this ensures that the timing of these tasks is as precise as possible, and on the other hand, the processor does not lose any "unproductive time" for task switching. And since each cyclic task — once started — runs to completion without interruption (except by interrupts) before the next cyclic task is started, all cyclic tasks can use the same stack together.

However, what happens if a task runs longer than the base cycle time (which actually should be avoided by the programmer) or if several cyclic tasks were started in the same base cycle, where the sum of the runtimes exceeds the base cycle time (which is quite legitimate)? Here another feature of Metronom comes into operation: The cyclic tasks have priorities: The task with the shortest cycle time has the highest priority, the "second fastest" task has the second-highest priority, and so on. If not all cyclic tasks started at the same time can be completed within the basic cycle time, the task currently running is continued to its end after the basic cycle time has elapsed — but then the highest-priority "fastest" task is executed again first, and only after that, other cyclic tasks started previously are executed again.

An example: The implementation of the cycle times causes all cyclic tasks to be started simultaneously once every second. What happens then is shown in **Figure 1**.

This means that, as an absolute upper limit, each cyclic task must not take longer than the shortest cycle time — i.e., 1 ms in our example. This corresponds to about 6000 instructions for an ATtiny running at 8 MHz and about 14000 instructions for an ATmega at 16 MHz (the rest of the instructions are used — on average — by the operating system itself and for the handling of interrupts).

Background Tasks

However, there are certain operations which take longer due to their nature:

- Write access to the EEPROM, for example, takes a few milliseconds (typically about 3.3 ms), i.e., an intolerably long time for a basic cycle of 1 ms.
- Transmitting text at 9600 Bd is not feasible with a basic cycle of 1 ms because even the transmission of a single character already takes more than 1 ms.
- When longer calculations (for example, emulated arithmetic operations!) are to be carried out or character strings are to be processed, this often takes too long within a cyclic task and thus blocks the time-bound processes.

This means that there still needs to be a way to delegate such processes to some type of interruptible task. A combination of two methods is used for this:

- Using interrupts instead of active waiting: This allows the waiting for the end of an operation (e.g., the transmission of a character) to be "delegated" to the hardware; this method is used for interrupt-controlled operations. This solves the problems for a single character transmission or for writing a single value to the EEPROM, but not waiting for the end of the overall operation (e.g., transmitting an entire text).
- Implementation of background tasks: A background task runs only during those times that are not occupied by cyclic tasks. In addition, it can be interrupted at any time, so it does not interfere with the timely execution of the cyclic tasks.

However, once a background task is running, it cannot be interrupted by other background tasks. Thus, only one background task is processed at a time, and if it has to wait, the entire processing of background tasks waits. Although this slows down the processing of the background tasks, it means that only one stack area needs to be reserved for all background tasks.

Background tasks are characterized by the following properties:

- A background task can be interrupted at any time in favor of cyclic tasks, but not in favor of another background task.
- The execution of a background task is triggered by a start call to the dispatcher.
- Background tasks are executed one after the other in the order they were started.

- Background tasks can wait for events (WAIT_EVENT), which are triggered, for example, by interrupt-controlled processes.
- Background tasks can also wait for predefined times (DELAY).
- Each background task can be given a start message of 3 16-bit words, which can be used to specify its task (a 4th word is reserved for the task's start address).
- Any number of background task routines can exist within the user program; however, a maximum of 8 can be started simultaneously.

The coordination of the tasks among each other ("When is which task allowed to run?") is handled by what is called the *dispatcher*. It executes all "administrative processes", such as starting tasks, backup/restore of the processor registers, or disabling/enabling interrupts.

Exceptions

Since microcontrollers usually do not have text-oriented peripherals, debugging is very difficult, especially for time-dependent functions, since breakpoints or the like completely disrupt the timing behavior. Therefore, the operating system kernel provides a simplified mechanism for exception handling, which is divided into two stages:

- A global try-catch area catches all exceptions (exceptions/errors) occurring in the kernel and in the arithmetic emulations. The exception-specific data can be stored in the EEPROM and/or output via USART; after that, the operating system performs a total system RESET (including user-reset). This exception area is always active.
- In addition, an application-oriented try-catch area can be used, which only covers the actual user program. The handling of such exceptions is initially the same as above: The exception data is stored and/or output via USART; then an "application restart" routine to be specified by the user is executed (subroutine user_restart).

Interrupt Handling

Interrupts are handled in four different ways:

- The reset interrupt is used by the operating system and is not directly accessible by the user. However, since the user also needs this interrupt to initialize their own processes, the operating system calls the subroutine user_init after its own initialization, which the user can fill with their application-specific initialization code.
- Timer/counter0 is used for the generation of the basic clock for all cyclic processes; it is therefore not accessible by the user.
- For the use of the EEPROM and the USART, the operating system offers ready-made driver blocks, which can be integrated during the generation of the operating system (see below). However, the user can also couple their own service routines to these interrupts or simply leave them open when not in use.
- All other interrupts are directly available to the user. For each interrupt, an interrupt service routine as well as an interrupt initialization routine must be specified; if more than one interrupt belongs to a device (e.g., timers or USART), a shared initialization routine is sufficient. For this purpose, the user activates the corresponding parameters in the generation file and inserts the contents of the relevant initialization and service routines in his user program.
- Unused interrupts are automatically "intercepted" by the operating system.

Programming Environment

For efficiency reasons, Metronom is written in AVR assembler (Atmel/Microchip) and thus assumes that user programs are also written in assembler; an interface to C was not implemented. Instead, however, there is a library with many subroutines, e.g., for 8-bit arithmetic as well as 16-bit arithmetic (4 basic arithmetic operations); a 16-bit fractional library is in preparation.

To facilitate programming work, all operating system calls are available as macros. To avoid naming collisions, the following naming convention applies: All variables and jump targets within the operating system and libraries start with an underscore ("_") — therefore, all names in the user program should start with letters only. Characters other than letters, numbers and the underscore are not allowed.

The overall structure of Metronom and the corresponding user program can be seen in **Figure 2**.

Operating System Calls

For the complete list of operating system calls, please refer to the references at the end of the article; here is just a rough overview:

Macros for exception handling

- KKTHROW throws a system-wide exception, i.e., after saving/outputting the exception information, the whole system is restarted.
- KTHROW throws an exception limited to the user program, i.e., after saving/outputting the exception information, only the user subroutine user_restart is executed; afterwards the cyclic tasks are restarted.

Macros for using background tasks

- _KSTART_BTASK starts a background task.
- _KDELAY puts the calling background task to sleep for n (0 to 65535) ms.
- _KWAIT puts the calling background task to sleep, from which it can be resumed by means of ...
- _KCONTINUE.

Macros for 8-bit and 16-bit arithmetic

Generally, for arithmetic operations of all kinds, the registers r25:r24 are used as accumulator and r23:r22 as memory for the second operand (if needed). For this purpose, there are more than 20 different functions, such as _mul8u8 for an 8×8-bit multiplication or _abs16 for a 16-bit absolute value. Furthermore, there are many load and save pseudo-codes such as _ld16 (load 16-bit number into accumulator).

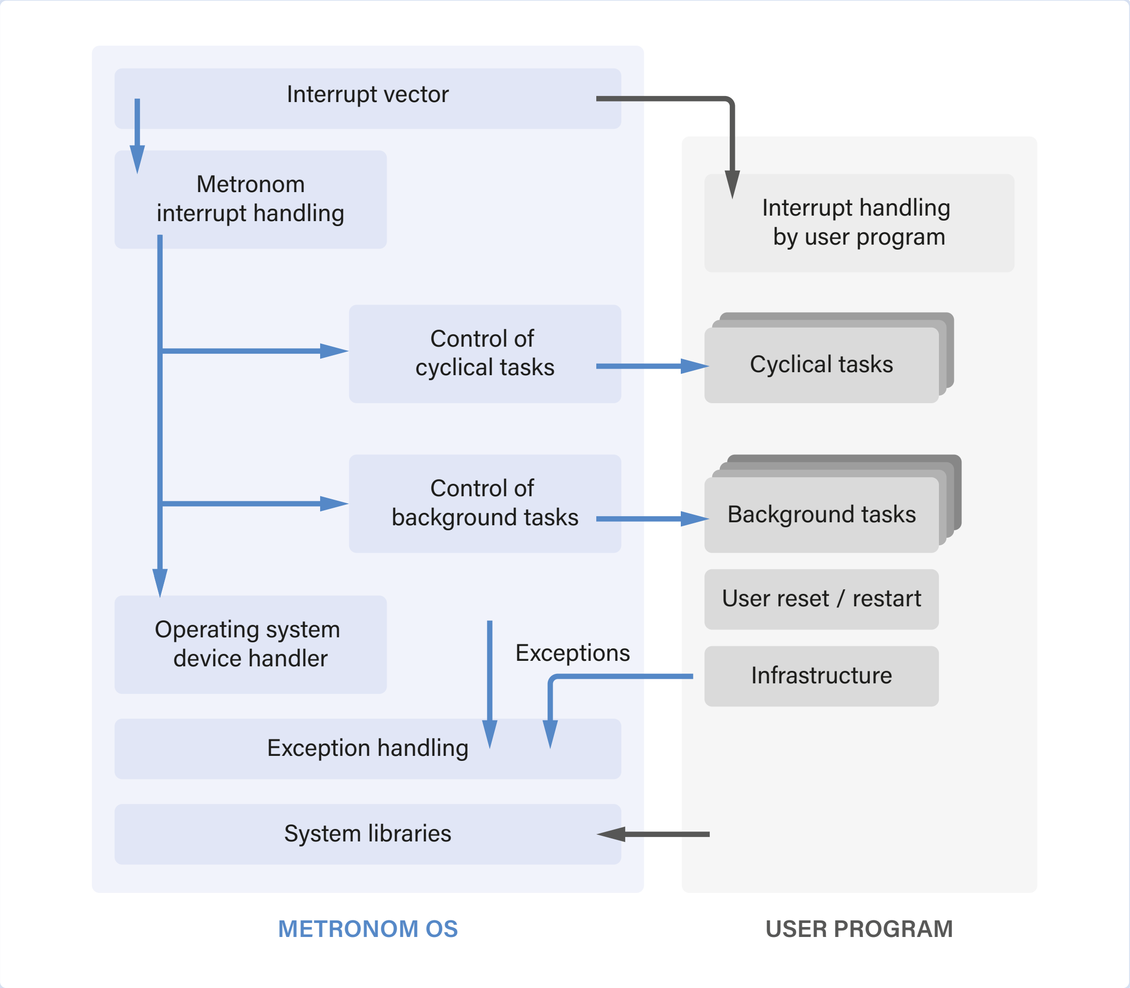


Figure 2: Overall structure of Metronom and the user program.

Macros for EEPROM use

- `_KWRITE_TO_EEPROM` for writing to the EEPROM
- `_KREAD_FROM_EEPROM` for reading from the EEPROM

Macros for USART use

- `_KWRITE_TO_LCD` is a specific USART driver, which adds the necessary control characters for a 2×16 LCD display to the text to be displayed.
- `_KREAD_FROM_USART` (not implemented yet).

System Generator SysGen

For generating a system (i.e., the complete code), a dedicated system generator SysGen is used, which is also part of the overall package. SysGen is not limited to Metronom, but can also be used for general generation tasks.

Some readers may wonder why a separate system generator was developed, considering that there is a wide variety of preprocessors and macro generators. But for the generation of the Metronom operating system, the functionalities of the preprocessors in Atmel Studio as well as standard C are not sufficient. In particular, since the preprocessor does not support string "arithmetic", it is not possible to specify a "default directory" or "library directory" and select files contained in it from there. A search on *Stack Overflow* showed that other people have the same problem as I, but none of today's preprocessors can handle it.

The preprocessor of Atmel Studio (as well as the GNU preprocessor) offers the following functions for assembling the required files:

- `define / set <name> = <numeric_expression>`
- `if … elif … else … endif`, also nested
- `ifdef, ifndef`
- `include <pfad> | exit`

The following functionalities are missing:

- `<path>` can only be passed as a fixed string, but a string expression of (any number of) partial strings, both string variables and string constants, would be necessary.
- `define` and `set` can only assign numeric values, no strings, no concatenation of strings, and also no logical expressions.
- For the (one-time) inclusion of library programs, the macro possibilities given in AVRASM or the C preprocessor are not sufficient; since macros in AVRASM cannot contain *include* statements, the automatic inclusion of emulation routines, for example, is not possible.

This leads to the following scope of functions:

- `define / set <name> = <numeric_expression> | <string expression> | <boolean expression>`
- `if … elif … else … endif`, also nested

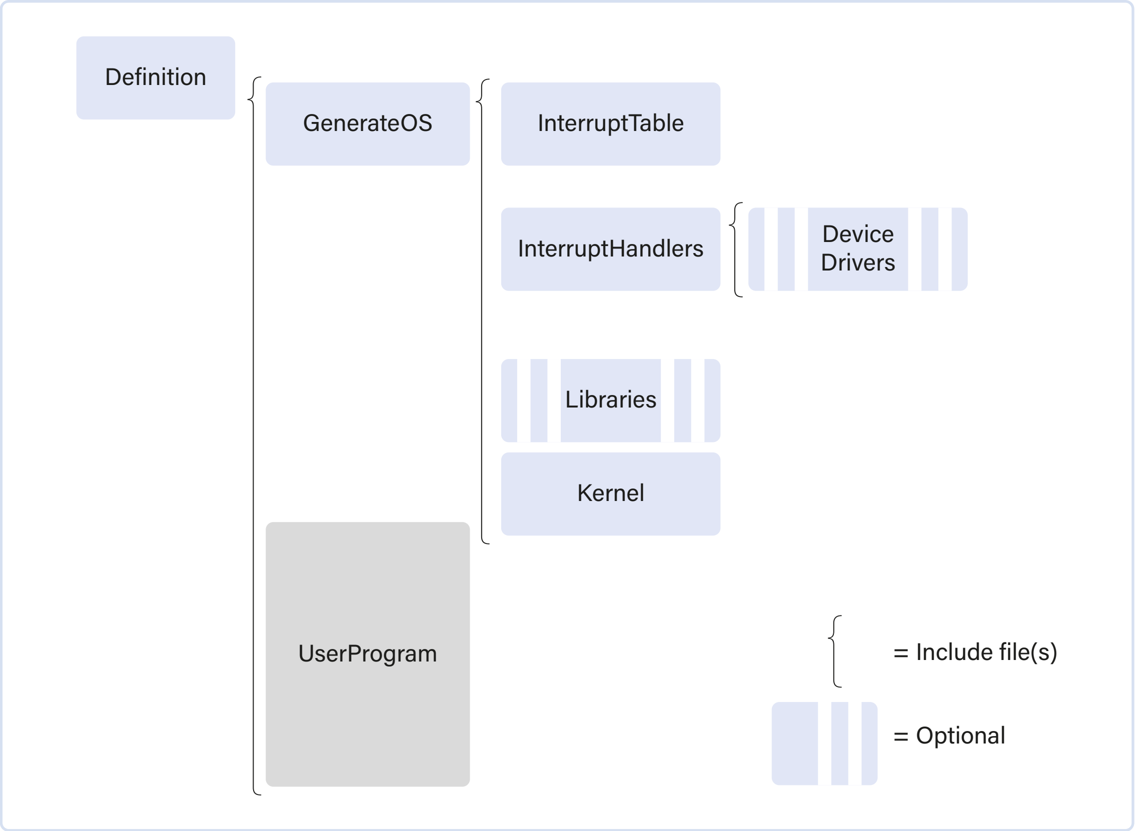


Figure 3: Generation structure of Metronom systems.

- `ifdef, ifndef` is converted into `if isdef(..)` or `! isdef(..)` and can thus also be used within Boolean expressions.
- `include <string ausdruck> | exit`
- `message | error`
- `code` (to create lines of code)
- `macro/endmacro` with suitable parameter labeling
- An additional requirement is that instructions of existing preprocessors can be mixed with those of SysGen without interfering with each other.

The SysGen program can also be downloaded from the specified web page [1]. SysGen is written in Java (version 12) and requires a corresponding Java installation to run.

Programming with Metronom

To save the user the tedious task of working through the operating system source code, the entire operating system is structured in a way to get automatically generated. This means that the user only has to fill in the definition file and — if necessary — the interrupt routines programmed by the user; where they belong and how they are connected is taken care of automatically by the generation process.

In its basic form, a user system is composed of the parts shown in **Figure 3**.

The interrupt table and the kernel are always incorporated as a whole into the resulting overall program. In case of the device handlers and the libraries, on the other hand, only the parts that are actually needed are incorporated.

To illustrate the generation process, the generation script of one of my own projects can be seen in **Listing 1**.

Translated to English by Jörg Starkmuth

210719-01

Questions or Comments?

Do you have technical questions or comments about this article? E-mail the author at profos@rspd.ch or contact Elektor at editor@elektor.com.

WEB LINKS

[1] Generation files for Metronome, generation program SysGen, documentation: www.elektormagazine.com/labs/metronom-a-real-time-operating-system-for-avr-processors

Listing 1

```
; **********************************************************
; Master Definition
; **********************************************************
; Stand: 03.05.2022
;
; This file contains all informations required to generate your user system for
; AVR processors.
; It consists of three parts:
;
; 1. Definitions
;    A bunch of variable definitions defining which functionalities to include.
;    This part must be edited by the user.
;
; 2. An $include statement for the actual generation of the operating system.
;    DO NOT MODIFY THIS STATEMENT!
;
; 3. The $include statement(s) adding the user program(s).
;    This part must be edited by the user.
;

; **********************************************************
; PART 1: DEFINITIONS
;
; This script is valid for ATmega8, ATmega328/P and ATtiny25/45/85 processors.
; If you want to use it for any other processors feel free to adapt it accordingly.

$define processor = "ATmega8"

; Remove the ; in front of the $set directive if you want to use the EEPROM
; $set _GEEPROM=1
; if you want to write your own routines to write to the EEPROM use the following
; definition:
; $set _GEEPROM=2
; Enabling this definition will insert an appropriate JMP instruction to your
; interrupt service routine e_rdy_isr in the InterruptHandlers.asm file
; Remove the ; in front of the $set directive if
; ... you want to output serial data via the USART, or
; ... you want exception messages to be sent outside via the USART
; $set _GUSART=1
; if you want to write your own routines to use the USART
; use the following definition instead
; $set _GUSART=2
; Enabling this definition will enable the interrupt service routines usart_udre_isr,
; usart_rxc_isr and usart_txc_isr in the InterruptHandlers.asm file.

; ----------------------------------------------------------
; Define the division ratios of the time intervals for cyclic tasks
; The definition shown here is the standard preset for 1 : 10 : 100 : 1000 ms
; The first ratio being 0 ends the divider chain.
.equ _KRATIO1 = 10◦◦◦◦◦◦◦◦◦◦◦ ; 1 -> 10ms
.equ _KRATIO2 = 10◦◦◦◦◦◦◦◦◦◦◦ ; 10 -> 100ms
.equ _KRATIO3 = 10◦◦◦◦◦◦◦◦◦◦◦ ; 100ms -> 1s
.equ _KRATIO4 = 0◦◦◦◦◦◦◦◦◦◦◦◦ ; end of divider chain
.equ _KRATIO5 = 0
.equ _KRATIO6 = 0
.equ _KRATIO7 = 0
; NOTE: Do not remove "superfluous" .EQU statements but set them to 0 if not used!

; ----------------------------------------------------------
; Define the constants used for generation of the 1ms timer interrupt
; IMPORTANT: The following definitions depend on the processor being used
; and the frequency of the master clock
```

```
$if (processor == "ATmega8")
; The definitions below are based on a system frequency of 12.288 MHz (crystal)
; This frequency has been chosen in order to use the crystal also for USART@9600 Bd
;
; set prescaler for counter0 to divide by 256, yields 48kHz counting freq for Counter0
.equ _KTCCR0B_SETUP = 4
; Counter0 should divide by 48 in order to produce interrupts every 1ms;
; since counter0 produces an interrupt only at overflow we must preset
; with (256-48) - 1 = 207.
$code ".equ _KTCNT0_SETUP = " + (256 - 48) - 1

$elif ... similar for other processors
;
$endif
;
; ---------------------------------------------------------
; Define the characteristics of USART transmission
; (if you don’t use the USART just neglect these definitions):
$set fOSC = 12288000
$set baud_rate = 9600
$code ".equ _KUBRR_SETUP = " + (fOSC / (16 *baudrate) - 1)

; parity: 0 = Disabled,
; (1 = Reserved), 2 = Enable Even, 3 = Enable Odd
.equ _KPARITY = 0

; stop bits: 0 = 1 stop bit, 1 = 2 stop bits
.equ _KSTOP_BITS = 1

; data bits transferred: 0 = 5-bits, 1 = 6-bits, 2 = 7-bits, 3 = 8-bits, 7 = 9-bits
.equ _KDATA_BITS = 3
;
; ---------------------------------------------------------
; Connect a user defined interrupt handler (except RESET and Timer0)
; by removing the ; in front of the appropriate $set directive;
; don’t change any names but just let the $set statement as is

; Interrupts for ATmega8
; $set _kext_int0 = 1    ; IRQ0 handler
$set _kext_int1 = 1      ; IRQ1 handler/initializer is supplied by user
; $set _ktim2_cmp = 1    ; Timer 2 Compare Handler
; $set _ktim2_ovf = 1    ; Timer 2 Overflow Handler
; $set _ktim1_capt = 1   ; Timer 1 Capture Handler
;
; etc. etc. etc.

;
; *********************************************************
; PART 2: GENERATING THE OPERATING SYSTEM
;
.LISTMAC
;
$include lib_path + "\GenerateOS.asm"
;
;
; *********************************************************
; PART 3: ADD THE USER PROGRAM
;
$include user_path + "\MyApplication.asm"
;
$exit
```

DVI on the RP2040

An Interview with Luke Wren, Chip Developer at Raspberry Pi

By Mathias Claußen (Elektor Lab)

Luke Wren, one of the engineers of the Raspberry Pi RP2040 microcontroller, has achieved some amazing things by pushing this little chip to its limits. Elektor engineer Mathias Claussen wanted to know more, especially about the tricks and hacks required to pull off video output from such a tiny device.

The Raspberry Pi RP2040 is a sub-$1 MCU with amazing capabilities. One of its engineers is Luke Wren, who has not only worked on the Raspberry Pi RP2040, but also has shown that the small RP2040 can do DVI output. (Refer to the article, "Video Output with Microcontrollers (2)" [1]). If he's not working on secret projects for Raspberry Pi, he shares some of his leisure projects with the community on Twitter (@wren6991) and the code on GitHub [2].

Mathias Claussen: Can you tell us a little about yourself?

Luke Wren: Starting with the hardest question! I'm an engineer at Raspberry Pi, and when I'm not doing that, I'm usually working on my hobby projects, playing guitar badly, or, more recently, putting time into language learning. I live in Cambridge, UK, not because it's a particularly exciting city, but more because of inertia after graduating from university. I lived in Germany when I was younger, but my German is pretty rusty these days, so I'm glad we're doing this in English.

Mathias: How long have you been with Raspberry Pi?

Luke: I joined as an employee in September 2018, but I interned at Raspberry Pi previously.

Mathias: What was your role in the development of the RP2040?

Luke: I worked on some of the digital design — mainly PIO, DMA, XIP cache, bus fabric and PWM. I also worked on the boot ROM and the SDK, and, of course, I had to pitch in for the documentation as well.

Mathias: Getting video from an MCU/CPU is something the Sinclair ZX81 could do, but DVI is something new for a sub-$1 MCU. While VGA output can be done by most MCUs, what is the challenge with DVI?

Luke: There are two things that make DVI-D harder than VGA. The first is serializing the data: The minimum pixel clock for DVI-D is 25 MHz, and the bit clock is 10 times that, so, at a minimum, you are driving 3 × 250 Mbps differential serial lines (red/green/blue).

The second is that DVI-D does not just send raw pixel data, but encodes it first. The encoding is simple in hardware, but a bit fiddly in software, especially when that software has to keep up with the raw speed of the serial output.

Everything else is similar. It really is just DPI through a faster pipe. *(Editor's note: Display Pixel Interface is a parallel RGB pixel interface — including a pixel clock — to transport pixel data to a display device)*.

Mathias: What was your motivation to try DVI on the RP2040?

Luke: Once the stress of silicon bring-up had passed, a couple of us wanted to see how high we could push the system clock frequency. There is, in practice, some margin over the 133 MHz rated frequency. I had been playing with DVI-D on FPGA, as part of my RISCBoy project [3], and when I noticed that there was an overlap between the lowest DVI bit clock frequencies and the highest system clock frequencies on RP2040, a lightbulb went on in my head. The motivation was, “I wonder if this is possible.”

Mathias: What was the most challenging part of getting a DVI output (Figure 1) on the RP2040?

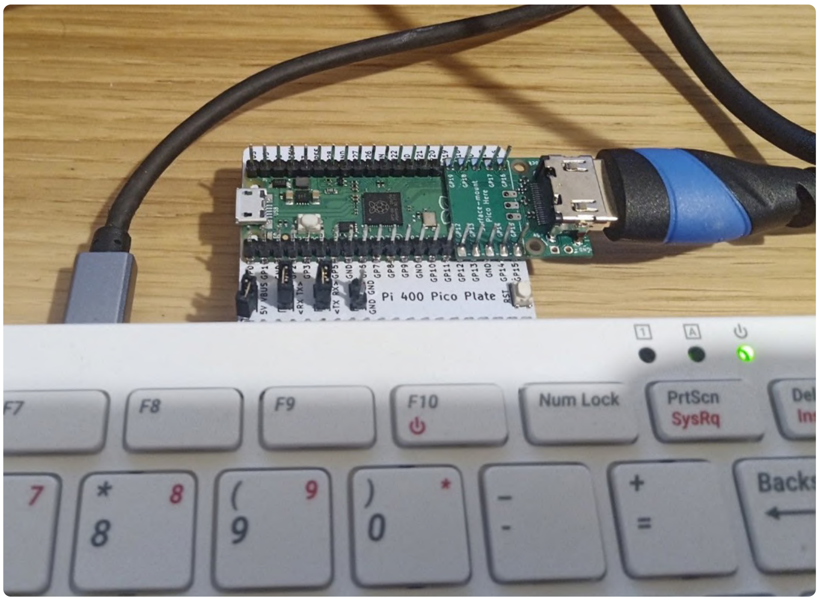
Luke: TMDS encode. If you follow the algorithm in the DVI specification, there is no hope of getting it fast enough on two Cortex-M0+ cores running at the bit clock frequency. So, there are some tricks and shortcuts to make it possible, and then some carefully handwritten code to make it usefully fast. The RP2040 has a lot of memory, but not enough to store a frame’s worth of TMDS-encoded pixels, so you do have to “race the beam” during encoding.

Mathias: You had to overclock the RP2040 slightly (from 133 MHz stock to 252 MHz). Is there a critical path in the chip for the DVI signals (you need to drive I/O pins with speeds that are also faster than stock speed)?

Luke: The first constraint you will hit on the RP2040 is that the system clock has to be 1:1 with the bit clock, so if you try to go to higher-resolution modes, then the processors are just going to crash. The critical setup path for the system clock domain on the RP2040 is processor address-phase signals into SRAMs.

That said, we’re also pretty close to the limits of what you can drive through those general-purpose 3V3 pads; if you look at the eye mask (**Figure 2**) for 720p30 (372 Mbps) on my GitHub [2], it passes, but it’s marginal. I doubt you’d see 1080p30 without dedicated hardware.

Mathias: Besides the speed increase, how crucial are the PIOs and the interpolator inside the RP2040 to get DVI working?



▲

Figure 1: Raspberry Pi Pico with the Pico DVI Sock. (Source: Luke Wren)

Luke: It’s a hard requirement that you need to present 3 serial data bits, plus their differential complements, on GPIOs at 250 Mbps minimum. Being able to do the single-ended to pseudo-differential conversion in the PIO cuts the DMA bandwidth in half, and having the TMDS lanes split into 3 FIFOs is useful if you’re doing the encoding in software because it lets you specialize your code for the encoding of the red/green/blue components. So, something *like* PIO is crucial if you don’t have dedicated hardware.

The interpolators help with address-generation performance in the TMDS encode, which is certainly key to some of the demos you have seen, but my pixel-doubled TMDS-encode trick would still fit onto a single Cortex-M0+ core without the interpolators.

Mathias: Your Pico DVI Sock for the RP2040 uses a physical HDMI connection (Figure 3), clearly labeled as DVI-only, so we won’t get audio. Is it “just” a licensing issue with the HDMI consortium?

Luke: There is nothing stopping you from adding HDMI data islands and doing audio output. In fact, someone has done it with an NES emulator port! [4] [5] There are no extra physical connections required for the audio signals, although, strictly speaking,

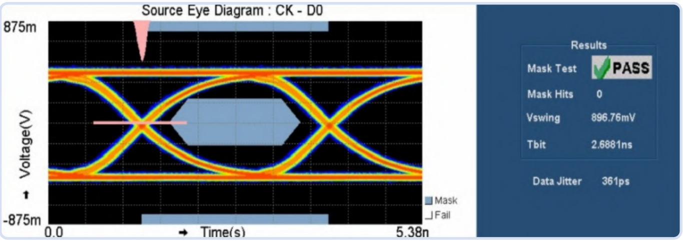
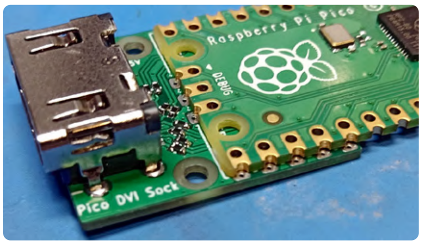


Figure 2: Eye diagram for RP2040 DVI at 720p30. (Source: Luke Wren)



◄

Figure 3: Pico DVI Sock for the Raspberry Pi Pico (Source: Luke Wren).

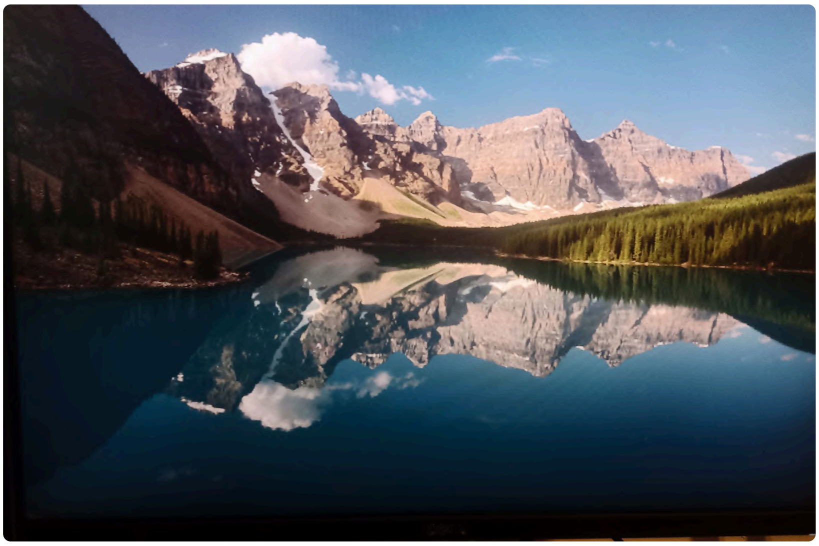


Figure 4: High-resolution images with some tricks. (Source: Luke Wren)

you are not supposed to use HDMI features before interrogating the display data channel, which is not hooked up on my Pico DVI Sock. The HDMI licensing situation is certainly a can of worms I don't want to open, and I also backed myself into a corner by calling the repository "PicoDVI," so I'll leave this one to the community.

Mathias: When using the DVI output, how much of the RP2040 resources are bound to that task? Is there time to spare to run other code on the MCU?

Luke: It depends on the video mode. Say, for pixel-doubled RGB565 output, you end up spending around 65 % of one core for TMDS encoding and DMA interrupts, and the other core is then fully available for generating the video and running your main program logic.

*Editors note: Besides the pure video generation, some applications for the Pico DVI Sock were added. One being the moving several Eben Upton faces around the screen. If we do some math, having a 640×480-pixel image stored as a full frame with 8-bit resolution would take ~308 KB of RAM (more than the RP2040 has), so we fix it at a max of 320×240 with 16-bit color (154 KB) in RAM, but the demo (***Figure 4***) is not pixelated that way. So, there seems to be some software trickery involved.*

Mathias: With the software to generate a DVI signal comes a library that also handles sprites. Can you talk more about it?

Luke: Sure! So, when you write a video output, the very next problem you come across is needing to have some video to actually output, and the ARMv6-M sprite library is something I hacked together whilst working on PicoDVI for exactly that purpose.

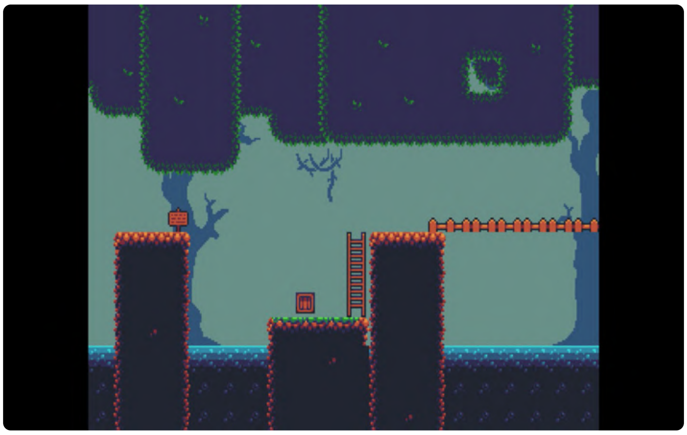
The critical feature of this library is that it does not require a frame buffer to render into, just a scanline buffer. Your rendering races the beam, just ahead of the TMDS encode. This means you can support video output resolutions that would not fit into memory as a flat frame buffer, and it leaves most of your memory free for the actual graphics assets.

There are some reasonably fast blit and fill routines, some tiling routines, and some affine transformed sprite routines that let you do scaled/rotated/sheared sprites. Enough for some games on the level of a Game Boy Advance or so. (*Editor's note: You can see an example in* **Figure 5**.)

Mathias: Where did you get the inspiration for the library — and the time to write it?

Luke: I spent some time working on scanline-based graphics hardware for RISCBoy, so, having done that in hardware, it was fairly easy to replicate it in software. Everything in the PicoDVI repository was done in my spare time on my laptop, except for the eye diagrams, which used a scope at work.

Figure 5: Sprite Demo for the Raspberry Pi RP2040.



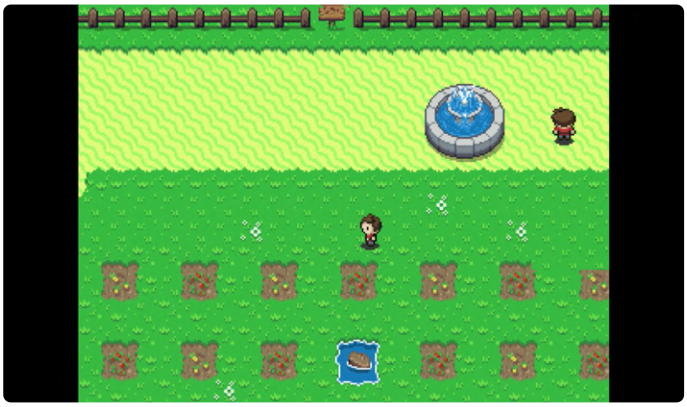


Figure 6: Walker Demo with DVI output.

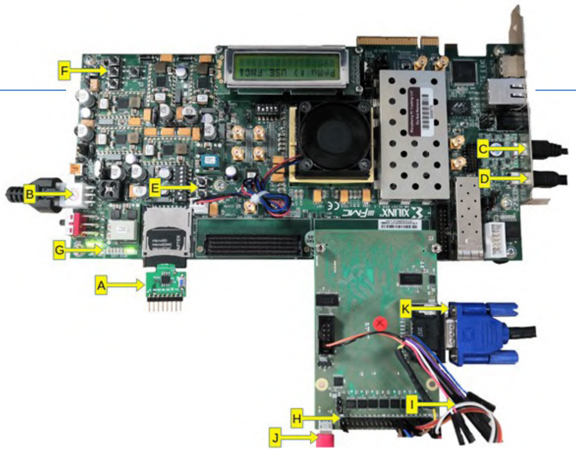


Figure 7: Raspberry Pi RP2040 FPGA prototype. (Source: Luke Wren)

Mathias: There is a Zelda-inspired Sprite demo for the RP2040 (Figure 6). Can you tell us about the idea behind that? An NES or SNES would use dedicated hardware to compose images like that, and here we have just two CPUs moving pixels.

Luke: That is actually a port of one of the RISCBoy demos. Like you say, there is a lot of overhead in doing all of this in software, and RISCBoy running at 36 MHz can put as many sprites on screen as the RP2040 at 252 MHz. [6]

Mathias: In the documents for the library mentioned, there's the idea of a Mario Kart clone for the RP2040? Was it just an idea, or has tinkering started to get it working? Also, there's mention that the interpolator would be useful for it.

Luke: So I have to cough up at this point and admit that wanting to do SNES MODE7-style texture and tile mapping is the original reason for the interpolator being in the chip, although between then and tape-out we spent some time making it a more generally useful and capable piece of hardware.

We've never put together an actual MK clone, although you can see plenty of examples of people using similar techniques online; we did have a texture-mapped 3D cube with Eben's face on it running on FPGA.

Mathias: In the documentation for your Pico DVI Sock for the RP2040 Pico, you mention a 48 MHz FPGA prototype. Can you tell us a little about that prototype?

Luke: We had a nightly job to build an FPGA image from the latest RP2040 source code so that, the next day, we could use it for software development.

We just used an off-the-shelf Virtex 7 development board, with a daughterboard for level-shifting the FPGA IOs up to 3.3 V (**Figure 7**), and another little board that puts a QSPI flash into the SD socket, which is connected to the RP2040 XIP interface [7].

The FPGA build is pretty much a full-featured RP2040 [8] — the clock/reset circuitry is simplified, and the ADC is stubbed out, but, other than that, it is all present and correct. This makes it an ideal platform for software development, although the actual verification of the chip used conventional simulations and formal model-checking.

Mathias: Besides the DVI output, you are/were working on some other projects. One of those is the PicoStation 3D. Can you tell us a little bit about it? If you could source all of the parts, would it still be the same design today?

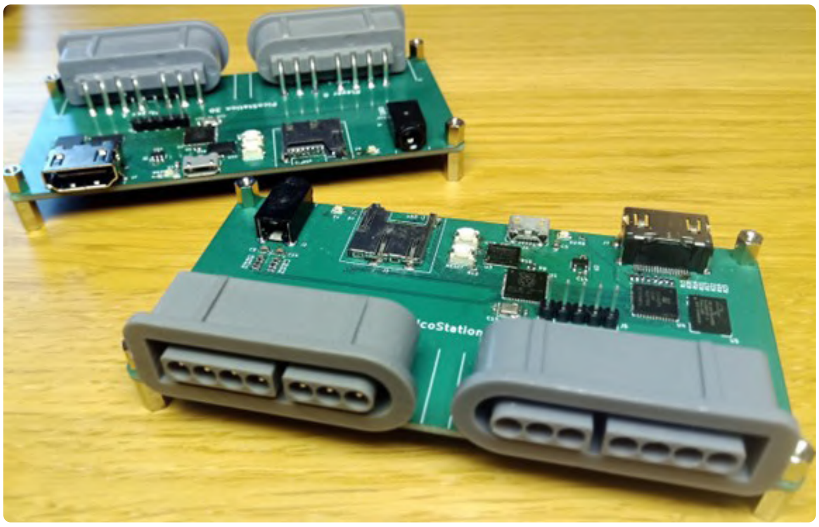
Luke: So, PicoStation 3D (**Figure 8**) is one of the many hobby PCBs I had in flight leading up to the RP2040 launch. It's a board with an RP2040, an iCE40UP5K FPGA, microSD, audio out, DVI-D out via HDMI socket, and two SNES controller sockets. I was reading a lot about 3D graphics hardware at that point and wanted a platform to play with that, in the context of a small games console kind of thing.

It pains me that I have left that project on the back burner for so long, but besides the parts issues, I also just have too many other projects on the go. It's all open-source, so I would love it if someone else picked up the idea and ran with it.

I think the choice of FPGA is just about right — it's small and slow enough to make you work hard for your demos, just barely DVI-D-capable, and it has generous onboard memory and a handful of 16-bit DSP tiles, so it's a brilliant platform to play with toy graphics hardware. It also pairs up well with RP2040. What I would like to spend time thinking about is the

Figure 8: PicoStation3D prototype. (Source: Luke Wren)

▼



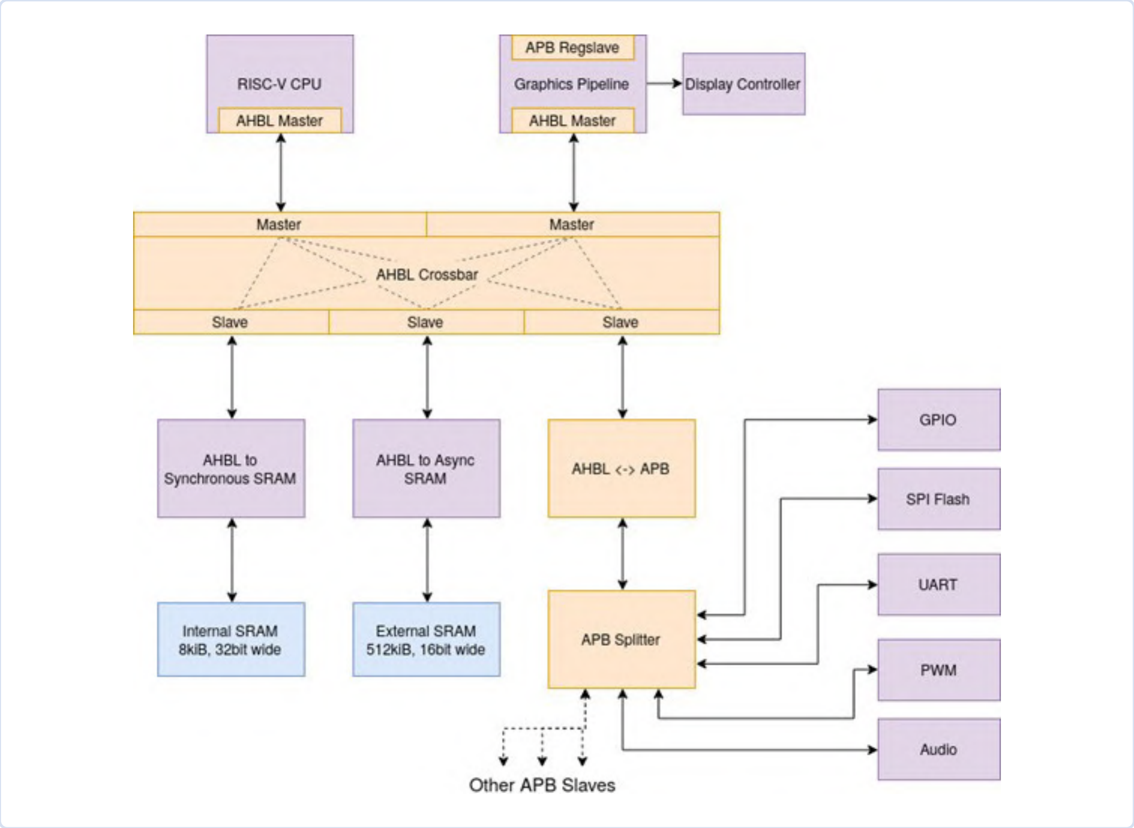


Figure 9: RISCBoy architecture (Source: Luke Wren)

IO. For example, what if you moved the DVI across to the microcontroller and moved the SNES controllers across to the FPGA, and what if you made the audio circuit a little better, that kind of thing.

I think the physical form factor is about right, since it's defined by the two SNES controller connectors.

Mathias: Besides the Raspberry Pi-based items, you also did a RISCBoy, which is powered by a self-developed RISC-V Core and other peripherals, such as a graphics engine (2D-sprite-based?). Can you say some words about this development?

Luke: RISCBoy is my slightly belated competitor to the Game Boy Advance. The full-fat hardware fits into an iCE40 HX8K with some external parallel SRAM (**Figure 9**). Eventually, there will be a physical version, but, right now, it's still an HX8K dev board with some SRAM and buttons, and an SPI display hanging off of it (**Figure 10**). I started working on it around the time I left university.

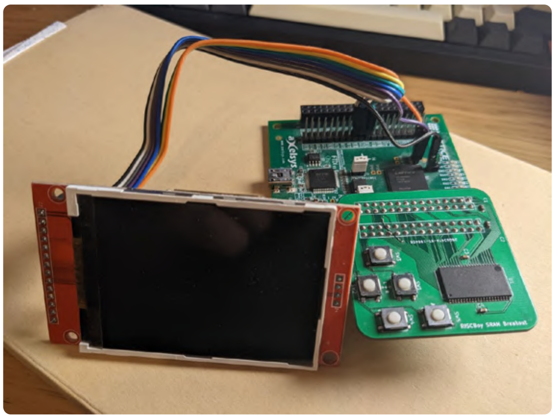


Figure 10: RISCBoy prototype (Source: Luke Wren)

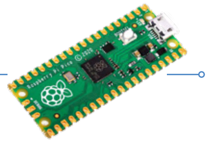
Everything is unapologetically written from scratch — not a good way to do engineering, but a great way to learn, and it makes debugging more fun when there's not a single part of the hardware/software stack that you can trust. There's a 32-bit processor (Hazard5), some programmable 2D graphics hardware, and all the infrastructure to tie it together.

The graphics hardware does all of your usual sprites, tiles, affine-transformed sprites/tiles, and so on, but it does this by executing, from memory, command lists that provide limited support for control flow and branching to subroutines. During each frame, the processor is writing out the command list for the next frame. There are a pair of scanline buffers in hardware, one that is being rendered into, and one that is being sent to the display, so you avoid the bandwidth, latency and memory footprint costs of rendering into a frame buffer.

It's on the back burner for now, but definitely a project I intend to finish one day.

Mathias: Your RISC-V Core becomes maturer by the day. Can you tell us about its origins?

Luke: I have a few processors in various stages of completion at the moment, but I guess you're asking about Hazard3. It's a 3-stage in-order scalar proces-



sor, originally forked from the Hazard5 source (the RISCBoy processor). Really, Hazard3 is a learning platform for me — it's all well and good reading RISC-V specs, but that's not the same as actually going and implementing them, and using a simple 3-stage as the base lets me focus on the periphery because the actual core pipeline just works.

That said, the performance is fairly competitive these days (around 3.2 CoreMark/MHz) and I have sunk a lot of time into verification and documentation. Bringing up hardware debugging, and running end-to-end tests with GDB, OpenOCD, and my own JTAG transport module has probably been a high point of the project so far, but I have been learning a lot all the way through.

In the future, I think Hazard3 will be a goldmine of components for any future 32-bit embedded-class processors I work on. I have already seen someone on Twitter use my Debug Module to add debug support to someone else's core. The source is all self-contained and should be pretty portable. It's also Apache-2.0 licensed.

Mathias: Currently, the design is a 32-Bit RISC-V, which is done fully open-source. Did you get resources for its development from Raspberry Pi (time, hardware, software tools) at some point?

Luke: All of my home projects are done on my own time and hardware, with open-source tools. Right now, I have a Ryzen 7 5800X machine at home, which helps with running batch jobs. I use Yosys for FPGA synthesis, nextpnr for place and route, CXXRTL for simulation.

I use an iCEBreaker (iCE40UP5K) and a ULX3S (ECP5 85F) as reference platforms for Hazard3, although I do have quite an embarrassing number of other FPGA development boards that I ought to use more. When I do have to debug something on FPGA, I get by fine with my Saleae Logic 8 that I bought when I was a student, but most of the time I can rely on simulations.

Mathias: Currently, you are moving from 32-bit to 64-bit with your core. What was the most difficult obstacle during its development so far?

Luke: I've only spent a weekend playing around with RV64 so far, and that was enough to pass RV64IC compliance and the debug compliance tests, so there isn't a huge learning curve — things just get bigger and slower.

I was hacking on the Hazard3 codebase for expediency, but if I wanted to get serious about an RV64 implementation, then there would be some micro-architectural changes required. There's a reason you don't see many 3-stage 64-bit processors, and certainly not ones with the branch decision in stage 2. Hazard3 is going to remain a 32-bit embedded-class processor.

Mathias: Are there any plans to try the core at some point in the future on real silicon? Or even combine the core with a 2D engine to get a RISCBoy64?

Luke: I'd love to try a SkyWater PDK tape out at some point, but, for now, I'm concentrating on other projects. I want to build something more interesting than "just another RISC-V system" and I'm not quite sure what that will be yet. For something like RISCBoy, there is not much benefit to using a 64-bit processor.

Mathias: Thank you for your time, Luke Wren.

220575-01

WEB LINKS

[1] Mathias Claußen, "Video Output with Microcontrollers (2)," ElektorMag 3-4/2023: http://www.elektormagazine.com/220614-01
[2] Luke Wren's GitHub Repository: https://github.com/Wren6991
[3] RISCBoy @ GitHub: https://github.com/Wren6991/RISCBoy
[4] pico-infones @ GitHub: https://github.com/shuichitakano/pico-infones
[5] Video @ Twitter: https://twitter.com/shuichi_takano/status/1477702448907419649
[6] Sprite demo running on the RISCBoy hardware at 36 MHz: https://twitter.com/wren6991/status/1333708886956707840
[7] Picture of the QSPI SD board: https://twitter.com/wren6991/status/1134719550027632640
[8] RP2040 Microcontroller: https://raspberrypi.com/products/rp2040/

Elektor lab • Elektor lab TESTED Elektor lab • Elektor lab

Display HAT Mini

Show the Weather Forecast on Raspberry Pi!

By Clemens Valens (Elektor)

The Display HAT Mini from Pimoroni is equiped with a rectangular 2.0″ 320-by-240-pixel LCD, four tactile buttons and an RGB LED. Intended for the Raspberry Pi Zero and Zero 2 W it is great for many IoT and home automation applications.

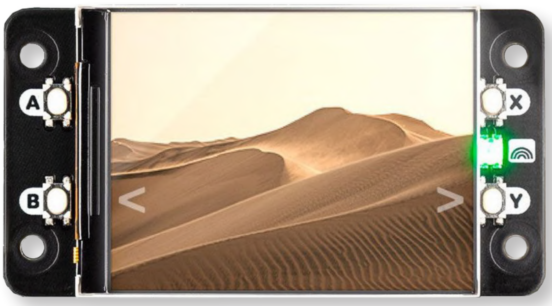


Figure 1: The Display HAT Mini also has four tactile buttons and an RGB LED.

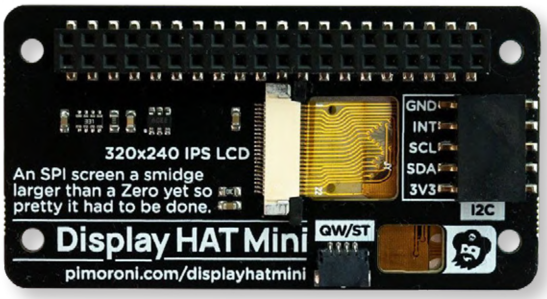

320x240 IPS LCD
An SPI screen a smidge larger than a Zero yet so pretty it had to be done.
GND
INT
SCL
SDA
3V3
I2C
QW/ST
Display HAT Mini
pimoroni.com/displayhatmini

Figure 2: I^{2}C extension connectors are available on the rear side.

The Display HAT Mini from Pimoroni [1] is equiped with a rectangular 2.0″ diameter IPS (In-Plane Switching) LCD with SPI interface. It is intended for the Raspberry Pi Zero and Zero 2 W but, as it has the standard 40-pin HAT connector, it can be plugged onto any Raspberry Pi equipped with such a connector. However, you have to be careful as the HAT's I^{2}C connector ('Breakout Garden header') collides with the Raspberry Pi's Display connector.

Specifications

The display's resolution is 320 by 240 pixels (3:2), which corresponds to about 200 ppi (pixels per inch). It has 65K color depth. Mounted on a Raspberry Pi Zero without stand-offs, the total height is approx. 15 mm. The display comes with two 10-mm-high stand-offs and four little screws (five in my case). To me these stand-offs are too high, 8.5 mm would have been much better (but non-standard).

Besides the display the HAT also features four tactile buttons and an RGB LED (**Figure 1**). The push buttons are placed very close to the display making them a bit difficult to use. I suspect that they were placed that way to provide some extra mechanical support for the display.

Even though the HAT blocks access to the 40-pin HAT connector, extensions are still possible thanks to either the Qw/ST (Qwiic/STEMMA QT) connector and the so-called Breakout Garden header. Both headers provide access to the I^{2}C bus (**Figure 2**).

Supported by Python Libraries

To use the Display HAT Mini on a Raspberry Pi you must install a library. Detailed instructions on how to do this are given in the manufacturer's GitHub corner [2]. Examples for using the display with Pygame and PIL are provided too.

I plugged the Display Mini HAT on a Raspberry Pi Zero 2 W running Buster and controlled it over SSH. After installing the libraries, all the examples worked without a hitch.

Let's Make a Weather Forecast Display

With the display up and running, I wanted to do something with it. On the Internet I had found a collection of nice-looking weather icons, and so I decided to create a weather forecast display in Python 3 showing the corresponding weather icon together with temperature, atmospheric pressure, humidity and wind direction and speed (**Figure 3**). This data can be obtained from an online weather forecast server, there are plenty of those that offer free access.

I used the push buttons *A* and *B* to control the brightness of the backlight of the display. Button *X* lets you choose between degrees Celsius and degrees Fahrenheit while the *Y* button toggles between wind direction in degrees or as cardinal letters (e.g., 'SW' or 'N'). The RGB LED provides some status information. Initially green should indicate proper operation, but I found the brightness too high even at very low values, so I switched it off. Now the LED will only light up in case of a problem: red when the weather data could not be fetched and orange when the weather data is invalid.

Note that by default the screen is upside-down with respect to the push button print on the HAT. For this reason, the program rotates the display buffer by 180° before copying it to the screen.

The current consumption of my system at maximum backlight intensity was around 200 mA.

My code (including the complete icon collection) can be found at [3].

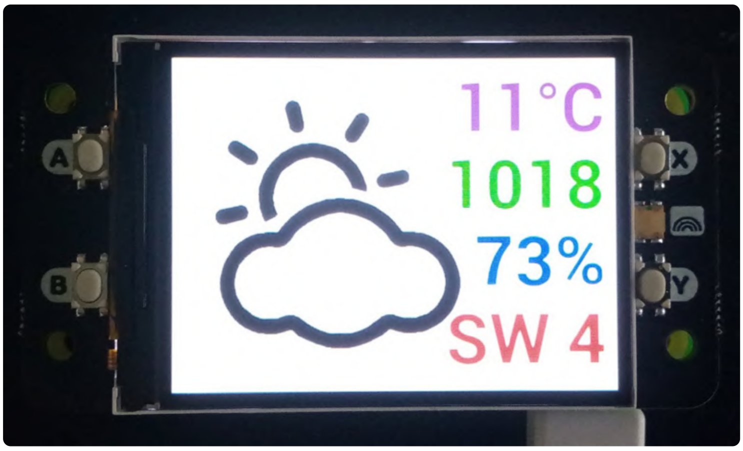


Figure 3: The Display HAT Mini makes for a great IoT or home automation display.

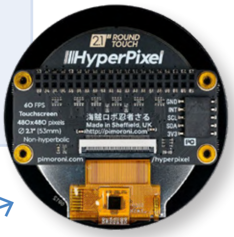
A Cool Add-On

The Display HAT Mini is a cool add-on for a Raspberry Pi Zero (2 W). The image quality is great, and it is easy to use in your own applications. The supporting library also provides access to the push buttons (placed a bit too close to the display for my taste), RGB LED (a tad on the bright side) and backlight. Note that due to the display the HAT is slightly larger than a Raspberry Pi Zero (2 W), 35 mm instead of 30 mm, so choose your enclosure wisely.

220126-01

Questions or Comments?

Do you have technical questions or comments about this article? Email the author at clemens.valens@elektor.com or contact Elektor at editor@elektor.com.



WEB LINKS

[1] Display HAT Mini from Pimoroni: https://www.elektor.com/pimoroni-display-hat-mini-for-raspberry-pi-zero
[2] Pimoroni at GitHub: https://github.com/pimoroni/displayhatmini-python
[3] ClemensAtElektor at GitHub: https://github.com/ClemensAtElektor/rpi_weather_display

WEEF 2022 Awards Celebrate the Good

By Priscilla Haring-Kuipers (The Netherlands)

On November 15, 2022, at the electronica fair in Munich, the second annual World Ethical Electronics Forum (WEEF) featured in-depth talks and debates on a range of electronics-related ethical issues. Join us in celebrating the four WEEF Awards recipients.

At our second World Ethical Electronics Forum (WEEF) in Munich on November 15, 2022, we handed out four awards to some very impressive people. From our WEEF Index, our jury selected four winners by looking at their contribution to ethical electronics on three dimensions:

1) Their level of influence
2) Their level of innovation
3) Their willingness to share

Our World Ethical Electronics Forum Index is filled with people that we at WEEF know are doing their very best for ethics in electronics **and** with people that were nominated by our members and people in the industry. If you know someone that should be included in this collection of good people (and should be in the running for the next WEEF awards), you can nominate them by filling out this application form on the WEEF website.[1]

WEEF WINNER Sven Krumpel

Sven Krumpel [2] is the owner of CODICO, which has a great emphasis on work-life-balance for their employees. They offer nearly full flexibility in work place and work time and are a great example of really investing in your employees.

“Thank you for nominating me. I didn’t know I was an ethical person. We are a family company; we design and move forward electronics. We see how important it is to develop projects in energy efficiency. I think ethics is a people thing; you as a person are an example. As a company owner you do not only have employees, but you have people with the same values. Our people are part of the family. We just opened a 12,000-meter park next to our offices, not just for working outside but also for leisure time. Our people do sports in the park, invite their families and have birthday parties. We grow vegetables and herbs. We have bees. I think we have been living an extremely blessed life where we used too much resources and used other peoples’ resources and we need to give some back. This is what we are doing and it’s nothing special.”

WEEF WINNER Gopal Kumar Mohoto

Gopal Kumar Mohoto [3] is an engineer working on electrifying transport in Bangladesh. He was chosen the Global No-1 entrepreneur at the ClimateLaunchpad of 2020 and was a Climate Ambassador at the Global Youth Climate Network. He has worked on battery swapping stations for electric tuk-tuks and is about to release the first cluster of retrofitted gasoline-to-electric cars to clients in Bangladesh. You can read an interview with him here [4].

“I am very honoured to be nominated, although I don’t know by who. The options for electric driving available in the West are too expensive in the Global South. This is why I am working on a sustainable and affordable option to electrify transport here. We must take better care of the things we already have and use them as they are precious. We must take care of all the materials we mined. We must take care of Mother Earth.”

WEEF WINNER Frank Stührenberg

Frank Stührenberg [5] is the CEO of Phoenix Contact and a board member of the KlimaWirtschaft Foundation. He is a major contributor to the All Electric Society, which is a world envisioned where regeneratively generated electrical energy is available worldwide as a primary form of energy in sufficient quantities for everyone.

”I am not aware who nominated us. The awards our company gets are usually technical, so I was surprised by this one. The tone of WEEF fits to our company. We are family-owned and now have more than 20,000 family members. We have to work on not losing our moral compass and we are actively working on this. We dedicated our long-term strategy to go for 100% renewable electrical energy. We felt we should do this and so we started. We need energy to

Milda Pladaitė (left) receives an award from Elektor's Beatriz Souza. (Photo Messe München)

Sven Krumpel (right) receives an award from Johann Wiesböck of ELEKTRONIKPRAXIS. (Photo Messe München)

Gopal Kumar Mohoto.

Frank Stührenberg (right, with CEO of Messe München Reinhard Pfeiffer, Photo Messe München).

develop the world, and what is the purpose of being a €10 billion company while not having a world we want to live in? Thank you for supporting this development and supporting this initiative."

WEEF WINNER Milda Pladaitė

Milda Pladaitė [6] is a civil engineer from Lithuania and part of the World Federation of Engineering Organizations (WFEO) Young Engineers Future Leaders Committee. She set up a working group on Sustainable Development Goal 13 at the WFEO aiming to contribute to the sustainable development of countries by deploying WFEO young engineers' work in climate action. Currently, she is working on a social initiative, helping engineers to be more entrepreneurial by connecting them with industry, social investors and academic organisations.

"I have nominated many people for awards, but I don't know who nominated me for this. A friend said when she heard I was nominated for this: 'I didn't know you were a specialist in ethics!' Of course, this means that everyone can be into ethics. This forum is a great initiative, and I am sure it will be growing in the future. This is a unique opportunity to discuss ethics in electronics."

We can see by the surprise of our WEEF award winners that it is not always obvious what is ethical in electronics. We are perhaps reluctant to talk about ethics as an industry while we don't have to be. The aim of our World Ethical Electronics Forum is to provide a platform to discuss what we do and why we do this, to share stories and practices with each other, and to celebrate the good together. ◀

220650-01

About the Author

Priscilla Haring-Kuipers writes about technology from a social science perspective. She is especially interested in technology supporting the good in humanity and a firm believer in effect research. She has an MSc in Media Psychology and makes This Is Not Rocket Science happen.

The World Ethical Electronics Forum

The World Ethical Electronics Forum (WEEF) inspires global innovators with open discussions and publications about ethics and sustainable development goals. Visit **worldethicalelectronicsforum.com** for inspiration and to participate.

WEB LINKS

[1] WEEF Index Nomination Form: https://worldethicalelectronicsforum.com/nomination-form

[2] WEEF, Sven Krumpel: https://worldethicalelectronicsforum.com/index/184374/Sven__Krumpel

[3] WEEF, Gopal Kumar Mohoto: https://worldethicalelectronicsforum.com/index/182949/Gopal_Kumar_Mohoto

[4] P. Haring-Kuipers, "Retrofitting and Upcycling: Interview with Gopal Kumar Mohoto," ElektorMagazine.com, 2022: https://www.elektormagazine.com/gopal-interview-for-next-eia

[5] WEEF, Frank Stührenberg: https://worldethicalelectronicsforum.com/index/184378/Frank__St%C3%BChrenberg

[6] WEEF, Milda Pladaitė: https://worldethicalelectronicsforum.com/index/182955/Milda__Pladait%C4%97

The Elektor Store

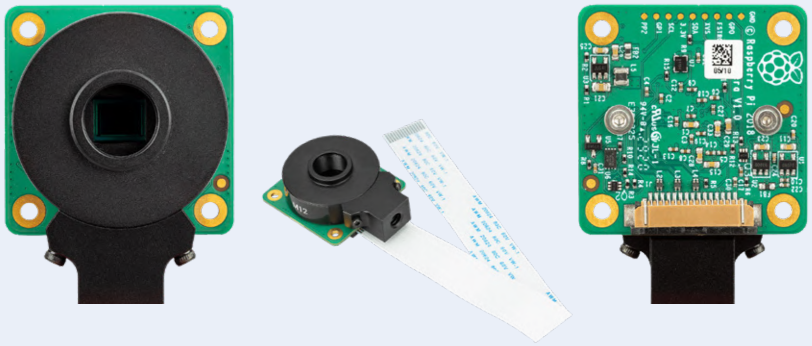
Never expensive, always surprising

The Elektor Store has developed from the community store for Elektor's own products like books, magazines, kits and modules, into a mature webshop 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). Our main conditions: **never expensive, always surprising**!

Raspberry Pi High Quality Camera Module (M12 Mount)

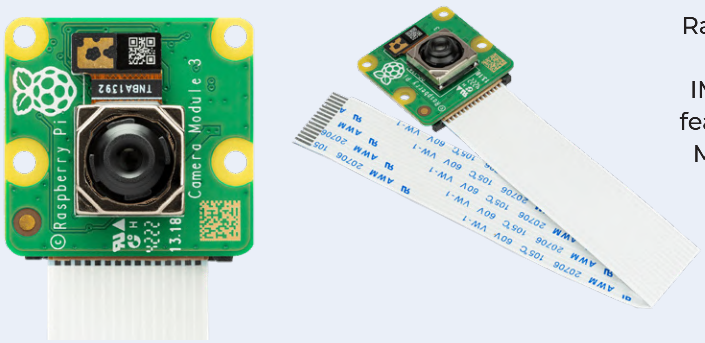
The Raspberry Pi High Quality Camera offers 12-megapixel resolution and a 7.9-mm diagonal sensor for impressive low-light performance.

Price: €59.95



www.elektor.com/20366

Raspberry Pi Camera Module 3



Raspberry Pi Camera Module 3 is a compact camera from Raspberry Pi. It offers an IMX708 12-megapixel sensor with HDR, and features phase detection autofocus. Camera Module 3 is available in standard and wide-angle variants, both of which are available with or without an infrared cut filter.

Price: €32.95

www.elektor.com/20362

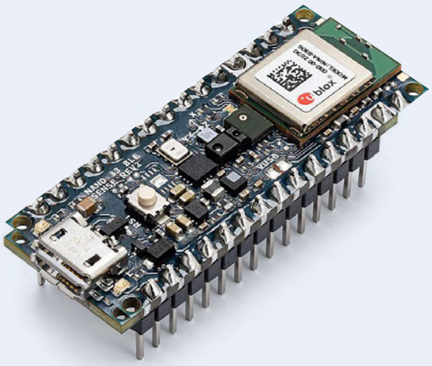
YDLIDAR OS30A 3D Depth Camera

Price: €139.95

Member Price: €125.96

www.elektor.com/20350

Arduino Nano 33 BLE Sense Rev2 with Headers



Price: €54.95

Member Price: €49.46

www.elektor.com/20404

Arduino Student Kit

Price: €74.95

Member Price: €67.46

www.elektor.com/20329

Home Appliance Hack-and-IoT Guidebook

+
FREE ESP8266 Board
elektor books
Home Appliance Hack-and-IoT Guidebook
Affordable solutions with the ESP8266 and 3D printing
Hans Henrik Skovgaard
elektor

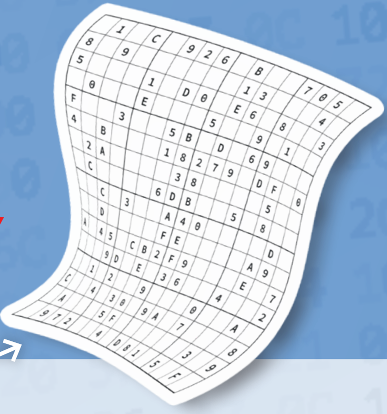
Price: ~~€54.90~~

Special Price: €39.95

www.elektor.com/20370

Hexadoku

Puzzles with an Electronic Touch



Traditionally, the last page of *Elektor Magazine* is reserved for our puzzle with an electronics slant: Welcome to Hexadoku! Find the solution in the gray boxes, submit it to us by email, and you automatically enter the prize draw for one of five Elektor Store vouchers.

The Hexadoku puzzle employs digits in the hexadecimal range 0 through F. In the diagram composed of 16×16 boxes, enter digits such that **all** hexadecimal digits (that's 0–9 and A–F) occur once only in each row, once in each column, and in each of the 4×4 boxes (marked by the thicker black lines). A number of clues are given in the puzzle, and these determine the starting situation.

Correct entries received enter a prize draw. All you need to do is send us **the digits in the gray boxes**.

SOLVE HEXADOKU AND WIN!

Correct solutions received from the entire Elektor readership automatically enter a prize draw for five Elektor store vouchers worth **€50 each**, which should encourage all Elektor readers to participate.

PARTICIPATE!

By April 15th, 2023, supply your name, street address and the solution (the digits in the gray boxes) by email to: **hexadoku@elektor.com**

PRIZE WINNERS

The solution of the Hexadoku in edition 1-2/2023 (January & February) is: **AEF1C**.
Solutions submitted to us before February 15th were entered in a prize draw for 5 Elektor Store Vouchers.
The winners are posted at elektormagazine.com/hexadoku.

Congratulations, everyone!

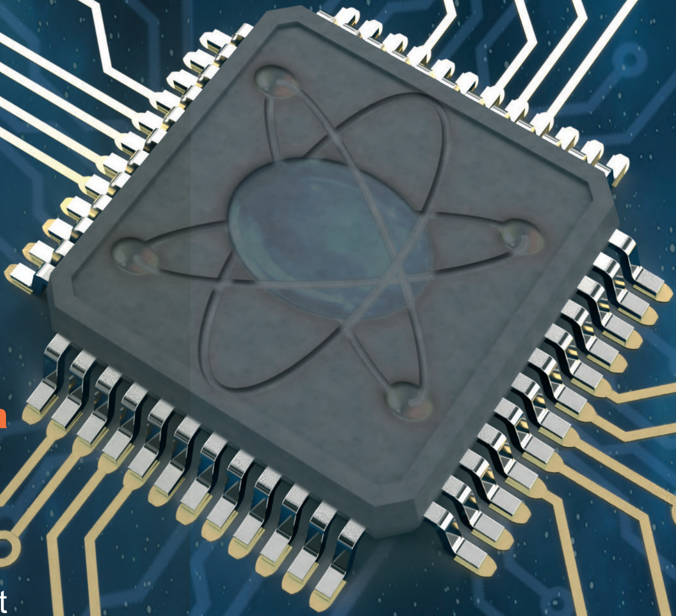
	1	3	F									E	2	7	
0	6	5				▨	▨	▨	▨	▨			8	C	D
	8				2					E				6	
9	B	D	E	4		7			C		2	F	3	A	5
F	7	E	8			D			B			A	6	5	1
	9	0		6	1					3	F		4	B	
	D	4			F					9			7	8	
1		6	3	2			B	D			8	9	E		F
			6		8	0			D	2		5			
	E	A			D	9	2	8	5	F			0	1	
5				1			E	4			9				8
	0		9		5		4	A		1		C		F	
D						1			8						E
		9		8	7					D	A		5		
	A		1		9	3			2	7		8		D	
7	5					4	F	E	3					9	2

E	5	2	A	4	B	8	C	6	3	F	0	D	7	9	1
0	3	B	4	6	F	2	7	D	9	1	A	E	5	8	C
C	F	1	7	D	E	9	A	B	4	5	8	2	0	3	6
6	D	8	9	0	1	3	5	C	7	E	2	B	A	F	4
1	8	F	3	5	7	4	0	E	B	2	6	9	D	C	A
B	4	C	D	3	2	6	9	8	0	A	5	F	E	1	7
5	A	E	2	8	C	D	1	7	F	9	4	6	B	0	3
7	0	9	6	B	A	E	F	1	C	3	D	4	2	5	8
F	9	7	B	E	6	5	4	0	8	D	C	1	3	A	2
3	E	5	C	7	8	A	2	F	1	B	9	0	4	6	D
D	2	0	8	9	3	1	B	4	A	6	E	C	F	7	5
4	6	A	1	C	0	F	D	5	2	7	3	8	9	B	E
8	C	3	F	1	5	B	E	9	D	4	7	A	6	2	0
9	1	4	0	2	D	7	6	A	5	C	B	3	8	E	F
A	B	6	5	F	4	0	3	2	E	8	1	7	C	D	9
2	7	D	E	A	9	C	8	3	6	0	F	5	1	4	B

The competition is not open to employees of Elektor International Media, its subsidiaries, licensees and/or associated publishing houses.

PROTEUS DESIGN SUITE

Design Quality Assurance



Constraint Driven Design

Flexible and scalable rule system

Full support for design rule rooms

Manufacturing solder mask rules

Live display of violation areas

Zone Inspector

Analyze plane coverage and stitching

Grid view of plane configurations

Edit plane settings and draw order

Dedicated Reporting Module

Tables automatically populate with design data

Compliance status for diff pairs and length matched routes

Make custom reports with data object tables

Generate reports from templates

Pre-Production Checklist

Set of board tests before Gerber Output

Includes placement, connectivity and clearance testing

Completely independant code for clearance checks

labcenter Electronics

www.labcenter.com

info@labcenter.com

☎ +44 (0)1756 753440