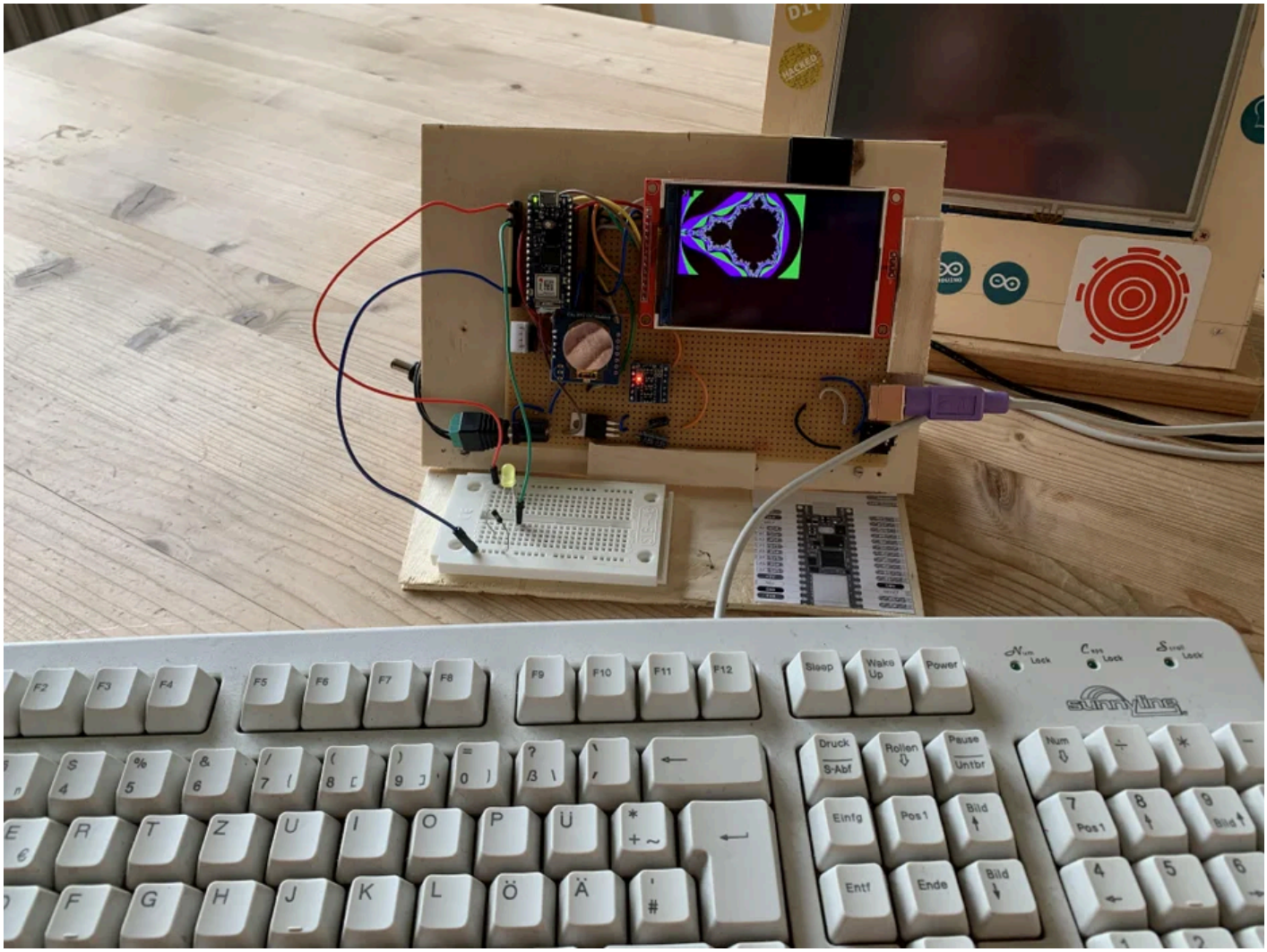
AUTODESK Instructables

A Arduino RP2040 Standalone IoT Computer Running BASIC.

By slviajero in CircuitsArduino

Introduction: A Arduino RP2040 Standalone IoT Computer Running BASIC.



This instructable is about building standalone computers running a BASIC dialect suitable for IoT and interactive work. It uses low cost components that are readily available and can be integrated without much difficulty.

So why running BASIC on a microcontroller? Everyone who has programmed on 80s microcontrollers misses one thing on modern computer. It is the ease of use when you want to do something quickly. You could just type in a small program interactively, debug it and step by step extend it. No compiler, IDE, and other stuff in your way. Many IoT programs are really simple. Read a sensor and transfer the data. This can very well done with a really simple programming language.

The computer which is the main character in this story is based on an Arduino RP2040 connect. It has an SD card filesystem, and a 480x320 colour display with 30x20 text character and a 16x16 default font. It has a real time clock for exact time and a PS2 keyboard for input. A thermo printer

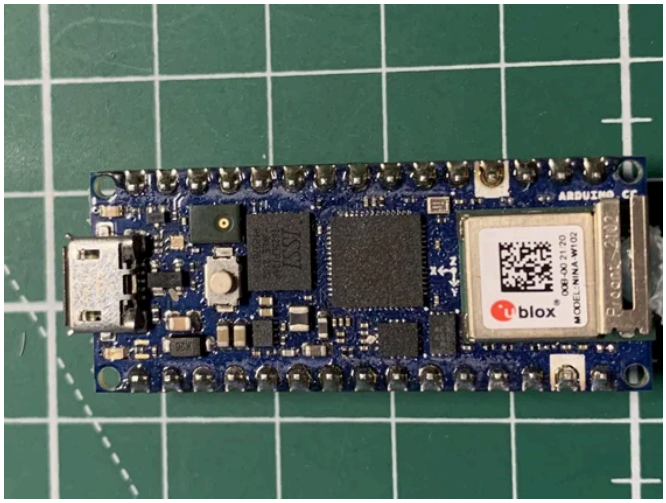
can be connected serially. Any 7-12V power supply can be used as the board has a 5V voltage regulator. There is 64kB of usable BASIC memory and 2 GB disk space.

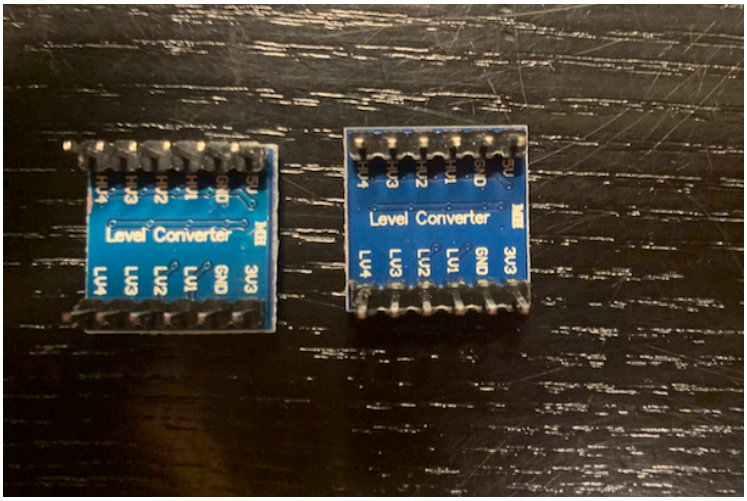
The BASIC interpreter is a full featured language with strings, graphics, floating point support, Arduino I/O, and a few other useful features.

The computer can be build on a large breadboard almost without any soldering (except for the PS2 plug). Preferably a 10x15 cm circuit board is used for a real stable hardware setup.

Alternatively an Arduino MKR or an ESP32 can be used for a smaller version of the computer.

Supplies





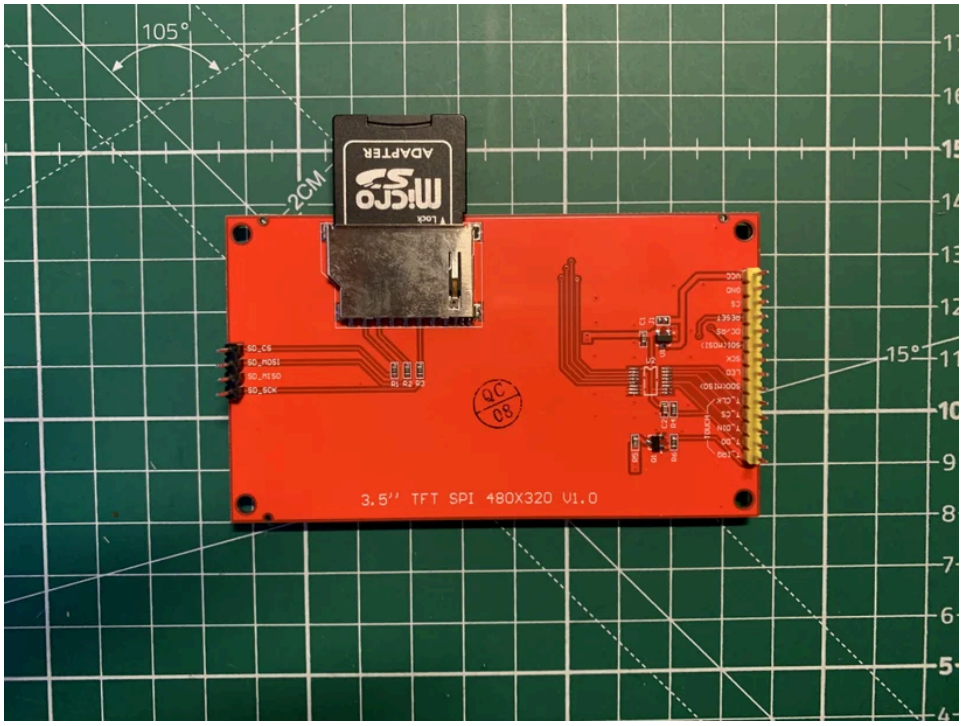
Core components of the computer are

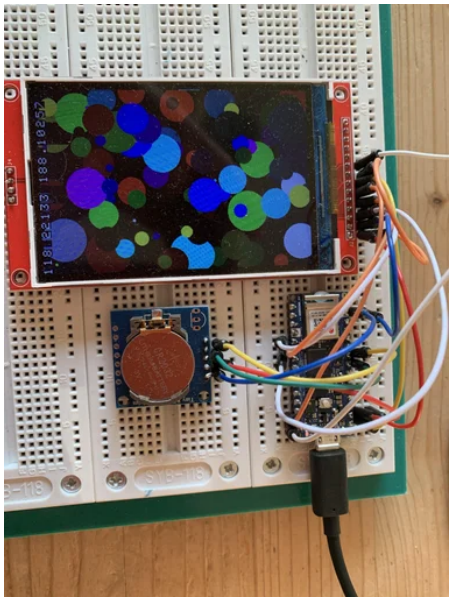
- Arduino RP2040 connect microcontroller board (https://store.arduino.cc/products/arduino-nano-rp2040-connect)
- ILI9488 based display shield. These shields are SPI based, have an SD card slot, and an optionally touch input which will not be needed here. The shields are very common. Please make sure to buy the SPI based ones for this hardware project.
- A voltage level converter.
- An old PS2 keyboard and a PS2 plug.
- Either a large breadboard or preferably a 10x15 cm circuit board.
- Breadboard cables if you build on a breadboard.
- Optionally an SD card.
- Optionally a DS1307 or DS3231 real time clock.
- Optionally a 7805 voltage regulator, the 0.1 uF and 0.33 uF capacitors, plus a power plug.
- Optionally a serial thermo printer.
- Some plywood and a small breadboard for the stand, plus a power supply.

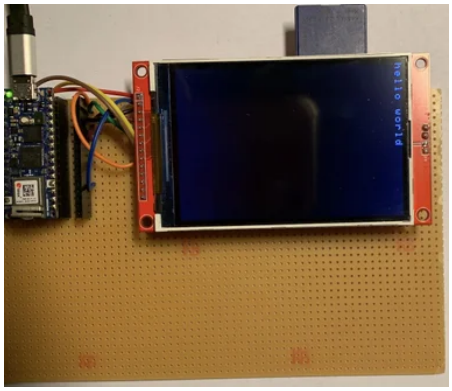
In addition to the hardware supplies, you also need a few software components.

- Arduino IDE. I use 1.8.15 but newer versions also work.
- Arduino Mbed OS Nano board files which can be downloaded from the board manager in the Arduino IDE.
- Adadfruit_GFX libraries which can be downloaded in the Arduino IDE with the library manager (https://github.com/adafruit/Adafruit-GFX-Library). I used version 1.10.14.
- uRTC library from the Arduino IDE (https://github.com/Naguissa/uRTCLib). I use version 6.4.0.
- LittleFS_Mbed support from the Arduino IDE library manager (https://github.com/khoih-prog/LittleFS_Mbed_RP2040). I use version 1.1.0.
- The ILI9488 library, I recommend my fork https://github.com/slviajero/ILI9488
- The patched keyboard library https://github.com/slviajero/PS2Keyboard
- If you want to use the EEPROM filesystem you will need the EEPROM library https://github.com/slviajero/EepromFS as well.

Step 1: Connect the Display to the Microcontroller







The ILI9488 SPI have to rows of connectors on the back side. On the left there are the connectors for power, the display, and the touch screen if you have a touch model. On the right there are 4 connectors for the SD card. For this project the touch connectors are not used.

If you use a circuit board, place the screen on the upper right corner of the board. By default the BASIC interpreter will used the screen in landscape mode. With the layout in the picture the SD card can be removed from above and there is room on the lower side of the board for other components.

Also, mount the Arduino so that the power connector is on the upper side of the board. This makes cabling and mounting the computer easier.

The ILI9488 displays are 3.3V systems and so is the Arduino RP2040. No level converters are needed here.

Connect the 3.3V power pin and the GND pin of the Arduino with the respective pins on the display. They are the uppermost pins.

To drive the display, the CS pin has to be connected with pin 9, DC with pin 8, and RESET with pin 7 of the Arduino.

The SPI bus pins on the Arduino are 13 for SCK, 12 for MISO (=CIPO), and 13 for MOSI (=COPI). These are standard Arduino settings. Connect these pins with the respective pins on the display.

The LED pin can be used to control the brightness of the display. Either you connect it with a potentiometer or you connect it to the analog pin A3. The latter lets you control the brightness of the display from a BASIC program.

After these connections are made you are done. If you are impatient you can try the BASIC interpreter right now.

Upload the files from https://github.com/slviajero/tinybasic/tree/main/IoTBasic into you Arduino IDE.

Open the file IoTBasic.ino and edit the language definition section:

```
#define BASICFULL
#undef BASICINTEGER
#undef BASICSIMPLE
#undef BASICMINIMAL
#undef BASICTINYWITHFLOAT
```

Only BASICFULL should be defined. This compiles all the language features.

Open hardware-arduino.ino and edit the hardware feature section:

```
#undef USESPICOSERIAL
#undef ARDUINOPS2
#undef ARDUINOPRT
#define DISPLAYCANSCROLL
#undef ARDUINOLCDI2C
#undef ARDUINONOKIA51
#define ARDUINOILI9488
#undef ARDUINOSSD1306
#undef LCDSHIELD
#undef ARDUINOTFT
#undef ARDUINOVGA
#undef ARDUINOEEPROM
#undef ARDUINOEFS
#undef ARDUINOSD
#undef ESPSPIFFS
#undef RP2040LITTLEFS
#undef ARDUINORTC
#undef ARDUINOWIRE
#undef ARDUINOWIRESLAVE
#undef ARDUINORF24
#undef ARDUINOETH
#undef ARDUINOMQTT
#undef ARDUINOSENSORS
#undef ARDUINOSPIRAM
#undef STANDALONE
```

Everything should be undefed except ARDUINOILI9488 and DISPLAYCANSCROLL.

You will need to download the ILI9488 library as well. I recommend to use the copy from my repo https://github.com/slviajero/ILI9488. This is a fork of Jaret Burkett's library based on Adafruit_GFX. I will work on hardware scrolling in the future. You also need to install the Adafruit_GFX library either through the Arduino IDE library manager or directly from the repo adafruit/Adafruit-GFX-Library: Adafruit GFX graphics core library, this is the 'core' class that all our other graphics libraries derive from (github.com).

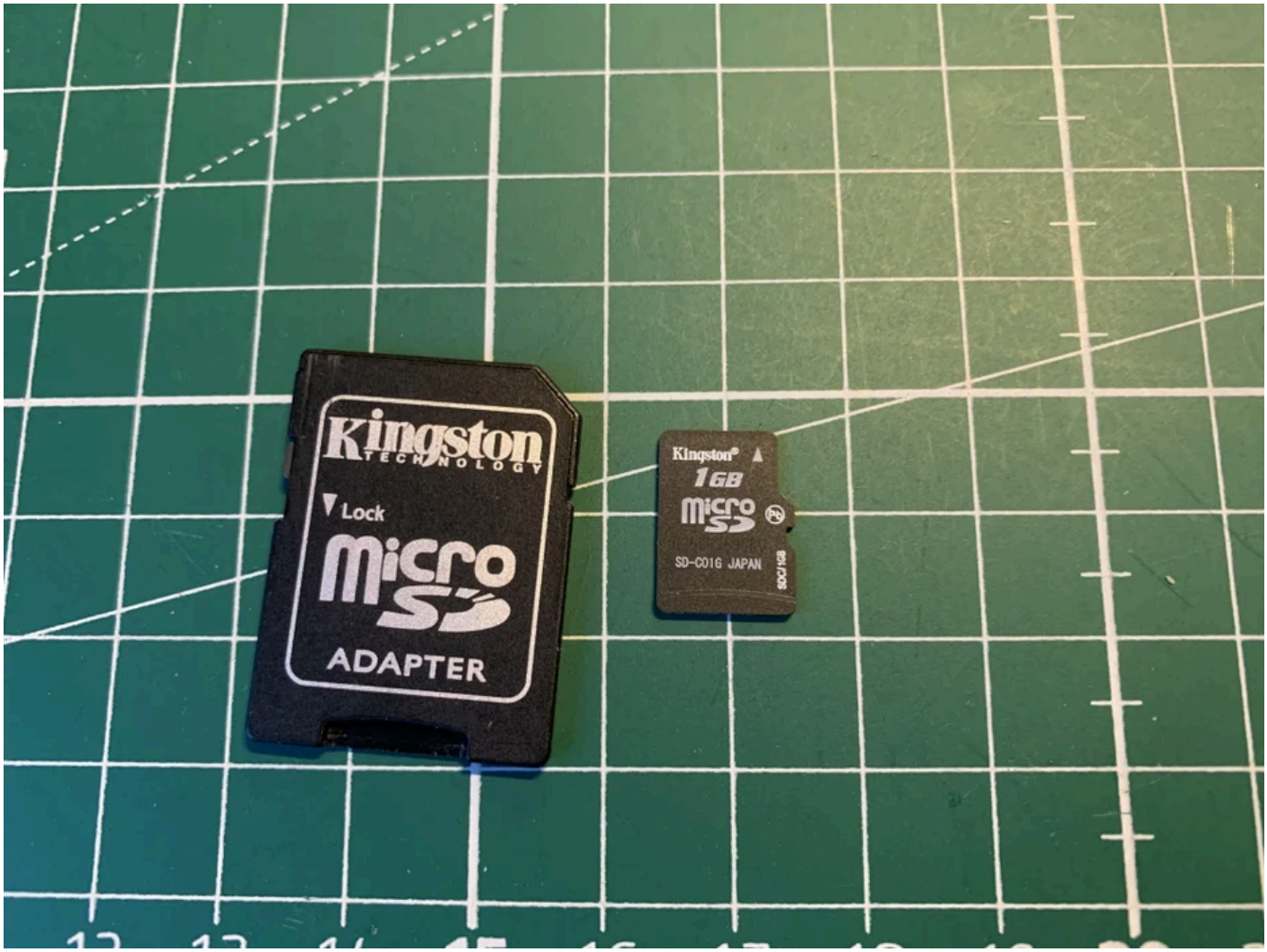
Complile the sketch and upload it to the Arduino. The BASIC interpreter should prompt for input on the serial line.

Enter

```
PRINT &2, "Hello World"
```

This should show on the display if the wiring is correct.

Step 2: Connect the SD Card or Activate LittleFS



For a standalone computer, you will need a file storage. Two option exist for the RP2040. Either use the SD card slot of the display or use the internal flash of the controller with LittleFS. Given the instability and limitations of SD cards I recommend LittleFS as a file system for your standalone computer.

SD Cards

A word of caution on SD cards is in order: Some work on an Arduino with the standard SD library and some don't. In my experience a cleanly formatted and error checked card on a Windows system is the best option. If your system shows sporadic errors check the wiring but also try a different card. The cards are directly attached to the SPI bus with only a few resistors. Cards with messy timing behaviour can cause strange sporadic errors. I use Kingston 1GB cards for this computer.

To use the SD card slot you need to connect SD_MISO, SD_MOSI, and SD_SCK for the SD port on the right hand side of the display to the SPI bus pins like above.

In addition to this, connect pin 10 of the Arduino to the SD_CS pin.

Open hardware-arduino.h and add the SD card system to the hardware definition section:

```
#undef USESPICOSERIAL
#undef ARDUINOPS2
#undef ARDUINOPRT
#define DISPLAYCANSCROLL
#undef ARDUINOLCDI2C
```

```
#undef ARDUINONOKIA51
#define ARDUINOILI9488
#undef ARDUINOSSD1306
#undef LCDSHIELD
#undef ARDUINOTFT
#undef ARDUINOVGA
#undef ARDUINOEEPROM
#undef ARDUINOEFS
#define ARDUINOSD
#undef ESPSPIFFS
#undef RP2040LITTLEFS
#undef ARDUINORTC
#undef ARDUINOWIRE
#undef ARDUINOWIRESLAVE
#undef ARDUINORF24
#undef ARDUINOETH
#undef ARDUINOMQTT
#undef ARDUINOSENSORS
#undef ARDUINOSPIRAM
#undef STANDALONE
```

Only ARDUINOILI9488 and ARDUINOSD are defined all others are undef.

We use the standard Arduino SD libraries in this code.

Compile the sketch and upload it. Insert the SD card and restart the Arduino.

Again, BASIC should show a command prompt on the serial monitor.

Enter a small program like

```
10 PRINT &2, "hello world"
20 PRINT "hello world"
```

Then type

```
SAVE
CATALOG
```

BASIC should list the content of the SD card after the second command. Your program appears as file.bas.

If any of these commands gives an error, check the wiring and the try a different SD card.

LittleFS

If you do not need a removable file storage or you don't have a suitable SD card, best use the internal flash. Instead of the compiler setting ARDUINOSD, use RP2040LITTLEFS. You also need to install the library LittleFS_Mbed from the Arduino IDE library manager or download it from here https://github.com/khoih-prog/LittleFS_Mbed_RP2040.

The compiler settings in this case would be

```
#undef USESPICOSERIAL
#undef ARDUINOPS2
#undef ARDUINOPRT
#define DISPLAYCANSCROLL
#undef ARDUINOLCDI2C
#undef ARDUINONOKIA51
#define ARDUINOILI9488
#undef ARDUINOSSD1306
#undef LCDSHIELD
#undef ARDUINOTFT
#undef ARDUINOVGA
#undef ARDUINOEEPROM
```

```
#undef ARDUINOEFS
#undef ARDUINOSD
#undef ESPSPIFFS
#define RP2040LITTLEFS
#undef ARDUINORTC
#undef ARDUINOWIRE
#undef ARDUINOWIRESLAVE
#undef ARDUINORF24
#undef ARDUINOETH
#undef ARDUINOMQTT
#undef ARDUINOSENSORS
#undef ARDUINOSPIRAM
#undef STANDALONE
```

Compile and upload the program.

After BASIC starts and you see the first command line, type

```
FDISK
```

to initially format the internal LittleFS file storage. After this all file command can be used on the internal storage. Default file system size is 1GB. This can be changed with the parameter "RP2040_FS_SIZE_KB" at compile time.

SAVE "filename" saves a program.

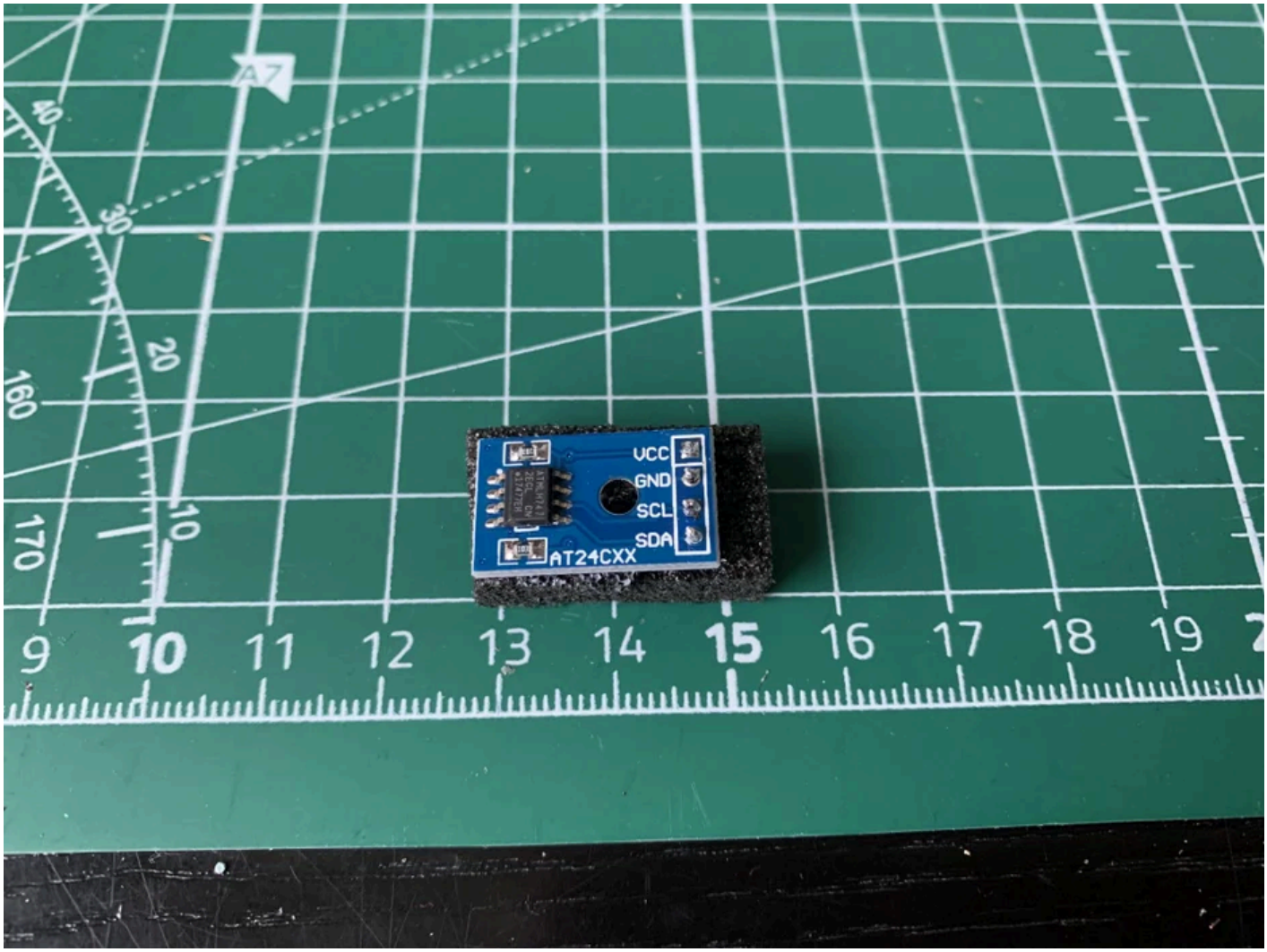
LOAD "filename" loads a program.

CATALOG displays the stored files.

DELETE deletes a file.

OPEN and CLOSE open and close files for read. See the File I/O section of the manual https://github.com/slviajero/tinybasic/blob/main/MANUAL.md for more information.

Step 3: Alternative Mass Storage - EEPROM



If you have no suitable SD card at hand and still want to add a removable mass storage device to your computer, EEPROM modules are a great alternative.

Connect SDA and SCL to the respective Arduino PINs A4 and A5 and power to VCC and GND. I recommend 32kB or 64kB EEPROM modules.

Download the EEPROM file system library from my repo https://github.com/slviajero/EepromFS. This library creates a very simple file system on any I2C EEPROM. It can be used without the BASIC interpreter as a standalone component. The filesystem divides the EEPROM in a number of equal size slots and stores files in it. The API is C style.

Instead of ARDUINOSD or RP2040LITTLEFS, set the compiler flag ARDUINOEFS.

```
#undef USESPICOSERIAL
#undef ARDUINOPS2
#undef ARDUINOPRT
#define DISPLAYCANSCROLL
#undef ARDUINOLCDI2C
#undef ARDUINONOKIA51
#define ARDUINOILI9488
#undef ARDUINOSSD1306
#undef LCDSHIELD
#undef ARDUINOTFT
#undef ARDUINOVGA
#undef ARDUINOEEPROM
#define ARDUINOEFS
#undef ARDUINOSD
#undef ESPSPIFFS
```

```
#undef RP2040LITTLEFS
#undef ARDUINORTC
#undef ARDUINOWIRE
#undef ARDUINOWIRESLAVE
#undef ARDUINORF24
#undef ARDUINOETH
#undef ARDUINOMQTT
#undef ARDUINOSENSORS
#undef ARDUINOSPIRAM
#undef STANDALONE
```

You also need to set the EEPROM size and the I2C address. This is done in the code section

```
#define EEPROMI2CADDR 0x050
#define RTCI2CADDR 0x068
#define EFSEEPROMSIZE 32767
```

Leave the RTC setting unchanged for now and add you data there.

Compile the sketch and upload it.

Once the BASIC interpreter starts up you can format the EEPROM file system

```
FDISK 4
```

would create 4 file slots of 8kB on an 32kB EEPROM.

File commands like CATALOG, SAVE, LOAD, OPEN, CLOSE, and DELETE can be used now on the EEPROM.

EEPROM modules can be exchanged easily between different BASIC computers. They are a bit like the program modules of the old ATARI computers. The advantage of the EEPROM modules is the low energy consumption.

Alternatively, the EEPROM of a real time clock can be used as a file system. Please see the section "Add Time" for more on this.

More on EEPROMs can be found on a section below "Build The Little Brother" and in the tutorial https://www.instructables.com/Use-I2C-EEPROMs-As-a-File-System-on-an-Arduino/

Step 4: Intermezzo - Try the Graphics

Once you have a running SD card and the display, you might try a bit of the graphics. Switch of the Arduino, remove the SD card, put it into your computer.

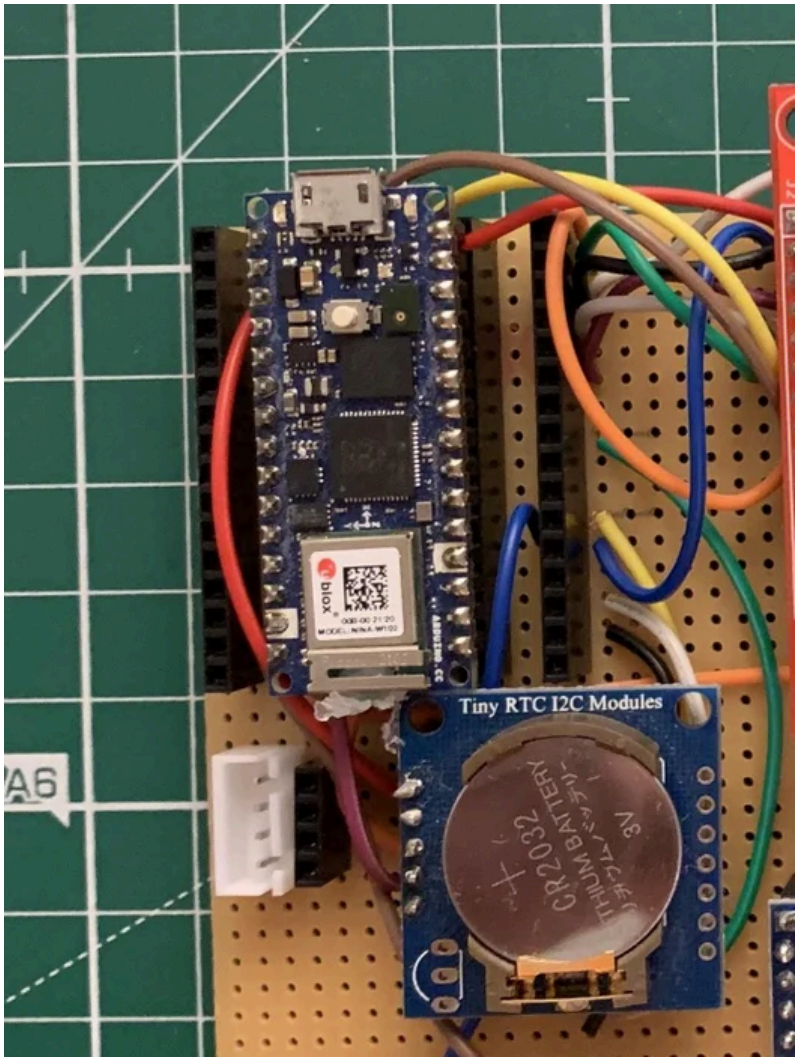
Put the program https://github.com/slviajero/tinybasic/blob/main/examples/00tutorial/mandelv.bas from the tutorial section on the SD card, reinsert it to the Arduino.

From the BASIC command prompt type

```
LOAD "MANDELV.BAS"
RUN
```

The Mandelbrot set should appear on your display.

Step 5: Add Time.



The real time clock board are connected to the I2C bus of the computer. I placed the board just below the Arduino slightly to the right.

Connect the VCC and GND connectors to the Arduino.

SDA of the clock is connected with A4 on the Arduino and SCL with A5. This is the standard pinout. The two plugs to the right of the clock in the picture are I2C connectors. On both there is the I2C bus in the order (GND, VCC, SDA, SCL). This is the standard Grove pinout. These connectors are used later to connect sensors.

Again, recompile the sketch with the settings

```
#undef USESPICOSERIAL
#undef ARDUINOPS2
#undef ARDUINOPRT
#define DISPLAYCANSCROLL
#undef ARDUINOLCDI2C
#undef ARDUINONOKIA51
#define ARDUINOILI9488
#undef ARDUINOSSD1306
#undef LCDSHIELD
#undef ARDUINOTFT
#undef ARDUINOVGA
#undef ARDUINOEEPROM
#undef ARDUINOEFS
#define ARDUINOSD
#undef ESPSPIFFS
#undef RP2040LITTLEFS
```

```
#define ARDUINORTC
#define ARDUINOWIRE
#undef ARDUINOWIRESLAVE
#undef ARDUINORF24
#undef ARDUINOETH
#undef ARDUINOMQTT
#undef ARDUINOSENSORS
#undef ARDUINOSPIRAM
#undef STANDALONE
```

ARDUINORTC and ARDUINOWIRE are defined in addition to the definitions before.

You will the the uRTC library for this. It can be downloaded with the Arduino library manager.

Please check the I2C settings

```
#define EEPROMI2CADDR 0x050
#define RTCI2CADDR 0x068
#define EFSEEPROMSIZE 32767
```

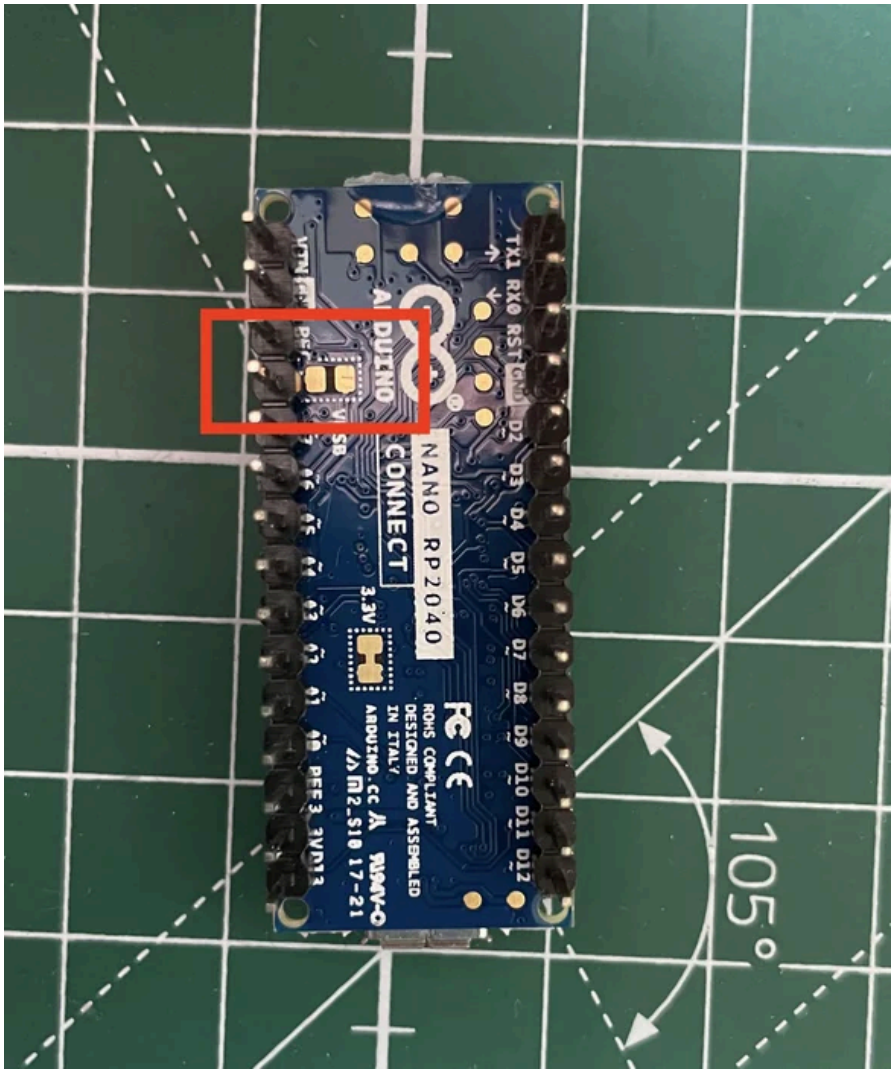
The value 0x068 is the standard I2C real time clock value. If you use modules with a different address, you need to change it here. If you use the EFS filesystem described above and have an extra EEPROM attached to the I2C bus, check if there is no address conflict. Some clocks have their EEPROM on 0x057 and some on 0x050. You can have only one EEPROM on the system.

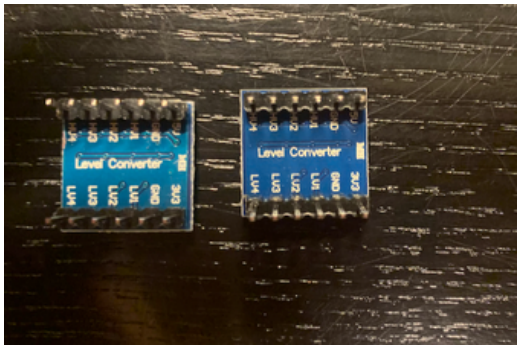
Once the BASIC prompt appears on the command line, type

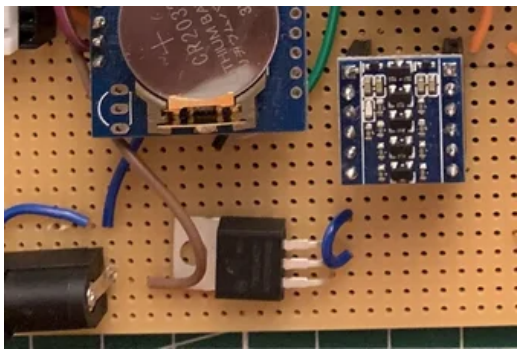
```
@T(0)=0
PRINT @T$
```

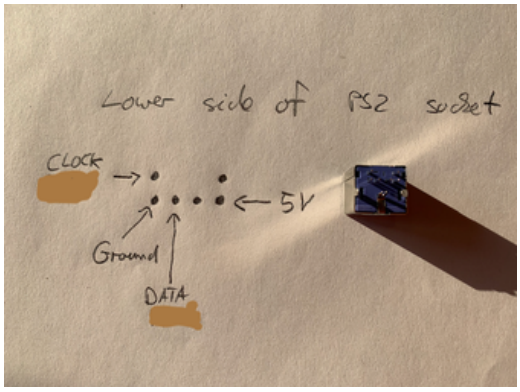
The first command initialises the clock and the second command should show the date string.

Step 6: Add the Keyboard









So far the entire work has been done the the 3.3V world. Making a mistake rarely causes permanent damage to the Arduino of any of the components.

PS2 keyboards need 5V. To make 5V available on the board you need to shortcut a connection on the back side of the Arduino as it is shown in the picture. Solder this connection. Now the 5V pin of the Arduino has power. Connecting it with any other pin will probably damage the board.

You will need the level converter now. Connect 3.3V, GND as well as the Arduino pins 2 and 5 to the 3.3V side of the level converter. Connect 5V and GND to the 5V side of the level converter. I placed the level converter and all the other 5 V stuff on the lower side of the board to avoid any accidents.

Now you can connect the PS2 socket like it is shown in the drawing. The clock pin on the right side of the socket goes to the pin of the level connectors 5 V side that has the pin 2 data of the Arduino. The data pin is connected with the level connector pin that has the pin 5 data of the Arduino. Power and ground are connected respectively.

You can now again extend the hardware definitions in hardware-arduino.h

```
#undef USESPICOSERIAL
#define ARDUINOPS2
#undef ARDUINOPRT
#define DISPLAYCANSCROLL
#undef ARDUINOLCDI2C
#undef ARDUINONONOKIA51
#define ARDUINOILI9488
```

```
#undef ARDUINOSSD1306
#undef LCDSHIELD
#undef ARDUINOTFT
#undef ARDUINOVGA
#undef ARDUINOEEPROM
#undef ARDUINOEFS
#define ARDUINOSD
#undef ESPSPIFFS
#undef RP2040LITTLEFS
#define ARDUINORTC
#define ARDUINOWIRE
#undef ARDUINOWIRESLAVE
#undef ARDUINORF24
#undef ARDUINOETH
#undef ARDUINOMQTT
#undef ARDUINOSENSORS
#undef ARDUINOSPIRAM
#undef STANDALONE
```

Add the definition ARDUINOPS2. You will need the Arduino PS2 library. I recommend to use my fork of it from https://github.com/slviajero/PS2Keyboard as I have added some useful features.

You can now connect the keyboard and restart the Arduino. The status lights of the keyboard should blink after restart. PS2 keyboards often need a lot of power. Sometimes the keyboard initialises too slowly. It may be necessary to restart the Arduino by disconnecting the power and reconnecting it again if there is a problem.

Once the keyboard has blinked you can try it with the BASIC program

```
10 FOR I=1 TO 100
20 GET &2, A: PRINT A
30 DELAY 500
40 NEXT
```

Type RUN to start the program. The ASCII values of the keys should appear on the serial monitor.

Step 7: Make It Standalone

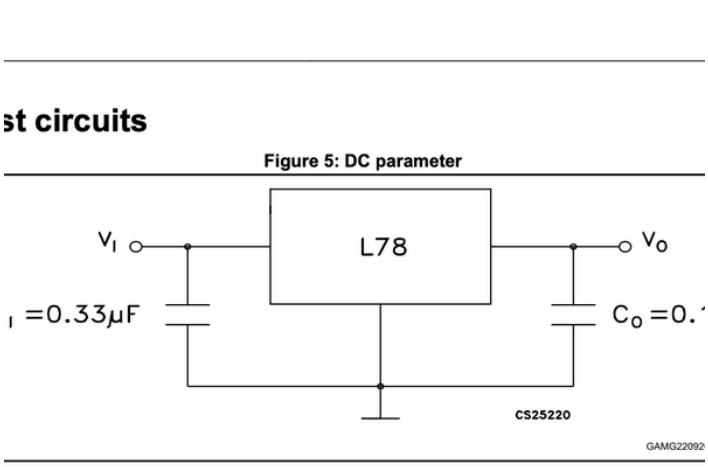
Now you are ready for standalone.

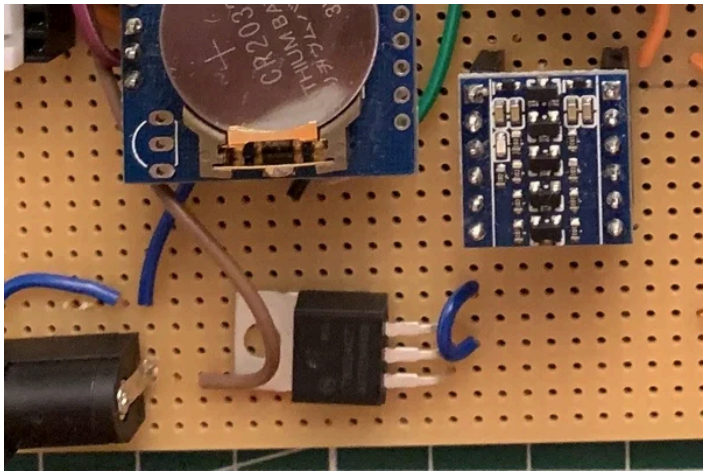
Set the STANDALONE option in hardware-arduino.h by defining the respective line. Recompile and upload the program. After a while the command prompt should appear on the screen

Congratulations! You have build a standalone computer from scratch.

The prompt shows the version, the free memory and the EEPROM size (which is 0 on an RP2040). No blinking cursor appears. Just start typing and programming.

Step 8: Power Play - Adding a Voltage Regulator





For a true standalone computer, independence of the USB power can be helpful. You either can use a USB power supply plus a suitable cable or you add a power plug and a voltage regulator.

The the regulator circuit is straightforward. The 7805 voltage regulator has one input that goes to the power plug. Plus is on the inside pin and minus on the outside of the plug. The output pin is connected directly with the 5V side of the level converter. Only the level converter and the keyboard use the 5V pin. There is no protection of the computers USB pin with this circuit. Never plug in a power supply and the computer at the same time.

Two capacitors as shown in the schematics are recommended but not really needed here. Typically a power supply and the PS2 keyboard have this already inside.

Step 9: Add a Printer

Little thermo printers are quite useful to get quick hard copies of data collected by the Arduino.

These thermo printers use 5V logic and typically need a 5V 2A power supply. Connecting the printer should be done through the level converter.

Connect the TX and RX pin of the Arduino to the 3.3V side of the level converter. Then connect the 5V side of these pins to a suitable plug.

I use standard stereo audio cables and plugs for this. This may be unorthodox but these cables are cheap, good quality and well shielded. Baud rates of 9600 can be transferred safely through them.

Ground is connected to the outer side of the plug, TX and RX through the inner side. Make sure that the Arduino's TX is connected with the printer's RX and vice versa.

To activate printing, recompile the interpreter after configuring hardware-arduino.h

```
#undef USESPICOSERIAL
#define ARDUINOPS2
#define ARDUINOPRT
#define DISPLAYCANSCROLL
#undef ARDUINOLCDI2C
#undef ARDUINONOKIA51
#define ARDUINOILI9488
#undef ARDUINOSSD1306
#undef LCDSHIELD
#undef ARDUINOTFT
#undef ARDUINOVGA
```

```
#undef ARDUINOEEPROM
#undef ARDUINOEFS
#define ARDUINOSD
#undef ESPSPIFFS
#undef RP2040LITTLEFS
#define ARDUINORTC
#define ARDUINOWIRE
#undef ARDUINOWIRESLAVE
#undef ARDUINORF24
#undef ARDUINOETH
#undef ARDUINOMQTT
#undef ARDUINOSENSORS
#undef ARDUINOSPIRAM
#define STANDALONE
```

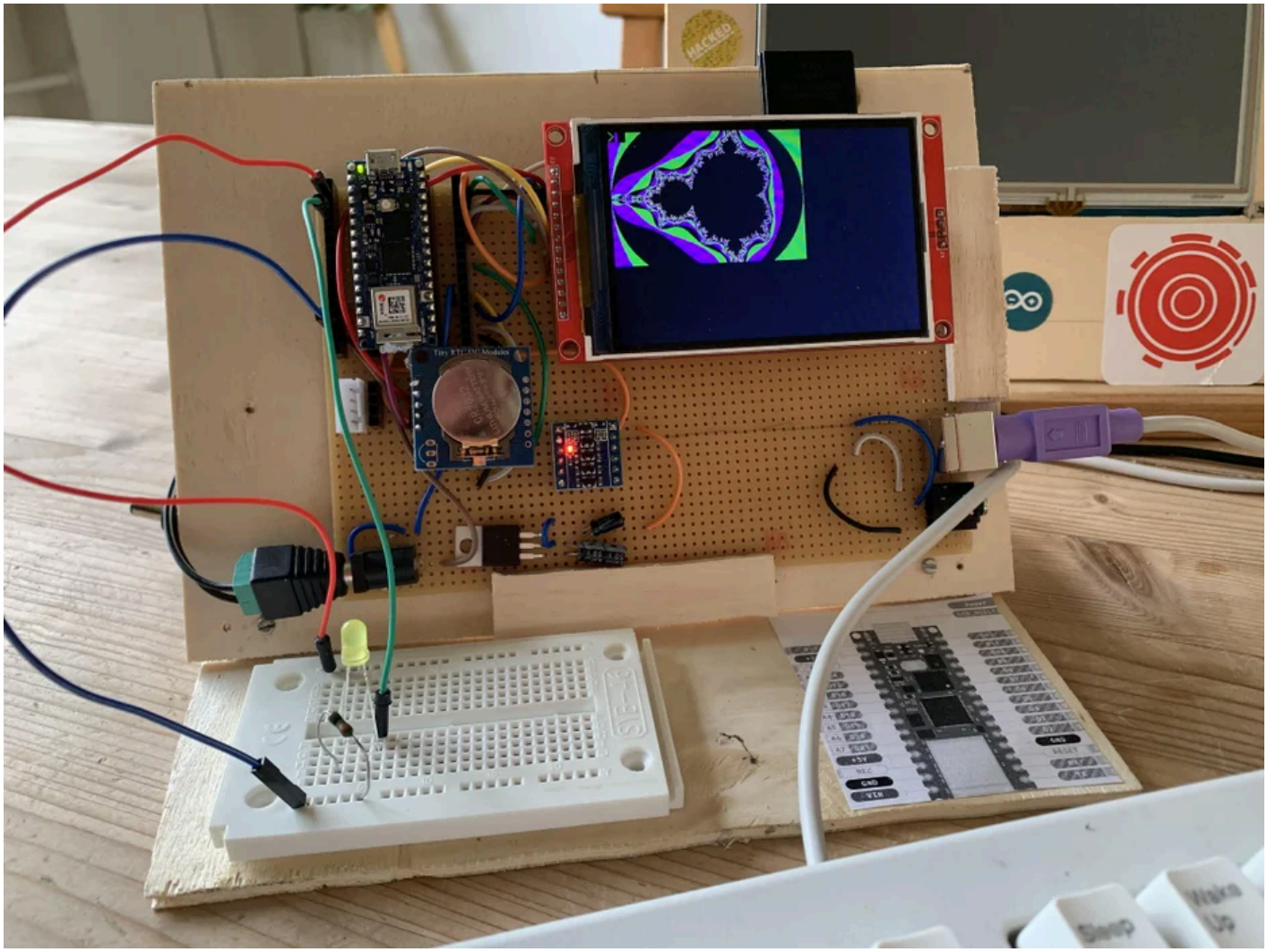
ARDUINOPRT is defined. This makes Serial1 available in BASIC as output stream &4.

Data is sent to the printer with the command

```
PRINT &4, "Hello World"
```

Please read the manual https://github.com/slviajero/tinybasic/blob/main/MANUAL.md for more information on BASIC output streams and the use of the & modifier.

Step 10: Mount the Computer in a Frame



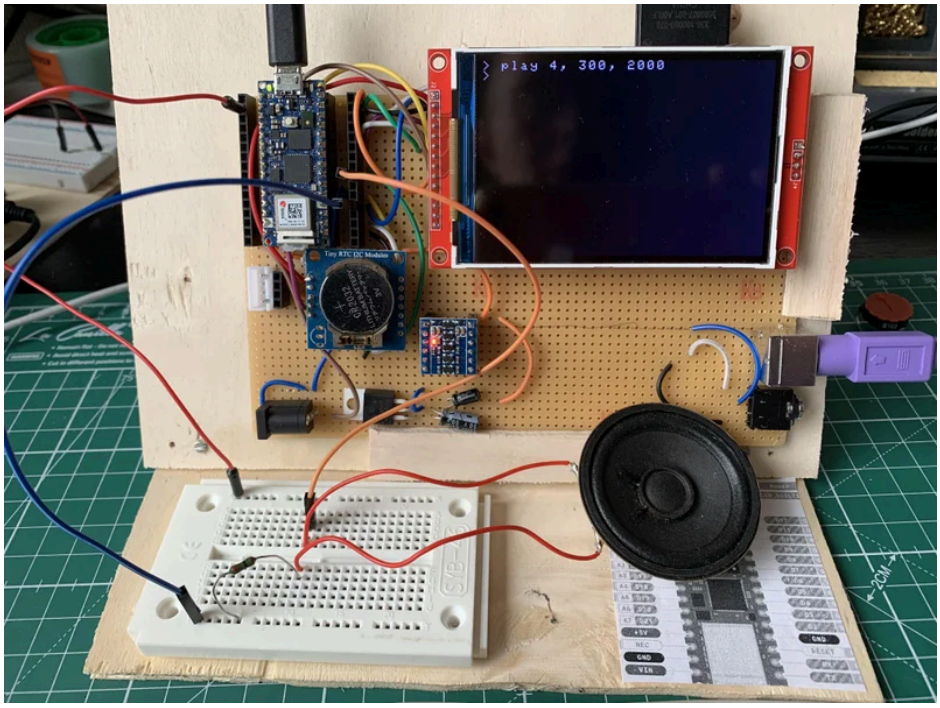
Once the board is finished it is best mounted into a frame. I use a plywood board holding it in place in a 45 degree angle. The stand has enough room for a medium size breadboard to to experiments and a small printout of the Arduino RP2040 pinout.

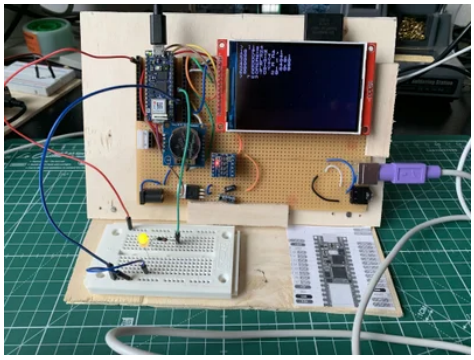
The peripherals of the computer use pins D2, D5 (PS2 keyboard), D7, D8, D9, and A3 (display), as well as D10 (SD card CS). If the clock is used, A4 and A5 are used for the I2C bus. All other pins are free. This means that D3, D4, D6, A0, A1, A2, A6, and A7 can be used for applications.

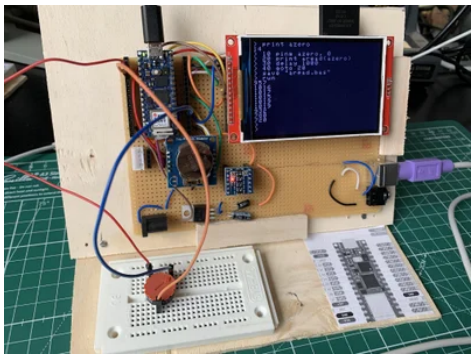
My main use case for this system is testing sensors and devices interactively on the breadboard in front of the computer. No time consuming uploading of sketches is needed.

For computers like this, I also like to print the pinout of the microcontroller and glue it somewhere where it is visible. This helps to wire things quickly.

Step 11: Do All the Arduino Things







The computer you have build up to this step has a full featured BASIC interpreter like the home computers of the 80s plus most of the features the Arduino IDE offers for controlling devices.

If you connect a standard 8 Ohm speaker plus a 200 Ohm resistor to pin D4 and GND you can play sound. Type

```
PLAY 4, 300, 2000
```

on the command line. A 300 Hertz tone is play for 2 seconds on pin 4.

The use the same resistor and a led on pin 4. Then type the blink program and run it

```
10 PINM 4,1
20 DWRITE 4,1
30 DELAY 1000
40 DWRITE 4,0
50 DELAY 1000
60 GOTO 20
RUN
```

The led now blinks in 1 second intervals. The running program can be interrupted by typing the "#" character.

Analog data can also be read easily. Connect a 10 kOhm potentiometer to GND and 3.3V and the variable pin of it to D14 which is the first analog pin A0. The program

```
10 PINM 14, 0
20 PRINT AREAD(14)
30 DELAY 1000
40 GOTO 20
```

will print the raw input level of the analog pin to the screen. It is a number from 0 to 1024. An output showing volts would need one more command

```
10 PINM 14, 0
20 A=AREAD(14)
30 V=MAP(A, 0, 1023, 0, 330)
40 PRINT V/100
```

```
50 DELAY 1000
60 GOTO 20
```

Map is the well known Arduino MAP command in BASIC. It handles long integers in the background.

Step 12: Go to the Network

Very basic IoT support has been added to the BASIC interpreter. To connect your system to the network, first open the file wifisettings.h. This file should look like this

```
const char* ssid = "";
const char* password = "";
const char* mqtt_server = "test.mosquitto.org";
const short mqtt_port = 1883;
byte mac[] = {0xDE, 0xAD, 0xBE, 0xE9, 0xE9, 0xE9};
```

Add your SSID and the password of your network here. This is not secure. I use a playground network for this.

The next two lines contain the address and port of an mqtt server. Only unauthenticated and unencrypted MQTT is supported right now. Again, this is not secure for any real world stuff. The preconfigured server is the mosquitto test server.

After configuring this file, open hardware-arduino.h again and make the following change:

```
#undef USESPICOSERIAL
#define ARDUINOPS2
#define ARDUINOPRT
#define DISPLAYCANSCROLL
#undef ARDUINOLCDI2C
#undef ARDUINONOKIA51
#define ARDUINOILI9488
#undef ARDUINOSSD1306
#undef LCDSHIELD
#undef ARDUINOTFT
#undef ARDUINOVGA
#undef ARDUINOEEPROM
#undef ARDUINOEFS
#define ARDUINOSD
#undef ESPSPIFFS
#undef RP2040LITTLEFS
#define ARDUINORTC
#define ARDUINOWIRE
#undef ARDUINOWIRESLAVE
#undef ARDUINORF24
#undef ARDUINOETH
#defineARDUINOMQTT
#undef ARDUINOSENSORS
#undef ARDUINOSPIRAM
#define STANDALONE
```

ARDUINOMQTT is now defined in addition to all other settings.

You will need the Pubsub MQTT library and the Arduino WifiNINA library from the Arduino library manager now.

Recompile and upload the sketch to the board.

After reboot type

```
NETSTAT
```

If the computer is connected to the network the output should be

Network connected

```
MQTT state -1
MQTT out topic
```

```
MQTT inp topic
MQTT name iotbasicxxx
```

Now the computer is ready to receive and send MQTT messages.

A typical send program without any error handling could look like this

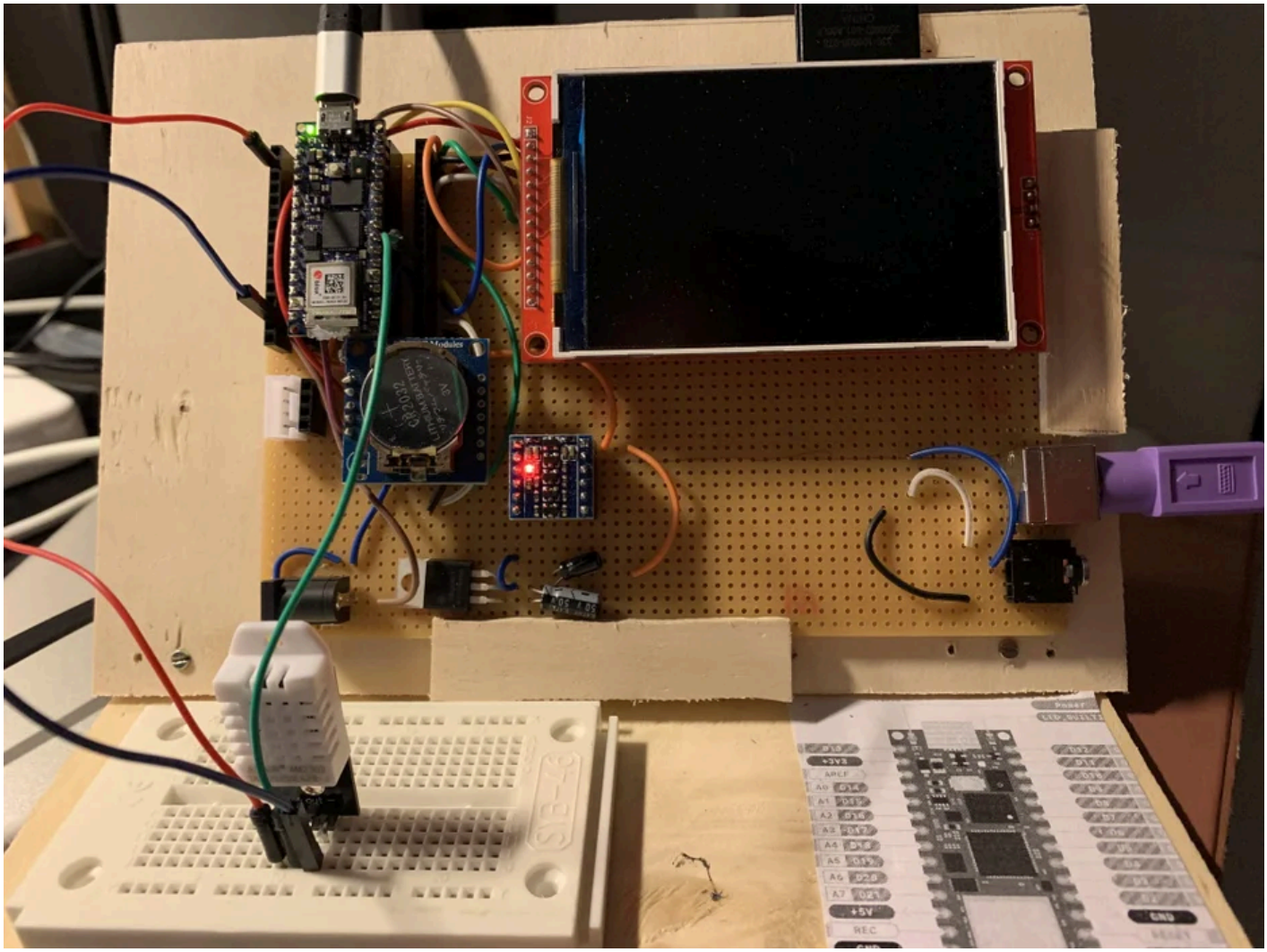
```
10 OPEN &9, "iotbasic/testdata", 1
20 PRINT &9, "data:", AREAD(AZERO)
30 DELAY 2000
40 GOTO 20
```

This program would open the MQTT topic iotbasic/testdate for write, print the string "data:" and the analog value on A0, wait for two seconds and then measure again.

There is a tutorial on the MQTT features in
https://github.com/slviajero/tinybasic/tree/main/examples/16mqtt.

This code is work in progress. More will come here soon.

Step 13: Connect I2C Devices and Sensors



BASIC has a set of sensors build in and it can access the I2C bus directly through BASIC commands.

Download the program https://github.com/slviajero/tinybasic/blob/main/examples/13wire/ident.bas from the repo or type it in.

```
10 REM "Identify devices on the I2C bus"
20 REM ""
100 REM "the setup()"
110 FOR I=1 TO 127
120 @S=0
130 REM "Try to open a device and send a byte"
140 OPEN &7, I
150 PUT &7, 0
160 IF @S=0 THEN PRINT "device found on", #3, I;": ";:GOSUB 500
190 NEXT
200 PRINT
210 END
500 REM "Search the device name"
505 IF I=56 THEN PRINT "AH10" : RETURN
510 IF I=60 THEN PRINT "Oled" : RETURN
520 IF I>=80 AND I<=87 THEN PRINT "EEPROM" : RETURN
530 IF I=104 THEN PRINT "Real Time Clock" : RETURN
540 IF I=118 OR I=119 THEN PRINT "BMP/E280" : RETURN
590 PRINT "Unknown"
600 RETURN
```

If you run the program, it should detect the real time clock on the I2C bus. Connect any other I2C sensor to SDA, SCL and power and rerun the program. The sensor is now detected.

The BASIC commands GET, PUT, PRINT and INPUT on the I/O channel &7 allow direct access to I2C devices. Sensors can be read out on byte level and data can be used in programs.

Please look into the I2C tutorial for more information
https://github.com/slviajero/tinybasic/blob/main/examples/13wire/README.md

Some sensors are predefined in BASIC. They can be activated as part of the code. In hardware-arduino.h look into the section for ARDUINOSENSORS

```
/*
 * Sensor library code - experimental
 */
#ifdef ARDUINOSENSORS
#undef ARDUINODHT
#define DHTTYPE DHT22
#define DHTPIN 2
#undef ARDUINOSHT
#ifdef ARDUINOSHT
#define ARDUINOWIRE
#endif
#undef ARDUINOMQ2
#define MQ2PIN A0
#undef ARDUINOLMS6
#define ARDUINOAHT
#undef ARDUINOBMP280
#undef ARDUINOBME280
#endif
```

These sensors are currently supported as build in sensors of BASIC. Activating ARDUINOSENSORS by setting the define AND activating the sensor here will make the sensor values available in BASIC.

If a DHT22 sensor is available and would be connected to pin D3 the DHT section above would be changes to

```
#define ARDUINODHT
#define DHTTYPE DHT22
#define DHTPIN 3
```

After this in the hardware definition, we would add

```
#undef USESPICOSERIAL
#define ARDUINOPS2
#define ARDUINOPRT
#undef DISPLAYCANSCROLL
#undef ARDUINOLCDI2C
#undef ARDUINONOKIA51
#define ARDUINOILI9488
#undef ARDUINOSSD1306
#undef LCDSHIELD
#undef ARDUINOTFT
#undef ARDUINOVGA
#undef ARDUINOEEPROM
#undef ARDUINOEFS
#define ARDUINOSD
#undef ESPSPIFFS
#undef RP2040LITTLEFS
#define ARDUINORTC
#define ARDUINOWIRE
#undef ARDUINOWIRESLAVE
#undef ARDUINORF24
#undef ARDUINOETH
#define ARDUINOMQTT
```

```
#define ARDUINOSENSORS
#undef ARDUINOSPIRAM
#define STANDALONE
```

Now ARDUINOSENSORS is also defined.

Download the DHT22 library and recompile the sketch.

Connect a DHT sensor to pin 3 then restart the Arduino.

To read out the sensor, the SENSOR function is used. Its first argument is the sensor number and the second argument is the value to be read out by the sensor.

```
PRINT SENSOR(1,2), SENSOR(1,1)
```

would print the current temperature and humidity.

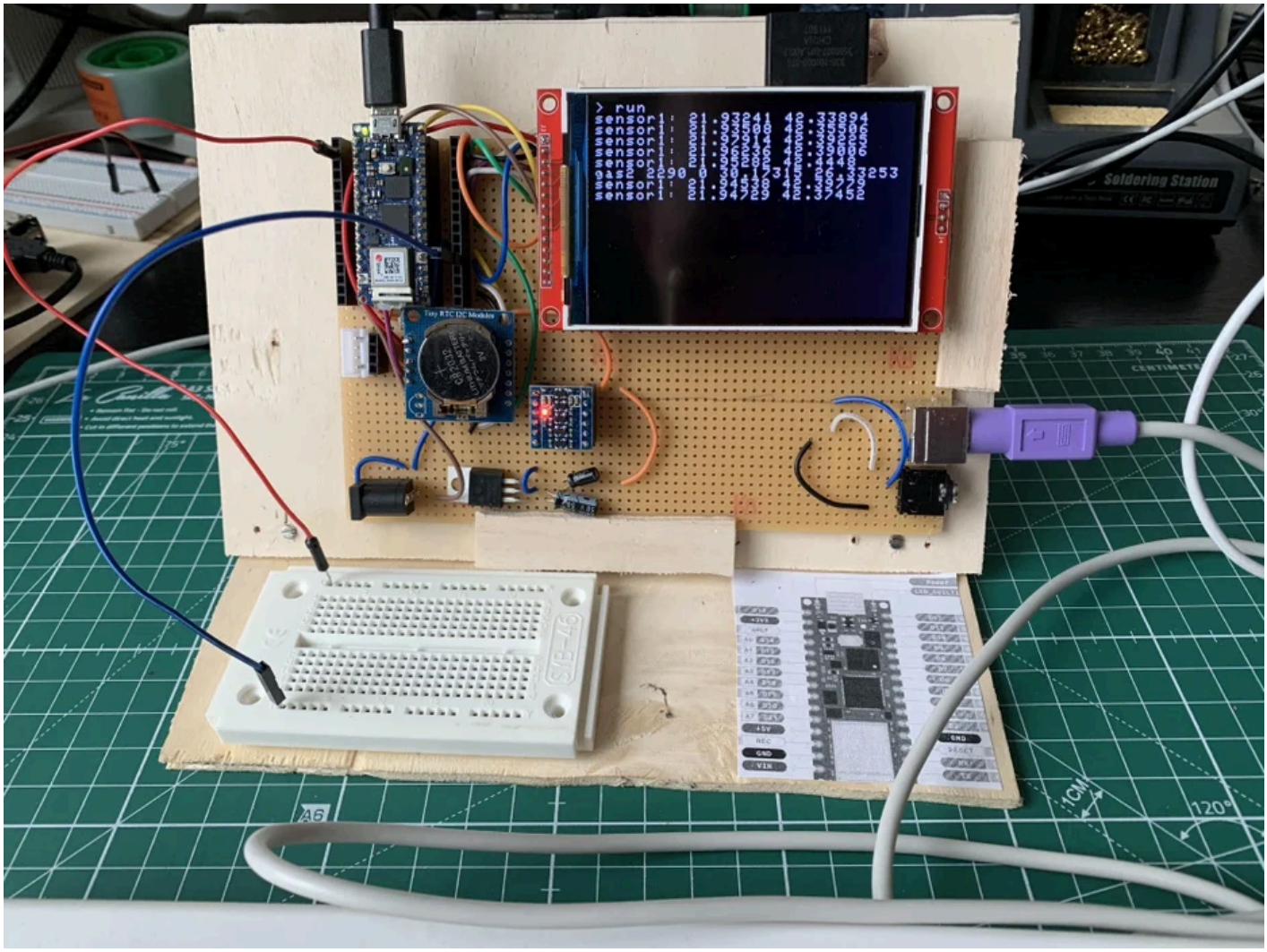
A program sending this data to an MQTT server would be

```
10 OPEN &9, "iotbasic/testdata", 1
20 PRINT &9, "dhtdata:", SENSOR(1,2), SENSOR(1,1)
30 DELAY 2000
40 GOTO 20
```

With four lines of BASIC code data can be measured and transferred. Saving this file as "autoexec.bas" to the SD card would autorun it after every restart of the computer. With this the computer could run as a standalone sensor device.

Sensor code is also work in progress. More sensors from the Adafruit library will be integrated in future.

Step 14: Read and Display Sensor Data



In the example above sensor data was published to an MQTT topic in the internet. Data can also be read from the internet and displayed on the screen.

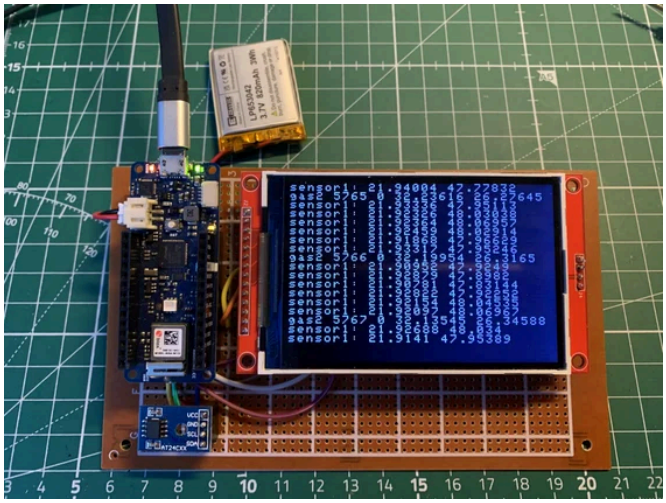
The most elementary reader program would look like this

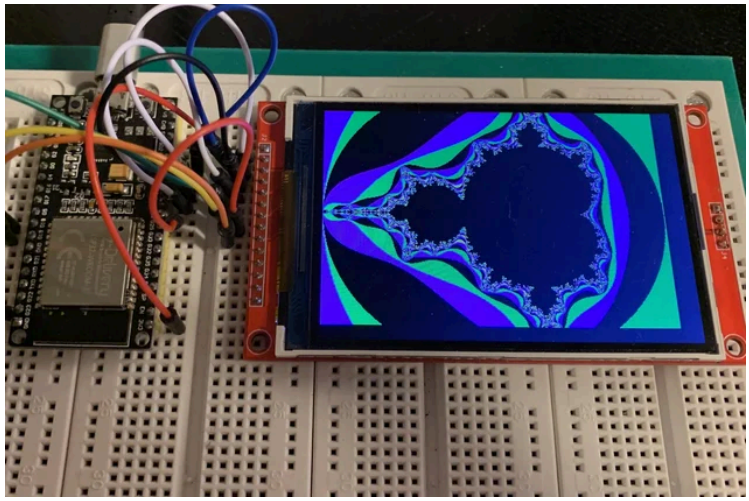
```
10 OPEN &9, "iotbasic/testdata"
20 IF AVAIL(9)=0 THEN DELAY 1000: GOTO 20
30 INPUT &9, A$
40 PRINT A$
50 GOTO 20
```

The program opens the MQTT topic for read, loops every 1 second to see if there are characters available in stream 9 which is the MQTT stream, and prints them.

The output in the photo shows the values of two sensors currently running in my home automation test. One measures CO_2 values and the other temperature and air humidity.

Step 15: Build the Little Brother





The hardware design in this text is for the RP2040 but can be adapted easily for other microcontrollers.

One of my favourite boards is an Arduino MKR Wifi 1010. It is a low energy board with a lot of computing power.

The pinout is different from the RP2040 but all the steps can be done the same.

To build a display and EEPROM only system like in the picture connect the display SPI pins to the Arduino SPI pins. For the other pins of the display connect CS to pin 7, RST to pin 6, DC to pin 4, and LED to pin A3 just like in other example.

The compiler settings for the BASIC interpreter would be

```
#undef USESPICOSERIAL
#undef ARDUINOPS2
#undef ARDUINOPRT
#define DISPLAYCANSCROLL
#undef ARDUINOLCDI2C
#undef ARDUINONOKIA51
#define ARDUINOILI9488
#undef ARDUINOSSD1306
#undef LCDSHIELD
#undef ARDUINOTFT
#undef ARDUINOVGA
#undef ARDUINOEEPROM
#define ARDUINOEFS
#undef ARDUINOSD
#undef ESPSPIFFS
#undef RP2040LITTLEFS
#define ARDUINORTC
#define ARDUINOWIRE
#undef ARDUINOWIRESLAVE
#undef ARDUINORF24
#undef ARDUINOETH
#define ARDUINOMQTT
#define ARDUINOSENSORS
#undef ARDUINOSPIRAM
#undef STANDALONE
```

As the system has not keyboard STANDALONE has to be undone including the ARDUINOPS2 settings.

EEPROM size and connection has to be set as in the section above.

Set the following definitions

```
#define ILI_CS 7
#define ILI_DC 4
#define ILI_RST 6
```

They override the standard ILI settings and tell the BASIC interpreter which pins to use for the display.

Compile and upload.

The BASIC interpreter will start saying something like

```
> Stefan's Basic 1.4a Memory 22455 0
>
```

and expect input from the command line. It has 22 kB of memory ready for programs.

If you add a suitable LiPo battery, the system can be disconnected from the computer with a running program and reconnected later without a restart of the program.

On Arduino MKR the sleep mode is supported for low power application. Using commands like

```
10 SLEEP 10000
```

will put the Arduino to sleep for 10000 mili seconds and then resume program execution. The USB bus will not be reattached after sleep. The device will run standalone as a signage, network or sensor device.

Alternatively, the same system can be build with a ESP32. The pinout for the display would be

```
#define ILI_CS 12
#define ILI_DC 27
#define ILI_RST 14
#define ILI_LED 26
```

An EEPROM can be used as mass storage like in the example above. SPI bus defaults are 18 for SCK, 19 for MISO and 23 for MOSI.

Alternatively, on an ESP the build in SPIFFS file system can be used. Make the following change to the compiler definition

```
#undef ARDUINOEFS
#define ESPSPIFFS
```

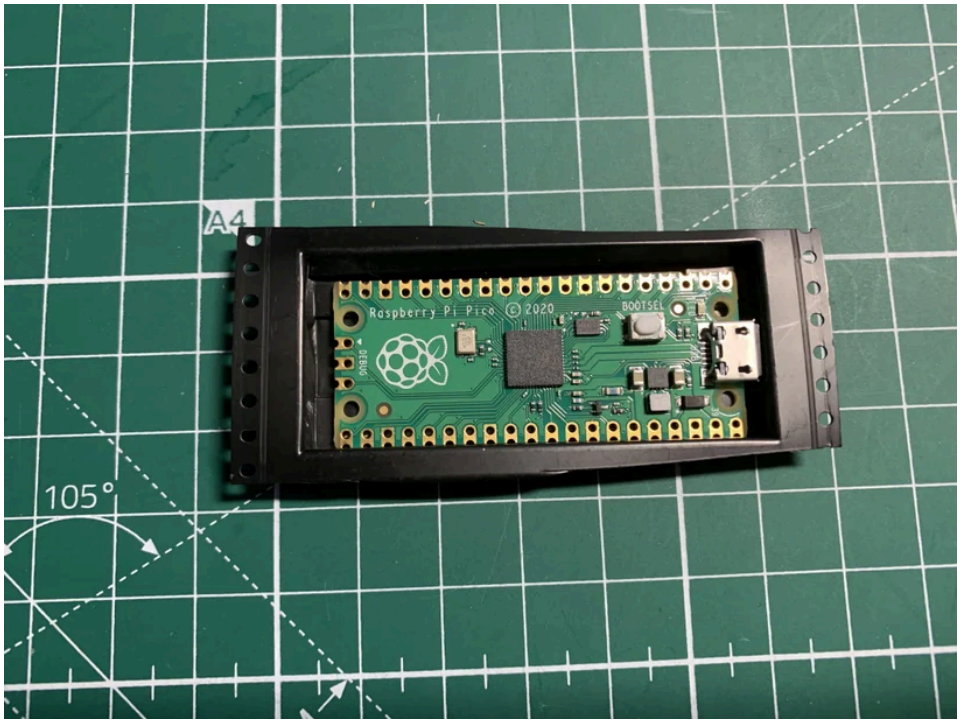
After the first start of the BASIC interpreter, use FDISK to format the internal disk.

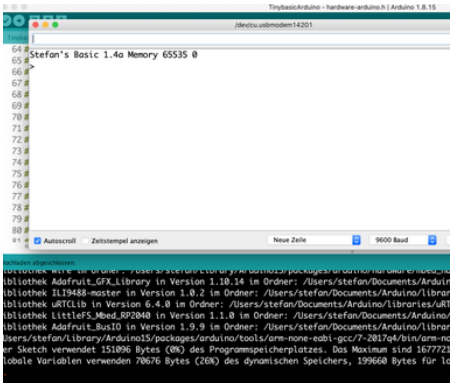
Sleep does behave differently on ESP32 than on Arduino MKR. Unlike the MKR

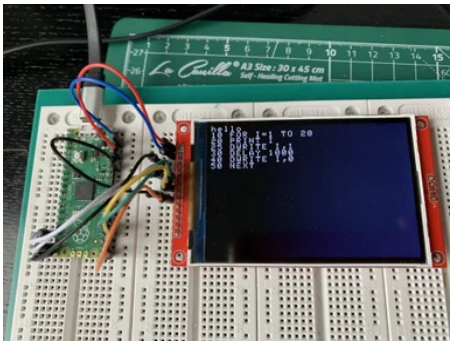
```
10 SLEEP 10000
```

will not resume the BASIC program but restart the controller after wakeup. The program has to be stored as "autoexec.bas" to be reloaded and restarted after a deep sleep. It has to handle entry points after sleep by itself e.g. by using an EEPROM variable. This difference is due to the differences in ESP sleep and ArduinoLow power libraries.

Step 16: On Raspberry Pi Pico







This instructable is mainly on the Arduino RP2040 board. Question have been asked it the computer could also be build on the basis of a Raspberry PI Pico.

I tested the boards without WLAN like the ones shown in the photo and used the Arduino RP2040 and board definitions core from the Arduino IDE. The compiler definition ARDUINOMQTT has to be "undef" as there is no WLAN on these boards. With these settings the BASIC interpreter can be compiled and uploaded with the correct board definitions.

In the Arduino IDE, look for Arduino Mbed OS RP2040 boards, download them and use the Raspberry Pi Pico board from this. With this core, native Raspberry PI Pico pin numbers can be used: https://datasheets.raspberrypi.com/pico/Pico-R3-A4-Pinout.pdf

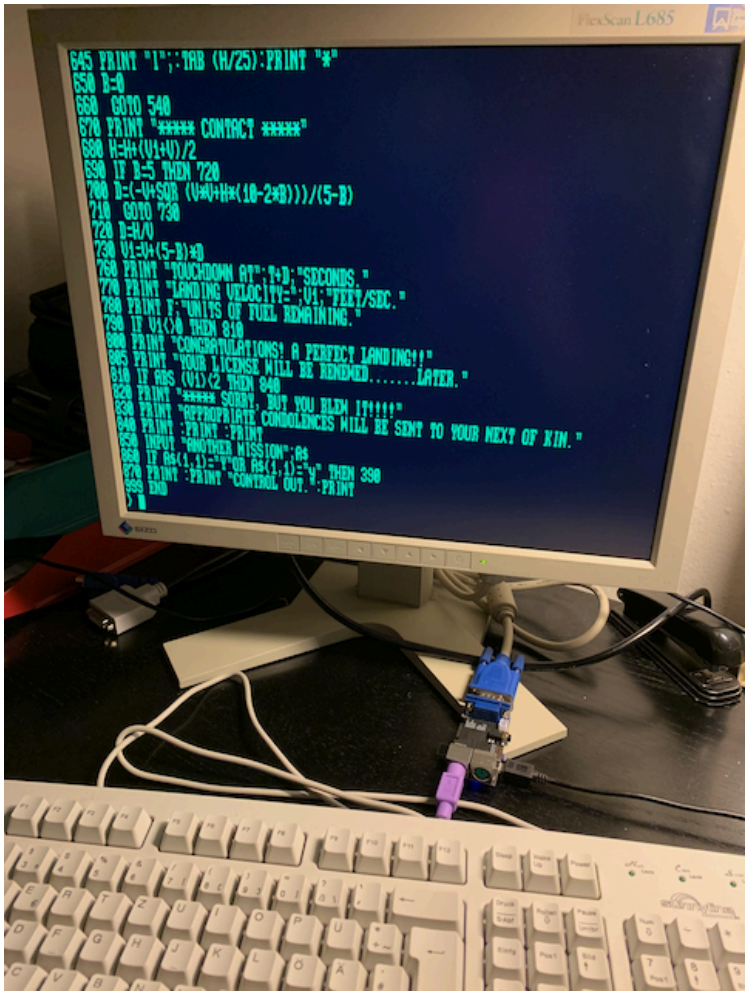
As there is no pin A3 on these boards, the ILI_LED pin has to be changed to A2 instead. SPI is on pins 16 (MISO), 18(SCK), and 19 (MOSI) on these boards.

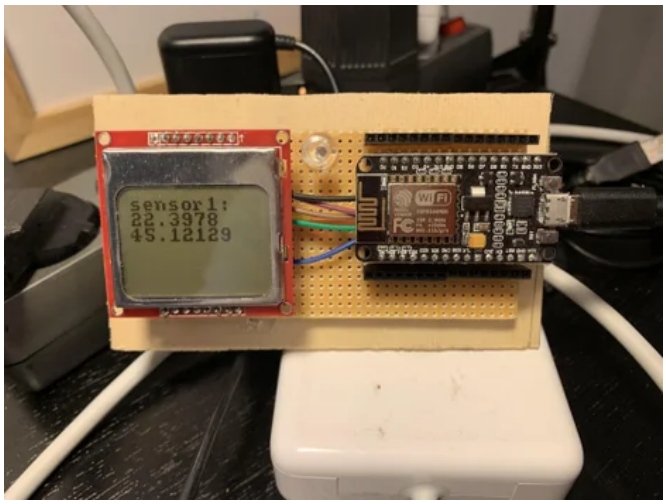
ILI_CS, ILI_RST, and ILI_DC can be set to any of the digital pins. My choice was

```
#define ILI_LED A2
#define ILI_CS 15
#define ILI_RST 14
#define ILI_DC 13
```

as these pins are at the bottom of the board and wiring would be straightforward.

Step 17: Topics Not Covered Here





The BASIC interpreter has a number of features not discussed in this tutorial. It can run as a Wire slave device and also receive and send data with a radio module.

There is a big brother of this system https://www.instructables.com/Build-a-80s-Style-Home-Computer-From-Scratch-From-/ based on an Arduino DUE that resembles an 80s home computer.

There is a smaller system based on an ESP8266 board and a data logger shield https://www.instructables.com/The-10-Euro-IoT-Computer-With-ESP-8266/. It is super low cost. I felt inspired by an AIM 65 when I started to build this.

The little brothers are NOKIA5110 and ESP8266 based with no keyboard, ESP01 based with just sensors, or AVR 8bit based with no internet.

Currently ESP8266 and ESP32 as well as the Arduino 8bit platforms are well supported. All BASIC programs from one platform run on the other systems if there are enough resources.

The interpreter runs on Mac, Windows, Linux and MSDOS with the same features.

A various predefined hardware designs are supported from small to big.

Please look at the wiki in my repo for more information https://github.com/slviajero/tinybasic/wiki.