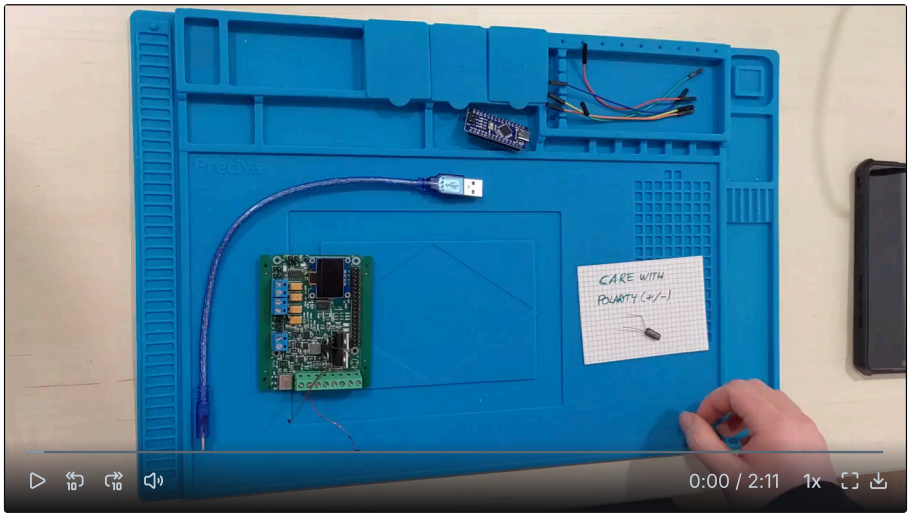
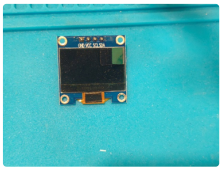
Connect the display

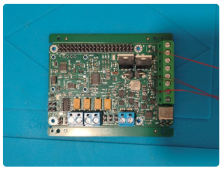


Required components

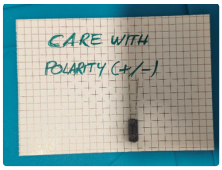
Components



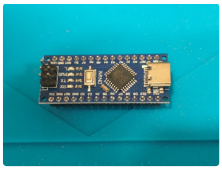
[0.96 Inch OLED Module]



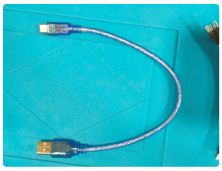
ChargerUnitV1p0



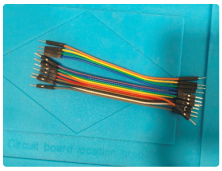
[Capacitador 25v 10uF]



[Arduino NANO]



[Cable USB A a USB C]



[Cables para Arduino hembra – hembra]

Connections

Board 1	Board 2	Cable
Arduino NANO	ATiny 1604	Colours
5V	Pin 2 up	
GND	Pin 10 up	
D6	Pin 10 down	
RST	+ Capacitor	
GND	-Capacitor	

These are the colours we have chosen, but you are free to use others.

⚠ Before doing all the connections, have the `Arduino Nano` connected with nothing but the `USB` cable to the PC to upload the sketch. Once done, it is secure to connect the rest of component.

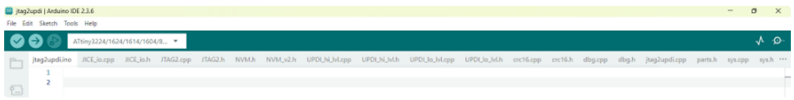
Arduino IDE platform

The first thing you will need to do is install latest version of Arduino IDE (Windows or Linux depending host to use).

Software

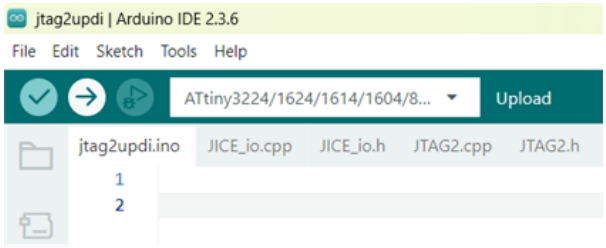
To use the Arduino Nano as a programmer follow the next steps:

1. Download jtag2udpi from GitHub. To do this, enter the link, click on the green button named "Code" and then on "Download ZIP".
2. Save the ZIP file somewhere on your computer and unzip it.
3. In the folder jtag2updi-master you will find the subfolder source, which you rename to jtag2updi.
4. Move the folder jtag2updi into your Arduino Sketch folder (any folder in host where you are developing arduino code). If you want, you can now delete the zip file you previously downloaded.
5. Open the file jtag2updi.ino located inside the folder jtag2updi, with the Arduino IDE. Once you do this you will see all the filese inside the folder, including the sketch file, which, as you can see below, is empty.



- Choose Tools→ Board → Arduino AVR Boards → Arduino Nano.
- Choose Tools → Ports with valid ports where is connected Arduino Nano.

6. Upload the sketch to your Arduino. (Verify or tick symbol for compiling and later, right arrow symbol to upload the compiled file).



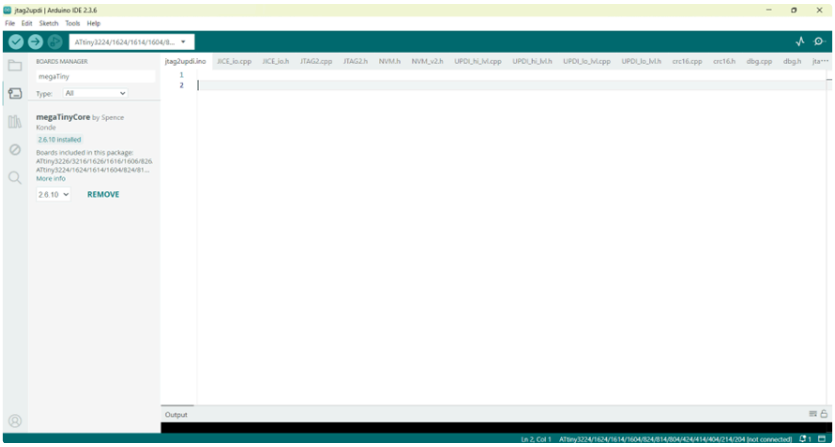
NOTE: Once you have done the steps above and uploaded the compiled file jtag2updi.ino the Arduino Nano will be on bridge mode. In the next steps we will configure Arduino IDE platform for compiling code for our destination microcontroller (Attiny1604).

Configuring Arduino IDE platform for compiling code for Attiny1604 microcontroller.

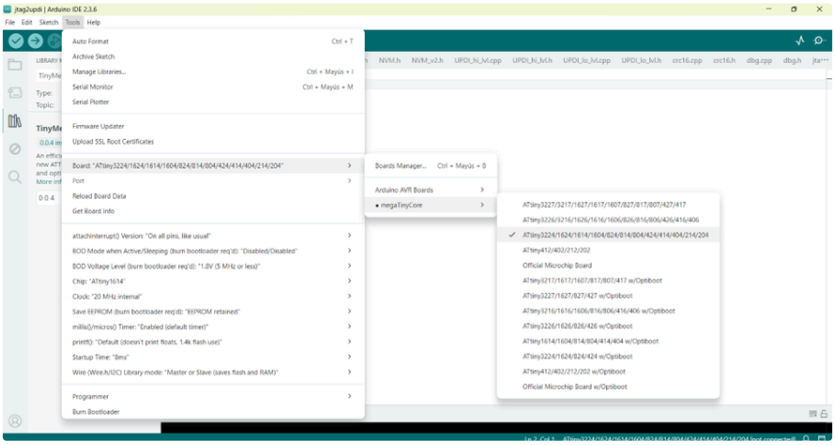
- Install megaTinyCore in the Arduino IDE. To do this, go to File → Preferences. Click on the icon behind "Additional Boards Manager URLs" and enter "**https://descartes.net/package_drazzy.com_index.json**" in a separate line (**the original board manager URL that you find in the installation instructions on GitHub does not work anymore!)**

Close the windows by clicking "OK".

Now navigate to Tools → Board → Boards Manager. Search for the megaTinyCore package from Spence Konde and install it:megaTinyCore



- If everything works fine, you can now choose a board to compile (in our case Attiny 1604). Selecting from “Tools” section. (Following image choose Attiny3224. **You must select Attiny1604**).

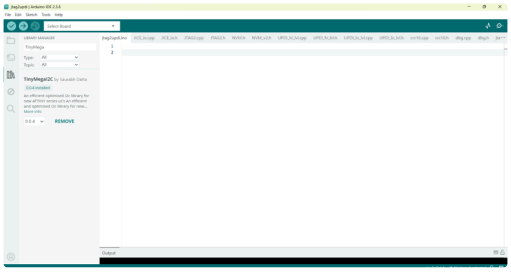


- You also must select as Programmer “jtag2updi” in Tools → Programmer.
- Keep ports same used when Arduino Nano is plug to USB wire.

Now, you have configured Arduino IDE to develop code for Attiny1604 microcontroller.

Library for LCD management

- Install following library: **TinyMegaI2C.**
- Go to program → Include libraries → Library Manager (CTRL + Shift + I) and search for: **TinyMegaI2C.**



Use following code for testing OLED display and using it as base software to more development around the installed display.

Clone or get the zip files here: GitHub - SimonMerrett/tinyOLED: A [WIP] basic SSD1306 OLED driver for ATtiny 0 and 1 series chips using megaTinyCore

Open file examples → basicTest.ino with Arduino IDE platform previously configured to compile for ATtiny1604 microcontroller.

NOTE: You must change these things:

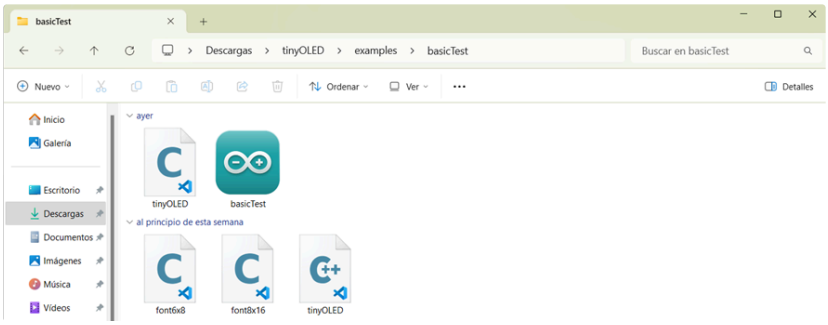
In the file basicTest.ino:

#include <tinyOLED.h> à #include "tinyOLED.h"

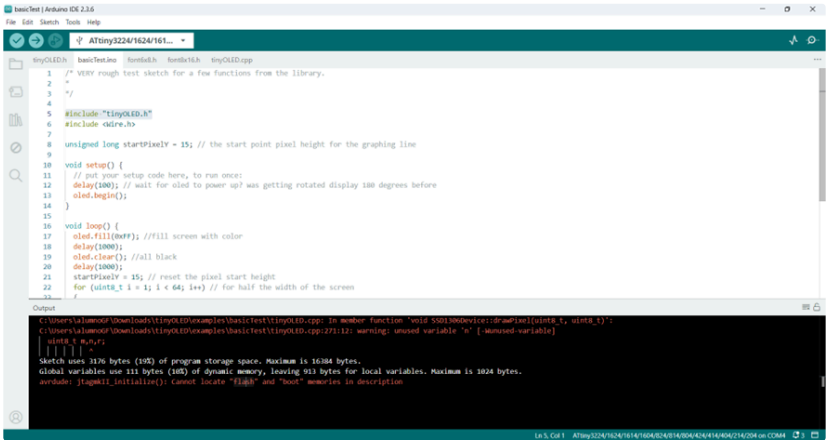
In the file tinyOLED.h:

#include <TinyMegaI2CMaster.h> à #include <TinyMegaI2C.h>

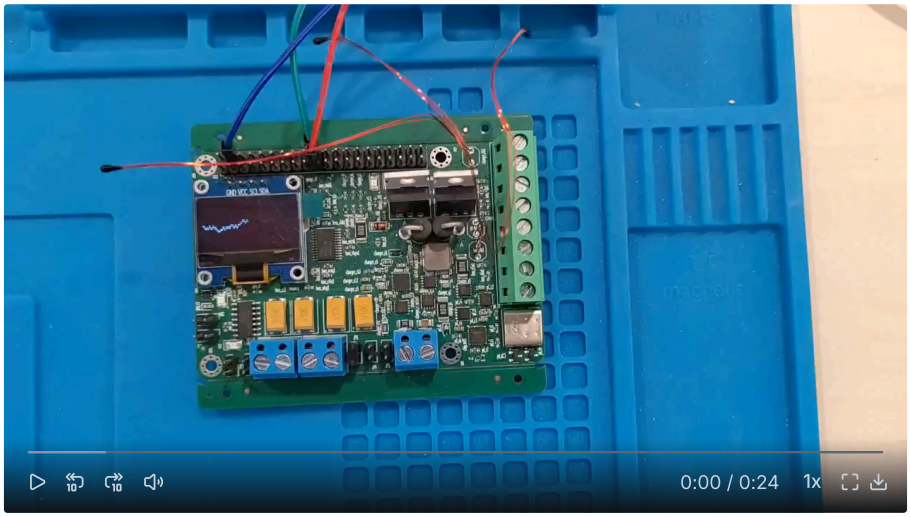
You also must put all this file together. Put all source files (tinyOLED→src files) in same folder where is basicTest.ino file.



Once done this, you can Verify (compile) and Upload to Attiny1604 microcontroller. This is the result message:



And this is the output that is shown on the screen once the sketch is fully uploaded:



Software_LCD_Review

Another example:

Design and implement a battery level indicator using an OLED display (SSD1306) controlled by an ATtiny1604 microcontroller. The display should show a graphical battery icon that depletes over time, and provide visual alerts when the battery level is low or critically low.

The original version of this project was developed and tested **without the ATtiny1604 chip**, using a standard Arduino board as a development platform. In that initial phase, we used the **Adafruit_SSD1306** and **Adafruit_GFX** libraries, which are widely supported and provide high-level functions for drawing graphics and text on OLED displays. These libraries allowed us to implement a complete user interface with battery animations, warnings, and shutdown sequences.

However, when we moved to the final hardware implementation using the ATtiny1604 microcontroller, we encountered compatibility and memory limitations with the Adafruit libraries. In particular, the Adafruit_SSD1306 library requires more resources than the ATtiny1604 can comfortably provide, and functions like display() and drawLine() were no longer usable due to underlying hardware constraints or lack of support in the minimal OLED drivers.

We initially considered the tinyOLED library, which is optimized for smaller chips like the ATtiny series. However, it proved to be too limited for our needs, as it lacks essential graphical functions such as drawRect(), drawBox(), and robust font handling. This made it impossible to replicate the visual features of the original version.

To solve this, we switched to the **U8g2 library**, a lightweight yet powerful graphics library specifically designed for displays with tight memory constraints. U8g2 supports a wide variety of drawing primitives, screen buffering, and font customization, while remaining compatible with the ATtiny1604 through I2C communication. This allowed us to **re-implement the entire interface**, preserving the animated behavior and layout from the original version, while ensuring reliable operation on the final hardware.

Here is the code:

```
#include <U8g2lib.h>
#include <Wire.h>

U8G2_SSD1306_128X64_NONAME_1_HW_I2C u8g2(U8G2_R0, U8X8_PIN_NONE);

int percentage = 100;
bool shuttingDown = false;
bool warningShown = false;

const int batteryWidth = 20;
const int batteryHeight = 8;

const int screenWidth = 128;
const int screenHeight = 32;


  // warning box size
  const int warningWidth = 100;
  const int warningHeight = 24;

  // animation speed
  const int animationDelay = 50;

  // relative offsets for the triangle inside the warning box
  const int triangleWarningOffsetX = 6;
  const int triangleWarningOffsetY = 6;

  // vertex A (bottom right)
  const int triangleWarningAxOffsetX = triangleWarningOffsetX;
  const int triangleWarningAxOffsetY = triangleWarningOffsetY + 8;

  // vertex B (top)
  const int triangleWarningBxOffsetX = triangleWarningOffsetX + 6;
  const int triangleWarningBxOffsetY = triangleWarningOffsetY;

  // vertex C (bottom left)
  const int triangleWarningCxOffsetX = triangleWarningOffsetX + 12;
  const int triangleWarningCxOffsetY = triangleWarningOffsetY + 8;

  // center of the warning box horizontally
  int warningX = (screenWidth - warningWidth) / 2;

  // calculate absolute positions of the triangle
  int triangleWarningAx = warningX + triangleWarningAxOffsetX;
  int triangleWarningAy = triangleWarningAxOffsetY;

  int triangleWarningBx = warningX + triangleWarningBxOffsetX;
  int triangleWarningBy = triangleWarningBxOffsetY;

  int triangleWarningCx = warningX + triangleWarningCxOffsetX;
  int triangleWarningCy = triangleWarningCxOffsetY;


void drawWarning(int y){
  const int rightMarginX = 20;
  const int rightMarginY = 14;

  const String textWarning = "LOW BATTERY";

  // draw the warning box
  u8g2.drawFrame(warningX, y, warningWidth, warningHeight);

  // draw the triangle in the warning box
  u8g2.drawTriangle(triangleWarningAx, y + triangleWarningAy,
                    triangleWarningBx, y + triangleWarningBy,
                    triangleWarningCx, y + triangleWarningCy);
```

```
67
68    // write the warning text to the right of the triangle
69    u8g2.setCursor(warningX + triangleWarningOffsetX + rightMarginX, y + rightMarginY);
70    u8g2.print(textWarning);
71  }
72
73  void showLowBatteryWarning() {
74    if(warningShown)
75      return;
76    // entrance animation loop (moving down)
77    for (int y = -warningHeight; y <= 10; y += 2) {  // animate from above the screen
78      u8g2.firstPage();
79      do {
80        drawWarning(y);
81      } while (u8g2.nextPage());  // draw the page on the screen
82
83      delay(animationDelay);  // animation delay between each step
84    }
85
86    delay(1000);  // keep the warning visible for 1 second
87
88    // Exit animation loop (moving upwards)
89    for (int y = 10; y >= -warningHeight; y -= 2) { // does the same thing as the other for loop but in reverse
90      u8g2.firstPage();
91      do {
92        drawWarning(y);
93      } while (u8g2.nextPage());
94
95      delay(animationDelay);
96    }
97
98    delay(1000);
99    warningShown = true; // marks that the warning has been shown
100 }
101
102 void setup() {
103   u8g2.begin(); // initialize the OLED screen
104   u8g2.setFont(u8g2_font_6x12_tr); // set the font to use for text
105 }
106
107 void updateBatteryStatus() {
108   int batteryLevel = map(percentage, 0, 100, 0, batteryWidth); // converts the percentage into a proportional value to repre
109
110   // formats the percentage text as "xx%"
111   char text[6];
112   sprintf(text, "%d%%", percentage);
113
114   const int widthCharacterPx = 6;
115   const int batteryToTextSpacing = 4;
116
117   int textWidth = strlen(text) * widthCharacterPx; // calculates the text width in pixels (each character is 6px wide) --> o
118   int paddingX = 4; // spacing between elements
119   int totalWidth = batteryWidth + batteryToTextSpacing + textWidth; // calculates the total width of the battery and the tex
120   int startX = screenWidth - totalWidth - paddingX; // horizontal position to align the content to the right
121
122   // coordinates for drawing the battery and the text
123   int batteryX = startX;
124   int batteryY = paddingX;
125   int textX = batteryX + batteryWidth + batteryToTextSpacing;
126   int textY = batteryY + 7;
127
128   // starts the drawing process
129   u8g2.firstPage();
130   do {
131     u8g2.drawFrame(batteryX, batteryY, batteryWidth, batteryHeight); // draws the battery outline
132     u8g2.drawBox(batteryX + 1, batteryY + 1, max(batteryLevel - 2, 0), batteryHeight - 2); // draws the inner charge bar tha
133
134     // draws the battery percentage text
```

```
    // draws the battery percentage text
    u8g2.setCursor(textX, textY);
    u8g2.print(text);

  } while (u8g2.nextPage());

  // displays the warning message if battery is 20% or below and it hasn't been shown yet
  if (percentage <= 20) {
    showLowBatteryWarning();
  }

  if (percentage <= 4) {
    turnOff();

    delay(2000); // time the "shutting down" message stays on screen

    turnOn();
  }

  delay(500); // time delay before reducing the battery percentage
  percentage -= 2; // reduce battery by 2%
}

void turnOn(){
  // resets battery to 100% and clears flags
    percentage = 100;
    shuttingDown = false;
    warningShown = false;
}

void turnOff(){
  if(shuttingDown)
    return;
  const String textTurnOff = "Shutting down...";
  shuttingDown = true; // marks that the system is shutting down
  // displays "shutting down" if battery is 4% or lower
    u8g2.firstPage();
    do {
     // calculating the width of the text
    int textPowerOff = u8g2.getStrWidth(textTurnOff.c_str());

    // Center the text
    u8g2.setCursor((screenWidth - textPowerOff) / 2, screenHeight);
    u8g2.print(textTurnOff);
    } while (u8g2.nextPage());
}

void loop() {
  updateBatteryStatus(); // calling the "updateBatteryStatus()" function
}

```

Code Explanation

This Arduino sketch simulates the behavior of a battery level indicator on a 128x64 OLED display. The battery percentage starts at 100% and decreases over time, with visual updates shown on the screen.

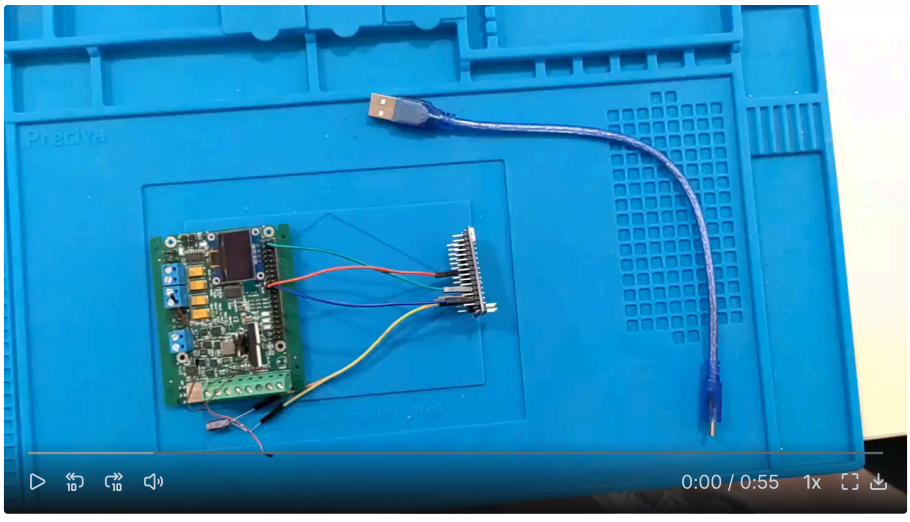
Main Functionality:

- **Battery Indicator:**
  A rectangular battery icon is drawn, with a filled portion representing the current battery level. Next to it, a text label shows the percentage value (e.x., "84%").
- **Animated Warning Message:**
  When the battery drops to 20% or below, an animated warning box labeled “LOW BATTERY” slides down from the top of the screen and then slides back up. This only appears once per cycle.
- **Shutdown Simulation:**
  When the battery drops to 4% or lower, the display shows the message “Shutting down...” centered on the screen. After a short delay, the battery

level resets to 100%, and the process starts again.

- **Loop Behavior:**
  In each loop cycle, the screen is updated with the current battery level. The visual changes occur with short delays to simulate real-time behavior and animations.

And this is the output that is shown on the screen once the sketch is fully uploaded:



Software_LCD_Test2