Metro Minimalist Clock

Created by John Park

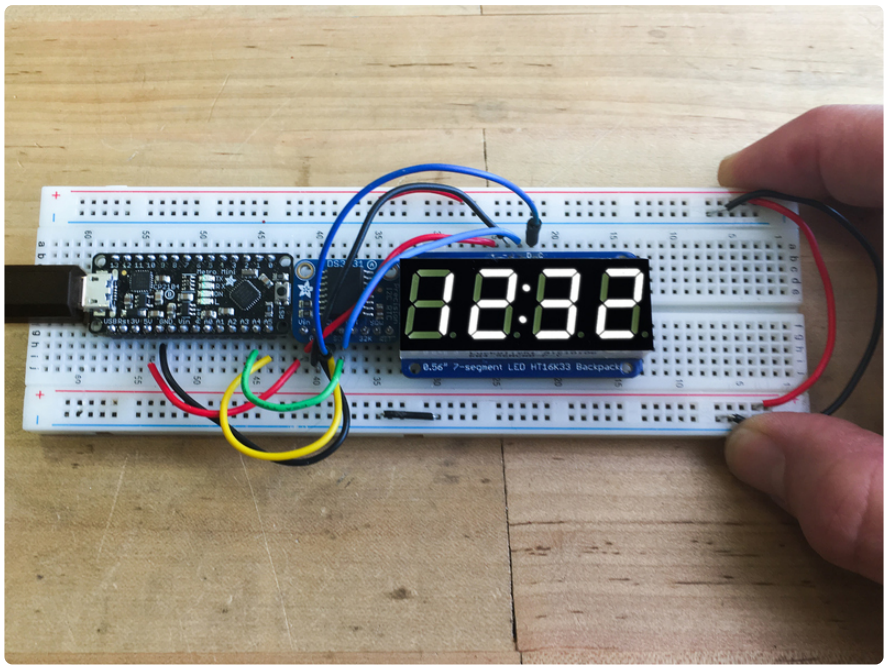
https://learn.adafruit.com/metro-minimalist-clock

Last updated on 2024-03-08 02:37:49 PM EST

Table of Contents

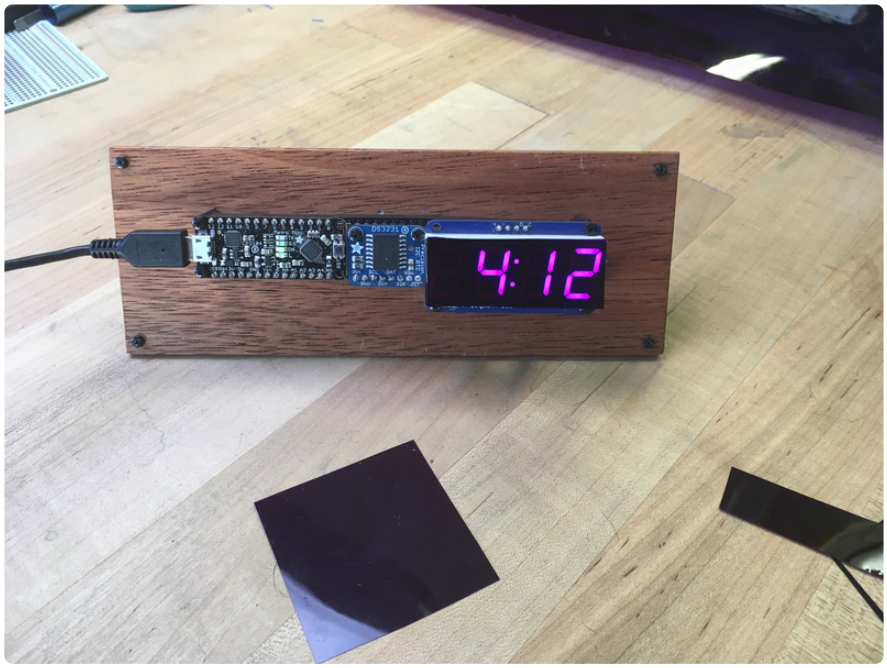
Overview 3
• For this build you'll need:
Clock Circuit 4
Code the Clock 6
Display the Clock 11
• Display Stand

Overview



This Metro Mini microcontroller-based clock is a minimalist timepiece that celebrates, rather than hides, its components. It's a straightforward build, electronically, and it's up to you to house and display it any way you like!

You can build it onto a breadboard, a perma-proto board, or even wire it dead-bug style without a board at all.



For this build you'll need:

Arduino-IDE compatible microcontroller!

1 x Metro Mini Microcontroller — https://www.adafruit.com/product/2590

Arduino-IDE compatible microcontroller!

---

1 x DS3231 breakout — https://www.adafruit.com/product/3013

Real Time Clock breakout board

---

1 x 4 Digit 7-segment Display — https://www.adafruit.com/product/1002

Clock display, pick any color!

---

1 x 5V USB Power supply — https://www.adafruit.com/product/501

We stock or use one you have around the house

---

1 x Breadboard — https://www.adafruit.com/product/239

A full-size breadboard will give you plenty of workspace

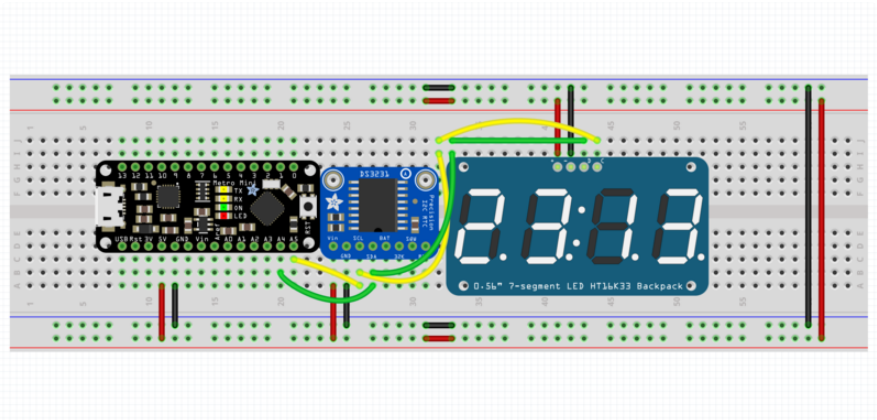
Optionally:

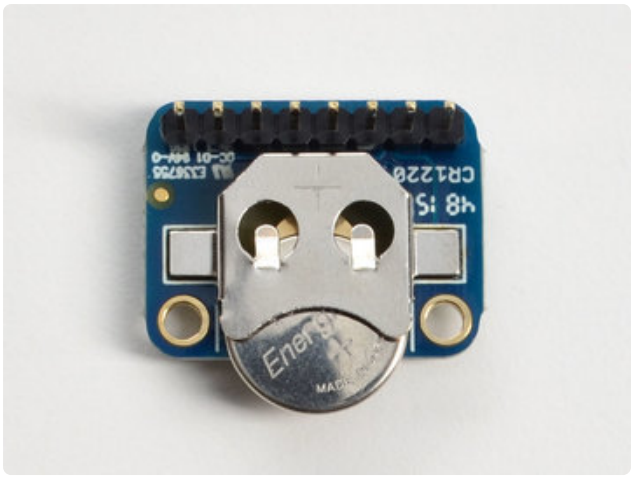
- Perma Proto board, or
- Female header pin strips, x2
- Colored filter gel, tape

Let's have a look at building the electronics next.

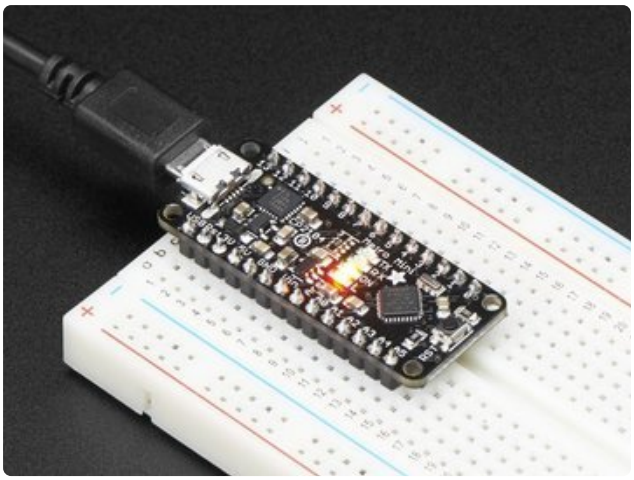
Clock Circuit

You can technically make a clock using only the Metro Mini and the display, but the clock would need to be re-set every time it looses power or is reset. It also isn't super accurate. That's where the real-time clock module comes in. It has a coin cell battery backup, and is a much more accurate timekeeper than the crystal onboard the Metro Mini.





First, solder on the male header pins for the Metro Mini, the RTC module (https://adafru.it/xvA), and the display backpack (https://adafru.it/xvB). You can find more info by following the assembly notes linked here.



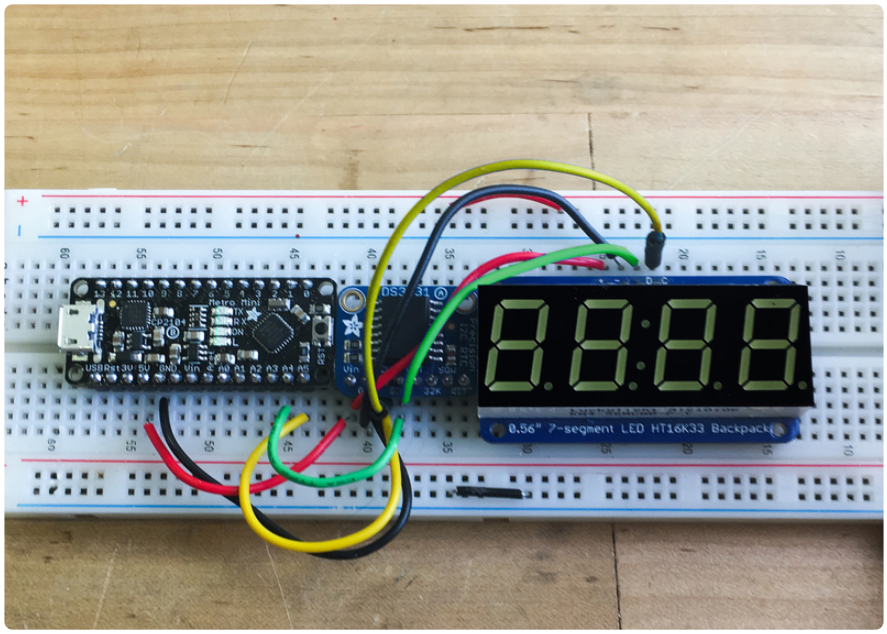
Next, place the three boards onto the breadboard as seen in the diagram above.

Both the RTC module and the display can communicate with the Metro Mini over the I2C protocol, so there are only two pins for needed to connect them all for communications. Grab a breadboard, and then wire them this way:

- Metro Mini pin **A4** to RTC **SDA** to display **SDA**
- Metro Mini pin **A5** to RTC **SCL** to display **SCL**

The other wiring needed is common ground and 5V power among them all. Wire this:

- Metro Mini **5V** to RTC **Vin** to display +
- Metro Mini **GND** to RTC **GND** to display -



Code the Clock

I based my code off of Tony D's excellent Arduino GPS Clock project (https://adafru.it/xvC) and modified it to work with the specific hardware we're using here, as well as eliminating the GPS receiver code he used to auto-set the clock.

First, make sure you're comfortable using the Arduino IDE to upload code to your Metro Mini. You can start here (https://adafru.it/iqd) if you aren't familiar with this, and make sure you can successfully upload the Blink sketch to your Mini.

Next, you'll add three libraries to your Arduino IDE,

Go to **Sketch** > **Include library** > **Manage libraries...** to open the **Library Manger.** Here, search for and install these three libraries:

- **Adafruit GFX Library**
- **Adafruit BusIO Library**
- **RTCLib by Adafruit**
- **Adafruit LED Backpack Library**

Now, you're ready to upload the clock code. Copy the code shown here, then paste it into a new Arduino sketch. Save the sketch as **metroMiniClock.ino**, and then upload it to your board.

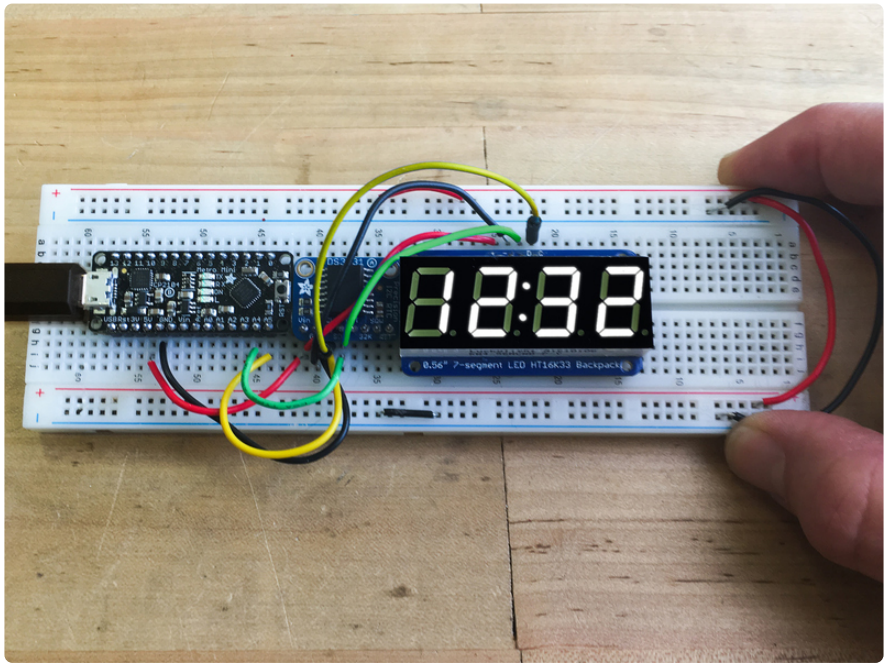
```
// Clock example using a seven segment display &amp; DS3231 real-time clock.
//
// Must have the Adafruit RTClib library installed too!  See:
//   https://github.com/adafruit/RTClib
//
// Designed specifically to work with the Adafruit LED 7-Segment backpacks
// and DS1307 real-time clock breakout:
// ----&gt; http://www.adafruit.com/products/881
// ----&gt; http://www.adafruit.com/products/880
// ----&gt; http://www.adafruit.com/products/879
// ----&gt; http://www.adafruit.com/products/878
// ----&gt; https://www.adafruit.com/products/264
//
// Adafruit invests time and resources providing this open source code,
// please support Adafruit and open-source hardware by purchasing
// products from Adafruit!
//
// Written by Tony DiCola for Adafruit Industries.
// Released under a MIT license: https://opensource.org/licenses/MIT

#include &lt;Wire.h&gt;
#include &lt;Adafruit_GFX.h&gt;
#include &lt;RTClib.h&gt;
#include "Adafruit_LEDBackpack.h"


// Set to false to display time in 12 hour format, or true to use 24 hour:
#define TIME_24_HOUR      false

// I2C address of the display.  Stick with the default address of 0x70
// unless you've changed the address jumpers on the back of the display.
#define DISPLAY_ADDRESS   0x70


// Create display and DS1307 objects.  These are global variables that
// can be accessed from both the setup and loop function below.
Adafruit_7segment clockDisplay = Adafruit_7segment();
//RTC_DS1307 rtc = RTC_DS1307();
RTC_DS3231 rtc;

// Keep track of the hours, minutes, seconds displayed by the clock.
// Start off at 0:00:00 as a signal that the time should be read from
// the DS3231 to initialize it.
int hours = 0;
int minutes = 0;
int seconds = 0;

// Remember if the colon was drawn on the display so it can be blinked
// on and off every second.
bool blinkColon = false;


void setup() {
  // Setup function runs once at startup to initialize the display
  // and clock.

  // Setup Serial port to print debug output.
  Serial.begin(115200);
  Serial.println("Clock starting!");
  delay(3000); // wait for console opening

  // Setup the display.
```

```
  clockDisplay.begin(DISPLAY_ADDRESS);
  clockDisplay.setBrightness(15);
  // Setup the DS3231 real-time clock.
  rtc.begin();

  // Set the clock if it hasn't been set before.

  if (! rtc.begin()) {
    Serial.println("Couldn't find RTC");
    while (1);
  }

  if (rtc.lostPower()) {
    Serial.println("RTC lost power, lets set the time!");
    // following line sets the RTC to the date &amp;amp; time this sketch was compiled
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
    // This line sets the RTC with an explicit date &amp;amp; time, for example to set
    // January 21, 2014 at 3am you would call:
    // rtc.adjust(DateTime(2014, 1, 21, 3, 0, 0));
  }


  /*
  bool setClockTime = !rtc.isrunning();
  // Alternatively you can force the clock to be set again by
  // uncommenting this line:
  //setClockTime = true;
  if (setClockTime) {
    Serial.println("Setting DS3231 time!");
    // This line sets the DS1307 time to the exact date and time the
    // sketch was compiled:
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
    // Alternatively you can set the RTC with an explicit date &amp;amp; time,
    // for example to set January 21, 2014 at 3am you would uncomment:
    //rtc.adjust(DateTime(2014, 1, 21, 3, 0, 0));
  }
*/

}

void loop() {
  // Loop function runs over and over again to implement the clock logic.

  // Check if it's the top of the hour and get a new time reading
  // from the RTC clock.  This helps keep the clock accurate by fixing
  // any drift.
  if (minutes == 0) {
    // Get the time from the DS3231.
    DateTime now = rtc.now();
    // Print out the time for debug purposes:
    Serial.print("Read date &amp;amp; time from DS3231: ");
    Serial.print(now.year(), DEC);
    Serial.print('/');
    Serial.print(now.month(), DEC);
    Serial.print('/');
    Serial.print(now.day(), DEC);
    Serial.print(' ');
    Serial.print(now.hour(), DEC);
    Serial.print(':');
    Serial.print(now.minute(), DEC);
    Serial.print(':');
    Serial.print(now.second(), DEC);
    Serial.println();
    // Now set the hours and minutes.
    hours = now.hour();
    minutes = now.minute();
  }

  // Show the time on the display by turning it into a numeric
```

```
  // value, like 3:30 turns into 330, by multiplying the hour by
  // 100 and then adding the minutes.
  int displayValue = hours*100 + minutes;

  // Do 24 hour to 12 hour format conversion when required.
  if (!TIME_24_HOUR) {
    // Handle when hours are past 12 by subtracting 12 hours (1200 value).
    if (hours &gt; 12) {
      displayValue -= 1200;
    }
    // Handle hour 0 (midnight) being shown as 12.
    else if (hours == 0) {
      displayValue += 1200;
    }
  }

  // Now print the time value to the display.
  clockDisplay.print(displayValue, DEC);

  // Add zero padding when in 24 hour mode and it's midnight.
  // In this case the print function above won't have leading 0's
  // which can look confusing.  Go in and explicitly add these zeros.
  if (TIME_24_HOUR &amp;&amp; hours == 0) {
    // Pad hour 0.
    clockDisplay.writeDigitNum(1, 0);
    // Also pad when the 10's minute is 0 and should be padded.
    if (minutes &lt; 10) {
      clockDisplay.writeDigitNum(2, 0);
    }
  }

  // Blink the colon by flipping its value every loop iteration
  // (which happens every second).
  blinkColon = !blinkColon;
  clockDisplay.drawColon(blinkColon);

  // Now push out to the display the new values that were set above.
  clockDisplay.writeDisplay();

  // Pause for a second for time to elapse.  This value is in milliseconds
  // so 1000 milliseconds = 1 second.
  delay(1000);

  // Now increase the seconds by one.
  seconds += 1;
  // If the seconds go above 59 then the minutes should increase and
  // the seconds should wrap back to 0.
  if (seconds &gt; 59) {
    seconds = 0;
    minutes += 1;
    // Again if the minutes go above 59 then the hour should increase and
    // the minutes should wrap back to 0.
    if (minutes &gt; 59) {
      minutes = 0;
      hours += 1;
      // Note that when the minutes are 0 (i.e. it's the top of a new hour)
      // then the start of the loop will read the actual time from the DS1307
      // again.  Just to be safe though we'll also increment the hour and wrap
      // back to 0 if it goes above 23 (i.e. past midnight).
      if (hours &gt; 23) {
        hours = 0;
      }
    }
  }

  // Loop code is finished, it will jump back to the start of the loop
  // function again!
}
```

Make sure under the **Tools -> Board** menu the **Adafruit Metro** is selected, and under the **Tools -> Port** menu the serial port for the **Metro Mini** is selected. Then press the upload button or click the **Sketch -> Upload** item to send the code to the board.

Once uploaded, the **Metro Minimalist Clock** will start up, and display the time! You can now unplug the USB cable from the computer, and plug it into a wall adapter -- it only needs to get the time from the computer once, from now on it will maintain correct time from the RTC module. Even when unplugged, the RTC clock's battery will keep it ticking.



If you'd like you can change whether the clock displays time in 24-hour or 12-hour time format by slighting changing the sketch code. By default the clock uses 12-hour format and it's set by this line:

```
// Set to false to display time in 12 hour format, or true to use 24 hour:
#define TIME_24_HOUR      false
```

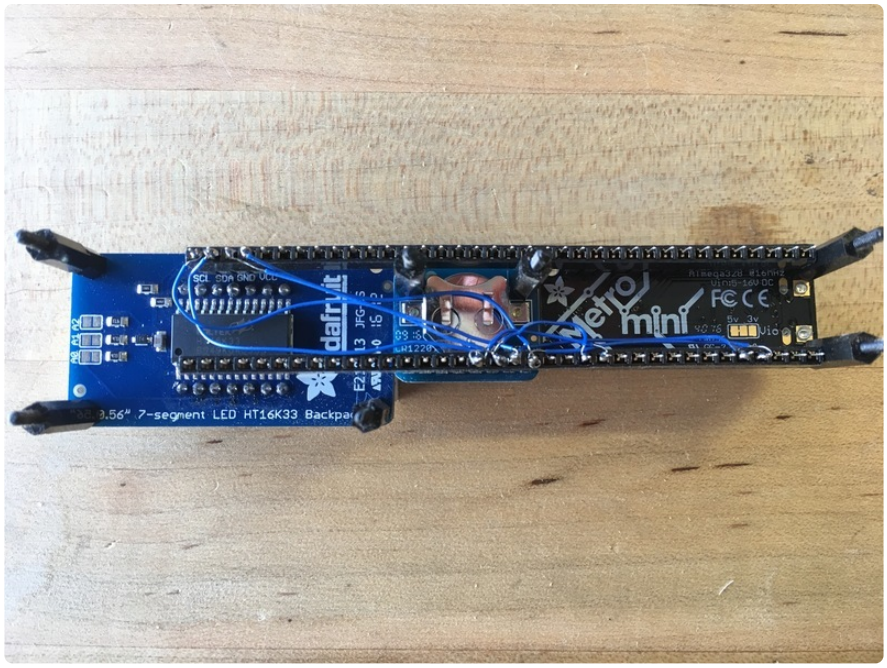
If you'd like to use 24-hour time change the line to set the define as true, like:

```
// Set to false to display time in 12 hour format, or true to use 24 hour:
#define TIME_24_HOUR      true
```

If you ever need to re-set the clock (such as for Daylight Savings Time) simply remove the battery from the RTC module, unplug the Metro Mini so it power cycles, and then re-upload the code. Once it's finished, it'll have the proper current time and you can re-insert the coin cell battery.

Display the Clock

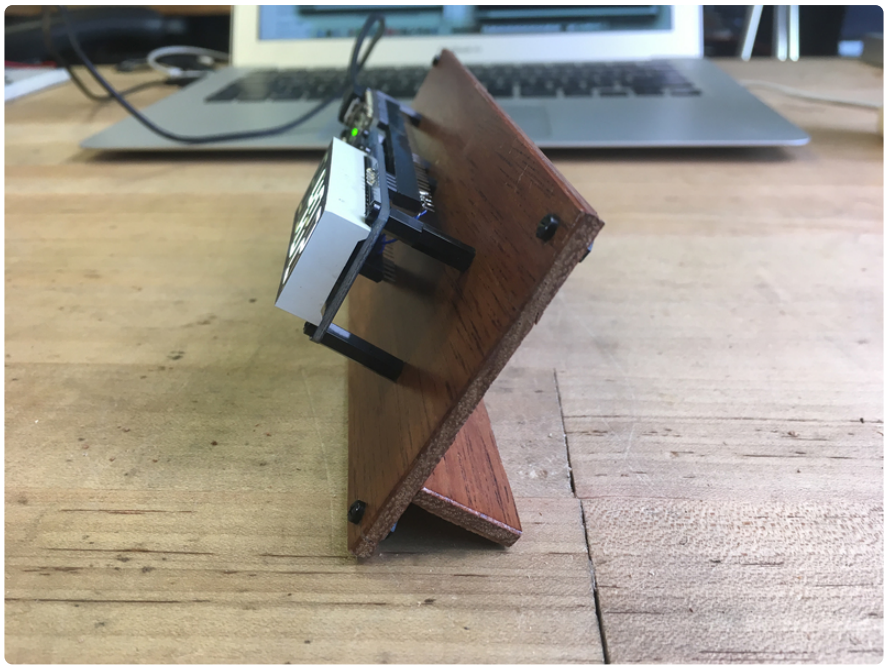
You can use your Metro Minimalist Clock as is, on the breadboard. Or, for a slightly more permanent look, go ahead and solder it onto a Perma Proto breadboard, using the same wiring pattern as before. You can get really minimal and creative with your design by doing a bit of "dead bug" soldering with a couple of strips of female header as seen here.



All of the same connections are being made as before on the breadboard, only now the relevant pins of the female header strips are connected to each other with short lengths of thin wire.

Display Stand

I decided to reuse a short length of hardwood flooring as a display for my Metro Minimalist Clock. This video shows the process, which involved measuring and drilling holes, screwing the electronics in place with 2.5mm nylon standoffs, and gluing on a small stand.

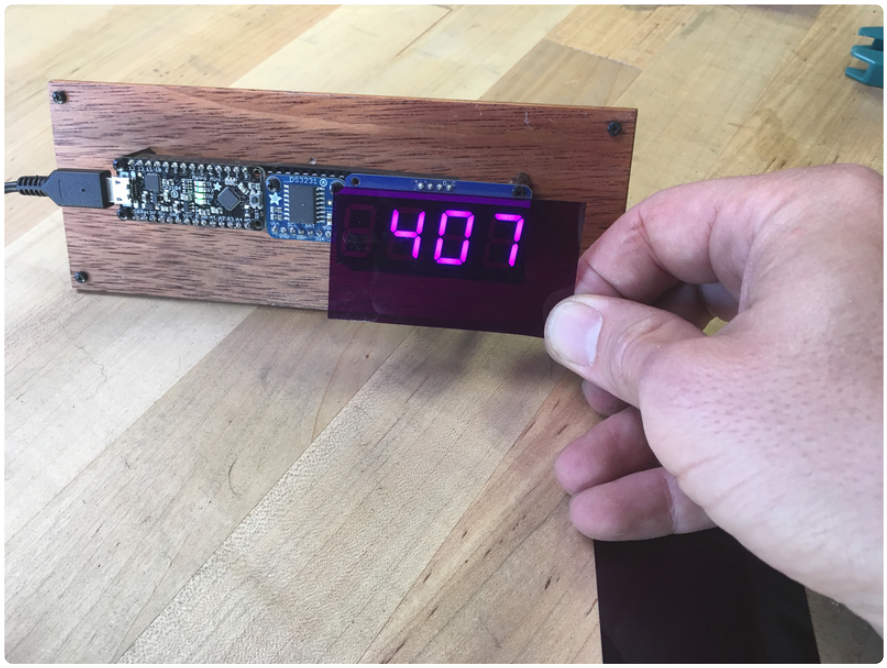


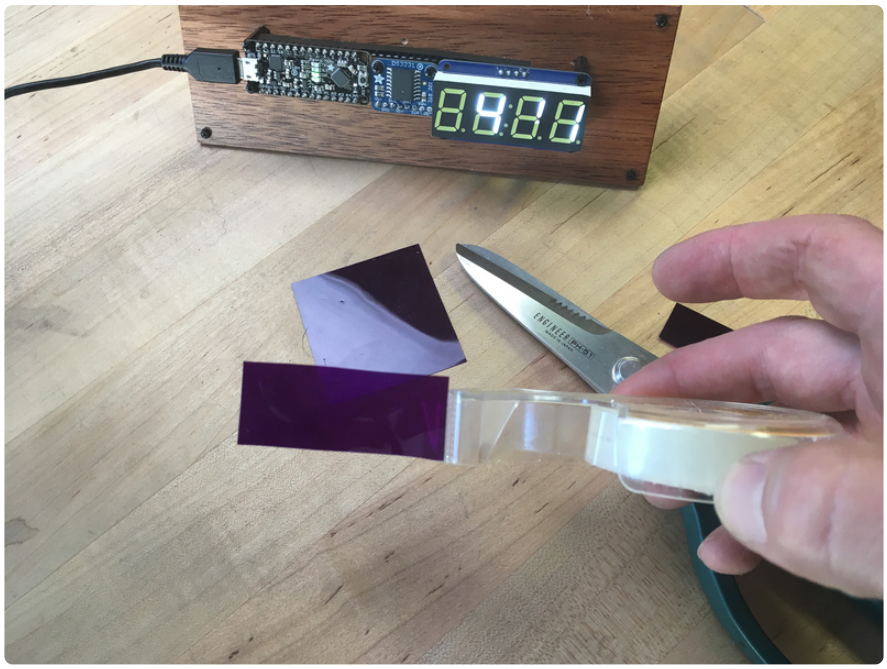
In the end, I decided to cover the raw display with a small piece of purple gel filter -- this helps increase the contrast of the lit segments vs. the unlit ones. Plus, purple is pretty.

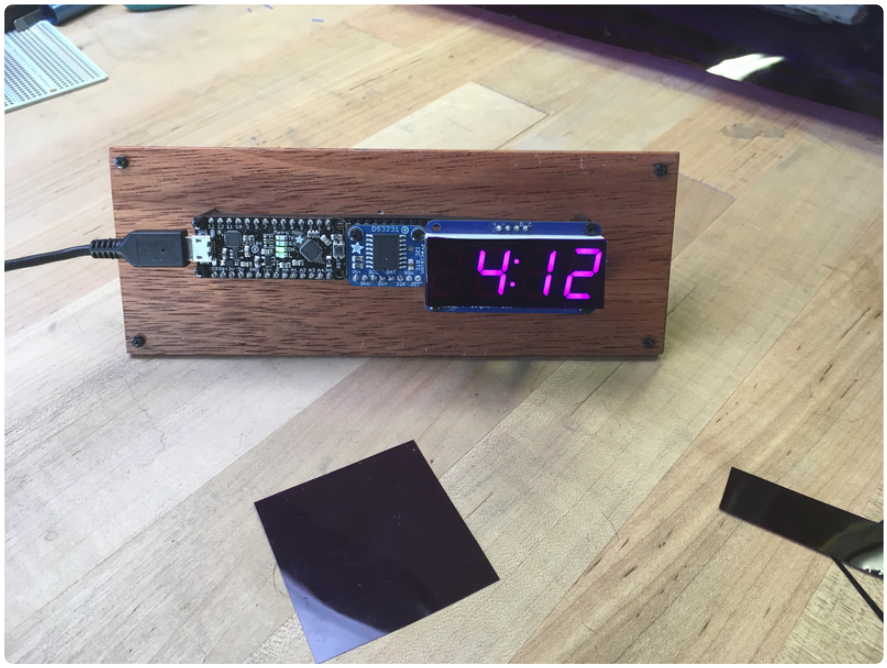
14:04

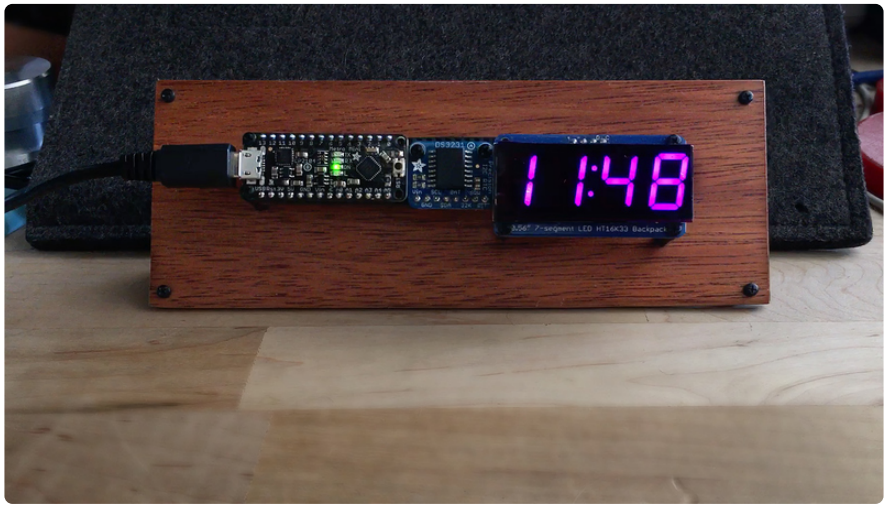
8885

4:05


407




4:12


11:48