adafruit learning system

Mindfulness Clock OF DOOM

Created by Phillip Burgess

Last updated on 2018-08-22 03:54:41 PM UTC

Guide Contents

Overview

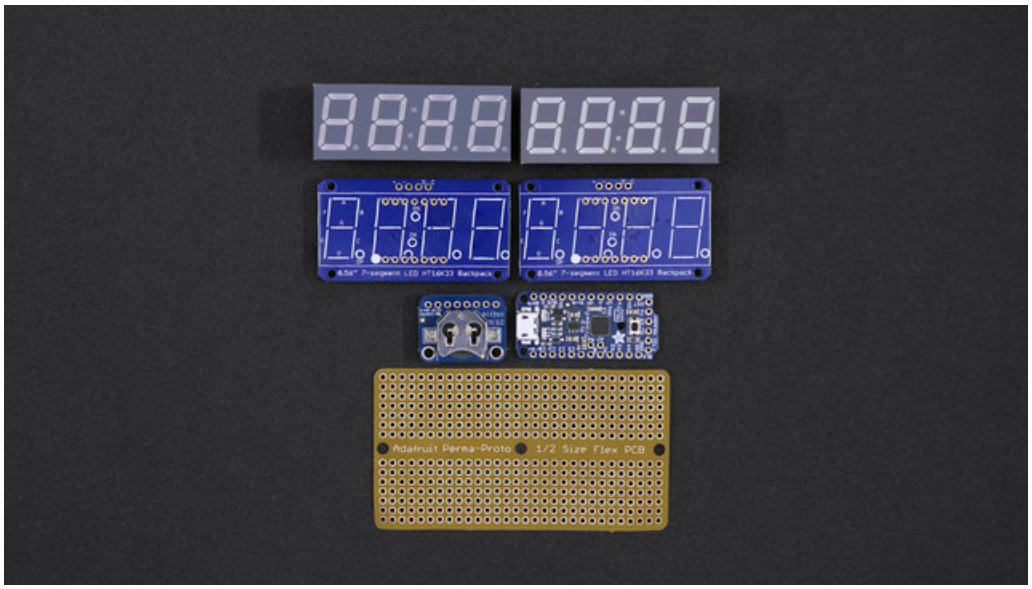
It's not pleasant thinking about one's mortality, but that's the point of this clock design: perspective. To make one aware of the passage of time and **how precious each minute is**.

Use an **online calculator** (some links below) to gauge your **life expectancy**, then plug that date into our code for this clock project. You'll get a running count of how many minutes you have left. Or you can pick another event with a date of your choosing...the return of Halley's comet (28 July 2061), a predicted date for the technological singularity (https://adafru.it/oHC) (2045 according to Ray Kurzweil), a favorite future moment from science fiction, you name it.

As you look at the clock, remember that you are *never* getting those minutes back. Stop watching internet cat videos and **make the most of them!**

Here's a few links to life expectancy calculators that can provide a final date. If one's not working (or you don't like their prediction), try the next:

- **The Death Clock** (https://adafru.it/oGF)
- **Death Timer** (https://adafru.it/oHa)
- **Find Your Fate Death Clock** (https://adafru.it/oHb)



Parts and Tools

- **Adafruit Pro Trinket (http://adafru.it/2000)**, either the **5V (http://adafru.it/2000)** or **3V (http://adafru.it/2010)** version will work, whichever is in stock (with your own enclosure, the code can also work with most ATmega 328P-based microcontrollers such as an Arduino Uno…but NOT 32u4-based boards like the Arduino Leonardo or Adafruit Feather 32u4, nor ATtiny-based boards. The code relies on hardware-specific features.)
- **DS3231 Precision RTC Breakout** (http://adafru.it/3013) (or other **DS3231** realtime clock breakout board, such as the MaceTech ChronoDot (http://adafru.it/255) — again, will need your own enclosure)
- **CR1220 Lithium coin cell battery** (http://adafru.it/380) for RTC. If you can pick this up locally at a nearby drugstore or Radio Shack, do that! This may slow down your shipment otherwise, due to safety regulations with batteries.
- **2** (two) **0.56" 4-digit 7-segment LED displays with I2C backpack** (http://adafru.it/878) (any color — mix or match!)
- **USB power supply** (http://adafru.it/1995) (or use an old MicroUSB phone charger if you've got one)
- **Optional: Perma-Proto board (https://adafru.it/oIC)** for splitting connections.
- **3D printer** and **filament**…or **design your own** enclosure with materials and tools that best suit your abilities.
- M2 x .4 x 4mm Phillips Flat Head Machine Screws (https://adafru.it/pek)

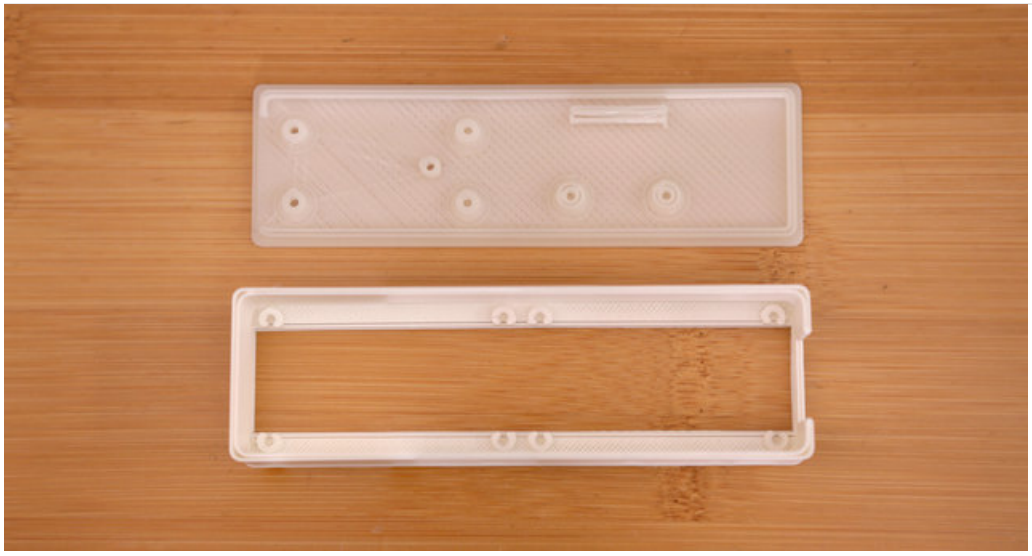
You will also need a **soldering iron** and related paraphernalia, some wire, etc. **Read through the whole guide** before planning any acquisitions.

Realistic Expectations

This is an **art piece** and should not be taken too seriously. Any longevity calculation is loosely based on statistical probabilities; reality will vary, and *we really hope you'll long outlive your clock!*

Of course, unforseen events could happen too. Giant meteor, Cthulu may call, or Drillcat...no clock can predict these surprises.

3D Printing



Let's start with the printing first. You can work on the electronics and software while the printer runs the job.

If doing your own custom build — for example, if using different components — skip ahead to the next page. **The 3D-printed parts are specifically designed around the Pro Trinket board and components listed previously.**

https://adafru.it/pxd

https://adafru.it/pxd

Customize The Design

The parts where modeled in Autodesk Fusion 360, so it's easy to modify the design. The design is public and available to download in different formats. If you'd like to use a different CAD software package, you are free to import the files and remix them.

https://adafru.it/pxe

https://adafru.it/pxe

Slicer Settings

To slice the parts, we used Simplify3D. We recommend using the settings below or use them as reference. We 3D printed the part on a BCN3D Sigma with a 6.mm nozzle. If you have Simplify3D, you can download our profiles below.

https://adafru.it/oHD

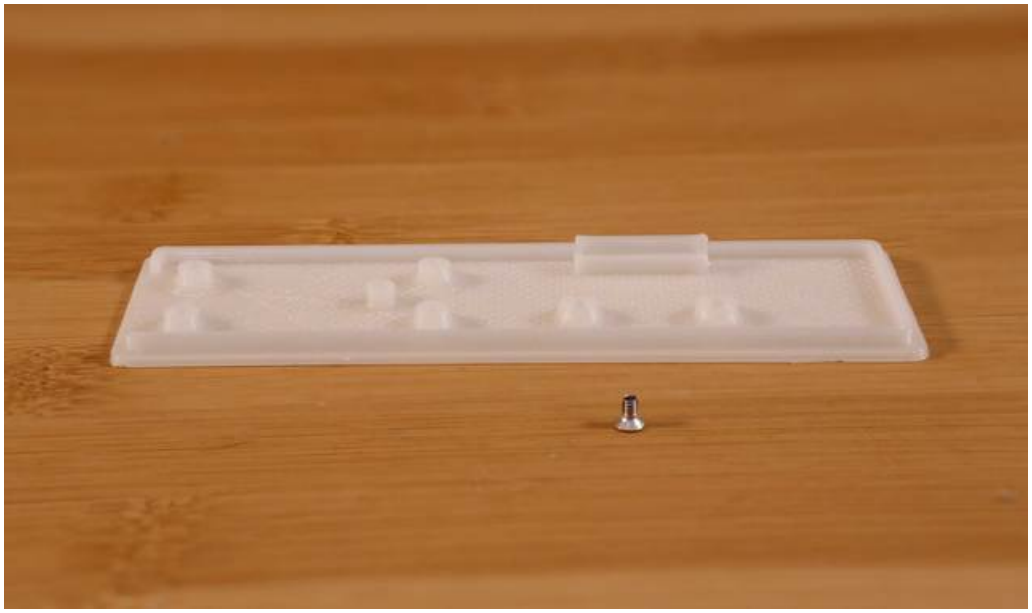
https://adafru.it/oHD

death-lid.stl death-box.stl graveYard.stl	245c extruder 65c bed 50mm/s print speed 120mm/s travel speed 0.6mm Nozzle .3mm layer height .72mm Extrusion width	To speed up printing we used a .6mm nozzle. Using a large nozzle cuts printing down to 20 mins for the box and 15 mins for the lid.
Faceplate for Othermill Pro	Acrylic	2.2mm thick 4in x 4in stock 1/32" Flat End Mill 16400 RPM Spindle 1500 Feed Rate 381 Plung Rate 0.08mm Max Depth

Filament Materials

We recommend using PLA material to reduce wrapping while 3D printing. The parts can be printed in different types of filament, such as ABS, PET or Nylon.

PLA filament is easiest to work with. If printing in ABS instead, you might need to scale the model up slightly (about 2%) to allow for shrinkage.



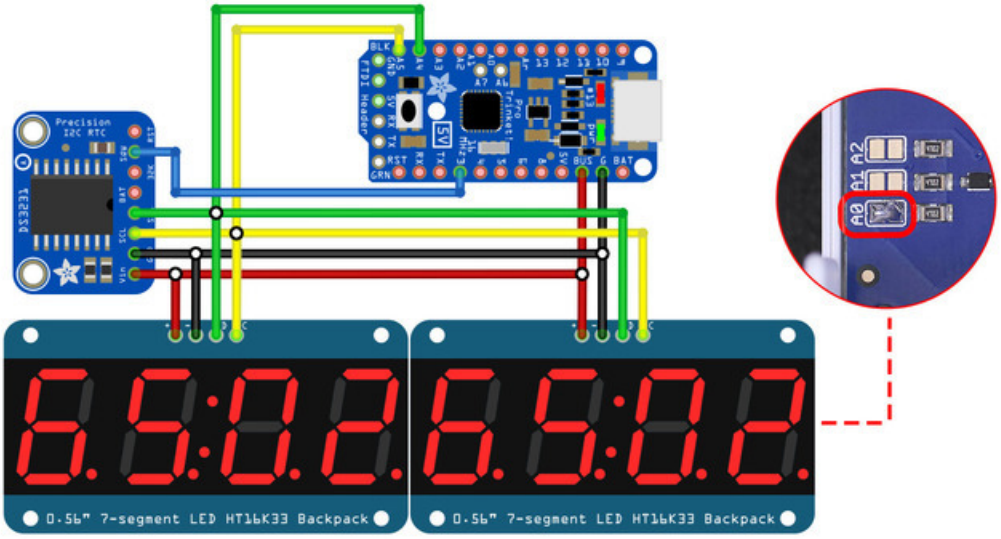
Threads and screws

We used M2 x .4 x 4mm screws (https://adafru.it/pek) for the Trinket Pro, RTC and display boards.

Use a screw (or a *tapping* tool, if you have one) to create the threads for each standoff before mounting the boards. Make sure to apply a bit of pressure and to keep the screw straight while turining.

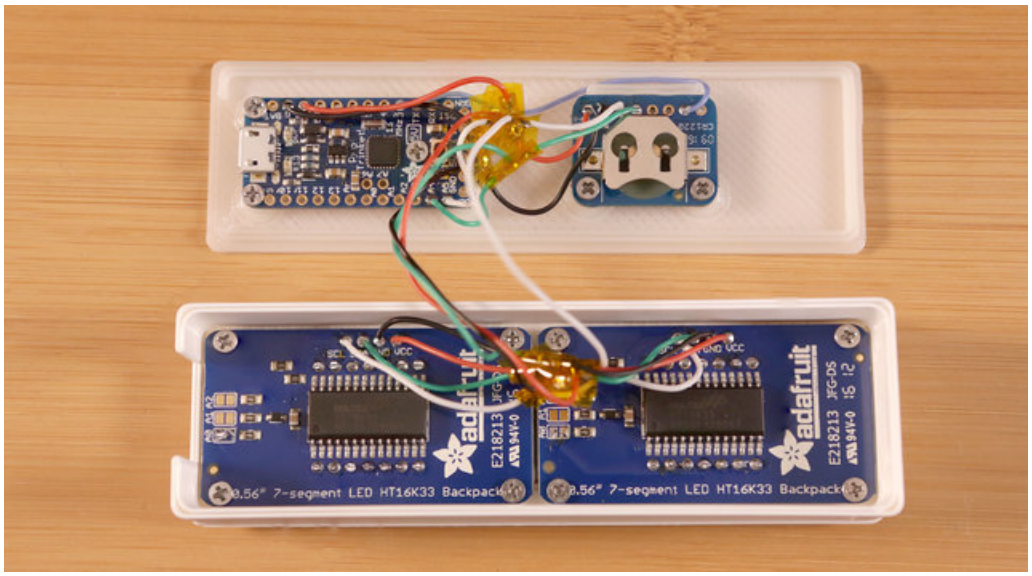
Circuit

In schematic-like form, here's what we're aiming for:



It's basically just four connections in common between all four boards, plus one extra wire between the Pro Trinket and RTC:

Pro Trinket	**DS3231**	**LED Backpacks (2)**
BUS	Vin	+
G or GND	GND	–
A4	SDA	D
A5	SCL	C
3	SQW	

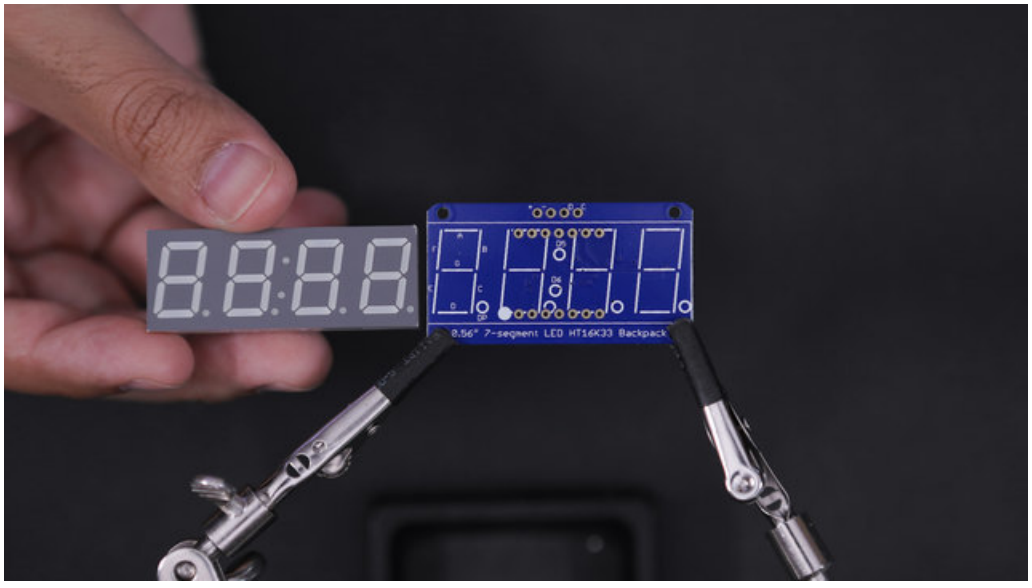


When making connections to multiple points like this, you may find it helpful to use a small piece of **Perma-Proto (https://adafru.it/ola)** board to join one end of all the wires. Perma-Proto board can be cut with tin snips or a scroll saw; you don't need the whole board in there. Save the rest for future projects!

Make sure to use the BUS pin on the Pro Trinket, *not* the 5V or 3V pin; the LED displays are a bit much for the board's voltage regulator, so everything's powered off the USB connection instead.

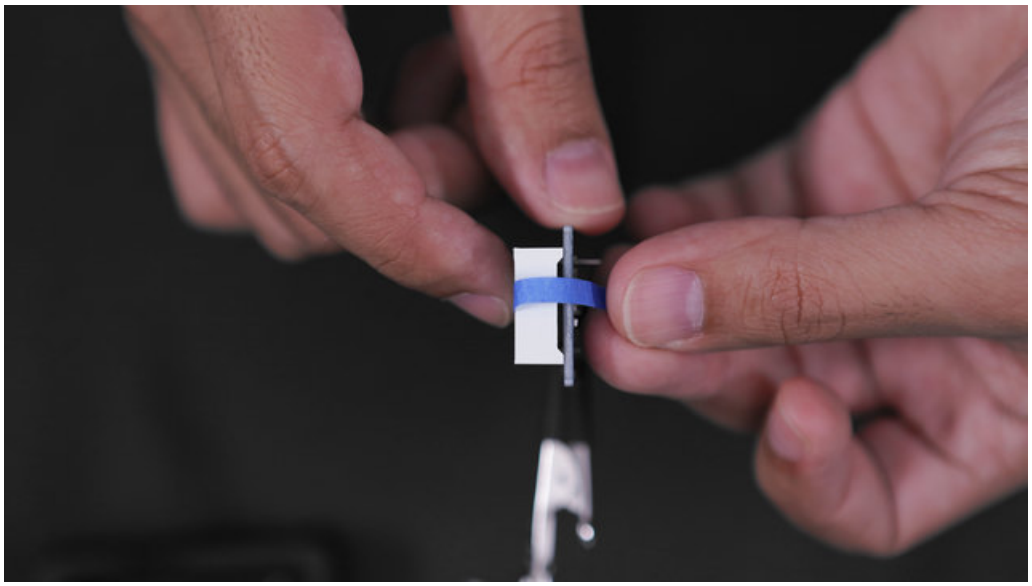
Follow these instructions (https://adafru.it/olf) for guidance on assembling the LED backpacks, making sure the decimal points are properly aligned and the display is on the correct side of the board. Just don't add the 4-pin breadboard header, we'll be wiring up directly.

Bridge the "A0" solder pads on *one* of the two LED backpacks (https://adafru.it/olf). This will be the "low" four digits, while the un-bridged one will be the "high" digita.



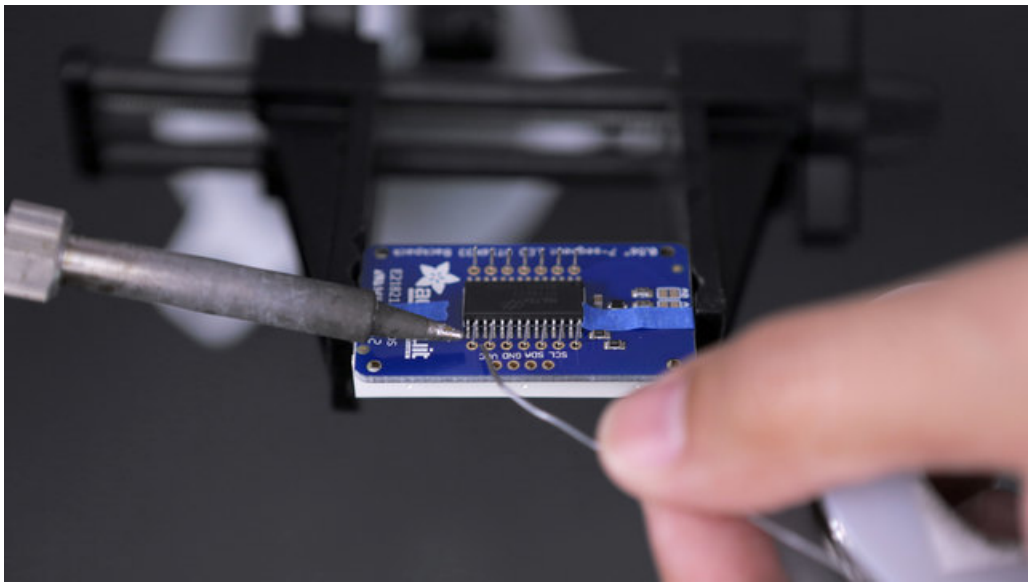
Align display

Note where the decimal points are and properly align the display to the points on the correct side of the board.



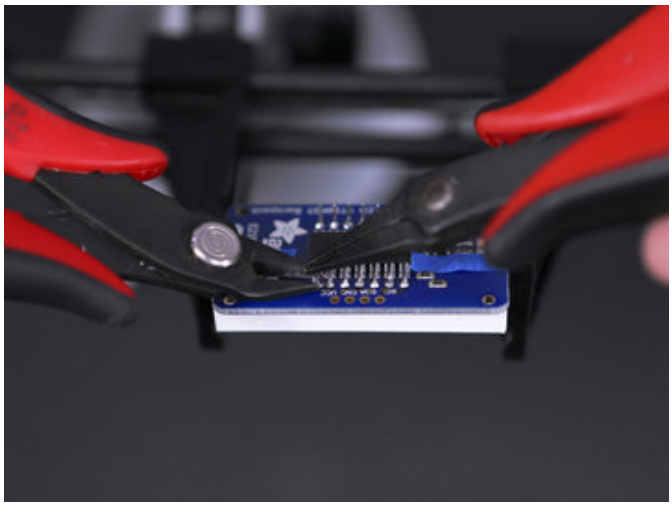
Prep Display

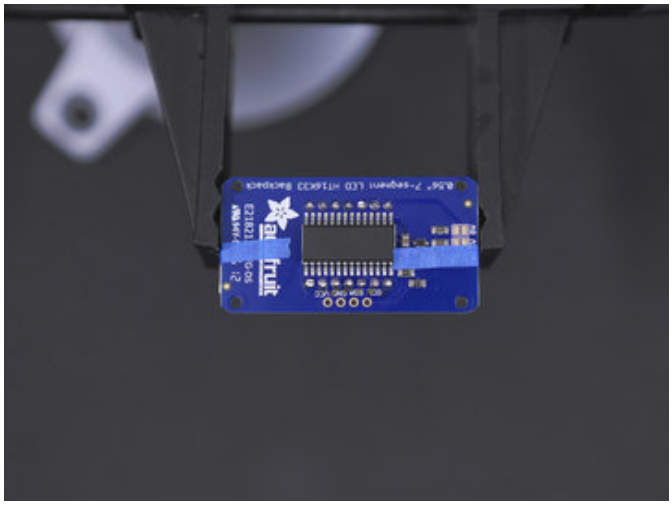
Use tape to hold the displays in place before soldering to the backpacks. Make sure the displays are leveled against the backpack and then apply small strips of tape to both sides.



Solder display

Solder all of the pins to the backpack. A panavise is super helpful and will allow you angle the boards towards you to easily solder each pin.

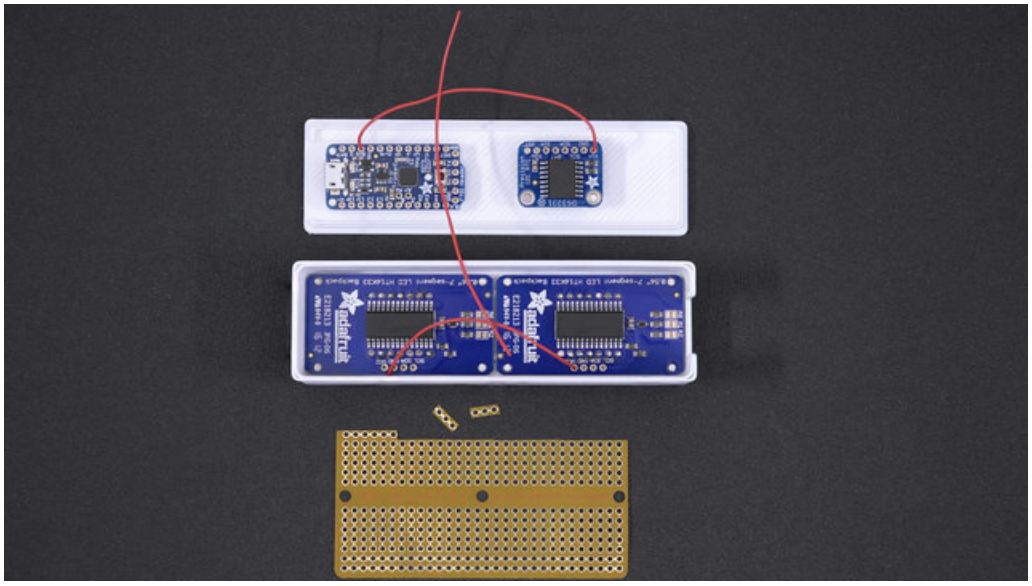




Trim pins

Once all of the pins are soldered, we'll need to cut the excess off each pin off to fit the board inside the enclosure.

Grip each pin with a plier and then use flush cutters to remove each pin. This will prevent the pins from flying everywhere when cutting.



Measure wires

Lay out the boards on the enclosure and lid parts so we have a good idea of how long we need each wire to be. Measure the wires with enough slack so we can easy access the boards when we need to change the battery.

Flex Perma-Proto

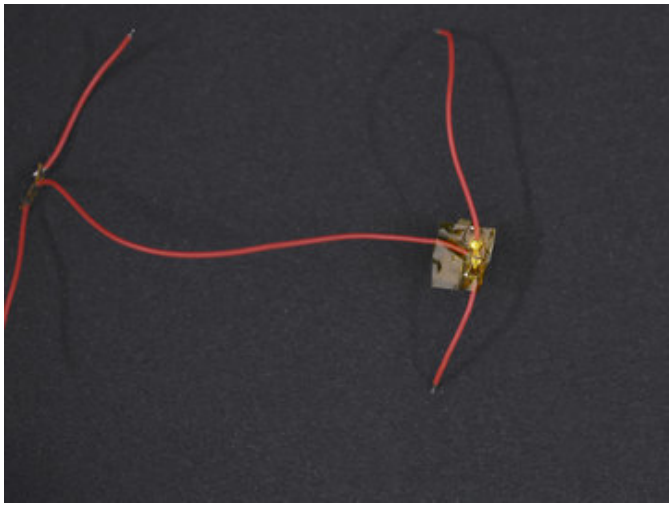
We can use a flex perma-proto to create a splitter to join wires together. Cut a piece with three connected through holes (the ground or power rails) to build a "T" shaped wire.

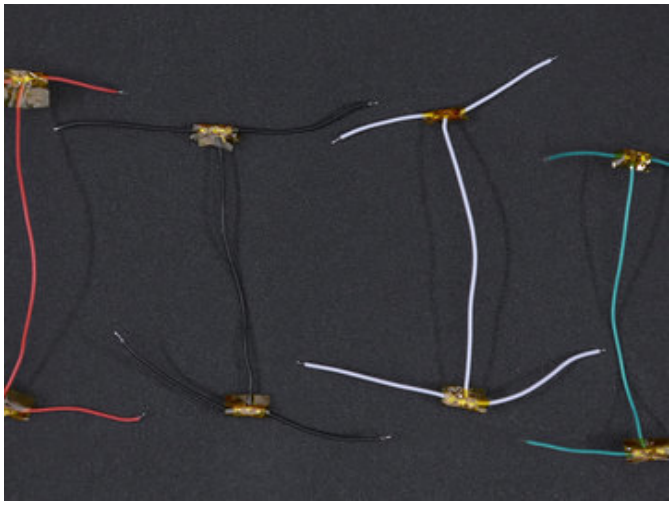
High Temperature Polyimide Tape

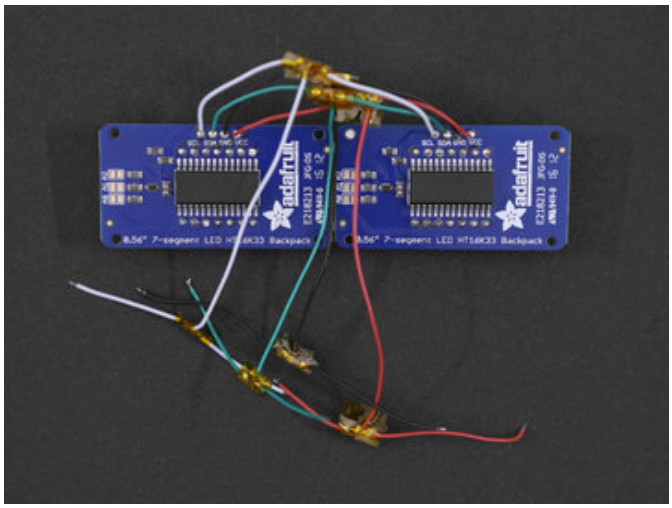
Insulate the connections with a piece of Kapton tape (Polyimide Tape) to prevent any connections from touching.

Solder Displays

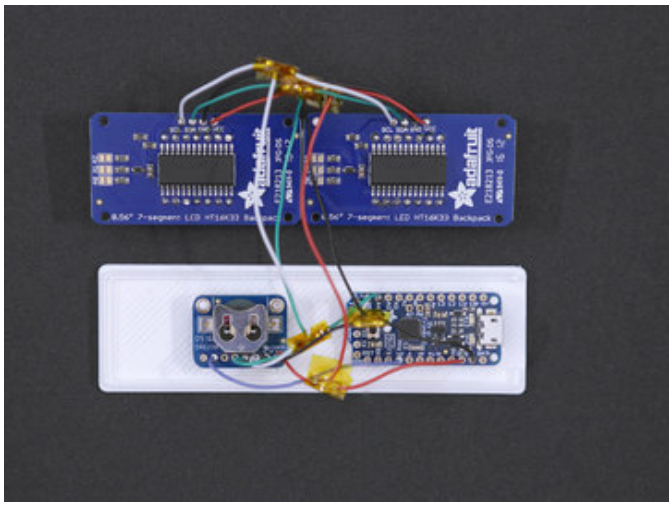
After all of the splitters are built, solder each wire as shown in the circuit diagram.





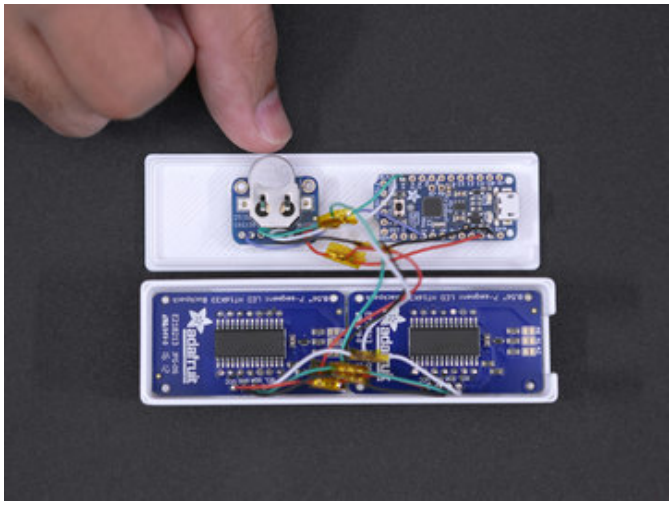


Solder Trinket and Coin Cell Breakout



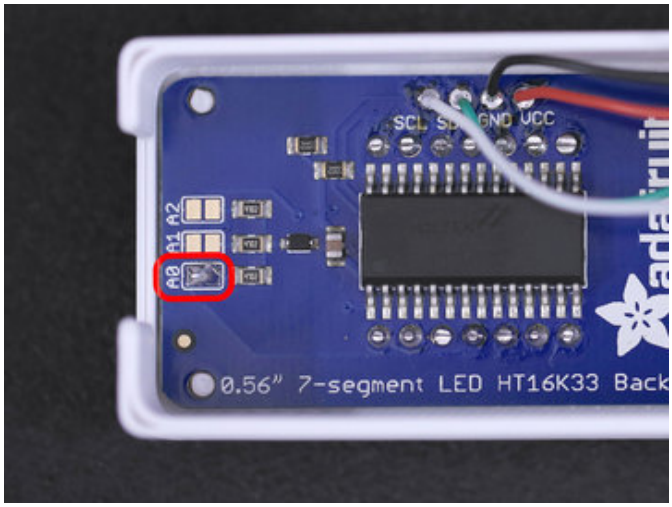
Solder breakouts

Solder the breakout boards to the the rest of the circuit. You can use tweezers to hold
wires while soldering to each through hole.



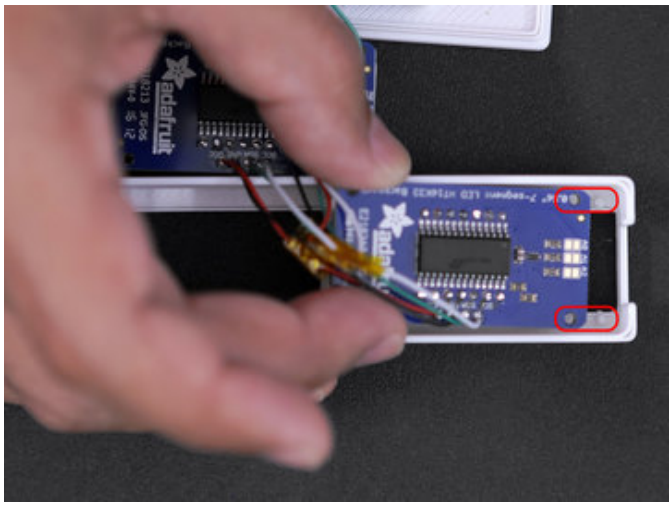
Coin cell

The RTC breakout board uses a 12mm CR1220 coin cell battery (http://adafru.it/380). Note the positive side on the Coin Cell Retainer and insert the battery.



Bridge A0

Bridge the “A0” solder pads on the “low” four digit display (https://adafru.it/olf), the un-bridged one will be the “high” digits.

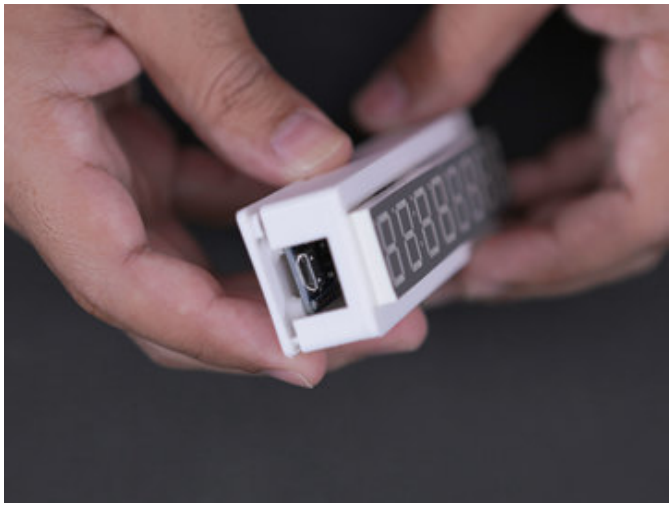


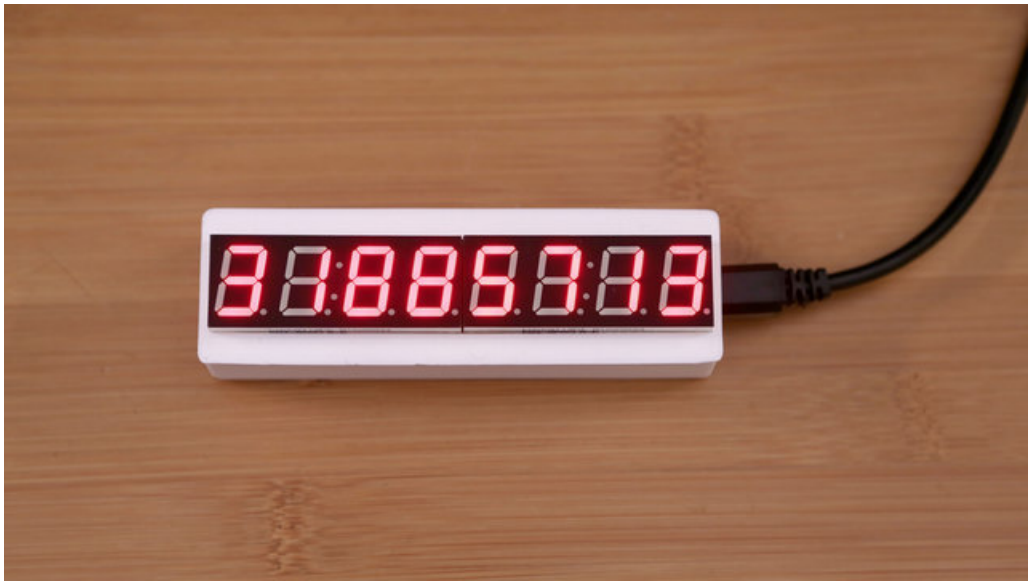
Mount display

Align the displays to the standoffs on the enclosure box and the breakout boards to the lid part. Use M2 x .4 x 4mm machine screws to mount all of the displays and boards.

Close case

The lid part snap fits into the box enclosure. Align the USB cutout on the lid with the port opening on enclosure box to press fit together.





For power, the clock connects to any USB power supply.

Now you're ready to add code!

Software Part 1

If this is your first time using the Pro Trinket microcontroller, you'll want to begin with our guide for setting that up. The Pro Trinket works a little differently from "normal" Arduinos are requires some extra installation and a different upload procedure:

Introducing Pro Trinket (https://adafru.it/iPe)

To confirm that you have the driver installed and IDE properly configured, load the basic Arduino "blink" example sketch and try uploading to the board. If it won't cooperate, work carefully through each of the steps in the guide linked above.

> Do not continue until you have the "blink" sketch successfully working on the Pro Trinket board.

Using the realtime clock and LED displays requires a few extra libraries. These can be installed using the Arduino IDE's Library Manager: **Sketch→Include Library→Manage Libraries...**

Scroll down or use the search field to install **RTClib**, **Adafruit_GFX** and **Adafruit_LEDBackpack**.

The code below will set the initial time on the DS3231 realtime clock. You only need to run this once, or a few years down the line when replacing the clock's backup battery. Copy and paste the code into a new Arduino sketch and upload to the Pro Trinket.

A SOLID RED status LED on the Pro Trinket means the realtime clock was found and time was set. Off or blinking indicates a problem (troubleshooting tips below).

```
// Clock-setting utility for "Mindfulness Clock."  Sets DS3231 battery-
// backed RTC time based on compilation time or a specific data/time value.
// Only needs to be run once to set the clock, or when battery is replaced.

#include <Wire.h>
#include <RTClib.h>

RTC_DS3231 rtc;

void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);

  if(rtc.begin()) {
    // This line sets RTC to date & time this sketch was compiled:
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
    // Or this line sets the RTC with an explicit date & time,
    // for example to set January 21, 2014 at 3am you would call:
    // rtc.adjust(DateTime(2014, 1, 21, 3, 0, 0));
    Serial.println(F("RTC time set"));
    digitalWrite(LED_BUILTIN, HIGH); // LED steady on = RTC set
  } else {
    Serial.println(F("Couldn't find RTC"));
    for(boolean b; ; b = !b) {       // Blinking = RTC error
      digitalWrite(LED_BUILTIN, b);
      delay(250);
    }
  }
}

void loop() { } // Not used -- it's all in setup()!
```

I get an error when compiling the sketch!

- Confirm the correct board is selected (“Pro Trinket 5V/16MHz (USB)” or “Pro Trinket 3V/12MHz (USB)”, whichever you’re using).
- Confirm the RTClib library is correctly installed.

I get a “Done uploading” message, but then the Pro Trinket’s red LED is blinking.

- This usually indicates a communication problem between the Pro Trinket and the RTC. Check all four connections: BUS, GND, A4/SDA/D and A5/SCL/C, make sure the latter two aren’t reversed.
- Look the circuit over for accidental solder bridges or cold solder joints.
- Is the coin cell battery installed on the RTC board?

I just get a solid red LED on the Pro Trinket after uploading.

Success! That means the clock has been set. Proceed to the next page to load the actual countdown software.

Do not continue until you have a solid red LED on the Pro Trinket board.

Software Part 2

Now you can copy and paste all the code (further down this page) into a new sketch. Before uploading to the board, you'll want to edit this section near the top (around line 38):

```
#define YEAR  2061 // Date and time of some future event
#define MONTH    7 // (ostensibly your demise)
#define DAY     28 // But here it's set for Halley's comet
#define HOUR    12 // Example time here is set for noon,
#define MINUTE   0 // DON'T use leading zero or value is octal!
```

This sets the date and time of some future event to count down toward.

HOUR should be in **24-hour** format; e.g. midnight is "0", noon is "12", while 3pm is "15" (12+3). **MINUTE** is **0 to 59**. For both of these values, do not use a leading zero; e.g. for 9:09 am, use "9" for each, not "09". (Numbers with leading zeros are interpreted by the computer as *octal* — a different representation, like hexadecimal or binary.)

After uploading this sketch to the Pro Trinket, you should be greeted with a (hopefully very large) number — the minutes until your set date — and the **rightmost decimal** will be **blinking**. One second on, one second off. If not, or if having other problems, see the troubleshooting tips at the bottom of this page.

```
// Mindfulness clock.  Counts down minutes to some event; presumably one's
// demise.  The display is purposefully not divided into years and days and
// so forth, just an absolute number of minutes, counting down.  It's a
// mindfulness thing; makes you think.  Minutes you're never getting back.
// STOP WATCHING INTERNET CAT VIDEOS AND GET SOMETHING DONE!

// GLOBAL STUFF ------------------------------------------------------------

// PARTS USED:
// - Adafruit Pro Trinket microcontroller, the 5V or 3V versions work
//   equally well, whichever is in stock (adafru.it/2000 or 2010).  Other
//   ATmega 328P-based microcontrollers may work (e.g. Arduino Uno), but
//   NOT 32u4-based boards like the Feather 32u4 or Arduino Leonardo,
//   nor ATtiny-based boards.  Code uses specific hardware features.
// - Adafruit DS3231 Precision RTC Breakout (http://adafru.it/3013).  Other
//   DS3231-based models like the ChronoDot (255) can also work.
// - CR1220 lithium battery (or whatever battery your RTC requires)
// - TWO 4-digit, 7-segment LED Backpacks (adafru.it/877) with LED displays
//   in your choice of colors (red is always very dramatic!)

// NOTES:
// - The two 7-segment displays will be placed side-by-side to create a
//   contiguous 8-digit display.  Bridge the 'A0' pads on the back of ONE
//   display to change its I2C address.
// - Install the LED displays correctly: display goes on the BLANK side of
//   the backpack board and does NOT cover the driver chip.  Make sure the
//   DECIMAL POINTS match the positions on the silkscreen, or it won't work.
// - Connect 'SQW' pin on the RTC board to Digital Pin 3 on microcontroller.
// - The DS3231 dates are valid through year 2100, this code through 2099.
//   Behavior is undefined past those dates and probably won't work, but
//   call it a feature...you and your Arduino have survived over 80 years.

#include <Wire.h>
#include <RTClib.h>
#include <Adafruit_GFX.h>
```

```
#include <Adafruit_LEDBackpack.h>

#define YEAR  2061 // Date and time of some future event
#define MONTH    7 // (ostensibly your demise)
#define DAY     28 // But here it's set for Halley's comet
#define HOUR    12 // Example time here is set for noon,
#define MINUTE   0 // DON'T use leading zero or value is octal!

// GLOBAL STUFF ------------------------------------------------------------

RTC_DS3231 rtc;

struct {
  uint8_t           addr;         // I2C address
  Adafruit_7segment seg7;         // 7segment object
} disp[] = {
  { 0x70, Adafruit_7segment() },  // High digits
  { 0x71, Adafruit_7segment() }   // Low digits
};

// These are declared 'volatile' because they change in an interrupt handler
volatile uint32_t minutes = 0;    // Time remaining (counts down)
volatile uint8_t  seconds = 0;    // Next-minute counter (counts up)
volatile boolean  hzFlag  = true; // true = refresh display

// UTILITY FUNCTIONS -------------------------------------------------------

// This function is called when a high-to-low transition is detected on
// digital pin 3 -- this is connected to the SQW pin on the RTC, which
// provides a super-precise 1-second square wave; won't drift the way
// delay() or millis() would (could even sleep MCU if desired).
// The displays are NOT refreshed inside this function -- it's poor form
// to try doing too much inside an interrupt routine, especially when the
// functions used in updating the displays (via the Wire library) may
// themselves rely on other interrupts.  Instead, a global flag is set here
// and the loop() routine polls this & updates the display when ready.
void interrupt1hz(void) {
  if(++seconds >= 60) {
    seconds = 0;
    if(minutes) minutes--;
  }
  hzFlag = true;
}

// Number of days since 2000/01/01, valid for 2000-2099.  Essentially a
// copy of the date2days() function in RTClib, but it's been declared static
// there and is inaccessible through the API.  So, a duplicate...
uint16_t date2days(uint16_t y, uint8_t m, uint8_t d) {
  static const uint8_t daysInMonth[] PROGMEM =
    { 31,28,31,30,31,30,31,31,30,31,30,31 };
  if(y >= 2000) y -= 2000;
  uint16_t days = d;
  for(uint8_t i=1; i<m; ++i) days += pgm_read_byte(daysInMonth + i - 1);
  if((m > 2) && (!(y & 3))) days++;        // Leap day for current year
  return days + 365 * y + (y + 3) / 4 - 1; // Add leap days for prior years
}

// SETUP() FUNCTION - RUNS ONCE AT STARTUP ---------------------------------

void setup() {
```

```
void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, HIGH);   // LED on = initializing

  for(uint8_t i=0; i<2; i++) {       // Initialize displays
    disp[i].seg7.begin(disp[i].addr);
    disp[i].seg7.clear();
    disp[i].seg7.writeDisplay();
  }

  if(!rtc.begin()) {                 // Initialize RTC
    Serial.println("Couldn't find RTC");
    for(boolean b; ; b = !b) {       // Blinking = RTC error
      digitalWrite(LED_BUILTIN, b);
      delay(250);
    }
  }

  // Determine number of minutes until YEAR/MONTH/DAY/etc.
  DateTime now = rtc.now();
  uint32_t d1  = date2days(YEAR, MONTH, DAY),                    // End day
           d2  = date2days(now.year(), now.month(), now.day()); // Today
  if(d2 > d1) { // Date already passed
    minutes = 0;
  } else {
    int32_t s1 = (HOUR * 60 + MINUTE) * 60L,
            s2 = (now.hour() * 60 + now.minute()) * 60L + now.second();
    if((d2 == d1) && (s2 >= s1)) { // Same day, hour/min/sec has passed
      minutes = 0;
    } else {
      s1     -= s2; // Seconds time difference, may be negative, this is OK
      seconds = 60 - (s1 % 60); // Counts up 0-60 for next minute mark
      minutes = (d1 - d2) * 24 * 60 + s1 / 60; // Minutes remaining
    }
  }

  // Enable 1 Hz square wave output from RTC.
  // rtc.writeSqwPinMode((Ds3231SqwPinMode)SquareWave1HZ); *should* handle
  // this, but for whatever reason is not, so fiddle registers directly...
  uint8_t ctrl;
  Wire.beginTransmission(DS3231_ADDRESS);
  Wire.write((byte)DS3231_CONTROL);
  Wire.endTransmission();
  Wire.requestFrom(DS3231_ADDRESS, (byte)1);
  ctrl = Wire.read() & 0b11100011; // 1 Hz, INTCN off
  Wire.beginTransmission(DS3231_ADDRESS);
  Wire.write((byte)DS3231_CONTROL);
  Wire.write((byte)ctrl);
  Wire.endTransmission();

  // Attach interrupt on pin 3 to 1Hz output from RTC
  pinMode(3, INPUT_PULLUP); // Is open drain on RTC
  attachInterrupt(1, interrupt1hz, FALLING);
  digitalWrite(13, LOW); // LED off = successful init
}

// LOOP() FUNCTION - RUNS FOREVER ----------------------------------------

void loop() {
```

```
  while(!hzFlag); // Wait for interrupt to indicate elapsed second

  uint16_t hi = minutes / 10000, // Value on left (high digits) display
           lo = minutes % 10000; // Value on right (low digits) display
  disp[0].seg7.print(hi, DEC);   // Write values to each display...
  disp[1].seg7.print(lo, DEC);

  // print() does not zero-pad the displays; this may produce a gap
  // between the upper and lower sections.  Add zeros where needed...
  if(hi) {
    if(lo < 1000) {
      disp[1].seg7.writeDigitNum(0, 0);
      if(lo < 100) {
        disp[1].seg7.writeDigitNum(1, 0);
        if(lo < 10) {
          disp[1].seg7.writeDigitNum(3, 0);
        }
      }
    }
  } else {
   disp[0].seg7.clear(); // Clear 'hi' display
  }

  // Flash rightmost dot (1 sec on, 1 sec off)
  disp[1].seg7.writeDigitNum(4, lo % 10, seconds & 1);

  disp[0].seg7.writeDisplay(); // Push data to displays
  disp[1].seg7.writeDisplay();

  hzFlag = false; // Clear flag for next interrupt
}
```

I get an error when compiling the sketch!

- Confirm the correct board is selected (“Pro Trinket 5V/16MHz (USB)” or “Pro Trinket 3V/12MHz (USB)”, whichever you’re using).
- Confirm *all three* libraries are correctly installed: RTClib, Adafruit_GFX and Adafruit_LEDBackpack.

The displays are blank. I get nothing.

- Did you install the wire from **SQW** on the clock board to **digital pin 3** on the Pro Trinket?
- There may be a power or communication problem between the Pro Trinket and the displays. Check all four connections: BUS, GND, A4/SDA/D and A5/SCL/C, make sure the latter two aren’t reversed.
- Look the circuit over for accidental solder bridges or cold solder joints.
- Are the displays correctly assembled per these directions?

Both displays are showing the same four digits. I get two blinking decimals, or no blinking decimal.

Bridge the “A0” pads on the back of *one* display with a bit of solder. This assigns each display a unique I2C address.

The blinking decimal is in the middle, and the digits aren’t counting correctly.

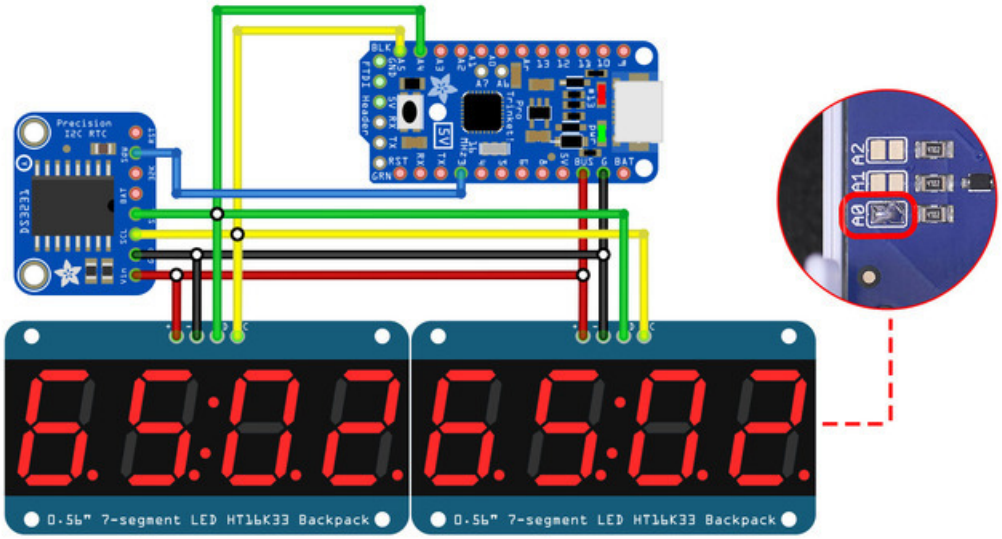
The LED displays are mounted in the wrong positions. No need to dismantle anything, just switch the “0x70” and “0x71” on these two lines (around line 52) and re-upload:

```
  { 0x70, Adafruit_7segment() },  // High digits
  { 0x71, Adafruit_7segment() }   // Low digits
```

Tidbits

Nothing vital to the project here...you can call it done and put it on display now if you like. Just some extra tidbits and trivia for the curious...

That extra wire between SQW and Pin 3...



On most Arduino-type boards, the onboard oscillator (from which all timing — such as the delay() function — is derived) is good, but it's not *that* good. Over time, it may pick up or lose a few seconds per day. But the DS3231 RTC might drift by just a few seconds per *year!*

Rather than having our code continually poll the RTC or using delay() or millis() to refresh the display, a special feature of the DS3231 is used: an optional precise **1 Hz square wave output** can be enabled. After initially reading the clock on startup (and doing some math with dates and times), the code then watches this signal as a second counter. Any subsequent math is *much* simpler! Just needs to count 60 seconds, then decrement the number of minutes.

Pin 3 on the Pro Trinket is special. We can "attach an *interrupt*" to it...when the signal on that pin changes, some code is automatically run. We don't even need to continually call digitalRead(), it just *happens.*

That's how we blink the decimal point at precise 1 second intervals and count minutes, and avoid doing all that time and date math over and over and over...in fact, if we wanted, we could stop the processor completely in a low-power state, having it wake up each time this interrupt occurs. Since this isn't a battery-powered project, and the always-on LED displays use more current than the microcontroller, there's no real practical need for that...but I at least wanted to demonstrate the interrupt feature in action, it might come in handy for your own projects later.

```
void interrupt1hz(void) {
  if(++seconds >= 60) {
    seconds = 0;
    if(minutes) minutes--;
  }
  hzFlag = true;
}

setup() {
  // This will make the interrupt1hz() function run automatically every
  // time Pin 3 changes from HIGH to LOW logic state (a "FALLING" signal):
  attachInterrupt(1, interrupt1hz, FALLING);
}
```

You'll see this same technique in action in our Talking d20 project (https://adafru.it/lkC), which *does* use low-power sleep modes to maximize battery life. It watches for an accelerometer in free-fall, waking the microcontroller when detected.

Mind Tricks

When you glance at your clock, you might notice the decimal point "sticks" for a moment before it resumes blinking. How does the clock know when you're looking at it? Or is this a bug in the code?

Neither! It's an optical illusion known as *chronostasis* or the "stopped clock illusion." Wikipedia has this to say about chronostasis (https://adafru.it/oOE):

> Chronostasis is a type of temporal illusion in which the first impression following the introduction of a new event or task demand to the brain appears to be extended in time. This effect can extend apparent durations by up to 500 ms and is consistent with the idea that the visual system models events prior to perception.

Basically, *perception* — making *sense* of what you see — runs a little slower than *visual sensation* — when photons are absorbed by the retina. Certain times (e.g. fast eye movements, as when glancing over to a clock) the mind has to fill in the gaps. Hilarity ensues.

Science!