We will build several kinds of clocks, including ones based on NTP, GPS, and WWV time sources.

This Arduino project will get you experienced in building with Arduino and with interfacing various parts to an Arduino.

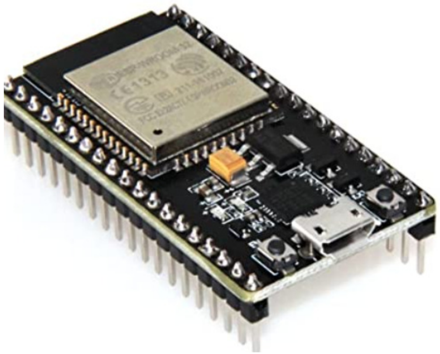
We will build this project in several steps. If you acquire the parts in the Bill of Material, then you can also build these clocks. I will first demonstrate the clocks and then we can hold a workshop where you can build your own clocks.

After you build a clock, what now? See the chapter WHAT NOW? below.

Contents

BILL OF MATERIALS

- HiLetgo ESP-WROOM-32 ESP32 ESP-32S Development Board, available at Amazon for $11. You can use a different microcontroller but you will need to adjust your program and wiring.



<< Upload button

<< Reset button

- Computer to host software and power the ESP32
    - Windows, Mac, Linux, including Raspberry Pi, are supported
- USB cable to connect them together: MicroUSB to USB-A
- 10 uF electrolytic capacitor to make it easier to upload a sketch to the ESP32.
- Breadboard and jumper wires
  From Amazon for $11: DEYUE 3 Set Standard Jumper Wires Plus 3 Set of Solderless Prototype Breadboard 830 tie Points Breadboard | 3 Set of M/F, M/M, F/F - Each 40pin Electronic Jumpers Wire
  https://www.amazon.com/Standard-Jumper-Solderless-Prototype-Breadboard/dp/B07H7V1X7Y/
    - Breadboard full size
    - Jumper wires
        - Male-male if all parts are on breadboard
        - Male-female if some parts are not on breadboard
        - Female-female if all parts are not on breadboard
    - Or you can just use wires that you cut, strip, and insert into the breadboard
- From Amazon for $17: Display 320x240 pixel TFT on a carrier board using the ILI9341 driver and an SPI interface. I used “HiLetgo ILI9341 2.8" SPI TFT LCD Display Touch Panel 240X320 with PCB 5V/3.3V STM32”
  https://www.amazon.com/HiLetgo-240X320-Resolution-Display-ILI9341/dp/B073R7BH1B/
  which includes a touch controller, which we will use, and an SD card reader, which we will not use for these clocks.
- GPS receiver. There are several that you can use. I ordered one from Universal-solder.ca as they also provide a WWV receiver and I could save on shipping. The GPS receiver is ATGM336H GPS Receiver Module, SKU: 26793, for $10. Other ATGM336H modules are available from Amazon but I have not tested them. I also tried Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates (PA1616S) for $30. These GPS modules typically include a battery to power the included Real Time Clock to provide time while the module is not receiving a GPS signal. The Universal-Solder module includes a very tiny battery while the Adafruit module includes a much larger battery. The Universal-Solder module includes an external antenna while the Adafruit module can be attached to an external antenna but that is extra. I recommend the Universal-Solder module.
- WWV receiver from Universal-solder.ca, 60kHz WWVB Atomic Clock Receiver, SKU: 26019, for $13. I could find no alternate sources for this. Read about the WWV receiver below before you decide on this module. After my experiments, I cannot recommend it.

Note that some breakout boards (printed circuit boards that break out the signals from the surface mounted modules to pins that we can use on our breadboards) arrive with the header pins attached, and others with the pins loose. In the latter case, you will need to solder the pins to the breakout board. If you haven't done this, we can do this in a workshop. The WWV receiver also includes a crystal that must be soldered in place.

Pins can be soldered to either side of the breakout board. I usually solder them so that I can see the labels on the pins. This may cause it to be easier or harder to see what is happening on the board. For example, on the ATGM336H GPS board, the LED is on the underside, but it is bright enough that this is workable.

SET UP ARDUINO IDE

If you haven't worked through Session 1 on how to use the Arduino IDE software, you must do that before you work through this Session, such as the chapter on **Building a Simple Arduino Project** starting on page 7, so that you have practiced creating a sketch (Arduino program) and uploading it to the ESP32 microcontroller.

INSTALL LIBRARIES

You will need to add libraries to make the clocks functional. I found that it was easy to install some libraries but a little more work to install others.

What libraries do you need? Look at the beginning of the sketch for statements like:

```
#include <TFT_eSPI.h>
```

In this case, the library name is "TFT_eSPI." When you install the library, several files are included including "TFT_eSPI.h" and "TFT_eSPI.cpp" and you will always find both files in a library.

To install a library, in the Arduino IDE, select Tools > Manage Libraries. The library manager will appear at the left of the current sketch. You will also see the icon 📚 appear, and you can click this icon to open the library manager.

In the Search box, type the name of the library that you want. In this case, it is "TFT_eSPI" and if you type that in, the library manager will show the status of this library. If it is already installed, you will see INSTALLED under the library name. If there is a newer version or if you have not installed the library yet, you will see INSTALL and you can click this to install this library.

If the library you want is not shown, you may need to download it from its github.com repository and then install it.

NTP CLOCK

You can view this clock and how to build it at http://w8bh.net/NTP_DualClock.pdf

This clock provides an NTP based clock and will run on the circuit we have breadboarded without any changes to the sketch. I presented this clock at the first Electronics Workshop session where I also discussed the Arduino.

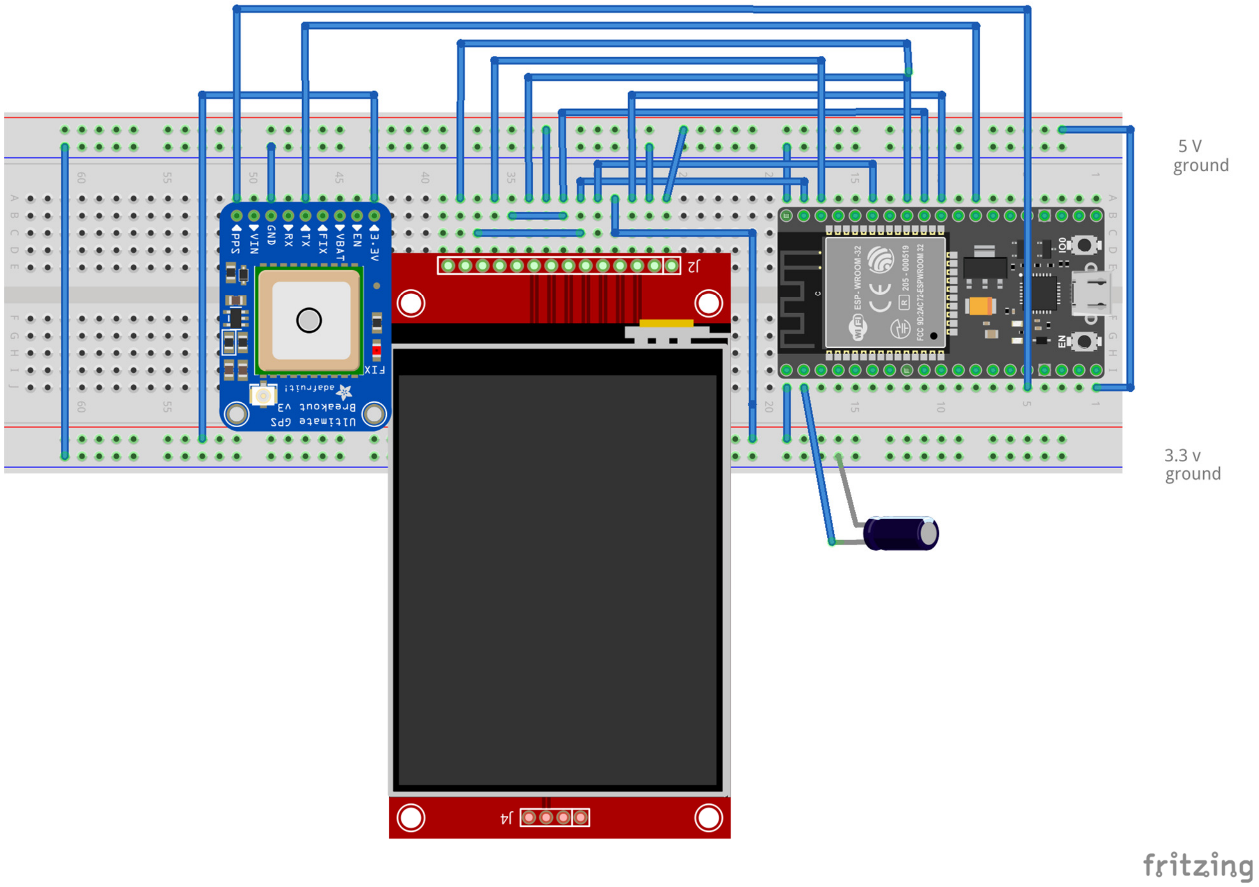
In order to wire the modules together, you will need to know where to connect to the ESP32. If you got the Hiletgo ESP32, the shipment included a diagram of what physical pin corresponds to what GPIO connection. The board is also labeled on the underside but the labels are really small. If you got a different ESP32, then you will need to find the GPIO pins. The pins may be labeled or you may need to search using Google.

The TFT Display and touch Control are labeled on the module.

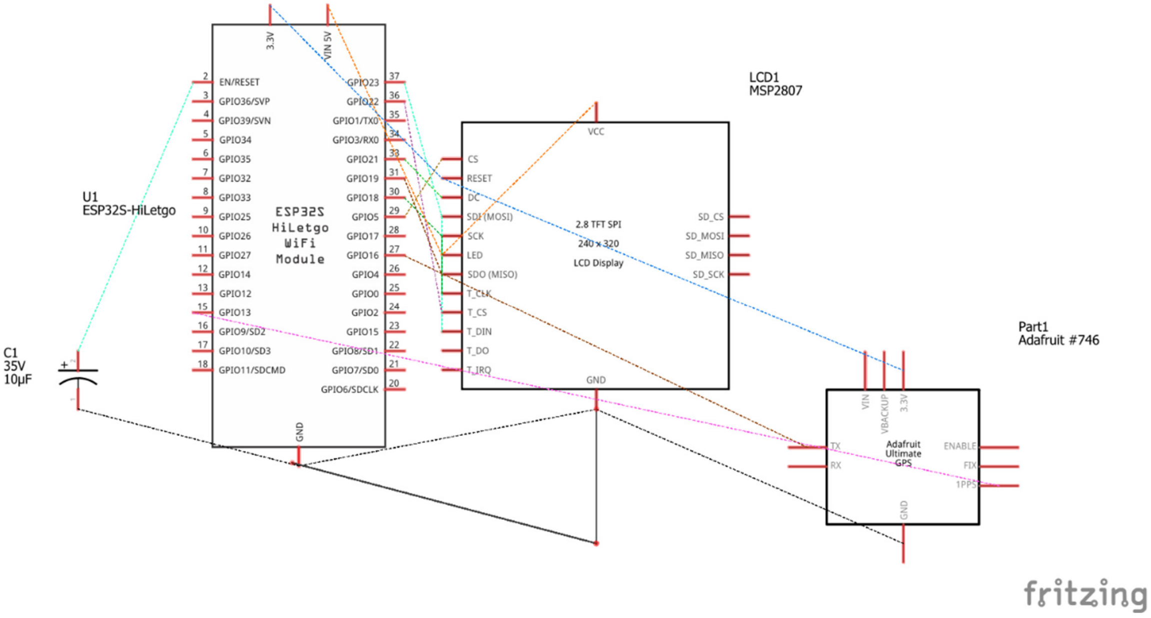
Wire the TFT Display and Touch Control as follows:

Display Pin	**Display Label**	**Connects to ESP32**	**Function**
1	Vcc	"5V"	Power
2	Gnd	"Gnd"	Ground
3	CS	GPIO 5	Chip Select
4	RST	"3.3v"	Display Reset
5	DC	GPIO 21	Data/Cmd line
6	MOSI	GPIO 23	SPI Data in
7	SCK	GPIO 18	SPI clock
8	LED	"5v"	Backlight power
9	MISO	GPIO 19	SPI Data out
10	T_CLK	GPIO 18	Touch SPI clock
11	T_CS	GPIO 22	Touch Select
12	T_DIN	GPIO 23	Touch Data in
13	T_DO	GPIO 19	Touch Data out
14	T_IRQ	no connection	Touch interrupt

Here is the breadboard layout for the ESP32, display with touchpad, and GPS which we will use later. Wiring details for the GPS are included below.



Here is the wiring schematic.

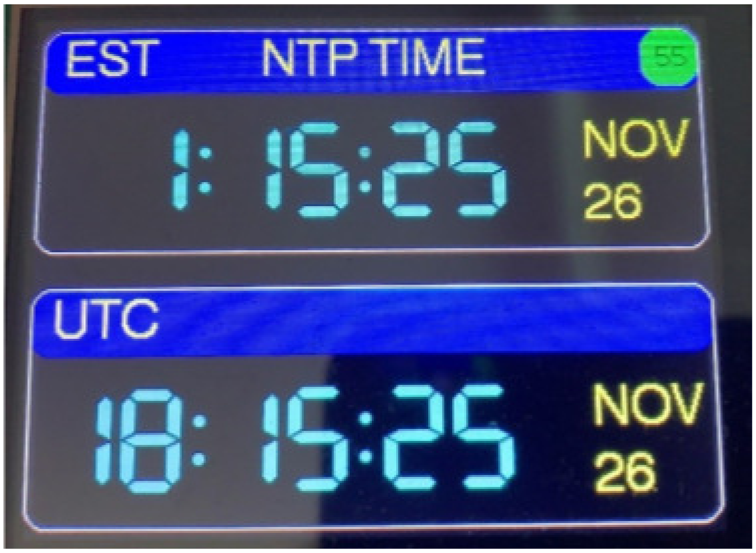


Follow this procedure:

a. Follow the steps in the NTP_DualClock instructions from http://w8bh.net/NTP_DualClock.pdf as detailed below.
    i. The project provides test code for each step so you can verify that a part of the project is working before going on to the next step.
    ii. Do step 1, configuring the ESP32, and step 2, connecting the display.
    iii. Do step 3, Install the Display Software. Note that you must download the "TFT_eSPI" library, then modify that library's User_Setup.h file as shown. You can copy and paste. I suggest that you save the User_Setup.h file that was distributed with the library to a different name in case you should ever want to start over. Make sure that you get this step to work before going on. You can find this modified User_Setup.h file at https://github.com/hwdornbush/Arduino-clocks/blob/main/User_Setup.h
    iv. If you live in another time zone, such as Central time, you can modify the sketch for this. Ask me if you need help with this.

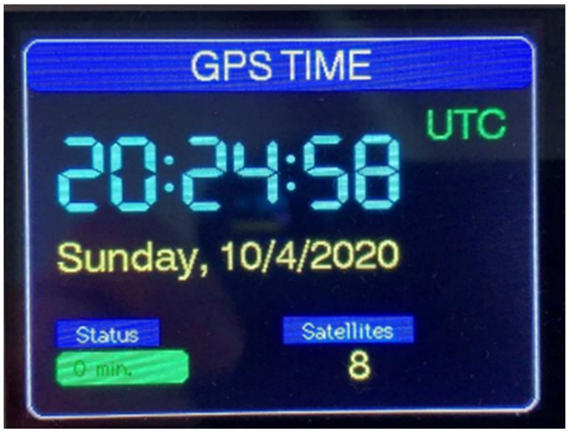
b. Well, I didn't want to go through each remaining step. You might if you want to learn how the clock is assembled. I did steps 1-4 then copied the entire sketch
`NTP_DualClock.ino`
to my computer, and uploaded it to the Arduino. It ran just fine and I had a nice clock that would be handy in my shack.

When started, the NTP Dual Clock will appear.



GPS CLOCK

You can view this clock and how to build it at http://w8bh.net/gps_clock.pdf



This clock has three display modes: a single clock that you can control to show local or UTC time, a dual clock showing local and UTC times, and a display showing GPS coordinates and more GPS information.

This clock also includes using the 2.8" display's touch screen to control the clock.

I started by adding a GPS clock breakout board to my breadboard. The GPS clock I used is ATGM336H GPS Receiver Module, SKU: 26793, although any GPS board that provides serial output and 1 pulse per second should work.

Wire the GPS breakout board as follows. The actual pin will depend on which breakout board you use but the names are consistent.

GPS Label	Connects to ESP32	Function
VCC/3.3V	"3.3v"	Power
GND	"GND"	Ground
TX	GPIO 16	GPS Serial out
RX	no connection	GPS Serial in
PPS	GPIO 13	1 Pulse/sec

TOUCH CONTROLLER

The touch controller should be built into the 2.8" TFT display. First, test that you have the touch controller working.

1. Use the instructions from http://w8bh.net/touch.pdf to learn how the touch controller works.
2. Test that the touch controller works by using touch_demo1.ino. Navigate to: https://github.com/bhall66/Touch-Control/blob/main/touch_demo1/touch_demo1.ino
3. Find ⧉ at the right. Click ⧉. This will copy this sketch to your clipboard.

4. In Arduino IDE, click File > New Sketch.
5. A new sketch window will open with a skeleton sketch in it. Delete the skeleton sketch by keying Ctrl-A then Delete. Key Ctrl-V to paste the touch demo sketch into the IDE window. Key File > Save As. Enter "touch_demo1" and press Save. This will save this sketch so you can use it again as needed.
6. Press → to compile and upload the sketch to your ESP32.
7. Test the touch panel by pressing anywhere on the screen, and you should see a yellow circle appear.

GPS TESTING

I wanted to be sure that my GPS breakout board was functioning so I used W8BH steps to test it. This is not essential but it ensures that you have wired the GPS correctly to the ESP32.

"Step 4: Testing the GPS 1PPS Signal" tests that the GPS can receive signals and that the 1 pulse per second signal works. This sketch introduces a new concept for us: use of the Interrupt. This makes use of the ability to interrupt the ESP32 processor to run a special routine, that return to what it was doing. In this case, when the 1PPS signal arrives, the interrupt routine increments a variable. Then the processor continues its work and when it next displays, it uses the new variable value, and you can see that a new signal has arrived as the number displayed increments.

The sketch as shown by W8BH is for the "Blue Pill" processor, so I had to change it for the ESP32. Google is your friend, and I used it to find how to alter the code for the ESP32.

When you upload the sketch and power on the clock, it will take a few seconds for the GPS to acquire a signal. Different GPS breakout boards indicate this differently. The ATGM336H GPS Receiver Module shows a solid red LED until the GPS acquires a signal then it begins to blink once a second. The Adafruit module has a different behavior.

Here is the "GPS_CLOCK_step4.ino" sketch as I modified it for the ESP32.

```
/**************************************************************************
HWD 2023-04-18  modified for ESP32 pins and interrupt handling
       Title:   GPS clock with TFT display  (STEP 4: COUNTING 1PPS SIGNAL)
      Author:   Bruce E. Hall, w8bh.net
        Date:   29 Sep 2020
    Hardware:   Blue Pill Microcontroller, 2.8" ILI9341 TFT display,
                Adafruit "Ultimate GPS" module v3
    Software:   Arduino IDE 1.8.13; STM32 from github.com/SMT32duino
                TFT_eSPI and TimeLib libraries (install from IDE)
       Legal:   Copyright (c) 2020  Bruce E. Hall.
                Open Source under the terms of the MIT License.
```

```
 Description:   Build a working GPS-based clock with TFT display
                In this sketch, the GPS 1pps signal is verified
                by using a hardware interrupt.

                See w8bh.net for a detailled, step-by-step tutorial

 **************************************************************************/

#include <TFT_eSPI.h>
#define GPS_PPS              13                // GPS 1PPS signal to hardware
interrupt pin

TFT_eSPI tft = TFT_eSPI();                       // display object
volatile byte pps  = 0;                          // GPS one-pulse-per-second
flag

void ppsHandler() {                              // 1pps interrupt handler:
  pps++;                                         // increment pulse count
}

void setup() {
  tft.init();
  tft.setRotation(1);                            // portrait screen orientation
  tft.fillScreen(TFT_BLACK);                     // start with blank display
  attachInterrupt(GPS_PPS, ppsHandler, RISING);   // enable 1pps GPS time sync
}

void loop() {
  tft.drawNumber(pps,50,50,7);                    // show pulse count on screen
  delay(100);                                     // wait 0.1s; no need to
hurry!
}
```

This sketch is also available at https://github.com/hwdornbush/Arduino-clocks/blob/main/GPS_CLOCK_step4.ino where you can copy and paste it to your Arduino IDE sketchbook.

Upload and test the GPS 1PPS using the same steps 2-6 as shown above for the touch controller, except using the “GPS_CLOCK_step4.ino” sketch.

When I uploaded it and ran it, once the GPS module acquired a signal, the display showed the incrementing pulse count.

“STEP 5: TESTING THE GPS SERIAL DATA” is next. It will verify that the GPS is transmitting data to your ESP32. Here is the sketch as modified for the ESP32.

This sketch is simpler than the Step 4 sketch in that no interrupt processing is needed. We just read the serial data arriving at the ESP32 from the GPS.

```
/**************************************************************************
hwd 2023-04-18 modified for ESP32
       Title:   GPS clock with TFT display  (STEP 5: GPS SERIAL DATA)
      Author:   Bruce E. Hall, w8bh.net
        Date:   29 Sep 2020
    Hardware:   Blue Pill Microcontroller, 2.8" ILI9341 TFT display,
                Adafruit "Ultimate GPS" module v3
    Software:   Arduino IDE 1.8.13; STM32 from github.com/SMT32duino
                TFT_eSPI and TimeLib libraries (install from IDE)
       Legal:   Copyright (c) 2020  Bruce E. Hall.
                Open Source under the terms of the MIT License.

 Description:   Step 5 of building a GPS-based clock with TFT display
                The sketch tests connectivity with GPS serial data
                by showing the incoming data stream on the display

                Make sure that the GPS Tx line is connected to PA11
                on the Blue Pill.

                See w8bh.net for a detailled, step-by-step tutorial

 **************************************************************************/


#include <TFT_eSPI.h>
TFT_eSPI tft = TFT_eSPI();                         // display object

void setup() {
  tft.init();
  tft.setRotation(1);                              // portrait screen orientation
  tft.fillScreen(TFT_BLACK);                       // start with blank display
  Serial2.begin(9600);                             // set baud rate of incoming
data
}

void loop() {
  if (Serial2.available()) {                       // if a character is ready to
read...
    char c = Serial2.read();                       // get it, and
    tft.print(c);                                  // show it on the display
  }
}
```

This sketch is available at https://github.com/hwdornbush/Arduino-clocks/blob/main/GPS_CLOCK_step5.ino

When I uploaded and ran this sketch, I saw the GPS data appear on the display.

I suggest that you read the rest of W8BH GPS Clock document to see how he processed the GPS data to make his clock display. In the code for this sketch, he shows how to derive your Grid Square from GPS location.

I modified his sketch "sketch_GPS_CLOCK_triple.ino" to use the ESP32, and to comment out the sections on sound and battery, as I chose to not include those in my device. I usually comment out code that I don't want to use rather than deleting it. You could uncomment sections you want to try, but additional hardware is needed.

This sketch is rather long so I don't include it here. It is available at https://github.com/hwdornbush/Arduino-clocks/blob/main/sketch_GPS_CLOCK_triple.ino

More libraries are needed for this sketch. You can find which ones by examining the sketch. At the top of the sketch, you will see the following statements:

```
#include <TFT_eSPI.h>                  // https://github.com/Bodmer/TFT_eSPI
#include <TimeLib.h>                   // https://github.com/PaulStoffregen/Time
#include <TinyGPS++.h>                 // https://github.com/mikalhart/TinyGPSPlus
#include <Timezone.h>                  // https://github.com/JChristensen/Timezone
```

The libraries are named as shown in the <…h> statement with a comment that links to the library if needed. We have already installed TFT_eSPI. We will need to install the next 3 libraries shown. See the details above under "INSTALL LIBRARIES" if you are not sure how to install them.

The "#include <….h>" statements direct the compiler to include the code in the files named. The .h files are "header" files that show the components of the library so that the compiler can set up the linking to the libraries as it processes the sketch.

When I uploaded it and ran it, the display showed the GPS time and location information, and the touch control could be used to alter the display as noted in the sketch.

We now have a functional GPS clock.

BOTH NTP AND GPS CLOCKS

When I examined the sketches for the NTP clock and the GPS clock, I saw that W8BH had used one as the basis for the other. Much of the display part of the code was the same or very similar. I decided to take up the challenge to merge these two sketches into one sketch which would allow you to choose which clock you wanted when you started the sketch.

Why? If you built this clock and were using it in the shack where a window was not convenient so that you could not get a GPS signal but you could use your home Wi-Fi, then you would want the NTP clock. If you were using it in the field where you could get a GPS signal but not your home Wi-Fi, then you would want a GPS clock.

When you start the sketch, you can use the touch screen to select which time source you wanted to use. If you need to switch to the other time source, just restart the sketch. To restart a sketch, remove power to your clock and then restore power and you will see the select screen.

When using the NTP time source, the touch screen is disabled. When using the GPS time source, the touch screen functions to allow you to choose one of the three available screens.

This sketch is available at https://github.com/hwdornbush/Arduino-clocks/blob/main/sketch_GPS_NTP_CLOCK_triple.ino

WWV CLOCK

You can view this clock and how to build it at http://w8bh.net/wwvb_clock.pdf

This clock is not as robust as the NTP and GPS clocks. The radio receiver on the WWV module is not that great. Bruce Hall says:

> "Do you have a radio-controlled clock at home? You know, the ones that set themselves? If you live in the eastern US like me, you might also know they don't always work. Mine won't update unless I take it off the wall and put it by a window overnight. Which defeats the purpose of a self-setting clock…
>
> So why would I want to build one? There are other choices for precision timekeeping, like GPS and NTP. For some locations, WWVB indoor reception is significantly better than GPS. No clear view of the sky is required. WWVB is well suited for low-cost, low-power, battery operation. And, unlike NTP, no internet access is required. Finally, for a dyed-in-the-wool ham operator like me, getting information by radio has a certain appeal that's difficult to describe."
>
> "To use the receiver, orient the antenna just that it is horizontal and broadside (perpendic99ular) to the direction from your location to the transmitter in Fort Collins CO. Next, make sure that there are no electronic devices nearby. For example, my receiver does not work when placed near my computer monitor. Use a battery or low-noise supply for your power. Finally, use the receiver later at night or early in the

> morning, avoiding the afternoon. If you live near Ft. Collins these steps may not be necessary, but they'll help avoid frustration and failure here in the Eastern US."

You can see that although there are rewards, there are also challenges.

The first challenge is that W8BH built it using a Blue Pill processor, and not the ESP32. I have modified his sketch for the ESP32, I think. The challenge here is that his sketch uses an interrupt timer, which is not a trivial change.

The second challenge is the limits on reception here in the Eastern US. You will need to determine for yourself if you want to tackle this challenge.

I recommend that you start with the NTP and GPS clocks, and move on to the WWV clock once you have them working and you are up to the challenge.

WHAT NOW?

If you want to build your own clock, use the Bill of Materials to acquire the needed parts. In our next workshop, bring your parts and a computer ready with Arduino IDE and we can work together to build your clock.

I will have a soldering iron available if you need to attach some headers to your breakout board.

I am working on some code enhancements:

Merge the NTP and GPS clock programs so that you can choose which clock that you want to use at the time you start up the Arduino. DONE!

Add a 10 or 3 minute countdown timer that you can use to help you remember to identify your station while on the air.

What other code enhancements can you suggest?

After you build a clock, what now? You can use a solderable breadboard to make a more permanent version, or you could design and build a clock on a printed circuit board. A design is available for the NTP clock using the hardware shown in the BOM. Designs are available for the other clocks but with different hardware which you could acquire and then build.

Who wants to design and build a printed circuit board?