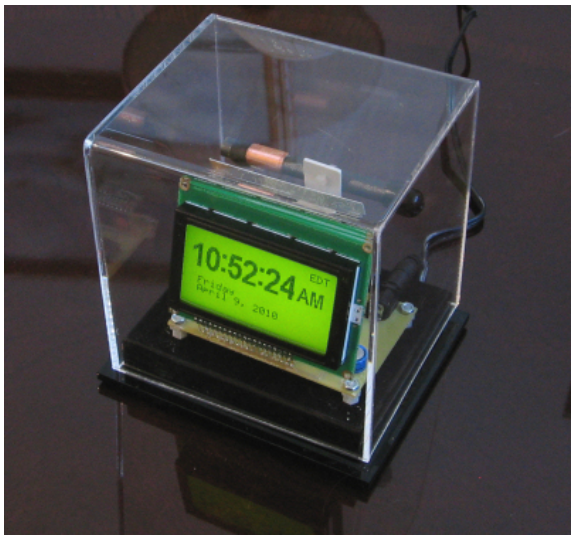
# Build Your Own Atomic Clock




Based on the PIC18 microcontroller, you can build this clock for around $50US.

**Features**

- Big LCD display with easy to read time/date.
- Supports 7 Different time zones.
- Slick looking clear acrylic case.

By Joe Jaworski

---

## WWVB Primer

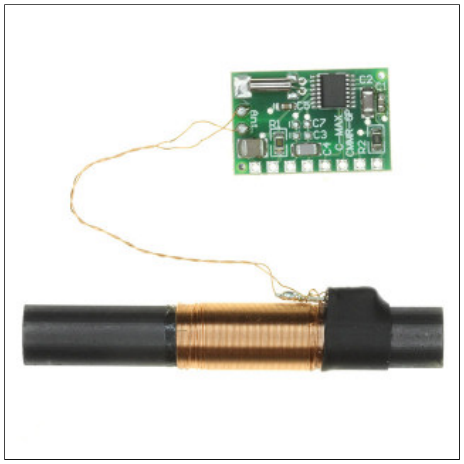
Since 1965, the US National Institute of Standards and Technology (NIST) has been broadcasting a digital time signal known as WWVB. The time is generated from a *Cesium Fountain*, which measures the vibrations of a cesium atom to determine the time. Amazingly, the accuracy of this time keeping device is +/- 0.1 nanoseconds per 24 hour period.

The signal is continuously broadcasted from multiple towers in Fort Collins, Colorado at a power level of 50,000 watts. This doesn't sound like much. After all, many local AM and FM radio stations broadcast at this same power level. The difference is that the WWVB frequency is quite low, being broadcast at 60Khz. This is barely above the range of ultrasonic. The signal is able to traverse land contours and easily penetrate buildings and structures because of its huge wavelength (approximately 5,000 meters).

Just like GPS, WWVB was developed for military and academic use. The WWVB signal is now used by millions of commercial and consumer time pieces and equipment. The system is highly reliable, employing two identical transmitters that each share the 50kW load. Should one go down or require maintenance, the second transmitter can be boosted in power and handle the entire load. Should both go down, there is a standby transmitter waiting to take over.

---

## The WWVB Signal



The WWVB signal is a serial digital data stream composed of a 60Khz carrier frequency being broadcast at two different power levels. A logic '1' is defined as the full power signal. The transmit power is then reduced by -17dB, indicating a logic '0'.

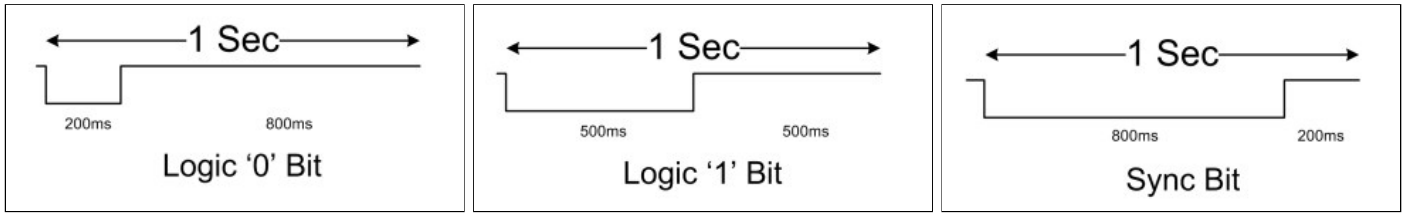
While is it not that difficult to build a demodulator, I opted to go with the CMAX CMMR-6 WWVB Receiver. This sells for about $11US and is available from www.digikey.com (P/N 561-1014-ND). It comes with a postage stamp-sized PCB and a 60mm antenna. The output from the PCB is a simple TTL logic signal that outputs a Logic 1 for the high and logic 0 for the lower power level. You can feed this line directly into an input pin on the microcontroller.

The speed of the transmission is at an incredibly slow 1 Baud. That is, each data bit takes one second to transmit. There are exactly 60 bits in the whole transmission, so it takes exactly one minute to transmit the time. Because of this, there is no seconds time transmitted, because at the end of the transmission, it is always zero seconds for the next starting minute.

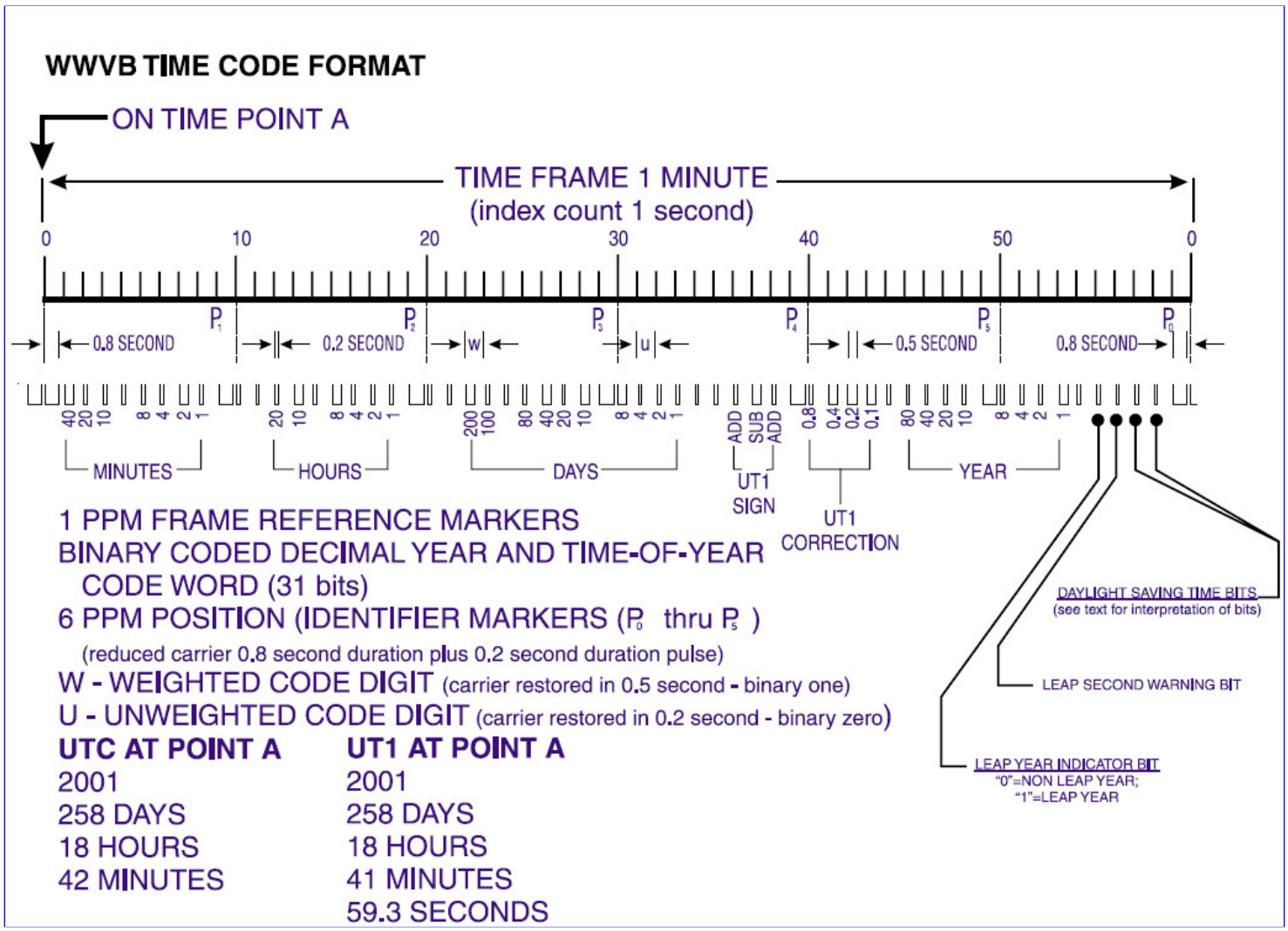
The signal is broken down into 6 segments that are each 10 seconds long. Each of these segments holds things like the hour, the minute, the day of year, etc. Each 10 second segment holds nine bits of data followed by a special sync bit. The sync bit indicates the end of the segment. So there are three different types of bits in the stream. They are a Logic '1', a Logic '0', and the sync bit.

A logic 1 is indicated by the signal going low for 0.5 seconds and high for 0.5 seconds. A logic 0 goes low for 0.2 seconds and high for 0.8 seconds. The Sync bit goes low for 0.8 seconds and then high for 0.2 seconds.

There is one exception to this rule and that is the very first segment contains a sync bit at both the beginning and the end of the segment. Therefore, the first segment only holds 8 bits of data. Why do this? Well, when you begin to decode the signal, you need a way to determine where the whole thing starts. Since the very first segment starts with a sync bit and all segments end with a sync bit, then two sync bits in a row means you are at the start of the transmission.




So let's look at all the bits in the signal as shown below. The first segment contains 8 bits of data that represents the current minute. The value can be from 0 to 59. All the data bits in the WWVB stream are broadcast in BCD notation, so the value that would actually be transmitted would be as a hexadecimal number from 0x00 to 0x59.




The 2nd segment contains the hours. The first two bits are set to logic zero and are not used. The remaining 7 bits define the hour of the day, which can be from 0x00 to 0x23.

The 3rd segment contains the day of the year. There is no month information in the transmission, so you have to figure out the month and date from the day of the year. Once again, the first two bits are not used and set to logic zeroes. The next six bits define the day of year, but there aren't enough bits to hold it all so the last 4 bits spills over into the 4th segment. The day of year value can be 001 to 366 (there are 366 days in a leap year).

The 4th segment contains the last digit (4-bits) of the day of year plus the UT1 Sign bits. You can ignore the UT1 sign and correction bits unless you are building a clock with an accuracy/display of tenths of a second.

The 5th segment contains the UT1 correction time, followed by a zero bit, followed by the upper digit of the year. Again, the only important info needed from here is the upper digit of the year. The possible values for the year are 0x00 to 0x99, with 0x00 being the year 2000.

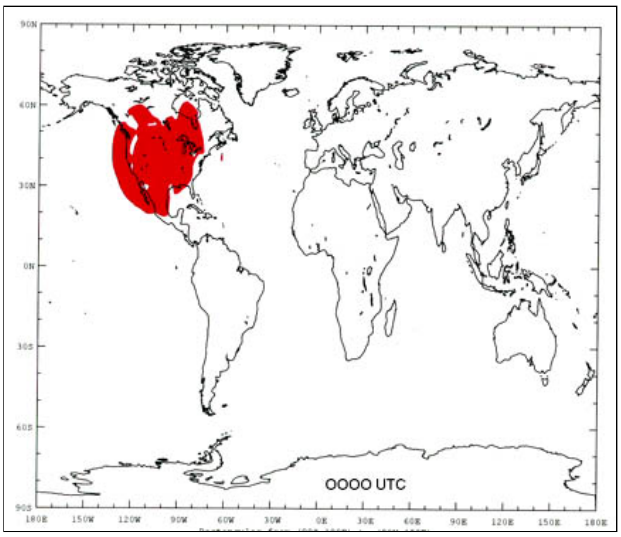
The 6th and final segment contains the lower digit of the year followed by several status bits. The first status bit is set to 1 if this is a leap year. The next bit is the leap second warning bit, which means a second will be added to the end of the current month. This bit can be ignored because the time will always be correct when read. The last two bits indicate the status of daylight savings time. If 00, it is standard time. If 02, DST begins tonight. If 03, DST is in effect right now. And if 01, DST ends tonight. To accurately display daylight savings time, all you need to do is check for a value of 3, and if so subtract one hour from the time.

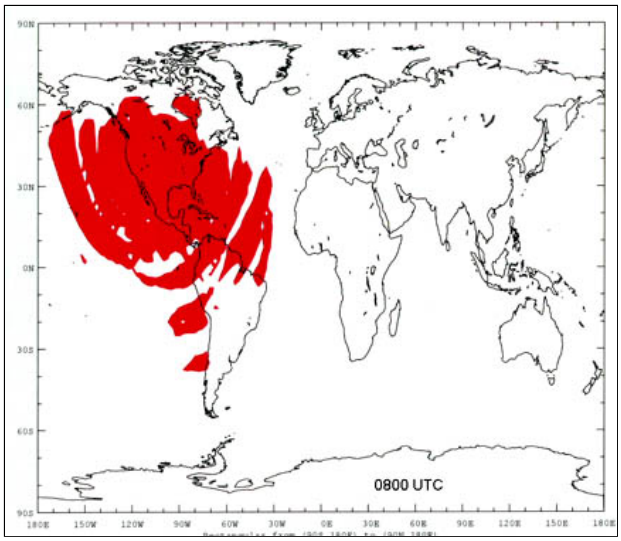
The time itself is broadcast in UTC, which is called *Coordinated Universal Time* or *Greenwich Mean Time*. To display the local time, you need to add or subtract a certain number of hours from UTC. For example, Eastern Standard Time is 5 hours behind UTC, so you need to subtract 5 hours to display the local time. If daylight savings time is in effect, you need to subtract another hour. This sounds simple enough, but there are some complications to deal with. For example, after subtracting the hours you might wind up going back a day, so you will need to subtract 1 from the date. That might push you back a month, so you'll need subtract 1 from the month. Not only do you need to know the number of days in a month, but you need to compensate for leap years. And if subtracting the month pushes you from January to December, you need to subtract a year.

---

## Receiving the Signal

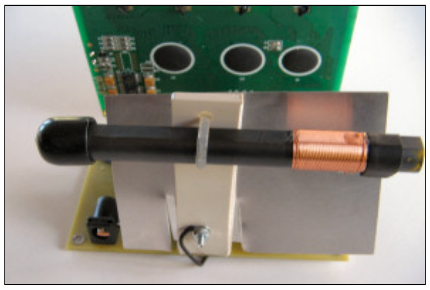
In a perfect world with no RF interference, you could continuously receive the WWVB data stream to keep your clock running. Unfortunately, the signal strength varies by time of day (solar and ionosphere influences) as well as interference from local appliances and RF signals and long distance radio waves. NIST has published some estimates on how far the signals travel based on time of day. At night, reception is strongest as solar radiation is not a factor, the ionosphere helps boost the signal, and most modern RF gadgets like cell phones are at their minimum use.



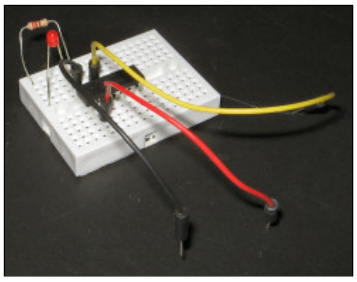

Because you can't always receive a good signal, you have to have a conventional clock to maintain the time between updates. Since we are shooting for the most accurate clock on the planet, it makes sense to use a fairly accurate clock (i.e., crystal controlled) to maintain the time.

When I started this project, I planned on updating the time from WWVB every few minutes to maintain perfect accuracy. Boy, was I wrong. There are so many sources of interference around the house/office that I was shocked to find out just how bad the signal can degrade.



I live in a rural section of North Carolina, well away from urban "RFI" sprawl. I figured the signal strength will be great. At about 9:30 AM every day, the signal strength goes way down. All I get is a bunch of noise and few WWVB bits coming through. It doesn't return cleanly until about 8:00 PM in the evening. Some days the entire signal is garbage until about midnight. Other days, I've gotten a perfect signal in the middle of the afternoon that lasted about 20 minutes, and then nothing for the rest of the day and night. One evening I was working on this project and discovered that my wife was running the microwave oven (located in the kitchen- a good 100 feet away). It added glitches to the WWVB signal which made it unusable.

The CMAX module comes with a 60mm antenna, but I figured I needed something better being 2,000+ miles away from Colorado. I purchased a 100mm antenna also from Digikey (cost of around $2US). It did improve the signal strength quite a bit, but it was no miracle fix. My suggestion is that if you live on the East coast or more than 1,000 miles away from Colorado, I would recommend spending the extra 2 bucks and upgrading the antenna. Just unsolder the wires on the old antenna going to the board and solder in the new one.

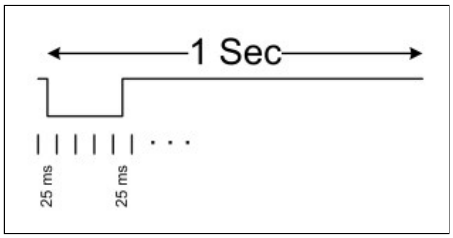


With all these signal problems, it was difficult to debug my code unless I was willing to work on it at 3 AM. Even so, electrical noise from my computer where I ran the compiler/debugger and the board sitting in front of it caused too much interference to get a good signal. What I finally did was write some PIC code and programmed an auxiliary chip that simulated the WWVB signal.

During this project I had the opportunity to tear apart a retail atomic wall clock. I found that the internal antenna was a bit smaller, maybe 40-50mm in length with a rather clumsy looking copper windings. Unless the CMAX module just has bad reception, I would guess that these retail atomic clocks that now sold everywhere go a long time (maybe days or weeks or even months) between receiving a clean WWVB signal. None of them I know of have any sort of display that tells you when the last time it was updated. This was definitely a feature I wanted to put in my design.

My final scheme was to check the WWVB signal every 4 hours. While the signal is almost always rejected during the daylight hours, on the average I get synced up with WWVB twice a day.

## Error Correction

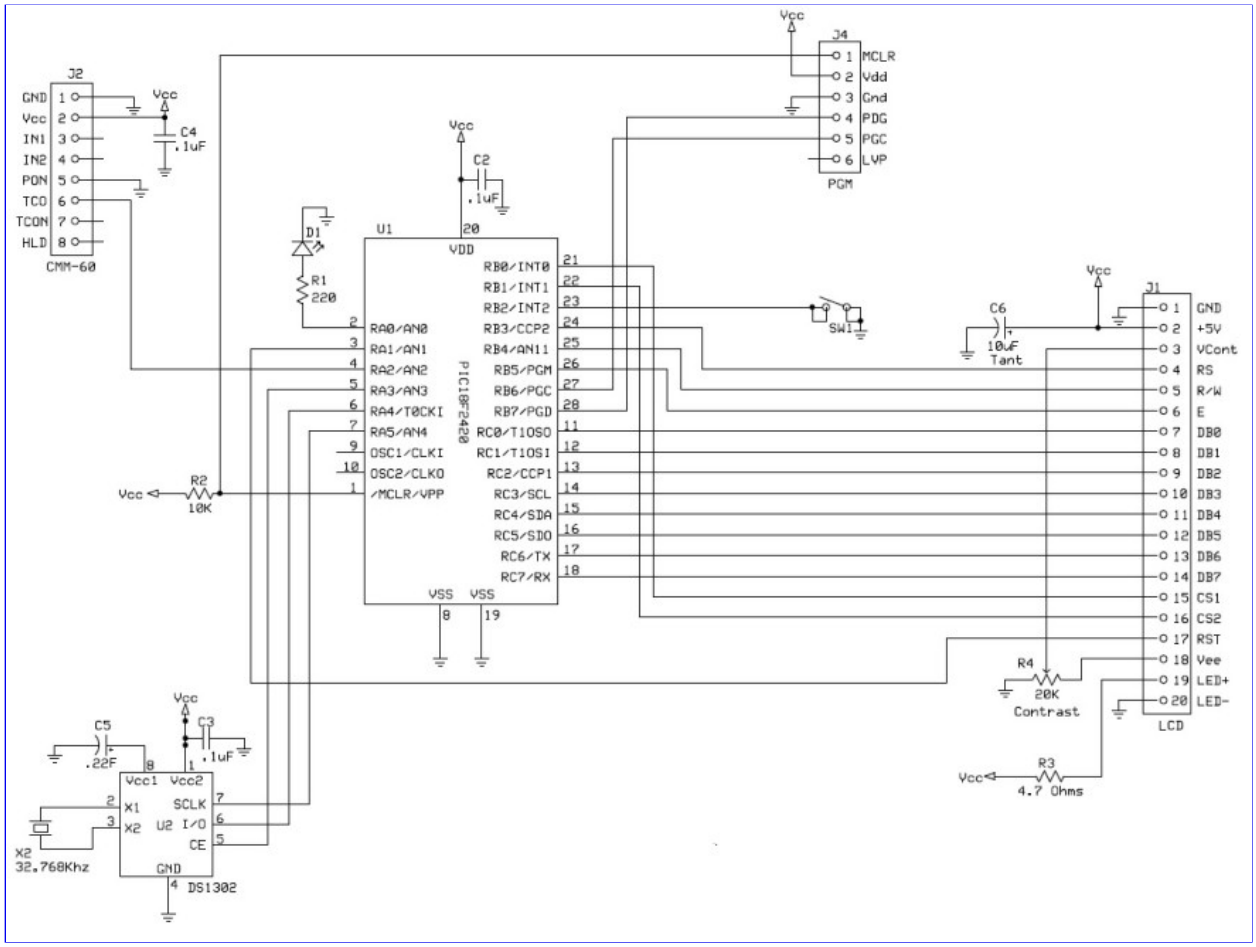



There is no error correction, CRC, or checksum in the WWVB data stream. So how do you know the time is right, i.e., doesn't contain garbage bits or noise? My approach is two-fold. First, I know that the signal must make a high going transition at either 200ms (logic '0'), 500ms (logic '1'), or 800ms (sync bit) after the start of the data bit. My firmware divides the 1 second bit time into forty 25ms periods. I check to see if a transition occurred within the three 25ms time frames that constitute a valid bit. If it doesn't, I throw out the whole transmission and start over.

The second check I do is on the data values themselves. Obviously, the minutes data must be a BCD value between 0 and 59, the months must be from 1 to 12, the year must be from 00 to 99, etc. If any of these values is out of range, the whole transmission is rejected. There also a few bits in the stream that are always set to zero. By checking for valid data ranges and these unused bits, the error detection is quite robust. In the hundreds if not thousands of times my clock has received the WWVB signal, I have never had the clock display the wrong time.

## The Schematic

The circuit is based around the PIC18F2420. If you're still using PIC16's for all of your projects, it's time to move on. The PIC18 is a great 8-bit microcontroller. It's got everything the PIC16 has plus more of it. There are 4 timers, dual PWM/analog compares, more oscillator options, and the list goes on. Most importantly, you have register mappings to both the I/O port bits and port latches (which eliminates the read-modify-write problems inherent in all PIC16's). You get all of this for about a buck more in price. Unless your designing a high volume low cost product, seriously consider one of the many PIC18's for your next project.



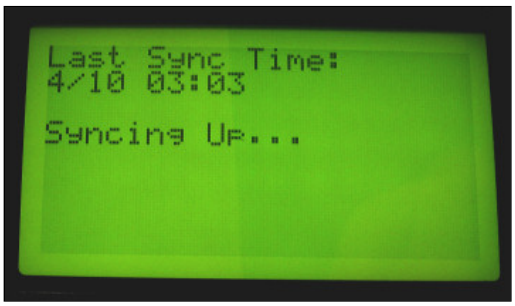

The schematic is deceptively simple. There are only 2 IC's on the board. U2 is the very common DS1302 clock chip. Rather than use a lithium coin battery, I opted for a .22F DLC or Super Cap to maintain the time during power failures. My estimates are that the clock can run off the cap for around 4 weeks without power. This is more than adequate, and I never have to change batteries.

Aside from the DS1302, the only other chip is U1, the PIC 18F2420. There is an 8 pin IDC female socket where the CMAX WWVB receiver plugs in. J4 is a debug connector that ties to the PICKit-3 debugger, which I used for in-circuit debugging. I opted to use a 5V switching wall adapter so no voltage regulator is required. You could certainly use an unregulated wall transformer and add a 5V regulator.

I got a PC board made by ExpressPCB using their low cost miniboard service. The circuit is so simple though, that you could easily wire it up by hand using a piece of vector board and some sockets and push pins.

There is a pushbutton switch and an LED on board. Normally, the clock is updated every 4 hours. By pressing the switch, you can manually do an update at any time. The LED flashes in sync with the WWVB transmission. At 1 Baud, It is slow enough to recognize one and zeroes and the sync bit by the flashes. During an update, the clock shows the last date/time of a successful update, followed by a string of digits while it progresses through the data stream. An 'S' is displayed when the Start of the stream is detected followed by 12345 as each segment is read and decoded.

The push button is also used to set the time zone. If you push and hold the button down for more than 5 seconds, the screen switches to the Time Zone menu. Push and release the button to cycle through the time zones, then push and hold for another 5 seconds to lock in the new time zone setting. The setting is saved in the DS1302's battery back up RAM.


J1 is a 20 pin connector that goes to a 128x64 graphics LCD. Getting the graphics LCD to work was probably more effort that the WWVB portion. If you're familiar with character LCDs, you can pretty much buy any kind or brand and they all have the same pin outs. With graphic LCDs this is not the case. There are at least 3 different types of LCD controllers that go on these displays. The most popular seems to be the KS0108, and this is the one I chose.

The biggest pain in working with graphics LCD displays is that there are actually two controllers on board, each controller works on a 64x64 pixel area. The interface connector has two Chip Select lines to determine which side of the 128x64 pixel array you want to read or write. The difficulty comes from drawing a bitmap that happens to span both arrays. I've written the code so that the routines automatically switch Chip Selects and change the appropriate page/column addresses to make this run smoothly. Another problem is that KS0108-based LCD displays do not contain any internal fonts. So everything that is written to the display needs to be a bitmap. I wanted big digits for the clock display plus a smaller full ASCII character font for the text portions of the display. All of these bitmaps are stored in flash with some simple-to-use functions to write characters to any x,y coordinate on the display.

The LCD display is mounted to the board using a standard right angle 20 pin IDC connector. After soldering, I bent the connector slightly so that the display would be at an angle instead of being vertical to the board.

---

## The Firmware

The entire code is written in C using the Hi-Tech C PIC18 compiler. It can be easily ported to other PIC18 compilers since the I/O ports are all macro'd in.

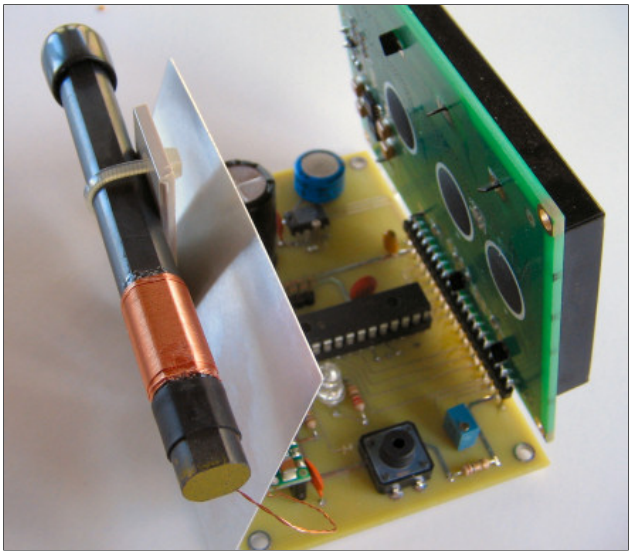
The main section of the code (in wwv.c) is dedicated to decoding the WWVB signal. I do this using two of the PIC's timers. The first is a 750 millisecond timer that uses interrupts to count up to a 2.5 minute period. When you first try to decode a WWVB signal, it can take up to 60 seconds to find the double sync bit which indicates the start of the data stream. Then it takes another 60 seconds to read in the entire stream and decode it. The 2.5 minute timer is sort of a watch dog. If it takes more time than this, then the code gives up and tries again 4 hours later.

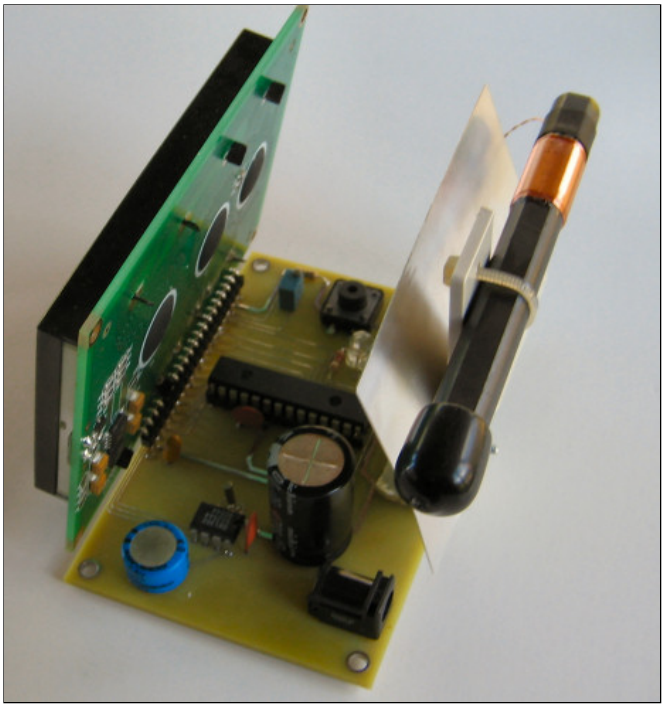
A second timer runs in polled mode (no interrupts) that creates and manages a 25 millisecond timer. What I do is break down the 1 second bit time into 40 intervals of 25 ms each. I know that a '1' bit, a '0' bit, and the Sync Bit must make a transition at certain times. By framing these bits within 25 ms windows, I can immediately decode (or reject ) a bit that doesn't match their transition spots in time.

---

## Noise Shield

After all the firmware was done, I swapped my WWVB simulator PIC for the CMAX demodulator to receive some real data streams. What I discovered is that the board was putting out so much noise that it was distorting the signal. My original design used a 10Mhz resonator to drive the PIC18. This caused way too much radiated noise. So I yanked the 10 Mhz resonator and switched to the internal 4Mhz RC oscillator. This made a huge difference, but the antenna was still picking up noise from the PCB. After experimenting a bit, I added a small aluminum shield between the antenna and the board and this fixed the problem. I had some thin aluminum sheet laying around, thin enough to cut with scissors, which I cut to size and attached a screw and nut with a grounding wire to it. You can probably use some doubled up aluminum foil, or the thicker foil that comes on the top of coffee cans. In any case, the grounded shield really cleaned up the signal strength. Just make sure the shield is grounded.

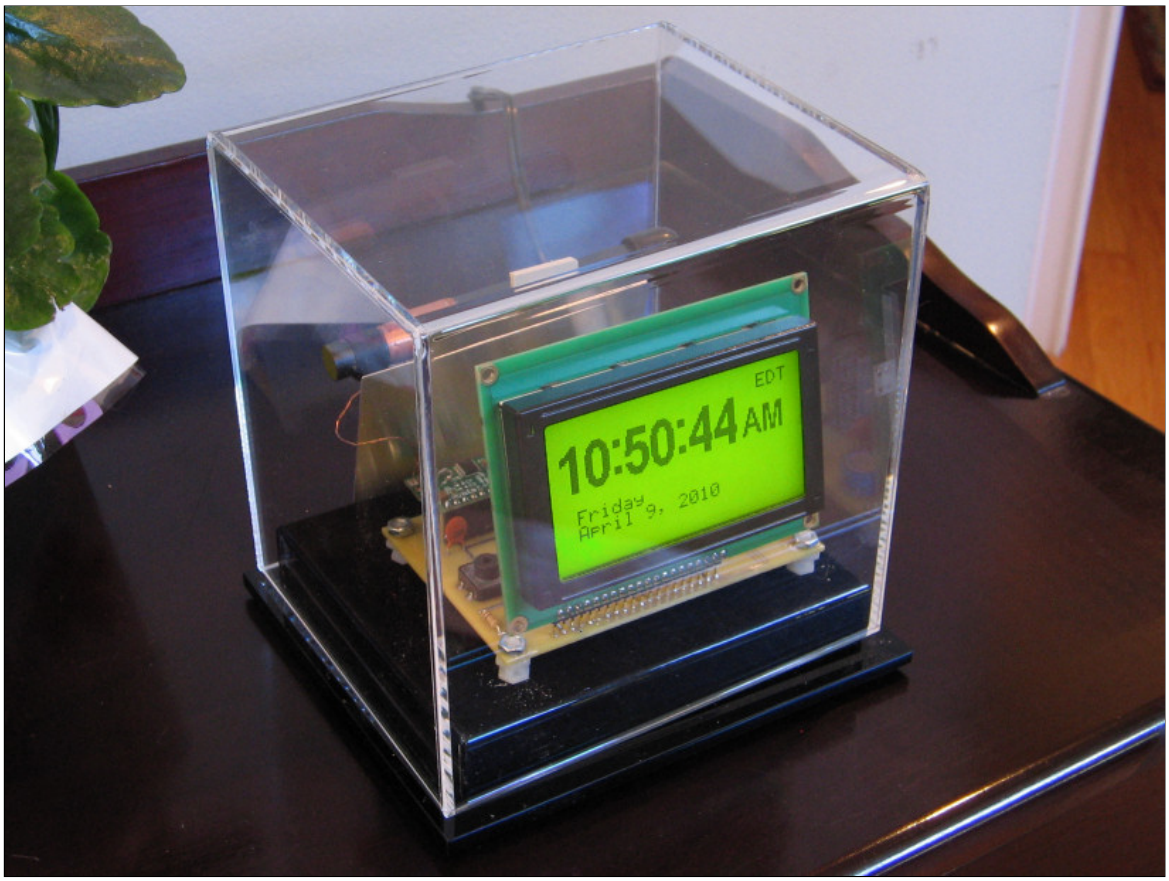




---

### Final Packaging

I wanted to display the final project on my desk and wanted it to look good. After searching around for an enclosure, I discovered a clear acrylic box that is made for displaying a baseball. These trophy display cases are available from lots of places on the internet. The one I bought was around $15US. The base of it is weighted, so it makes for as very solid enclosure. To mount the PCB to the base, I drilled some holes in the bottom and used some nylon PC motherboard standoffs and nuts to fasten it down. I think the look of it is kinda cool. At least everyone who sees it on my desk always asks about it.




---

### Files

*Last Update: 05-MAR-2012*

hex.zip - The Hex file ready to burn into a PIC18F2420.

sources.zip - All the source code.

schemo.pdf - The schematic as a pdf.

1383.pdf - The official US government Spec on WWVB.

big.jpg - A really big picture of the clock.

I'd love to hear your comments on my design, especially if you're building one, so send me some Email.