President's Corner

By Steven Bible, N7HPR, President, TAPR

At the 2012 ARRL-TAPR Digital Communications Conference (DCC) in Atlanta, Chris Testa, KD2BMH, received a lot of attention with his presentation of his handheld Software Defined Radio (SDR) transceiver. His low powered 144 and 440-MHz radio operates in a variety of digital modes, as well as AM, FM, and SSB.

Chris will be at Dayton for Hamvention in May and I have invited him to speak at the TAPR Forum.

By the way, Hamvention is May 17-19 and it is not too early to make your reservations. (Actually, it is never too early to make your reservations for Dayton!)

TAPR will be at Hamvention in full-force with our suite of booths in the Hara Arena, our annual Forum Friday morning, and our joint dinner with AMSAT Friday evening.

Getting back to KD2BMH and his SDR HT, check out what he has in store for Hamvention at https://docs.google.com/document/d/1WHknlNMDlTOderYMbFxFyw2kLa4m0EriGmIXNd_qaPw/edit# and I think you will be impressed.

Behind the scenes, TAPR veep Jeremy McDermond, NH6Z, has unveiled a new membership management application that will handle TAPR membership applications and renewals in a timely manner and increase our membership numbers in the process.

Jeremy has also worked closely with TAPR Directors John Ackerman, N8UR, and John Koster, W9DDD, to move TAPR's online presence to a new server . During the move, the server movers also updated its contents. Hopefully, the move has been transparent to the users, but if you notice anything out of kilter, please contact one of the gentlemen mentioned above and they will address the issue in a timely manner.

Also behind the scenes, the DCC moves west this year and we are currently considering a variety of sites on the left-hand side of the Mississippi for our big September get-together with the ARRL.

73,

Steve Bible, N7HPR, President TAPR

###

ARRL-TAPR 2012 DCC

By Tom McDermott, N5EG

(Originally published in *The Repeater*, the Rogue Valley Amateur Radio Club's newsletter)

The 2012 ARRL-TAPR Digital Communications Conference was held in Atlanta, GA, the weekend of September 21-23. I had missed the last several DCCs, and was glad to be able to attend this one. The last one attended was the one held near Portland in 2009.

Friday had a full day of technical and operations talks—all of which were excellent. There were some updates on AMSAT satellite programs. Hams apparently have taught satellite operators a few things:

Satellites can replace ballast on test launches. That plus launch insurance have eliminated pretty much all heavy-payload ham launch opportunities.

Microsats can be quite useful, as a result ham launch opportunities for those are now few, far between, and expensive.

Steve Hicks, N5AC, VP of R&D for Flex Radio gave a presentation on the Flex 6000 SDR radio, and showed how advances in FPGA computing capability essentially allow placing a supercomputer in each radio. (Steve worked for me briefly many years ago at Collins Radio.)

Heikki Hannikainen, OH7LZB, presented the story of the APRS.FI servers located in Finland, and the high-performance, high-availability European APRS central routers. They used surplus (but pretty recent) high-performance servers in a data center that hosts special projects. Heikki showed how they rewrote parts of the server code to increase the performance to thousands of web queries per second. An extremely impressive paper and accomplishment. Their system also tracks airplanes in flight and he demonstrated a live Google Earth map view of commercial flights in southern Finland.

Andrew Pavlin, KA2DDO, discussed a new open-source APRS client that he has written. It's very modern and utilizes OpenStreetMaps. Bob Bruninga, WB4APR, (the father of APRS) was in the audience and nodded approvingly at all the useful features that Andrew has incorporated. It has a tactical message data base, GPS interface, NWS drawing, and several protocols. It's available over the internet at: http://www.findtheater.com/ka2ddo/YAAC.html It runs on Windows, MAC, and Linux.

There were a number of toys available to play with in the demo room after the talks Friday, including 440-MHz high-performance data links, several D-STAR demos, several SDR radios, and a couple others that I've forgotten.

Saturday had two tracks, one technical, and one introductory. The introductory tracks had some interesting sounding sessions on D-STAR, but being unable to be in two places at once I attended the technical tracks.

David Bern, W2LNX, discussed a system he assembled from off-the-shelf products: an Ethernet router board that is essentially an open-source Ethernet switch/router that is DD-WRT compatible with two card cages for radios and no built-in RF. David used plug-in radios from Doodle Labs and XAGYL that can be run in the 440-MHz ham band. He achieved about 6 Megabits/sec back-to-back, and about 2 to 4 Mb/s using Yagis over various clear paths of 5 to 13 miles. He demonstrated full-motion digital video using the radios.

Chris Testa, KD2BMH, showed a presentation of a self-contained handheld transceiver (same size as a large handie-talkie) that was completely SDR. He concentrated on low-power consumption so that it would have reasonable battery life. It operates on 144 and 440 MHz, can be programmed for AM, SSB, FM, and various digital modes. Pretty slick.

Sunday morning Tom Rondeau, KB3UKZ, presented a four-hour seminar on GNU Radio. Tom has taken over development and maintenance of the GNU Radio project from Eric Blossom. http://gnuradio.org/redmine/projects/gnuradio/wiki

GNU Radio is an open source (free) software program that runs on Linux for prototyping and building real-time SDRs. In the past, these applications have been custom-written due to the very fast processing required. GNU Radio is a framework for doing the signal processing at speed on the PC (hundreds of thousands of samples per second or more) using drag-and-drop graphics. Many of the common processing blocks have already been written (such as filters, modulators, demodulators, Fourier transforms, phase locked loops, etc.). It also includes a data viewer that can show time, frequency, waterfall, and constellations.

Interfaces to the USRP radios designed by Matt Ettis come bundled with GNU Radio, as do interfaces to the FunCube and Ezcap dongles. The FunCube is a 60 to 1700-MHz receiver on a USB stick, and the Ezcap is a European format DVB-T television receiver. The Ezcap can be programmed to bring out straight I– and Q– samples at 3.25 Ms/s; the radio tunes from 48 to 863 MHz, but some can tune much farther. It's available

for between $20 and $40 from numerous Internet vendors (including Amazon.com). Tom demonstrated hooking up the Ezcap dongle to his computer and on-the-fly he constructed a wideband FM receiver. We were able to listen to some of the FM stations in Atlanta from his laptop speakers using GNU Radio. Any of the DVB-T dongles that use the RTL2832U chipset are supposed to be compatible.

For the seminar, each attendee got a bootable and runnable DVD with Ubuntu Linux and GNU Radio. It does not touch the hard drive of the computer. This allowed those attendees with Windows-only laptops to run the software and follow along with the seminar examples.

After returning home, I found WUBI—a program that allows installing Ubuntu Linux on a Windows NTFS file system. This allows dual-booting my Windows computer without having to partition the hard drive. WUBI then lead to installation of Ubuntu with the Gnome interface. Under Ubuntu I then installed GNU Radio 3.6.2 and the associated tools (GNU compilers, etc.). GNU Radio installs in the traditional Linux fashion from source, so the computer had to fetch the required support utilities and then compile and build the whole package. It took about eight hours, but went flawlessly. This allowed duplicating all the seminar exercises on the computer at home.

Since there is an actual application for all of this, it was desirable to learn how to write signal processing modules for GNU Radio. The fast-path code is written in C++ and uses SWIG to interface to Python. I took a quick on-line course on Python, but so far, it turns out it has not been needed. Fortunately I already knew C++. Some exploration turned up scripts and make files for creating a new GNU Radio module, and in a few hours I was able to generate a simple module, XML, compile and link, and load it into the GNU Radio environment, where it works as intended and interfaces with controls and sliders.

Next up is the desire to build software to interface to a ham SDR transceiver that Open HPSDR and TAPR kitted and manufactured this summer called Hermes (just arrived yesterday). But that's the subject of another newsletter article.

TAPR would like to host DCC in the western USA next year.

###

Paying TAPR

By John Koster, W9DDD

The TAPR website order checkout process will be changing soon.

In order to maintain PCI (http://en.wikipedia.org/wiki/Payment_Card_Industry_Data_Security_Standard) compliance with the minimum of continuing effort on the part of the staff, we will be migrating to a new payment gateway.

The big difference is that no one on staff will ever see the credit card information. It will be sent directly to the payment gateway that is responsible for all the PCI compliance.

The first step in the process will be for the gateway to 'authorize' the payment. In credit card industry parlance, this means the gateway will check that the card information is correct and funds are available. The funds will be reserved for future collection.

Once the office has verified the order, checked the shipping costs and determined that stock is ready to ship, the staff will initialize what is called "capture." That step actually starts the move of funds to the TAPR bank account where it arrives in two to three business days.

A customer should not see much difference from the previous process other than a different look to the webpages in the checkout process.

In the past, the office received the order, verified it, checked the shipping costs and readied it for shipping. Then the staff manually charged the credit card as an 'authorize and capture' transaction. In both scenarios the shipment takes place either the same day or the day following collection of funds.

###

Mammoth Cave Packet Test (March 2-3, 2013)

By Bob Bruninga, WB4APR

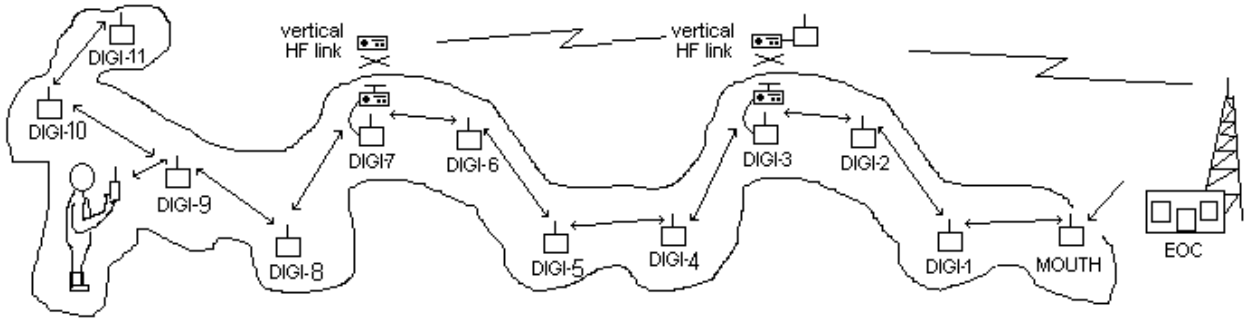


Fig. 1: A topside backbone can be linked using HF vertical links through a few hundred feet of rock spaced every mile.

Everyone knows that VHF radios do not "work" in caves because they don't work much beyond line of sight. But LF and MF work through reasonable distances of rock and have taken most of the focus of ham radio communications in caves. But this was all before APRS. We think that APRS can penetrate nearly a mile into a cave by using multiple APRS digipeaters up to 7 or even 14 hops as shown in Figure 1. The deep distance penetration can be multiplied if vertical links are possible every 14 hops. These links can be either VHF for shallow caves or HF for deeper penetration. KI4RDT conducted a test from the Mammoth Cave Hotel parking lot to the Discovery Tour passage and thinks that penetrations of up to 160 feet might be possible with 50W radios and beams.

What makes this test easy to conduct is the availability the last few years of the Kenwood TH-D72 HT which can also digipeat. This makes placement of these 14 digipeaters as easy as walking along the cave and placing an HT on the rock whenever we get to the end of a link. We have come up with some enclosures made from PVC pipe shown in figure 2.

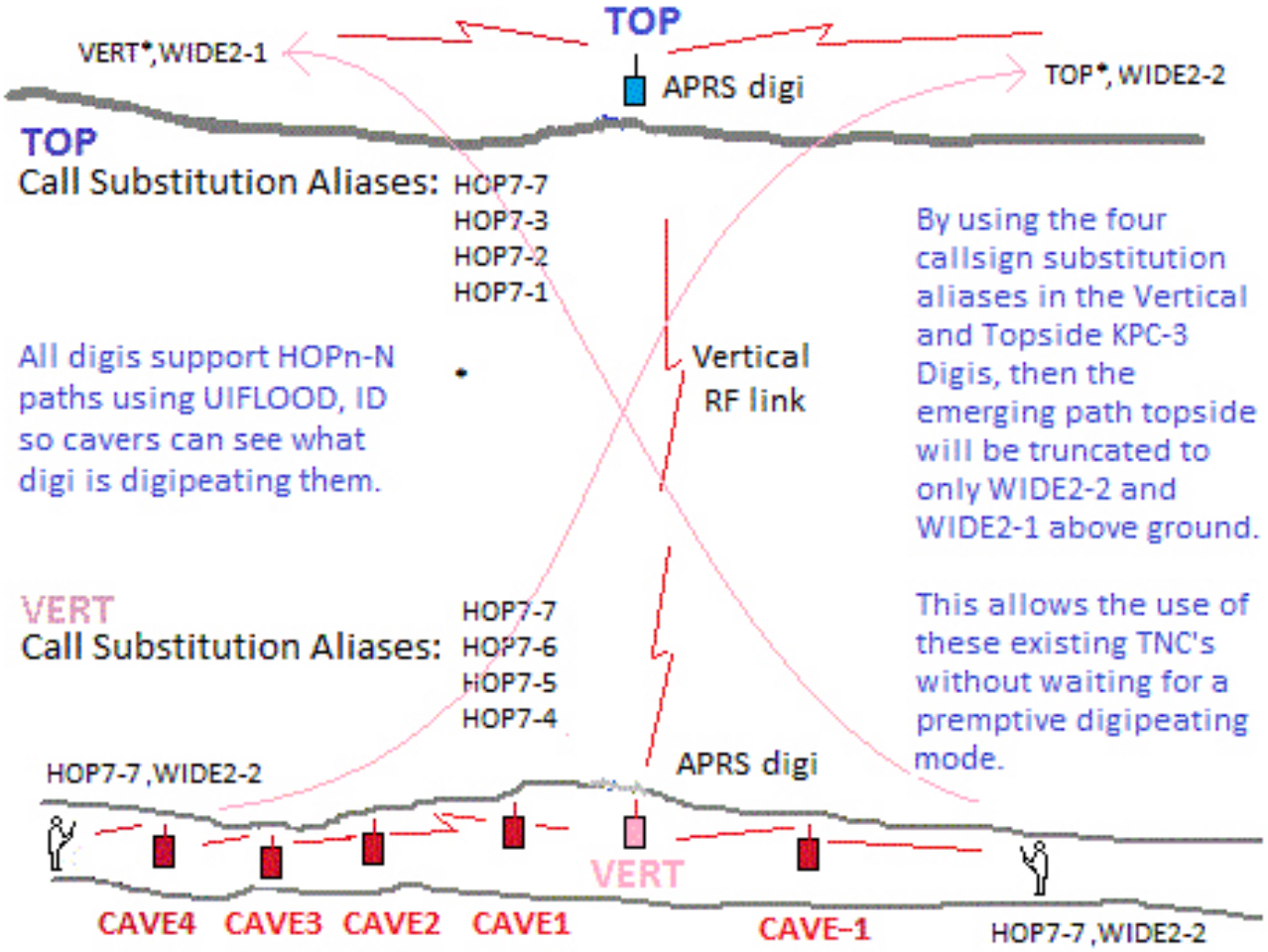
We have also come up with a clever packet PATH arrangement that allows us to easily gateway the multiple HOP7-7 packet path below ground into the topside global APRS network WIDE2-2 paths as shown in figure 3. By using the UIDIGI aliases in the KPC-3 digipeater to TRAP some of HOP7-N paths, we can truncate those from the above ground link. This way, they propagate normally topside into the APRS-Internet system as long as there is an IGATE within 1 or 2 hops on the surface.

In addition to the VHF tests we hope to also test some HF vertical links using FT-817 radios connected to KPC-3 TNC's as shown in Figure 4. The lower the band the better rock penetration, but also the higher the atmospheric noise topside. Also, if we have to use HF, then each cave-end of that link needs a VHF packet radio as well to link into the HOP7-N network.

Fig. 2: Three of the cave digipeaters (with or without tops). The HTs just drop in and out for easy adjustment.

If you are a ham radio caver, or you have a TH-D72 walkie-talkie to bring, or you have an FT-817 and KPC3 TNC, or even a D700 in a lunch box, then please consider volunteering for this big Mammoth Cave Ham Radio Test. We have scheduled 2-3 March 2013 it to coincide with the nearby Cave City, KY annual Hamfest which also turns out to be "Volunteer Weekend" at the cave so that we will have plenty of official Cave support. For more info see the web page: http://aprs.org/cave-link.html

###



GNU Radio

By Tom McDermott, N5EG

(Originally published in *The Repeater*, the Rogue Valley Amateur Radio Club's newsletter)

[The following article is a follow-up to Tom's report about the ARRL / TAPR DCC (see page 2

In the November, RVARC newsletter, I wrote about the ARRL / TAPR DCC conference and the new Hermes 0.5 watt digital software defined radio (SDR). At that conference was a four-hour introductory seminar on GNU Radio, a free open source software program (written for Linux) that allows generating real-time digital signal processing on a PC without needing to write any code. It has pre-defined processing blocks that you can drag and drop on the screen, and wire together with a few mouse clicks.

GNU Radio provides interfaces to a couple pieces of real hardware, the Ettus Research USRP, and the DVB-T dongle (see January QST for an article about the dongle). I decided to write C++ software to interface the Hermes radio to GNU Radio. Going from never having seen or installed Linux, and no behind-the-scenes familiarity with GNU Radio, it took about 9 weeks to dig for documentation (thank you Google), learn Linux, how to modify cmake files for Linux, and then write and debug the C++ code (which is threaded, so there's a few tricky bits) and a bit of XML.

Because all the I/O is done with the Hermes board and Ethernet, the computer does not need a sound card. The PowerSDR software uses the Hermes on-board audio amp to provide audio with no analog done anywhere in the PC. My GNU Radio implementation does use the PC audio card just to drive the speakers as it's nice to listen to what is being received. I have not yet gotten around to writing the software that uses the Mic preamp and audio amp that are on the Hermes radio board itself.

The net result is that the Hermes radio, which communicates with the PC using a 100M or a 1 Gigabit Ethernet connection can now be wired up to other software processing blocks just like any other GNU Radio block.

The first thing to do was build a SSB demonstration transceiver. It is based on the phasing approach. In analog circuitry, it can be difficult to make and adjust phasing networks for good amplitude and phase balance, but it's a piece of cake in software. The filters that produce the 90-degree phase shifts are drop-in blocks called a Hilbert transform, which acts just like a filter, but with some unusual symmetry in the taps.

The demo provides two independent receivers, adjustable filter widths, independent transmit frequency control and is full-duplex (now that's unusual in an HF radio!). The basic receiver demo took a couple hours to prototype in GNU Radio and have running. The hardest part of the receiver is the AGC, since in the phasing receiver, it is all done at audio. The default blocks in GNU Radio were used, but the AGC performance leaves a bit to be desired. Chatting with some of the authors of the commonly available SDR software, they spent a lot of work to get good AGC.

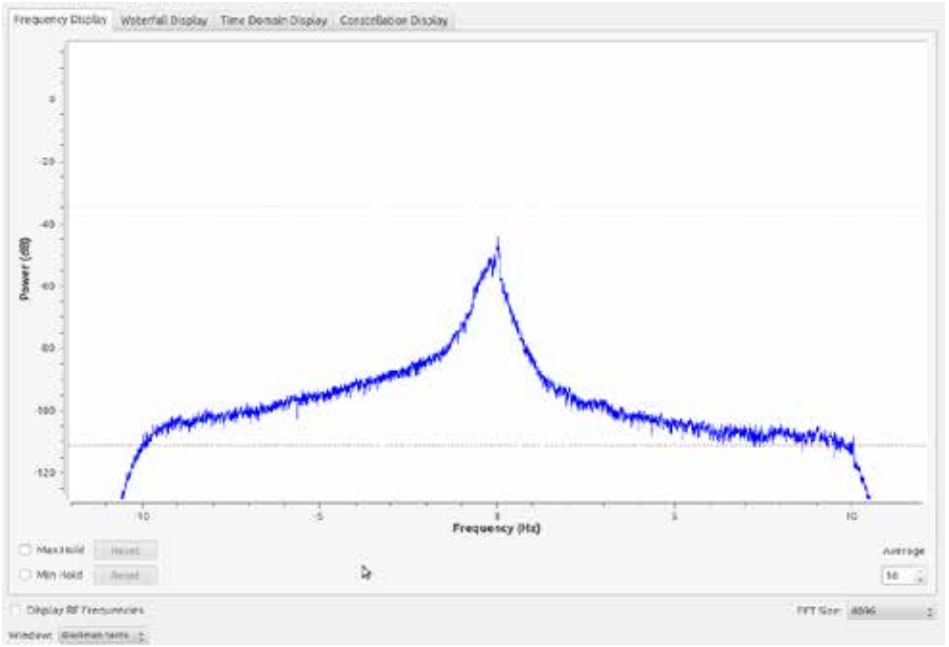


Figure 1: 5 MHz WWV baseband spectrum—span width is 20 Hertz

One of the things GNU Radio can do is to display signals in various ways—spectrum, waterfall, time domain, and constellation domain. Building a simple spectrum scope is almost no effort, just two or three blocks. The first time this was done, I set it to look at the AM broadcast band. The receiver has a maximum sampling rate of 192 KHz, so the widest spectrum that can be seen at any instant in time is 192 KHz (because the radio provides I and Q samples, the spectral width is the same as the sampling rate). It was immediately apparent that some of the AM stations in the Rogue Valley are transmitting digital HD signals on their AM carriers, the digital modulation is very readily apparent.

Another simple configuration was to build a setup for measuring the carrier frequency of WWV. The

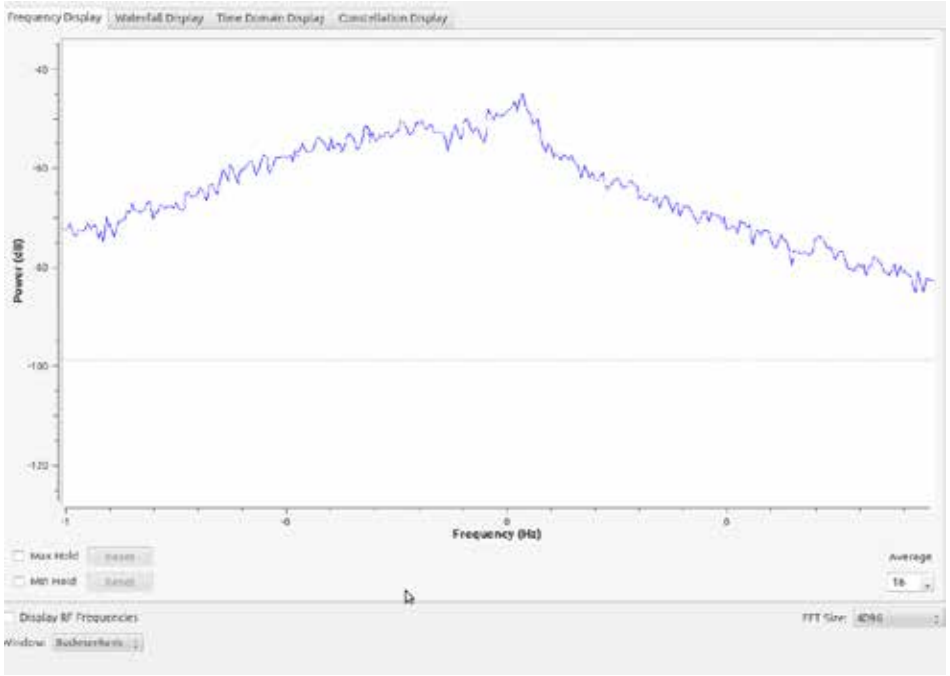


Figure 2: 5 MHz WWV baseband spectrum—span width is 2 Hertz

Hermes radio has an input to lock itself to a 10-MHz reference, and I connected an HP Z3801A GPS disciplined oscillator to it.

The audio I and Q samples are sent to the left and right channel of the speaker producing binaural audio—a slightly strange sounding audio, but quite listenable.

The same samples were sent to a 10-Hz low-pass filter (providing a passband from negative 10 Hz to positive 10 Hz around the carrier, a total of 20 Hz width) and then to one of the GNU Radio instruments. The test run collected many samples and performed a FFT (Fast Fourier Transform) of 4096 complex samples to look at the frequency domain from the time domain samples. The carrier of WWV was readily apparent. Sixteen different FFTs were averaged to look at the frequency closely. It was possible to isolate the carrier to a resolution of about 0.005 Hz.

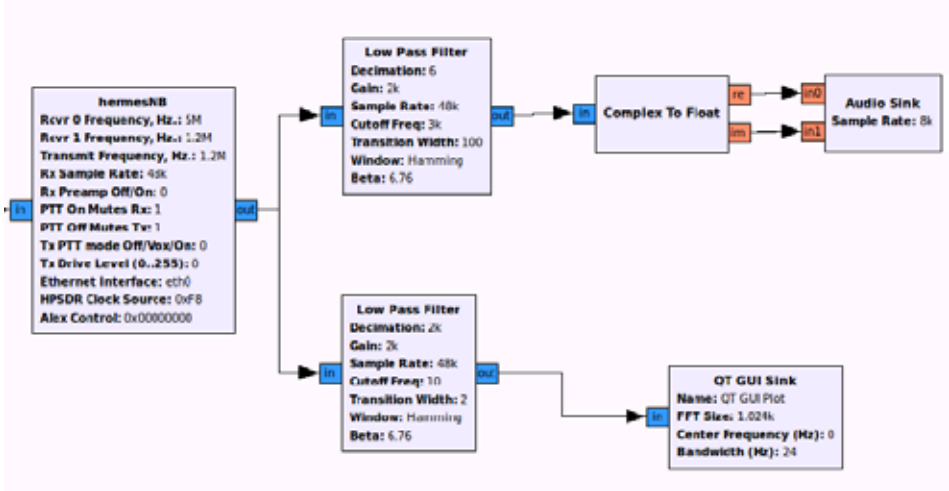


Figure 3. Test set-up

In an evening run, there was about –0.14 Hz of Doppler shift on the carrier due to the ionosphere drifting upwards. During a daytime run, the ionosphere was more stable and the frequency was within about 0.02 Hertz. The deterministic frequency error of the Hermes radio is roughly 0.01 Hz at 5 MHz (the accuracy changes with frequency due to roundoff error in the digital frequency generators within the radio), but that error can be precomputed and compensated for.

The three experiments were extremely fast to setup, run, and collect and analyze data. Here are some screenshots of the WWV spectrum obtained and the simple configuration required for the setup.

The peak of the spectrum is about –0.05 Hertz. There is some error due to the radio itself, and some due to ionosphere Doppler shift. This is just the tip of the iceberg for what could be done with Gnuradio.

If you want to learn DSP without writing code, this is a good and a free way to do it.

###

Packet Pioneer, Hank Oredson, W0RLI, Silent Key

By Stana Horzepa, WA1LOU

The Xerox 820-I was a computer, introduced around 1980 that used a Zilog Z80 central processing unit (CPU) and ran the CP/M operating system. It featured 64 kbytes of RAM, two 8-bit parallel ports, two serial ports, a disk controller and an 80-column by 24-line video display, all on a single printed-circuit board.

In the early 1980s, Xerox discontinued selling the "820" and sold off the remainder that they had on hand as surplus. Their warehouse in Texas was the source for the surplus computer and they could be had for as little as $50 per board!

Hank Oredson, W0RLI, obtained an 820 and wrote a software package for it that permitted it to function as a packet bulletin board system (PBBS) and more. Besides the normal functions that you would expect to find in any BBS, such as the ability to send and receive messages and files, Hank added two features that made his BBS even more powerful.

One feature took advantage of the two serial ports provided by the 820. With a TNC connected to each port, Hank's software permitted a user connected to the TNC on one port to communicate through the TNC connected on the other port. Assuming that each TNC was connected to radio equipment operating on different frequencies, this system provided a gateway from one frequency to the other. For example, if the TNC on one port was connected to a 2-meter transceiver and the TNC on the other port was connected to an HF transceiver, a PBBS user on 2 meters could use the gateway to make packet-radio connections on HF.

Another feature created a rudimentary packet-radio network for the automatic forwarding of messages between PBBSs. This mail-forwarding function allows PBBS users to send mail to the users of other PBBSs.

The W0RLI PBBS revolutionized ham radio and made packet radio the most popular mode of its day. Hank's PBBS begat other PBBSs and other networking schemes that so popularized packet radio that a TNC became as ubiquitous as a key or microphone in the ham shack.

I became a regular correspondent with Hank pestering him with endless questions as I researched packet radio articles, chapters, and books that the ARRL published during the heyday of packet radio. Hank never tired of my questions and patiently answered each one in a friendly and often humorous manner.

Hank Oredson, W0RLI, became a silent key on January 6 after a long battle with lymphoma. He will be missed, but his ham radio legacy lives on.

###

Hank Oredson, W0RLI nee WN0RLI, circa 1954, received the Dayton Hamvention Technical Excellence Award in 1987.

Homemade WWVB "Radial LED Clock"

By Will Beals, NOXGA

A Long, Long Time Ago

Many years ago, as a kid somewhere in the late '70s, I saw my first Radial LED clock. It consisted of two concentric rings of LEDs, an outer ring of sixty LEDs (one for each minute/second) and an inner ring of 48 LEDs corresponding to each quarter hour. It was pretty simple. The outer ring lit two LEDs, one for the current minute and one for the current second. The inner ring lit an LED for the approximate hour. The entire clock was about 18 inches on a side. It had an optional feature that lit the outer ring in a full circle each second (1/60th of a second per LED) too, that livened things up a bit. For reasons I can't explain, it was something I really enjoyed watching.

A Not-So-Long Time Ago

Fast forward almost as many years, I started traveling to Europe on a fairly regular basis for business and in a few airports and hotels, I started seeing similar clocks again, this time with the same outer ring of LEDs showing progressive seconds, but this time with the time and date using a 5x7 LED matrix inside the ring. Remembering the clock as a kid, I decided that was a clock I had to have. Alas, they are very hard clocks to find. Many internet searches later, I could only find a few home brew clocks, a bunch of "propeller clocks" (also very cool, but completely different) and only a couple very expensive commercial clocks. If there is ever a formula for a homemade project, this was it, a strong desire for something extremely unusual that you can't just go get for a reasonable price!

Choosing the Design Tools

At the time I was working on the TAPR T238+ weather station project. I wanted to finish it first before starting another project. I needed to wait about a year before beginning. With the OK from my wife, I started around January 2006. I started with the printed circuit design. Fresh off the T238 project, I stuck with the same tools, Circad's CC98 design capture program. The free version had the full schematic editor and layout support. All it could not do was crate the actual files to send to the board shop, called Gerber files. John (W9DDD) knew about my project and volunteered to create them for me.

Choosing the Microcontroller

The first decision was what microcontroller or MCU to use. The same MCU as the T238, the Freescale MC68HC908GP32 was an easy pick, it had about the right amount of horsepower, was cheap, and well-understood. I was also looking for an excuse to use the relatively new eZ80 from Zilog. I have always had a weakness for the Z80 since my TRS-80. It was the first processor I programmed extensively, including assembly language programming. It was serious overkill for this project, but an MCU I really wanted to play with. The development kit I had acquired included a full C development environment, something I really wished I had for the 6808. To this either/or question I finally picked YES. It turned out there was enough unused space to accommodate footprints for both.

A Short Diversion

Before I get into too much detail on the clock design, it would be handy to look at a YouTube video of the clock running. For this a picture is truly worth a thousand words. A video of a clock isn't exactly riveting content, but will certainly help with understanding what is being described next. A link to the video is in the references at the end of the article.

The LEDs

I initially chose two concentric rings for the clock hands. An outer ring of 60 LEDs to show seconds and minutes, and an inner row of 48 LEDs to show quarter hours, the same as the clocks I had seen earlier. After staring at this a while, I realized that it was going to be hard to tell very accurately where the "hands" were pointing, so I added a third ring of LEDs in between to show the hour tick marks, giving an easy-to-see reference for the other two rings. Centered inside the ring I wanted an LED matrix large enough to show the full time and date (month/day). This meant two rows of five characters each. I planned on a 5x7 matrix for the individual characters, but wanted "full graphics" capabilities, so added an extra row/column of LEDs between the characters so I could smoothly scroll around the screen. This gave me a matrix that was 30 columns by 16 rows. Between the matrix and the clock hands I had 600 LEDs!

LED Placement

I was planning on this being a physically big clock. I wanted it readable from a long distance, so wanted it at least 12 inches in diameter. I found some good-sized pre-made 5x8 LED modules which when assembled gave me characters that were 2.5 inches tall. By the time I put my three concentric rings around the center matrix I had an outer

ring that was 14 inches in diameter. A bit bigger than expected, but it looked about right. Placing the LEDs required some interesting planning. I created a spreadsheet to create the X,Y coordinates of each LED location in the boards coordinates starting with distance/angle, converting to X,Y around the center point, then finally the offset to the bottom left of the board. My target placement resolution was 0.025". After placing all of the LEDs, I was able to extract the LED locations from the layout program and plunk them back into the spreadsheet as a check. One LED was off by 0.025", not bad!

The LED Matrix

Before putting down any traces, the layout of the LED matrix needed a lot of thought. For both the 68HC08 and eZ80 I was going to be relying on software for the display refresh, so needed to ensure that the method of getting the bits out of the MCU to drive the matrix was as efficient as possible. I also wanted the representation of the LED matrix in memory to be something easy to manipulate in software for my drawing routines.

For the LED matrix I wanted to convert X/Y coordinates to a pixel location using nothing but shifts and adds, drawing ASCII characters to be nothing but shifts and ORs, and scrolling to be nothing but shifts as well. For this to work, I needed the mapping of bits to LEDs to be something like this:

Byte 0, bit:								Byte 1, bit:						Byte 3, bit:					
0	1	2	3	4	5	6	7	0	1	2	3	4	…	2	3	4	5	6	7
Byte 4, bit:								Byte 5, bit:						Byte 6, bit:					
0	1	2	3	4	5	6	7	0	1	2	3	4	…	2	3	4	5	6	7
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
Byte 60, bit:								Byte 61, bit:						Byte 63, bit:					
0	1	2	3	4	5	6	7	0	1	2	3	4	…	2	3	4	5	6	7

This mapping made the three LSBs of the pixel column the bit number and the byte address the column number/8 plus row *4. This ensured the drawing routines would be quite efficient.

I also spent a lot of time not only looking at the best way to implement the physical matrix, but also actually writing the display matrix drivers in enough detail to make sure I didn't have to do any unnecessary bit manipulations in software. At the most basic level, I needed to ensure that when writing data out the shift registers, it was a byte at a time, not a bit. With that and some very creative column numbering you see in the schematics, I was able to support the above logical matrix and still load all 64 bits of the shift register with 8 regular byte writes.

Duty cycle of the LEDs was another big factor. A 1/16th duty cycle would have been the logical choice, but would have sacrificed brightness, so I chose a 1/8th duty cycle. I ended up with an 8x80 matrix, two 8x30 matrices for the two rows of characters, and two 8x8 matrices for the minutes and hours plus tick marks. The matrix had a few gaps so each group started on an x8 boundary to simplify software drawing.

Display Drivers

The smart thing to do here would have been to pick a fully integrated LED matrix driver chip. They take care of all matrix and LED current management operations. However, they are set up more for "set and forget" displays. You load them up through a serial interface and let them manage the display. Since I wanted to have LEDs turning on and off up to 60 times a second, I was worried that serial communications jitter would result in a display with update jitters. Uncertain of what this would look like, I chose the safer "CPU drives the display matrix directly" approach. I chose a 1ms refresh rate where I would cycle through one row (80 LEDs) every 1ms in sequence, with any given row on for 1ms and off for 7ms. For the row drivers I chose 10 TPIC6C596 8-bit drivers. I have used chips like this before and they are simple, powerful, and convenient. They are essentially 8-bit shift registers with high power drivers and good enable/reset controls.

A hazard with a software-driven display refresh is that any variations in timing can result in display brightness variations since any timing jitter would leave a row of LEDs on for a longer or shorter time. It may have been overkill, but I did not want any software dependencies that could affect brightness. The solution was to have one control signal in charge of turning on and off all the drivers and have that signal under the direct control of a hardware timer. The timer firing also generates a software interrupt. The timer would immediately (and accurately) turn off the display, and software would then load up the new row data with some variability. After loading up the new row of data, the software would set another timer to precisely turn the LEDs back on. That way the turn-off and turn-on times would be crystal-accurate timing even

though the software to reload data would have all sorts of minor variations depending on interrupt times, some data variability, etc. I may not have needed this level of precision, but it was a known safe implementation.

Power

This was the first project where I really had to worry about and calculate power consumption. I wanted to maximize brightness, so looked at the maximum current I could drive though each LED. Assuming a 1/8th duty cycle, that turned out to be about 80ma. For driving 80 LEDs at a time, that was a peak current of almost 6.5A. The worst case power for the current limit resistors for each column turned out to be just over ¼ watt, so I used ½ watt resistors. The display runs from a 5V supply, giving a worst-case peak power of 32W. By my standards, this was a lot of power. The 80 column drivers weren't being stressed as they were only driving one LED at a time.

For the row driver, however, it was the full 6.5A worst case. From a power supply perspective, this could be 100% duty cycle, but for a given driver, no more than 1/8th duty cycle. I was going to need a big power supply and it was going to need to be inside the clock, not an external wall wart! I knew this was serious worst-case design as having all 600 LEDs on all the time was not going to be typical.

The row drivers were a lot harder since each driver had to source enough current to source 6.5A. For that I needed to consult an expert at my work place. Again I had the choice of a highly integrated driver or a discrete solution using ordinary parts. I also relearned that in the world of MOSFETS, sourcing current is a lot harder than sinking it. I chose the more discrete solution. Instead of a ready-to-go solution, my work buddy gave me the basic circuit and a freeware analog circuit simulator called SWCAD III from Linear Technology to simulate it and tweak as needed. This was actually quite fun to play with. A unique challenge for this driver was that it ran from a +5V power supply, but hand to accept control signals from either MCU. The HC08 had 5V CMOS control signals, but the eZ80 had 3.3V IO. I wanted the drivers to work with either control signal without needing any changes when changing between the two. One or the other wasn't too bad, supporting both was more involved.

One area where I didn't have the clear guidance I had hoped for was trace width. I was hoping for something simple like "for 6.5A using 1oz copper, you need a trace x mils wide". No such luck. The guidance was less clear. It was simply "for a trace x mils wide using 1oz copper, you get this many ohms per inch, go figure out if that is good or bad for your application". Without clear guidance I just picked the widest traces I practically could for the row drivers and made sure the resistance was much lower than the current limit resistors so as not to be a factor.

No Smoke

The general rule on LED matrix displays is that if you are driving them with a low duty cycle like 1/8th, you can drive the LEDs with a lot more current than if you were to just turn the LED on 100%. For the LEDs I was looking at, the max steady-state current was 15-ish ma, but with a 1/8th duty cycle, I could drive 80ma. Very good for maximizing brightness, but if I had a software bug that stopped refreshing the matrix I'd be driving 80ma to one row 100% of the time. I have learned the hard way that overdriven LEDs have a rather exciting habit of exploding and wanted to avoid that. The solution was a sort-of display watchdog. I had a single signal that enabled the display column drivers. It was enabled for most of the 1ms period, then turned off while I shifted in new data, then back on again. I ran that signal through a 74HCT123 so that as long as the enable time was less than about 1.5ms, it was constantly retriggering. If the enable time exceeded that 1.5ms, the timer fired and disabled the driver. I really wanted to use an LM555 timer, but could not figure out how to do it with just one chip, the '123 was a single-chip solution.

Schematics odds and Ends

Only a few design issues remained. The debug interfaces for the HC08 and the eZ80 were copies from either the T238 design or the eZ80. While I was confident I could use the HC08's internal flash for storing variables on the T238 project I wasn't so sure for the eZ80, so I added support for an external NVM. Finally, I added the connector for the external WWVB radio. This was a bit more theoretical than it should have been, just power, ground, and signal. That turned out to be a problem later.

The Layout

By my standards, this was going to be a huge board, just over 14 inches on a side. I knew it was going to be expensive too. More than two layers would have been much more. Any unfixable errors would not only result in a scrapped board, but likely most of the components loaded would have to be scrapped too, so accuracy was a high priority. Most of the schematic capture and layout work was accomplished during another series

of international business trips where I had a lot of time sitting in airport terminals and on cramped tourist class seats. It was a case of going very slowly and carefully and triple-checking everything. While the LED matrixes looked very nice on the schematic sheets, implementing them in a circle took several tries before coming together in a nice orderly fashion. The Circad software is an excellent tool for checking your routing and making sure you do things right. Hooking up the 5x8 LED modules turned out to be a lot more random-looking than I had hoped. Fortunately, I had a lot of space under the modules to do this. One thing I suspect I would never pass would be an EMI test. I had this big board with traces everywhere switching high currents and very little ground plane. It was a 2-layer board, so with some very careful routing I was able to have a pretty solid ground under the MCUs and drivers, the very nature of a matrix for the LEDs required both layers for traces.

Ordering and Getting Boards

Since boards were the major expense, I went ahead and ordered all of the components first, partly to make sure the symbols I had looked right, but mostly to make sure I didn't have an ugly surprise with a component unavailable after I had already made boards.

I finally ran out of things to check and ordered a pair of boards in February 2007. I got the Gerbers, checked them twice, and finally sent them to a fab house here in the Denver area. I did what negotiating I could, but they were still expensive, $260 for just two boards. When the call came in the boards were ready, I remember it being a particularly nasty snowstorm. It was a Friday afternoon and if I didn't pick up the boards that day, I would not be able to work on them over the weekend. The ride was exciting, especially for my subcompact car in snowdrifts. No other cars were on the streets in the industrial area the fab house was in, I am pretty sure for good reasons. It is one thing to be designing a board for almost a year looking at it on a screen, but still a surprise to see it as a real board for the first time. I did try explaining what it was to the people at the fab house, but don't think I was successful.

Uh-Oh!

Despite lots of checking, there were still a few problems I discovered while loading up the board. The first was that I used too small of a footprint for the ½ watt current limiting resistors. I had 80 of them, so this was a big deal. Very sharp bends for the leads sort of solved the length issue, but for the width issue had them stacked up in groups of eight right next to each other. The resistors were too wide, so I had to alternate resistors that were touching the board and ones that were raised above the board. The second issue was that I got the pin assignment wrong for the reset switch. That hurt as there was no easy way to make it look nice. That was the worst of it, so no scrap!

Initial Bring-Up

I decided to start with the more known MCU, the 6808. I loaded just enough power circuitry for the MCU and the "anti-smoke" logic. I wrote up some basic code to service a 1ms hardware timer tick and from that tick, generate the sequence of eight row drivers and the code to generate the enable/disable control line that is checked with the 74HCT123. I then stopped the MCU using the debugger and make sure the enable line still went to disable even if the MCU hit a breakpoint with its control line set to enable. It looked good. With that logic confirmed I loaded up all of the row and column drivers along with just one of the 5x8 LED matrix modules. That was enough logic and lights to debug the basics of my LED refresh code. That worked OK, so I finally loaded everything up. Soldering that many LEDs, current-limit resistors, and row/column drivers took the better part of a full day!

Finally getting the whole board loaded and seeing all the LED test patterns lighting up as they should was nice to see. I had some concerns about the 6.5A row drivers, but they worked fine, just as the simulations had promised. This was a good testimony to their accuracy. The other fun thing to do was the worst-case power test, lighting up all 600 LEDs all the time. It was an evening test, and LEDs are very efficient in terms of lumens per watt - 32 watts of red LED power really lit up the room! I fired up the test, and every few minutes checked the temperature of the various critical power components, things got warm, but nothing got uncomfortably hot to the touch. This test was done with the board fully exposed, not in a confined case. This implies I was probably way too conservative in my power and temperature deratings.

6808 Coding Challenges

The two biggest challenges for programming the 6808 were limited RAM space and CPU speed. I used the same CPU as the T238, an MC68CH908GP32. It has 32Kbytes of flash which was more than enough, but only 512 bytes of RAM. Of that 512, only

192 bytes were directly accessible by fast instructions. The rest required indirect access which is a lot slower. The display memory clearly needed to be in this fast area of memory. Ideally I wanted two copies of the display memory, one copy to draw in and another copy for what was being actively displayed. That way you can avoid seeing strange artifacts when drawing to a "live" display. That wasn't practical, however. Each copy would be 80 bytes and I still needed RAM for variables and such. From the T238 project, I reused my boot loader program, which also provided a convenient set of routines for using the 6080s flash memory for variable storage. There wasn't much to store, just the offset from GMT to the local time.

The other big challenge was CPU speed. To keep the display from visibly flickering, I needed a refresh rate greater than 30Hz. All eight rows needed refreshing every 30Hz, meaning a maximum of 4ms per row. I opted for a 1ms refresh rate with the idea that if that was too fast for the CPU, I could slow it down a little, but not much. With a CPU clock speed of 8MHz that gave me 8000 clocks or about 1600 instructions per refresh cycle. Making matters worse though was that if I was going to run the MCU in debug mode (most of the time during development), the CPU frequency was reduced to 2.4576 MHz, giving me less than 500 instructions per interrupt! You know you are pushing things when you are counting instructions! With the planning I had done earlier on how to efficiently drive the display logic, the code to do it did end up being fairly efficient, on the order of 10% of the 1 ms cycle time.

I still needed to ensure that I would not get any unintentional flickering when drawing to the display memory. The method I chose was to actually perform the drawing functions as part of the display refresh interrupts. This was particularly important for the "sub-seconds" hand. This was a "hand" that rotated once a second in order to make the display a bit more lively. For this to look smooth, I had to ensure that one LED was turned on for exactly 1/60th of a second. If I missed by even a little bit, the display would look like it was jittering and an LED would look either brighter or dimmer depending. By doing the "drawing" inside the display refresh routine I could precisely control how long each LED was lit.

By extension it made sense to perform all of the drawing functions inside the interrupt routines. In general this worked OK, but I did run into a problem when I was updating the entire display for something like an hour roll. The problem was that the amount of time it was taking to completely redraw the LED matrix for a time update was taking longer than the 1ms display refresh time. In particular, it was the necessary bit manipulations to redraw 10 characters worth of 5x8 matrix data. There weren't enough CPU cycles in debug mode to complete the drawing before the next interrupt. Since I was still in the interrupt routine, the interrupt was being missed. The display was flickering and even though I couldn't measure it, time wasn't being tracked accurately either. My solution was to very carefully make the timer interrupt re-entrant. This is something not normally done with small microcontrollers. After performing the timing critical display refresh functions and if I was about to jump into the LED matrix drawing functions, I re-enabled interrupts. This allowed the drawing routines to be further interrupted by the display refresh interrupt. Upon completing that interrupt, the CPU returned to the display drawing. I was a little worried this would result in some visible drawing artifacts, but it did not. I knew by design it was not possible to have two back-back LED matrix drawing commands, so this was a fairly safe solution.

Timer Task State Machine

Besides the very time critical chore of running the display refresh, there were a lot of other tasks that needed attention. None needed attention every millisecond though, so I created a state machine to handle the other important, but not so frequent tasks. It is basically a sequencer that goes through 50 states and then starts over. These other tasks included realtime keeping, time "drawing", button scanning, WWVB signal decoding, and in the case of the eZ80, a transition engine. The idea behind the state machine is to take many of the other tasks that need attention on a regular basis and space them out over these 50 states so that you can ensure each task get the attention it needs, but that the tasks would not happen at exactly the same time. One place I cheated a bit was that 1/60th of a second counter. Its period is 16.666 ms, but I had a resolution of 1 ms. My solution was to increase the 60ths of second count slightly unevenly, doing so in the sequence 17 ms, 17 ms, 16 ms. I was worried this might not look consistent when flashing 1/60th of a second, but could not tell the difference.

Getting Started with the eZ80

With the 6808 the challenge was getting the features and performance I needed with the very limited resources of an MCU I was very familiar with. With the eZ80, the challenge was much more about (re-)learning a new CPU. Most of the code for this MCU was going to be in C. Zilog offered a very affordable design kit that included an

eZ80 development board, full C compiler, debugger, and IDE for a very reasonable $100. Even though it was my first CPU to program in any language including assembly, it had been many (25+) years since I had last done any Z80 programming. The basic register set looked familiar and while the Zilog folk made sure that the eZ80 still ran old Z80 assembly code, they made massive updates to the instruction set to extend the arithmetic and addressing functions to be 24 bits instead of 8/16. The changes are all for the better. I am still not sure I was using the new modes to their fullest, but it was enough to get me going. Bandwidth was certainly not an issue, the eZ80 CPU clock was 50 MHz and while the CPU isn't the classic RISC "1 instruction per clock", it was a whole lot better than the old Z80s 5-12 clocks per instruction.

eZ80 Coding

Since I already had a debugged and working clock using the 6808, the eZ80 programming was mainly taking the 6808 routines one at a time and rewriting those assembly language routines in C for the eZ80. I stuck to the same basic software architecture. My goal was to do everything in C unless there was a compelling reason to drop to assembly. Since CPU bandwidth wasn't an issue, I could have done everything in C, but I still like efficient code. This was my first time doing embedded C programming, so with the help of the eZ80 C compilers assembly language source code output, I spent a lot of time looking at what kind of assembly language it generated for specific C coding practices. I had never done this before and it was extremely educational. For pretty much every routine I wrote it in C, I looked at the assembly language output. The results were often surprising, but after learning to "think like a compiler", I started getting better at writing C code that created efficient assembly code. With this knowledge, much of the code improvements were simply by rewriting the C code in a manner better suited for the compiler instead of dropping to assembly language. The biggest lesson is that while in assembly it is easy to deal with 8, 16, (and in the case of the eZ80 24)-bit integers, in C it is best to stick with whatever the native integer format is no matter what size you actually need. For the eZ80, that is 24-bit. The reason is that in order to not run into unexpected truncation issues, almost all math is performed at the native integer size and if your operands and/or result are anything but that native size, extra instructions are added to convert operands to 24 bit and results from 24 bit. This isn't unique to the eZ80 either; in my programming experience on other CPUs since, this has turned out to be common.

Another big C issue is that it does not handle shifting that well. You can shift single bytes/words, but cannot efficiently shift multibyte arrays of bits. There is no way in C to get to the assembly language "shift-through-carry" instructions that are designed for this kind of operation. For a lot of the bitmap operations where shifting was needed, I did drop into assembly language.

Overall, I was very pleasantly surprised at how much of the clock code was handled quite efficiently in C.

eZ80 Enhancements

With a much more powerful processor and more memory, there were a few extra things I wanted to do with the eZ80 that were not practical with the 6808. The big one was a transition engine. This involved first drawing out your new image in a staging buffer, then calling this engine to copy the image in the staging buffer to the "live" buffer. This decouples the drawing functions from being time critical and then give you lots of flexibility in how you copy the new image to the live buffer. At the simple level it is just a block copy, but I got fancy and did wipes, scrolls and shifts from any direction. The 6808 certainly didn't have the memory for two buffers and probably didn't have the horsepower either. This turned out to be a lot of code, but was fun to write and certainly added some flair.

The WWVB Radio

I am very much a fan of WWVB timepieces. I have had an "atomic watch" for over a decade and any clock where I have had a choice in the matter has been WWVB-capable too. I'm big on accurate time and having a precise timepiece suits me just fine. So, naturally, if I am going to make a clock, it had better be WWVB-capable. My online research at the time for a standalone WWVB radios was very disappointing. I could find nothing consumer grade, just a few commercial radios that were crazy expensive. I finally decided to be creative and just go buy a couple cheap $10 WWVB wall clocks hoping I could find the radio guts and "borrow" the WWVB signal. It turned out to be more of an adventure than I expected.

My first surprise after getting two clocks was that there were no standard packaged chips in the clocks with nice convenient part numbers printed on them. Instead, the clocks were built using chip-on-board methods where the die is put on the board and then just a dollop of glue is put over the top. No markings! I started with some very

wide Google searches for WWVB radio chips and with a little luck came across some chip datasheets where it was pretty obvious the chip was mostly sold as bare die. That seemed to be a good lead. Of the three anonymous blobs of goo, the one with the radio chip was easy to identify, it had the antenna hooked up to it! I got out my trusty oscilloscope again and started probing all of the signals around that chip. I had a match! The match had two surprises though. The first was that the radio chip required an enable signal. I could not just leave the radio permanently enabled as the chip used the transition from disabled to enabled to set some AGC levels. This required some wires be added to the board. The second surprise took me a bit to figure out. I could see the digital WWVB signal with my oscilloscope just fine, but as soon as I hooked it up to my MCU, the signal died. I thought it might be power quality, interference from the clocks display logic, bad logic levels for the enable signal, but nothing fixed it. After much head-scratching I wondered if it was output impedance and a more careful review of the datasheet showed this was likely the case. That is a specification I normally do not pay much attention to. At this point I didn't want to pay a fortune to mail order a single 10-cent part, so my plan B was to find a way to make whatever I could find at Radio Shack work. All they had that fit the bill was an LM339 quad comparator. With a little fiddling I got it breadboarded up and it provided the necessary high impedance load to the radio and low impedance drive to the MCUs. The only last detail was the radio chip was designed for a single 1.5V battery and I had MCUs that had either 5V (6808) or 3.3V (eZ80) IO. Dividing down the MCU control signals was easy and now that I had a comparator for the WWVB signal, getting the output amplified to 3.3V was pretty easy too. I had control and I had a signal to decode!

Decoding the WWVB Signal

This was fun. I decided to just start with the WWVB specification and design something from scratch. I suspect I could have found some software to do this, but wanted to do it on my own. The NIST website has some very informative documents on not only how the signal is formatted, but a lot of advice on how best to decode the signal as well.

As with the rest of the clock, I started with the 6808 assembly, then migrated to the eZ80. I already had my 1ms display timer tick, so used a 1/50th divisor of that as my signal sampling clock. I took those samples and ran it through a "best of three" filter to handle any simple signal glitches. Then, based on the three official pulse lengths, had a symbol decoder state machine report symbols as they were timed. This symbol decoder state machine would essentially report one symbol (or an error if the pulse length wasn't valid) per second from the WWVB signal.

I then constructed another state machine to do the actual data gathering. It simply waited for two Position Markers in a row (how WWVB identifies the start of the minute as well as start of the data sequence) and then started gathering symbols as the symbol decoder reported them. Besides the pair of markers that signify the start of a minute, there are also markers within the minute. If I ever got an invalid symbol or a marker in the wrong place or no marker where one should be, I restarted the data gathering process.

This very simple data gathering process with limited data integrity checking is not very robust. The noise filtering could be much more sophisticated and even the WWVB best practices documentation strongly suggests getting two complete sets of time data and making sure they are exactly 1 minute different as a data integrity check. I live in Colorado about 60 miles from the Fort Collins WWVB transmitter. I get fantastic signal quality 24/7. If this clock ever had to work outside Colorado, this code would need considerably more work.

Debugging the WWVB decoding process presented its own challenges. It was well and good to see that I could correctly decode the present time (I had 6+ other atomic clocks and watches to compare against), but there were a big number of special cases that I had to check out like time rolls, going in and out of Daylight Saving Time, leap years, leap seconds and other such special cases. What about acquiring on the day of a DST transition, for example? For this I created two spreadsheets, one where I could plunk in the raw data I received and see that it was the right time and a second spreadsheet where I could enter a desired time and condition and it would give me the desired string of symbols that represented that time. I then plunked that data into my program as if it had come from the symbol decoder state machine. If you look at the source code, I have a whole array of these test cases for validation purposes.

Two WWVB Time Formats

One of the more interesting discoveries while working on the WWVB decoder was the transmission supports two different ways of dealing with the pesky issue of the world not quite rotating in exactly 24.00000000 hours. The transmission provides a

1/16th of a second correction value as well as data for the more famous leap second. This is something pretty safe to ignore, but it looked like something fun to play with. By coincidence, the December 2008 leap second was announced as I was wrapping up the leap second support code, so I had a deadline! I was able to finish it in time and had the clock perform some special antics when a leap second happens. There have been two leap seconds since and they are always fun to watch.

Things To Do Someday

There are two things I would still like to do some day (besides better WWVB signal processing). The first is clock calibration. That is, looking at how far off your local time is every time you resync to WWVB to see how far off frequency your local crystal is. Knowing that, you can then make adjustments during the day so that the time is really accurate. For the eZ80 board I sync up to WWVB at midnight and by late evening, I can be off as much as 15 seconds. This would be a big deal if I didn't resync every night. I don't have a WWVB receiver hooked up to my 6808 clock, but by luck, the crystal for that clock is extremely accurate. We seem to go about 6-12 months between power failures at work and it has never been off more than 10 seconds between those power failures.

Second, I would really like an excuse to play with the eZ80's Ethernet support. My original intent was to have it get time from a network time source, but since have seen one article in Circuit Cellar, Ink where someone had a project with a WWVB decoder but then uses that to become a network time server. This would be a pretty big effort as in order to support Ethernet traffic I would have to use a Zilog-supplied RTOS. That would mean messing up the very carefully crafted display refresh timing logic, something I am hesitant to disturb.

Third, I really should try to put the clocks in a case. As any of the T238 kit builders know, I punted on case making for that project too. I am not good at it and when I do try, they end up looking pretty bad. I have ideas on a nice looking case, but don't have the ability to make it. So, for now they are still naked printed circuit boards just as you see it on the YouTube video. I do have a 120VAC power supply on the back, so not the greatest from a safety perspective!

Conclusion

There is a certain fear in doing a project you have thought about and wanted to do for several decades. Will it live up to your expectations? This clock wasn't a rational project in that it wasn't doing anything that couldn't be done with a bought clock one tenth the price. It just looks cool and was a kind of clock I wanted since being a pre-teen. In hindsight, I believe a lot of the conservative design choices both for power and CPU timing were overkill and it could have been simpler and cheaper, but I'm not a believer in chasing hindsight. I can even rationalize the lack of a case adds to the tech look. I have one clock in my home office and one in my work office, one running the 6808 and one the eZ80. I never tire of looking at them and enjoy the queries from people visiting my office and seeing the clock for the first time. The clock certainly did live up to my expectations!

References

Circad CC98: http://www.holophase.com/ This is the schematic capture and layout program I used for the design. Like any program of this type, it does take a bit to get up to speed, but it does everything you need including symbol creation. The only thing the free download does not do is output Gerber files. The nice part is that if you know someone with the paid version, then you can share files. This is a great setup for a club or small organization.

Freescale MC68HC908GP32: http://www.freescale.com/ (Search for mc68hc908gp32) A very nice general purpose 8-bit microcontroller.

eZ80: http://www.zilog.com Full part number is the eZ80F91, letters after that are packaging options. For the modest cost of a development kit you get everything you need, a motherboard with a removable module you can then plug into your project, a full (and very nice) C development environment complete with in-circuit debugging capabilities and plenty of libraries and example code. As nice a development environment as I have ever played with.

SWCAD III http://www.linear.com/designtools/software/. The original software I used has now been replaced with LTspice IV. Spice is a general purpose analog simulation tool and this is Linear Technologies free version of that software. You get to enter in your schematics, and they have models of not only their power devices but good models of many generic devices as well. With this software I was able to simulate and tweak the power driver circuitry for this board to make sure it would work with the specifics of my particular constraints. With a working simulation, the relatively

complex drivers worked the first time without any issues or tweaks needed.

WWVB radio chip: http://www.mas-oy.com/products/radio-controlled-clock-rcc/mas6180/. If the chip in the cheap clocks I bought were not this chip, they were effectively clones of it. Everything I noted/measured lined up with the specifications of this chip specification.

WWVB signal specs and best practices: http://www.nist.gov/pml/div688/grp40/radioclocks.cfm Good description of what you are supposed to do if you are making a WWVB-controlled clock. The spec at the top of the page was my reference for what to do. Very good reading if you are thinking of anything WWVB-related.

YouTube video of the clock: http://www.youtube.com/watch?v=16RcBb6JFmc

Design files: http://www.beals5.com/clock. Schematics, silkscreen, and source code for your viewing enjoyment. If interested in the Gerber files, let me know.

###

ARRL Board of Directors' Digital Matters

From the minutes of the ARRL Board of Directors Annual Meeting, January 18-19,2013:

38. On motion of Dr. Woolweaver, seconded by Mr. Bodson, the following resolution was ADOPTED:

Whereas, the symbol rate limitations found in the FCC's rules governing the Amateur Radio Service were codified in 1989, and

Whereas, advances in digital communications techniques have rendered these limitations obsolete, and Minutes,

Whereas, the continuation and extension of the amateur's proven ability to contribute to the advancement of the radio art is a fundamental purpose of the Amateur Radio Service;

Now therefore be it resolved, the ARRL Board of Directors establishes an Ad Hoc Symbol Rate Rule Modernization Committee (the Committee), and

Be it further resolved, that the Committee shall evaluate potential modifications to the Amateur Service rules to permit and facilitate the use and development of high symbol rate digital communications techniques, and

Be it further resolved, that the Committee shall recommend modifications to the ARRL Board of Directors for consideration at the Board's July 2013 meeting.

###

Write Here!

PSR is looking for a few good writers, particularly ham radio operators working on the digital side of our hobby, who would like to write about their activities here.

You don't have to be Hiram Percy Maxim to contribute to *PSR* and you don't have to use *Microsoft Word* to compose your thoughts.

The *PSR* editorial staff can handle just about any text and graphic format, so don't be afraid to submit whatever you have to wa1lou@tapr.org. The deadline for the next issue of *PSR* is April 15, so write early and write often.

If *PSR* publishes your contribution, you will receive an extension to your TAPR membership or if you are not a member, you will receive a TAPR membership.

###

On the Net

By Mark Thompson, WB9QZB

Facebook

As you may know, TAPR has a Facebook page, www.facebook.com/TAPRDigitalHam.

However, recently I also created a TAPR Facebook Group, www.facebook.com/groups/TAPRDigital/.

If you have a Facebook account, "Like" the TAPR Facebook page and join the TAPR Facebook Group.

If you join the group click on the Events link and indicate you're Going to the events.

On Twitter, Too

Access the TAPR Twitter account at www.twitter.com/taprdigital.

Also on YouTube

TAPR now has its own channel on YouTube: the TAPR Digital Videos Channel: www.youtube.com/user/TAPRDigitalVideo.

At this time, there are over 30 videos on our channel including many from the TAPR-ARRL Digital Communications Conference (DCC) that you may view at no cost, so have at it!

###

PSR

#121 Winter 2013, ISSN: 1052–3626
Published by
TAPR
Phone 972–671–TAPR (8277)
E-mail taproffice@tapr.org
URL www.tapr.org
Facebook www.facebook.com/TAPRDigitalHam
Twitter www.twitter.com/taprdigital
TAPR Office Hours: Monday to Friday, 9 AM to 5 PM Central Time

Submission Guidelines

TAPR is always interested in receiving information and articles for publication. If you have an idea for an article you would like to see, or you or someone you know is doing something that would interest TAPR, please contact the editor (wa1lou@tapr.org) so that your work can be shared with the Amateur Radio community. If you feel uncomfortable or otherwise unable to write an article yourself, please contact the editor for assistance. Preferred format for articles is plain ASCII text (OpenOffice or *Microsoft Word* is acceptable). Preferred graphic formats are PS/EPS/TIFF (diagrams, black and white photographs), or TIFF/JPEG/GIF (color photographs). Please submit graphics at a minimum of 300 DPI.

Production / Distribution

PSR is exported as Adobe Acrobat and distributed electronically at www.tapr.org
PSR Editor:
Stana Horzepa, WA1LOU
E-mail wa1lou@tapr.org

TAPR Officers

President: Steve Bible, N7HPR, n7hpr@tapr.org
Vice President: Jeremy McDermond, NH6Z, mcdermj@xenotropic.com
Secretary: Stana Horzepa, WA1LOU, wa1lou@tapr.org
Treasurer: Tom Holmes, N8ZM, n8zm@tapr.org

TAPR Board of Directors

Board Member, Call Sign, Term Expires, e-mail address
John Ackermann, N8UR, 2013, n8ur@tapr.org
Steve Bible, N7HPR, 2014, n7hpr@tapr.org
Dan Babcock, N4XWE, 2013, n4xwe@tapr.org
George Byrkit, K9TRV, 2015, k9trv@tapr.org
Tom Holmes, N8ZM, 2015, n8zm@tapr.org
Stana Horzepa, WA1LOU, 2014, wa1lou@tapr.org
John Koster, W9DDD, 2015 w9ddd@tapr.org
Jeremy McDermond, NH6Z, 2013, mcdermj@xenotropic.com
Darryl Smith, VK2TDS, 2014, vk2tds@tapr.org

TAPR is a not–for–profit scientific research and development corporation [Section 501(c)(3) of the US tax code]. Contributions are deductible to the extent allowed by US tax laws. TAPR is chartered in the State of Arizona for the purpose of designing and developing new systems for digital radio communication in the Amateur Radio Service, and for disseminating information required, during, and obtained from such research.

PSR Advertising Rates

Full Page Ad for 1 issue: $100, 4 issues: $350
Half Page Ad for 1 issue: $75, 4 issues: $250
Quarter Page Ad for 1 issue: $50, 4 issues: $175

Membership Application

TAPR
P. O. Box 852754, Richardson, TX 75085–2754
Phone 972–671–TAPR (8277), Monday–Friday, 9AM–5PM Central Time
E–mail taproffice@tapr.org URL http://www.tapr.org
Join online at http://www.tapr.org/organization.html#membership

Benefits of a TAPR Membership:

- *Subscription to the quarterly PSR*
- *10% off most TAPR kits and publications*
- *Access to the TAPR digital library*
- *Latest information on TAPR R&D projects*
- *Co-sponsor of the annual TAPR-ARRL Digital Communications Conference (DCC)*

Name_____ Call Sign_____

Address_____

City_____ State/Province_____ Postal Code_____

Country_____ Daytime Phone No._____

E-mail Address_____

New - $25 ☐ Renewal - $25 ☐

Payment Method: Check ☐ Money Order ☐ Credit Card ☐

STOP! Provide the following information only if paying **by mail** with a credit card:

VISA ☐ Mastercard ☐ Discover ☐ JCB ☐

Credit Card No. _____ Expiration Date_____ Security Code_____

Card Holder's Name_____

TAPR is a community that provides leadership and resources to radio amateurs for the purpose of advancing the radio art.