RUDI VAN DRUNEN

a home-built NTP appliance

Rudi van Drunen is a senior UNIX systems consultant with Competa IT B.V. in The Netherlands. He also has his own consulting company, Xlexit Technology, doing low-level hardware-oriented jobs.

rudi-usenix@xlexit.com

AS PART OF THE HARDWARE SERIES, I will describe how to build your own Stratum 1 NTP server. I will give you the recipe for connecting an OEM serial port GPS device to a Soekris 4501 board and building an embedded image for it to operate as a Stratum 1 time server with an accuracy of better than five microseconds. It is not only a fun hardware project for a time-nut [12] but also results in a cheap piece of hardware that actually will improve your infrastructure at home or in the data center.

NTP

NTP (Network Time Protocol) [1] is a standard that does clock sync and is formalized and described in detail in RFC1305 for version 3. NTP version 4 is a significant overhaul. The simple version of NTP v4 is described in RFC 2030.

Currently NTP, or the urge that machines keep in sync with each other as far as time is concerned, is extremely important in logging and journaling, stock market, air traffic control, and gaming systems. Almost every modern application nowadays that relies on distributed infrastructure needs some kind of time synchronization.

NTP ARCHITECTURE

NTP relies on a number of different servers that provide time information to the client. These servers are organized hierarchically: Stratum 1 servers get the time information from a direct time source such as a radio clock, GPS, or specialized hardware, such as a Meinberg or Lantronics device. Stratum 2 servers take this time information and distribute it onto either Stratum 3 servers or clients. Stratum 3 servers do the same.

The ntp daemon takes the time information and inputs this into an adaptive algorithm to discipline the local clock against using a phase/frequency locked control loop (see Figure 1). The time protocol over the network is designed to be robust against lag and jitter. NTP also uses algorithms to mitigate multiple time sources and detect or avoid improperly configured servers.

A good article about NTP can be found in an earlier issue of *;login:* [2].

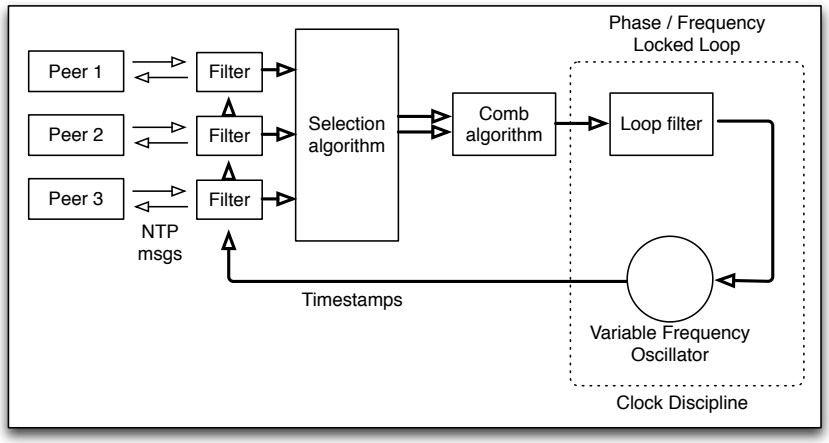

FIGURE 1: NTP ARCHITECTURE

BUILDING YOUR OWN

It is not that difficult to build a local Stratum 1 time server appliance for a relatively low price. A small embedded system can (as a rough estimate) provide time information to over 200 clients easily. The remainder of this article will provide a recipe for building a Stratum 1 time server using a Soekris 4501 embedded board and a Garmin OEM GPS.

GPS

The Global Positioning System (GPS) relies heavily on time information. The information that the receiver gets from the different GPS satellites is contained in timestamps. All of the GPS satellites contain a very accurate clock and time standard from which they derive the timestamps. The difference in the timestamps the GPS receiver receives from the satellites and the position of the satellites makes it possible for the receiver to triangulate and calculate the absolute position of the receiver.

As a byproduct, the GPS receiver is aware of the accurate time, which is often output in a string, together with the position and some other information, on a serial port. The time information in this string has a large jitter due to the serial port communication. Uncertainty exists about at which moment the communicated time corresponds with the exact time, i.e., at the beginning of the string or at the end.

To overcome this problem, many GPS devices have a pulse output, Pulse Per Second (PPS), that marks and synchronizes the start of a second. This pulse enhances the accuracy and reduces the jitter of the time information. We will use both the serial output string and the PPS output of a GPS device here.

For this setup we will be using a mouse-like GPS device with serial output and a PPS output. I used a GPS-18xLVC OEM [4] from Garmin. This hockey-puck-sized GPS has a serial output and a PPS signal. It runs on 5V and uses approximately 60mA, easily obtained from the host.

The de facto communication format for most GPS devices is called NMEA output. This is an ASCII data string, containing lines comprised of an info string (starting with $) followed by a number of parameters. Many different "sentences" are available: outputting satellite constellation, position, time, velocity, direction, etc.

To connect the GPS to your serial port, you need to configure the GPS to use 4800 baud output and to only output the $GPRMC sentence, approximately once per second. The PPS output also needs to have a defined pulse length of about 200ms.

If your GPS's PPS signal is too small and you cannot change it, use a pulse stretcher such as the FATPPS device from TAPR [10]. The NTP device driver for GPS accepts the $GPRMC NMEA sentence on the serial port RxD line and the PPS signal on the DCD pin of the serial port.

For the Garmin GPS, Windows-only (sorry) software (SNSRCFG) needs to be used for setting the operating mode, which sentences are sent, how often they are sent, PPS pulse width, etc.

Soekris

The embedded board I used is a Soekris 4501 system [5]. This embedded board contains a 133 MHz AMD Elan SC 520, 64MB SDRAM, three Ethernet interfaces, a CF card slot, and two serial ports. The device runs FreeBSD without problems and fits in a small metal enclosure. The board is shown in Figure 2.

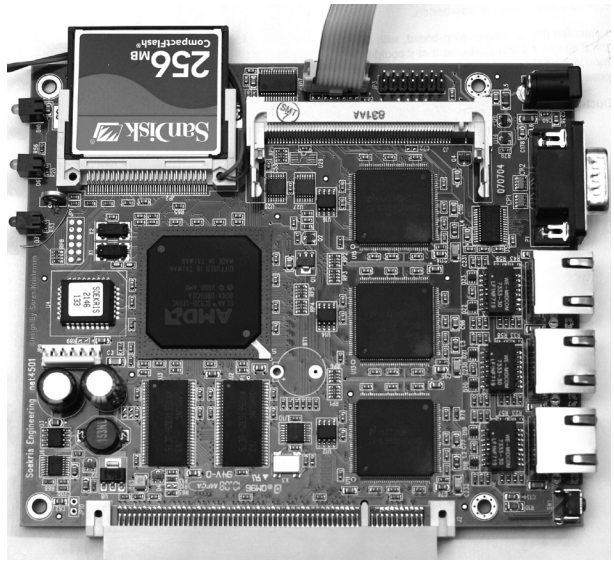

FIGURE 2: THE SOEKRIS 4501 EMBEDDED BOARD CAN EASILY SERVE 200 HUNDRED CLIENTS WITH NTP TIME INFORMATION.

FreeBSD has an implementation of the API for accessing timestamps attached to external signals, PPS API (RFC 2783). This API is implemented using the DCD line of the serial port as an input for the external signal. However, for Elan 520 processors there is a special kernel option for using one of the chip's time counters to do hardware timestamping of external signals, yielding a resolution of approximately 125 nanoseconds and a precision of +/- 125 nsec. This results in a far better timestamp compared to using the DCD line of the serial port. Note that this feature is only applicable to the AMD Elan SC520 processor.

Software

NanoBSD [6, 7] is a FreeBSD distribution aimed at small devices such as the Soekris 4501. It is packaged as script which comes with the stock distribution of FreeBSD. This script runs on a development host and takes a

configuration file which generates an image that can be flashed onto a compact flash card. The CF card then works as an ATA disk in the target board. NanoBSD is built with the use of flash in mind, so it generates a system that has a read-only root file system and memory file systems for /etc and /var to overcome problems with flash cards, which by design have limited write cycles.

Your NanoBSD system also contains a script to update the complete system without removing the CF card from the device. NanoBSD has facilities to include packages into the flash image. Here we will be (only) using NTP from the ports tree as a package.

Putting Things Together

HARDWARE

In order to connect the GPS to the Soekris, the second serial port on the board will be used. A header-flatcable connector to JP11 can be used for this. The COM2 port (JP11) on a 4501 takes 5V levels. If you happen to have a GPS that supplies 3.3V levels, you might need level-shifter circuitry; MAXIM has chips for that. Note here that the PPS signal as well as the serial port will be using 3.3V signaling. As you introduce extra hardware on the PPS line (such as a level shifter or pulse stretcher), you need to take into account a delay that this circuit has on the software side in /etc/ntp.conf.

If your GPS does not come equipped with a 9-pin D serial connector and it accepts standard RS232 levels, it would be good to put one on, as you then can test the output of the GPS on another system and/or change settings.

Often the OEM GPS devices with PPS output need to have a separate power supply. Small devices can be powered from the Soekris device using pin 11, 12, or 13 (Gnd, 0V) and pin 2 (+5V) of connector JP3. Be careful not to make any shorts here, as the power conditioning/supply on the Soekris board probably is not short-circuit protected.

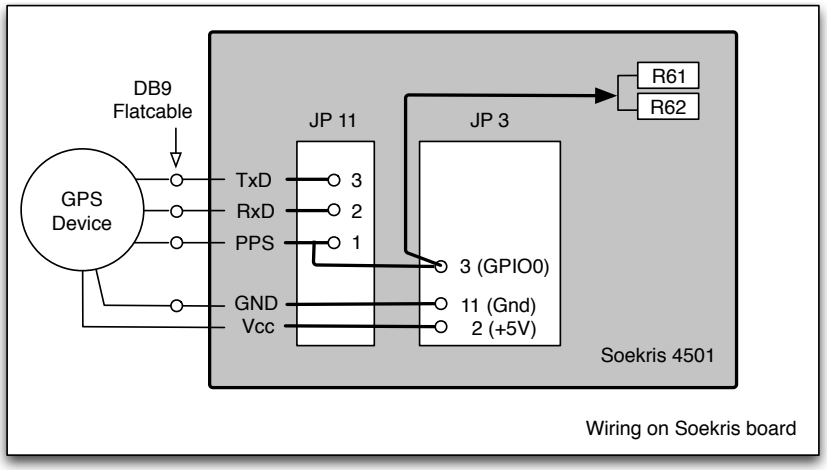

FIGURE 3: WIRING FROM GPS TO 4501

The PPS signal has to be wired to a GPIO pin. Here we use GPIO 0, which is pin 3 of JP3. In order to also be able to use the high-resolution timer on the Elan chip, an additional wire needs to be run on the Soekris board, from the PPS signal (JP3, pin 3) to the junction of R61 and R62 (in turn connected to

the TMR1IN pin of the processor, grid number AA24, but unreachable as it is underneath the Elan chip carrier).

Figure 3 shows the wiring between the 4501 and the GPS. Doing this mod requires some soldering on the Soekris board and will probably void your warranty. It needs to be done using a well-grounded, small soldering iron and a steady hand. The result will be better accuracy of your NTP appliance!

Figure 4 shows the wiring on the back (solder) side of the 4501 board where pin 3 of JP3 is connected to pin 1 of JP11, and the power for the GPS is connected to pins 11 and 2 of JP3.

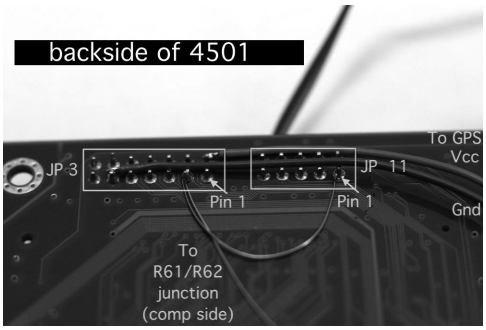

FIGURE 4: ADDITIONAL WIRING ON THE BACK SIDE OF THE 4501 PCB

Figure 5 shows the R61–R62 junction on the component side of the board, which is connected to a wire that runs to JP3 pin 3 on the reverse side of the board. Wiring is done with thin insulated copper wire. You can also use so-called wire-wrap wire for making these modifications.

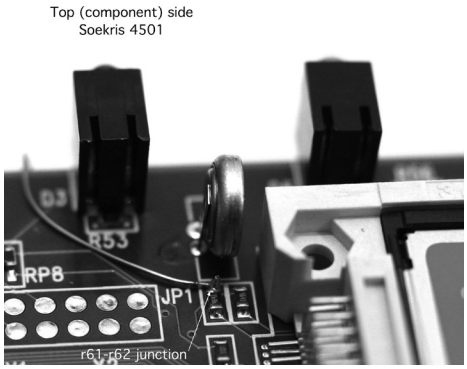

FIGURE 5: WIRING ON TOP (COMPONENT) SIDE OF 4501 PCB

If you happen to be using another board, you will have to input the PPS signal through the serial port, using the DCD line (on a 9-pin connector this is pin 1); it does not hurt to do this on the Soekris as well, so that rewiring the board will be easier. To use the DCD input instead of the GPIO pin (on a non-Elan 520 board) you should *not* be using any special PPS kernel option [8]. In the software config you should not link /dev/mmcr to /dev/pps0 in /etc/devs.conf, and you should leave out the sysctl defining the GPIO pin being used

FLASH CARDS

The complete image that will be generated needs to be put on a CF card. To get the correct geometry of a flash card you can use fdisk on the card (using, e.g., a CF to USB converter) and read the capacity/heads/sectors from here. There have been issues with flash cards in Soekris devices, so please refer

to the Soekris tech mailing list [9] for these discussions. Generally good experiences have been reported with SanDisk devices. The size of the CF card does not matter much—256MB and up will do just fine and will hold two versions of the NanoBSD system easily. Please check the Soekris tech mailing list [9] if you want to use really big flash cards; you might need to upgrade your BIOS to recognize larger (> 2GB) CF cards.

You can test if the Soekris accepts your flash card by installing it in your Soekris, connecting a terminal to it (using a 9-pin female to 9-pin female-female serial null modem cable), and seeing if the Soekris tries to boot from the card.

While you are in the BIOS of the Soekris, it is wise to set the real-time clock to a correct time (UTC). The Soekris factory-default uses mostly 19200 baud, (8 bits, no parity), but your mileage may vary.

SOFTWARE

To generate a NanoBSD image you will need a developers install of FreeBSD on a reasonably fast machine, since you will be (re) building a kernel and all userland utilities, which takes a considerable amount of resources. For example, I used a quad core 2.5 GHz machine with 2 GB of RAM and installed FreeBSD 7.0-RELEASE. Building everything on this box took just over half an hour. You can also take a slower machine or even a virtual instance of FreeBSD if you are ready to wait longer. Be sure that you have enough (> 512 MB) memory.

We need a couple of configuration files in order to generate a CF image. First, we specify what is needed in terms of package sets on the target machine and thus what will be built into the flash image. This file also specifies the kernel configuration file that normally resides in /usr/src/sys/i386/conf and the size and geometry of the compact flash card used.

CONFIGURATION FILES

An example of a NanoBSD configuration file can be found in the NanoBSD how-to section of the FreeBSD handbook [6]. It is too long to be printed here, but the file that was used to build the prototype is available in this issue of *;login:* online. Here I have summarized a number of key items.

timelord.nano:

```
NANO_NAME=NET4501_RVD_1.3
NANO_KERNEL=NET4501_PPS
# to run make in parallel on a multicore machine
NANO_PMAKE="make -j 4"
CONF_BUILD='
NO_NETGRAPH=YES
NO_PAM=YES
'
CONF_INSTALL='
# See Default config in FreeBSD handbook NanoBSD howto [6]
'
CONF_WORLD='
# See Default config in FreeBSD handbook NanoBSD howto [6]
'
# Kingston CF card 512Mbyte
NANO_MEDIASIZE=1000944
```

```
NANO_SECTS=63
NANO_HEADS=16
```

Then we need a kernel definition file that specifies the kernel that is going to be built. Here is the place to specify the special settings for the 4501 Soekris board and the Elan processor. To do this, specify at least the following options, which differ from the GENERIC kernel config file:

/usr/src/sys/i386/conf/4501_PPS:

```
machine     i386
cpu         I486_CPU
ident       NET4501_PPS
# Options Specific to the Soekris NET45XX and PPS
options     CPU_ELAN
options     HZ=1000
options     CPU_SOEKRIS
options     CPU_ELAN_PPS
# Use the high res timer (PPS to be connected to R61/R62)
# More options as in GENERIC kernel to follow.
```

A complete kernel configuration file is available in this issue of *;login:* online.

As a last step we need to add some configuration files that need to end up in the /etc directory on the flash card:

The global rc.conf: Defining the (default) name and IP#

```
# rc.conf
hostname="timelord.xlexit.nl"
ifconfig_sis0="192.168.18.99 netmask 255.255.255.0"
defaultrouter="192.168.18.1"
#
background_fsck="NO"
syslogd_enable="NO"
devd_enable="NO"
cron_enable="NO"
#
sshd_enable="YES"
sendmail_enable="NONE"
#
ntpdate_enable="YES"
ntpd_enable="YES"
ntpd_program="/usr/local/bin/ntpd"
#
```

Of course we need resolv.conf: defining your resolver:

```
#
domain xlexit.nl
nameserver 192.168.18.6
nameserver 192.168.22.1
```

In the sysctl.conf, we define the PPS input on the GPIO line. The "P" shows the GPIO line the PPS signal is connected to. In our case, the Soekris-board GPIO 0 corresponds with the Elan520 PIO 5 line [5, sec. 4.6] (hence the P at position 6).

```
machdep.elan_gpio_config=-----P...--..---------..---------
```

In ntp.conf, we define the clock settings. The time1 fudge factor applied to the PPS discipline defines the offset of the leading edge of the PPS signal to

the "real" start of the second. This is dependent on the GPS and the way the PPS is processed internally in the GPS. Here, I took 100ns. For the NMEA output, I use 150ms if no PPS used. These parameters are GPS-dependent and can be fine-tuned.

```
# local NMEA (20) and PPS (22) discipline
#
server      127.127.22.0 minpoll 4 maxpoll 4
fudge       127.127.22.0 time1 0.0001
server      127.127.20.0 prefer minpoll 4 maxpoll 4
#
server 0.europe.pool.ntp.org
server 1.europe.pool.ntp.org
#
driftfile   /etc/ntp/ntp.drift
# Statistics and logging, use for debugging
statsdir            /etc/ntp/
statistics  clockstats
statistics  rawstats
statistics  loopstats
#
```

/etc/devfs.conf:

```
# Let NTP know to find clocks (serial:COM2, PPS on GPIO)
linkcuad1           gps0
link                elan-mmcr       pps0
```

Copy all those files to the ./files/etc/ directory in the NanoBSD build directory on the development host (usually /usr/src/tools/tools/nanobsd). Now all we need to do is generate a binary package for NTP and add this package to the NanoBSD development directory. We must configure NTP to use reference clock input from both the GPS NMEA and PPS.

Check that the default configuration includes the GPS NMEA and the PPS discipline as reference clock. In the file /usr/ports/ntp/work/ntp/config.h the following three definitions must be present to let the NTP daemon make use of the NMEA output of the GPS and the PPS output:

```
#define CLOCK_NMEA      1
#define CLOCK_ATOM      1
#define HAVE_PPSAPI     1
```

Then make and install the NTP package on the development machine. After that make a binary package of the NTP installation:

```
Make install
Create package –b ntp
mv ntp.tgz /usr/src/tools/nanobsd/Pkg
```

Then start the NanoBSD build:

```
sh nanobsd.sh –C <nanoconfigfile>
```

After the build and some [coffee|tea], you will be faced with a number of files in a directory under /usr/obj/<nanoconfigfilename>. Now the ._diskimage.full file can be put on a compact flash card. You can do that using a (USB or firewire) CF reader and dd (1).

Next you can put the compact flash card in your target device and test the software. Be sure to hook up a terminal (emulator) to the first serial port (the DB 9 connector on the 4501) to see the boot messages coming through with 9600 baud, the default baud rate used with NanoBSD.

Setup and Logging

You should be able to log into the console over the serial connection as root. First, set the root password and make the change permanent (remember, the /etc file system is in RAM) by executing the reset_password script in ~root. Another important task is making the generated ssh keys permanent (thus saving them to the /cfg partition) by running the save_keys script.

After startup of the NTP part of the appliance, it will set the time to start syncing. This is shown in the /var/log/messages files as:

```
May 3 19:58:46 timelord ntpd[536]: time reset +557.475659s
```

To check on the NMEA string the GPS supplies, use `cat /dev/gps0` to see the data that is sent from the GPS device to the appliance. It should at least consist of the $GPRMC, $GPGLL, or $GPGGA sentence. For example:

```
$GPRMC,095639,A,5210.7602,N,00429.7604,E,000.0,142.7,040509,001.0,W*62
```

This shows UTC, Status (A=valid), Latitude, N|S, Longitude, E|W, speed, heading, date, variation, direction of variation, *, and checksum. And, yes, you can plot my house on Google Maps from this…

Now you can check on the NTP software running, by using the `ntpq –p` command. At least two lines as follows should be returned:

Remote	refid	st	t	when	poll	reach	delay	offset	jitter
GPS_NMEA(0)	.GPS.	0 l	10	16	377	0.000	0.002	0.015	
PPS(0)	.PPS.	0 l	3	16	377	0.000	0.002	0.015	

This shows the status of the two time sources, NMEA and PPS. It will take some time to get the offset and jitter numbers down. They show that the PLL is working. If you define more (external) peers in the ntp.conf file, you will see them here as well.

After some time, the ntp daemon will be syncing to the PPS signal. This is noted in the /var/log/messages file as

```
May 3 20:13:45 timelord ntpd[536]: kernel time sync status change 2001
```

Now you can also show the NTP statistics:

```
Ntptime:
ntp_gettime() returns code 0 (OK)
 time cda93d41.64af9204 Mon,May 4 2009 10:09:05.393, (.393304594),
 maximum error 1018 us, estimated error 15 us, TAIoffset 0
ntp_adjtime() returns code 0 (OK)
 modes 0x0 (),
 offset 1.623 us, frequency -6.086 ppm, interval 1 s,
 maximum error 1018 us, estimated error 15 us,
 status 0x2001 (PLL,NANO),
 time constant 4, precision 0.001 us, tolerance 496 ppm,
```

The output above shows that the appliance currently is running at around 1.6 microseconds from real UTC (remember, this data is after the device has been running for about 14 hours). The "PLL,NANO" tells us that the clock PLL is running, using nanosecond timing, which is good. It also shows that the P/F locked loop is imposing a correction of –6 ppm (parts per million) to the (hardware) clock oscillator on the board.

Now you can edit the ntp.conf file to restrict clients connecting to the NTP appliance and add keys, etc. Please refer to the NTP documentation and the ntp.conf manual page on the development box, as you probably did not install man pages on the target machine.

Remember: if you change anything in the /etc directory, and want to make it permanent, mount the /cfg partition, copy over the file to /cfg and unmount /cfg again.

Of course, logging needs to be done properly. Best is to get the ntp-logs (in /etc/ntp) rotated or log externally to a log server, or send them to another machine. Furthermore, there is a lot of tweaking and tuning that can be done in the ntp.conf file. Please refer to the NTP documentation [1] and the man page for the syntax and options of this file. This is left as an exercise to the reader.

Results and Conclusion

You now have a reasonably cheap setup for a Stratum1 NTP server to drive your network with the correct time. The accuracy is comparable to much more expensive devices. Figures 6a and 6b plot the accuracy of the NTP server by showing the error bars on the offset of the clock and the offset itself. You can see that the longer the clock is running, the smaller the offset becomes, the system becoming more stable.

The maximum size of the error bar is approximately 3.5 microseconds, and the maximum actual time error is 5 microseconds.

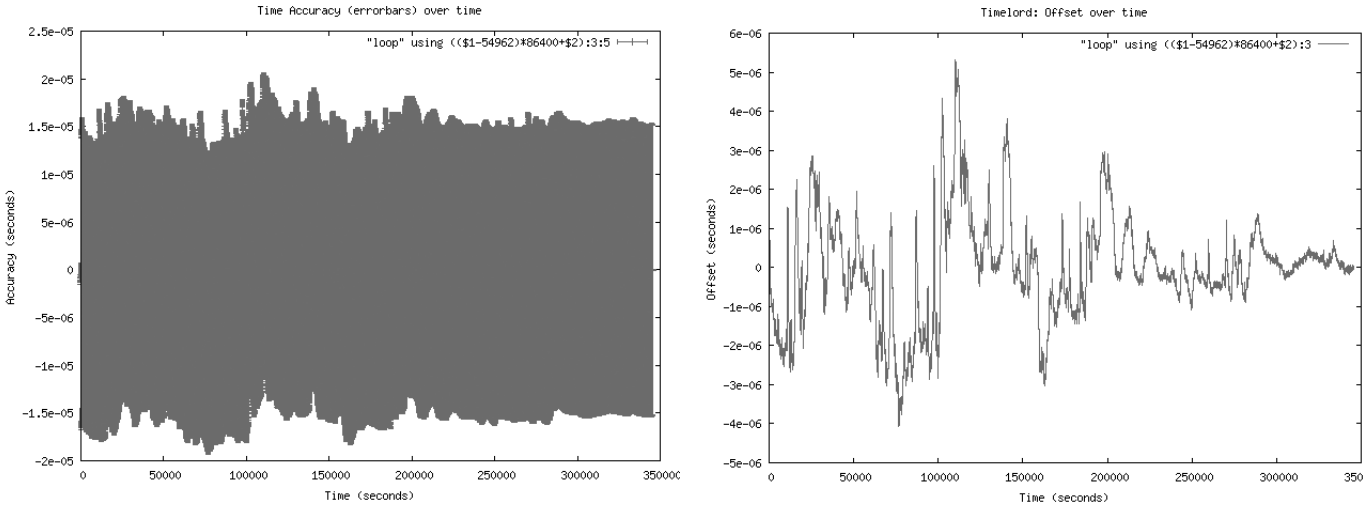

FIGURE 6A, B: ACCURACY (ERRORBARS) AND OFFSET

The graphs are generated using gnuplot [11] to plot values from the /etc/ntp/loopstats file.

You can even add a temperature-controlled crystal (TCXO, crystal oven) and a frequency converter, such as the TAPR clock box [10], and run the 4501 clock from it to get lower clock jitter. This is described in detail by John Ackermann [13]. Right now with the current setup, if you plot the clock correction over time, it shows (see, e.g., Figure 6c) a strong correlation with the ambient temperature.

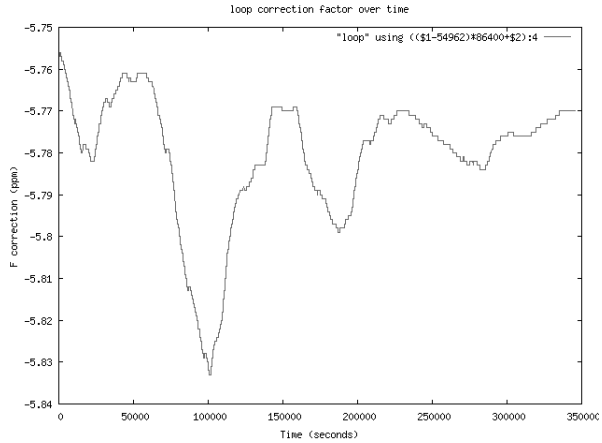

FIGURE 6C: V/F LOOP CORRECTION FACTOR OF PROCESSOR CLOCK OVER TIME

ACKNOWLEDGMENTS

A number of people provided me with excellent resources and information to help me get started and become a bit of a time nut [12], although I have not yet acquired a cesium beam clock.

Ralph Smith helped me with some valuable resources to get this project bootstrapped. Of course, I want to thank Poul-Henning Kamp for the work on ELAN_PPS and sorting out the way to use the TMR1IN, and for the creation of NanoBSD.

I'd like to also thank Rik Farrow for the great comments and help with the article. A last word of thanks goes to Competa IT, my employer, for giving me the freedom to write this article.

REFERENCES

[1] NTP documentation: http://www.ntp.org.

[2] *;login:*, USENIX Association, October 2008: http://www.usenix.org/publications/login/2008-10/pdfs/knowles.pdf.

[3] Poul-Henning Kamp, "Before the Last LORAN-C Receiver": phk.freebsd.dk/loran-c/intro/.

[4] Garmin GPS 18: http://www.garmin.com/manuals/GPS18x_TechnicalSpecifications.pdf.

[5] Soekris 4501 manual: http://www.soekris.com/manuals/net4501_manual.pdf.

[6] NanoBSD handbook: http://www.freebsd.org/doc/en/articles/nanobsd/howto.html.

[7] Poul-Henning Kamp, "Building a FreeBSD Appliance with NanoBSD": http://phk.freebsd.dk/pubs/nanobsd.pdf.

[8] Poul-Henning Kamp communications.

[9] Soekris-tech mailing list: http://lists.soekris.com/mailman/listinfo/soekris-tech.

[10] Tucson Amateur Packet Radio: http://www.tapr.org.

[11] gnuplot: http://www.gnuplot.info.

[12] Quinn Norton, "Amateur Time Hackers Play with Atomic Clocks at Home": http://www.wired.com/science/discoveries/news/2007/12/time_hackers.

[13] John Ackermann's pages: http://www.febo.com.