A Cheap SDR Loran-C frequency receiver

(Formatted Sun Nov 9 00:00:59 UTC 2008)

Poul-Henning Kamp

<phk@FreeBSD.org>
Den Andensidste Viking
Herluf Trollesvej 3
DK-4200 Slagelse
Denmark

ABSTRACT

Loran-C radio-navigation signals are broadcast in most of the northern hemisphere and offer a solid alternative to GPS disciplined frequency standards, but a regrettable lack of affordable receivers means that it sees very little use for this purpose.

This article describes a simple, cheap and efficient Loran-C frequency receiver for such purposes.

The three main components of the receiver is a homebuilt loop antenna, the TAPR/N8UR "ClockBlock" and the OliMex.com ADUC-P7026 microcontroller prototype card, for a total cost well under $200.

1. Introduction

LORAN-C signals, like GPS signals, offer wireless access to very stable and calibrated, and thus expensive, frequency sources and can therefore be used to steer cheaper oscillators to correct frequency.

Despite stern warnings to the contrary, the world today is more or less entirely frequency locked to the GPS satellites because that is the most convenient and cheap technology available.

This little project attempts to show that an equally cheap solution can be made based on the Loran-C signals, which are broadcast over most of the northeren hemishere.

Compared to GPS, Loran-C signals are just about as different as can be, where GPS satellits transmit their signals with only 50W using 1.2 GHz microwaves from 20,000 km above the Earth, Loran-C transmits are 200 meter tall steel towers which transmit several hundred kW at a frequency of 100 kHz.

This makes Loran-C the perfect backup for GPS, since there are almost no overlap between threads to the two systems.

This paper documents what is clearly a "work in progress", and as time and energy permits, I will update both this paper and the source-code that goes with it.

Poul-Henning Kamp

2. Hardware

The entire reason for this article is that a new breed of microcontroller combines a real 32bit CPU with fast analog/digital converters on a single chip.

Amongst these chips my fancy took to Analog Devices ADuC7026.

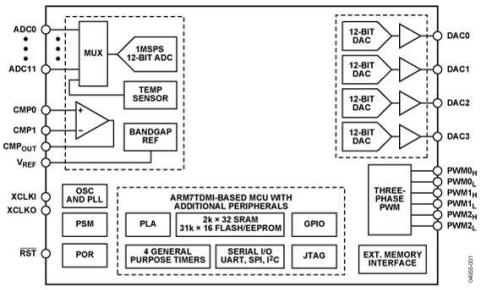

Figure 1. ADuC7026 block diagram

For one thing, Analog Devices are experts at the analog/digital border, and the specifications on the ADC in is much tighter than on on competing chips. Furthermore the ADC offers a very convenient fully differential input mode and is not picky about the ADC reference voltage.

There are also some downsides: The chip needs a 42 MHz clock frequency to run the ADC at 1 megasample per second, and it only has 8kilobytes of RAM.

A very interesting feature is the built in "programmable logic array", which makes it possible to totally avoid external glue-logic in our case.

The main market for the ADuC7026 seems to be the variable frequency motor control market, and certain onchip facilities are less than well documented, for instance the internal timers barely have any connection to the outside.

As it transpires, there is a way to hook things up in a way we can use.

2.1. The 42MHz clock frequency

The 42MHz clock frequency is much less of a problem than it sounds, thanks to John Ackermann, aka N8UR we can simply pick up a "ClockBlock" from TAPR.org, set the jumpers correctly and get 42MHz out of almost any input frequency we care to use.

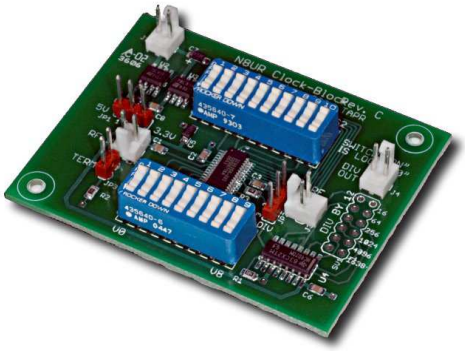

Figure 2. TAPR.org *ClockBlock* card

2.2. The ADuC7026 microcontroller

Being mostly of the software persuation, I find the prospect of soldering 80 pin SMD packages something to be avoid if reasonably possible.

Fortunately my favourite Bulgarian electronics pusher, OliMex.com, has an ADUC-P7026 prototype card which is perfect for this project.

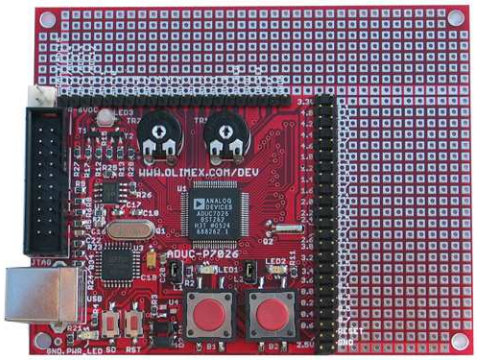

Figure 3. Olimex.com *ADUC-P7026* card

2.3. The Antenna

I do not really consider the antenna part of the project, because I reused a very simple loop antenna I built for a previous round of Loran-C experiments, based loosely on a diagram I found on the website VLF.it using the AD797 operational amplifier.

Figure 4. $20 homebrew loop antenna

It goes without saying that I have not spent a lot of time on antennas, but roughly speaking, the requirements for the antenna input signal are something like:

- □ A Relatively flat passband from at least 70 to 130 kHz.
- □ Not too much signal above 300-400 kHz.
- □ At least 100mV peak-to-peak Loran-C signal.
- □ No more than 3V peak-to-peak total output signal.

The downfeed from my antenna is coax cable, and therefore I use a small balun transformer, in my case and old T1/E1 telecom balun, to convert from single-ended to differential.

The midpoint of the secondary is held to ½ A_{Vdd} potential by a voltage divider consisting of two 470Ω resistors.

2.4. Connecting it all

That is really all for the hardware, now we just need to connect it together:

- □ Jumper the ClockBlock for 3.3V supply.
- □ Connect the ClockBlock output to pin P0.7 on the ADUC-P7026.
- □ Connect your chosen frequency standard to the ClockBlock.
- □ Connect pin P2.7 ($PWM1_L$) to P0.6 (*T1*) on the ADUC-P7026 to establish a connection from the PWM output to the Timer1 input.
- □ Connect pin XXX to XXX on the ADUC-P7026 to establish a connection from DACXXX output to the ADC's VREF input.
- □ Connect the differential antenna signal to pin ADC1 and ADC2, remember that they both must stay between A_{gnd} and A_{vdd} at all times.

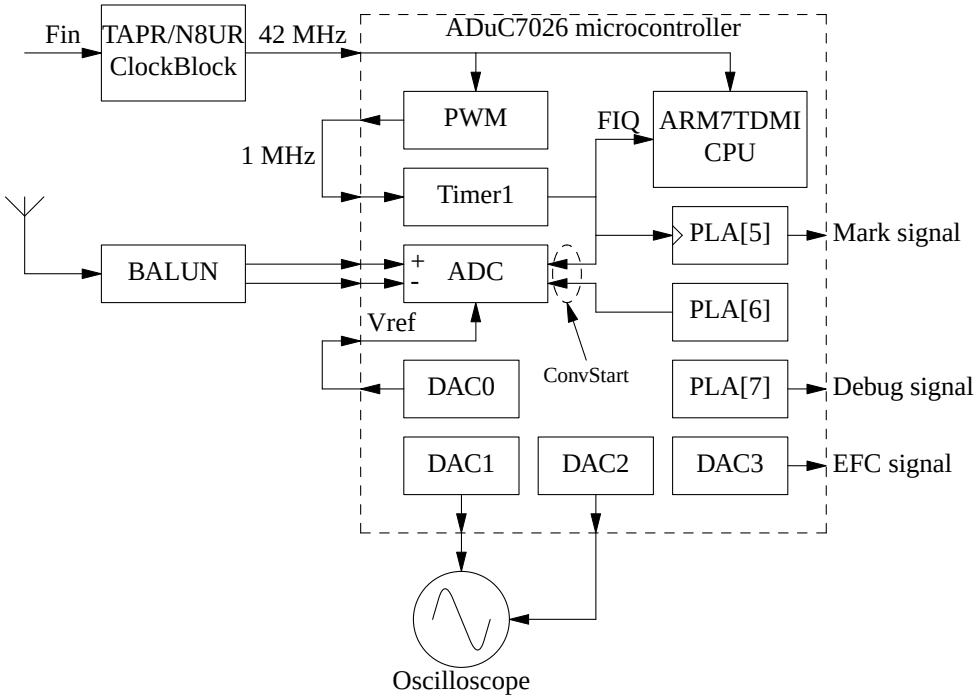

Figure 5. Blockdiagram of the receiver hardware.

- □ Find a suitable power supply for both the cards, it should be at least 6VDC to avoid drawing power from the USB port, and needs to supply about 100mA total.
- □ Connect your computer to the USB port on the ADUC-P7206 card.

2.5. Available output signals

The following output signals are available for regulation and debug use.

- □ P2.1 (*PLAO[6]*) "Mark" output. The raising edge marks the start of measurements in the A-code GRI interval. The software reports how much later than this flank the 3rd zero crossing is found.
- □ P2.2 (*PLAO[7]*) "Debug" output. This pin flips low whenever and as long as the fast interrupt routine runs.
- □ P0.5 (ADC_{busy}) This signal is logic high while the ADC is performing a conversion
- □ DACXXX Analog replica of the averaged signal. Show this on your oscilloscope, use the "mark" output as timebase trigger.

3. Loran-C signals

A Loran-C navigation chain consists of a master transmitter and from one to five slave transmitters, all broadcasting on the same "GRI", Group Repetition Interval.

A Physical transmitting station can participate in multiple chains, although for reasons of time-contention, typically it will only participate in two chains.

In our case, we are interested only in a single transmitter in a single chain, typically the nearest and strongest signal at the location of reception, although there is nothing preventing this construction from being used for more ambitious reception projects.

3.1. What the station transmits

There are many ways to describe the Loran-C signal, but for our analysis of how and why this receiver works, we will treat it as three components which are convoluted to give the actual on-air signal.

3.1.1. The GRI periodic function

This is basically a pulse generator which emits a pulse every GRI * 10 μseconds, so that for instance the Sylt chain with GRI 7499 has a period of 0.07499 seconds.

A very important feature of the GRIs used, is that they do not result in periods that are integral multiples of 1 msecond, so signals CW radio transmitters will average out, allowing us to use essentially no other frequency domain filtering.

The easiest way to find the signal inside the GRI window, would be to average all ADC samples in the GRI period, until we can spot the signal. With our chosen sample rate of 1MSPS, this would require 390 kilobytes of RAM and the ADuC7026 has only 8 kilobytes.

3.1.2. The signal code function

This is where much of the noise resistance of the Loran-C signal structure comes from: in alternating GRI periods two different pulse-trains are used for each of Master and Slave stations.

Whenever the GRI periodic function triggers a transmission eight or nine pulses will be transmitted with 1 msec interval, with polarity according to the following two figures.

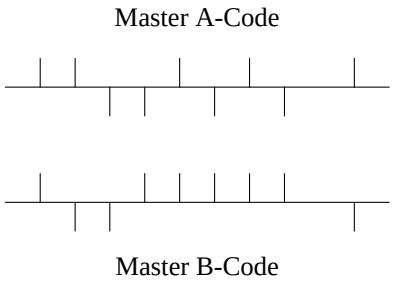

Figure 6. Master Signal Codes

In addition to the frequency filtering provided by the GRI averaging, the non-periodic nature of these codes provide significant improvements in S/N ratio.

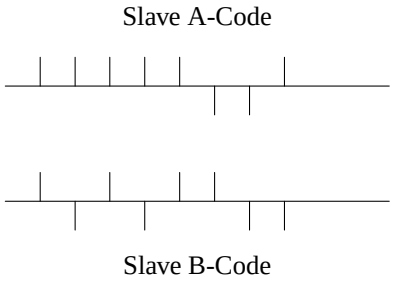

Figure 7. Slave Signal Codes

One important feature of both codes is that if the A and B interval codes are summed up, the odd numbered pulses cancel out leaving only four signals, spaced 2 msec apart.

3.1.3. The Radio Frequency Pulse function

The final step of the modulation is that each pulse is transmitted as

$$f(t) = \sin\left(\frac{2t\pi}{10}\right) t^2 \exp^{-2t/65}$$

where t has units of μseconds.

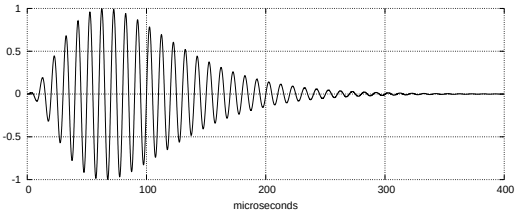

Figure 8. Loran pulse

In reality it is only the first part of the pulse-shape which is controlled, and in particular, the reference point is the 3rd positive zero-crossing of the signal

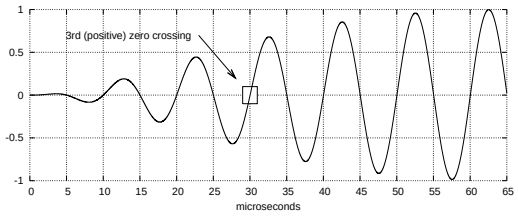

Figure 9. Loran pulse front

When we combine all these components, the result is something like this:

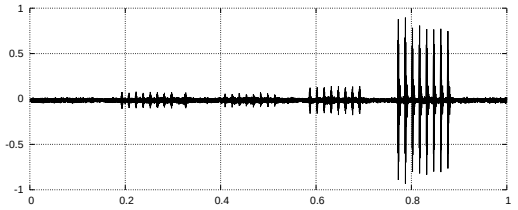

Figure 10. 6731 Lessay chain

All four stations in the Lessay chain are visible in this plot, first the master in Lessay, notice the "extra" 9th pulse 2 msec after the 8th, the follows the Xray slave in Soustons, the Yankee slave in Rugby and finally the Zulu slave on the island of Sylt.

Knowing the exact geographical locations of these four transmitters, and the time interval from the master transmission until the slaves transmit their pulses, we could calculate our geographic position from the measured time-intervals between these signals.

4. Reception

So, how do we find and lock onto the Loran-C signal with only 8 kilobytes of RAM ?

4.1. Locating the strongest signal

Our first task is to locate the strongest signal in the chain, typically the transmitter closest to the antenna.

Experiments and simulations has shown that if we average every 114th sample over the GRI period, we can expect to catch at least 30% of the peak amplitude of the strongest signal in one of the buckets.

It is important to note that numbers which are divisible by 5 would not work here, as they would risk the 100 kHz periods at the same point every half-cycle, and therefore just might end up sampling all the zero-crossings.

The longest GRI in use is 9990 in the North Pacific chain so that requires 876 integers of 4 bytes, less than half of what we have available.

The downside to this, is that we know where one of the four peaks of the GRI-summed code is located in the GRI interval, but we do not know which of the four it is, we fix that in the next state.

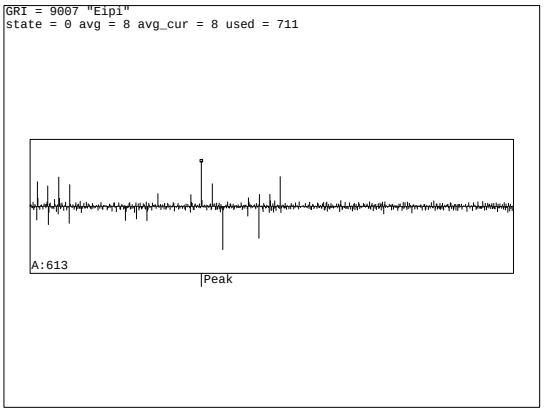

Figure 11. Tek4014 view of state 0

This is how the tek4014 plot looks in state zero, we see the averaging buckets plotted, and the peak signal is marked.

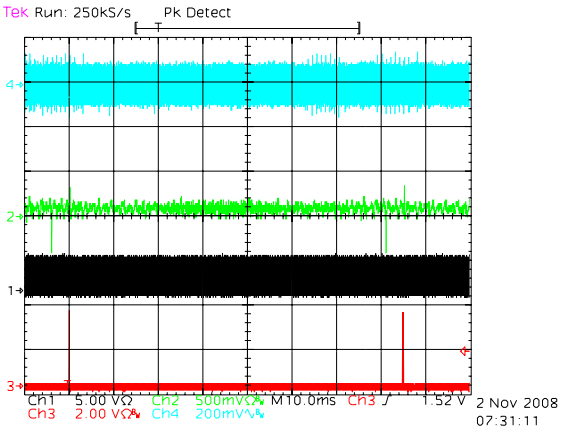

Figure 12. State 0 GRI wide signals

This is what the signals look like on the oscilloscope.

Trace number 3 (red), is the marker signal which triggers the oscilloscope at the GRI rate.

Trace number 1 (black), shows the ADC_{busy} signal and we can see that samples are continuously being taken through out the GRI interval.

Trace number 2 (green), is the averaged bucket output and the peaks are quite visible here also.

Trace number 4 (cyan) is the raw antenna signal.

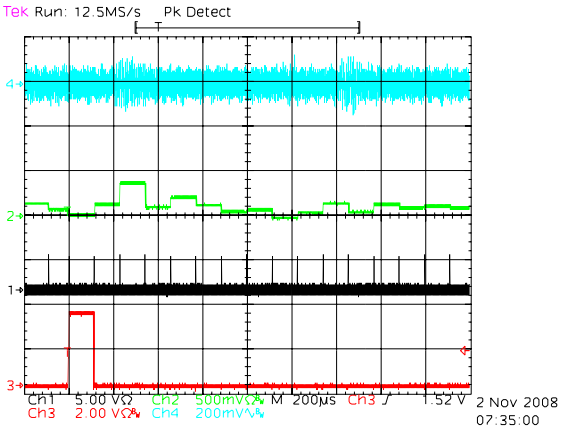

Figure 13. State 0 zoomed signals.

Here we soom in on the first part of the GRI period, and we can clearly see the individual ADC samples.

4.2. Finding the code

The peak signal from phase 1 is within 57 samples, of the true peak value of one of the four peaks, 2 msec apart, resulting from the GRI

summing.

Now we need to find out if this is a master or slave station, and to lock onto either the A or B code.

If we caught the last of the four peaks, the code starts 6 msec ahead of this point, if we caught the first of the four peaks, it spans another 9 msec after this point, so all in all, there are 16 spots we need to check for the precense of pulses.

We have to use a repetition frequency of 2 * GRI, sometimes called "FRI" for Frame Repetition Rate, in order to not add the A and B codes together now.

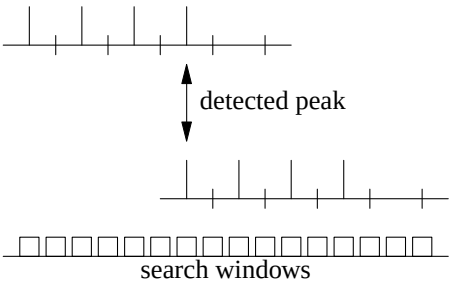

Figure 14. mode1 code windows

Using no more than the 876 integers used in phase zero, each bucket can get 54 integers and if we space them 13 samples apart, each bucket will cover 702 μseconds.

Again, 13 is a good number because it will make the sample points "wander" over the 100 kHz period of the pulses.

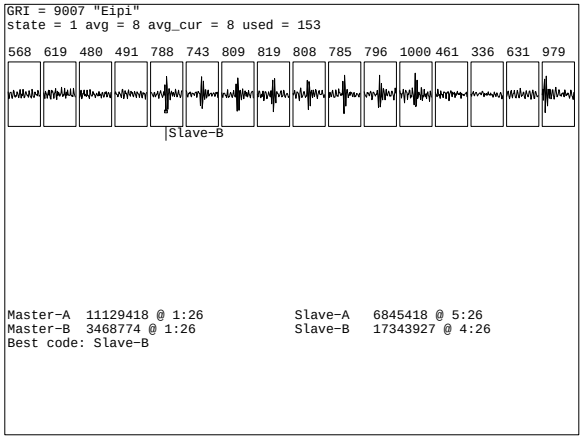

Figure 15. Tek4014 view of state 1

As shown on the tek4014 screen, we try to find the strongest signal for the four possible codes and use that as input to phase 2.

Above each of the 16 buckets are shown the signal energy in parts of thousand relative to the strongest bucket.

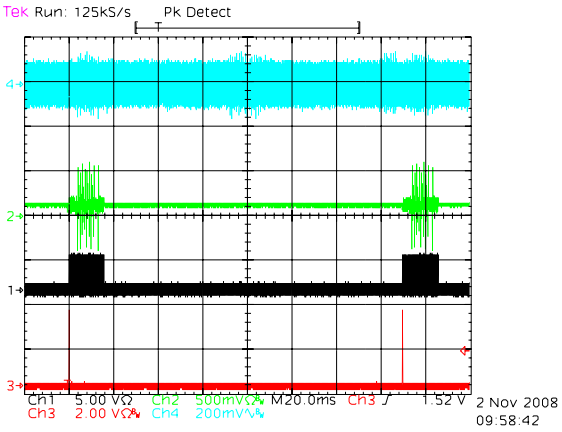

Figure 16. mode1a

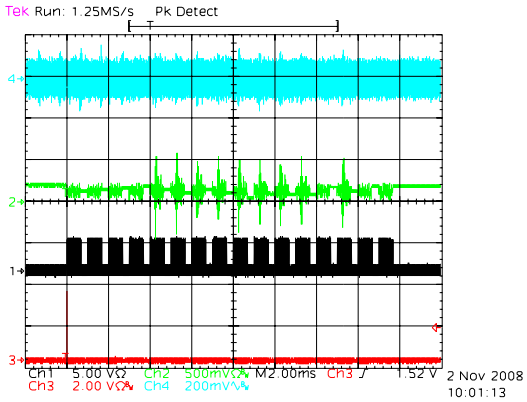

Figure 17. mode1b

4.3. Locking on to the signal

Now we have all the timing informaton we need to start integrating the signal for good: we know where the maximum is of the first pulse in the code, and we know which code it is.

In this phase we integrate every ADC conversion in a 750 sample window around the pulses.

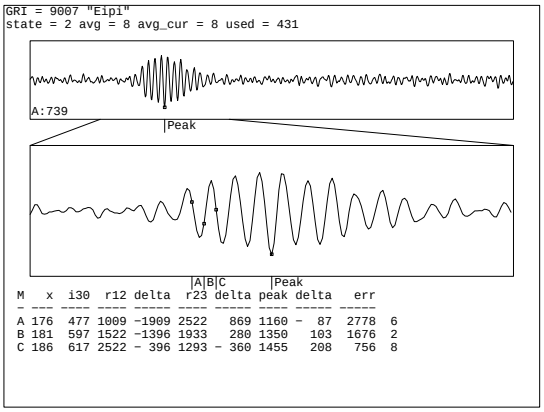

Figure 18. Tek4014 view of state 2

The tek4014 screen shows two views of our integration buffer, with the bottom one zoomed in on the peak value.

The various candidates for 3rd zero crossings are marked and their statistics given in numeric form below.

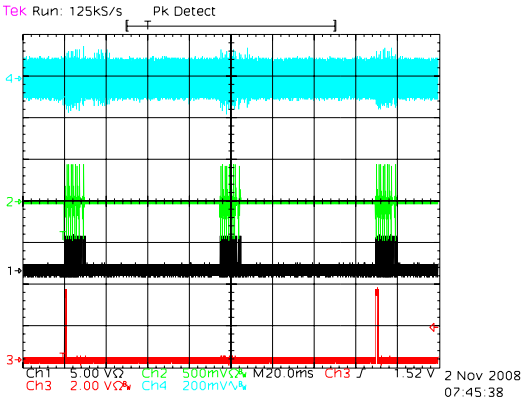

Figure 19. mode2a

The MARK pulses are spaced 2 * GRI apart now, and we can see how the signal is sampled twice in that period.

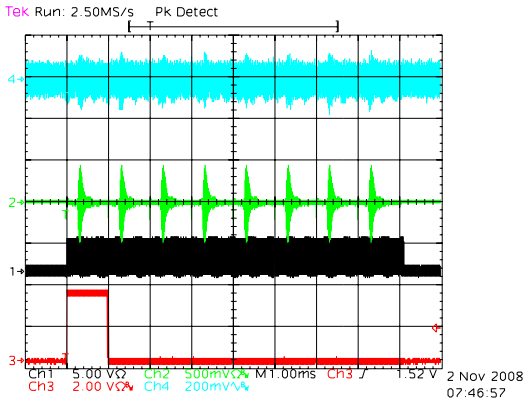

Figure 20. mode2b

When we zoom in, we can see how each of the pulses are sampled into the same buffer, but with the polarity specified by the chosen code.

4.4. 3rd zero crossing

In state 3 we have chosen the zero-crossing we want to track, and hold on to it to the best of our ability.

In practice we cannot simply nail a particular sample in the GRI window and track that, as the signal and the timebase may wander relative to each other.

Instead we move the chosen sample number earlier or later if the signal moves enough to give them the same sign, and we keep track of these steps using a counter.

If eventually, the tracking point wanders out of the sample window, we will have to move the sample point to put it back inside, but this is not yet implemented.

We interpolate the true zero crossing between the chosen point and the next, using a simple linear model.

Earlier experiments of mine have shown that it is possible to interpolate against the ideal Loran-C wave shape, and this may be implemented later, if precision warrants it.

This is the Tek4014 display in state three:

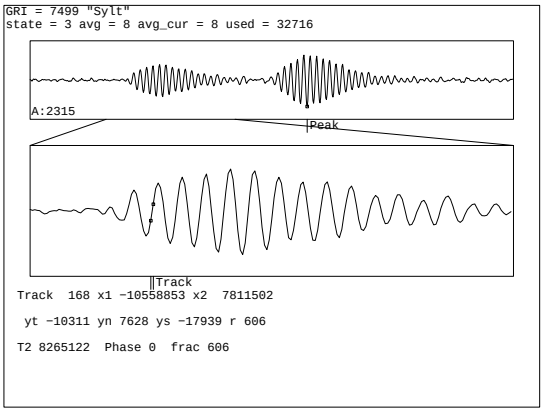

Figure 21. Tek4014 view of state 3

The top plot shows the collected and integrated data and the bottom plot zooms in around the chosen zero crossing, with the tracking sample and its successor marked.

Notice that the sky wave signal is stronger than the ground wave signal.

5. Performance

These are the first performance data collected, and they should therefore be taken with all sensible precaution.

The signal is 7499M, 205 km away but received with my loop antenna oriented to attenuate that transmitter to approximately the same level as 9007M, 1294 km away.

The exponential averaging factor is 1/256.

The timebase is a PRS10 Rb steed by a Motorola Oncore UT+.

A phase measurement is recorded once per second using Timer2, from approx 21:00 to 23:00 UTC, 2008-11-08.

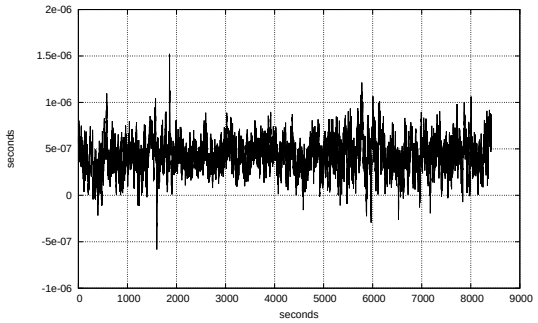

Figure 22. Raw offset samples

The standardeviation of the raw samples is 185 nanoseconds, and their linear trend is $4.6 \cdot 10^{-12}$ second per second.

A histogram of the samples show a gaussian distribution, which seems well suited to more aggresive filtering:

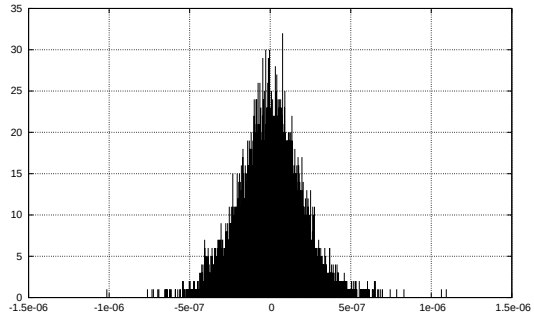

Figure 23. Offset value interval histogram

The modified allan variance is plotted below, with guide lines corresponding to the functions $\frac{185 \cdot 10^{-9}}{\tau}$ and $\frac{185 \cdot 10^{-9}}{\tau^{1.5}}$

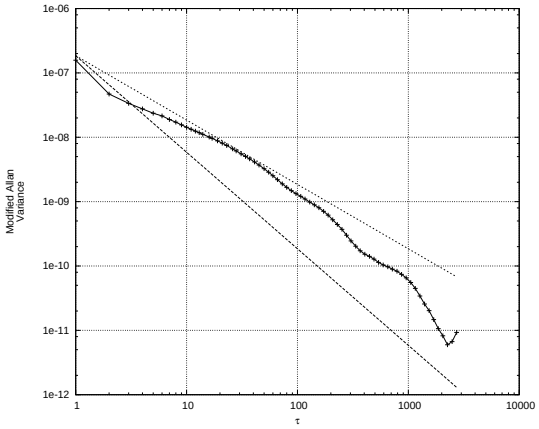

Figure 24. Modified allan Variance

6. Software

The software that goes with this article is writte as proof of concept, but is written such that experimentation and further development is possible.

6.1. The tricky assembler bits

The tricky bit of the software is the assembler coded fast interrupt routine in the file *crt0.S*.

I have written this code so that it does not encode any of the high-level logic, but instead mechanically walks a chain of structures that tell it what to measure and when.

For instance, when we scan the 6731 GRI in phase zero, the specification looks like this:

```
{
    mark         = 1;
    ptr          = ss->ptr;
    polarity     = "+";
    avg          = 6;
    cnt          = 1;
    timer_repeat = 114;
    repeat       = 590;
    timer_after  = 50;
    next         = this;
}
```

Figure 25. Phase zero specification

This specification tells the assembler code to pulse the MARK pin high when starting this specification, then 590 times take one ADC measurement 114 microseconds apart, then skip 50 microseconds and start over.

Since 590 * 114 + 50 = 67310 this will repeatedly scan the 6731 GRI signals.

The *polarity* member specifies that the samples all have positive phase, the *avg* specifies he exponential average time constant and the *next* member tells it what to do next (the same thing).

With none of few modifications, this scheme should also support reception of multiple stations in the same chain or even several stations from different chains.

6.2. The high level logic

The highlevel logic, such as it is, is written in C code in the *loran0.c* file.

Right now the code is semi-manual, controlled by single characters received on the serial/USB port, running at 115200 bps.

The following commands are recognized:

c This will present a menu with Loran-C chains from which you can choose one by entering the character in the [...].

```
---------------------------------
[@]  5543  Calcutta
[A]  5930  Canadian East Coast
[B]  5980  Russian-American
[C]  5990  Canadian West Coast
[D]  6042  Bombay
[E]  6731  Lessay
[F]  6780  China South Sea
[G]  7001  Bø
[H]  7030  Saudi Arabia South
[I]  7270  Newfoundland East Coast
[J]  7430  China North Sea
[K]  7499  Sylt
[L]  7950  Eastern Russia Chayka
[M]  7960  Gulf of Alaska
[N]  7980  Southeast U.S.
[O]  7990  Mediterranean Sea
[P]  8000  Western Russia
[Q]  8290  North Central U.S.
[R]  8390  China East Sea
[S]  8830  Saudi Arabia North
[T]  8930  North West Pacific
[U]  8970  Great Lakes
[V]  9007  Eiði
[W]  9610  South Central U.S.
[X]  9930  East Asia
[Y]  9940  U.S. West Coast
[Z]  9960  Northeast US
[[]  9990  North Pacific
---------------------------------
Please select chain:
```

Figure 26. Chain selection menu

t Make TEK4014 plot via serial/USB port

u Zoom in on curves in TEK4014 plots

d Zoom out on curves in TEK4014 plots

a Double the exponential average time constant.

b Half the exponential average time constant.

1 Skip to mode 1

2 Skip to mode 2

6.3. TEK4014 plotting

Before windows and mice, Tektronix produced a series of high quality graphical terminals, based on the storage-CRT principle.

These plotting instructions to these terminals became the de-facto format for plotting files and as a result, the original X11 "xterm" program offered, and still does, TEK4014 emulation.

If you connect to the serial/USB port using the X.org xterm program and press ’t’, then you will be rewarded with a nice plot as seen earlier in this paper.

If you use another terminal program which does not support TEK4014 plotting commands, you will get a blast of random ASCII characters.

6.4. Utility functions etc.

The other three C language files contains the code to configure the hardware, code to use the serial/USB port and a function which calculates

basic statistics for an array of integers.

The files divdi3.c, qdivrem.c and divsi3.S, are runtime support files for the GCC compiler.

6.5. Downloading firmware to the ADuC7026

You can either use a JTAG gadget or download the firmware via the serial/USB port.

Included in the source-code package is an "aduc" program I have written for that purpose.

6.6. Compiling the firmware

Cross-compiling is notoriously tricky, but it shouldn’t be too hard in this case.

My compile-environment consists of a FreeBSD-8-Current system where I (ab)use the FreeBSD cross-compilation facilities by reaching into the FreeBSD source-tree for the necssary compiler tools.

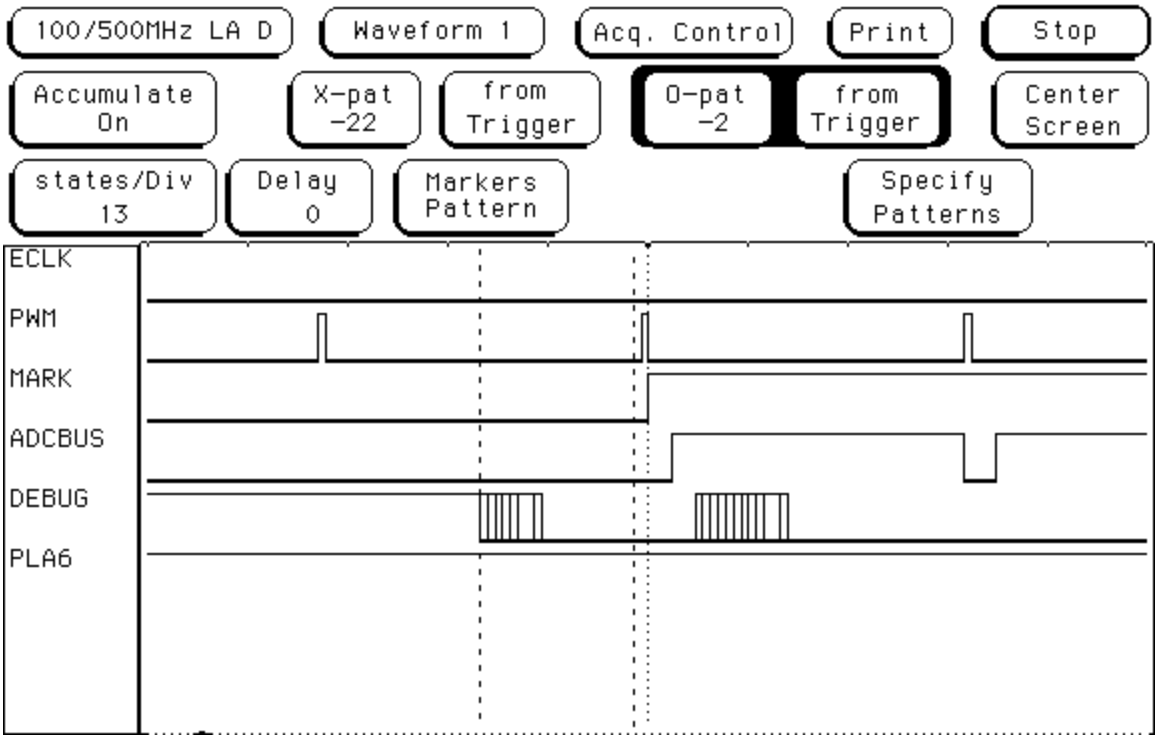

Figure 27. Protocol analyser trace

7. Timer1 unexplained

This is a confession: I have no idea exactly how Timer1 is supposed to work in the ADuC7026 chip, and the way I have observed it work baffles me, but the software manages to work around it.

The PWM section provides us with a 1MHz frequency which, through an external pin, becomes Timer1's clock signal.

In Fig XXX above this external signal is named PWM, which for this experiment, was give a very short "on" time.

The signal named "MARK" is from PLA[5], which is configured so the flip-flop is latched by Timer1's count-complete signal.

The signal named "ADCBUS" is the ADCbusy signal, which indicates that the ADC has started performing a conversion, and the ADC is also triggered by Timer1's count-complete signal.

So far, so good, it looks like Timer1 expires around the middle of the figure.

But now look at the "DEBUG" signal, which is programatically lowered as the first thing in the fast interrupt handler, then pulsed high after Timer1 has been reloaded.

The fast interrupt is *also* triggered by Timer1's count-complete signal, but this happens well before PLA[5] and the ADC receive the signal.

From the FIQ request is raised until the first instruction of the handler is executed takes at least five clock cycles, but it can take as much as 13 (XXX ?) cycles, depending which instruction the CPU was executing.

This is why the "DEBUG" trace shows multiple transititions: it is not deterministic in time.

But he mystery gets deeper here, because the previous PWM signal, presumably the one that counds down to one, is a fair bit further ahead of the DEBUG signal than 5-13 (XXX) clock cycles.

But it gets more mysterious still: The fast interrupt handler loads a value into the T1LD register right before the high pulse on the DEBUG signal (right of the dotted line).

If the leftmost PWM signal is the 1 count, then the central one is the 0 count, it would be reasonable to expect that loading N or N-1 into the T1LD register at this point, would cause the next Timer1 count complete event to happen N cycles

of the PWM signal later, starting with the one in the right hand side of the trace.

Experiementally I have found that I have to load N-3 to make that happen.

For this to make sense, the PWM edge that triggered the count down to zero must be just outside the papers left edge so that the central PWM pulse is number 3 in the new counting cycle.

If that is the case, then the chip applies approximately 3μs worth of metastability latching on Timer1s count-complete output, before it reaches the ARM7TDMI core and the ADC unit.

At the 42MHz clock, 3μ correspond pretty precisely to 128 clock cycles.

If that is the depth of the latch-line that resolves metastability, then I would pressume to think that it is sufficient deep.

Enquiring minds wants to know...