Draft 15-04-11

DSP-based DCF77 time-code receiver

Design Targets

To extract the highest possible accuracy from the German DCF 77.5kHz time-code broadcast this project uses DSP algorithms (running on a low cost dsPIC33 micro-controller) to filter and demodulate both the AM and phase modulated signals, it also produces a 10Hz carrier-locked reference clock output. While commercially available receivers / demodulators (e.g. from Galleon Systems or Conrad electronics) work well and produce reliable time-code pulses, the exact timing of these pulses (with respect to a stable 1Hz clock) have large amounts of jitter - in the order of tens of ms. The reason for this is that they all rely on crystal filters to extract the carrier frequency, the ultra-low bandwidth of these filters is ideal for the job but brings with it inherent timing issues for the demodulation stage. These low-cost commercial receivers also do not provide a carrier-locked reference frequency output nor do they decode DCF's pseudo-random phase modulation scheme which can provide an order of magnitude improvement in timing accuracy and operate much more robustly on low signal strengths. With good quality, low jitter second markers and a carrier-locked reference frequency it is possible to build a clock with sub-millisecond accuracy.

Hardware

The system consists of a receiver board and an active antenna board which can be remotely located at the end of a length of coax cable - proper siting of the antenna way from interference sources can greatly improve signal quality. The antenna is a commercially available ferrite coil and tuning capacitor assembly pre-tuned to 77.5kHz, this is combined with an op-amp gain stage to form the line-powered active antenna capable of driving a reasonable length of coax cable to the receiver board.

Looking now at the receiver schematic, the first op-amp stage provides some more AC gain which drives into three further stages comprising a 6th order low-pass anti-aliasing filter with 1dB flat pass-band to 78KHz and -50dB stop-band from 232.5kHz. Note that we subsequently sample the signal at 310kHz (Fs = 4xFc) and implement a DSP notch filter at 77.5kHz so the first potential alias we need worry about is at 232.5kHz. The analog filter was designed using Microchip's free FilterLab software and should be constructed with 1% components throughout. A fifth op-amp stage provides additional programmable AC voltage gain. All the op-amp stages have unity gain at DC and so the mid-rail bias voltage applied at the first stage simply propagates through to the last. The output of the fifth stage feeds directly into an ADC pin on the dsPIC33 where the signal is sampled at a rate of 310kS/s. Note that the gain of the last stage of the circuit is optimised by software control to provide a roughly 3V pk-to-pk signal for the ADC. After demodulation by software, see below, the MSF time-code signal is buffered by an inverting, open-collector output driver. The programmable gain stage makes use of an H11F1 opto-coupled bi-lateral FET - effectively a current-controlled isolated variable resistor - in the feedback path of an op-amp gain stage. The higher the current through the LED the lower the resistance of the FET and the higher the gain of the op-amp stage. The H11F1's LED is driven by a voltage to current converter (implemented with a spare op-amp and a PNP transistor) driven by a filtered PWM output from the PIC enabling software control of the overall RF gain. To provide the carrier frequency-locked clock source the master crystal oscillator circuit of the PIC is implemented with a voltage-controlled crystal oscillator to allow its precise tuning under software control. This is done using another RC filtered PWM output from the PIC.

I/O Connectors

<u>J1 Antenna header 3-pin</u>

J1.1	GND
J1.2	RF signal
J1.3	GND

<u>J2 Power and output signals header 8-pin</u>

J2.1	5V in, about 100mA
J2.2	TEST output toggles whenever the main loop exceeds its allocated time
J2.3	The VCXO PWM signal to the master oscillator
J2.4	n/u
J2.5	9600 baud serial out (TTL polarity, not RS232 polarity)
J2.6	10Hz reference output
J2.7	DCF pulse output, clean & accurate leading negative edge
J2.8	GND

Note all outputs are open collector if a 74LS06 buffer is used at U3.
Replace U3 with a 74HCT04 if totem pole outputs are preferred.

<u>J3 PIC in-system programming header 5-pin</u>

J3.1	MCLR
J3.2	3v3
J3.3	GND
J3.4	ISP Data / GPS reference pulse input
J3.5	ISP Clock

Software

The dsPIC33 provides a 40MIPs 16-bit DSP-capable core with RAM, FLASH and a host of peripherals, most relevant of which is the 500KS/s 12-bit ADC used to sample our RF signal at 310kHz. The brilliantly designed ADC sub-system has a built-in buffer that can hold 2 pages of up to 8 samples and flips from one to the other automatically setting a flag to tell the software when new samples are ready. The software processes 4 samples at a time in an endless loop that must complete each pass in less than 13us - the cycle time of the carrier frequency. In fact 40MIPS is not adequate for this application and we have to over-clock the PIC to obtain enough CPU power.

After obtaining our four samples (one cycle's worth) the first job is to cross-correlate (multiply) them with a sine and a cosine waveform of the frequency we want to extract, Fc. In the digital world that simply consists of {0,1,0,-1} and {1,0,-1,0} respectively, the multiplications are trivial. Each of the four sin results are added together, as are each of the four cos results, these totals then update two 120 entry ring buffers and a running total of each buffer is maintained. These running totals can be vector added using sqrt(sin^2 + cos^2) to produce an amplitude result. The amplitude signal is filtered and buffered over a two second period to obtain maximum and minimum values from which are computed an upper and a lower threshold with which to demodulate the amplitude signal into binary. This binary signal is of course the raw time code signal and could be used as input to a suitable clock - but we can do much better, more of which later.

In order to get this far though there are a few more things to consider, firstly the matter of the AGC. The software checks the four raw ADC samples to see if any are approaching either the upper or lower limit of the ADC's input range, a simple control loop strives to keep just a handful of samples at the outer limits of the ADC's range. If none or too few samples are pushing the limits the gain control is increased if too many, the gain is decreased. The RF gain is programmable in hardware and is controlled by a PWM output from the PIC, the software simply changes the PWM register to obtain a corresponding change in RF gain.

Next is the matter of fine tuning the master oscillator, this is required in order to produce the phase-locked reference output and to make decoding the phase modulated signal much easier. Given the sin and cos cross correlation data it's a simple matter to calculate the relative phase of the signal with respect to the sample rate, once you have a measure of the phase you can construct a phase-locked loop. Either the sin or the cos data, divided by the amplitude is a measure of phase, this is used both directly and via a software filter to control another PWM output from the PIC. After a little RC filtering in hardware this is fed to the voltage control input of the crystal oscillator, closing the control loop. With the phase-locked loop in operation the master oscillator becomes locked to a multiple of the carrier frequency and from this can be derived a 10Hz reference output. The software must also detect the state of lock - should the loop go out of lock the 10Hz signal is squelched until lock has once more been achieved.

Finally we are in a position to consider decoding DCF's pseudo-random sequence phase modulation. This is a sequence of 512 bits, each bit lasting 120 carrier cycles, used to modulate the phase of the carrier by +/- 13 degrees. The modulation starts 200ms after the leading edge of the AM time code, i.e. the second marker, and continues for approximately 793ms. There are as many zeros as there are ones in the sequence and so the overall phase of the carrier is unaffected. The software already has a measure of carrier phase (stabilised by the action of the PLL) and you now see why a buffer length of 120 cycles was chosen above. The code implements another cross-correlator, a reference bit sequence (stored in code memory) is multiplied by the carrier phase measurement every 120 cycles and summed over 512 bits. The result is a measure of the correlation between reference and transmitted bit sequences and depends strongly on how well aligned the two sequences are.

If we can get the reference sequences optimally aligned then we have a strong handle on the second marker timing. In fact DCF encodes one bit of time-code data on the bit sequence using either the true or complimented sense of the sequence accordingly. Consequently we get either a positive or negative correlation result depending on the data bit encoded. But how do we achieve good alignment - in a perfect world, with unlimited CPU power, we'd store all the phase data we get over the one second period (77500 samples) and hunt for the peak correlation by re-sampling against our reference at different starting points within the data. In the real world we have only enough CPU power to do one set of correlations per second, so we have to pick a starting point and run with it. We need some way of knowing at the end whether we started too early or too late so that we can make an adjustment ready for the next second.

To obtain this information we use yet another cross correlator running half a bit-time out of phase with the first and using a differential bit sequence derived from the main sequence. The result is a signal that (when corrected for encoded data polarity) is zero when the timing is perfectly aligned, negative when the starting point was too early and positive when the starting point was too late. We use this to constantly adjust the next starting point and to home in on the correct alignment, once good alignment is achieved we can start outputting second marker pulses with very much more accurate leading edges. One problem is choosing an initial starting point for the cross correlator, the correlation results explained above only work as long as we are within +/- a bit time i.e. 120 carrier cycles or 1.5ms of the optimum. To obtain our initial starting point we have to rely on the AM signal - the software looks for a clean negative edge and uses that as a reference point, under good signal conditions that's usually ok but if the phase decoder fails to lock it has to go back for another try. However once the phase decoder is in lock it's very robust and can survive periods of signal strength so low that the AM decoder fails to produce useful data.

Serial data output

The software makes the phase encoded time-code data bit, plus a collection of other debug data, available via a 9600 baud serial output - a packet of data is transmitted every second on the second with the first start-bit of the first character accurately aligned to the second marker. Note that serial output is only active when the phase decoder is running. The packet is of one of two formats:

```
 "!LDsseeeeddddpppaaayy" or
 ":LDsseeeeddddpppaaayyooccrrrrrzzzzz"
```

Where...

!	the phase decoder is transitioning between locked and unlocked
:	the phase decoder is stable
L	locked status of master clock PLL. 0 or 1
D	phase decoder output i.e. time code bit from the previous second
ss	signal strength, <40 is poor, >100 is very good, max 160.
eeee	main pseudo-random phase correlator output*
dddd	differential pseudo-random phase correlator output*
ppp	filtered PWM value to VCXO, the PLL control voltage 0..1023
aaa	current AGC PWM value, the RF gain control voltage 0..1023
yy	signal amplitude at the end of phase modulation. 0..160
oo	phase correlator happiness factor -1 fail, 0 poor, 60 max*
cc	last adjustment made to the phase correlator starting point*
rrrrr	absolute phase of GPS reference input if present. 0...77499
zzzzz	absolute phase of second marker. 0...77499

All fields are expressed in hexadecimal notation. * means that the value can be negative, if its MSB is set then compliment the data, add one and interpret it as a negative number.

The DCF output from the board is just the AM receiver pulse but with a phase decoder-disciplined leading edge. The clean leading edge lasts for 50ms, after that the AM pulse shape takes over until 200ms have elapsed, the section of pulse from 50ms to 200ms may therefore contain noise under low signal conditions. Also, in the case of second marker 59, which is omitted by DCF in AM the 50ms pulse is still output. If the pseudo-random phase decoder is not locked then no pulses are output. Note that the phase encoded data for seconds 59 to 9 are currently transmitted as all ones (this is not the case for the corresponding AM data bits) this can be used as a framing sequence.

The LED (if it's in the right way round) shows the raw AM pulse in green when the master clock PLL is locked or red at start-up and when the PLL is not locked.

Signal quality

The DCF77 radio signal from Mainflingen has two routes to your receiver - the ground wave should provide useable signal up to 1000km or so from Mainflingen. Then there's the "sky wave" which is a reflection of the transmitted signal via the ionosphere, this signal component is dependent on the state of the ionosphere at any given time and it varies hugely with both time of day and time of year. Usually it's stronger at night and in the winter months. Unfortunately, since it has travelled a significantly longer distance to get to your receiver its relative phase will vary with respect to that of the ground wave, this causes all kinds of drop-outs and signal phase-shifts. Thus with increasing distance from Mainflingen it becomes increasingly difficult to obtain reliably accurate timing from the signal during the night. During the day however, 10am to 2pm in winter for example the signal is good for reproducability in the region of +/- 250us; a figure supported by monitoring over a period of many months, see below.

Accuracy

To check the prototype receiver's accuracy, provision was made to input a GPS timing pulse (positive going 3v3 pulse on J3.4) and to report the phase of this pulse and that of the decoded second marker as part of the debug serial output stream. In my case for example, on a good day, I see an absolute GPS phase of typically 32547 and a decoded second marker phase of 32768, the units are in carrier cycles so that's a 221 cycle offset, corresponding neatly to the propagation delay due to the receiver's 858km distance from Mainflingen which works out at 2.86ms or 221.8 cycles. Long-term GPS-referenced monitoring results from a prototype receiver are available on-line, see *resources* below.

For optimum absolute accuracy, a calibration is required, in either the clock or the receiver, to compensate for the ground wave propagation delay. You'll find a link below to daftlogic.com - a site that'll allow you to work out your distance from the Mainflingen transmitter from which you can then calculate your propagation delay using the speed of light expressed as 300km/ms or 3.33us/km or 0.2585 carrier cycles/km. The result in carrier cycles, rounded to the nearest integer, if less than 378 can be plugged into the source code - see the "Propagation delay adjustment" section. Note that a value greater than 377 cannot be used in this way as it will partly or completely stop the generation of second marker pulses. Once re-compiled the new firmware will output second markers tuned exactly for your location and absolute accuracies in the region of +/-250us will be available during the day. The standard issue firmware contains no propagation delay correction.

Debugging

The receiver was developed in West Yorkshire UK, some 850km from the transmitter, consequently the signal strength was fairly low and significant RF gain was required, it may be that your receiver could benefit from some reduction of gain for use very much closer to Mainflingen, to this end reduce R8 / increase R10. Antenna orientation and location is also fairly critical in low signal strength areas, the ferrite rod should be broad-side on towards the direction of Mainflingen (50N, 9E) and away from low frequency interference sources such as computer monitors, switch-mode power supplies etc. Cheap LV lighting transformers are a particular problem and, rather inconveniently, having an ICD2 debugger connected also seriously compromises received signal quality. The serial debug output will not start up until the phase decoder is in lock, from power-up this can only happen once the master clock PLL locks and the LED has changed from red to green (this takes at least 10 seconds) then a clean AM signal is briefly required - the LED will clearly show if the AM signal is noisy. Once the serial data has started up there's plenty of debug info available, signal strength should be > 40 or so for reliable reception the AGC PWM value should not be stuck at either the zero or 1023 extreme, the master clock PLL PWM signal should be relatively stable, not wildly fluctuating and not stuck at an extreme.

Further software development

Those wishing to dive in at the source code level or make a propagation delay adjustment are welcome to download the code and experiment with it as they wish but please make bug-fixes and improvements available to all. You'll need Microchip's MPLab IDE development software and C30 compiler, all free to download from:

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=81

To program or re-program the dsPIC33 you'll need a suitable in-circuit programmer, the cheapest unit supported by the MPLab environment is the PICKit3, I used an ICD2. You'll have to make up a lead to use the board's Molex 5-pin programming port J3. When a programmer/debugger is connected expect to see severe loss of signal quality!

On-line resources

For Gerber files, part-list, schematic, source code, object code, photos, performance graphs & statistics, links etc.
www.marvellconsultants.com/DCF

The software design for the cross-correlating digital notch filter was inspired by "THE SCIENTIST & ENGINEER'S GUIDE TO DIGITAL SIGNAL PROCESSING" by Steven W. Smith available from analog.com: http://bit.ly/faQrb2
(original: http://www.analog.com/en/embedded-processing-dsp/
adsp-21xx/processors/design-handbooks/
scientist_engineers_guide/resources/fca.html)

Developing in C30 for the dsPIC processor range:
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=81

In-system programming PIC microcontrollers:
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2519&param=en534451&page=wwwdevMPLABEmulatorDebuggers

Microchip's filterlab software:
www.microchip.com/filterlab

DCF77 specifics...
http://www.ptb.de/en/org/4/44/442/dcf77_1_e.htm
http://hw-server.com/design/pcf8574/komponenta_pcf8574.html
http://electronic-engineering.ch/microchip/datasheets/dcf77/dcf77.html

Googlemaps distance calculator
http://www.daftlogic.com/projects-google-maps-distance-calculator.htm

Steve Marchant
steve@marvellconsultants.com