Watcom Fortran-77 PC-COHRTW System

cmeek

June 20, 2007

Institute of Space and Atmospheric Studies
University of Saskatchewan
Saskatoon, Canada

pccrtw32.tex, Dec. 8 1999 version

Contents

1 Introduction

1.1 Historical Background

In 1978 we set up the first (in the world, I think) real time wind analysis system using an Apple II programmed in mini-assembler (basically machine language) and a Pet (Commodore) computer programmed in interpreted BASIC. The Apple has 32K RAM (yes K!), and the Pet had about 7.5K RAM free for program and data (in 1983? we switched to a Commodore CBM-2001 beacuse the Pet RAM ICs ran hot and became unreliable after a couple of years. It had a whole 16K! of RAM). Recording of analysed data was (and still is ... NOT ANYMORE SINCE THE FIRE) on 800bpi 9-track tape; first with an incremental drive and later with a slew rate Pertec. The receiver is non-coherent, and we switch the spaced antennas to it in sequence. Gain control is by voltage - and can't be done within 1 height gate separation (20μsec) because of the receiver design.

Because of the limited memory and execution time, the only way to make the system work was to use bit-amplitudes (that is each amplitude is converted to a '1' or a '0' depending on whether it is above or below a 30s mean.) The use of bit amplitudes degrades correlation values, and they have to be corrected by a theoretical function.

In 1985(?) we had Rob Strother-Stewart build a prototype coherent Rx using the existing non-coherent Rx front end as far as the IF stage. We just recorded raw data (byte amplitudes) directly on incremental 9-track tape drive. The receiver still had some problems - the same slow gain switching, and also non-negligible drift in differential I/Q channel gain (which has little effect on horizontal wind, but a serious effect on vertical Doppler).

Subsequently we asked him to design a good receiver - that is one with the required bandwidth, sensitive to $\frac{1}{4}\mu$V across 50Ω, and able to switch gains well within the 20μsec height gate separation. [Later, since there are only 8 10dB gain steps, we found it better to put a 10dB pad in front of the

Park receivers so we wouldn't lose control of strong signals .. which sometimes even now clip the receiver o/p at minimum gain]. Because of the prototype Rx drift problems, we included monitor routines and display to warn of such problems in real time analyses – which, in the event, have not materialized – but the display lingers on...

2 The Apple/C128 COHRTW system

We looked at a 68000 CPU (we still have the single board CPU developement system somewhere around in an anonymous aluminum box), but the clock rate was only 4MHz, and the assembler instructions were quite complicated – after adding up clock cycles for some of them, it didn't look much faster than the 1MHz Apple II - and there was no arithmetic co-processor. So I decided to use the RTW system experience to program the COHRTW system using an Apple II and C128 (2MHz). Again bit amplitudes were used - especially to reduce correlation computation time - although memory was also a limitation, because each byte correlation would take 4 bytes per lag per component, whereas bit correlation takes 2 bytes (for say 512 point data sequences). The output analysed record format was expanded to include phase parameters (angle-of-arrival phase, and "vertical" Doppler velocity).

3 Xiang Deng, Nick Lloyd's C++ program, Li Wentao

Although bit-amps do work, byte-amps were expected to work better, and also it should be possible to try shorter records e.g. 90s to 2min, for which the statistics of bit-amps were not suitable. We hoped our grad student Deng would try a byte-amp program, but he got discouraged and moved to Computer Science. Then we approached Nick Lloyd - who agreed to write a C++ program - which he did, and it worked fine in bit-amp mode (which we needed to match our other operating

systems in CNSR). The one working system produced was needed right away, so we did not test the byte-amp mode, but later we found it did not work - due to some simple error in calculating signal strength. In C++ such a simple error is not really simple at all; we never did find it - and by the time we got back to Nick, he had upgraded his development system. He proposed to convert the program to the new (32 bit) system - which looked as if it would have to be at least partially re-written. At that point I decided that this would take a long time, and looked around for a Fortran system. WATCOM seemed to have the best (programs could be compiled for 16 bit DOS, 32 bit Dos , Win3.x, win32, win95 etc.) AND the cheapest by a long shot (US$149) - the next possible (Absoft, I think) was about US$500.

In the middle of all this - Li Wentao, a research engineer visited us from China. He wrote a program in C for a 80286 computer which did byte-amps directly, without using one of our Rx interfaces - he designed his own hardware based partly on our interface. This ran on a real hardware interrupt (shift registers held the results of each Tx pulse until they could be read). This could run one of our old voltage-controlled-gain or new digital gain (RSS) receivers, and switched the one Rx to different antennas through relays, including coherent integration. Again because of the C it wasn't very readable - but easier than C++ because the programming was mostly linear rather than modular. All in all a very good system for a year's work.

4 WATCOM features

After 3-4 months, I have only scratched the surface of WATCOM F77 (and that's fine with me!). The manuals do not explain everything, and a lot of the work in programming is fishing around for methods which don't bomb the PC even if it's unclear or totally unknown what caused the previous problem. One big advantage of WATCOM is that assembly language functions can be included in the Fortran source, and although there are "header" files and special WATCOM

functions for getting the clock time, date etc., I prefer to get as close to the computer as possible and have everything out in the open – i.e. in the Fortran source. WATCOM is very good this way - in fact it's easier - and seems to be more reliable - to include the assembler in the Fortran source code than to link with a separate assembler program segment. [So far I've had troubles using labels in included assembler ... don't know whether this is permitted.]

5 DOS calls

I don't understand this completely, nor do I wish to , but the 80x86 CPU has an instruction called 'INT' , called a "software interrupt". It seems to me this is just a jump to subroutine ... but we'll let the PCers have their little jargon ... and who knows, maybe there are some small differences, e.g. some registers automatically pushed on the stack. In any case there are 256 possible interrupts: INT 00 - INT FF (note also that in IBM jargon, a hex number is designated by 'h', e.g. 3dh). Each of these INTs can be subdivided according to numbers pre-loaded into CPU registers, notably AX. Different manufacturers and software-engineers have different sets of DOS routines available, many seem to be virus checkers, but some are "standard" - and these are the ones I try to use. Ralf Brown's Interrupt list (archived files INTER53A thorough INTER53D at Simtel FTP sites: e.g. oak.oakland.edu/pub/simtelnet/msdos/info) is the bible. Also there is some timing information in the Kris Heidenstrom's PC timing FAQ (PCTIM003.TXT) at the same FTP site (I think).

6 80x86 assembler

And here a word on nomenclature: AX is a 16 byte register made up of al and ah – here you see the problem with using 'h' to mean hex. Also there are BX,CX,DX which can also be subdivided

in selected instructions - but there are restrictions - e.g. an immediate number (i.e. a numerical value) can only be loaded into AX (or al or ah), and tranferred from there to its desired location; memory or another register. From 386 (some 286s too?) on, registers are 32 bit, designated eax, ebx ... ("extended" AX). The 8 and 16 bit instructions are still available, and in my DOS16 version (PCCOHRTW) these were used exclusively; but this program has now been superseded by PCCRTW32, the 32 bit version, which must run in conjunction with a "DOS extender" program, DOS4GW.EXE (supplied by WATCOM), and here 32 bit assembler instructions were used..

Because of the ambiguity inherent in using ah (register) as opposed to 0ah (hex number), *immediate hex numbers* in assembler are preceeded by a '0' where necessary to resolve the ambiguity. [Needless to say, but I will say it anyway, they *could* have used the Apple nomenclature where hex numbers are preceded by '$'.]

7 General description of the PCCOHRTW system

Much of the required information can be found in the COHRTW manual (Apple II/ C128 system) - the receiver interface is the same - but the EPROM program in it has been effectively "short-circuited" – viz. most of the work, e.g. long term running means, signal strength, conversion to bit-amps, is not required for byte-amps - since the PC can figure out these parameters from the byte amps. The analysed 864 byte o/p file has the same format as the bit-amp system; a couple of flags may be different, but the "data" storage is the same. The EPROMs are labelled "CPU#1BK28-8bit" (and an upgraded version CPU#1BK30-8bit), and there is also a PC file (transferred from the Apple) called BYTEPROM.V28 which can be used to make EPROMs if the Apple EPROM programmer-card quits. Any amount of coherent integration can be used (1-255 pulses), not just powers of 2 as in the early bit-amp EPROMs.

Raw data storage: When it was found that raw data written to file was not correct - i.e. the

comm. link check bytes were garbled around the 64K byte counts, it was thought that this was a 64K segment problem in the program itself, so the matrices were broken up into <64K chunks; a little messy but it works. Subsequently it turned out that at least part of the problem was in writing large data objects to a file – maybe they have to fit into an internal RAM buffer before going to disk. Putting implied do-loops in the write statement solved the problem. This leaves the question open whether there ever was any real problem with the storage in the program – but it's done now, and there is no advantage to going back to one data matrix (unless we sometime want to make NPTS greater than 3x248). Two such (sets of) matrices are used, one for input and one for analysing. A parameter, ISEL, which is 1 or 2 and is flipped every record (in SNDCMD) tells the two program sections (collection and data analysis) what set to use. At the time I was worried about execution time, so didn't want to copy all the raw data over to another matrix; I shouldn't have been worried! [The present program - QYCRTW32 - uses one big matrix, maximum time index = 1100 at the moment.]

8 Detailed subroutine description

8.1 MAIN

This controls the system by calling subroutines in order. The following must be organized: reading initial parameters (so that many differing sites' features can be accomodated without re-compiling) (INIT), then it must watch the clock (WFSTRT) so the first record will be started at the proper time. Then it must send an NMI and command parameters (SNDCMD) to the Rx interface CPU (always called CPU#1), and start to look for data coming back (RDDATA). At the same time it must analyse the previous record (CORREL). The way this is done "simultaneously" is to alternate between data i/p (RDDATA) and analysis (CORREL) - very similar to the RTW system

Table 1: Calling tree (logger and display routines not shown)

MAIN	UPDATE()			
	INIT()			
	WFSTRT()	UPDATE()		
		TIMADJ		
		DMPHMS		
	SNDCMD()	ITIMOUT()		
	RDDATA()	UPDATE()		
		ITIMOUT()		
		GETREC()		
	CORREL()	WEED()	FCA()	HMOUT()
	FILOUT()			

in fact (but unlike the COHRTW system where incoming data were collected in interrupt mode). The feature that makes this possible is the 500Kbyte printer buffer in the PC interface between the PC and CPU#1 – in fact for normal record parameters (~130KB per record) it can easily hold the whole thing! The tasks are alternated as each height is analysed. If analysis finishes early, the full attention is devoted to incoming data (apart from a few key commands). Then the analysed data are dumped to file in routine FILOUT.

Along the way various display routines are called, and a logger routine writes important info and error messages to a .log file.

8.2 INIT

The initialization file is PCCOHRTW.INI (now pccrtw32.ini) in C:\COHRTW directory.

Notes:

- Format: The program reads a line character by character until it comes to '!'. This is replaced by a ',' and the line is re-read with the proper format (e.g. floating or fixed) for the expected parameter(s). In lines with more than one parameter (e.g. array), numbers MUST be separated by commas (the last number should be followed directly by '!') . Anything after the '!' is ignored (except for the external drive ID line) – use it for comments. If a line only contains spaces up to the first '!', the line is ignored.

- External Drive: the format MUST be "n!X", where leading space(s) are permitted, and n<>1 means: "don't write to external drive" - in our case a *ZIP*™ *DRIVE* - but as long as it looks like a normal disc drive, it could be anything. n=1 means write to X:rtwxxxxx.dat and X:rtwxxxxxx.log (as well as the same name C:\rtwdata\ files). If there is a way of using a resident "link" program to a network-connected PC such that a disc in the network-PC may be specified by a letter ≥D in the main PC (or ≥E if a *ZIP* drive is also used),

this would be a great(!) way to put stuff on the network without affecting the operation or analysis. The reason this line comes first is that the file name must be specified right away for use by the logger routine, which writes out all the parameters as they are read in.

- SITE ID and output .dat, .log file prefix:- Originally the file name was always "RTW" followed by the year, day# and .log/.dat. Since we now have Tromsø as well as Park, and are about to add Platteville, it would be convenient to have the site identified by the file name, so data don't get mixed up as we re-organize them to CD. Note that the data are always correctly identified internally by the site ID (which is a 1 byte number set in the .ini file), but the data must be read to see this – an extra complication. Consequently, we now have the option of following the siteID number with, e.g. *,'SAS'!*,where the "!" marks the end of the data in the line as usual: e.g. for Tromsø the line in pccrtw32.ini would be *212,'TRM'! [plus following descriptive text].* If the file prefix is not there, the default is "RTW".

- Port Base Addr.- this is set by dip switches on the PCL750 prototype card. If the system sticks when a record starts - an address error is a likely cause - either change the switches, or change the address in the .ini file, which MUST be entered in DECIMAL (not hex).

  The address is set by the "io sel" dip switches on the PCL750 card. If the card is held with the edge connector down, looking at the component side, the dip switches are A9-A8-A7-A6-A5-A4-A3-XX and down = '1'. So for example our usual address ($380) would be DDDUUUUX.

- ARRAY: At the moment the analysis only does equilateral arrays, and there are good reasons for only using equilateral, so the spacings must all be the same. The antenna numbers (Rx Interface designation 1,2,3 or 1,2,3,4) and the pairs (3 ant.) 2–>1, 3–>2, 1–>3 or (4 ant.)

4–>3, 4–>2, 4–>1 define the antenna pair azimuths - which must be entered in this order, and in the sense *degrees-east-of-North.* Note that the data (i.e. a "logical record" = 256 bytes data + 8 bytes status) coming from CPU#1 is the same length for 3 or 4 antennas; in the 3 antenna case, the 4th antenna bytes are ignored.

- TEST MODE: A flag was put in to invoke this, but never implemented. So this flag should always be zero. I may come up with a test feature in the future.

The following is a listing of PCCRTW32.INI

```
!COHRTW.INI(now pccrtw32.ini) - contains i/p parameters for COHRTW system
!  All numbers to be read must be in F format and come before "!"
!all #'s must be separated with commas.  NO "null lines".
!extra lines like *this*(no numeric-digits before first"!")can be put anywhere
0!E --write to extern.  drive(if=1),drive ID immediately follows !
208,'SAS'! SITE ID, Park=208("P"),Tromsø =212("T"), Platteville=214("V")?
!  use 'TRM' for Tromsø and 'PLA' for Platteville
0!  ITEST; simulate run if itest=1 (use stored real data file);nowork yet!
!antenna array:
!Three antenna pairs:2V1,3v2,1v3 (these numbers=rx# i/p on Rx interfce)
!Four antenna pair:4v3,4v2,4v1
4!NANT: 3 assumes equilateral array, 4 assumes "Y"(also equilateral)
156,156,156!ARRAY(1-3)antenna pair spacing (m), MUST BE Equilatrl!!
90,210,330!ARRAY(4-6) antenna pair direction from 1st to 2nd ant above
512!NPTS=# recs from cpu#1, tot rec len=npts*ninteg/PRF
60!PRF= Tx prf (Hz)
32!NINTEG=number of pulses to integrate,must=2**n on early eproms!
!  gate params next:  ALWAYS 125,127,8 unless you know what you're doing!
125,127,8!IDLY,ICLKIN,IGSEP:delay to 1st gate,clkin(64gates),gate sep.
300!ITSTRT=time between record starts - locked to 0000UT
16!MAXLAG - limit is 32 at present, normal (5m recs) = 16
0!  if == 1 then save "RAW" data records in disc file BYTEAMPS.DAT
!  or XXXxxxxx.RAW; NB these are 128K/rec (5m) for normal params
!  DO NOT LEAVE THIS FLAG SET!!!  FOR LONG - IT IS ONLY FOR SHORT TESTS
!  E.G. 1/2 HR
135!  radio wave length (metres) = 300./RadarFreqInMHz
896!  DECIMAL!!!  base address for pcl750 port (8255A)
32,140!  signal limits for retaining same gain next rec.
!  should be ~10dB apart
0.4,0.1!1st lag auto limit, and peak cross limit
-115 !clock correction (seconds per week -> signed-ADD to time)
```

8.3 GETKEY

This routine solves the problem of an kbd buffer (the Apple just held the last-pressed key .... and that's what I want!).

The potential problem arises becauses two different subroutines have key commands, a key command could be held up in the queue by an accidently typed key. For example if it desired to put the system into wait at the end of a record, a "W" might have been typed during a record .. but the UPDATE routine is looking for ^B (break) or "+", "−" (clock adjust commands) - so it either has to clear the kbd queue to make sure there aren't any, or wait until WFSTRT clears the "W" key. To solve this lock-up problem, the GETKEY routine checks the kbd and returns with the last pressed key. If one of the calling routines uses this key, it calls GETKEY back with a clear-key-code request, so the next call will see a null key (code = 0) until another key is pressed.

8.4 UPDATE

This reads the system clock directly with an assembler program and updates the common/dattim clock store.

It contains an ENTRY for *automatic* clock adjust, TIMADJ(msec). This is called exclusively from WFSTRT, and is done every 6 hrs (if there is any adjustment to make). **NOTE:** An adjustment will *NOT* be made if it requires a change in the minutes value ! This problem can happen if the records are very closely packed together or end near a minute mark - e.g. 4min59sec every 5min, so its best to leave a little slack here, or for special runs, turn off the clock adjust (by setting the clock adjust parameter to zero in the .ini file).

It also traps out key commands "+" and "-" and adjusts the clock accordingly. During a record, UPDATE is not called when the timer is being used, and it's also not called during the FCA analysis for a height - which may take a couple of seconds. Thus the efficient place to manually adjust the clock is between records, or after the FCA analysis is finished for the record.

8.5 WFSTRT

Automatic clock adjust: PC clocks are notoriously BAD — usually gain 10–20 sec/day. The main thrust for putting automatic adjust in (in addition to the +,−, manual seconds adjust) is that we wanted to run in parallel with the Apple/C128 system(which controls the Rx gains etc.). Actually the PC clocks are SO bad I would have had to add a manual minutes adjust as well. Anyway - the .ini file has a line specifying the number of seconds to be *added* to the time per week (can be positive or negative). The total number of seconds is loaded every week. Corrections are attempted every 6hr (0,6,12,18). Since the seconds register is the only one modified, a large correction, e.g. 50 sec will not usually be possible (i.e. can't subtract 50 from 20). This may cause trouble if we pack our records much closer together. The solution is to correct more often, e.g. every 6 hr, so each correction is less. There *should be* no truncation error involved here - the total number of seconds is loaded every week (there will be a glitch at the end of the year) and reduced by the actual amount corrected each time (which could have truncation errors), so the sum of all the corrections is right over a week. In the case of very short records it is safer to use 0 clock correction (defined in pccrtw32.ini), otherwise there may be skipped records or significant time gaps (relative to the record length) when the clock is adjusted.

NOTE: To avoid time-out errors while the program is running, the date/time clock should only be adjusted by key commands (e.g. not by GPS clock ... or other external correction system).

8.6 SNDCMD

Sets gains, loads hdr for record in progress. Sets isel ... Returns msg<>0 if something is wrong

8.7 RDDATA

Runs software timers to determine end of record (if no more data are found within $\sim 1.4\Delta t$), and dumps raw data to BYTEAMPS.DAT (but now, in QYCRTW32, it is XXXxxxxx.RAWwhere XXX is RTW by default, or the requested prefix, e.g. SAS, PLA etc., and xxxxx is the yrday) if requested. Calls UPDATE at safe times (when it can read and use clock adjust commands without affecting the software timer). A simplified signal estimate, $\Sigma|I - 128| + \Sigma|Q - 128|$ is maintained for use in setting the gains for the next record (in SNDCMD) - since the accurate signals are calculated in CORREL, too late for that use.[Actually the analysis is so fast there is lots of time to do a "real" signal calculation, i.e. s.d., but leave it for now.

Moves hdr (which was loaded at the beginning of the record) to the output store, IBLK(864).

The logic of this routine is not simple (to me anyway) so I have included (maybe) a flow chart. The main trick is to not wait too long for data coming in - otherwise there's less time for analysis.

8.8 GETREC

Uses a 1 ms hardware timer located on the PCL750 proto-card plugged in to the PC ISA bus ; returns on 1 ms time-out or when a 264 byte "logical-record" is complete. NBYTES contains the number of bytes "available" in rec() – the reason for sending this back is that it is available for RDDATA to print out with an error message if timout occurs in mid-record.

Note that since the analysis is very fast, we probably could get along without the 1ms timer ... we would be taking 18-30ms (2 tick counts) to time-out rather than 1 ms. There are also other (obscure) timers available - mentioned in the timing FAQ - which could probably take the place of the 1 ms when used in a function similar to itimout(). But what we have works now and we *do* need the plug-in card (although that could possibly be replaced with the printer port if we didn't want a printer), so it's little extra effort to add a 74LS123 timer chip. Wait 'till we get an order

for 20 units, then it's time to consider this !

8.9 STATUS

This displays flags and error condx (from the 8 bytes status tagged onto the end of each Lrec) from CPU#1 on the screen, e.g. A/D status, also writes the important and/or fatal errors (e.g. CPU#1 power-up reset, comm link error) to the log file.

8.10 FILOUT

Appends analysed data for one record to an existing file/or opens a new file (and to external drive if requested). The file(s) is only open during writing. It is designated "shared - read access only" which presumably means it can be read, say by an external link, as it is being written (However we had some trouble at Tromsø – because the external link wouldn't share with our program!). The file name is determined from the data to be written - so that each day's data will occupy a different file RTWyyddd.dat (note: the name prefix, e.g. PLA, can now be defined in pccrtw32.ini).

8.11 CLRSCRN

Called from DISPLAY only.

8.12 DISPLAY

Sets up background display – called from WFSTRT only.

8.13 LOGGER

Write a message string (16 char. time-stamp plus 64 char. message to the log file – opening and closing files before and after each write respectively. Each day's log occupies a different file RTWyyddd.log whose name is taken from the the "present" D/T (i.e. from the present "screen clock" time). [I may sometime reduce the size of the string to 64 char, so total length $\leq$80, and it can be easily printed without double spacing.] Note that the 3 character file prefix "RTW" can now be defined in pccrtw32.ini.

8.14 DISPLAY1

Write n characters out of an 80 char string "STRING" to the screen, starting at the specified row (0-top to 24-bottom) and column (0-lhs to 79-rhs) with normal attribute.

8.15 DISPLAY1A

Same as display, but here the attribute of the displayed string can be specified (e.g. flashing).

8.16 DISPLAY2

This writes a single character to the screen at row,col, with specified attribute.

8.17 DISPLAY3

This is for the RHS of the screen, to show numerical results, messages, etc. This writes a '!' and 39 characters out of an 80 char string "STRING" to row=24, col=40 and then (or is it vice versa?) scrolls up the RHS (except for the top line) of the screen by 1 row.

8.18 INCSCREEN

This increments a selected (row,col) screen byte. If it was ' ' (i.e. message count = 0) to start, it is set to '1'. The idea to to keep running totals of various kinds of errors until they are noted by an operator. They are cleared by re-drawing the screen ("S" after "W" will clear/re-draw). If there are more than 9 errors, the increment creates characters (following the ascii code). I am not expecting more than a couple of errors per week, thus the 1 character representation.

These actions are accompanied by a note to the logger which explains the problem in more detail.

8.19 CORREL

This extracts the signal data for the present analysis height, works out "good" signals strengths, keeps track of worst Rx offsets and I/Q gain ratios for display etc., and does auto and cross correlations. Messages from here on go exclusively to the RHS of the screen (results go directly to the output store - C programmers curse COMMON statements but they are invaluable here – anyway C programmers also have COMMON, they just call it "global" storage).

8.20 WEED

Find xcorrel peaks, and looks at NTD

8.21 PMFCA

Accumulate hourly means. Dump hourly means if the hour changes, do PMFCA analysis.

8.22 HMOUT

write hourly means to HRLYMEAN.DAT in C:\rtwdata\ (creates the file if it doesn't exist already)

8.23 DMPHMS

Copies the complete c:\rtwdata\hrlymean.dat file to a:\hmlist.dat on a manual key command "h" or "H" sent from wait-mode. This takes the place of printer o/p for now- and can be done without stopping the program – although we may lose one record if the hrlymeans file is very large (i.e. takes a long time to write to floppy).

9 Sofware timers

Apart from the 1 ms hardware timer on the plug-in card (we don't use the 100μsec one), time-outs are all determined from the *clock-tick-count*. This is an internal PC counter derived by dividing down (one of the) hardware oscillators. The results is a count 0-1800AF (hex) over the day; $\sim$ 18.2 ms per tick. Ancient PCs used to use this to generate a clock .. , later this count was inititialized on power-up from a clock chip. Now they appear to be connected somehow - at least changing the clock chip time (finally) changes the clock tic (probably at the next regular tic). A timer is started by reading this count CTICK() and adding a number to it which specifies the desired time-out length in units of 18.2 ms; result = ISET. When itimout(ISET) = ISET-CTICK() becomes negative, the time interval has run out. Since ISET is maintined in the routine which is using the timer, the timer can be used by several routines simultaneously (but I don't do that here). [Note that the difference must be wrapped-around – i.e. the original ISET could be > 1800AF, in which case the difference would always be +ve if no wrap-around was done.] [Note there were potential problems in the 16-bit DOS version, because the clock tic had to be obtained in two reads. The

32 bit DOS verison just uses one read, so no problem.]

10 Key commands(upper or lower case)

Note that I have programmed around the keyboard buffer – that is, the getkey( ) routine only holds the last pressed key. There is good reason for this.

Commands available anytime (but slow response during a record) are:

- "^{c}B" - stop system on second key = "y" (cancel this procedure if the next key is not 'y')
- "+" add one second to the clock time AND wait until the clock tick count changes (1 sec change = 18 tics change), which presumably happens on the next regular clock tick interrupt.
- "-" subtract one second from the clock time AND wait until the clock tick count jumps – which presumably happens on the next regular clock tick interrupt.

Commands available just between records (when an 'R' is not shown to the right of the screen clock are:

- "W" - put system on hold ("WAIT"). Note that since the last key is kept until read, this "W" command can be typed during a record, it just won't take effect until the record is over.

Commands just available from "wait" (when a flashing "W" appears to the right of the clock, or in Pentiums/W95/98 it just appears as non-flashing inverted) are:

- "S" - exit from wait/hold, clear/re-draw screen and go back to checking time for next record start

- "M" - clear/re-draw screen return from "wait" and start a record RIGHT NOW. This is useful for testing.
- "H" - copy c:\rtwdata\hrlymean.dat to a:\hmlist.dat . Must insert the floppy in disc drive a: before entering this command.

Note that key commands during a record ('R' mode) are done in the UPDATE (screen clock) routine, which is only called a maximum of twice in RDDATA (in special places where no timer is running), and not at all in CORREL (because changes in the D/T clock change the clock tic count, which could upset the software timer in RDDATA). Because CORREL tends to take a long time (several seconds), and UPDATE only deals with one key command per call, it is difficult to adjust the clock during a record.

[By re-writing so that COMMON/DATTIM/ had an extra "timer in progress" flag, we could call UPDATE from inside correl (but could only use timers in one routine- so it is KNOWN when a timer is not being used). I only use the timer in CORREL at the moment, so this *would* work. The jury is out on this so far.]

11 PC interface

Operation :::: — ???? A few simple comments- since I don't have time for a full writeup. We used the 8255A (5MHz) as peripheral in/out. It is now clear why no-one uses this chip for anything !! It spontaneously generates random (?) ~200ns spikes. We finally got over that problem by filtering the o/p !!! (in the PC interface box). The PC-to-PC-interface-box lines comprise 8 bi-directional data, 2 handshake (one each way), and one data direction line (controlled by the PC). At the Rx interface end there are 8 bi-directional, 2 handshake, and one NMI (non-maskable interrupt to CPU#1). (There *is* also a BNC in the schematic labelled "Tx control" hooked to the date

direction control - we don't use it). The sense of the electronics is *high = inactive* and *-ve edge = trigger*. The exception is the 8255A which operates on levels (not edges) in the mode we use.

A strange(!) feature is that it seem to take one or two routine (not manual) record starts for the PC-CPU#1 communication to become synchronized.

12 CPU#1 EPROM

A listing of the pertinent assembler is(?) included (just those routines used by the BYTE-amp EPROM). There are two byte-amp EPROM programs (not counting CPU#1BK28L' for London ... which has special handshaking) CPU#1BK28-8BIT and CPU#1BK30-8BIT. The differences appear to be

- 1. BK30 operates the Tx on/off control in the receiver interface (BNC connection on back of Rx interface - which we have never used before). Since the PC has a less usable one at the moment - i.e. it only switches when command data are being sent to CPU#1 – a short prf change which is probably too short to hear, this would be valuable for Park anyway.

- 2. BK30 attempts to fix a potential problem with the raw data port (RDP) o/p – which is not used usually – the problem is that while CPU#1 is finishing the last operations of the record, interrupts are still getting through – beacuse the program didn't know the record was over until it had sent the last Lrec – and so data are still being sent to the RDP. Of course any system hanging onto the RDP can just be programmed to only accept a certain amount - so it's not a big worry. BK30 looks ahead to anticipate the last analysis and skips RDP ops when the official record is ended.

- Future improvements ? ... two things (maybe) use a check sum as well as check bytes $00 $FF for each l-record sent to the PC, and put sequence numbers (pulse count) in the RDP

output in the lowest height gate so it will be known whether some have been missed. The latter assumes a secondary PC hooked directly to the RDP, or to a 256 byte buffer - not a big buffer such as we use between CPU#1 and the PC

13 pcAnywhere

The "remote" computer pcAnywhere setup should have: 115,200 Baud, RTS/CTS, parity <none>. We make the selection: *USRobotics Sportster 14400 modem*; DTR always ON, RTS always ON. [We use pcAnywhere for Windows Version 1.0 in our 386 (Windows 3.1). These settings work – maybe other settings will also work.]

[NOTE: Platteville MFR is the "HOST" computer; do not modify any of its settings.]

14 Tx synch/RF pulse generation:

The trigger is derived from the 60Hz line frequency rather than from a master oscillator. A good reason for this is that most interference is from the local power line, and we can shift our phase to move away from it. The other *potential* (i.e. I haven't tested it) advantage is that it is not stable with respect to the o/p 2.22MHz phase, or more importantly any external RF signal. (We are assuming some line power jitter here). Thus external RF interfering signals which *are* running on master oscillators will (should) look like random phase signals to us, and thus coherent integration will reduce their magnitude significantly.

At Tromsø we had to be synchronized with a digisonde ... so we put in a free running oscillator to randomize the our phase. All it takes is a random ~500ns delay inserted into the external synch line.

The Tx trigger pulse (100μsec +ve pulse with the Adelaide Tx) operates a div-by-8 circuit

on a free running 17MHz oscillator. The Adelaide Tx require such a long RF pulse so it can do Gaussian shaping.

15 Receiver operation:

There are 2 reference signals - the 17MHz and the PLP. The 17MHz is the same signal that was divided by 8 to generate the Tx pulse, the PLP is a +ve pulse which starts at the 1st zero crossing of the 2.22MHz signal (which was tapped off 1 stage previous to provide 2f for the Tx), and lasts as long as the Tx pulse ($\sim$100μsec for the Adelaide). In the receiver the PLP is used to initiate a $\sim$3msec monostable which controls a div-by-8 chain operating on the 17MHz to get back the 2.22MHz. The phase may be different from the Tx o/p, but the difference is constant - which is the important thing. The reason for this somewhat complicated reference frequency is to avoid having 2 MHz floating around outside the receivers on coax. Because the receiver is only *on* for this 3mse, it is not useful for looking at the full 60Hz timebase, e.g. for powerline corona discharge. We usually use a non-coherent receiver for this job; either hooked to an un-used antenna, by splitting the antenna with the strongest signal between the RS-S Rx and the non-coherent Rx, or by temporarily switching one of the antennas to the non-coherent receiever.

The gain control is a 3-bit signal (10dB/step); the gain is set separately for each height gate (20μsec). The gains are clocked into the receiver 1 or 2 μsec *after* the A/D sample/hold has been triggered. This avoids switching spike interference. The new gain takes a couple of μsec to settle.

Receiver blanking: blanking is done in the first stage. For some reason, *blanking* is very slow - thus we can't start blanking the receiver at the beginning of the Tx pulse. Unblanking is also slow, so what we did was unblank the receiver for $\sim$3ms starting at the Tx pulse. This works well on most of the receivers ... but one of the ones going to Platteville unblanks very fast, and

rings for a long time ! .. so what to do ... well, an open collector circuit drives an optic isolator in each Rx (the 4 connectors on the back of the Rx interface are wired directly in parallel); so what we do is take the $\sim 100\mu$sec synch pulse and "add" it to the blanking through a special cable.

Phase flip: There is a phase flip control (0/180°) which changes the reference phase inside the receivers. We have never used this ... so no more about it here.

Front panel toggle switches: The philosophy behind the coherent receiver design was that it should be able to operate without any external signals. Thus, for example, the gain control inputs are pull down, and if not connected, the Rx is at max. gain. It can also produce its own reference signals.

- TEST/NORM toggle switch: when in TEST position, the receiver is completely unlocked - the div-by-8 is free running. When in NORM position, the div-by-8 is taken from the PLP i/p. If the Rx does not operate in NORM, maybe there is no detectable PLP i/p.

  In NORM position, the Rx should be phase-locked within 1-cycle of 17MHz (45°).

- EXTERNAL/INTERNAL - in INTERNAL position, the 17MHz reference comes from an internal Xtal oscillator. In this mode there will be 45° jitter (or more if no PLP). In EXTERNAL position, *provided there is some sort of switching i/p ... not necesarily 17MHz*, the reference is taken from the external input, and there should be no phase jitter. If no signal is detected on the external i/p, the receiver reverts to internal 17MHz reference. [Note in the early, 1988, design a relay was used to switch - these finally froze up, disabling the Rx. So we removed the relays and put jumpers in to take external i/p only. The newer receivers use electronic switching.]

-

16 Routine system checks:

- **Date/Time Clock:** must be in UT; use "+" and "-" keys to adjust seconds. Any larger adjust must be done outside the program. If the errors are usually large, then estimate how many more seconds per week need to be added or subtracted, and modify pccrtw32.ini (last line) accordingly.

- **Rx Phase Check:** The Tx synch box puts out 2f into the optical fibre; the Tx divides this by 2 because it needs an exact 50% duty cycle. It starts dividing (and forming the Gaussian envelope) on the 1st edge from the optical fibre – this 1st edge is sometimes not defined very well. If this trigger is unreliable, the o/p RF will sometimes flip by 180°. Bad news!! In our experience the problem has been the optical fibre plug/socket combo at the Tx synch box.

  To see whether there is a problem, look at the ground (GND) pulse on the scope, or at any Rx signal, I or Q, when the gain is low (look at the Rx interface gain o/p on the scope - this is a representation of the 3-bit gain number). The GND pulse, say the 1st 20-40 μsec, should not vary at all. Signal at low gain, say less than number 4, should fade smoothly up and down. The exception to the latter is that 1 or 2 μsec after a gain change the signal can look jittery. [At high gain any jitter or apparent phase flipping is likely due to noise.]

  If there is a problem, first check the Rx front panel control toggle switches: they should all be in position NORM and EXTERNAL and the green LED should be on. If those were ok, try wedging the optical fiber at the back of the Tx synch box with foam rubber or something similar – it seems that the pointing angle is very critical (so is any vibration).

  The other (very unlikely) cause is if the RFref/PLP buffer box triggers have drifted. There are trimpot adjustments on the front.

- **Signal strength per Rx:** This is a line at the bottom of the screen – it shows the signal at

each Rx for the signal (over the 32 heights, which ht. number is shown in the worst offset - I/Q display) for the height closest to mid signal (128 out of 0-255). These vary with scatter conditions, gain etc., but at relatively low gain, they should be within 1 or 2 dB on the average. These signal values are saved in the "864" o/p data.

By the time you see these numbers, the system is running the next record, with possibly different gains, but since the gain changes at most 1 step per record, looking at the 'scope gives an idea of the gain (actually I should have put it on the screen). With the 'scope at 100μsec/div the approx. height is 15km/div; the height given by the height number is approx. 46 + 3*ht# km.

If they do not look equal, check the antennas.

- **Rx offsets & I/Q ratios:** Ideally the offsets should all be zero and the I/Q all 1.0. [note: to avoid *** I have shown I/Q or Q/I whichever is <1.] The Rx offsets are $\pm$128, but anything > 2 digits shows as asterisks. Usually these should be within $\pm$10 – but they do vary a little. The I/Q values should vary around 1.0, usually not less than 0.90.

  If something is wrong with either of these when there is significant signal at the selected height, then there may be a Rx or Rx buffer box problem.

- **Rx 'scope signals:** Use the I/Q monitor card button to look at each Rx I or Q on the scope. If the interference level is significantly larger on one antenna, there may be a loose or corroded connection.

- **Local power-line-locked interference:** Usually this is corona dischange. Since the RS-S RXs are only on for 3ms, the best way to look for a quiet spot in the 60Hz baseline is to use the non-coherent receiver. We hope there is one " strong" antenna whose input we can

tap for the non-coherent Rx ; or maybe we should have a push button box to temporarily switch one of the antennas to the non-coh RX.

While looking at this Rx o/p on the 'scope, say at 2ms/div, the Tx pulse should be moved into a quiet spot using the phase adjust (0-180°) and toggle switch(0 or 180°) on the front of the Tx synch box. NOTE that because of the strange circuitry, turning the phase control too far may change the prf – and cause the PC to timeout and abort the record.

- **A/D status:** This line shows whether all bits of the A/D convertors have changed state (up/down) since the beginning of the record. If all have done so, the display says "OK". If some have not, then those are printed as flashing (or inverted), with the others as normal characters - in order I1 Q1 I2 Q2 I3 Q3 / I4 Q4 . Usually, unless a receiver is not working or the I/Q outputs are not hooked to the Rx interface, the status is OK – that is all bits have tested OK in cpu#1 before the 1st Lrec is sent back. Once in a while - e.g. if signals and noise at all heights are very low - this display may appear on the screen. [To see what it looks like, just disconnect one of the I or Q cables for a short while at the beginning of a record.]

-
-
-
-

- Screen "error" counters/flags: These are cleared when the screen is re-drawn by going into and out of "wait" mode.

Intfce reset: This counts the number of times CPU#1 has been reset (either manually or by power-up).

Comm.Link errs: When the PC receives a logical record (256+8 bytes) it checks the last 2 bytes for $00, $FF . If these are not correct, it aborts the record and increments the screen counter.

Total#Fail: This is the number of record aborts - e.g. time-outs, comm. link errors, or other communication (with CPU#1) problems. When the error occurs, an error message also appears at the bottom of the screen with details (and in the log file) – but that screen area can be overwritten by another similar message, or blanked out if the next test (maybe for a different problem) is ok.

CPU#1Status: CPU#1 maintains a 1 byte message containing (so far) 2 message flags : one for "bad command received" which just means that the last command bytes were not $00,$FF, and one for "data overload" which means that CPU#1 was not able to get rid of its last integrated data before the space was required for the next - i.e. the synch rate is too high or the amount of integration too low. A value $00 means "OK" (and "OK" is written on the screen); if not zero then the byte is written out in bits. This message will remain until CPU#1 is reset, *whether or not the error continues to occur!* Changes in this byte are logged.

Thus, if there is any message other OK, CPU#1 should be reset (...preferably between records ...but if not the system will recover anyway) so it can be seen whether this was an isolated error, or common. If common, then there is a problem which must be fixed.

•

17 Y2K

We use a 1 byte (2-digit BCD) year number for internal data year number, and o/p filenames. We have checked its operation over the 1999-2000 year change. Locally (Saskatoon) this will cause a problem in 2008 (because we used a one digit year in 1988 raw data - however, there are other operating parameters which distinguish the earlier and later data), and after 2088 (I am not going to worry about that !). In other sites the problem would occur 100 yr after their start – I'm going to worry even less about that.

18 Height designation/calibration

The system always records 32 heights (although we could do 64 with a modified EPROM program); the A/D sample times are set by the DLY, CLKIN, GSEP parameters (in the .ini file). Except for GSEP, these are software timing loop parameters: DLY sets the delay to the 1st height gate, and CLKIN opens the shift-register clockin to record 64 gates. These paramters are NOT simple to set properly! It is very easy to *lose* a height gate without knowing it!

The raw data file uses a height number designation 00–1f (0-31), referring to *nominal* heights 49 - 142 step 3Km (given our normal DLY,GSEP,CLKIN parameters). This height number or the nominal height are used in all routine/compressed data files. To get "virtual height", corrections for extra delays (relative to the Tx synch pulse) in the Tx/Rx feeder cables and Tx need to be made. We have compared observations of research balloons at Parksite (Saskatoon), one with our old tube Tx and one with the Adelaide Tx (the balloon that went off course, couldn't be shot down, and landed in Finland!), and concluded that for Saskatoon data virtual height equals

nominal height minus 9 Km. Other sites may vary from this depending on feeder type (coax vs. open wire), site size etc.

19 Routine operation:

At Saskatoon we make weekly visits, since we are fairly close (90Km round trip). Some kind of regular visit is necessary, because the data do not tell you everything: number so data can change because of external interference, ionospheric conditions etc., as well as equipment/antenna problems, and by the time you decide there is really something wrong (by examining the data remotely), you have lost a couple of weeks. The main source of on-site information is the 'scope; this shows the phase lock, the gain range (i.e. shows how strong the signals are), the noise/interference level (but even this doesn't tell you whether there's a better place to be on the 60Hz time base, and even if we were able to send back digital 'scope pics, at present there is no remote way to change the antenna select to look for a bad antenna ... although presumably something could be done with the printer port).

The ZIP disc can hold a full year's data (~91Meg) – but probably the FAT overflows before that. In normal operation here we change the ZIP disk every 3 mo. and keep track of operation with the ascii hrlymean.dat file. So the data would be copied at UofC every 3 months, and the ZIP disc sent to us, or the data put on a FTP site .. whatever works. pcAnywhere is somewhat of a new wrinkle for use (although we did use it at one of our sites some years ago), we will just have to try various procedures to see what works. It may be that we can forget about the ZIP disc (except for safety backup) and copy everything over the phone lines.

We always keep a duplicating log book at our site here. We have packed one for Platteville. It's a good idea to keep this up-to-date, and write something down each visit - anything odd about the 'scope view, screen error messages, e.g. Intfce rst shows there was a power failure, etc. Then

if the data take a jump (e.g. the phase calibration) we will know why.

20 Quick look (QL) programs:

I have included various QL programs in the Platteville PC in directory C:\CMPROGS\ plus a few utility, e.g. wind data compression, programs. I have run into some DOS/16M-DOS4GW "conflicts" here which I don't understand ... however they do seem to work at certain phases of the moon and if you execute them from the *right* directory (e.g. C:\). These did not occur at all when I had WATCOM Fortran77 installed ... but we need WATCOM elsewhere - and it is not sold anymore.

[Some of the programs do plots (our plot system puts out PCL and is intended to go directly to HP LaserJetii). Or they can be o/p as HPPLOTxx.PLT files (to be copied later by copy/b to a HP LJii printer - and note that other makers who claim their printers are HP-PCL compatible are usually lying, in our experience), or .BMP files (2400x3000 dot). These can be looked at with IMAGING: access by windows icon.

They have been upgraded to be Y2K compliant (at least before 2008) - but no time to check all. Here is a list:

- FPLTNEW: Read the QL hrlymeans.dat-type file (9 km layers) and plot non-linear-scale spectra for a selected day range. One page plot.

- FILEDUMP: list (part of a) a specifed file in hex; various options, e.g. record length, print length per record etc., available. This is intended for a quick on-screen check of the contents of binary files. Run this one from C: !

- COHDSP: This a one page plot display of *all* raw data parameters for 2 hrs (12 recs) and 3 heights. It probably assumes 5m records. It requests zero-lag phases so the AOA plot is

correct .. come to think of it , it probably only does Park - Robsart - Sylvan Lake array orientations.

Since I don't know the Platteville array yet, but Platteville is like Robsart/Sylvan Lake (1.5λ/202m spacing and one leg E-W), I have used the Robsart array. If the Robs. & Platt. arrays are not the same the AOA results will be basically correct but rotated and/or inverted.

One page plot.

- CXHVW: Reads raw (864) data (or compressed raw from cxsfvw) and produces a compressed (binary) hrlymeans file with fixed 8K (that is 8000!) byte blocks - I think no special selection of Vtr - so could be some spikes, and just 64-118 nKm (n = nominal height; refers to the height gate #, the 32 gates = 49-142 nKm): the height designations in the o/p file are "nKm".

- CRTWPHIL: Do histogams of zero-lag phase, optional multi-page plot o/p (I think) – location of most probable phase is the cable correction to be used in AOA calculation in programs such as PHIPLT.

- READRAWP: Reads the XXXxxxxxx.RAW data o/p when the "raw data flag" is '1' in the .ini file, and does FCA (of some sort) on it.

- TRMDIAG: Creates a HMLIST.DAT file (9km layer) – basically duplicates the hrlymeans.dat file produced in real time, but from raw (864) data.

- HMCHECK1: Produces an hrlymean plot (6 9km nominal height layers: 64-70, 73-79 etc.) from the QL hrlymeans file in C:\rtwdata - full page. One page plot. NOTE THAT for this program (and HMCHECK3), you must call it with a parameter (any parameter , e.g." >

hmcheck1 0 "), otherwise, the program will assume the hm-file is a:\ hmlist.dat, (which is the file produced when you use the "H" key command).

- HMCHECK3: Like HMCHECK1 but produces a "web-page-sized" .bmp - for conversion and viewing on a web site as a .gif. This is the program that produces our routine web plots. One page plot.

- HPLOT: Single height hrly means plots, probably 18 hts (selectable?) and selectable # days per frame. Keeps plotting pages until specified day range ended - intended to be glued together to produce a calcomp-type plot; various optional input data formats. Use cxhvw-type data files for i/p. Multi-page plot.

- CXSFVW: Make compressed raw data, fixed 8k records, the same sort of format as compressed hrlymeans 8k files. Keeps Vz, Vtr, Phitr (I don't recall whether it keeps a Vz if no Vtr)

- CXRAWARO: Arrow plots of raw (5m wind) data, max 6 hrs, max 18(?) heights; reads from raw data files (or compressed raw ?). One page plot.

- RDCRTW: Sample of subroutine which automatically scans through file names of raw (864) data and returns (864) records in sequence - for information/transfer.

- RDCXHVW: Sample subroutine to read compressed (binary) hrlymeans file - for information/transfer to another PC etc.

- SFCOV: "Swept frequency covariance" — Plots non-linear-scale Fourier xforms for an m-day interval moved along by n-day shifts at 1 height. Uses compressed hrlymeans ("8k") file(s). One page plot.

- RESIDPLT: Do 1 day, 12(?) heights; show raw data, smoothed raw, tide fits – specially for looking for waves on top of "background" wind. Multi-page plot.
- phiplt: Plot raw AOA phases (and Vz ?) - may have other options. Multi-page plot(I think)
- GENPSH2: Originally used to plot daily frames of accurate (to 1 km by parab. fit) signal height-profile peaks – good for looking for sporadic-E, descending layers etc. Other options/parameters are available. Combined plot(s) for full day range output at end. 32 sub-frames per page. One or multi-page plot.