IMPLEMENTATION OF A PROTOTYPE REAL-TIME SNOW THICKNESS RADAR

L.A. Lalumiere

Ocean Sciences Division
Maritimes Region
Fisheries and Oceans Canada

Bedford Institute of Oceanography
P.O. Box 1006
Dartmouth, Nova Scotia
Canada B2Y 4A2

1998

Canadian Contractor Report of Hydrography and Ocean Sciences 48
Implementation of a Prototype Real-Time Snow Thickness Radar

by

L. A. Lalumiere*

Ocean Sciences Division
Maritimes Region
Fisheries and Oceans Canada

Bedford Institute of Oceanography
P.O. Box 1006
Dartmouth, Nova Scotia
Canada B2Y 4A2

*Formerly of Aerodat Inc., 6300 North West Drive, Mississauga, Ontario L4V 1J7
Presently at: 217 Lorne Avenue, Newmarket, Ontario L3Y 4K5
This report was prepared under DSS Contract No. F5955-6-0246/001/HAL
Correct Citation for this publication:

TABLES OF CONTENTS

Abstract ................................................................................................................................. iv
Résumé .................................................................................................................................. v

1. Introduction ........................................................................................................................ 1

2. Project Tasks ..................................................................................................................... 1
   2.1 Hardware Development ............................................................................................... 1
      2.1.1 System description ............................................................................................... 1
      2.1.2 Power supply ........................................................................................................ 2
      2.1.3 EZkit-Lite board .................................................................................................... 3
      2.1.4 Installation .............................................................................................................. 3
   2.2 Software Development .................................................................................................. 4
      2.2.1 Radar Control and Data Acquisition ................................................................. 4
      2.2.2 Snow Thickness Processing .................................................................................. 4
      2.2.3 Functions implemented in the DSP ........................................................................ 5
      2.2.4 Data Output ........................................................................................................... 6
      2.2.5 Lab Testing ............................................................................................................. 6
      2.2.6 Ice Probe Software Modifications ........................................................................ 7
   2.3 Field Work .................................................................................................................... 7

3. Conclusions and Recommendations .................................................................................. 9

Tables .................................................................................................................................. 11

Figures .................................................................................................................................. 13

Appendix A - Log of Field Activities .................................................................................. 22
Appendix B - Tables of Radar Flight Information ............................................................... 24
Appendix C - Radar Processing Implemented for 'C' for DOS ............................................. 26
Appendix D - Radar Processing Implemented for the 2181 DSP ....................................... 34
ABSTRACT

Results are described of development work and field test for the integration of a Digital Signal Processing (DSP) board with a ground penetrating radar system for real-time snow thickness measurement of a helicopter-borne sensor package. This is the first airborne ground penetrating radar system to provide real-time processed results.

Ground penetrating radar systems have been flight tested in Canada for use as snow thickness sensors in 1991 at Tuktoyaktuk, N.W.T. and in 1992 near St. Anthony, NFLD. Radar electronics were contained in a small package mounted in a helicopter-towed bird containing, in addition, an electro-magnetic (EM) sensor to measure ice thickness. Data was logged on analog tape or laptop computer with an analog-to-digital converter board and processing was performed after the flights. Results showed that the ground penetrating radar could measure minimum snow depths of approximately 25 cm with a resolution of 5 cm.

The present work takes advantage of new DSP-based micro-controllers and stereo analog-to-digital and digital-to-analog converters in the implementation of the control electronics and real-time signal processing for the measurement of snow thickness and flying height. The ground penetrating radar system is packaged in a small 30 cm by 35 cm by 15 cm package, with measurement results outputted over an RS-232 serial link.

Tests of the new snow thickness radar sensor in March 1997 showed that (1) the addition of the on-board digitization and signal processing to the radar worked well; (2) the software developed to process the incoming data provided real-time output data over a standard serial interface; and (3) the radar was able to return real-time results for flying height, with noisy returns for snow thickness.

Overall noise levels were higher than with the radar electronics used in 1992. Further work should compare the 1997 and 1992 radar electronics to establish reasons for the higher 1997 noise levels. Although for the 1997 flight tests, the radar was mounted inside the Ice Probe bird, it can be flown in its own bird or hard-mounted to the helicopter. In addition to sampling real-time snow depths, this system lends itself to the measurement of level fresh water ice thicknesses.
RÉSUMÉ

Présentation des résultats des travaux de développement et d'essais sur le terrain en vue de l'intégration d'une carte de traitement numérique des signaux (DSP) à un géoradar embarqué à bord d'un hélicoptère en vue de la mesure en temps réel de l'épaisseur de neige. Il s'agit du premier géoradar aéroporté fournissant des données en temps réel.


Les présents travaux tirent parti de nouveaux microcontrôleurs à base de processeurs numériques de signaux (DSP) et de convertisseurs stéréo analogique-numérique/numérique-analogique mis en œuvre dans l'électronique de commande et le traitement numérique des signaux pour la mesure de l'épaisseur de neige et de la hauteur de vol. Le système de géoradar est intégré à un petit boîtier de 30 cm x 35 cm x 15 cm et les résultats des mesures sont transmis par une liaison série RS-232.

Les essais du nouveau capteur radar de mesure d'épaisseur de neige qui ont eu lieu en mars 1997 ont démontré : (1) que l'ajout de fonctions embarquées de numérisation et de traitement des signaux au radar donnait un fonctionnement correct; (2) que le logiciel élaboré pour traiter les données d'entrée fournissait des données de sortie en temps réel au moyen d'une interface série standard; et (3) que le radar était capable de fournir des résultats en temps réel au sujet de la hauteur de vol, l'épaisseur de neige étant représentée par des échos bruyants.


Bien que, pour les essais de 1997, le radar ait été monté à l'intérieur de l'«oiseau» Ice Probe, il peut être installé à bord de son propre «oiseau» ou fixé directement à un hélicoptère. Outre la mesure d'échantillons de profondeur de neige en temps réel, ce système se prête à la mesure de l'épaisseur de glace d'eau douce plane.
1. Introduction

This final report documents the work performed on the development of a real-time snow thickness radar system. This system was designed to be an additional sensor in CCG's Ice Probe system.

A commercial ground penetrating radar system was flight tested for use as a snow thickness sensor in 1991 at Tuktoyaytuk, N.W.T. as part of the Transport Development Corporation's (TDC) electro-magnetic (EM) ice thickness measurement system. For the 1992 trial near St. Anthony, NFLD., new electronics were developed so that all the radar electronics could be contained in a small package mounted in the TDC EM bird. Each of these systems required data to be logged on analog tape or on a laptop computer with an analog-to-digital converter board. All processing of data were performed after the flight.

This new work takes advantage of new DSP-based micro-controllers and stereo analog-to-digital and digital-to-analog converters in the implementation of the control electronics and real-time signal processing for the measurement of snow thickness and flying height. The snow thickness radar system is packaged in a small 30 cm by 35 cm by 15 cm package, with measurements results outputted over an RS-232 serial link.

Test flying of the new snow thickness radar sensor was performed during March 1997. The testing showed that the real-time processing software worked as expected and but there is a problem with the radar electronics. Overall noise levels were higher than with the previous radar electronics (last flown in 1992). Further work with the electronics will be required before further flight testing can occur. To facilitate testing, the 1992 radar electronics should be re-assembled and tested for use in evaluating the new radar's performance prior to more flight testing.

2. Project Tasks

This project had three main tasks, hardware development, software development and testing.

2.1 Hardware Development

2.1.1 System description

The snow thickness radar system has been developed around a GSSI model 3102DP ground penetrating radar (GPR) system. This system originally
contained the radar antennas and the transmitter and receiver electronics. This unit was designed for use with an external control unit, which an operator would use to set system settings and view the data on an oscilloscope. Data were typically recorded on analog tape. This prototype was built upon the original factory unit which was purchased in 1990.

There are 7 subsystems in the new snow thickness radar system. They are:

i. the antenna elements;
ii. the sampling receiver;
iii. the transmitter;
iv. the sampler control board;
v. the power supply, digital logic and signal conditioning board;
vii. the Analog Devices EZkit-Lite board.

A photograph of the inside of the radar unit is shown in Figure 1. The antenna elements are mounted on the inside of the bottom of the radar package. A dimensional drawing of the main components mounted inside the radar is shown in Figure 2. The Analog Devices EZkit-Lite board is mounted on the lid of the box.

The first 3 subsystems listed above came with the radar unit. The forth unit, the sampler control board (made by GSSI), was a spare part for an old GSSI System-7 radar control unit. The fifth subsystem is a new component built for this radar.

The sixth module, the calibration unit, was assembled for the 1992 St. Anthony field work. It contains a GSSI sampling receiver board and a circuit which passes the transmitter trigger to a delay line to create a signal with a series of pulses 25 ns apart.

The last module, an Analog Devices EZkit-Lite board provides the following:
- provides the base clock for the overall system;
- generates the slow ramp required by the sampler control board;
- digitizes the analog input from the radar receiver and the calibration unit;
- processes the incoming data for flying height and snow thickness;
- outputs the result over an RS-232 serial interface.

Figure 3 shows a how the various sub-systems are interconnected.

2.1.2 Power supply

The power supply was redesigned from scratch. Some of the components from the old power supply were obsolete and spares could no longer be obtained.
Two VICOR DC-DC converters were used to supply +/- 12 VDC from 28 V helicopter power (available in the Ice Probe bird). Two Endicot Research 12 VDC to 150 VDC converter modules were used to generate the high voltage required for the radar transmitter and receiver electronics. The receiver requires -70 volts. This was generated using a 5 V input and reversing the output of the Endicot supply. There are two +5 V supplies, one of them was for the Endicot that produces -70 V, the other was for the new digital logic elements and for the sampler control board. A schematic of the power supply is shown in Figure 4.

2.1.3 EZkit-Lite board

Several modifications on the EZkit-Lite board were required along with the creation of several small ancillary electronic circuits.

The analog output from the digital-to-analog converter (DAC) chip is AC coupled. Modifications were required to tap off a reference voltage and the DAC output signal, to a circuit (shown in Figure 5) which would provided a DC coupled output. The DAC produces a 'slow ramp' at the rate of 15.64 Hz (16 kHz / 1023 points per ramp). The ramp varies from 0 to -6 V each sweep. The rate and slope of the ramp sets the output scan rate and is one of the controls used to set the radar range window length. Potentiometers are mounted inside the radar to scale the ramp for fine adjustment of the range window and to set the time within the radar range window when the transmitter fires. (A capacitor on the sampler control board is used to set the slope of the fast ramp. Together, the fast ramp and slow ramp set the radar range window and scan rate.)

The signal level from the radar receiver and calibration unit is too high for the EZKit-Lite board’s analog-to-digital converter (ADC). These signals are passed through an attenuator circuit to reduce the signal level so it is within the ADC’s input range. This circuit is shown in Figure 6.

The 12.288 MHz clock used with the on-board analog to digital converter chip was tapped off and divided down to 48 kHz for use as the radar system clock. The circuit that performs this clock division is shown in Figure 7.

These ancillary circuits for the radar system are mounted on the same board as the power supply components.

2.1.4 Installation

The calibration unit was mounted in an aluminum box and secured inside the radar unit. The transmitter and receiver triggers are fed into the box using coaxial bulkhead connectors. It is important that noise from the calibration unit not be picked up be the radar receiver.
The low power transmitter was removed from the radar and the high power transmitter was installed. The high power transmitter was found (after 5 years of storage) with its wiring attached incorrectly, and some work was required to determine the problem and get it going.

The sampler control board connector was rebuilt. Stand-offs were epoxied to the inside of the box so that this board could be bolted securely.

The EZKit-Lite board was mounted to the lid of the radar box. A new connector was built to bring various radar signals outside the radar box. For debugging purposes, a start of scan pulse and the radar and cal. unit signal can be viewed on an oscilloscope. For normal use in the Ice Probe bird, the only connection is the 28 V input and the RS-232 serial connection.

2.2 Software Development

Software development was centred around radar control and data acquisition tasks, processing the radar data into an interpreted result and output of the real-time results to a data logging system.

2.2.1 Radar Control and Data Acquisition

The DSP on the EZKit board interfaces to a analog-to-digital converter (ADC) and a digital-to-analog converter (DAC) chip. This chip provides two channels for each of the ADC and DAC sides of the chip. The ADC is used to digitize the analog signal from the radar receiver and calibration unit. One channel of the DAC is used to generate a slow ramp, which is used by the radar's control electronics to set the output scan rate. The other DAC channel is used to generate a start-of-scan pulse. The start-of-scan pulse is used with external devices such as an oscilloscope (to view the analog wave forms) or to log data with an external analog-to-digital converter (such as a PCMCIA ADC card used with a laptop).

The fact that the DSP generates the slow ramp, means that there is now software control over several radar parameters. The software can be changed to select a different scan rate and change the window length.

2.2.2 Snow Thickness Processing

As part of the 1991 and 1992 field trials with the earlier radar systems, a model was established as a basis for classifying radar echoes as ice echoes or snow echoes. A software algorithm was implemented following the model to automatically estimate snow thickness.
The model used the following assumptions:

- the radar footprint diameter is approximately equal to antenna height (about 15 m);
- over a smooth flat reflector most of the energy is returned from a region with a radius of less than one tenth antenna height (first Fresnel zone);
- small radar targets in a rubble field return echoes with much smaller amplitudes than large flat targets (flat ice);
- the echo from the ice surface (whether covered by snow or not) has the largest amplitude in the trace; and
- the echo from the air/snow interface is the largest signal greater than random noise levels that arrives before the ice echo.

Before peak location begins, the raw radar data is filtered to remove high frequency random noise, low frequency noise and background system noise. For every trace the maximum peak value is found and its value and location are stored. This peak might correspond to an echo from the top of the ice. The data is searched for the first peak that has a value greater than a given threshold. The threshold is chosen to be well above the RMS noise level. If a peak is found, its location and value are also recorded. This peak might correspond to an echo from the air/snow interface.

The peak locations are processed for snow thickness by subtracting the snow peak position from the ice peak position, dividing by the sampling frequency and multiplying by the radar velocity in the snow (0.15 m/ns). No snow thickness determination is made if:

- the ice echo is too small;
- no snow echo is found; or
- the ice peak amplitude is smaller than the snow peak amplitude.

In preparation for the coding of a real-time snow thickness processing system, the radar data archive from the 1991 Tuktoyaktuk field work was restored. The Matlab version of the snow thickness processing algorithm were also restored from an archive and made to work again with the data from Tuktoyaktuk.

The algorithm was coded in ‘C’ and tested on a DOS-based computer. Once the ‘C’ code was working, it was recoded in assembler to run on the Analog Devices 2181 DSP chip, where further debugging was required.

2.2.3 Functions implemented in the DSP.

1. A/D conversion of data and calibration channel.
2. Process cal channel for RF sampling interval and first rising edge of the cal. signal.
3. Acquire background - stacking and band-pass filtering.
5. Subtract background.
6. Peak detection.
7. Peak extraction - apply model.
8. Choose dielectric constant.
9. Output snow thickness and flying height*.

Note: *indicates that this function is requires further development.

Figure 8 shows a block diagram of the software implemented for the DSP.

2.2.4 Data Output

By default, the DSP outputs a string with the detected flying height and the snow thickness measurement. Other output strings are available for debugging the system or to get additional information to assist with modifications to the processing algorithm.

A command can be sent to the radar to get it to output the raw radar waveform, the calibration waveform and the radar waveform after background subtraction. This waveform dump is very slow and has limited used, but it does enable the acquisition of raw radar waveforms without extra cables added to the tow cable or having a VCR or laptop to record analog data.

2.2.5 Lab Testing

The radar unit, a laptop computer and a power supply were taken outside to the rear of Aerodat's office. A tape measure was laid out from the wall of the building so that the distance of the radar from the wall would be known. Raw radar and calibration waveforms were collected at half metre intervals from 9 metres out to 17 metres. Figure 9 shows a plot of the radar data at the various ranges from the wall. The difference of the time of arrival of the echo from the wall at the different distances is used to confirm the sampling interval of the system.

This sampling interval measured with the test radar data was compared with the sampling interval determined from the cal. signal. Figure 10 shows a plot of the calibration signal. The number of samples between the 25 ns pulses in the calibration signal is used to measure the sampling interval. This measurement will be performed by the DSP in the radar when the system initializes.

The time of arrival of the radar pulse, and the known distance from the wall is used to find the time the transmitter fired. The location of the first rising edge of
the calibration unit is located. The offset between this starting point in the cal. data and the time the transmitter fired in the radar return is recorded for later use. With this offset, the transmitter firing position can be located by processing the cal. unit signal by the DSP when the system is initialized. This procedure is necessary due to possible drift in the analog electronics. The drift leads to uncertainty in the radar range window length (changing the sample interval) and the start time of the transmitter. The transmitter start time is not required for snow thickness measurement, but is it required for the measurement of flying height.

2.2.6 Ice Probe Software Modifications.

The Bird computer software was modified to use an ACL multi-port serial card as all of the on-board serial ports were already in use. The bird computer packages the snow thickness results in a special embedded-packet for logging by the DataServer. When the Bird receives a command to have the EM system do a Q-coil, a message is be passed to the radar system for it to collect a new background average.

The data server logs the radar data (as embedded packets) but it does not provide a real-time display or chart record. After a flight, the radar results were extracted from the file created by the data server for analysis and plotting. The embedded packets are time-stamped for synchronization with the rest of the data logged with Ice Probe.

Several other features were added during the field work. The changes were to allow the recording of raw radar waveforms and the upload of the DSP program when the system initializes.

These Bird computer software modifications have not yet been fully debugged.

2.3 Field Work

A log of field activities is shown in Appendix A.

The radar field trial began while the newly rebuilt Ice Probe system was still being debugged. Ice Probe software that was to log the real-time snow thickness radar results was not working properly. To view raw radar waveforms during flight, extra cables were added to Ice Probe's tow cable to bring analog data from the Radar up to an analog-to-digital converter card in a laptop computer. When flight tested, the analog data was too noisy to see any radar echoes from the ground of ice surface. At this time it was thought that the noise was due to problems with the interface to the ADC card.
New software was added to radar to have it upload raw radar waveforms for storage with the \textit{Ice Probe} logging system. Problems continued with the logging of radar data with the \textit{Ice Probe} logging system. To get around this problem, the cables that were added to bring analog signals up to the helicopter were modified to bring the serial data directly up to the helicopter cabin. The EZKit DSP EPROM with the radar program was replaced with Analog Devices ‘Monitor’ program. The Monitor program allows uploading of the DSP program over the serial port, allowing quick program updates (even while flying) as a new EPROM did not have to be programmed each time the program was changed.

The serial data rate was increased to 115 kbaud from 9600 baud. This enabled the download of single raw waveforms in about one second. During the next test flight raw waveforms could be acquired over specific areas of interest.

On the afternoon of Saturday March 22, a test flight was made logging real-time and raw waveforms on a laptop computer. Figure 11 shows a plot of Bird height versus time for the real-time radar processing output and the laser altimeter output. This data was collected over the airport runway at Charlottetown. The laser and radar return follow quite closely, but the radar return can be seen to alternate between two heights.

Analysis of the real-time results and the raw waveform showed that the polarity of the radar signal was opposite of what was expected and the DSP code was changed accordingly. Also, the DSP code was modified to provide a continuous download of the digitized raw radar waveforms at a rate of about 1 Hz.

Flights were made over ice and open water on March 24 and 25. Flight notes and fid synchronization information is shown in Appendix B. For the March 25 data, an additional modifications to the DSP code allowed the upload of stacked raw waveforms. This modification was made to check whether or not stacking would make a noticeable improvement in the noise levels. It appeared that the stacking did not make an improvement, indicating that the noise problem was not with random noise but with the radar control circuitry.
3. Conclusions and Recommendations

Three main conclusions can be stated:

1. The small radar package has been modified to add on-board digitization and signal processing capability.
2. Software was developed to process the incoming data in real-time and provide an output over a standard serial interface.
3. Test flying the system showed that the radar was able to return real-time results for flying height, with noisy returns for snow thickness.

The first two were successes, the third was partially successful. The achievement of the first two represent overall success as these points were the technically risky items of the new development. The third represents the first time an airborne ground penetrating radar has provided a real-time output of a processed result. The real-time implementation of the snow thickness processing algorithm appears to work. Though the input to the processing was noisy, the surface echo was detected and the correct flying height was returned. Further work with the radar hardware is required to improve the snow thickness results.

The noise source may be in the power supply or perhaps in the DSP generated timing ramps. The EZKit board had to be modified to work around its AC coupling of the DAC output signal. There may be a problem with a floating ground in this circuit.

To date, data analysis has been limited to the work performed in the field. Conclusions about the noise levels were made in the field and it does not appear to be worthwhile to spend more time analyzing the March ‘97 data set at this time.

The performance of the radar in the presence of the EM system (Ice Probe) has not been fully tested. Extra analog or digital filtering may be required for accurate snow thickness measurements when operated with the EM system.

For the 1997 flight tests, the radar was mounted inside the Ice Probe bird. In the future, the radar could be mounted in the Ice Probe bird, in its own bird or hard-mounted to the helicopter. It is likely that the packaging can be made to be in the format that is most useful for Coast Guard’s helicopter operations.

The capability of this system to do real-time processing lends itself to other uses. A complimentary application of this radar, either alone or with Ice Probe, would be the measurement of fresh water ice thickness in the St. Lawrence River.
The following items are recommended for future work:

1. Re-assemble the 1992 St. Anthony configuration to provide a comparison benchmark.
2. Debug the power supply.
3. Debug the DSP output conditioning circuitry.
4. Replace EZKit-Lite DSP board with newly released board.
5. Finish debugging the logging of radar data with Ice Probe.
6. Field test if the system performs as well as the St. Anthony system.
7. Evaluate the system for fresh water ice thickness measurement.

The configuration used in St. Anthony in 1992 should be re-assembled. Data should be collected from a building wall as was done with the current system (results shown in Figure 9). This will provide a benchmark noise level that the new system must meet before further field testing is performed.

The new power supply needs to be debugged and its noise level should be compared with the one used in St. Anthony in 1992.

The ground level of the DSP output conditioning circuitry appear to float. This circuits needs further work. Filtering may be required for the DSP generated slow ramp as (at present) the only filtering is within the DAC chip. The ramp stability should be checked.

Analog Devices has recently release a floating point version of their EZKIT DSP board with the 21061 DSP chip, stereo ADC, stereo DAC and serial output. This board replicates many of the features of the EZKit-Lite board but the implementation is better for this application. The floating point DSP would be better suited for the implementation of digital filters that are probably necessary to filter out noise from the EM equipment in Ice Probe. The serial interface is implemented using hardware components (a UART chip) instead of software.

Re-integration of the Radar back into Ice Probe may require a small amount of further debugging. Thought the cost for this activity would be small, having the access to the Bird for debugging may be difficult.

Field testing would only be performed if the system performs as well as the St. Anthony system. It is estimated that field work be for a one week period for two people plus travel.

Table 2 lists the tasks and the estimated costs involved for the recommended work.
Table 1. Pin out for various connectors inside the radar unit.

<table>
<thead>
<tr>
<th>Connector for</th>
<th>Pin</th>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Calibration Box DE-9P</strong></td>
<td>1</td>
<td>+12</td>
<td>+12V power for calibration receiver board</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>GND</td>
<td>ground for calibration receiver board</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-12</td>
<td>-12V power for calibration receiver board</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>cal scan</td>
<td>Calibration scan from calibration receiver</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>GND</td>
<td>(not used at present)</td>
</tr>
<tr>
<td><strong>Receiver</strong></td>
<td>1</td>
<td>+12</td>
<td>+12V power for receiver board</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>GND</td>
<td>ground for receiver board</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>-12</td>
<td>-12V power for receiver board</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>radar scan</td>
<td>Radar scan from receiver board</td>
</tr>
<tr>
<td><strong>EZ-KIT</strong></td>
<td>1</td>
<td>+5</td>
<td>+5V power for EZ-KIT board</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>GND</td>
<td>analog ground for EZ-KIT board</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>cal (att)</td>
<td>attenuated calibration scan for EZ-KIT ADC</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>radar (att)</td>
<td>attenuated radar scan for EZ-KIT ADC</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>ref</td>
<td>reference voltage from EZ-KIT board</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>ramp</td>
<td>ramp from EZ-KIT DAC</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>SOS</td>
<td>start of scan pulse from EZ-KIT DAC</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>clock</td>
<td>~12 MHz clock from EZ-KIT board</td>
</tr>
<tr>
<td><strong>Fast Ramp Board DE-9S</strong></td>
<td>1</td>
<td>+150</td>
<td>+150V power for board</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>+5</td>
<td>+5V power for board</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>+12</td>
<td>+12V power for board</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>GND</td>
<td>ground for board</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>-12</td>
<td>-12V power for board</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>-70</td>
<td>-70V power for board</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>clock</td>
<td>clock for triggering transmitter</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>ramp (amp)</td>
<td>amplified slow ramp signal from EZ-KIT</td>
</tr>
<tr>
<td><strong>Output Connector</strong></td>
<td>1</td>
<td>+28V</td>
<td>+28 V power for power supply</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>GND</td>
<td>ground input for power supply</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>TX</td>
<td>serial port transmit</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>RX</td>
<td>serial port receive</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>GND (digital)</td>
<td>ground for serial port</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>radar analog</td>
<td>radar signal from receiver</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>cal analog</td>
<td>calibration signal from calibration box</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>SOS pulse</td>
<td>start of scan pulse</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>GND (analog)</td>
<td>ground for radar and cal signals</td>
</tr>
</tbody>
</table>
Table 2 - List of the tasks and cost estimates.

<table>
<thead>
<tr>
<th>Labour</th>
<th>Days</th>
<th>Cost ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put St. Anthony system together and test</td>
<td>2</td>
<td>$1,350.00</td>
</tr>
<tr>
<td>Debug power supply</td>
<td>2</td>
<td>$1,350.00</td>
</tr>
<tr>
<td>Rebuild power supply from scratch</td>
<td>5</td>
<td>$3,375.00</td>
</tr>
<tr>
<td>Debug DSP output conditioning circuit</td>
<td>5</td>
<td>$3,375.00</td>
</tr>
<tr>
<td>Replace fixed point DSP board with floating point DSP</td>
<td>5</td>
<td>$3,375.00</td>
</tr>
<tr>
<td>Ground test system</td>
<td>6</td>
<td>$4,050.00</td>
</tr>
<tr>
<td>Field work 2 weeks</td>
<td>10</td>
<td>$6,750.00</td>
</tr>
<tr>
<td>Write report</td>
<td>3</td>
<td>$2,025.00</td>
</tr>
<tr>
<td><strong>Labour Subtotal:</strong></td>
<td>38</td>
<td>$25,650.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Direct Costs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating point DSP board, with spare and one for development</td>
<td></td>
<td>$1,000.00</td>
</tr>
<tr>
<td>Travel and Living to Charlottetown 2 weeks</td>
<td></td>
<td>$3,000.00</td>
</tr>
<tr>
<td><strong>Direct Costs Subtotal:</strong></td>
<td></td>
<td>$4,000.00</td>
</tr>
</tbody>
</table>

**Total:** $29,650.00
Figure 1. Photograph of the inside components of the ice thickness radar system.
Figure 2. Block diagram of electronic components in the ice thickness radar system. The Analog Devices EZkit-Lite board (not shown) is mounted on the lid of the box.
Figure 3. Wiring diagram of major components
Figure 4. Schematic diagram of power supply
Figure 5. Schematic diagram of D/A converter DC coupled output circuit for slow ramp.

Figure 6. Schematic diagram of attenuator circuit to reduce radar signal levels for into A/D converter.
Figure 7. Schematic diagram of clock divider circuit to convert the 1847 XTAL 1 output of 12.288 MHz. When divided by 256 the output clock rate is 48 kHz. The output clock from this device goes to the sample control board, setting the receiver sampler rate and the p.r.f. (pulse repetition frequency) of the radar transmitter.
Figure 8. Block diagrams of the DSP software.

Main Program

Initialization

Interrupt Service Routines

**CODEC ISR**
- RX scan data
- TX cal data
- CODEC clock (interrupt rate)

**UART ISR**
- send/receive data from TX/RX buffer
Figure 9. A plot of Radar waveform at several distances from a building wall

Figure 10. A plot of the calibration waveform.
Figure 11. A comparison of the real-time radar bird height and the laser altimeter.
Appendix A - Log of Field Activities

The personnel involved with the snow thickness radar work were Louis Lalumiere and Santanu Paul, both of Aerodat Inc.

Monday, March 17

Travel from Toronto to Charlottetown

SP sets up computer in hotel room.

Tuesday, March 18

*Ice Probe* being debugged. Working on latest mods to Radar DSP program. Do video flight.

Wednesday, March 19

*Ice Probe* being debugged. Debugging latest mods to Radar DSP program. Do video flight. PM - finish debugging A/D logging program on laptop.

Thursday, March 20

Fly *Ice Probe* with radar in morning. Radar data not coming up through Bird link. In afternoon add extra wires to tow cable to bring analog signals to helicopter. Fly *Ice Probe* (with radar) in afternoon and log analog radar data. The analog radar data is too noisy.

Friday, March 21

Weather down in morning. Work on serial port waveform downloading. Work briefly on analog noise problem with radar. Work on compiling and running *Ice Probe* post-flight inversion routines. Do video over fixed link and out to pancake ice north of PEI.

Saturday, March 22

Rebuild radar after damage cause by Friday afternoon *Ice Probe* flight. Change wiring to bring serial data directly from the radar up to laptop in helicopter. Do a radar flight. Look at radar data - change DSP processing and increase serial data rate to 115 kbaud. Do test flight over airport runway in snowy weather. Work on binary logging of 115 kbaud serial data using terminal program.

Sunday, March 23
Continue debug of binary logging of 115k baud serial data using terminal program while *Ice Probe* is being repaired. Andy Maillet operates *Ice Probe* with radar logging performed separately - Radar data seems noisy with EM on. Second flight radar only - no markers indicating where they were or what they were flying over - problem with waveform logging part of DSP program, still looking at positive peak - real-time processing alternating between the 2 lower amplitude positive peaks - should look at single negative peak.

Do video flight over the Northumberland Strait.

Monday, March 24

Change radar DSP code to pick negative peak. Debug continuous raw waveform logging. Morning flight with EM and radar - had string to turn on EM transmitter when radar logging was complete. EM debugged all afternoon - new laser rejection code added to bird program. Look at radar data. Spend two hours on VideoGPS playback program.

Tuesday, March 25

Stacked radar output debugged. Fly radar over harbour over ice and open water. Fly EM over cal lines and over ridge line on the north side of PEI. Set up Video/GPS laptop for Simon Prinsenberg to use that afternoon on a video flight. Depart for Toronto.
<table>
<thead>
<tr>
<th>Date</th>
<th>Ice Probe Filename</th>
<th>Radar Filename</th>
<th>Ice Probe Fid Number</th>
<th>Radar Hex Number</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>March 24</td>
<td>FLT008</td>
<td></td>
<td></td>
<td></td>
<td>Use <em>Ice Probe</em> logging system to record GPS positions and laser altimeter data. EM transmitter off.</td>
</tr>
<tr>
<td>AAA1.txt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Free run raw waveforms</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(test fids)</td>
</tr>
<tr>
<td>2</td>
<td>FDB0</td>
<td></td>
<td></td>
<td></td>
<td>Along East River</td>
</tr>
<tr>
<td>3 and 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(test fids)</td>
</tr>
<tr>
<td>5</td>
<td>1279</td>
<td></td>
<td></td>
<td></td>
<td>go up to 250 feet</td>
</tr>
<tr>
<td>6</td>
<td>13FF</td>
<td></td>
<td></td>
<td></td>
<td>Beyond power lines East River</td>
</tr>
<tr>
<td>7</td>
<td>1535</td>
<td></td>
<td></td>
<td></td>
<td>Head back up to alt.</td>
</tr>
<tr>
<td>8</td>
<td>18C5</td>
<td></td>
<td></td>
<td></td>
<td>At alt. before going back down</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>False fid</td>
</tr>
<tr>
<td>19C4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Free run waveforms ends</td>
</tr>
<tr>
<td>AAA3.txt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Real-time processing. Start point set to 400</td>
</tr>
<tr>
<td>10</td>
<td>0FD0</td>
<td></td>
<td></td>
<td></td>
<td>Run over landfast ice north of PEI</td>
</tr>
<tr>
<td>11</td>
<td>0AE0</td>
<td></td>
<td></td>
<td></td>
<td>“ (photo 27)</td>
</tr>
<tr>
<td>12</td>
<td>0FFF</td>
<td></td>
<td></td>
<td></td>
<td>“</td>
</tr>
<tr>
<td>13</td>
<td>1300</td>
<td></td>
<td></td>
<td></td>
<td>300 feet</td>
</tr>
<tr>
<td>AAA4.txt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Change processing start point to 430. In Tracadie Bay</td>
</tr>
<tr>
<td>14</td>
<td>29D0</td>
<td></td>
<td></td>
<td></td>
<td>On snow line heading west</td>
</tr>
<tr>
<td>15</td>
<td>2C20</td>
<td></td>
<td></td>
<td></td>
<td>over line</td>
</tr>
<tr>
<td>16</td>
<td>2E10</td>
<td></td>
<td></td>
<td></td>
<td>background done</td>
</tr>
<tr>
<td>17</td>
<td>3160</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Near shore</td>
</tr>
<tr>
<td>19</td>
<td>3300</td>
<td></td>
<td></td>
<td></td>
<td>snow line heading east</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>near men and trucks on ice</td>
</tr>
<tr>
<td>21</td>
<td>3540</td>
<td></td>
<td></td>
<td></td>
<td>end logging real-time results</td>
</tr>
<tr>
<td>AAA5.txt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Log raw waveforms</td>
</tr>
<tr>
<td>22</td>
<td>3870</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>3965</td>
<td></td>
<td></td>
<td></td>
<td>Snow line E-W</td>
</tr>
<tr>
<td>24</td>
<td>3A10</td>
<td></td>
<td></td>
<td></td>
<td>at alt.</td>
</tr>
<tr>
<td>25</td>
<td>3AF0</td>
<td></td>
<td></td>
<td></td>
<td>low alt</td>
</tr>
<tr>
<td>26</td>
<td>3B43</td>
<td></td>
<td></td>
<td></td>
<td>on line W-E</td>
</tr>
<tr>
<td>27</td>
<td>3B71</td>
<td></td>
<td></td>
<td></td>
<td>over trucks</td>
</tr>
<tr>
<td>AAA6.txt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Radar waveforms with EM on.</td>
</tr>
<tr>
<td>Date</td>
<td>Ice Probe Filename</td>
<td>Radar Filename</td>
<td>Ice Probe Fid Number</td>
<td>Radar Hex Number</td>
<td>Comment</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------</td>
<td>----------------</td>
<td>----------------------</td>
<td>------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>March 25</td>
<td>FLT011</td>
<td></td>
<td></td>
<td></td>
<td>Use <em>Ice Probe</em> logging system to record GPS positions and laser altimeter data. EM transmitter off.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BBB1.txt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 1D11</td>
<td></td>
<td></td>
<td>Logging stacked raw waveforms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 1D42</td>
<td></td>
<td></td>
<td>Low over east river</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 2062</td>
<td></td>
<td></td>
<td>up over bridge heading to outer harbour</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 2502</td>
<td></td>
<td></td>
<td>low again</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 2682</td>
<td></td>
<td></td>
<td>grey ice</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 2712</td>
<td></td>
<td></td>
<td>open water with small ice floes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 2972</td>
<td></td>
<td></td>
<td>just at end of open water - lots of small floes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 2AC2</td>
<td></td>
<td></td>
<td>solid ice then go up</td>
</tr>
<tr>
<td></td>
<td>BBB2.txt</td>
<td></td>
<td></td>
<td></td>
<td>real-time output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9 3229</td>
<td></td>
<td></td>
<td>low over ice</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 3430</td>
<td></td>
<td></td>
<td>onto grey ice</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 3600</td>
<td></td>
<td></td>
<td>open water with some traces of ice</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12 3910</td>
<td></td>
<td></td>
<td>swing left towards ice - still over open water</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13 D3B5?</td>
<td></td>
<td></td>
<td>Solid ice</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14 3CE0</td>
<td></td>
<td></td>
<td>open water with some solid ice floes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15 3E40</td>
<td></td>
<td></td>
<td>solid ice</td>
</tr>
<tr>
<td></td>
<td></td>
<td>41B0</td>
<td></td>
<td></td>
<td>flat floe</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4320</td>
<td></td>
<td></td>
<td>small open water</td>
</tr>
<tr>
<td></td>
<td></td>
<td>44B0</td>
<td></td>
<td></td>
<td>Flat floe then go up</td>
</tr>
</tbody>
</table>
Appendix C - Radar Processing Implemented in 'C' for DOS.

// snowth.c - Snow thickness radar processing

#include <stdio.h>
#include <stdlib.h>

/*********************************************/
/* Structure used for passing parameters */
/*********************************************/
typedef struct {
  int  wave_size;
  int  skip;
  int  threshold;
  short *bgnd_ptr;
  short *fil_bpf;
  short bpf_len;
  short *fil_bpf2;
  short bpf2_len;
  short *fil_bpf3;
  short bpf3_len;
  short *fil_lp1;
  short lp1_len;
  short *fil_lp2;
  short lp2_len;
  FILE  *ifp;
  FILE  *ofp;
  FILE  *bfp;
} param;

/*********************************************/
/* Function Declarations */
/*********************************************/
int get_parameters(char *par_filename, param *info);
long get_file_size(FILE *fp);
int snowth(char *par_file);
int load_filt(char * file_name, short **filter, short *fil_len);
void sub_background(param *info, short *input, short *output);
void diff4(param *info, short *input, short *output);
void loc_peaks(param *info, short *input, short *data, short *output);
void thresh_peak(param *info, short *input, short *thresh_value, short *thresh_index) ;
void find_max(param *info, short *input, short *max_value, short *max_index);
void fir_filter(param *info, short *filter, short filt_len, short *input, short *output);
short cont_fir_filter(short *filter, short filt_len, short data, short *buffer);
void cleanup(param *info);

/*********************************************/
/* Main Program */
/*********************************************/
int main(int argc, char *argv[])
{
  int err_val;

  err_val = 1;
  if (argc != 2)
printf("Usage: snowth <parameter_file> ");
else
    { printf("\nSnow Thickness Radar Processing\n ");
        err_val = snowth( argv[1] );
        printf("\nDone. \n ");
    }

    return err_val;
}

/*************************************************************************
 * Main processing routine *
 *************************************************************************/
int snowth(char *par_file)
{
    int err_val;
    long file_size, data_read;
    float percent, prev_percent;
    #if (0)
        short *raw_data, *background, *bgnd_out, *bpf_out;
        short *bpf2_out, *bpf3_out, *diff_out, *peak_out;
    #else
        short raw_data[1024], background[1024], bgnd_out[1024],
            bpf_out[1024];
        short bpf2_out[1024], bpf3_out[1024], diff_out[1024],
            peak_out[1024];
    #endif
    short max_val, max_index, thresh_val, thresh_index;
    short max_index_filt, thresh_index_filt, thickness;
    static short lpl_buff[200], lp2_buff[200];
    param par;

    err_val = get_parameters(par_file, &par);

    if (err_val == 0)
    {
        #if (0)
            if ( (raw_data=malloc(par.wave_size, sizeof(short))) == NULL )
                { printf("Error: Unable to allocate memory for raw data.\n ");
                    err_val = 5;
                }
            else if ( (background=malloc(par.wave_size, sizeof(short))) == NULL )
                { printf("Error: Unable to allocate memory for background
waveform. \n ");
                    err_val = 5;
                }
            else if ( (bgnd_out=malloc(par.wave_size, sizeof(short))) == NULL )
                { printf("Error: Unable to allocate memory for background
subtraction output buffer. \n ");
                    err_val = 5;
                }
            else if ( (bpf_out=malloc(par.wave_size, sizeof(short))) == NULL )
                { printf("Error: Unable to allocate memory for band-pass filter
output buffer. \n ");
                    err_val = 5;
                }
        
        }
else if ( (bpf2_out = calloc(par.wave_size, sizeof(short))) == NULL)
{ printf("Error: Unable to allocate memory for band-pass filter 2 output buffer.\n");
  err_val = 5;
} else if ( (bpf3_out = calloc(par.wave_size, sizeof(short))) == NULL)
{ printf("Error: Unable to allocate memory for band-pass filter 3 output buffer.\n");
  err_val = 5;
} else if ( (diff2_out = calloc(par.wave_size, sizeof(short))) == NULL)
{ printf("Error: Unable to allocate memory for difference filter output buffer.\n");
  err_val = 5;
} else if ( (peak2_out = calloc(par.wave_size, sizeof(short))) == NULL)
{ printf("Error: Unable to allocate memory for peak value output buffer.\n");
  err_val = 5;
} else
  #else
  if (1)
  #endif
  { 
    fread(background, sizeof(short), par.wave_size, par.bfp);
    if (feof(par.bfp))
      printf("Warning: background waveform file doesn't contain enough data.\n");
    fread(background, sizeof(short), 1, par.bfp);
    if (!feof(par.bfp))
      printf("Warning: background waveform file contains too much data.\n");
    fclose(par.bfp);
    par.bgnd_ptr = background;

    file_size = get_file_size(par.ifp);
    percent = prev_percent = 0.0f;
    data_read = 0;
    fread(raw_data, sizeof(short), par.wave_size, par.ifp);
    do
    { 
      data_read += par.wave_size * sizeof(short);
      percent = 100.0f * (float) data_read / (float) file_size;
      if ( percent >= prev_percent + 10.0 )
      { prev_percent = percent;
        printf("%3.0f%% ",percent);
      }
      sub_background(&par, raw_data, bgnd2_out);
      fir_filter(&par, par.fil_bpf, par.bpf_len, bgnd2_out, bpf2_out);
      fir_filter(&par, par.fil_bpf2, par.bpf2_len, bpf2_out, bpf2_out);
      fir_filter(&par, par.fil_bpf3, par.bpf3_len, bpf3_out, bpf3_out);
      diff4(&par, bpf2_out, diff_out);
      loc_peaks(&par, diff_out, bgnd2_out, peak2_out);

    }
find_max(&par, peak_out, &max_val, &max_index);
thresh_peak(&par, peak_out, &thresh_val, &thresh_index);
  fwrite(peak_out, sizeof(short), par.wave_size, par.ofp);
  max_index_filt = cont_fir_filter(par.fil_lp1, par.lp1_len, max_index, lp1_buff);
  thresh_index_filt = cont_fir_filter(par.fil_lp2, par.lp2_len, thresh_index, lp2_buff);

  thickness = max_index_filt - thresh_index_filt;
  if (thickness<0) thickness = 0;
#endif
#endif
}

get_file_size(FILE *fp)
{ long file_size;
  fpos_t file_pos;
  rewind(fp);
  fseek(fp, 0, SEEK_END);
  fgetpos(fp, &file_pos);
  rewind(fp);
  file_size = (long) file_pos;
  return file_size;
}

get_parameters(char *par_file, param *info)
{ FILE *pf;
  char cbuff[200], in_file[80], out_file[80], back_file[80];
char bpf_file[80], bpf2_file[80], bpf3_file[80], lpl_file[80], lp2_file[80];

if ( (pfp=fopen(par_file,"r") == NULL) {
    printf("\nError: Unable to open parameter file.\n\n");
    return 1; }
    fgets(cbuff, 200, pfp);
    fgets(cbuff, 200, pfp);
    fgets(cbuff, 200, pfp);
    fgets(cbuff, 200, pfp);
    fgets(cbuff, 200, pfp);
    fgets(cbuff, 200, pfp);
    fgets(cbuff, 200, pfp);
    fgets(cbuff, 200, pfp);
    fgets(cbuff, 200, pfp);
    fgets(cbuff, 200, pfp);
    sscanf(cbuff, "%s", in_file);
    sscanf(cbuff, "%s", out_file);
    sscanf(cbuff, "%s", back_file);
    sscanf(cbuff, "%s", bpf_file);
    sscanf(cbuff, "%s", bpf2_file);
    sscanf(cbuff, "%s", bpf3_file);
    sscanf (cbuff, "%s", lpl_file);
    sscanf(cbuff, "%d", &info->wave_size);
    sscanf(cbuff, "%d", &info->skip);
    sscanf(cbuff, "%d", &info->threshold);
    fclose(pfp);
    if ( (info->ifp=fopen(in_file,"rb") == NULL) {
        printf("\nError: Unable to open input file.\n\n");
        return 2; }
    if ( (info->ofp=fopen(out_file,"wb") == NULL) {
        printf("\nError: Unable to open output file.\n\n");
        return 2; }
    if ( (info->bfp=fopen(back_file,"rb") == NULL) {
        printf("\nError: Unable to open background waveform file.\n\n");
        return 2; }
    if ( load_filt(bpf_file, &info->fil_bpf, &info->bpf_len) != 0 )
        return 3;
    if ( load_filt(bpf2_file, &info->fil_bpf2, &info->bpf2_len) != 0 )
        return 3;
    if ( load_filt(bpf3_file, &info->fil_bpf3, &info->bpf3_len) != 0 )
        return 3;
    if ( load_filt(lpl_file, &info->fil_lpl, &info->lpl_len) != 0 )
        return 3;
    if ( load_filt(lp2_file, &info->fil_lp2, &info->lp2_len) != 0 )
        return 3;
    return 0;
}

/*****************************************************************
* Load filter from file
* *****************************************************************/
int load_filt(char *file_name, short **filter, short *fil_len)
{
    short *filt_buff, filt_buff_len, err;
    FILE *fp;
    if ( (fp=fopen(file_name,"rb") == NULL) {
        printf("\nError: Unable to open filter file: %s\n",
    file_name);
        filt_buff = NULL;
        filt_buff_len = 0;
        err = 1;
    }
else
    { filt_buff_len = get_file_size(fp) / sizeof(short);
      if ( (filt_buff=calloc(filt_buff_len, sizeof(short))) == NULL )
      {  filt_buff_len = 0;
          err = 2;
        }
    else
      { fread(filt_buff, filt_buff_len, sizeof(short), fp);
          err = 0;
      }
    fclose(fp);
    }

*filter = filt_buff;
*fil_len = filt_buff_len;
return err;
}

/*******************************************************************************/
* Subtract background waveform                                             *
*******************************************************************************/
void sub_background(param *info, short *input, short *output)
{  int i;
    for (i=0; i<info->wave_size; i++)
      output[i]=input[i]-info->bgnd_ptr[i];
}

/*******************************************************************************/
* Difference Filter                                                        *
*******************************************************************************/
void diff4(param *info, short *input, short *output)
{  int i;
    output[0] = 0;
    output[1] = 0;
    output[info->wave_size-1] = 0;
    output[info->wave_size-2] = 0;
    for (i=2; i<info->wave_size-2; i++)
      output[i] = -input[i-2] - input[i-1] + input[i+1] + input[i+2];
}

/*******************************************************************************/
* Apply FIR filter to waveform                                              *
*******************************************************************************/
void fir_filter(param *info, short *filter, short filt_len, short *input, short *output)
{  int grp_delay, i, j;
    long filt_sum;
    grp_delay = (filt_len - 1)/2;
    for (i=0; i<grp_delay; i++)
    {  output[i] = 0;
        output[info->wave_size-i-1] = 0;
    }
for (i=grp_delay; i<info->wave_size-grp_delay; i++)
{
    filt_sum = 0;
    for (j=0; j<filt_len; j++)
    {
        filt_sum += (long)input[i-grp_delay+j] * (long)filter[filt_len-j-1];
    }
    filt_sum = filt_sum >> 15;
    if ((filt_sum >= 32768) || (filt_sum < -32768))
    {
        printf("*");
        output[i] = (short)((float)filt_sum + 0.5);
    }
}

/**************************************************************
* Apply FIR filter to continuous data, for one data value       *
/***************************************************************/
short cont_fir_filter(short *filter, short filt_len, short data, short *buffer)
{
    short i;
    long filt_sum;

    /* move data back */
    for (i=0; i< filt_len-1; i++)
    {
        buffer[i] = buffer[i+1];
    }

    buffer[filt_len-1]=data; /* load new data value */

    /* apply filter */
    filt_sum = 0;
    for (i=0; i<filt_len; i++)
    {
        filt_sum += buffer[i] * filter[filt_len-i-1];
    }
    filt_sum = filt_sum >> 15;
    if ( (filt_sum >= 32768) || (filt_sum < -32768) )
    {
        printf("*");
    }
    return (short)((float)filt_sum + 0.5);
}

/**************************************************************
* Locate Peaks                                               *
/***************************************************************/
void loc_peaks(param *info, short *input, short *data, short *output)
{
    int i;

    for (i=0; i<=info->skip; i++)
    {
        output[i] = 0;
    }

    for (i=1+info->skip; i<info->wave_size; i++)
    {
        if ( input[i-1] > 0 )
        {
            if ( input[i] <= 0 )
            {
                output[i] = data[i];
            }
        }
        else
        {
            output[i] = 0;
        }
    }
}

32
{  if ( input[i] >= 0 )
    output[i] = data[i];
  else
    output[i] = 0;
}

/*********************************************************************/
/* Find maximum value and position in vector                       */
/*********************************************************************/
void find_max(pararn *info, short *input, short *max_value, short *
max_index)
{  int i;
    *max_value = input[0];
    *max_index = 0;
    for (i=0; i<info->wave_size; i++)
    {  if ( abs(input[i]) > *max_value )
        {  *max_value = input[i];
            *max_index = i;
        }
    }
}

/*********************************************************************/
/* Find first value over threshold in vector                      */
/*********************************************************************/
void thresh_peak(pararn *info, short *input, short *thresh_value, short *
thresh_index)
{  int i;
    *thresh_value = 0;
    *thresh_index = 0;
    for (i=0; i<info->wave_size; i++)
    {  if ( abs(input[i]) >= info->threshold )
        {  *thresh_value = input[i];
            *thresh_index = i;
            i=info->wave_size;
        }
    }
}

/*********************************************************************/
/* Exit routine                                                   */
/*********************************************************************/
void cleanup(pararn *info)
{  free(info->fil_bpf);
    free(info->fil_lpl);
    free(info->fil_lp2);
    fclose(info->ifp);
    fclose(info->ofp);
}
Appendix D - Radar Processing Implemented for the 2181 DSP

```assembly
.module/RAM/ABS=0     snow_thickness_radar;

.include   <system.k>;
#include    "ezk_vars.ini";
.extern    init_uart;    { UART initialize baudrate etc. }
.extern    out_char_ax1;  { UART output a character }
.extern    turn_rx_on;    { UART enable the rx section }
.extern    process_a_bit; { timer ISR for UART }
.extern    flag_rx_no_word;
.extern    get_char_axl_to;
.extern    get_char_axl;
.extern    out_int_ar;
.extern    get_int_ar_to;
.extern    dndsp;
.extern    sendLine;
.extern    stop_uart;
#if (0)
.extern    irq1ISR;
.extern    next_cmd;
.extern    irq0ISR;
.extern    sport_rx;

.entry int_to_ascii_hex;
.global num_string;
#endif

*******
Defines for conditionaly assembly:
    SIMULATOR - call sport_rx in main loop for faster simulations in sim2181.exe
    DEBUG - show waveform values as incrementing numbers when in binary mode
    DEBUG_MEM - log binary mode values in program memory buffer
    FAKE_DATA - use data from fake_datal buffer as the DAC input
    SKIP_ONE - skip point in waveform
Defines for constants:
    WAVE_SIZE - size of radar scan
    OUT_BUFF - size filter buffers for ice thickness and altitude (not used)
    BUFF_HEADER - size of header for buffer structures
    NULL_HEADER - size of header for nulling buffer structure
    ACK_SIZE - size of acknowledge packet string
*******

#define SIMULATOR 0
#define DEBUG 0
#define DEBUG_MEM 0
#define KLUDGE 0
#define FAKE_DATA 0
#define WAVE_SIZE 1023
#define SKIP_ONE 1
#define OUT_BUFF 10
```

34
#define BUFF_HEADER 4
#define NULL_HEADER 3
#define ACK_SIZE 20

{*******************************************************************************
*******
*          
** Variables
*       
*******************************************************************************
******}

#if (DEBUG_MEM == 1)
{*******************************************************************************
* Debug Variables
*******************************************************************************}
.var/pm/ram/circ debug_mem[3000];
.var/dm/ram debug_ptr;
.var/dm/ram debug_count;
#endif

{*******************************************************************************
* Parameters
*******************************************************************************}
.var/dm/ram codec_delay; { delay between DAC/ADC }
.init codec_delay: 0;
.var/dm/ram stack_size; { size of stack for raw radar scans }
.init stack_size: 1;
.var/dm/ram stack_shift; { stack is scaled by shifting }
.init stack_shift: 0;
.var/dm/ram null_stack; { size of nulling stack }
.init null_stack: 32;
.var/dm/ram null_shift; { size of nulling stack }
.init null_shift: -5;
.var/dm/ram cal_thresh; { calibration peak detection threshold - not used, calibration not implemented }
.init cal_thresh: 5000;
.var/dm/ram rad_thresh; { snow peak threshold }
.init rad_thresh: 350; (50/600)
.var/dm/ram cal_tol; { calibration tolerance - not used, calibration not implemented }
.init cal_tol: 10;
.var/dm/ram tx_offset; { offset between TX fires and receiver detects it - not used }
.init tx_offset: 10;
.var/dm/ram rad_transient; { transient period for TX to die down }
.init rad_transient: 430; (500/360)
.var/dm/ram sos_on_time; { Start of scan duty cycle in samples per waveform }
.init sos_on_time: 5;
.var/dm/ram ramp_start; { Start value for ramp }
.init ramp_start: 0;
.var/dm/ram ramp_inc; { Increment/Decrement value for ramp }
.init ramp_inc: -16;
.var/dm/ram ramp_val; { Current ramp value }
Counters for waveform output

.var/dm/ram line_count; (line/packet counter for waveform output)
.var/dm/ram col_count; (column count for waveform output)
.var/dm/ram temp_count; (temporary counter)

CODEC Rate Parameter

.var/dm/ram codec-param;
.init codec-param: 2; (used to set CODEC rate)

Command Parameters

.var/dm/ram command_string [50]; (command input buffer)
.var/dm/ram command_flag; (says whether in the process of receiving a packet command)
.var/dm/ram command_letter;
.var/dm/ram poll_cnt[2]; (command poll counter)
.var/dm/ram wave_cnt[2]; (wave counter)
.var/dm/ram stack_cnt; (stacked waveform counter)
.init stack_cnt: 0;

Initialization string

.var/dm/ram init_string[100]; (initialization string)
.init init_string: '$', 'A', 'D', '-', ...

Invalid command response

.var/dm/ram invalid_string[9]; (invalid command response)
.init invalid_string: '$', 'A', 'D', 'i', 'n', 'v', '0', 't', 'w', 'a', 'r', 'e', 't', 'w', 'a', 'r', 'e', '

Not ready to output waveform

.var/dm/ram not_ready_string[20]; (not ready to output waveform)
.init not_ready_string: '$', 'A', 'D', 'e', 'r', 'r', '-', ...

Acknowledge strings

.var/dm/ram ack_strings[400]; (acknowledge that command was received)
.init ack_strings: '$', 'A', 'D', '0', '0', 'a', '-', ...

36
```

{****************************************************************
* Command strings
****************************************************************}

.var/dm/ram comm_help[9]; { help }
.init       comm_help: '$', 'A', 'D', '0', '0', 'h',13,10,0;

.var/dm/ram comm_auto[9]; { auto output }
.init       comm_auto: '$', 'A', 'D', '0', '0', 'a',13,10,0;

.var/dm/ram comm_back[9]; { update background waveform }
.init       comm_back: ' $', 'A', 'D', '0', '0', 'b',13,10,0;

.var/dm/ram comm_back2[9]; { update background waveform, display
  background/calibration waveforms }
.init       comm_back2: ' $', 'A', 'D', '0', '0', 'B',13,10,0;

.var/dm/ram comm_back_wave[9]; { show background waveform }
.init       comm_back_wave: '$', 'A', 'D', '0', '0', 's',13,10,0;

.var/dm/ram comm_debug[9]; { switch to debug output }
.init       comm_debug: '$', 'A', 'D', '0', '0', 'd',13,10,0;

.var/dm/ram comm_normal[9]; { switch to normal output }
.init       comm_normal: '$', 'A', 'D', '0', '0', 'n',13,10,0;

.var/dm/ram comm_toggle[9]; { toggle auto/prompted output }
.init       comm_toggle: '$', 'A', 'D', '0', '0', 't',13,10,0;

.var/dm/ram comm_prompt[9]; { switch to prompted output }
.init       comm_prompt: '$', 'A', 'D', '0', '0', 'p',13,10,0;

.var/dm/ram comm_reset[9]; { reset software }
.init       comm_reset: '$', 'A', 'D', '0', '0', 'r',13,10,0;

.var/dm/ram comm_status[9]; { show status message }
.init       comm_status: '$', 'A', 'D', '0', '0', 'i',13,10,0;

.var/dm/ram comm_stop[9]; { exit software to monitor }
.init       comm_stop: '$', 'A', 'D', '0', '0', 'x',13,10,0;

.var/dm/ram comm_cb_wave[9]; { show cal and background waveforms }
.init       comm_cb_wave: '$', 'A', 'D', '0', '0', 'c',13,10,0;

.var/dm/ram comm_menu1[9]; { set stack size to 1 }
.init       comm_menu1: '$', 'A', 'D', '0', '0', '1',13,10,0;
```
.var/dm/ram  comm_menu2[9];  { set stack size to 8 }
.init  comm_menu2:  '\$', '\A', '\D', '0', '0', '2', '13,10,0;
.var/dm/ram  comm_menu3[9];  { set stack size to 16 }
.init  comm_menu3:  '\$', '\A', '\D', '0', '0', '3', '13,10,0;
.var/dm/ram  comm_menu4[9];  { set codec rate to 8.0 kHz }
.init  comm_menu4:  '\$', '\A', '\D', '0', '0', '4', '13,10,0;
.var/dm/ram  comm_menu5[9];  { set codec rate to 9.6 kHz }
.init  comm_menu5:  '\$', '\A', '\D', '0', '0', '5', '13,10,0;
.var/dm/ram  comm_menu6[9];  { set codec rate to 16.0 kHz }
.init  comm_menu6:  '\$', '\A', '\D', '0', '0', '6', '13,10,0;
.var/dm/ram  comm_menu7[9];  { set codec rate to 27.4 kHz }
.init  comm_menu7:  '\$', '\A', '\D', '0', '0', '7', '13,10,0;
.var/dm/ram  comm_menu8[9];  { set codec rate to 32.0 kHz }
.init  comm_menu8:  '\$', '\A', '\D', '0', '0', '8', '13,10,0;
.var/dm/ram  comm_menu9[9];  { set codec rate to 48.0 kHz }
.init  comm_menu9:  '\$', '\A', '\D', '0', '0', '9', '13,10,0;
.var/dm/ram  comm_dndsp[9];  { download dsp program }
.init  comm_dndsp:  '\$', '\A', '\D', '0', '0', '-', '13,10,0;
.var/dm/ram  comm_param[7];  { set parameter }
.init  comm_param:  '\$', '\A', '\D', '0', '0', 'z', '13,10,0;

*****************************************************************
* Help Screen
*****************************************************************
.var/dm/ram  help_string[500];  { help screen - some new commands are missing from this list }
.init  help_string:  '\$', '\A', '\D', '0', '0', 'a', '-',
' ', 'h', ', 'e', ',', 'l', ',', 'p', ',', '13,10,
' '\$', '\A', '\D', '0', '0', 'h', '-', ',', 'a', ',', '-'
' ', 'a', ',', 'u', ',', 't', ',', 'o', ',', 'u', ',', 't', ',', 'p', ',', 'u', ',', 't', ',', '13,10,
' '\$', '\A', '\D', '0', '0', 'h', '-', ',', 'p', ',-
' ',', 'p', ',', 'r', ',', 'o', ',', 'm', ',', 'p', ',', 't', ',', 'e', ',', 'd', ',', 'o', ',', 'u', ',', 't', ',', 'p', ',', 'u', ',', 't', ',', '13,10,
' '\$', '\A', '\D', '0', '0', 'h', '-', ',', 't', ',', '-'
' ', 'a', ',', 'u', ',', 't', ',', 'o', ',', ',', 'p', ',', 't', ',', 
' ', 't', ',', 'o', ',', 'g', ',', 'g', ',', 'l', ',', 'e', ',', '13,10,
' '\$', '\A', '\D', '0', '0', 'h', '-', ',', 'n', ',', '-'
' ', 'n', ',', 'o', ',', 'r', ',', 'm', ',', 'a', ',', 'l', ',', 
' ', 'o', ',', 'u', ',', 't', ',', 'p', ',', 'u', ',', 't', ',', '13,10,
' '\$', '\A', '\D', '0', '0', 'h', '-', ',', 'd', ',', '-'
' ', 'd', ',', 'e', ',', 'b', ',', 'u', ',', 'g', ',', 
' ', 'o', ',', 'u', ',', 't', ',', 'p', ',', 'u', ',', 't', ',', '13,10,
' '\$', '\A', '\D', '0', '0', 'h', '-', ',', 'c', ',', '-'
' ', 's', ',', 'h', ',', 'o', ',', 'w', ',', 
' ', 'c', ',', 'a', ',', 'l', ',', ',', 'b', ',', 'a', ',', 'c', ',', 'k', ',', 'g', ',', 'r', ',', 'o', ',', 'u', ',', 'n', ',', 'd', ',', 
' ', 'w', ',', 'a', ',', 'v', ',', 'e', ',', 'f', ',', 'o', ',', 'r', ',', 'm', ',', '13,10,
' '\$', '\A', '\D', '0', '0', 'h', '-', ',', 's', ',', '-'
' ', 's', ',', 'h', ',', 'w', ',', 'w', ',', 'b', ',', 'a', ',', 'c', ',', 'k', ',', 'g', ',', 'z', ',', 'o', ',', 'u', ',', 'n', ',', 'd', ',', 
' ', 'w', ',', 'a', ',', 'v', ',', 'e', ',', 'f', ',', 'o', ',', 'r', ',', 'm', ',', '13,10,
' '\$', '\A', '\D', '0', '0', 'h', '-', ',', 'b', ',', '-'
' ', 'u', ',', 'p', ',', 'd', ',', 'a', ',', 't', ',', 'e', ',', 
', 'b', ',', 'a', ',', 'c', ',', 'k', ',', 'g', ',', 'r', ',', 'o', ',', 'u', ',', 'n', ',', 'd', ',', '13,10,
{ 
' '\$', '\A', '\D', '0', '0', 'h', '-', ',', 'f', ',', '-'
' ', 'u', ',', 'p', ',', 'd', ',', 'a', ',', 't', ',', 'e', ',', 
', 'b', ',', 'a', ',', 'c', ',', 'k', ',', 'g', ',', 'r', ',', 'o', ',', 'u', ',', 'n', ',', 'd', ',', '13,10,

'}
{****************************************************************
* Filter coefficients
****************************************************************}
.var/pm/ram fil_bpf[25], fil_lpf[25], fil_diff4[10];
.init fil_bpf: { header: length=21, dec_fac=1, grp_del=10 }
  h#1500, h#100, h#A00,
  h#FEC700, h#00E00, h#07E00, h#E4300, h#FB7800,
  h#FEC600, h#F4600, h#ECD000, h#E7F00, h#19B900,
  h#38E00, h#19B900, h#EE7F00, h#ECD000, h#FD4600,
  h#FEC600, h#FB7800, h#F4300, h#00E00, h#000000,
  h#FEC700;
.init fil_lpf: { same as above }
  h#1500, h#100, h#A00,
  h#FEC700, h#00E00, h#07E00, h#E4300, h#FB7800,
  h#FEC600, h#F4600, h#ECD000, h#E7F00, h#19B900,
  h#38E00, h#19B900, h#EE7F00, h#ECD000, h#FD4600,
  h#FEC600, h#FB7800, h#F4300, h#00E00, h#000000,
  h#FEC700;
.init fil_diff4: { header: length=5, dec_fac=1, grp_del=2 }
  h#500, h#100, h#200,
  h#400000, h#400000, h#000000, h#C00000, h#C00000;
{****************************************************************
* Command/UART input buffer
****************************************************************}
.var/pm/ram/circ command_buff[1024]; { command input buffer }
.var/dm/ram command_ptr; { command buffer pointer }
.init command_ptr: ^command_buff;
.var/dm/ram command_count; { unprocessed command buffer counter }
.init command_count: 0;
.var/dm/ram command_total; { total character count for command buffer }
.init command_total: 0;
{****************************************************************
* Flags
****************************************************************}
.var/dm/ram init_flag;  
{ run initialization code from main loop? }  
.init init_flag: 0;  
.var/dm/ram stop_flag;  
{ exit software to monitor? }  
.init stop_flag: 0;  
.var/dm/ram null_flag;  
{ update nulling waveform? }  
.init null_flag: 0;  
.var/dm/ram cont_flag;  
{ auto (continuous) output? }  
.init cont_flag: 1;  
.var/dm/ram data_off_flag;  
{ in the process of showing waveform? }  
.init data_off_flag: 0;  
.var/dm/ram debug_flag;  
{ debug mode output? }  
.init debug_flag: 0;  
.var/dm/ram show_back_flag;  
{ in the process of updating background waveform? }  
.init show_back_flag: 0;  
.var/dm/ram show_rad_flag;  
{ in the process of showing radar waveform? }  
.init show_rad_flag: 0;  
.var/dm/ram cont_wave_flag;  
{ continuous radar waveform output? }  
.init cont_wave_flag: 0;  
.var/dm/ram bin_flag;  
{ binary format for radar waveform? }  
.init bin_flag: 0;  
.var/dm/ram data_bank;  
{ which data bank to use for radar/cal waveform? (double buffering is used) }  
.init data_bank: 1;

Nguồn: Waveform stacking parameters

(.var/dm/ram stack_size;  
.var/dm/ram stack_shift;)
.var/dm/ram stack_counter;  
{ how many waveforms have been stacked so far? }
.var/dm/ram stack_done;  
{ finished stacking waveforms }

Nguồn: Data bank pointers

.var/dm/ram data_bank_ptr;  
{ pointer to current raw data buffer }
.var/dm/ram cal_bank_ptr;  
{ pointer to current calibration buffer }

Nguồn: Counters

.var/dm/ram counter;  
{ line/waveform counter }
.init counter: 0;
.var/dm/ram sample_count;  
{ waveform sample count, reset after each waveform }
.init sample_count: 0;
I/O Buffers

.var/dm/ram num_string[300]; { string output buffer }
.var/dm/ram raw_data1[BUFF_HEADER]; { raw data buffer header }
.var/dm/ram raw_data2[BUFF_HEADER];
.var/dm/ram cal_data1[BUFF_HEADER];
.var/dm/ram cal_data2[BUFF_HEADER];
.var/dm/ram raw_copy[BUFF_HEADER]; { copy of raw data (header) }
.var/dm/ram bgnd_out[BUFF_HEADER]; { output after background subtraction (header) }
.var/dm/ram bpf_out[BUFF_HEADER]; { output from first FIR filter (header) }
.var/dm/ram bpf2_out[BUFF_HEADER]; { output from second FIR filter (header) }
.var/dm/ram diff_out[BUFF_HEADER]; { differentiation FIR filter output (header) }
.var/dm/ram peak_out[BUFF_HEADER]; { peak detection output (header) }
.var/dm/ram max_out[BUFF_HEADER]; { maximum detected (header) }
.var/dm/ram thresh_out[BUFF_HEADER]; { threshold detected (header) }
.var/dm/ram/circ buff0[WAVE_SIZE];
.var/dm/ram/circ buff1[WAVE_SIZE];
.var/dm/ram/circ buff2[WAVE_SIZE];
.var/dm/ram/circ buff3[WAVE_SIZE];
.var/dm/ram/circ buff4[WAVE_SIZE];
.var/dm/ram/circ buff5[WAVE_SIZE];
.var/dm/ram/circ buff6[WAVE_SIZE];
.var/dm/ram/circ buff7[WAVE_SIZE];
.var/dm/ram/circ buff8[WAVE_SIZE];
.var/dm/ram/circ buff9[WAVE_SIZE];
.var/dm/ram/circ out_buff1[OUT_BUFF];
.var/dm/ram/circ out_buff2[OUT_BUFF];
.var/dm/ram null_data[NULL_HEADER+WAVE_SIZE+1];
#if (0)
.init null_data: <null.dat>;
#endif
#elif (FAKE_DATA == 1)
.global fake_data1;
global fake_data2;
global fake_dptr1;
global fake_dptr2;
.var/dm/ram/circ fake_data1[1024]; { buffer to hold simulated ADC data }
.var/dm/ram fake_dptr1;
.init fake_data1: <scan2.dat>;
.var/dm/ram/circ fake_data2[10];
.var/dm/ram fake_dptr2;
(.init fake_data2: <scan_new.dat>);
#endif
----------------------------- interrupt vector table -----------------------------

int_vec_table:
#if (SIMULATOR == 1)
jump again; rti; rti; rti; {00: reset }
#else
jump start; rti; rti; rti; {00: reset }
#endif

rti; rti; rti; rti;
rti; rti; rti; rti; {08: IRQL1 }
rti; rti; rti; rti; {0c: IRQL0 }
ar = dm(stat_flag);
if eq rti;
jump next_cmd;
jump sport_rx;
        rti; rti; rti; {14: SPORT1 rx }
jump irqesr; rti; rti; rti; {18: IRQE }
        rti; rti; rti; {1c: BDMA }
jump irqlir;
        rti; rti; rti; {20: SPORT1 tx or IRQ1 }
        rti; rti; rti; {2c: power down }
        rti; rti; rti; {28: timer }

#include "start.dsp"

-
again: { any thing from host ?}
ar = dm(init_flag); { do initialization ? }
ar = pass ar;
if ne jump stop_check;
ar=1; dm(init_flag)=ar;
#if (SIMULATOR == 0)
call init_uart;
call turn_rx_on;
#endif
dis ints;
ax0=dm(0x3FFE);
ay0=0xFFF8;
ayl=0x2;
ar=ax0 and ay0;
ar=ar or ay1;
dm(0x3FFE)=ar;
call do_init;
ena ints;
i7="init_string;
call sendLine;          { done initialization }

stop_check:             { exit to monitor? }
ar=dm(stop_flag); ar=pass ar; if eq jump data_ready_check;
i7="ack_strings+ACK_SIZE*8;
call sendLine;

________
- process data
- ________________

------

data_ready_check:
#if (SIMULATOR == 1)
call sport_rx;
{ call CODEC interrupt in
simulator to make things easier to follow and faster }
#endif

check_bank1:           { which of two buffer choices to
                        use? }
ar=dm(raw_data1+1); ay0=WAVE_SIZE; ar=ar-ay0; if lt jump
check_bank2;
ar="raw_data1;
dm(data_bank_ptr)=ar;
ar="cal_data1;
dm(cal_bank_ptr)=ar;
jump proc_data;
check_bank2:     
ar=dm(raw_data2+1); ay0=WAVE_SIZE; ar=ar-ay0; if lt jump
menu_check;
{     ar=dm(sample_count); ar=pass ar; if ne jump menu_check; }
ar="raw_data2;
dm(data_bank_ptr)=ar;
ar="cal_data2;
dm(cal_bank_ptr)=ar;

proc_data:     
ar=dm(counter); ar=ar+1; dm(counter)=ar;       { increment
            line/waveform counter }
{ make copy of raw data }
i0=dm(data_bank_ptr);
il="raw_copy;
call buff_stack;
ar=dm(stack_done);
ar=pass ar;
if eq jump menu_check;
#if'(1)
ar=dm(stack_cnt); ar=ar+1; dm(stack_cnt)=ar;
#endif
{ band-pass filter stage 1 }
i0="raw_copy;
il="bpf_outi
i4="fil_bpf
  call fir_dec;

  { band-pass filter stage 2 }
  i0="bpf_out;
  il="bpf2_out;
  i4="fil_lpf;
  call fir_dec;

  { update background nulling waveform }
  i0="bpf2_out;
  il="null_data;
  call update_null;

  { waveform nulling }
  i0="bpf2_out;
  il="bgnd_out;
  call buff_copy;
  i0="bgnd_out;
  il="null_data;
  call sub_background;

  { difference filter }
  i0="bgnd_out;
  il="diff_out;
  i4="fil_diff4;
  call fir_dec;

  { find peaks }
  i0="diff_out;
  il="peak_out;
  i2="bgnd_out;
  call loc_peaks;

  { find maximum and first peak above threshold }
  i0="peak_out;
  il="max_out;
  ax0=dm(rad_thresh);    { threshold value }
  ax1=dm(rad_transient); { values to skip }
  call find_max_and_thresh;

  ar=dm(cont_flag);  ar=pass ar;
  if eq jump menu_check;
  call show_results;

[-----------------------------]

------
- command menu
-
[-----------------------------]

menu_check:
  ax0=dm(poll_cnt);
  ax1=dm(poll_cnt+1);
  ay1=0;
  ar=ax0+1;
  sr0=ar, ar=ax1+ay1+C;
  dm(poll_cnt)=sr0;
  dm(poll_cnt+1)=ar;
cont_wave_check:
ar=dm(cont_wave_flag); ar=pass ar; if eq jump showback_check; ar=1; dm(show_rad_flag)=ar;

showback_check:
ar=dm(show_back_flag);
ar=pass ar;
if eq jump showback_skip;
ar=dm(null_flag); { background in progress? }
ar=pass ar;
if ne jump showback_skip;
ax0=dm(wave_cnt); ax1=dm(wave_cnt+1); ay1=0; ay0=2;
ar=ax0-ay0; ar=axl-ay1+c-1; if lt jump showback_skip; { wait for waveform to accumulate }
call show_ch_waveform;
ar=0; dm(show_back_flag)=ar;
dis ints;
call do_init; { re-initialize after doing waveform output }
enra ints;

showback_skip:
showrad_check:
ar=dm(show_rad_flag);
ar=pass ari
if eq jump showrad_skip
#if (0)
ax0=dm(wave_cnt); ax1=dm(wave_cnt+1); ay1=0; ay0=2;
ar=ax0-ay0; ar=axl-ay1+c-1; if lt jump showrad_skip;
#else
ar=dm(stack_cnt); ar=ar-1;
if lt jump showrad_skip; { wait for waveform to accumulate }
ar=0; dm(stack_cnt)=ar;
#endif
call show_rad_waveform;
ar=0; dm(show_rad_flag)=ar;
dis ints;
call do_init;
enra ints;

showrad_skip:
#if (SIMULATOR == 1)
ax1 = 0;
#else
#if (0)
    ar = dm(flag_rx_no_word);
    none = pass ar;
    if ne jump again;
    call get_char_ax1_to;
    if lt jump again; { time out }
#else
    ar=dm(command_count); { at least one character in command buffer? }
    none = pass ar;
    if eq jump again;
    call get_command_buff;
#endif
#endif
ar=dm(command_flag); { in the process of getting command packet? }
ar=pass ar;
if ne jump get_command;

menu_command:
ay0 = 36; { $ }
none = ax1 - ay0;  
{ command packets begin with '$'
}

if ne jump menu_enter;
ar=1;  dm(command_flag)=ar;
ar=1;  dm(command_letter)=ar;
i0=command_string;
l0=0;
m0=1;
dm(l0,m0)=36;
jump again;

menu_enter:
ay0 = 13;  
{ <Enter> }
none = ax1 - ay0;  
{ <Enter> is prompt command }
if ne jump menu_a;
call show_results;
jump again;

menu_a:
ay0 = 97;  
{ 'a' - auto output mode }
none = ax1 - ay0;
if ne jump menu_b;
ar=1;  dm(cont_flag)=ar;
i7=ack_strings+0;
call sendLine;
jump again;

menu_b:
ay0 = 98;  
{ 'b' - update background }
none = ax1 - ay0;
if ne jump menu_big_b;
ar=dm(null_flag);
ar=pass ar;  
{ background already in progress? }
if eq jump do_menu_b;
i7=ack_strings+ACK_SIZE;
call sendLine;
i7=not_ready_string;
call sendLine;
jump again;

do_menu_b:
i7=ack_strings+ACK_SIZE;
ar=1;  dm(null_flag)=ar;
call sendLine;
jump again;

menu_big_b:
ay0 = 66;  
{ 'B' - 'b' and 'c' commands together }
none = ax1 - ay0;
if ne jump menu_c;
ax0=dm(null_flag);
ay0=dm(show_back_flag);
ar=ax0+ay0;  
{ background or show waveform already in progress? }
if eq jump do_menu_big_b;
i7=ack_strings+ACK_SIZE*19;
call sendLine;
i7=not_ready_string;
call sendLine;
jump again;

do_menu_big_b:
ar=1;  dm(null_flag)=ar;
ar=1;  dm(show_back_flag)=ar;
i7=ack_strings+ACK_SIZE*19;
call sendLine;
jump again;

menu_c:
    ay0 = 99;  { 'c' - cal/background waveform }
    none = ax1 - ay0;
    if ne jump menu_d;
        ar=dm(show_back_flag);
        ar=pass ar;  { show waveform already in progress? }
        if eq jump do_menu_c;
            i7=A ack_strings+ACK_SIZE*9;
            call sendLine;
            i7=not_ready_string;
            call sendLine;
            jump again;
    do_menu_c:
        i7=ack_strings+ACK_SIZE*9;
        call sendLine;
        ar=1;  dm(show_back_flag)=ar;
        jump again;
    menu_d:
        ay0 = 100;  { 'd' - debug output }
        none = ax1 - ay0;
        if ne jump menu_h;
            ar=1;  dm(debug_flag)=ar;
            i7=ack_strings+ACK_SIZE*3;
            call sendLine;
            jump again;
    menu_h:
        ay0 = 104;  { 'h' - help }
        none = ax1 - ay0;
        if ne jump menu_i;
            i7=help_string;  call sendLine;
            dis ints;
            call do_init;
            ena ints;
            jump again;
    menu_i:
        ay0 = 105;  { 'i' - status info }
        none = ax1 - ay0;
        if ne jump menu_n;
            call show_status;
            dis ints;
            call do_init;
            ena ints;
            jump again;
    menu_n:
        ay0 = 110;  { 'n' - normal output }
        none = ax1 - ay0;
        if ne jump menu_p;
            ar=0;  dm(debug_flag)=ar;
            i7=ack_strings+ACK_SIZE*4;
            call sendLine;
            jump again;
    menu_p:
        ay0 = 112;  { 'p' - prompted output }
        none = ax1 - ay0;
        if ne jump menu_r;
            ar=0;  dm(cont_flag)=ar;
            i7=ack_strings+ACK_SIZE*5;
            call sendLine;
            jump again;
menu_r:
    ay0 = 114;  ('r' - reset)
    none = ax1 - ay0;
    if ne jump menu_s;
    ar=0;  dm(init_flag)=ar;
    i7='ack_strings+ACK_SIZE*6;
    call sendLine;
    jump again;

menu_s:
    ay0 = 115;  ('s' - show raw data waveform)
    none = ax1 - ay0;
    if ne jump menu_t;
    ar=dm(show_rad_flag);
    ar=pass ar;  (show waveform already in progress?)
    if eq jump do_menu_s;
    i7='ack_strings+ACK_SIZE*2;
    call sendLine;
    i7='not_ready_string;
    call sendLine;
    jump again;

    do_menu_s:
    i7='ack_strings+ACK_SIZE*2;
    call sendLine;
    ar=1;  dm(show_rad_flag)=ar;
    jump again;

menu_t:
    ay0 = 116;  ('t' - toggle auto/prompted output)
    none = ax1 - ay0;
    if ne jump menu_u;
    ar=dm(cont_flag);  ar=pass ar;
    if ne jump cont_off;
    cont_on:  ar=1;  dm(cont_flag)=ar;
               i7='ack_strings+0;
               call sendLine;
               jump again;

    cont_off:  ar=0;  dm(cont_flag)=ar;
               i7='ack_strings+ACK_SIZE*5;
               call sendLine;
               jump again;

menu_u:
    ay0 = 117;  ('u' - turn on binary mode)
    none = ax1 - ay0;
    if ne jump menu_v;
    ar=1;  dm(bin_flag)=ar;
    jump again;

menu_v:
    ay0 = 118;  ('v' - turn on binary mode)
    none = ax1 - ay0;
    if ne jump menu_w;
    ar=0;  dm(bin_flag)=ar;
    jump again;

menu_w:
    ay0 = 119;  ('w' - turn off continuous waveform mode)
    none = ax1 - ay0;
    if ne jump menu_y;
    ar=1;  dm(cont_wave_flag)=ar;
    jump again;

menu_y:
    ay0 = 121;  ('y' - turn off continuous waveform mode)
    none = ax1 - ay0;
if ne jump menu_1;
ar=0; dm(cont_wave_flag)=ar;
jump again;

menu_1:
ay0 = 49; ( '1' - no waveform stacking )
one = ax1 - ay0;
if ne jump menu_2;
ar=1; dm(stack_size)=ar;
ar=0; dm(stack_shift)=ar;
i7=^ack_strings+ACK_SIZE*10;
call sendLine;
jump again;

menu_2:
ay0 = 50; ( '2' - stack 8 waveforms )
one = ax1 - ay0;
if ne jump menu_3;
ar=8; dm(stack_size)=ar;
ar=-3; dm(stack_shift)=ar;
i7=^ack_strings+ACK_SIZE*11;
call sendLine;
jump again;

menu_3:
ay0 = 51; ( '3' - stack 16 waveforms )
one = ax1 - ay0;
if ne jump menu_4;
ar=16; dm(stack_size)=ar;
ar=-4; dm(stack_shift)=ar;
i7=^ack_strings+ACK_SIZE*12;
call sendLine;
jump again;

menu_4:
ay0 = 52; ( '4' - 8 kHz )
one = ax1 - ay0;
if ne jump menu_5;
ax0=0x0; dm(codec_param)=ax0;
call reset_codec;
i7=^ack_strings+ACK_SIZE*13;
call sendLine;
ar=0; dm(init_flag)=ar;
jump start;

menu_5:
ay0 = 53; ( '5' - 9.6 kHz )
one = ax1 - ay0;
if ne jump menu_6;
ax0=0xE; dm(codec_param)=ax0;
call reset_codec;
i7=^ack_strings+ACK_SIZE*14;
call sendLine;
ar=0; dm(init_flag)=ar;
jump start;

menu_6:
ay0 = 54; ( '6' - 16 kHz )
one = ax1 - ay0;
if ne jump menu_7;
ax0=0x2; dm(codec_param)=ax0;
call reset_codec;
i7=^ack_strings+ACK_SIZE*15;
call sendLine;
ar=0; dm(init_flag)=ar;
jump start;

menu_7:
ay0 = 55;  ( '7' - 27.4 kHz )
none = ax1 - ay0;
if ne jump menu_8;
ax0=0x4;  dm(codec_param)=ax0;
call reset_codec;
i7=ack_strings+ACK_SIZE*16;
call sendLine;
ar=0;  dm(init_flag)=ar;
jump start;

menu_8:
ay0 = 56;  ( '8' - 32 kHz )
none = ax1 - ay0;
if ne jump menu_9;
ax0=0x6;  dm(codec_param)=ax0;
call reset_codec;
i7=ack_strings+ACK_SIZE*17;
call sendLine;
ar=0;  dm(init_flag)=ar;
jump start;

menu_9:
ay0 = 57;  ( '9' - 48 kHz )
none = ax1 - ay0;
if ne jump menu_x;
ax0=0xC;  dm(codec_param)=ax0;
call reset_codec;
i7=ack_strings+ACK_SIZE*18;
call sendLine;
ar=0;  dm(init_flag)=ar;
jump start;

menu_x:
ay0 = 120;  ( 'x' - exit software )
none = ax1 - ay0;
if ne jump menu_invalid;
i7=ack_strings+ACK_SIZE*8;
call sendLine;
rts;

menu_invalid:
i7=invalid_string;
call sendLine;
jump again;

{*******************************************************************************
*******
Get command one character at a time
*******************************************************************************
******}
gevent_command:
index0=drn(command_count);  index=0x0;  m7=index;
index=ax0-1;  drn(command_count)=index;
i7=drn(command_ptr);  l7=%command_buff;
modify(i7,m7);  ax1=pm(i7,m7);
rts;

{*******************************************************************************
*******
Get command one character at a time
*******************************************************************************
******}
get_command:
   i0=command_string;
   10=0;
   ar=dm(command_letter);
   m0=ar;
   modify(i0,m0);
   m0=1;
   dm(i0,m0)=ax1;
   ar=ar+1;
   dm(command_letter)=ar;
   ay0=10;
   if (KLUDGE == 1)
      ay0=13;
   endif
   ar=ax1-ay0;
   if ne jump again;
      ar=0; dm(command_flag)=ar;
   }
   dm(i0,m0)=10;
   dm(i0,m0)=0;

   ( compare command string with available commands
      - set to one character command if possible
      - parameter entry is a special case )
   i0=command_string;

   c_help:
      il=comm_help; call strcmp; ar=pass ar;
      if eq jump c_auto;
      axl=104; jump getcomm_done;

   c_auto:
      il=comm_auto; call strcmp; ar=pass ar;
      if eq jump c_back;
      axl=97; jump getcomm_done;

   c_back:
      il=comm_back; call strcmp; ar=pass ar;
      if eq jump c_back2;
      axl=98; jump getcomm_done;

   c_back2:
      il=comm_back2; call strcmp; ar=pass ar;
      if eq jump c_back_wave;
      axl=66; jump getcomm_done;

   c_back_wave:
      il=comm_back_wave; call strcmp; ar=pass ar;
      if eq jump c_debug;
      axl=115; jump getcomm_done;

   c_debug:
      il=comm_debug; call strcmp; ar=pass ar;
      if eq jump c_normal;
      axl=100; jump getcomm_done;

   c_normal:
      il=comm_normal; call strcmp; ar=pass ar;
      if eq jump c_toggle;
      axl=110; jump getcomm_done;

   c_toggle:
      il=comm_toggle; call strcmp; ar=pass ar;
      if eq jump c_prompt;
      axl=116; jump getcomm_done;

   c_prompt:
      il=comm_prompt; call strcmp; ar=pass ar;
      if eq jump c_reset;
      axl=112; jump getcomm_done;

   c_reset:
il=`comm_reset;
call strcmp;
ar=pass ar;
if eq jump c_status;
axl=114; jump getcomm_done;
c_status:
il=`comm_status; call strcmp; ar=pass ar;
if eq jump c_stop;
axl=105; jump getcomm_done;
c_stop:
il=`comm_stop; call strcmp; ar=pass ar;
if eq jump c_cb_wave;
axl=120; jump getcomm_done;
c_cb_wave:
il=`comm_cb_wave; call strcmp; ar=pass ar;
if eq jump c_menu1;
axl=99; jump getcomm_done;
c_menu1:
il=`comm_menu1; call strcmp; ar=pass ar;
if eq jump c_menu2;
axl=49; jump getcomm_done;
c_menu2:
il=`comm_menu2; call strcmp; ar=pass ar;
if eq jump c_menu3;
axl=50; jump getcomm_done;
c_menu3:
il=`comm_menu3; call strcmp; ar=pass ar;
if eq jump c_menu4;
axl=51; jump getcomm_done;
c_menu4:
il=`comm_menu4; call strcmp; ar=pass ar;
if eq jump c_menu5;
axl=52; jump getcomm_done;
c_menu5:
il=`comm_menu5; call strcmp; ar=pass ar;
if eq jump c_menu6;
axl=53; jump getcomm_done;
c_menu6:
il=`comm_menu6; call strcmp; ar=pass ar;
if eq jump c_menu7;
axl=54; jump getcomm_done;
c_menu7:
il=`comm_menu7; call strcmp; ar=pass ar;
if eq jump c_menu8;
axl=55; jump getcomm_done;
c_menu8:
il=`comm_menu8; call strcmp; ar=pass ar;
if eq jump c_menu9;
axl=56; jump getcomm_done;
c_menu9:
il=`comm_menu9;
call strcmp;
ar=pass ar;
if eq jump c_dndsp;
axl=57; jump getcomm_done;
c_dndsp:
il=`comm_dndsp;
call strcmp;
ar=pass ar;
if eq jump c_param;
{ make sure only interrupts for UART occur }
ifc = b#00000011111111;  \{ clear pending interrupt \}
nop;  \{ wait for ifc latency \}
imask=b#0000000101;
jump dndsp;
jump start;

c_param:  \{ this has to be the last command
being checked because of the special code below \}
i0=command_string;
ar=0;  \{ add null for string compare \}

\{ clear pending interrupt \}
\{ wait for ifc latency \}

if eq jump c_invalid;
call decode_parm;  \{ update selected parameter with
value in packet \}
dis ints;
call do_init;
ena ints;
jump again;
c_invalid:
i7=invalid_string;
call sendLine;
jump again;

getcomm_done:
jump menu_command;

{***********************************************************************
*******
Show status information
Info:
  stack: XXXX\n  (1,2,4)
  null_stack: XXXX\n  cal_tol: XXXX\n  rate: XX.X\n  (08.0,09.6,16.0,27.4,32.0,48.0)
  delay: XXXX\n  rad_thres: XXXX\n  cal_thres: XXXX\n  tx_offset: XXXX\n  rad_trans: XXXX\n  sos_on_time: XXXX\n  ramp_start: XXXX\n  ramp_inc: XXXX\n
***********************************************************************
*****

.var/dm/ram status_info[250];
.init status_info: '$','A','D','s','-',',s','t','a','c','k',':',,''
                    ,',',',',',',',',13,10, \{ 17 \}
                    '$','A','D','s','-',
                    ',n','s','t','a','c','k','k',':',',',',',',',',',13,10, \{ 18 \}
                    '$','A','D','s','-',
                    ',c','a','l','-',',t','o','l','s','-',',r','a','d','t','h','r','e','s','-',

                      \} 16
                      \} 17
                      \} 18
                      \} 19
                      \} 20
                      \} 21

```
show_status:
    i7=ack_strings+ACK_SIZE*7;
    call sendLine;
    l0=0; m0=1;
    ( stack size info )
    i0=^status_info+11; ax0=dm(stack_size); call int_to_ascii_hex;
    ( null stack size )
    i0=^status_info+29; ax0=dm(null_stack); call int_to_ascii_hex;
    ( cal tolerance )
    i0=^status_info+48; ax0=dm(cal_tol); call int_to_ascii_hex;
    ( delay )
    i0=^status_info+81; ax0=dm(codec_delay); call int_to_ascii_hex;
    ( rad_thresh )
    i0=^status_info+103; ax0=dm(rad_thresh); call int_to_ascii_hex;
    ( cal_thresh )
    i0=^status_info+125; ax0=dm(cal_thresh); call int_to_ascii_hex;
    ( tx_offset )
    i0=^status_info+146; ax0=dm(tx_offset); call int_to_ascii_hex;
    ( rad_trans )
    i0=^status_info+167; ax0=dm(rad_transient); call int_to_ascii_hex;
    ( sos_on_time )
    i0=^status_info+190; ax0=dm(sos_on_time); call int_to_ascii_hex;
    ( ramp_start )
    i0=^status_info+212; ax0=dm(ramp_start); call int_to_ascii_hex;
    ( ramp_inc )
    i0=^status_info+232; ax0=dm(ramp_inc); call int_to_ascii_hex;
    ( CODE rate info )
    i0=^status_info+64; 10=0; m0=1; m1=2;
    ax0=dm(codec_param);
    ay0=0; none=ax0-ay0; if ne jump s096k;
    s080k:  dm(i0,m0)=48;  dm(i0,m1)=56;  dm(i0,m0)=48;
        jump sdone;
    s096k:  ay0=14; none=ax0-ay0; if ne jump s160k;
        dm(i0,m0)=48;  dm(i0,m1)=57;  dm(i0,m0)=54;
        jump sdone;
    s160k:  ay0=2; none=ax0-ay0; if ne jump s274k;
        dm(i0,m0)=49;  dm(i0,m1)=54;  dm(i0,m0)=48;
        jump sdone;
    s274k:  ay0=4; none=ax0-ay0; if ne jump s320k;
        dm(i0,m0)=50;  dm(i0,m1)=52;  dm(i0,m0)=55;
        jump sdone;
    s320k:  ay0=6; none=ax0-ay0; if ne jump s480k;
```
dm(i0,m0)=51; dm(i0,m1)=50; dm(i0,m0)=48;
jump sdone;
s480k: ay0=12; none=ax0-ay0; if ne jump sxxxk;
dm(i0,m0)=52; dm(i0,m1)=56; dm(i0,m0)=48;
jump sdone;
sxxxk: dm(i0,m0)=120; dm(i0,m1)=120; dm(i0,m0)=120;
sdone: i7=`status_info;
call sendLine;
rts;

{*******************************************************************************
******
Retrieve Parameter from command string and update appropriate variable
*******************************************************************************
******}
.var/dm/ram param_strings[33];
.init param_strings:
'd', 'e', 0, { codec delay }  
's', 's', 0, { stack size }  
'n', 's', 0, { null stack }  
'c', 't', 0, { cal thresh }  
'r', 't', 0, { rad thresh }  
'c', 'a', 0, { cal tolerance }  
't', 'o', 0, { tx offset }  
'r', 'a', 0, { rad transient }  
's', 'o', 0, { sos on time }  
'r', 's', 0, { ramp start }  
'r', 'i', 0; { ramp increment }
.var/dm/ram param_invalid[17];
.init param_invalid: '$', 'A', 'D', '0', 'a', 'p', '
', 'i', 'n', 'v', 'a', 'i', 'd', 13, 10, 0;
.var/dm/ram param_ack_string[12];
.init param_ack_string: '$', 'A', 'D', '0', 'a', 'p', ' 
', 'i', 'n', 'v', 'a', 'i', 'd', 13, 10, 0;
.var/dm/ram param_val;

decode_param_string:
    ar=0; dm(command_string+9)=ar;
    10=0; m0=1;
    i0=`command_string+10;
    call get_hex;  { extract parameter value }  
    dm(param_val)=ar;
    i0=`command_string+7;  
    { codec delay }
    pcodec:
        i1=`param_strings+0; call strcmp; ar=pass ar; if eq jump
    pstack:
        ar=dm(param_val); dm(codec_delay)=ar;
        jump param_ack;
        { stack size }
    pstack:
        i1=`param_strings+3; call strcmp; ar=pass ar; if eq jump
    pnull:
        ar=dm(param_val); ar=-ar; dm(stack_shift)=ar;
        jump param_ack;
        { null stack }
    pnull:
il= param_strings+6; call strcmp; ar=pass ar; if eq jump pcthresh;
    ar=dm(param_val); ar=-ar; dm(null_shift)=ar;
    jump param_ack;
    { cal thresh }

pcthresh:
il= param_strings+9; call strcmp; ar=pass ar; if eq jump prthresh;
    ar=dm(param_val); dm(cal_thresh)=ar;
    jump param_ack;
    { rad thresh }

prthresh:
il= param_strings+12; call strcmp; ar=pass ar; if eq jump pctol;
    ar=dm(param_val); dm(rad_thresh)=ar;
    jump param_ack;
    { cal tolerance }

pctol:
il= param_strings+15; call strcmp; ar=pass ar; if eq jump ptxoffset;
    ar=dm(param_val); dm(cal_tol)=ar;
    jump param_ack;
    { tx offset }

ptxoffset:
il= param_strings+18; call strcmp; ar=pass ar; if eq jump prtrans;
    ar=dm(param_val); dm(tx_offset)=ar;
    jump param_ack;
    { rad transient }

prtrans:
il= param_strings+21; call strcmp; ar=pass ar; if eq jump psos;
    ar=dm(param_val); dm(rad_transient)=ar;
    jump param_ack;
    { sos on time }

psos:
il= param_strings+24; call strcmp; ar=pass ar; if eq jump prstart;
    ar=dm(param_val); dm(sos_on_time)=ar;
    jump param_ack;
    { ramp start }

prstart:
il= param_strings+27; call strcmp; ar=pass ar; if eq jump prin;
    ar=dm(param_val); dm(ramp_start)=ar;
    jump param_ack;
    { ramp increment }

prin:
il= param_strings+30; call strcmp; ar=pass ar; if eq jump pinvalid;
    ar=dm(param_val); dm(ramp_inc)=ar;
    jump param_ack;

pinvalid:
i7= param_invalid;
call sendLine;
rts;

param_ack:
i0= command_string;
ar=dm(command_string+7); dm(parm_ack_string+7)=ar;
ar=dm(command_string+8); dm(parm_ack_string+8)=ar;
i7= param_ack_string;

56
call sendLine;

rts;

{******************************************************************************
*******
Decode 4 digit hex string
- i0 = pointer to string
- ar = return value

******************************************************************************
*******
}
get_hex:
    ar=0;  { clear accumulator }
    ax0=dm(i0,m0);  my0=h#1000;  call get_hex_digit;
    ax0=dm(i0,m0);  my0=h#100;  call get_hex_digit;
    ax0=dm(i0,m0);  my0=h#10;  call get_hex_digit;
    ax0=dm(i0,m0);  my0=h#1;  call get_hex_digit;
    rts;

get_hex_digit:
    si=ar;  { save accumulator }
    ay1=97;  ar=ax0-ay1;  if lt jump not_lowercase;
    ay1=10;  ar=ar+ay1;  { lowercase hex digit }
    jump digit_comp;
    not_lowercase:
        ay1=65;  ar=ax0-ay1;  if lt jump not_big_letter;
        ay1=10;  ar=ar+ay1;  { uppercase hex digit }
        jump digit_comp;
    not_big_letter:
        ay1=48;  ar=ax0-ay1;  { digit between 0 and 9 }
    digit_comp:
        mr=ar*my0 (uu);
        sr=lshift mr1 by -1 (hi);
        sr=sr or lshift mr0 by -1 (lo);
        ay1=si;  { restore accumulator }
        ar=sr0+ay1;
        rts;

{******************************************************************************
*******
Compare two strings
- return ar=1 if strings are the the same

******************************************************************************
*******
}
.var/dm/ram start_i0;
.var/dm/ram start_i1;
strcmp:  ar=0;
    l0=0;  l1=0;  m0=1;  m1=1;
    dm(start_i0)=i0;
    dm(start_i1)=i1;
    scmploop:
        ax0=dm(i0,m0);  ay0=dm(i1,m1);
        none=pass ax0;
        if eq jump strcmp_done;  { check for terminator on first string }
            none=ax0-ay0;
            if ne ar=ar+1;
            if eq jump scmploop;
        strcmp_done:
none=pass ay0;
if ne ar=ar+1;
            { check for terminator for second
string }
  ay0=ar;
ar=pass ar;
  if eq jump strcmp_match;
ar=0;
  jump strcmp_return;
strcmp_match:
ar=1;
strcmp_Return:
i0=dm(start_i0);
il=dm(start_il);
  rts;

{*******************************************************************************
********
  Show some results at output
*******************************************************************************
*******}
show_results:
toggle fl1;
i0="num_string";  10=0;  m0=1;
dm(i0,m0)=36;  { 'S' }
dm(i0,m0)=65;  { 'A' }
dm(i0,m0)=68;  { 'D' }
dm(i0,m0)=48;  { '0' }
dm(i0,m0)=48;  { '0' }
  ar=dm(debug_flag);
ar=pass ar;
  if ne jump debug_out;
normal_out:
dm(i0,m0)=110;  { 'n' }
call show_normal;
  jump show_line;
debug_out:
dm(i0,m0)=100;  { 'd' }
call show_debug;
show_line:
dm(i0,m0)=13;
dm(i0,m0)=10;
dm(i0,m0)=0;
i7="num_string; 
call sendLine;
  rts;

{*******************************************************************************
********
Reset CODEC
  Parameter: AX0 = sample rate choice parameter
*******************************************************************************
*******}
reset_codec:
ar=dm(init_cmds+8);
  ay0=0xFFFF0;
ar=ar and ay0;
  ay0=ax0;
ar=ar or ayO;
dm(init_cmds+8)=ar;
rts;

{******************************************************************************
******** Show normal info
******************************************************************************
******}

show_normal:
{
  height above ground
  mxO=dm(fm_max_index);
  myO=84;  {h#100}
  mr=mxO*myO (ss);
  axl=mr1;
  ax0=mr0;
  call num_to_ascii;

  { snow thickness }
  axO=dm(fm_max_index);
  ayO=dm(fm_thresh_index);
  ar=axO-ayO;
  myO=168;  {h#200}
  mr=ar*myO (ss);
  axl=mr1;
  ax0=mr0;
  call num_to_ascii;

  rts;
}

{******************************************************************************
******** Show debug info
******************************************************************************
******}

show_debug:
    ax0 = dm(counter);  axl=0;
    call int_to_ascii_hex;
    #if (0)
      dm(i0,m0)=32;

      ax0 = dm(sample_count);  axl=0;
      call int_to_ascii_hex;
      dm(i0,m0)=32;
    #endif

    ax0=dm(fm_thresh_val);  axl=0;
    call int_to_ascii_hex;
    { dm(i0,m0)=32;}

    ax0=dm(fm_thresh_index);  axl=0;
    call int_to_ascii_hex;
    { dm(i0,m0)=32;}

    ax0=dm(fm_max_val);  axl=0;
    call int_to_ascii_hex;
    { dm(i0,m0)=32;}

59
ax0=dm(fm_max_index); ax1=0;
call int_to_ascii_hex;
rts;

{***********************************************************************
*******
Show cal/background waveforms
************************************************************************
******}
.var/dm/ram cb-pointer;
show_cb_waveform:
  l1=0; m1=1;
  l2=0; m2=1;
  ar=1; dm(data_off_flag)=ar;
  ar=0; dm(line_count)=ar; dm(col_count)=ar;
  ar=dm(data_bank);
  ar=ar-1;
  if eq jump cb_bank2;
    cb_bank1:
      i1=Araw_data1; 
      il=A null_data+3;
      i2=A cal_data1;
      jump cb_header;
    cb_bank2:
      i1=A raw_data2; 
      il=A null_data+3;
      i2=A cal_data2;
    cb_header:
      ax1=dm(i2,m2); \{ pointer \}
      modify(i2,m2); \{ skip count \}
      modify(i2,m2); \{ skip total \}
      ax0=dm(i2,m2); \{ length \}
      ax0=WAVE_SIZE;
      i2=ax1;
      dm(cb_pointer)=ax0;
      jump cb_newline;
    cb_loop:
      ar=dm(cb_pointer); ar=ar-1;
      if lt jump cb_done;
      dm(cb_pointer)=ar;
      ax0 = dm(i1,m1);
      call int_to_ascii_hex;
      ax0 = dm(i2,m2);
      call int_to_ascii_hex;
      dm(i0,m0)=32; \}
      ay0=20; ar=dm(col_count); ar=ar+1; dm(col_count)=ar;
      ar=ar-ay0; if lt jump cb_loop;
      dm(i0,m0)=13; dm(i0,m0)=10; dm(i0,m0)=0;
      i7=A num_string;
      call sendLine;
    cb_newline:
      i0=A num_string; 10=0; m0=1;
      dm(i0,m0)=36; \{ ' ' \}
dm(i0,m0)=65; ( 'A' )
dm(i0,m0)=68; ( 'D' )
dm(i0,m0)=48; ( '0' )
dm(i0,m0)=119; ( 'W' )
dm(i0,m0)=99; ( 'c' )
ar=dm(line_count); ar=ar+1; dm(line_count)=ar;
ar=dm(counter);
ax0=ar; call int_to_ascii_hex;
{
dm(i0,m0)=32;}
ar=0; dm(col_count)=ar;
jump cb_loop;

cb_done:
ar=dm(col_count); ar=pass ar; if eq jump cb_term;
dm(i0,m0)=13; dm(i0,m0)=10; dm(i0,m0)=0;
i7=num_string;
call sendLine;

cb_term:
ar=0; dm(data_off_flag)=ar;
rt;

{***********************************************************************
********
Show background waveform
***********************************************************************
******}
.var/dm/ram rad_pointer;
show_rad_waveform:
ar=1; dm(data_off_flag)=ar;
ar=0; dm(line_count)=ar; dm(col_count)=ar;

#if (DEBUG == 1)
ar=0; dm(temp_count)=ar;
#endif
#if (DEBUG_MEM == 1)
i4=debug_mem;
dm(debug_ptr)=i4;
l4=%debug_mem;
m4=1;
#endif

ar=dm(data_bank);
ar=ar-li;
if eq jump rad_bank2;
rad_bank1:
{
il=raw_data1;
il=raw_copy;
jump rad_header;
rad_bank2:
il=raw_data2;
il=raw_copy;
rad_header:
ay0=dm(i1,m1); { pointer }
modify(i1,m1); { skip wave_count }
modify(i1,m1); { skip stack_size }
ay1=dm(i1,m1); { length }

ay1=WAVE_SIZE;
il=ay0;
dm(rad_pointer)=ay1;
l1=0; ml=1;
jump rad_newline;

rad_loop:
    ar=dm(rad_pointer); ar=ar-1;
    if lt jump sr_done;
    dm(rad_pointer)=ar;
    ax0 = dm(i1,m1);
    ar = dm(bin_flag);
    ar = pass ar;
    if eq jump text_show;

bin_show:
    ay0 = h#00FF;
    ar = ax0 and ay0;
    #if (DEBUG == 1)
    ar=dm(temp_count);
    { ar=3; }
    #endif
    dm(i0,m0)=ar;
    ay0 = h#FF00;
    ar = ax0 and ay0;
    sr = lshift ar by -8 (10);
    #if (DEBUG == 1)
    { srO=3; }
    srO=dm(temp_count);
    #endif
    dm(i0,m0)=sr0;
    #if (DEBUG == 1)
    { sr0=3; }
    sr0=dm(temp_count);
    #endif
    ar=srO+1; dm(temp_count)=ar;
    #endif
    jump text_show_done;

text_show:
    call int_to_ascii_hex;
text_show_done:
    { dm(i0,m0)=32; }
    ay0=50; ar=dm(col_count); ar=ar+1; dm(col_count)=ar;
    ar=ar-ay0; if lt jump rad_loop;
    dm(i0,m0)=13; dm(i0,m0)=10; dm(i0,m0)=0;
    i7="num_string;
    call sendLine;
    #if (DEBUG_MEM == 1)
    i0="num_string; 10=0; m0=1;
    i4=dm(debug_ptr);
    ax0=112;
    cntr=ax0;
    do debug_pm_transfer until ce;
    ar=dm(i0,m0);
    debug_pm_transfer: pm(i4,m4)=ar;
    dm(debug_ptr)=i4;
    #endif
    rad_newline:
    i0="num_string; 10=0; m0=1;
    dm(i0,m0)=36; { '$' }
    dm(i0,m0)=65; { 'A' }
    dm(i0,m0)=68; { 'D' }
    dm(i0,m0)=48; { '0' }
    dm(i0,m0)=119; { 'w' }
    ar=114; ay0=14;
    ax0=dm(bin_flag); none=pass ax0; if ne ar=ar-ay0;
    dm(i0,m0)=ar; { 'r' (hex) or 'b' (binary) }
ar = dm(line_count); ar = ar + 1; dm(line_count) = ar;
ar = dm(counter);
    ax0 = ar; call int_to_ascii_hex;
    { 
    dm(i0,m0) = 32;
    ar = 0; dm(col_count) = ar;
    jump rad_loop;
sr_done:
ar = dm(col_count); ar = pass ar; if eq jump rad_term;
dm(i0,m0) = 13; dm(i0,m0) = 10; dm(i0,m0) = 0;
i7 = "num_string; call sendLine;

rad_term:
ar = 0; dm(data_off_flag) = ar;
    rts;

{ **** Some Interrupt Service Routines }
{-----------------------------------------------------------------------
-------
- SPORT interrupt handler
- 
-----------------------------------------------------------------------
-------}
.var/dm/ram is_mem[10];
.var/ram/dm context[9];
sport_rx:
    ena sec_reg;
dm(context) = i0;
dm(context+1) = m0;
dm(context+2) = 10;
dm(context+3) = i1;
dm(context+4) = m1;
dm(context+5) = 11;
#if (0)
dm(is_mem) = ar;
dm(is_mem+1) = ax0;
dm(is_mem+2) = ax1;
dm(is_mem+3) = ay0;
#endif
    ar = dm(flag_rx_no_word);
    none = pass ar; { check UART receive buffer by polling in this interrupt }
    if ne jump skip_getchar;
call get_char_ax1;
#if (0)
    none = pass ax1; if le jump skip_getchar;
#endif
dm(context+6) = i7;
dm(context+7) = m7;
dm(context+8) = 17;
ar = dm(command_count); ar = ar + 1; dm(command_count) = ar;
ar = dm(command_total); ar = ar + 1; dm(command_total) = ar;
i7 = dm(command_ptr); 17 = &command_buff; m7 = 1;
prn(i7, m7) = ax1; dm(command_ptr) = i7;
i7 = dm(context+6);
m7 = dm(context+7);
17 = dm(context+8);

skip_getchar:
#if (0)

{ moved down below }

ar=dm(data_off_flag);
ar=pass ar;
if ne jump done_read;
#endif

m0=0; ml=1;
#if (FAKE_DATA == 1)
#else
#if (0)
#else
#endif
#endif

pos_sos:

iO=drn(fake_dptr1)i
lO=%fake_data1i
i1=drn(fake_dptr2)i
l1=%fake_data2i
axO=drn(iO,ml)i
ax1=drn(i1,ml)i

ar=-axOi
ax1=ari
drn(fake_dptr1)=iOi
drn(fake_dptr2)=i1i

axO=drn(rx_buf+2)i
axl=drn(rx_buf+1)i
/* R channel */
/* L channel */

{ display ramp by multiplying sample count by -16 }

ar=dm(sample_count);
ar=-ar;
ay0=ar; ar=ar+ay0;
ay0=ar; ar=ar+ay0;
ay0=ar; ar=ar+ay0;
ay0=ar; ar=ar+ay0;
dm(tx_buf+2)=ar;
/* L channel */

#else

ar=dm(ramp_val);
dm(tx_buf+1)=ar;
/* L channel */

ay0=dm(ramp_inc);
ar=ax0+ay0;
dm(ramp_val)=ar;
#endif

ar=dm(sample_count); ay0=1; ar=ar-ay0;
if le jump pos_sos;

zero_sos:
ar=0;
dm(tx_buf+2)=ar;
/* R channel */
jump inc_sample;

pos_sos:
ar=32000;
dm(tx_buf+2)=ar;
/* R channel */

inc_sample:
ar=dm(sample_count); ar=ar+1; dm(sample_count)=ar; { increment counter }
#if (SKIP_ONE == 1)
#else
#endif

ay0=WAVE_SIZE+1;
#define WAVE_SIZE

ay0=WAVE_SIZE;
#endif

ar=ar-ay0; if lt jump read_data;
ar=0; dm(sample_count)=ar;
ar=dm(ramp_start); dm(ramp_val)=ar;

{ increment wave count }

ax0=dm(wave_cnt); axl=dm(wave_cnt+1); ay1=0; ar=ax0+1;
sr0=ar, ar=ax1+ay1+C; dm(wave_cnt)=sr0; dm(wave_cnt+1)=ar;
#if (1)
    ar=dm(data_off_flag);
    ar=pass ar;
    if ne jump skip_switch;
#endif

    { switch data banks }
    ar=dm(data_bank);
    ar=ar-1;
    if eq jump bank2;
    ar=1;
    jump set_bank;

bank2: ar=2;
set_bank:
    dm(data_bank)=ar;
skip_switch:
    #if (SKIP_ONE == 1)
        jump done_read;
    #endif

read_data:
    ar=dm(data_bank);
    ar=ar-1;
    if ne jump read_bank2;

read_bank1:
    i0=dm(raw_data1); 10=dm(raw_data1+3);
    ar=dm(raw_data1+1); ar=ar+1; dm(raw_data1+1)=ar;  { increment counter }

    { total }
    dm(i0,m1)=ax0;
    dm(raw_data1)=i0;    { update pointer }
    i0=dm(cal_data1); 10=dm(cal_data1+3);
    ar=dm(cal_data1+1); ar=ar+1; dm(cal_data1+1)=ar;  { increment counter }

    { total }
    dm(i0,m1)=ax1;
    dm(cal_data1)=i0;    { update pointer }
    jump done_read;

read_bank2:
    i0=dm(raw_data2); 10=dm(raw_data2+3);
    ar=dm(raw_data2+1); ar=ar+1; dm(raw_data2+1)=ar;  { increment counter }

    { total }
    dm(i0,m1)=ax0;
    dm(raw_data2)=i0;    { update pointer }
    i0=dm(cal_data2); 10=dm(cal_data2+3);
    ar=dm(cal_data2+1); ar=ar+1; dm(cal_data2+1)=ar;  { increment counter }

    { total }
    dm(i0,m1)=ax1;
    dm(cal_data2)=i0;    { update pointer }

    done_read:
    #if (0)
        ar=dm(is_mem);
        ax0=dm(is_mem+1);
    #endif
```c
axl = dm(is_mem+2);
ay0 = dm(is_mem+3);
#endif
i0 = dm(context);
m0 = dm(context+1);
l0 = dm(context+2);
i1 = dm(context+3);
m1 = dm(context+4);
l1 = dm(context+5);
#if (SIMULATOR == 1)
dis sec_reg;
rts;
#else
rti;
#endif

{***********************************************************************
*******
** IRQE ISR - Push Button Interrupt Service Routine
***************************~********************************************
******}
.var/dm/ram irq_mem;
irqeisr: dm(irq_mem) = ar;
ar = 1; dm(stop_flag) = ar;
ar = dm(irq_mem);
rti;

{-----------------------------------------------------------------------
- transmit interrupt used for Codec initialization
- ---------------------------------------------------------------------
-----)
.var/dm/ram nc_mem[3];
next_cmd:
nc_mem[0] = ax0;
nc_mem[1] = ay0;
nc_mem[2] = ar;

ax0 = dm(i3, m1);        { fetch next control word and }
dm(tx_buf) = ax0;        { place in transmit slot 0    }
ax0 = i3;
ay0 = ^init_cmds;
ar = ax0 - ay0;
if gt rti;               { rti if more control words still
waiting }
ax0 = 0xaf00;            { else set done flag and    }
dm(tx_buf) = ax0;        { remove MCE if done initialization }
}
ax0 = 0;
dm(stat_flag) = ax0;     { reset status flag    }
ax0 = dm(nc_mem);
ay0 = dm(nc_mem+1);
ar = dm(nc_mem+2);
rti;
```
A high to low transition on flag_in signifies the start bit; it also triggers IRQ1 ISR which then turn on timer if the timer is off. This is at to most 1/3 bit period too late but we should still catch the byte.

irq1_isr:
pop sts;
ena timer;  { start timer now }
rts;  { note rts }

{-----------------------------------------------------------------------
-------
{ Initialize Variables
-------
-------
-------
-------
-------
-------
-------
-------
-------
-------
-------}

{ Initialize Variables

do_init:
{ initialize buffers }
m0=1; l0=0;
m1=1; l1=0;
i0=^raw_data1; i1=^buff0; ax0=WAVE_SIZE; call buff_init;
i0=^raw_data2; i1=^buff1; ax0=WAVE_SIZE; call buff_init;
i0=^cal_data1; i1=^buff2; ax0=WAVE_SIZE; call buff_init;
i0=^cal_data2; i1=^buff3; ax0=WAVE_SIZE; call buff_init;
i0=^raw_copy; i1=^buff4; ax0=WAVE_SIZE; call buff_init;
i0=^bgnd_out; i1=^buff5; ax0=WAVE_SIZE; call buff_init;
i0=^bpf_out; i1=^buff6; ax0=WAVE_SIZE; call buff_init;
i0=^bpf2_out; i1=^buff7; ax0=WAVE_SIZE; call buff_init;
i0=^diff_out; i1=^buff8; ax0=WAVE_SIZE; call buff_init;
i0=^peak_out; i1=^buff9; ax0=WAVE_SIZE; call buff_init;
i0=^max_out; i1=^out_buff1; ax0=OUT_BUFF; call buff_init;
i0=^thresh_out; i1=^out_buff2; ax0=OUT_BUFF; call buff_init;

{ initialize counters }
ar = 0; dm(counter) = ar;

ar = 0; dm(sample_count) = ar;
dm(poll_cnt) = ar; dm(poll_cnt+1) = ar;
dm(wave_cnt) = ar; dm(wave_cnt+1) = ar;

{ initialize waveform nulling structure }
i0=^null_data; ax0=WAVE_SIZE; ax1=dm(null_stack); call null_init;

{ initialize flags }
ar = 0;
dm(stop_flag) = ar;
dm(null_flag) = ar;

dm(cont_flag) = ar;
{  
  dm(debug_flag)=ar;  
  dm(data_off_flag)=ar;  
  dm(show_back_flag)=ar;  
  dm(show_rad_flag)=ar;  
  dm(command_flag)=ar;  
  dm(command_letter)=ar;  

  ( set stack parameters )  
  ar=1;  dm(stack_size)=ar;  
  ar=dm(stack_shift);  ar=-ar;  se=ar;  si=1;  sr=1shift si (lo);  
  dm(stack_size)=sr0;  
  ar=dm(null_shift);  ar=-ar;  se=ar;  si=1;  sr=1shift si (lo);  
  dm(null_stack)=sr0;  
  ar=0;  dm(stack_counter)=ar;  
  ar=0;  dm(stack_done)=ar;  

  ( set data bank pointers )  
  ar=1;  dm(data_bank)=ar;  
  ar="raw_data1i  
  dm(data_bank_ptr)=ar;  
  ar="cal_data1i  
  dm(cal_bank_ptr)=ar;

#if (FAKE_DATA == 1)  
i0="fake_data1i  
dm(fake_dptr1)=i0;  
i0="fake_data2i  
dm(fake_dptr2)=i0;
#endif

rts;

***********************************************************************  
* Initialize Buffer  
*  
i0,m0,10 - header to initialize  
i1,m1,11 - buffer to initialize  
ax0 - buffer length  
* Destroys: ar,i0  
* Structure Format: 0) pointer  
* 1) count  
* 2) total  
* 3) length  
* (data not adjacent to header in memory)  
***********************************************************************

buff_init:  
ar=1;  dm(i0,m0)=ar;  
    { pointer }  
dm(i0,m0)=0;  
    { count }  
dm(i0,m0)=0;  
    { total }  
dm(i0,m0)=ax0;  
    { length }  
ar=0;  
  
cntr=ax0;  
do binit_lp until ce;  

binit_lp:  
dm(i1,m1)=ar;  
    rts;

***********************************************************************  
* Copy Buffer  
*  
i0 - input header  
i1 - output header  
ax0,ay1,ar,m0,m1,10,11  
* Structure Format: 0) pointer  
* 1) count  
* 2) total  
***********************************************************************
* 3) length *
* (data not adjacent to header in memory) *

****************************************************************
.var/dm/ram bc_input_count;
.var/dm/ram bc_input_hptr;
.var/dm/ram bc_output_hptr;

buff_copy:
    mO=0;  ml=1;  lO=0;  l1=0;

    dm(bc_input_hptr)=i0;
    dm(bc_output_hptr)=il;

    axO=dm(i0,ml);   { input pointer }
    ar=dm(i0,m0);    { input count }
    none = pass ar;  if le rts;  { return if there is no new data }

    dm(bc_input_count)=ar;
    dm(i0,m1)=0;     { set input count to zero }
    modify(i0,m1);   { skip input total }
    ar=dm(i0,m1);
    l0=ar;  i0=axO;

pointer)
    ar=dm(bc_input_count);
    ar=ar;  ml=ar;  modify(i0,m1);  { rewind pointer to beginning of }
    data }

    m1=1;
    ar=i0;  i0=dm(bc_input_hptr);  ay1=lO;  l0=0;
    dm(i0,m0)=ar;  { update pointer }
    l0=ar;  l0=ay1;

    axO=dm(i1,ml);  { output pointer }
    ar=dm(i1,m0);  { output count }
    ay1=dm(bc_input_count);
    ar=ar+ay1;
    dm(i1,ml)=ar;  { increment output count }
    ar=dm(i1,m0);  { output total }
    ar=ar+ay1;
    dm(i1,ml)=ar;  { increment output total }
    ar=dm(i1,ml);
    l1=ar;  i1=axO;

pointer }

cntr=dm(bc_input_count);
do bcpy_lp until ce;
    ar=dm(i0,m1);

bcpy_lp:  dm(i1,ml)=ar;
    ax0=i1;
    i1=dm(bc_output_hptr);  l1=0;
    dm(i1,ml)=ax0;  { update output pointer }
    rts;

****************************************************************
* Copy into Stacking buffer *
* i0 - input header *
* i1 - output header *
* Destroys: ax0,ay1,ar,m0,m1,l0,l1 *
* Structure Format: 0) pointer *
* 1) count *
* 2) total *
* 3) length *

69
* (data not adjacent to header in memory) *

```assembly
.var/dm/ram st_input_count;
.var/dm/ram st_input_hptr;
.var/dm/ram st_output_hptr;

buff_stack:
    m0=0; m1=1; 10=0; 11=0;

    dm(st_input_hptr)=i0;
    dm(st_output_hptr)=i1;

    ax0=dm(i0,m1);  
    ar=dm(i0,m0);  
    none = pass ar; if le rts;  

    dm(st_input_count)=ar;
    dm(i0,m1)=0;  
    modify(i0,m1);  
    ar=dm(i0,m1);  
    10=ar; 10=ax0;

    ar=dm(st_input_count);
    ar=-ar; m1=ar; modify(i0,m1); { rewind pointer to beginning of

    data }
    m1=1;
    ar=i0; 10=dm(st_input_hptr); ay1=10; 10=0;
    dm(i0,m0)=ar;
    i0=ar; 10=ay1;

    ax0=dm(i1,m1);  
    modify(i1,m1);  
    modify(i1,m1);  
    ar=dm(i1,m1);  
    11=ar; 11=ax0;

    ar=dm(stack_counter);
    ar=pass ar;
    if ne jump do_stack;

    do_first:
    se=dm(stack_shift);
    cntr=dm(st_input_count);
    do stcpy_LP until ce;
        si=dm(i0,m1);
        sr=ashift si (lo);
    stcpy_LP:  dm(i1,m1)=sr0;
    jump stack_cleanup;

    do_stack:
    se=dm(stack_shift);
    cntr=dm(st_input_count);
    do stadd_LP until ce;
        ay0=dm(i1,m0);
        si=dm(i0,m1);
        sr=ashift si (lo);
        ar=sr0+ay0;
    stadd_LP:  dm(i1,m1)=ar;
    stack_cleanup:
        ax0=i1;
        i1=dm(st_output_hptr); 11=0;
        dm(i1,m1)=ax0;
```

70
null_init:
    dm(i0,m0)=ax0;  \{ waveform size \}
    dm(i0,m0)=-1;  \{ waveform count \}
    dm(i0,m0)=ax1;  \{ stack_size \}
    \#if (SIMULATOR == 1)
    cntr=ax0;
    do nclr_lp until ce
    nclr_lp:  \ dm(i0,m0)=0;
    \#endif
    rts;

********************************************************************
* Update background nulling waveform                          *
* i0, m0, 10 - input data                                      *
* i1 - nulling structure                                       *
* Destroys: ax0, ay1, ar, m0, m1, l0, l1                        *
* Data structure format:                                       *
* 0) pointer                                                 *
* 1) count                                                   *
* 2) total                                                   *
* 3) length                                                  *
* (data not adjacent to header in memory)                     *
* Nulling structure format:                                    *
* 0) wave_size                                               *
* 1) wave_count                                              *
* 2) stack_size                                              *
* 3) waveform vector                                         *
********************************************************************)

.var/dm/ram n_input_hptr;  \{ input header pointer \}
.var/dm/ram n_wave_size;
.var/dm/ram n_count;

71
.var/dm/ram n_stack_size;
.var/dm/ram n_wave_ptr;
.var/dm/ram n_null_hptr;

update_null:
  ar=dm(null_flag); ar=pass ar;
  if eq rts;
    m0=0; m1=1; 10=0; 11=0;
    dm(n_input_hptr)=i0;
    dm(n_null_hptr)=i1;
    ayl=dm(i1,m1);  { wave_size }
    ax0=dm(i0,m1);  { input pointer }
    ar=dm(i0,m1);  { input count }
    ar=ar-ay1;
    if lt rts;
      modify(i0,m1);  { skip input total }
      ar=dm(i0,m1);
      i0=ax0;  i0=ar;  { set input buffer length,
        pointer }
    ar=-ay1; m1=ar; modify(i0,m1);  { rewind pointer to beginning
    of data }
    m1=1;

    { save waveform structure header }
    dm(n_wave_size)=ay1;
    ar=dm(i1,m1); dm(n_count)=ar;
    ar=dm(i1,m1); dm(n_stack_size)=ar;
    dm(n_wave_ptr)=i1;

    { clear waveform if n_count is negative }
    ar=dm(n_count); ar=pass ar;
    if ge jump n_dosum;
    ax0=0;
    cntr=dm(n_wave_size);
    do nz_lp until ce;
    nz_lp:
      dm(i1,m1)=ax0;
      ar=0; dm(n_count)=ar;

n_dosum:
  ar=dm(n_count); ar=ar+1; dm(n_count)=ar;

  { sum current waveform with data in waveform structure }
  i1=dm(n_wave_ptr);
  se=dm(null_shift);
  cntr=dm(n_wave_size);
  do ns_lp until ce;
    ay0=dm(i1,m0);
    si=dm(i0,m1);
    sr=ashift si (10);
    ar=sr0+ay0;
  ns_lp:
    dm(i1,m1)=ar;
    i1=dm(n_null_hptr); ay1=i1;  li=0;
    modify(i1,m1);  { skip wave_size }
    ar=dm(n_count); dm(i1,m1)=ar;  { update wave_count }
    ay1=dm(n_stack_size);
    ar=ar-ay1; if lt rts;

#if (0)
{ check if it is time to divide summed waveform }
ar=dm(n_count);  ay1=dm(n_stack_size);
ar=ar-ay1;  if lt rts;
il=dm(n_wave_ptr);
se=dm(null_shift);
cnt=dm(n_wave_size);
do nd_lp until ce;
   si=dm(il,m0);
   sr = ashift si (lo);
nd_lp:  dm(il,ml)=sr0;
#endif

ar=0;  dm(null_flag)=ar;
il=dm(n_null_hpctr);  ll=0;
modify(il,ml);  { skip wave_size }
dm(il,ml)=-1;  { set wave_count to -1 }

sub_background:
m0=0;  ml=1;  10=0;  ll=0;
   { make sure that data count is greater than or equal to wave
size }
ay1=dm(il,ml);  { wave_size }
ax0=dm(i0,ml);  { skip pointer }
ar=dm(i0,ml);  { data count }
ar=ar-ay1;
if lt rts;
modify(il,ml);  { skip wave_count }
modify(il,ml);  { skip stack_size }
modify(i0,ml);  { skip total }
ar=dm(i0,ml);  { buffer length }
10=ar;  i0=ax0;

cntr=ay1;
do sb_lp until ce;
   ax0=dm(i0,m0);
   ay1=dm(il,ml);
   ar=ax0-ay1;
sb_lp:  dm(i0,ml)=ar;
   rts;

******************************************************************
* Subtract background (nulling) waveform                          *
* i0 - data to be nulled                                          *
* il - waveform nulling structure                                *
* Destroys: ax0,ayl,ar,m0,ml,10,ll                                 *
* Data structure format:                                          *
*   0) pointer                                                   *
*   1) count                                                     *
*   2) total                                                     *
*   3) length                                                    *
*   (data not adjacent to header in memory)                      *
* Nulling structure format:                                      *
*   0) wave_size                                                 *
*   1) wave_count                                                *
*   2) stack_size                                                *
*   3) waveform vector                                           *
******************************************************************

sub_background:
m0=0;  ml=1;  10=0;  ll=0;
   { make sure that data count is greater than or equal to wave
size }
ay1=dm(il,ml);  { wave_size }
ax0=dm(i0,ml);  { skip pointer }
ar=dm(i0,ml);  { data count }
ar=ar-ay1;
if lt rts;
modify(il,ml);  { skip wave_count }
modify(il,ml);  { skip stack_size }
modify(i0,ml);  { skip total }
ar=dm(i0,ml);  { buffer length }
10=ar;  i0=ax0;

cntr=ay1;
do sb_lp until ce;
   ax0=dm(i0,m0);
   ay1=dm(il,ml);
   ar=ax0-ay1;
sb_lp:  dm(i0,ml)=ar;
   rts;

******************************************************************
* Decimation FIR filter                                          *
******************************************************************
* i0 - data to be filtered
* i1 - output buffer
* i4 - filter structure
* Destroys: ax0,ay0,ar,m0,m1,m3,m4,10,11,14
* mr,mlx,mxl,17,17,m7
* Data structure format:
  * 0) pointer
  * 1) count
  * 2) total
  * 3) length
  * (data not adjacent to header in memory)
* Nulling structure format:
  * 0) length
  * 1) dec_factor
  * 2) group_delay
  * 3) coefficient vector
******************************************************************
{ ** ---------------------
  FIR Decimation Filter
{ Parameters:
  Filter Length: AX0
  Decimation Factor: AX1
  New Inputs: AYO
  Input Buffer: IO,LO
  Filter Coefficients: I4
Returns:
  il new output buffer pointer
}
.var/dm/ram f_fil_start;
.var/dm/ram f_input_count;
.var/dm/ram f_input_hptr;
.var/dm/ram f_output_hptr;
.var/dm/ram f_group_delay;

fir_dec:
  m0=0; m1=1; 10=0; 11=0;
  dm(f_input_hptr)=i0;
  ax0=dm(i0,m1);   { input pointer }
  ar=dm(i0,m0);   { input count }
  none = pass ar; if le rts;   { return if there is no new data }
  dm(f_input_count)=ar;
  dm(i0,m1)=0;    { set input count to zero }
  modify(i0,m1);   { skip input total }
  ar=dm(i0,m1);
  10=ar; i0=ax0;   { set input buffer length, pointer }
  ar=-ar; m1=ar; modify(i0,m1);   { rewind pointer to beginning of data }
  m1=1;
  ar=i0; i0=dm(f_input_hptr); ay0=10; 10=0;
  dm(i0,m0)=ar;   { update pointer }
i0=ar; 10=ay0;
  dm(f_output_hptr)=i1;
  ax0=dm(i1,m1);   { output pointer }
  ar=dm(i1,m0);   { output count }
  ay1=dm(f_input_count);
  ar=ar+ay1;
  dm(i1,m1)=ar;   { increment output count }
  ar=dm(i1,m0);   { output total }
74
ar = ar + ay1;
dm(i1, m1) = ar;  \{ increment output total \}
ar = dm(i1, m1);
11 = ar;  i1 = ax0;
\{ set output buffer length, pointer \}

i4 = 0;  m4 = 1;
ax0 = pm(i4, m4);  \{ filter length \}
ax1 = pm(i4, m4);  \{ dec_factor \}
ar = pm(i4, m4);  \{ group_delay \}
dm(f_group_delay) = ar;
m4 = ax0;
modify(i4, m4);  m4 = -1;  modify(i4, m4);
dm(f_fil_start) = i4;

AR = ax0 - 1;
AR = -AR;
ay0 = dm(f_group_delay);
ar = ar + ay0;
M3 = AR;
modify(i0, m3);
I7 = I0;  \{ store starting position \}
L7 = L0;
M7 = ax1;
CNTR = dm(f_input_count);
DO dat_dlp UNTIL CE;
  I0 = I7;  \{ start of data \}
  I4 = dm(f_fil_start);  \{ start of filter coefficients \}
  MR = 0;
  MX1 = DM(I0, M1), MY1 = PM(I4, M4);
  CNTR = AXO;
  DO fir_dlp UNTIL CE;
  
fir_dlp:  MR = MR + MX1*MY1 (ss), MX1 = DM(I0, M1), MY1 = PM(I4, M4);
          MR = MR (RND);
          DM(I1, M1) = MR1;
  
dat_dlp: MODIFY(I7, M7);
ar = i1;  i1 = dm(f_output_hptr);  11 = 0;
dm(i1, m1) = ar;  \{ update output pointer \}
RTS;

*****************************************************************************
* Locate peaks in zero-crossing data
* i0 - input header
* i1 - output header
* i2 - data header
* Destroys: ax0, ay1, ar, mr0, m0, m1, i0, i1
* Structure Format: 0) pointer
*   1) count
*   2) total
*   3) length
* (data not adjacent to header in memory)
*****************************************************************************

.var/dm/ram lp_input_count;
.var/dm/ram lp_input_hptr;
.var/dm/ram lp_output_hptr;
loc_peaks:
m0 = 0;  m1 = 1;  i0 = 0;  11 = 0;  12 = 0;

   dm(lp_input_hptr) = i0;
   dm(lp_output_hptr) = i1;
axO=dm(i0,ml);  \{ input pointer \}
ar=dm(i0,m0);  \{ input count \}
dm(lp_input_count)=ar;
dm(i0,ml)=0;  \{ set input count to zero \}
modify(i0,ml);  \{ skip input total \}
ar=dm(i0,ml);
10=ar;  i0=axO;  \{ set input buffer length, pointer \}
ar=dm(lp_input_count);
ar=-ar;  ml=ar;  modify(i0,ml);  \{ rewind pointer to beginning of data \}
ml=1;
ar=i0;  i0=dm(lp_input_hptr);  ay1=10;  10=0;
dm(i0,m0)=ar;  \{ update pointer \}
i0=ar;  10=ay1;

axO=dm(i1,ml);  \{ output pointer \}
ar=dm(i1,m0);  \{ output count \}
ay1=dm(lp_input_count);
ar=ar+ay1;
dm(i1,ml)=ar;  \{ increment output count \}
ar=dm(i1,m0);  \{ output total \}
ar=ar+ay1;
dm(i1,ml)=ar;  \{ increment output total \}
ar=dm(i1,ml);
ll=ar;  i1=axO;  \{ set output buffer length, pointer \}

axO=dm(i2,ml);  \{ data pointer \}
modify(i2,ml);  \{ skip count \}
modify(i2,ml);  \{ skip total \}
ar=dm(i2,ml);  12=ar;  i2=axO;  \{ set data buffer length, pointer \}
ar=dm(lp_input_count);  ar=ar-1;
ay1=dm(i0,m1);  modify(i2,ml);  dm(i1,ml)=0;
loc lp:  mr0=dm(i2,ml);  \{ get data value \}
axO=dm(i0,m1);  \{ get current difference \}
filtered value 
none=pass ay1;
if gt jump pos_check;
neg_check:
none=pass ax0;
if gt jump store_lp;
mr0=0;
jump store_lp;
pos_check:
none=pass ax0;
if le jump store_lp;
mr0=0;
store lp:
dm(i1,ml)=mr0;
ay1=ax0;  \{ current value is used in next iteration \}
ar=ar-1;
if gt jump loc lp;
ax0=i1;
i1=dm(lp_output_hptr);  ll=0;
dm(i1,ml)=ax0;  \{ update output pointer \}
rtsi;

76
Find max and threshold values in buffer

- iO - input header
- il - output header
- ax0 - threshold
- Destroys: ax0, ax1, ay0, ay1, ar, m0, ml, 10, 11
- Structure Format: 0) pointer
- 1) count
- 2) total
- 3) length
- (data not adjacent to header in memory)

```
.var/dm/ram fm_input_count;
.var/dm/ram fm_input_hptr;
.var/dm/ram fm_output_hptr;
.var/dm/ram fm_threshold;
.var/dm/ram fm_skip;
.var/dm/ram fm_max_val;
.var/dm/ram fm_max_index;
.var/dm/ram fm_thresh_val;
.var/dm/ram fm_thresh_index;

find_max_and_thresh:
    m0=0; m1=1; l0=0; l1=0;
    dm(fm_threshold)=ax0;
    dm(fm_skip)=ax1;
    dm(fm_input_hptr)=iO;
    dm(fm_output_hptr)=i1;

    axO=dm(i0,m1);    { input pointer }
    ar=dm(i0,m0);    { input count }
    none = pass ar; if le rts;    { return if there is no new data }

    dm(fm_input_count)=ar;
    ay0=dm(fm_skip); ar=ar-ay0; dm(fm_input_count)=ar;

    dm(i0,m1)=0;    { set input count to zero }
    modify(i0,m1);    { skip input total }
    ar=dm(i0,m1);
    l0=ar; i0=ax0;    { set input buffer length, pointer }
    ar=dm(fm_input_count);
    ar=-ar; ml=ar; modify(i0,m1);    { rewind pointer to beginning of data }
    ml=1;
    ar=i0; i0=dm(fm_input_hptr); ay1=10; l0=0;
    i0=ar; l0=ay1;

    axO=dm(il,m1);    { output pointer }
    ar=dm(il,m0);    { output count }
    ay1=dm(fm_input_count);
    ar=ar+1;
    dm(il,m1)=ar;    { increment output count }
    ar=dm(il,m0);    { output total }
    ar=ar+1;
    dm(il,m1)=ar;    { increment output total }
    ar=dm(il,m1);
    l1=ar; il=ax0;    { set output buffer length, pointer }
```
```c
#if (1)
    ay0=dm(fm_threshold);
#else
    ay0=dm(fm_threshold);
#endif

fm_lp:
    ax0=dm(i0,m1);
    ar=abs ax0;
    if le jump fm_next;
    none=ar-ay0;
    if le jump thresh_check;
        dm(fm_max_index)=ax1;  { save biased index }
        dm(fm_max_val) = ax0;  { save max value }
        ay1=ar;
    thresh_check:
        ar=dm(fm_thresh_val);
        none=pass ar;
        if ne jump fm_next;  { skip threshold check if peak already found }
        ar=abs ax0;
        ar=-ax0;
        if le jump fm_next;
        none=ar-ay0;
        if lt jump fm_next;
            dm(fm_thresh_index)=ax1;  { save biased index }
            dm(fm_thresh_val) = ax0;  { save threshold peak value }

fm_next:
    ar=ax1-1;
    ax1=ar;
    if gt jump fm_lp;
    ay1=dm(fm_max_index);  ar=dm(fm_input_count);
    ar=ar-ay1;  { unbias max index }
    ay0=dm(fm_skip);  ar=ar+ay0;
    dm(fm_max_index)=ar;
    ay1=dm(fm_thresh_index);  ar=dm(fm_input_count);
    ar=ar-ay1;  { unbias thresh index }
    ay0=dm(fm_skip);  ar=ar+ay0;
    dm(fm_thresh_index)=ar;
    ax0=i1;
    i1=dm(fm_output_hptr);  ll=0;
    dm(i1,m1)=ax0;  { update output pointer }
    rts;
```

References:

```c
{**********************************************************************
* Integer to ASCII hex string conversion *
* i0,m0,10 - output buffer *
************
```
* ax1, ax0 - number to convert

num_to_hex:
  dm(input)=ax1;
  dm(input+1)=ax0;
  
  { ar=48; dm(i0,m0)=ar; dm(i0,m0)=ar; }  
  ax0=dm(input);
  call int_to_ascii_hex;
  
  { ar=dm(digit); dm(i0,m0)=ar;   
    ar=dm(digit+1); dm(i0,m0)=ar;   
    ar=dm(digit+2); dm(i0,m0)=ar;   
    ar=dm(digit+3); dm(i0,m0)=ar;   
    ax0=dm(input+1);    
    call int_to_ascii_hex;
  }
  ax0=dm(input+1);
  call int_to_ascii_hex;
  
  { ar=dm(digit); dm(i0,m0)=ar;   
    ar=dm(digit+1); dm(i0,m0)=ar;   
    ar=dm(digit+2); dm(i0,m0)=ar;   
    ar=dm(digit+3); dm(i0,m0)=ar;   
  }
  rts;

int_to_ascii_hex:
  mr0=ax0;
  sr=lshift mr0 by -12 (lo);
  call digit_to_hex;
  dm(i0,m0)=ar;   
  ay0=0x0F00; ar=ax0 and ay0;
  sr=lshift ar by -8 (lo);
  call digit_to_hex;
  dm(i0,m0)=ar;   
  ay0=0x00F0; ar=ax0 and ay0;
  sr=lshift ar by -4 (lo);
  call digit_to_hex;
  dm(i0,m0)=ar;   
  ay0=0x000F; ar=ax0 and ay0;
  sr=ar;
  call digit_to_hex;
  dm(i0,m0)=ar;   
  rts;

digit_to_hex:
  ay0=0xA; ar=sr0-ay0; if ge jump letter_digit;
  ay0=48; ar=sr0+ay0; rts;

letter_digit:
  ay0=55; ar=sr0+ay0; rts;

  { Input: sr0, Output: ar }

digit_to_hex:
  ay0=0xA; ar=sr0-ay0; if ge jump letter_digit;
  ay0=48; ar=sr0+ay0; rts;

letter_digit:
  ay0=55; ar=sr0+ay0; rts;
.var/dm/ram digit_n2a;  
.var/dm/ram dig_index_n2a;  
num_to_ascii:  
  ay1=0;  
  ay0=0;  
ar=ax0+ay0;  
  mr0=ar, ar=ax1+ay1+c;  
ax1=ar;  
  ax0=mr0;  
ar=-1;  
  dm(digit_n2a)=ar;  
ar=0;  
  dm(dig_index_n2a)=ar;  
  dm(start_n2a)=10;  
  { save start - used by n2a_bad }

  { Subtract y (ayl,ay0) from x (ax1,ax0) }  
n2a_s:  
ar=0;  
  af=pass axl;  
if lt ar=ar+1;  
  mr0=ar;  
  { mr0 is used to flag MS bit }  
  ay1=dm(fac_n2a);  
  ay0=dm(fac_n2a+1);  
ar=ax0-ay0;  
  ax0=ar, ar=ax1-ay1+c-l;  
ax1=ar;  
ar=dm(digit_n2a);  
  ar=ar+1;  
  dm(digit_n2a)=ar;
  ar=pass axl;  
if ge jump n2a_s;  
  ar=pass mr0;  
  if ne jump n2a_s;  
  { flag check }  
  { Got digit in digit_n2a now, increment digit index }  
ar=dm(dig_index_n2a);  
  ar=ar+1;  
  dm(dig_index_n2a)=ar;  
  { Check to see if digit > 9 }  
ar=dm(digit_n2a);  
  ay0=9;  
  ar=ar-ay0;  
  if gt jump n2a_bad;  
  { Save digit }  
ar=dm(digit_n2a);  
  ay0=48;  
  ar=ar+ay0;  
  dm(i0,m0)=ar;

  ar=dm(dig_index_n2a);  
  ay0=6;  
  ar=ar-ay0;  
if ge jump n2a_done;  
  { Done last digit? }  
ar=dm(dig_index_n2a);  
  ay0=3;  
  ar=ar-ay0;  
if ne jump n2a_no_dec;  
  { Reached decimal point? }  
ar=46;  
  dm(i0,m0)=ar;  
  { Insert decimal point }  
  jump n2a_no_dec;

n2a_no_dec:  
  { Reset digit }  
ar=-1;  
  dm(digit_n2a)=ar;  
  { Update ax1, ax0 for next digit by finding remainder and  
    multiplying by 10 }  
  ay1=dm(fac_n2a);  
  ay0=dm(fac_n2a+1);  
ar=ax0+ay0;  
  mx0=ar, ar=ax1+ay1+c;  
  mx1=ar;  
  { remainder is in mx1,mx0 now }  
  my0=5;  
  { Do multiply }  
  mx=mx0*my0 (uu);  
  ax0=mx0;  
  mx0=mx1;  
  mr1=mx2;  
  mx=mx+mx1*my0 (uu);  
  ax1=mx0;
jump n2a_s;

n2a_done:
    rts;

    { set to 999.999 for overflow }

n2a_bad:
  i0=dm(start_n2a);
  ar=9+48;
  dm(i0,m0)=ar;  dm(i0,m0)=ar;  dm(i0,m0)=ar;
  ar=46;  dm(i0,m0)=ar;  ar=9+48;
  dm(i0,m0)=ar;  dm(i0,m0)=ar;  dm(i0,m0)=ar;
  rts;

.endmod;