DESIGN OF A SEISMIC DATA ACQUISITION SYSTEM AND AUTOMATIC TRIGGERING SOFTWARE

Robert Sidney John Cole

A dissertation submitted to the Faculty of Science
University of the Witwatersrand, Johannesburg
for the degree Master of Science

Johannesburg 1991

DECLARATION

I declare that this dissertation is my own, unaided work. It is being submitted for the Degree of Master of Science in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any other degree or examination in any other University.

R.S.J. Cole

(Robert Sidney John Cole)

_____ day of _____ , 1991

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation and gratitude to:

Rod Green for the supervision, advice, and support given for the duration of this project.

Marius Cronje for valued advice and discussion related to the project.

John Sorour - for the artwork for the final PC-boards and help with debugging the prototype.

Jeremy Maccelari - for help with programming and keeping me sane.

Gary Agnew - for suggestions for signal processing.

Doug Banks - for hours of discussion and endless cups of tea.

My parents - for putting up with an eternal student for the last few years.

Abstract

The recording of seismic signals in remote areas requires a portable, low power recording system that can be left in the field for a few weeks at a time. Three components of ground motion are generally measured, and some form of event recording, rather than continuous recording should be available. When continuous recording is used some form of mass medium backup, such as tape streamer could be used. Low power microprocessors are ideal for this sort of application. They offer the versatility and ease of modification required to tailor a data acq isition system for a specific survey problem. This dissertation describes such a system using low power microprocessors and low power random access memory (RAM). Gain ranging techniques are used to achieve an overall dynamic range of 126 dB. A time synchronisation method using a standard FM radio and time "pip" decoder achieved a time correlation between stations of better than 10 mS. Power requirements are low, the acquisition system requiring 2,4 W. The on board memory can be expanded up to 6,144 Mbytes and the maximum sampling rate (for three channels) is 2400 Hz. Various triggering algorithms were explored, and event recording was found to be practical for local events. Teleseismic triggering was investigated but a continuous recording system is suggested for recording teleseisms.

Contents

List of Figures

Chapter 1

Introduction

In 1968 an analogue tape based portable seismic data acquisition system was developed in the Bernard Price Institute (BPI) for field use, Green(1979). With the availability of cheap microprocessors and portable computers, this was replaced by a microprocessor based system in 1986 which is presently being used in the field. This system uses an 8-bit analogue to digital converter, yielding a dynamic range of 48 dB plus another 30 dB by using compression techniques. The present system also requires a fair amount of power for all its needs (intermittent data transfer is to a standard PC) and is powered by 2 car batteries which are charged from solar panels. Observations and theoretical considerations of rock dynamics suggest that a dynamic range of 120 db to 140 dB is needed to encompass all types of earthquake studies. A new system with greater dynamic range, and lower power consumption is required to replace the existing system.

1.1 Aims

According to Ambuter (1974) "To monitor earthquakes ... in remote areas, a seismic recording system should operate at low power for extended periods of time, should reproduce three components of ground motion with a large dynamic range". This design looks specifically at the dynamic range prob-

lem, and also at using event recording (automatic triggering) techniques to make more efficient use of storage capacity. The aims were to produce a low power seismic data acquisition system with high dynamic range to replace the present 8-bit stations, and to investigate the use of various automatic triggering algorithms.

Principal design requirements were low power consumption and to record three component, long period and short period, seismic data at a large dynamic range for a few weeks at a time. The form of mass storage will depend somewhat on the study being done. The stand alone system should have 6 Mbytes of RAM for storage. This is enough for a system incorporating automatic triggering and it could be left in the field for a number of weeks. The data should also be able to be downloaded to a PC as is done in the present 8-bit system. Some method of entering data such as time, station number and sampling rate, as well as the display of such data, is needed. To achieve a dynamic range of more than 120 dB an analogue to digital converter with 13 bit resolution (12 bits + sign) coupled with a gain switching, or gain ranging, system was employed. Three gain bits are used giving 7 possible gain steps, and if gain steps of 12 dB are used a theoretical gain of 66 dB can be achieved. This means a total dynamic range of 142 dB would be possible. The system requires that the real time is accurate to at least 10 mS. A further aspect of time accuracy is that of time synchronisation between the field stations. The stations also need to be synchronised to better than 10 mS. The problem of triggering on both local and teleseismic events need the capabilities of real time data analysis of the incoming signal. The use of a microprocessor greatly simplifies this.

1.2 Description of existing system

At present, BPI has five of these 8-bit seismic stations in the field and another at the BPI. They consist of an 8 bit analogue to digital converter and associated analogue electronics (anti-alias filter), a real time clock and display, three processor boards (one for each channel) each with a 32 Kbyte buffer

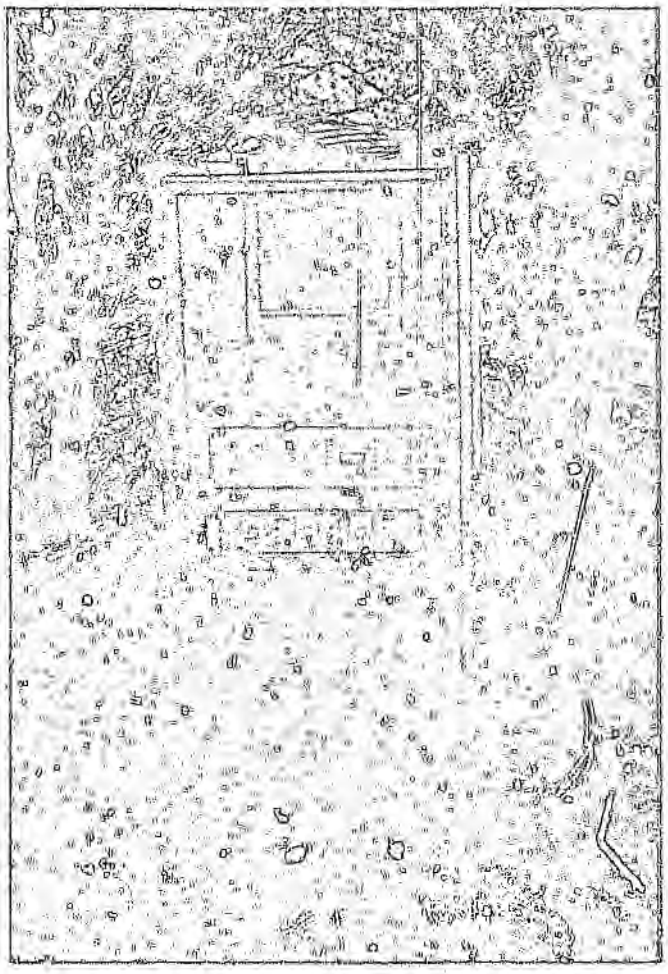


Figure 1.1: Picture of a typical field station

and a PC - XT clone computer and associated software. The PC system consists of a standard PC with an 8255 input port for data transfer between the processor boards. A 40 Mbyte hard disk is used as temporary data storage and a 60 Mbyte tape streamer for mass storage once the disk has 30 Mbytes stored. Power is supplied by two car batteries which are charged by solar panels. This section has been included as some valuable lessons were learnt from the problems encountered in the field with this system. Figure(1.1) shows a typical field station. The system specifications for the present 8-bit system are as follows:

- 3 Channels
- Sampling rate of 14 Hz
- Dynamic Range = 88 dB
- Buffer Memory = 28 Kbytes per channel

- Power consumption (not including PC) 8,6 W
- Time accuracy = 1 part in 10^8

1.2.1 Operation of the system

The Analogue to digital converter (A/D) is sampled in real time, with a real-time clock providing the reference sampling rate. Data is read from the three component amplifiers by three processor boards, and the data stored in each of the 32 Kbyte buffers. When 28 Kbytes of data have been generated, the master processor board triggers the real time clock which latches the real time and turns on the PC. After booting up the data is downloaded from each of the three acquisition cards and the time and date that had been latched is transferred as the filename of the data file. When the hard drive is nearly full the data files are transferred to tape streamer, and the files deleted from the hard disk.

1.2.2 Problems Encountered

Two major problem areas encountered with the present system were, power requirements, and hard drive reliability.

Power Problems

The original solar panels used did not always provide enough power to keep the batteries topped up if it was overcast for a few days. This was solved by using more solar panels. From experience with the 8-bit stations, assuming 6 hours of daylight, it was found that the mean power delivered by the solar panels must be 4 times the peak power drawn by the equipment.
For the 8-bit system : Vcc = 24 V

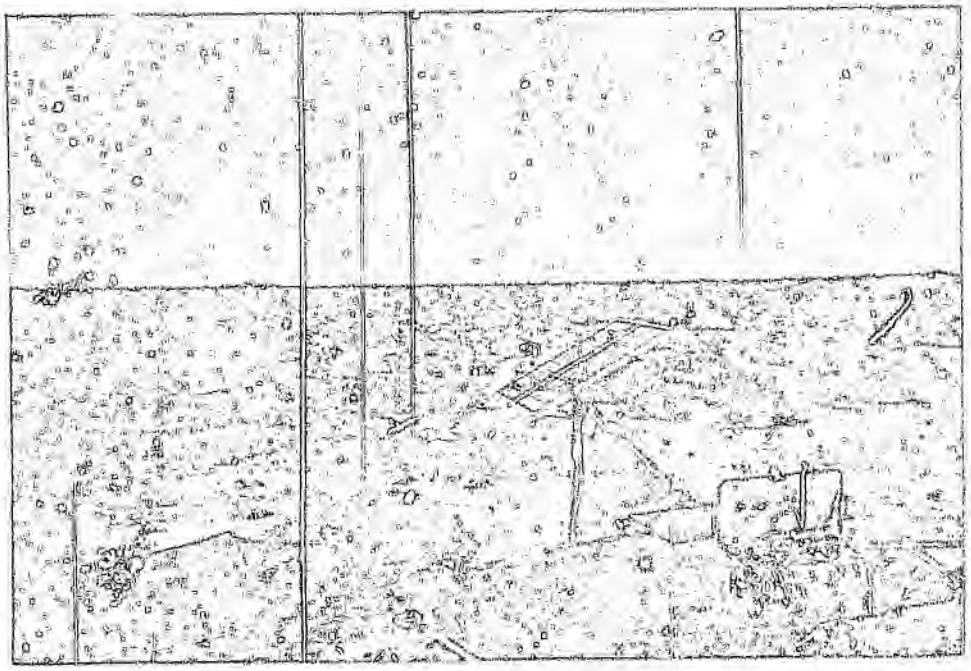


Figure 1.2: Two 48 Watt solar panels

-PC switched on for 100 seconds every 30 minutes - draws 3,6 A

$$\frac{3,6A \times 100S * 24hours}{1800S} = 4,8A.h.day^{-1}$$

- Continuous current - 360 mA

$$360mA \times 24V = 8,6W$$

- Total power

$$8,6W + 4,8A.h.day^{-1} = 13,4W$$

So for the solar panels

$$13,4 \times 4 = 53,6W$$

Initially only two 48 W panels were used, and power problems resulted as the batteries were being deep cycled. Figure(1.2) showing two 48 Watt solar panels.

It was found in fact, that in order to be safe, four panels needed to be installed. Subsequently the system has operated perfectly, even during fairly long periods (4-8 days) of overcast sky. In addition the batteries have lasted for more than a year.

The Hard Drive

The hard drive initially gave some problems, but they were probably related to the DC to DC convertor (DC-DC) which was unreliable. This is not an ideal backup medium especially when power may be a problem. When the drive is switched on up to 48 times a day a reliable power supply is needed to cope with the starting surges.

1.3 The 16-bit System

The new 16-bit system was designed, bearing in mind some of the problems of the 8-bit system. The new system can operate in a stand alone mode, with suitable triggering software for local events. However for teleseismic studies, a similar setup as for the 8-bit station's is adopted. Power problems are not as severe. The use of a large buffer memory means that the PC can be switched on less often (also better for the hard drive) and total power drain is less.

Because of the power constraints for the unit, CMOS Integrated circuits are to be used throughout. Unfortunately no DSP (Digital signal processing) chip was available that meets our power requirements, so digital signal processing had to be kept to a minimum. The microprocessor chosen for the design was the Hitachi 6309 (A CMOS version of the Motorola 6809). A 16-bit processor (such as the Motorola 68000 series) was considered. However all our design criteria, including the trigger algorithm could be met using the 6309 so it was chosen.

Time synchronisation of the systems was achieved by using a Real Time Clock (Accurate to 100 mS over a few days). By synchronising the clock to the standard commercial SABC time signal, given on the hour on the FM country wide stations, it was possible to improve the time correlation to better than 10 mS between stations.

The software for the acquisition unit was written in a modular form, so that future modifications to the system software can be easily achieved.

For comparison with the operation of the 8-bit system, a sampling rate of 16 Hz was chosen. The Buffer Length using standard 32 Kbyte CMOS RAM chips is 364 Kbyte/card. So with four cards installed and a 16-bit data word length and a sampling rate of 16 Hz for 3 Channels the time is :

$$= \frac{384KByte \times 4cards}{(16samples/sec).(3channels).(2Bytes/sample)} = 16000seconds$$

$$= 4 \text{ Hours } 26 \text{ minutes}$$

This means that the PC is turned on roughly every 4 and a half hours. At 12 V The PC draws 7,2 A. Because a larger buffer length is utilised the transfer time is longer, requiring the PC to be on for 200 seconds. So the PC energy requirement per day is :

$$\frac{7,2A \times 200 \times 24}{16000} = 2,16A.h.day^{-1}$$

The continuous power drain from the system electronics is 200 mA with a system voltage of 12 V. - Continuous power

$$P = 200mA * 12V = 2,4W$$

Therefore the total energy requirement for the system is :

$$E = 2,16A.h.day^{-1} + 2,4W = 4,56A.h.day^{-1}$$

For the solar panels, experience has shown that considering bad cloud cover and short days, you can average throughout the year a minimum of 6 hours of effective power generation per day. To keep the batteries from a deep cycle operation and thereby increase their life (and reliability) you should only cycle to 80% of the capacity of the batteries. In addition you need to have energy capacity for at least 5 days. In the present case this means $4,8A.h.day^{-1} \times 5days = 24A.h.$ If you cycle to 80% then the worst case drain is over 18 hours (24-6 hours) and the energy requirement is $0,2A \times 18h = 3,6A.h$ Thus for 80% cycling the capacity should be 18A.h. This is less than

the 24 A.h already calculated. The solar panels should exceed the daily input of $4,8 A.h.day^{-1}$ by at least 20% so that means $5,8 A.h.day^{-1}$ is required from a panel operating a minimum of 6 hours. Thus the panel needs to be able to supply 1 A at the rated 12 V or 12 W. One solar panel will be sufficient for the power needs.

1.3.1 Dynamic Range

According to Green (1984) mine observations and theoretical considerations of rock dynamics suggest that the range of velocity to be measured will be between 10^{-3} mm/sec (noise) and 5000 mm/sec (±6dB) (extrapolation). The transducer therefore needs a dynamic range of 134 dB. No standard transducer can provide the entire range and transducers will have to be selected for the study in hand. The analogue to digital converter will need a range of between 120 db and 140 dB. This translates (in bit terms) to a 21 or 22 bit analogue to digital converter!

A 22 bit converter is excessive in terms of precision, and various methods have been used to compromise between the useful dynamic range and precision. A top compression amplifier designed by R.W.E. Green was used in the 8-bit system. This method employs the logarithmic effects of diodes in the feedback loop of an operational amplifier, to provide the log response. Transistors are used in place of diodes because it is easier to find transistors with matching V_{BE}'s than matching diodes. See figure(1.3). More than one slope can be incorporated by using more than one feedback loop. See figure(1.4). Under low signal conditions the feedback resistor Rf_1 provides the feedback and the gain is simply $\frac{-Rf_1}{R_{in}}$. When the input signal is larger than the V_{BE} turn on voltage, the base emitter junctions of Q_{1a} and Q_{1b} start conducting and the circuit then exploits the fact that I is logarithmically related to V through a diode. The resistors Rf_1 and Rf_2 are also now in parallel. Transistors Q_{2a}, Q_{2b}, Q_{2c} and Q_{2d} will start conducting at $2 \times V_{BE}$ and the circuit then follows the log squared response. Problems with this approach are that transistors have to be selected for similar V_{BE}'s. This approach could also be used for the 16-bit system providing at least 40 dB of gain giving a total

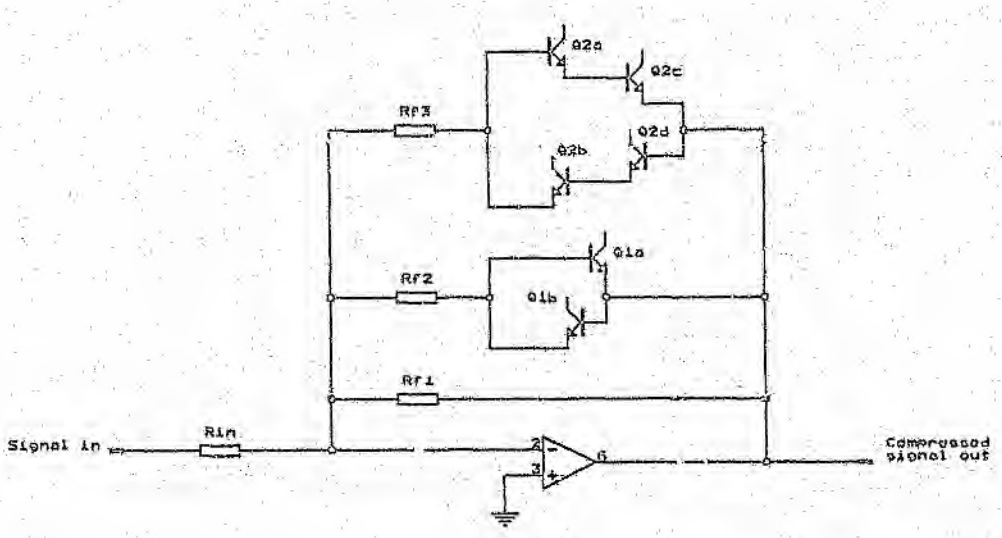


Figure 1.3: Top compression amplifier

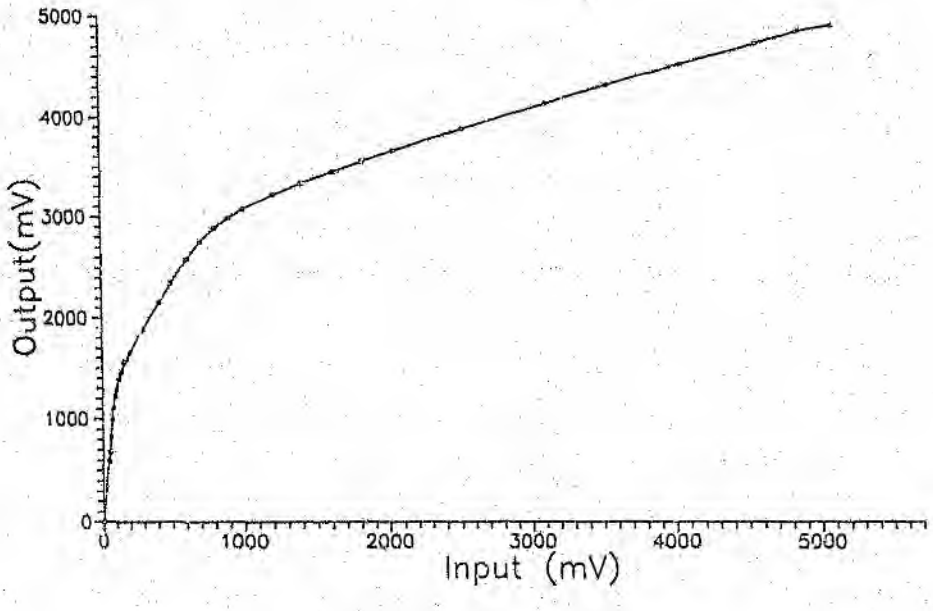


Figure 1.4: Response from the top compression amplifier

dynamic range of 124 dB. This method works well with matched transistors. However unless separate amplifiers have transistors with similar characteristics, when the data is deconvolved a separate (and specific) look up table would have to be used for each amplifier. This is impractical if a precision of better than 1-2% is required. The circuit also has temperature effects of about 0,1%/oC.

1.3.2 Gain Ranging

Another way of attaining this dynamic range is to use a gain ranging technique before the A/D converter. Figure(1.5) shows such a system. The very fast comparators, operating as flash converters, select the gain by utilising a D/A (digital to analogue converter) ladder line network to vary the feedback resistor of an operational amplifier(op-amp). Any errors in the step say due to offset problems will mean that the optimum step (-4 V to +4 V) in this case, is not utilised. The output will however not reflect this offset/noise error, as the output is wholly independent of these errors.

The incoming signal passes through a variable gain amplifier, whose gain is determined by the state of the various comparators U_1 - U_6. The input to these comparators is the rectified incoming signal. Comparator U_1 will switch when the input is greater than $Vref_1$. Similarly for U_2 to U_6. The comparator's switching points are selected so as to allow the maximum swing possible between gain steps. Stability of the comparators is ensured by including a certain amount of hysteresis. The decoding circuitry selects the gain step corresponding to the last comparator that has switched. This is decoded into 3 bits which are incorporated with the 13 bit data from the A/D as a 16 bit word. These bits are decoded for the amplifier (via a digital to analogue converter ladder line network) to gain steps of 6 dB. The operation of the ladder line network is shown in Figure(1.6). Depending on the selected analogue switch, the gain of the op-amp can be varied. Using a 12 bit converter a range from -6 dB to +66 dB (six 12 bit steps) was tried, but later the last two steps were decreased from 12 dB steps to 6 dB steps giving the range, -6, +6, +18, +30, +42, +48, +54 dB. The gain was decreased because

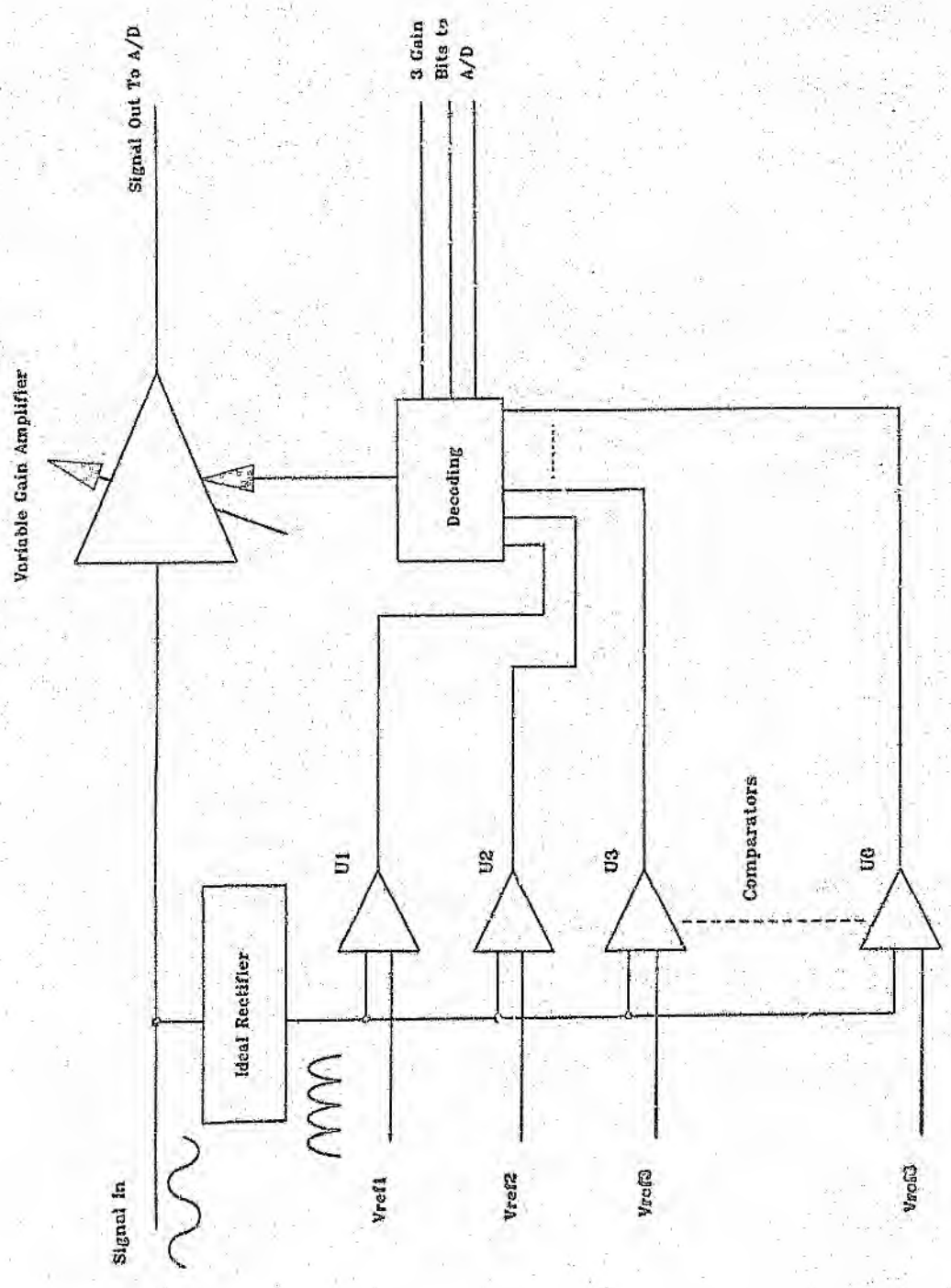


Figure 1.5: The gain ranging concept

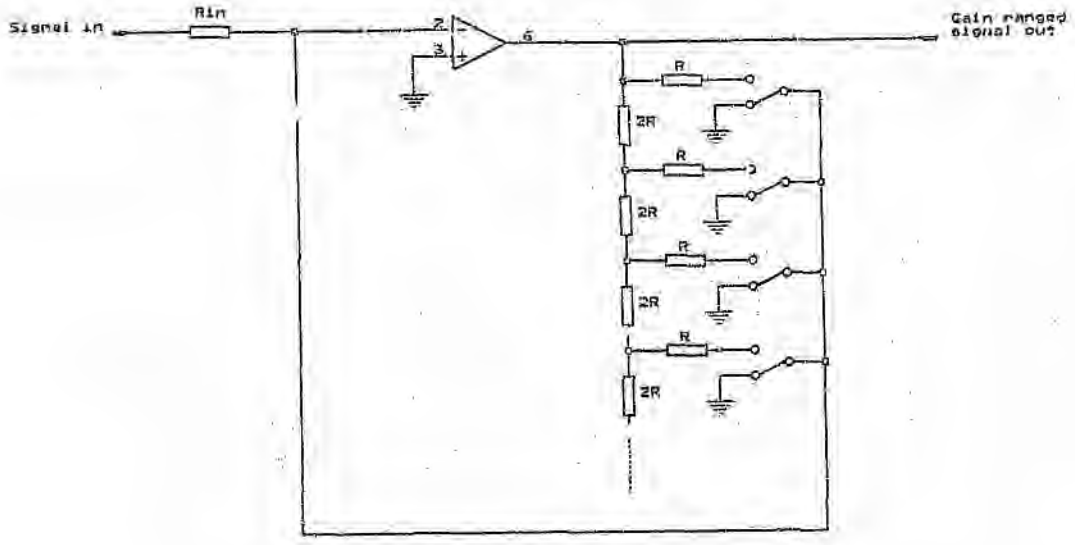


Figure 1.6: Variable gain amplifier

of noise problems with the higher gain stages. This corresponds to gains of $\times\frac{1}{2}, \times 2, \times 8, \times 32, \times 128, \times 256 \times 512$. This gives a theoretical dynamic range of 60 dB for gain ranging and a total of 138 dB when using a 13 bit analogue to digital converter. This is within the values stated as our stipulated range. Figure(1.7) shows the effect of the gain steps on a sine wave input and the reconstructed signal.

1.3.3 Triggering

A Pascal program was written using data from the system to test various triggering algorithms. The criterion for triggering is dependent on the particular study being done. In all cases however the effect of spikes, and other noise must not cause a trigger. Local events were successfully triggered with a simple long term short term averaging (LTA/STA) technique. Differentiating between long period teleseisms and other signals proved to be much more of a problem. A suitable scheme is described by Evans and Allen [1983]

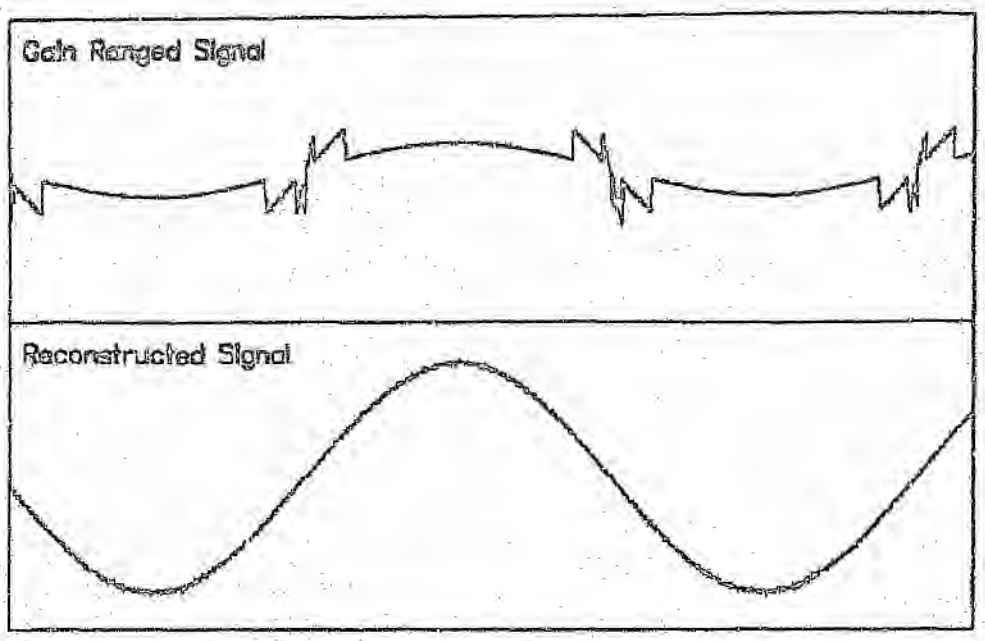


Figure 1.7: Gain Ranging, showing digitised and compressed sine wave, and the reconstructed signal

and is discussed but has not been tested. It was also felt that for teleseisms it was more important that no events be missed, than for memory saving. Continuous recording and dumping the data to a PC (as done in the 8-bit system) was implemented.

1.4 System Overview

Figure (1.8) shows the block diagram of the complete data acquisition system. The operation of the system, has been split into three logical sections. The data acquisition section is referred to as the DAC section. The [illegible] and real Time Clock as the CONTROL section. The memory [illegible] and peripheral output card as the MEMORY section.

This modular form of construction was felt to offer advantages in terms of further modifications. In fact some of the design changes, especially with respect to the time update, would have been much more difficult to achieve if a single processor had been used. The main reason for the multiple processor design was to free the DAC processor for as much data processing as

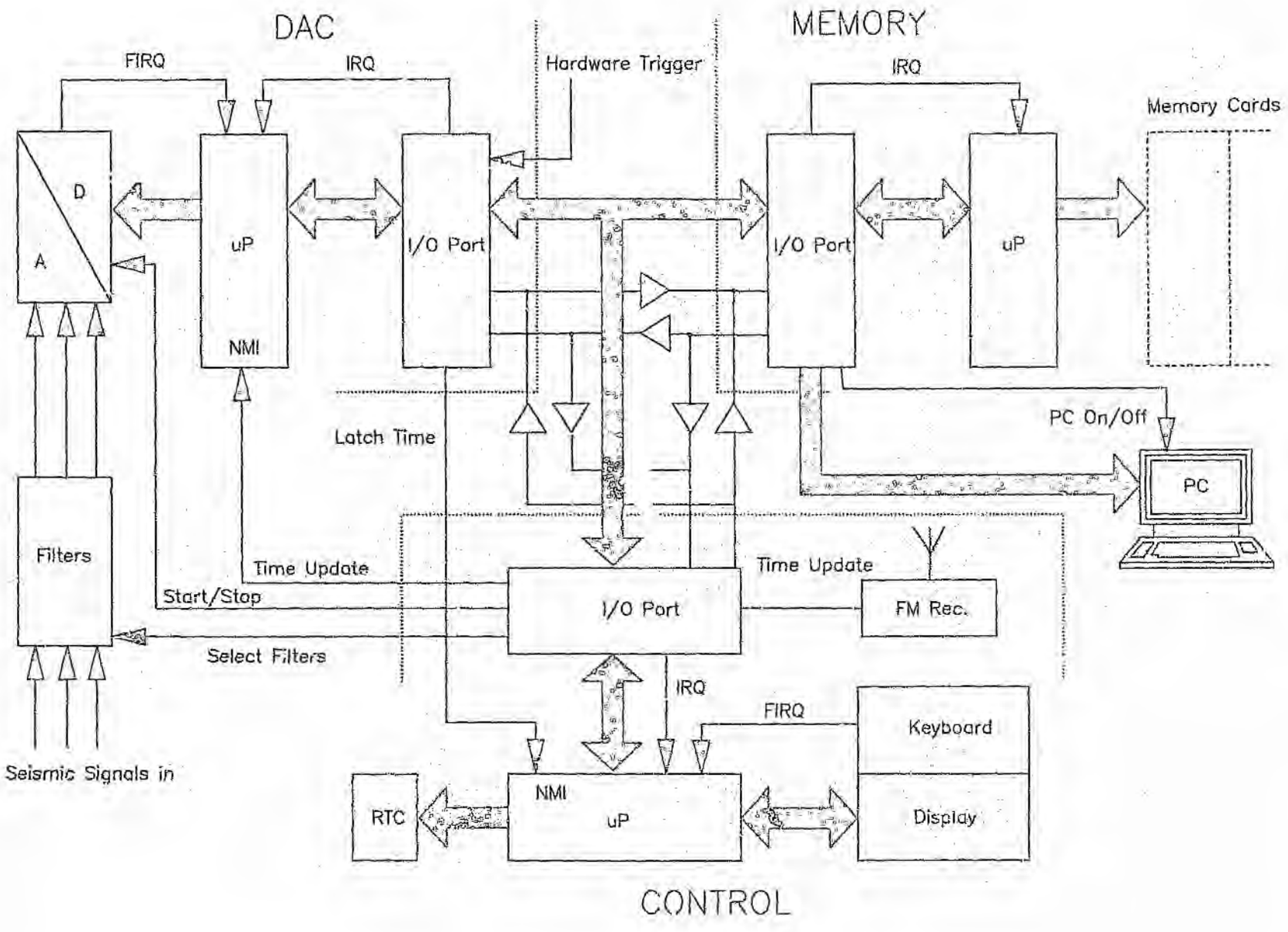


Figure 1.8: Block diagram of acquisition system

possible. The everyday upkeep such as time update, user input, time display and data direction selection is taken care of by the control board completely independently of the other boards. The data transfer to the PC from the MEMORY board is again completely independent of the other boards. Software debugging and modification is also much easier with this distributed system approach. Data transfer is slightly more complicated but a suitable handshaking scheme has been implemented and works very reliably. The rate of data transfer between boards is approximately 14 Kbyte/second. A shared memory scheme was considered with the MEMORY and DAC boards sharing the same memory. Memory access is slowed down however, so this did not seem to offer much advantage. The operation of each of the sections will be discussed in turn.

1.4.1 The DAC section

The 3 inputs are first filtered with an anti-alias filter, before passing to the gain ranging stage. The anti-alias filter frequency is selected by the control board depending on the sampling frequency. The A/D is clocked by a square wave from the real time clock, at the selected sampling rate. The follow and hold gates are opened at this time. The signals are then sampled sequentially by the A/D. A fast interrupt request (FIRQ) initiates a read cycle from the DAC microprocessor. Because the time from the real time clock can only be read to the nearest second, some internal timing is necessary to keep track of the samples between the seconds. This is accomplished by only allowing the A/D to start sampling on the transition of a second, and then a count of the number of samples (the "dragon count") is kept. When this count equals the selected sample rate (ie. the end of the next full second) the counter is reset. The data is read into three 28 Kbyte buffers. When the buffers are full (or a trigger is recognised) the "latch time" line on the I/O Port is used to initiate a non maskable interrupt (NMI) sequence on the CONTROL board which latches the time. The data is then transferred to the MEMORY board along with the "dragon count". The "latch time" line then initiates a second NMI sequence on the CONTROL board, and the latched date and time data is transferred from the CONTROL to the MEMORY board. A

hardware trigger input is also available which will also initialise the data transfer procedure described above.

1.4.2 The CONTROL section

This section does the upkeep such as user input, real time clock(RTC), display and data direction control. The operation of the CONTROL board is shown in the flow diagram, figure(1.9).

The various devices such as the RTC, display, and I/O ports are first initialised. The real time clock chip has 6 internal registers that are updated automatically giving the date and time (year, month, date, hours, minutes and seconds). The control program then checks for a response from the DAC and MEMORY boards to ensure that they have reset correctly. The user inputs such as time, date, and sampling rate are entered on the keyboard and sampling initiated. The Control card initiates the A/D sampling on the transition of a second by changing the state of the start/stop line. The time is latched by the CONTROL board when the trigger line from the DAC board is pulled high. This generates a non maskable interrupt (NMI) sequence that latches the real time. The DAC board then transfers the seismic data to the MEMORY card. Once the data is transferred a second NMI sequence on the CONTROL board is initiated and the CONTROL board transfers the date, time, sampling rate and station number to the MEMORY card. Absolute time synchronisation between stations is achieved by using a FM radio and time "pip" decoder described in section (2.7). The output from the radio is taken to the I/O port on the DAC card and the program checks the port for an update in a window of 10 seconds about every hour. If a sync pulse occurs the seconds are reset and the minutes updated if necessary. One problem involved with this method of data synchronisation is that if the time from the real time clock varies, and is corrected on the CONTROL board the "dragon counter" for the DAC board will no longer be synchronised with the seconds. To overcome this problem, a line from the I/O port is used to generate a NMI sequence on the DAC card. The NMI sequence resets the counter thereby keeping the time and counter in step.

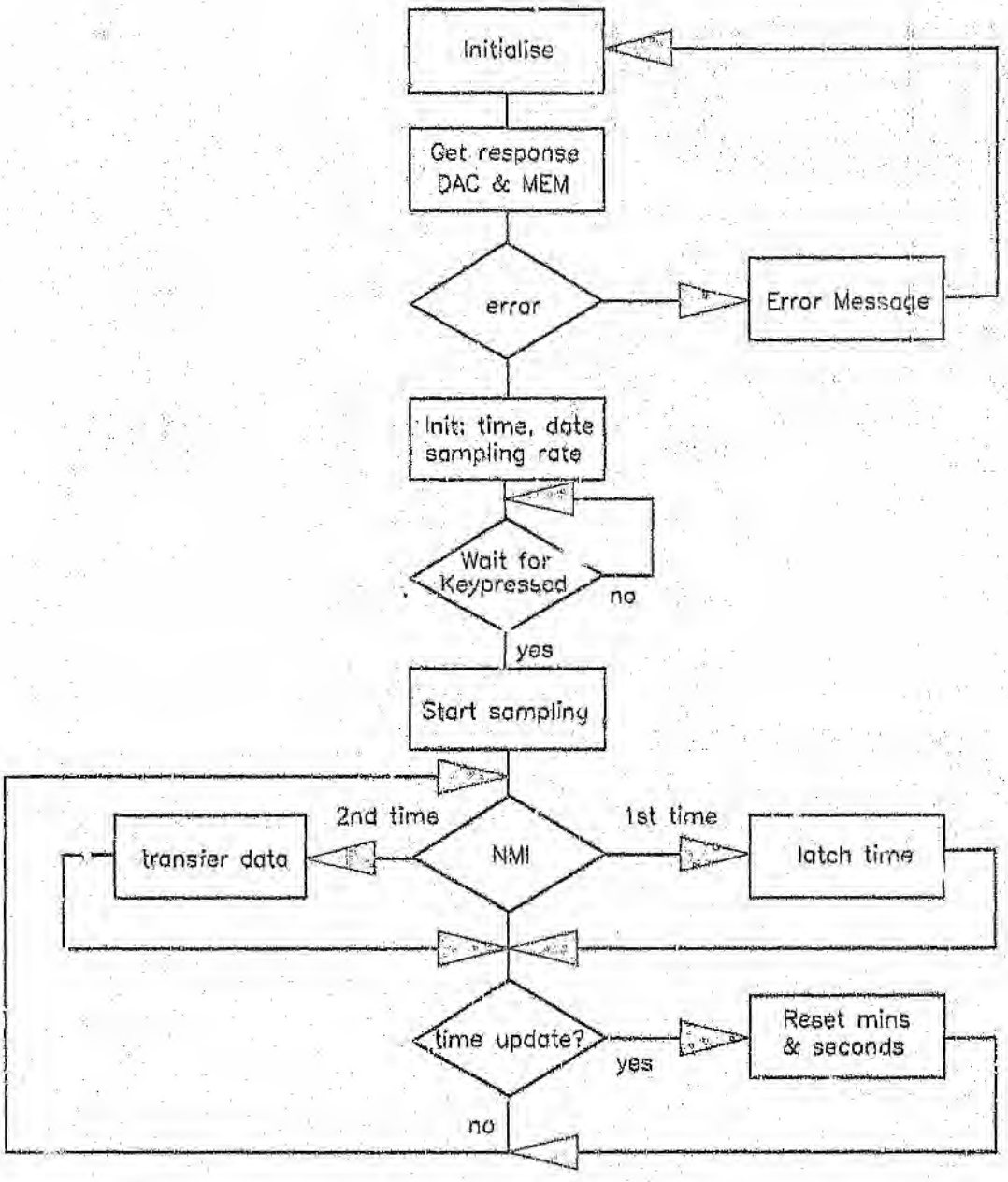


Figure 1.9: flow diagram for CONTROL section

Having utilised three microprocessors for the design rather than one 16-bit processor meant that communication between the boards was complicated. The 3 boards are connected via a common bus and the control of the handshake lines is done by the CONTROL board. Any one of the boards can be set up to communicate with either of the others. However care must be taken to ensure that the card not being used is in the tri-state mode. The handshake lines are connected to CMOS 4016 switches. These switches are selected via the CONTROL output port to allow data transfer between 2 of the boards, the other handshake lines being kept tri-state. The IRQ or interrupt request line on the microprocessor is used for the data communication between the boards.

1.4.3 The MEMORY section

The MEMORY section is used for the bulk storage (up to 6 Mbytes) of data, and for transferring data to peripherals.An output line is available that can be used to switch a relay to power up any bulk storage device such as a PC or tape streamer. Once the buffer memory is full, the MEMORY microprocessor will transfer the contents of the memory to the peripheral device.

1.4.4 Initialisation

The initialisation (hardware reset) process is as follows : All three boards go through their own initialisation and then the CONTROL board sets up data transfer MEMORY to CONTROL. The MEMORY board then transfers a status byte to indicate that it has reset correctly. The CONTROL board then waits for user input of : Station code, sampling rate, date and time. Once this is entered the CONTROL board sets up CONTROL to DAC and transfers the sampling rate to the DAC board. It then sets up transfer from DAC to MEMORY and starts the A/D on the transition of a second.

Chapter 2

Description of system

In this chapter the specifics of each of the three processor boards will be discussed, together with a discussion of the analogue components of the system : the analogue to digital converter, filters, gain ranging circuit, time synchronisation card and power supply.

The design of the microprocessor hardware was split into three parts, namely
- The data acquisition and processing hardware (DAC)
- Mass storage and Data transfer (MEMORY)
- Control and timing (CONTROL)

The general philosophy behind this was to use fairly simple microprocessor boards for each of these functions, and in this way expanding the system is simplified. For example Direct Memory Access (DMA) could be used for faster data transfer, or alternately the data acquisition hardware could be replaced with a digital signal processing (DSP) chip . None of these modifications would require a complete redesign of the system.

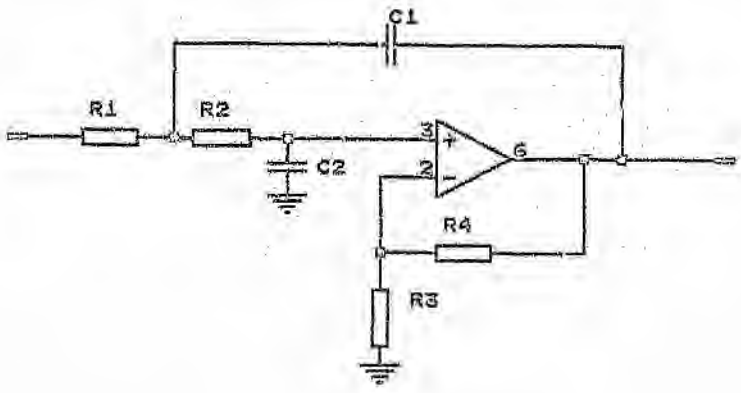


Figure 2.1: VCVS 2 pole stage

2.1 Anti-alias filters

A 6 pc. Butterworth filter was used for the anti alias-filters. Figure(2.2) shows the circuit diagram for the filter board. The Butterworth response is acceptable in terms of amplitude response, and is somewhat of a compromise in terms of phase response. The attenuation of a Butterworth filter is 6n dB/octave, where n is the number of poles. For the 6 pole configuration a slope of 36 dB/octave is obtained. A VCVS (voltage controlled voltage source - a variation of the Sallen- Key filter) was used as the building block for the 2 pole stages of the filter. The gain for this filter is given as $K = 1 + \frac{R_4}{R_3}$. Figure (2.1) shows a 2 pole filter stage. Three of these stages are cascaded to form the higher order filter. Each section of the filter represents a quadratic polynomial describing the response of the overall filter. Tables for a 6 Pole Butterworth response were obtained from Horowitz and Hill (1976) and these gave gain values of 1,068 1,586 and 2,483 for the 3 stages with a 3 dB cut at $RC = \frac{1}{2\pi F}$

The buffer stage has a gain of 1. Low offset operational amplifiers (op-amps) must be used throughout. The filters are cascaded with the gains as given above. The overall gain of the filter board is then ×4,2.

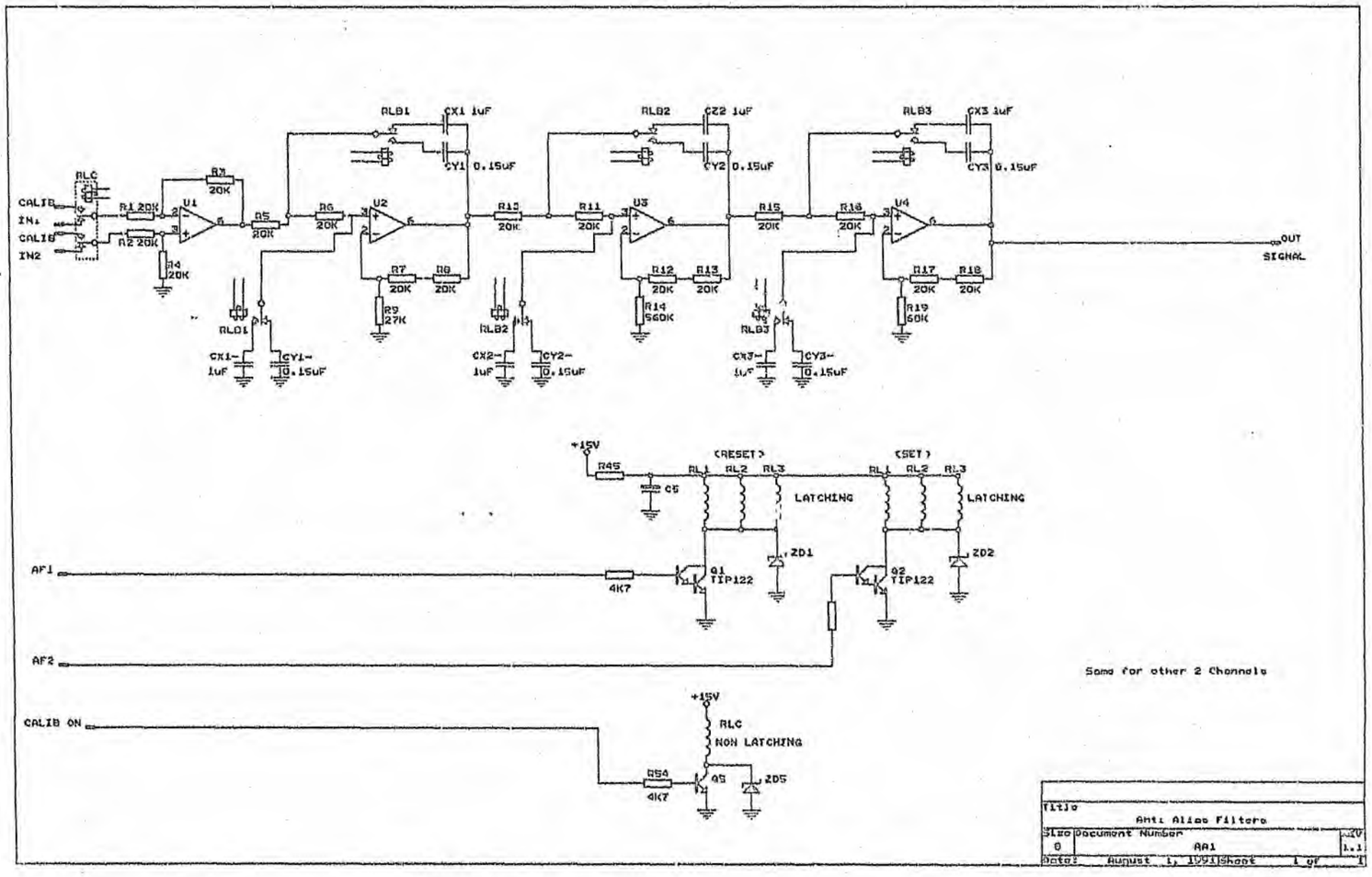


Figure 2.2: Circuit diagram of the anti-alias filter (one channel)

The filter can select 2 cut frequencies by switching capacitors. This selection is done by the CONTROL microprocessor board dependent on the sampling rate that has been selected. The two sampling rate options are 32 Hz and 256 Hz which correspond to -3 dB filter cuts of 8 Hz and 53 Hz respectively. Latching relays are used for the capacitor selection to ensure low power consumption. An input calibration facility is included, so that calibration of the electronic system can be ensured.

2.2 Gain Ranging boards

The gain ranging principle has already been discussed in (1.3.2). The circuit diagram for the gain ranging boards is shown in figure(2.3). The full wave rectifying circuit is formed by $U5_a$ and $U5_b$. U6, U7 and U8 are dual op-amps which form the 6 comparators. The component values for the comparators must be selected so as to optimise the gain for each step. The A/D card has a $\pm$ 5 V maximum input, so the comparators were designed to switch at $\pm$ 4 V. The circuit of the comparator is shown in figure(2.4) The requirements of high speed, low power and low offset required a careful selection for the optimum op-amp. For switching stability hysteresis was included in the feedback loop. It can be shown that for the circuit in figure(2.4) (c.f. Appendix A)

$$V_{on} = \frac{V_{ref}.R_g.R_f + V_0^+.R_g.R_s}{R_s.R_f + R_g.R_f + R_g.R_s} \tag{2.1}$$

$$V_{off} = \frac{V_{ref}.R_g.R_f - V_0^-.R_g.R_s}{R_s.R_f + R_g.R_f + R_g.R_s} \tag{2.2}$$

where V_0^+ is the voltage when V_0 is "high"
V_0^- is the voltage when V_0 is "low"

The values for R_g, R_f and R_s are then chosen for V_{on} and V_{off} of 4 V and 3.9 V respectively. Figure(2.5) shows a table of the values of V_{on} and V_{off} chosen for the various gain steps. The resistor values were calculated using

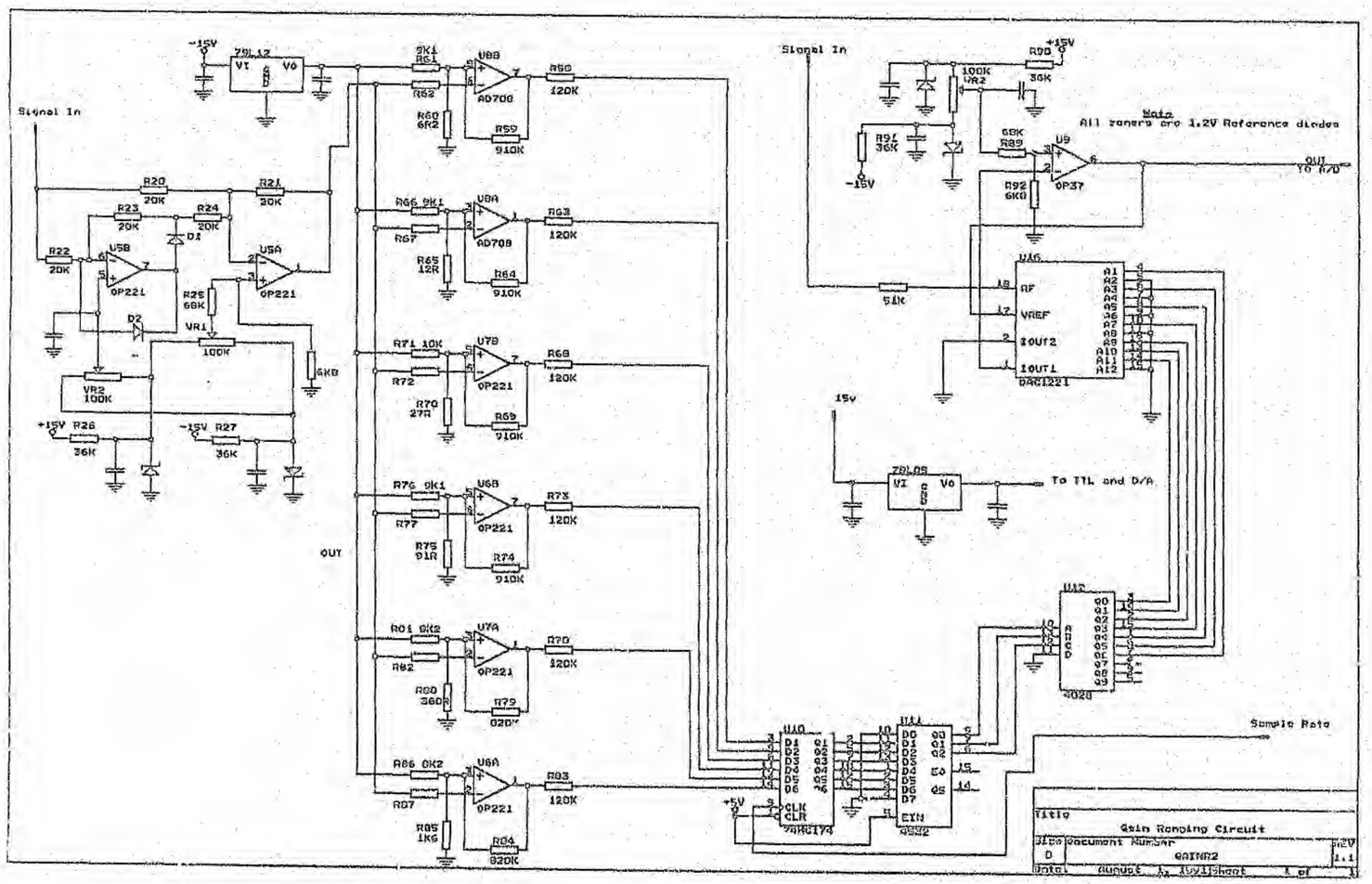


Figure 2.3: The Gain ranging board.

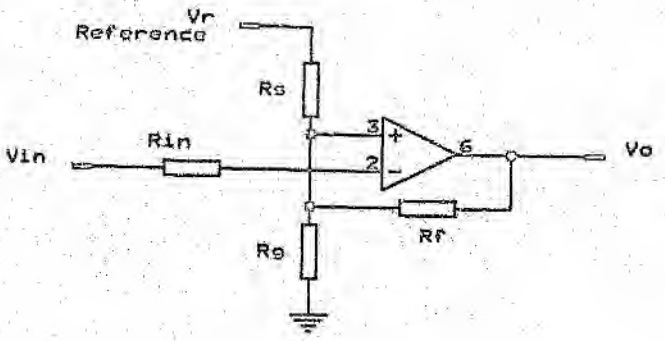


Figure 2.4: The comparator circuit.

4532 digital output	*Factor*	*Gain (dB)*	V_{on} *(mV)*	V_{off} *(mV)*
000	512×	+54		
001	256×	+48	7,8	7,6
010	128×	+42	15,6	15,2
011	32×	+30	31,3	30,4
100	8×	+18	125	122
101	2×	+6	500	488
110	0,5×	-6	2000	1950

Figure 2.5: Table of comparator switching points.

formulas (2.1) and (2.2) on the basis of the voltages V_{on} and V_{off} given in the table.

The outputs of these comparators are latched by U_{10} at the sampling rate. The Analogue to digital converter (A/D) conversion process is delayed by 6 μS from the time the sampling rate signal goes high, until the follow and hold operation is performed on the input data. This allows the decoding by U_{11}, U_{12} and U_{16} to select the gains and allows enough time for the output to settle down. The gain select steps are shown in figure(2.5). A digital to analogue converter (DAC1221) 12 bit ladder network is used in the feedback loop of an op-amp to vary the gain. The gain can be varied in binary steps from 1 - 4095. A prime consideration is that the op-amp's output must have settled down before the follow and hold is done. This necessitated a high slew rate for U_9 (15V/μS).

The decoding of the six comparator outputs (via the latch) is achieved by using a 4532 priority encoder, which gives a 3 bit binary output, depending on the state of the 7 input lines. In this way the output is as shown in figure(2.4). These 3 bits are transferred to the A/D board as the most significant bits (MSB) of the 16-bit word. (3 bits gain + 13 bits from the A/D). The gain bits are also used by the 4028, BCD to 1 of 10 decoder, which selects the switches to the ladder network achieving the desired gain steps.

2.3 Analogue to Digital Converter

The analogue to digital converter (A/D) board consists of a 3 channel multiplexer with input follow and hold gates, and a single ADC1225 A/D chip carrying out the conversion. Figure(2.6) shows the circuit diagram of the A/D, and figure(2.7) is the circuit diagram for the follow and hold circuitry. The DG309 is a low power high speed gate (200 nS) which exhibits low dynamic impedance (100 Ω). To start sampling the start/stop line is toggled by the control card and a latched $U1_c$ enables sampling. The follow and

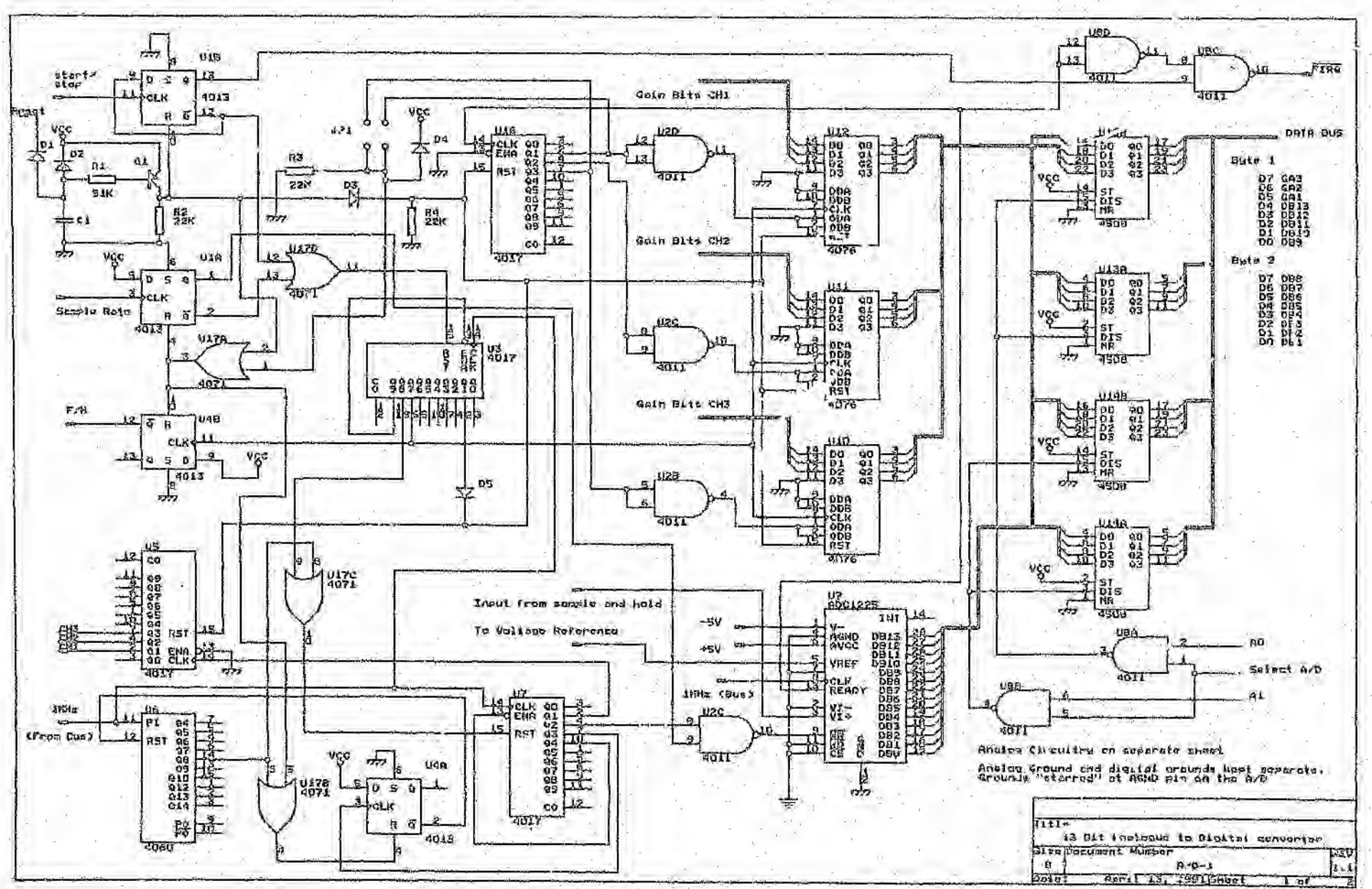


Figure 2.6: The Analogue to digital converter (A/D) board

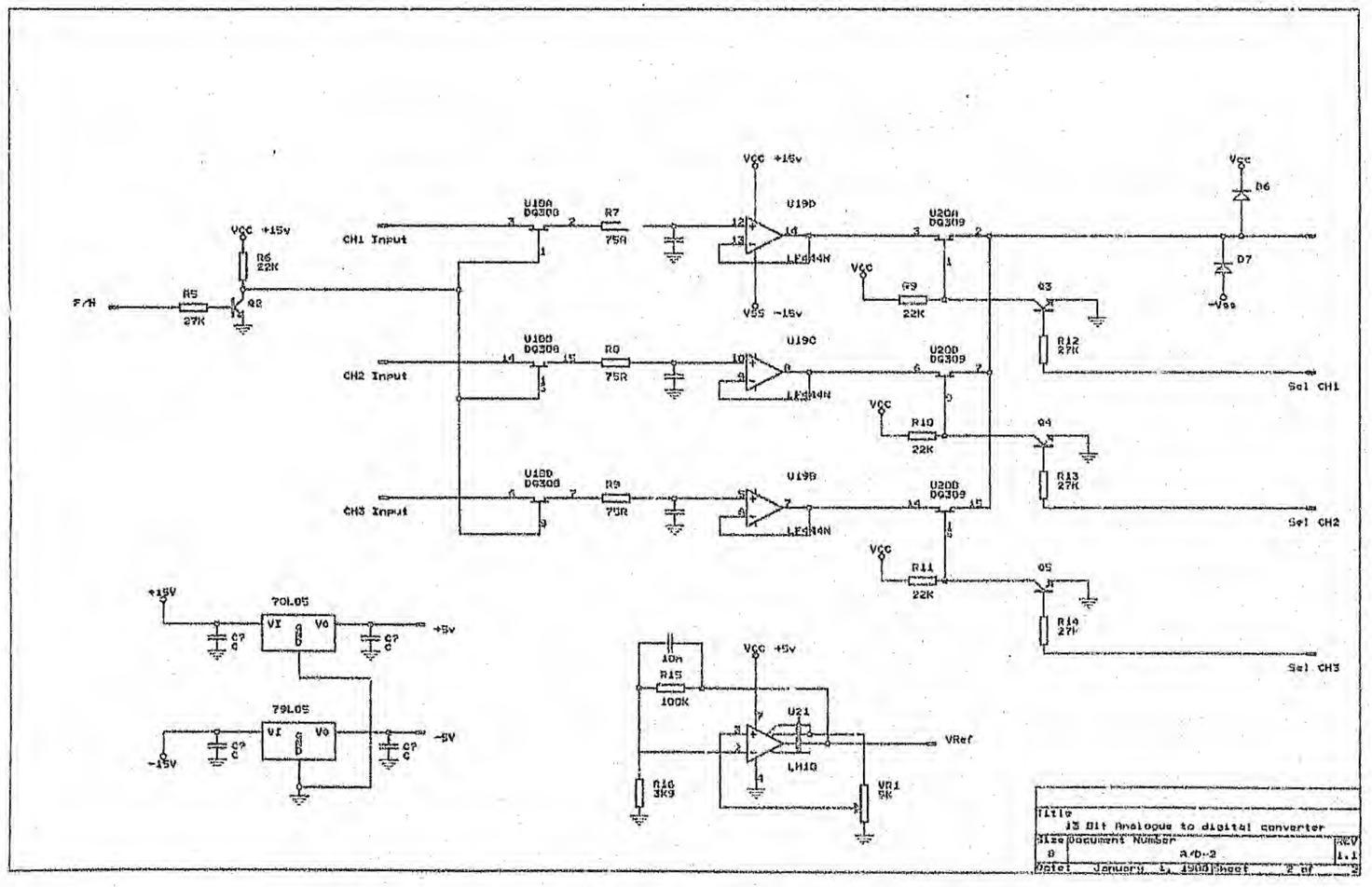


Figure 2.7: Follow and hold circuitry

hold operation is enabled 6 μS after the sampling request to allow time for the gain ranging circuitry to settle down. The first of the channels is then enabled (via a DG309) and read by the A/D (U9). The ready signal from the A/D is used to indicate to the DAC board that the data is ready, and to clock U17 so as to select either U10, U11, or U12 and the correct gain bits are then available at the inputs of the 4508 buffer $U13_b$. The 13 data lines from the A/D are latched by the 4508 buffers U13, and U14. These data together with the gain bits make up the 16bit word. The outputs of the buffers are available on the microprocessor data bus, and the DAC board can select the "high byte" (U13) or the "low byte" (U14). The selection is done via $U8_a$ and $U8_b$ and the select line (sel_AD) from the DAC board. The DAC board has 128μS to read the data for channel 1 (conversion time of A/D) before the data for the next channel becomes available. This sequence is repeated for the three channels (if jumper J1 is set for 3 channels) and then the multiplexing circuitry is reset for the next sample. The theoretical maximum sampling rate for three channels is then $\frac{1}{3\times 128\mu S} = 2600$Hz (Which translates to 2048 Hz in the described system).

Careful consideration must be paid to the slew rate for the sample and hold gates. In order to swamp the dynamic impedance of the gate and provide the driving op-amp with a resistive load the total series resistance requirement was taken as 1KΩ. assuming constant current -

$$V = \frac{IKt}{C}$$

$$t = \frac{VC}{I}$$

$$\text{for } C = 1\text{nF} \; ; \; V = 5\text{V} \; ; \; I = 5\text{mA}$$

$$t = \frac{1 \times 10^{-9} \times 5}{5 \times 10^{-3}} = 10^{-6} S = 1\mu S$$

This satisfies the sampling rate requirements.

2.4 The Data Acquisition Card

The core of the card is the Hitachi CMOS 6309 microprocessor (equivalent to Motorola 6809). The 63B09P version was used which has an internal clock generator. There is no need for external clock circuitry and a crystal is simply connected to the crystal inputs on the microprocessor. The circuit is shown in figure(2.8). The 63B09P was used in all three boards as the main processor. The layout includes space for an 8 Kbyte 27C64 EPROM. However only a 2 Kbyte EPROM was necessary for the prototype program. All the software for the microprocessors was written in 6809 Assembler code. (Further discussion of the software is dealt with in chapter 5). A 6264 8 Kbyte RAM is used for general program use, such as stacks, buffers and general storage requirements. Three 32 KByte RAM's are used as a temporary storage for the data of each seismic channel. The use of more than 64 Kbyte's of memory necessitated "paging" these three RAM chips. They are all accessed at the same memory location but are "paged" by selecting a specific chip via an I/O (input/output) port. The I/O port, a CMOS 6321 (equivalent to the Motorola 6821) is used both for paging memory and for data transfer. It is a two port device, one port (8 bits) being used for data transfer, three of the bits on the second port being used for memory paging. This is the essence of the Data acquisition card. The memory map for the DAC board is shown in figure(2.9). The select line (select_ad) is taken to the A/D board together with the two low order address lines A0 and A1. Address 2001h [1] reads the high byte, while 2002h reads the low byte. Address 2004h is used to reset the FIRQ latch, once the interrupt has been serviced. The two bytes read from each A/D are stored in the 32 Kbyte RAM buffers. If a trigger is generated by either internal or external triggering the data will be transferred via the 6321 I/O port. The data transfer is controlled by the I/O port handshake lines CA1 and CA2. These can be programmed to generate an interrupt (IRQ - the interrupt request line being used) when data is ready to be transferred or received.

[1] All hexadecimal numbers will be written with an h following the number

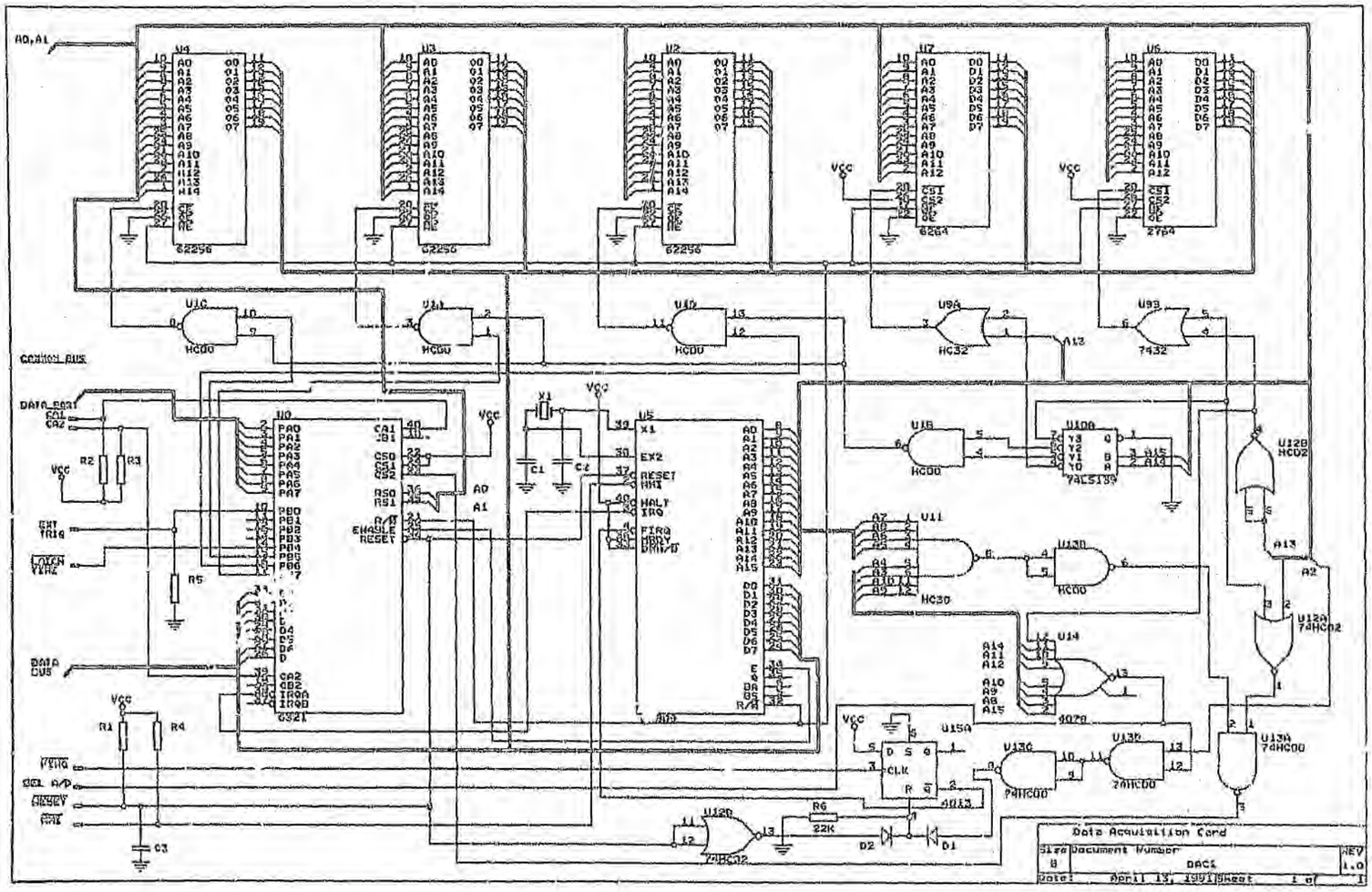


Figure 2.8: Data acquisition card

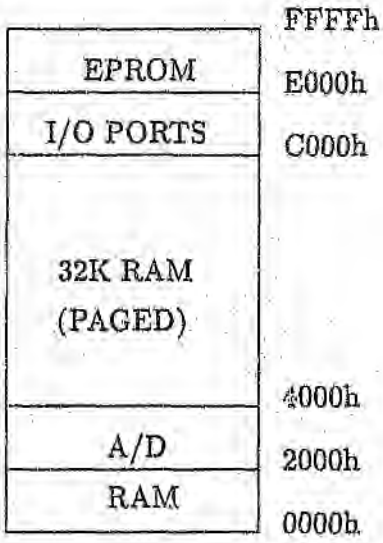


Figure 2.9: The memory map for the DAC board

2.5 Mass Storage and Data Transfer

This consists of a memory controller board utilising a 6309 microprocessor, and a number of low power, static random access memory (RAM) boards. (Up to maximum of 16). The memory controller board is shown in figure(2.10), the RAM board in figure (2.11). There is an addressing facility for 16 RAM boards, each with 384 Kbyte of storage which implies a total storage capability of 6 Mbyte or 3 Mwords. The RAM boards use six 32 Kbyte 62256 low power static RAM chips per board. These chips have very low standby power specifications, typically 10 μW. All 32K RAM chips occupy the same memory space (as shown in figure(2.12)) but are selected individually by the microprocessor. The microprocessor board is very simple. It consists of the 6309, two 6321 I/O ports, place for 8 Kbyte EPROM and 8 Kbyte RAM, and associated decoding circuitry. See figure(2.12) the memory map for the MEMORY board. Four bits of the one 6321 I/O port are input to a 16 line multiplexer to select which of the 16 RAM boards is to be written to. The multiplexer is a bidirectional device, and the I/O port can be set up to read the multiplexed line and thus read how many (and which) memory boards

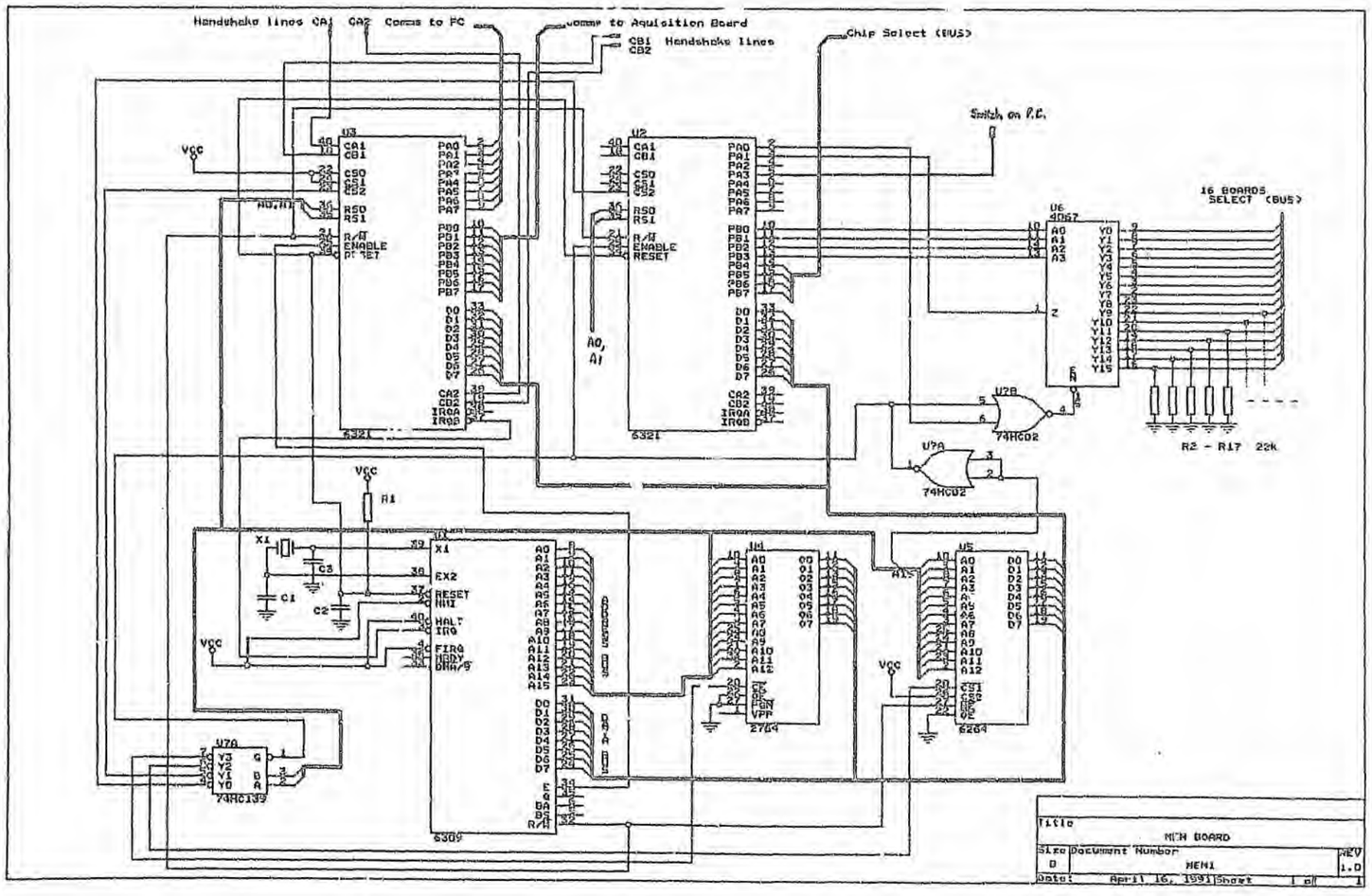


Figure 2.10: Memory controller card

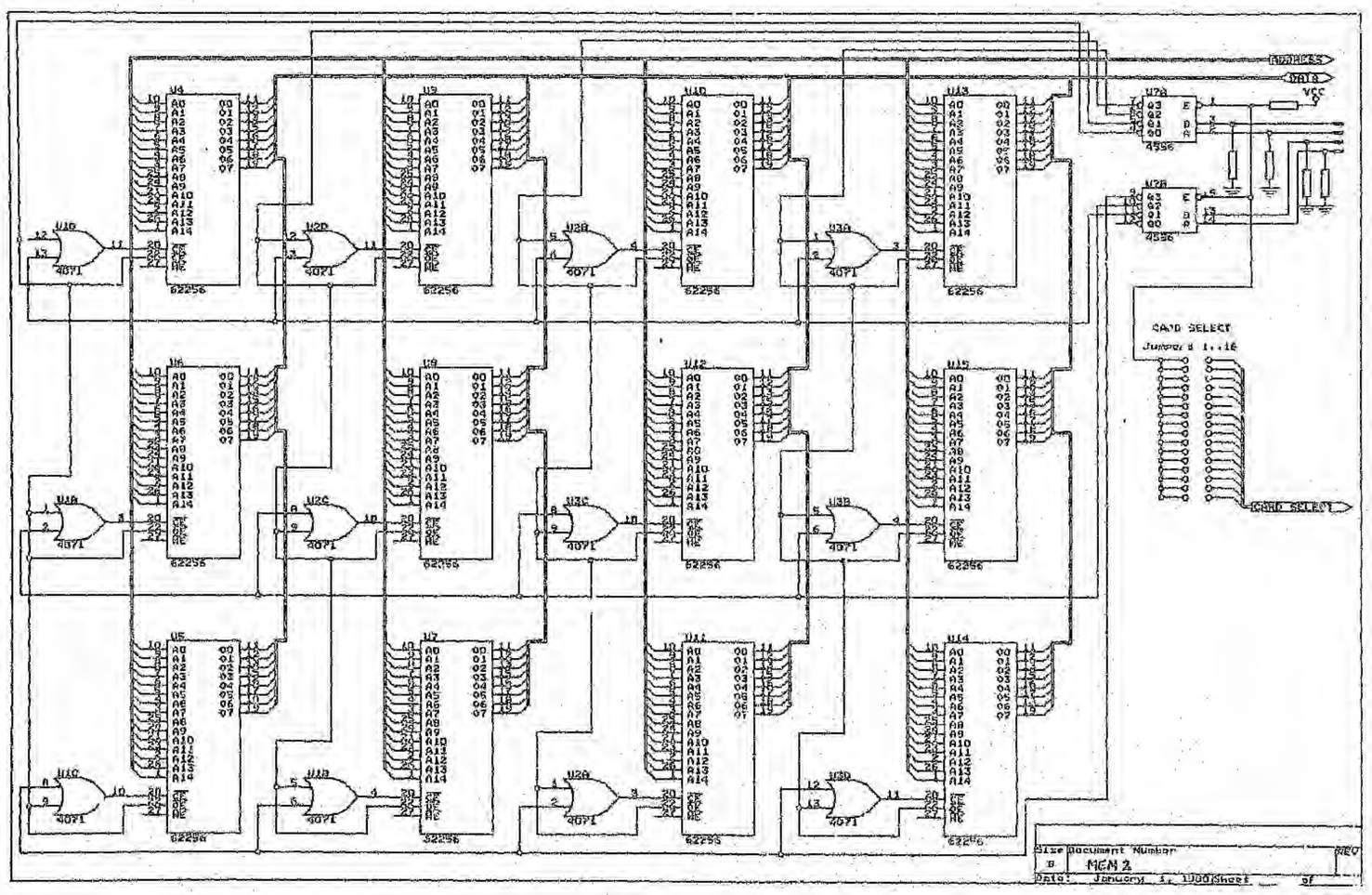


Figure 2.11: Static RAM board

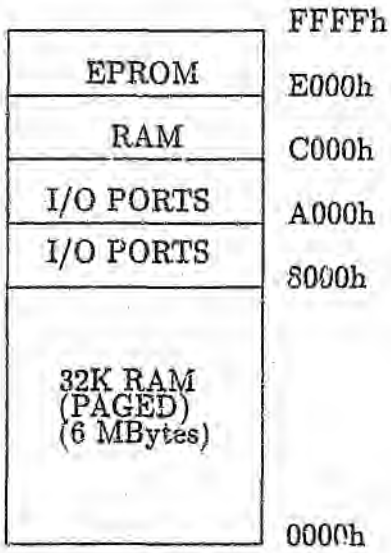


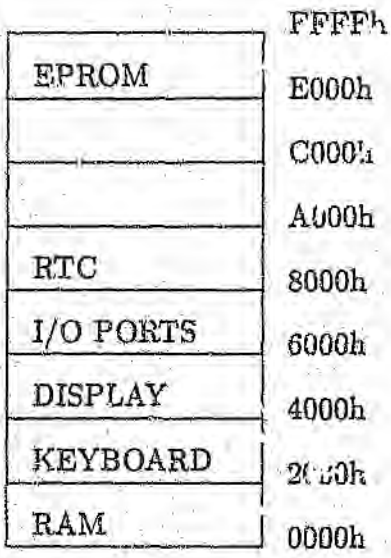
Figure 2.12: The memory map for the MEMORY board

are installed.The other 4 bits are used to select which 32 Kbyte RAM chip on the memory board will be written to. The second 6321 I/O port is used for data transfer. The one port is used to receive data from the data acquisition card, the other is used as a data output port that can be configured for data transfer to a peripheral such as a tape streamer or PC.

2.6 Control and Timing Card

This card again uses a 6309 microprocessor which controls the following :
- Real Time Clock and Sample Rate
- Anti alias filters
- Data transfer
- Keyboard and display

The configuration is similar to the DAC and MEMORY cards and the memory map is shown in figure(2.13). The circuit diagram for the CONTROL



3) memory map for the CONTROL board

board is sh·v. '4).

2.6.1 Real time clock

The real time clock (RTC) used is a Motorola MC146818 low power CMOS RTC chip. The reference frequency is derived from a parallel resonant 4,194304 MHz crystal. The time and date (down to seconds) is read from the RTC in binary coded decimal (BCD) format. All timing updates such as end of month recognition and leap year compensation are taken care of by the RTC. The square wave output from the device is used to control the sampling rate, and can be selected in binary steps from 2 - 32768 Hz. The frequencies used in this case were 32Hz and 256Hz. The RTC provides a 1.194304 MHz signal which is used for the A/D converter conversion clock signal.

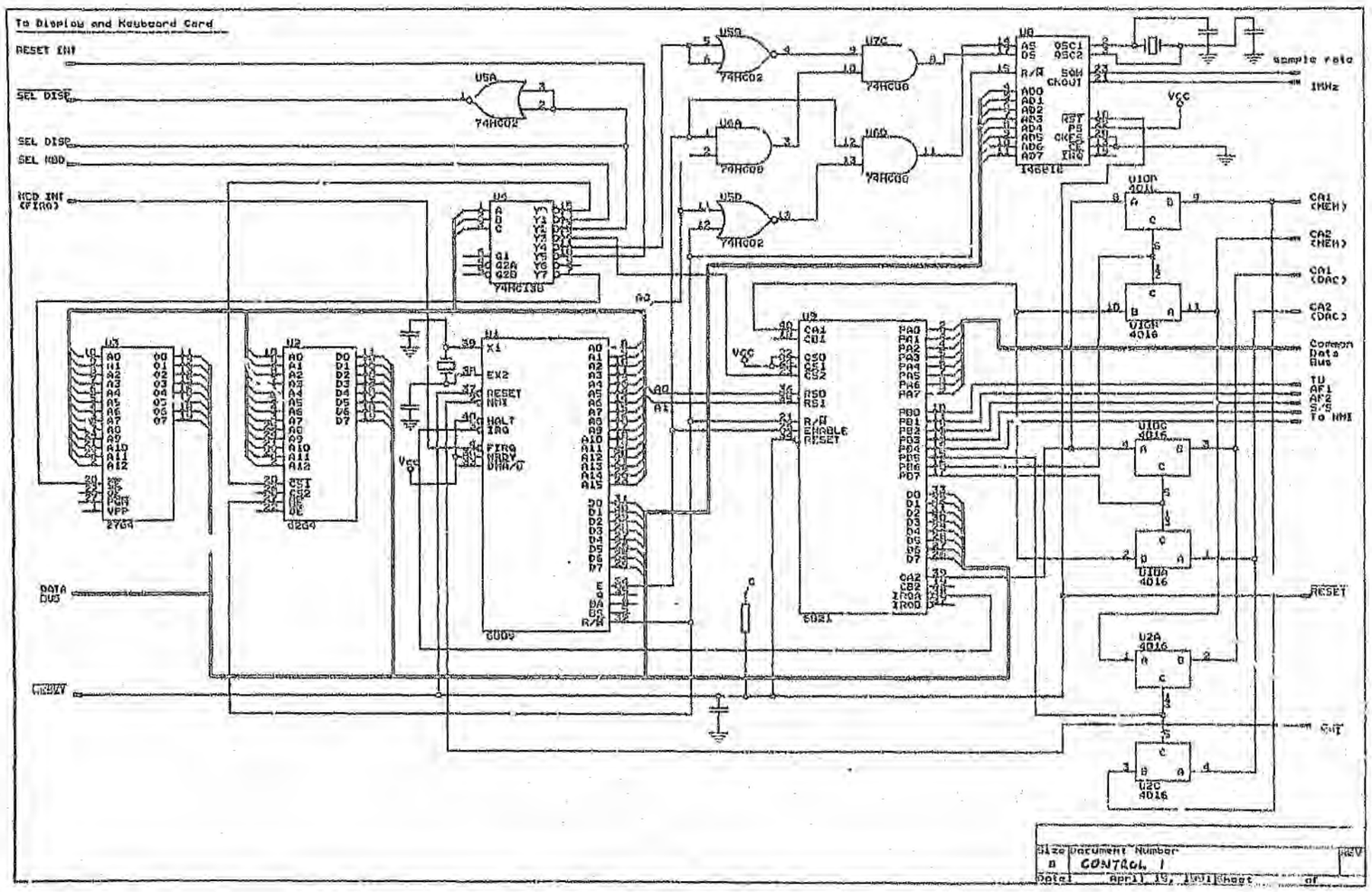


Figure 2.14: The circuit diagram for the CONTROL card

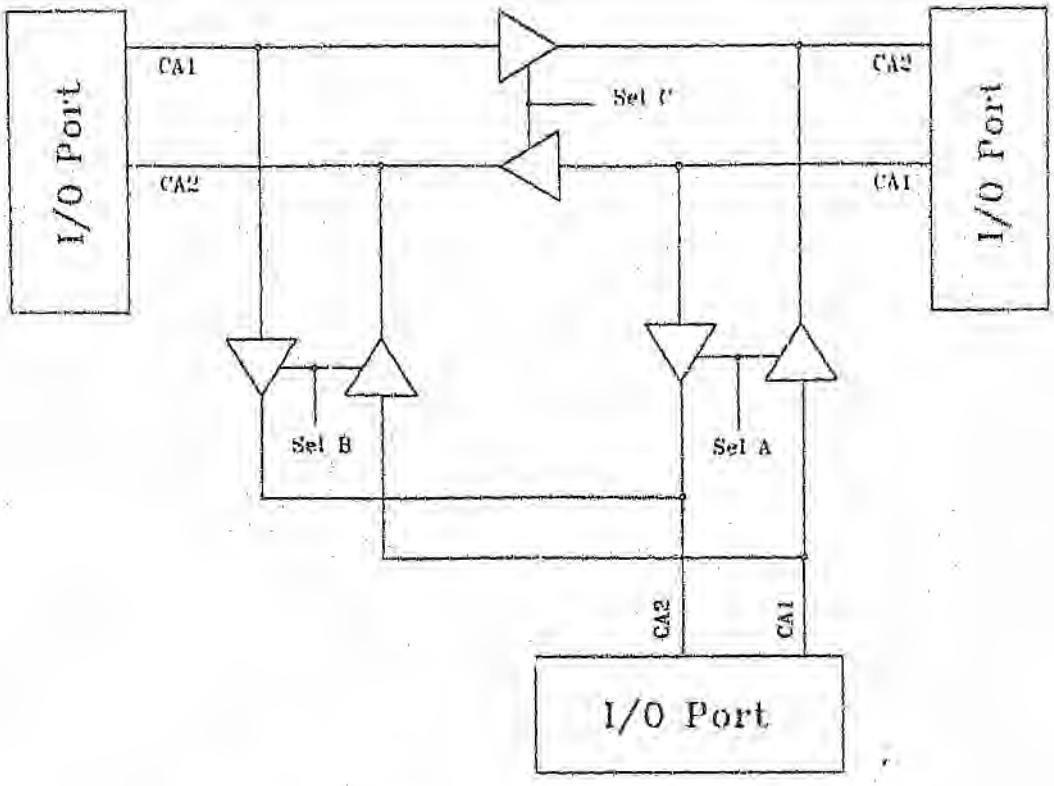


Figure 2.15: Multiple bus handshake control

2.6.2 Anti-alias filter selection

The anti-alias filters must be selected depending on the sampling rate chosen. The two cut frequencies available are 8Hz and 53Hz. The I/O port is programmed to control 2 lines which enable the latchable relays on the filter card (section 2.1). The "latch on" output enables the 53Hz filter and the "latch off" output enables the 8Hz filter. These outputs should be held high by the software for 10mS to ensure the relays are latched.

2.6.3 Data transfer

The data transfer between the boards was achieved by using a common 8 bit bus between the 3 microprocessor cards. The handshake lines were then selected using 4016 CMOS switches, for the desired transfer. The board not involved in the transfer would ensure its output was tri-state. See figure(2.15). Data transfer can be achieved in any direction between any of the 3 boards by using different combinations of the CA1 and CA2 handshake

lines of the 6321 I/O ports.

2.6.4 Keyboard and display

The circuit for the keyboard and display is shown in figure(2.16). The DMC liquid crystal display used has an internal character generator which means that normal ASCII can be sent directly to the display. The 74HC244 buffer is used to interface the display with the microprocessor data bus.

The keyboard consists of a matrix keypad which is continually scanned. The 4093 gate U1a forms an oscillator (frequency = 100KHz) which increments the 4040 BCD counter U2. The BCD outputs are taken to U3 and U4 (8 line multiplexers). The multiplexed line is used as an input on U4 and is held high. If a key is held down, the input of U4 (high) is clocked through the multiplexed line of U3. This in turn latches the 74HC374, a tri-state latch, which is connected to the 4040 binary outputs. In this way each key on the keypad has a unique code latched by 74HC374 when a key is pressed. The line used to latch the data is also used to generate a fast interrupt request (FIRQ) to the microprocessor to indicate that a key is pressed and its code can be read from the latch.

2.7 Radio receiver and Time Sync Card

In order to synchronise the time on the separate stations, a standard FM broadcast receiver covering the 88-108 MHz band was used. A time "pip" decoder was built which decodes the time "pips" given every hour on the regional stations. The original idea was obtained from the Magnetic Observatory CSIR.

Using this method of time synchronisation means that the stability of the clock is not that critical, as up to 6 or 7 updates are received per day. This synchronisation method was tried with the 8-bit system and has proved to

Fig 6: Keyboard and Display

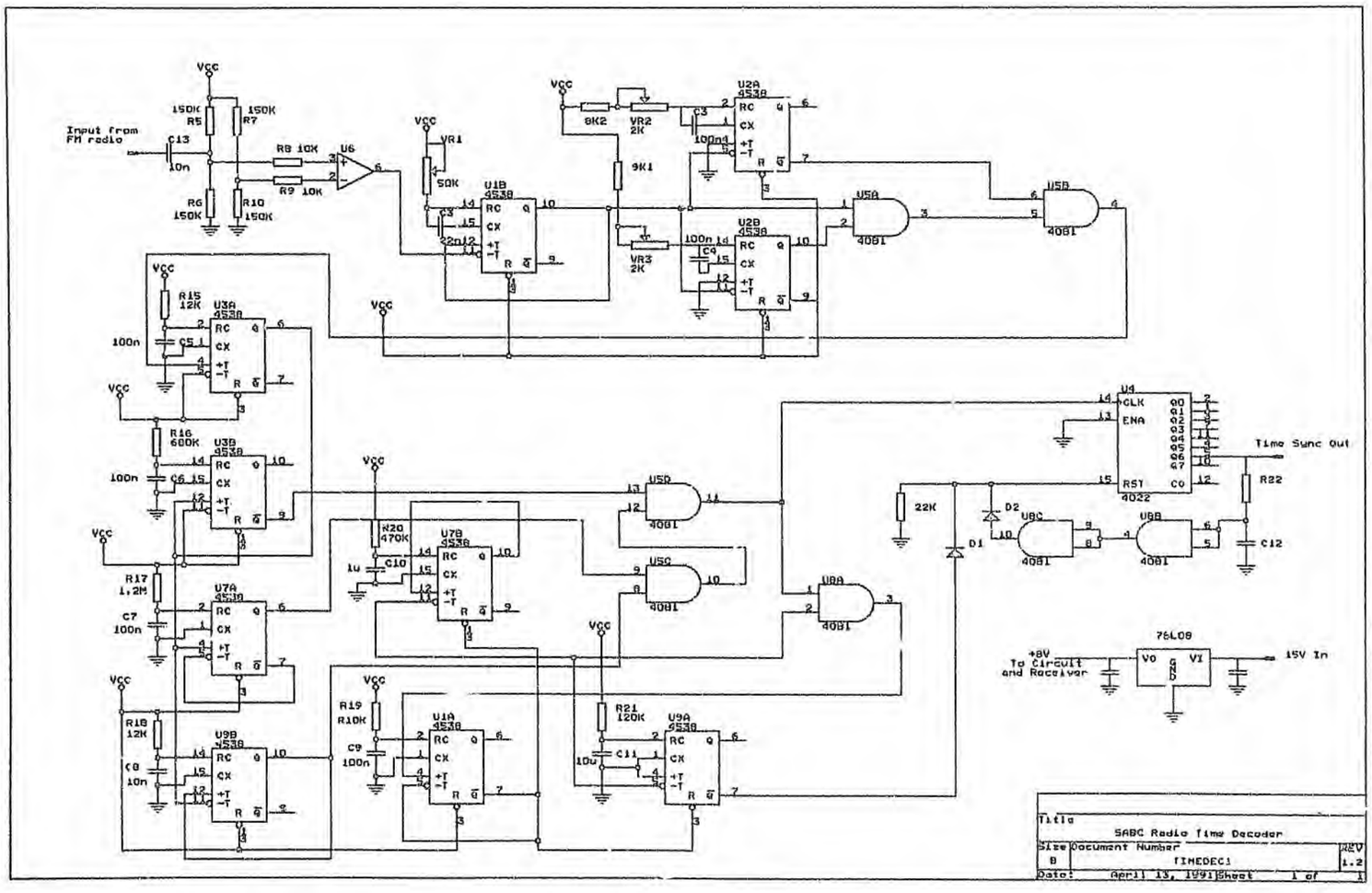


Figure 2.17: Time Sync Card

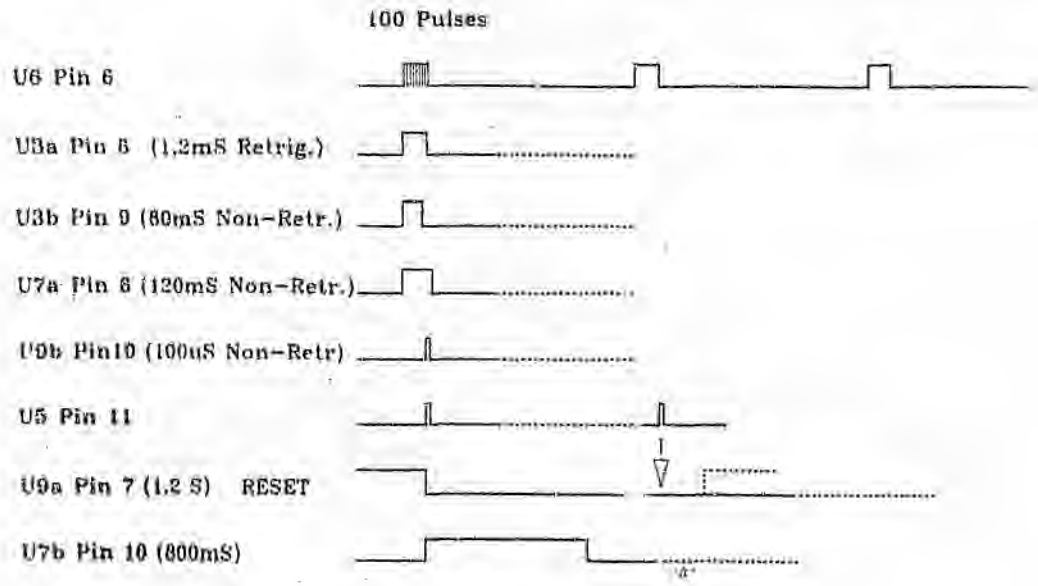


Figure 2.18: Timing Diagram

be very successful. The countrywide coverage by these stations is very good, and it was only necessary to use a high gain antenna at one of the sites, all the other radios operating perfectly well with just a length of wire. The circuit diagram is shown in figure(2.17). The audio output from the FM radio is passed to a comparator U6 where it is effectively clipped. A band pass "digital filter" is formed by monostables $U1_b$, $U2_a$, $U2_b$ and gates $U5_a$ and $U5_b$. Only signals close to 1KHz will be passed to $U3_a$. The rest of the circuit consists of cascaded monostables which effectively open various windows, which if the 1 KHz "pips" are present, allow the "pip" signals to clock through the 4022 counter and give a synchronisation pulse. The timing is shown in figure(2.18) the timing diagram. The output is a pulse 100 mS after the correct ZUO time. This 100 mS error is however consistent with all the stations utilising the time update receiver.

The time synchronisation circuit works satisfactorily. However if it had to be rebuilt monostables would not be used.

2.8 Input/Output Port for PC

An Input/Output (I/O) port was built for the PC using an 8255 I/O port. The circuit diagram is shown in figure(2.19). The port is used to transfer data to the PC, 8 bits being used as input lines, and 2 lines being used for data handshake. The port address can be selected by jumpers J_1 - J_6 for an unused address on the PC. The jumper selections are shown in appendix (C)

2.9 Power supply

The system runs off a 12 V battery supply which is recharged from solar panels. A power supply was constructed using PCB mount DC - DC converters (the NME05/12 series from Newport Components). These simplify the design of the power supply considerably, giving a $\pm$15 V and +5 V supplies. The DC-DC converters are not particularly efficient however (only 75%) The circuit diagram is shown in figure(2.20).

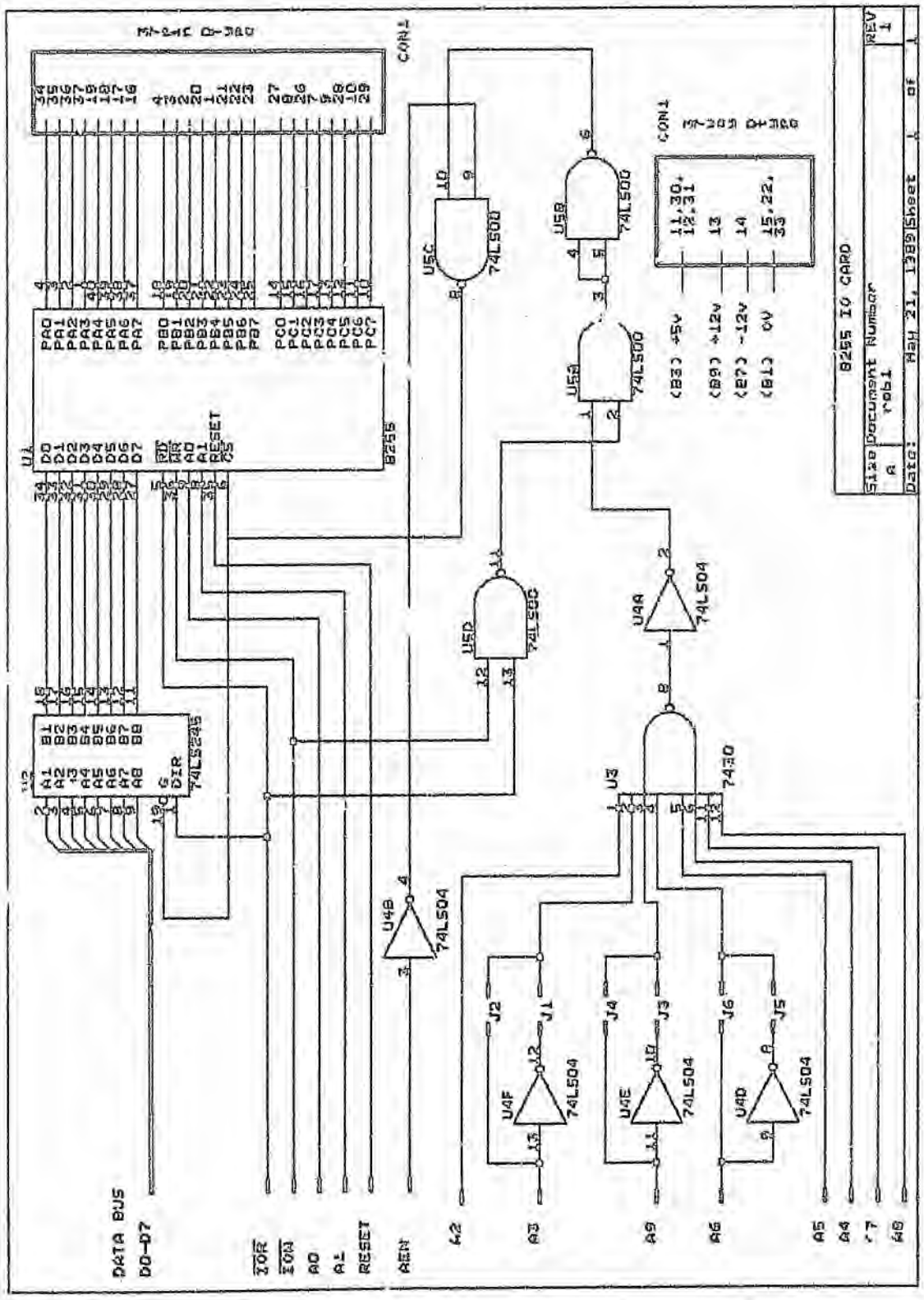


Figure 2.19: Circuit diagram of 8255 I/O Port

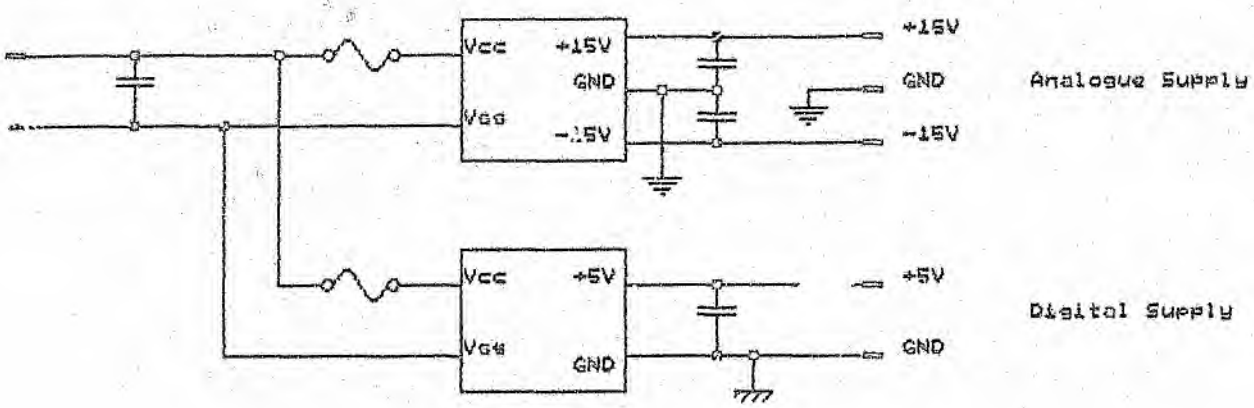


Figure 2.20: Circuit diagram of power supply

Chapter 3

Software

As explained in chapter(2) on the hardware, three microprocessor boards were used (one dedicated to transfer of data to peripherals, one for timing and user control and one for data sampling and processing).

Flow charts will not be used to describe the program operation but rather a PDL(program description language) will be used to convey the system operation.

Each of the programs will be discussed in turn. A PDL description will be given for each, and a brief outline of the operation. Addendum(1) contains the program details. The program and PDL listings are for a 3 channel system no triggering, with data downloading to a PC. Only one memory board is used (384 Kbytes of buffer storage).

To make the assembler program more readable certain conventions were used in programming. Variables are given small letters, constants are in capitals, while all procedures (or subroutines) have their first letter as a capital. Also a note about the Program Description Language (PDL). PDL is a pascal like description of the software, with general comments in order to describe the operation of the software. It is a 'top down' description of the program. General comments are denoted by *'s, Otherwise names refer to procedures (subroutines) within the assembler program. All software has been written

in the form of subroutines that can easily be incorporated into subsequent programs. Operations such as data transfer, RTC functions, and writing to the display can all be achieved by calling these subroutines. For example, for data transfer the data to be transferred is simply loaded into the accumulator and the subroutine Xfer_AReg called

```
LDA data
JSR Xfer_AReg
```

The specific explanation for each procedure is given in the program listing. In this way it is hoped that future modifications will be greatly simplified.

3.1 Controller Program

The controller card performs the following functions :

- Real Time Clock (RTC)
- Keyboard
- Display
- Time Update
- Control of data transfer between boards
- Start/Stop of data sampling
- Selection of anti-alias filters

The PDL description of the controller software is given in figure(3.1) As already discussed in the hardware section, a common bus is used to transfer data between the three boards. The control card selects the direction of the handshake lines via CMOS buffers. A card not in use during a transfer, MUST be kept tri-state. Even on reset the I/O ports must be forced into tri-state as experience with the original 8-bit system has shown that the 6321 I/O ports to be very susceptible to destroying each other if the outputs are accidentally driven into each other. The program first selects MEM-CNTRL (ie. Memory to Control) and an error byte is read from the memory board. This byte is presently used to indicate to the control board that the memory card has reset correctly, but will in future be used to transfer data

```
BEGIN (Control Program)
    Initialise (Ports, RTC, Display, Keyboard)
    SetTransfer (MEM - CNTRL)
    ReadMEM (error-byte)
    If error-byte = error Then
        * Reinitialise *
    * InputData (station, date, time, sampling-rate)*
    SetTransfer (CNTRL - DAC)
    * WriteDAC (sampling-rate) *
    * Set-AA-Filter (sampling-rate) *
    SetTransfer (DAC - MEM)
    * Wait for KeyPressed *
    Repeat * forever *
        * Output the time to display *
        * Check I/O port for time update *
        If time-update Then
            Service-Time
    End
END.


    Procedure (Service-Time)
        Begin
            * Reset (second,minutes) *
            * Toggle I/O line to DAC NMI *
        End
```

Figure 3.1: PDL Description of Controller Software

```
Procedure (Keyboard Interrupt) - FIRQ
    Begin
        * Read Character from keyboard latch *
        * Reset Interrupt *
    End
```

```
Procedure (Latch/Tranfer Time) - NMI
    Begin
        If first-time Then
        Begin
            GetTime(hours,mins,secs)
            first-time := false
        End
        Else
        Begin
            SetTransfer (CNTRL - MEM)
            TransferTime (hours,mins,secs)
            SetTransfer (DAC - MEM)
            first-time := true
        End
    End
```

Figure 3.2: Interrupt routines : Keyboard (FIRQ) and Latch/Transfer Time (NMI)

such as the amount of RAM available, and to indicate whether all the RAM is operational. The program then prompts for user input (such as station number, sampling rate, time and date). After selecting sampling rate the corresponding anti-alias filter is selected and the selected sampling rate is transferred to the DAC board. The program then starts sampling, and data transfer from the data acquisition card to the memory card is enabled (DAC-MEMORY). When the DAC card recognises a trigger (buffer full, external or internal trigger) it initialises a non-maskable (NMI) interrupt sequence on the CONTROL board. The NMI interrupt routine is shown in figure(3.2). The first NMI latches the time (subroutine latch time) and the time and date registers are stored. Once the DAC board has transferred data, a second NMI sequence is initiated by the DAC board and the data transfer selection is set up as CONTROL-MEMORY. The time and date information is sent to the MEMORY board together with any other system information such as sampling rate and station number. Transfer DAC-MEMORY is again enabled for the next transfer from the DAC buffer.

The PDL listing for the keyboard interrupt (FIRQ) is shown in figure(3.2). When a key is pressed, the keyboard latches the key code, and generates a fast interrupt request (FIRQ). This calls the keyboard procedure, which temporarily stores the key code. A look up table is used to get the value of the key pressed. The display has an internal character generator and ASCII code can be written directly to the display. Various routines for advancing or decrementing the display are also available together with routines such as WRITESTRING and WRITELN. The operation of these procedures is explained in the program listings submitted as an addendum.

3.2 Data Acquisition Program

The DAC program initialises the stack and I/O ports then waits for the control board to transfer the sampling rate. The PDL listing is shown in figure(3.3). The sampling rate is needed to set up the correct value for the "dragon count". The dragon count is used to note on which sample, between

```
BEGIN (DAC Program)
    Initialise (I/O Ports)
    * Wait for comms witl. CNTRL board *
    * Read (sampling-rate) *
    Dragon := 0 *
    * Buffer := empty (count := 0) *
    Repeat
        If buffer = 28KByte Then
        Begin
            * buffer = empty *
            * Toggle line to CNTRL NMI - latch time *
            * Transfer(buffer) *
            * Toggle line to CNTRL NMI - transfer time *
        End
    End
END



    Procedure (Read Analog to digital Converter) FIRQ
        Begin
            * Read Channel1 (data) *
            * Store(data,buffer1(count)) *
            * Read Channel2 (data) *
            * Store(data,buffer2(count)) *
            * Read Channel3 (data) *
            * Store(data,buffer3(count)) *
            dragon := dragon + 1
            If dragon = sampling-rate Then
                dragon := 0
            count := count + 1
            * Reset Interrupt *
        End



    Procedure (Reset Dragon Count) NMI
        Begin
            dragon := 0
        End
```

Figure 3.3: PDL description of DAC program and interrupt services

two full seconds, a trigger occurs. The time transferred to the mass storage card then consists of the date, hours, minutes, seconds, and the dragon count - which is the number of samples from the previous whole second. Once sampling has been enabled, the A/D signals that data is ready by initialising a FIRQ interrupt request. The DAC program must read the A/D and store the data in one of three buffers (the data could have been stored consecutively, but logically it made more sense to use one buffer per channel). The program must read the data within 128μS of the FIRQ signal (the conversion time of the A/D) before the next channel is sampled by the A/D. After reading the three channels, the FIRQ procedure restores the paged memory channel that was operative when the interrupt occurred. This is to ensure that if the interrupt occurred during data transfer to the MEMORY board, that the data transfer continues with the correct memory channel selected.

The main program waits in a loop until either the buffers have stored 28 Kbytes of data, or a trigger is recognised. A line on the [illegible] port is used to generate a NMI on the control card to latch the time. Data is then transferred via the I/O port (comms port) to the memory card. The comms port is set to tri-state and the I/O port initiates a second NMI sequence on the control board which transfers time and date information.

3.3 Data manipulation

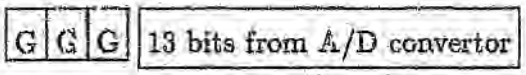
If triggering techniques are used, the data acquisition program will incorporate the algorithms discussed in the chapter on triggering. The data is however in a "compressed" form with the first 3 bits being gain bits. Data manipulation cannot be performed directly on this data and the data must first be transformed to some meaningful value. The transformed data would in effect be equivalent to a 22 bit value. It is impractical to use a variable of this size so some way of keeping the sensitivity, but using smaller numbers was needed. For the simple LTA , STA trigger algorithms all the computation can be achieved with add and shift operations. The 8-bit $\times$ 8-bit multiply feature of the 6309 microprocessor is rather slow, but would probably have

Factor	*Gain(dB)*	*logical operation*
512×	+54	clip
256×	+48	clip
128×	+42	shift 3 bits left
32×	+30	shift 2 bits left
8×	+18	shift 1 bit left
2×	+6	don't shift
0,5×	-6	shift one bit right

Figure 3.4: Table showing operations on the 13-bit gain ranged data.

to be used to implement the filters required for teleseismic triggering.

The gain ranged 16-bit data representation is as follows :



The three gain bits (G) form the three most significant bits of the 16-bit word. The 13 bits are in two's complement format. It was decided to compress the data to a 16-bit word. This means that the high gain data will be "clipped" (unless we are to lose resolution). The dominant frequency information has not been lost although harmonics will be introduced. This should not effect the trigger algorithms too badly, although if used with the teleseismic trigger algorithm, the high frequency harmonics caused by the clipping will interfere with the algorithm. (If a very large event is recorded, it may not trigger as high frequencies are present due to the clipping action - see section(4)). The 13 bits from the A/D are in 2's complement format. In order to scale the data, some form of division is necessary. Speed considerations meant that only shift operations could be employed (divide by 2) but the presence of the gain bit causes a problem. It is either necessary to move it to the most significant bit, or change the 13-bit data to normal binary format. It was decided to change to binary format, by complementing the gain bit. This could also be done in hardware to save computation time The operations performed on the data are then as shown in figure(3.4). So one bit of resolution is lost and

high gain signals are clipped.

3.3.1 Rectification, LTA and STA

In order to rectify the signal the DC offset of the signal must be calculated. It is assumed that if the transformed data is clipped that the clipping will be equal on both sides of the DC level and that no major errors will result. The DC offset is calculated by taking a very long term average (16384 samples) which is a factor of 2. This is done to allow division by simply shifting the data to the right. A 4 Byte variable is needed for this value. The average is calculated by dividing by 16384, or 14 shifts to the right ($16384 = 2^{14}$). The rectification is then done by simply subtracting the average from the input value and taking the absolute value. For a 64 sample STA a 3 byte variable and for a 1024 sample LTA a 4 byte variable is required. Again the divisions are simply done with shifts (6 shifts right for STA, 10 shifts right for LTA). The averages are also in 16 bit representation.

3.3.2 Filters

To implement the low-pass and band-pass filters needed for teleseismic triggering (see section(4)) we need the hardware multiply and divide feature of the 6309. Only an 8-bit $\times$ 8-bit multiply is available so some further manipulation is required. The answer variable must be 4 bytes long to accommodate a 16-bit $\times$ 16-bit multiply.

3.4 Memory Program

The operation of the memory program is very simple (PDL description given in figure(3.5)) and operates essentially as a large buffer memory. One set of I/O ports is used for data transfer (one port for transfer to PC, one for reading data from DAC and MEMORY) the other I/O chip being used for paging memory. The incoming data is read under interrupt (IRQ) into the RAM buffers (up to a maximum of 16 boards - 6 Mbytes). The main program is a loop, which waits until the buffer is nearly full, then transfers the data to the peripheral device (in this case a PC).

3.5 Data capture on the PC

The data is transferred from the memory board via the 8255 I/O port to a PC equipped with a tape streamer. The programs on the PC are all written in Turbo Pascal 5.5. Reading in large arrays of data in turbo Pascal is somewhat of a problem, so the arrays are declared as absolute arrays (ie. use a fixed memory location) so the programs should not be run with any terminate stay resident programs which may possibly already be using that memory space. The data is read using the program RD_SEIS.EXE (See Addendum for program listing). The data is stored in binary format on the hard disk.

3.6 Data format

One dump from the DAC board to the MEMORY board consists of (3× 28 Kbytes) + 16 bytes (the header). This is referred to as a block of data (84 Kbytes + 16 bytes long). As previously the old system used 360 Kbyte floppy disks for storage of data, the file size was kept as 360 Kbyte. The data is initially stored on a 40 Mbyte hard drive, and can then be transferred to a tape streamer. Pascal has a blockwrite facility which allows rapid transfer of data in block form to disk devices. The quickest way to transfer data is to

```
BEGIN (MEM Program)
    * Initialise (I/O Ports) *
    Transfer CNTRL (error-byte)
    Repeat
        If buffer = full Then
        Begin
            * Transfer buffer to PC *
            * buffer := empty *
        End
    End
END


    Procedure (Read data from DAC) IRQ
        Begin
            * Read seismic data *
            * Read (station,date,time,sampling rate) *
            * Store data to Buffer *
            * Increment buffer counter *
        End
```

Figure 3.5: PDL description of MEMORY software

open a file, and transfer as many blocks as possible using BlockWrite before closing the file again. 4 Blocks of data is 336 Kbytes long, so a file consisted of 4 blocks. The data is input in the form

The header consists of 16 bytes some of which are not in use at present. The 10 Bytes in use contain the following system information:

- "dragon count" : 2 bytes
- station number : 1 byte
- year : 1 byte, Binary coded decimal (BCD)
- month : 1 byte (BCD)
- day : 1 byte (BCD)
- hour : 1 byte (BCD)
- minutes : 1 byte (BCD)
- seconds : 1 byte (BCD)
- sample rate : 1 byte

The program RD_SEIS.EXE is used to read the data from the acquisition system. Only one memory board was used with the prototype, but at least four would be needed for a field system. The data is initially read into three 28 Kbyte arrays before being dumped to disk. One block is therefore 86032 bytes long, a file being 4 × this. Using 4 memory cards gives 1536 KBytes of storage before a dump is necessary. This means 17 blocks can be stored before a transfer to the PC is necessary. To keep this in blocks of 4, rather use 16 blocks (or 4 files). The transfer time MEMORY to PC is approximately 8 seconds/block, and the dump time to disk is approximately 2 seconds per block. This gives 160 seconds to transfer the data plus the turn on time for the PC (30 seconds). The total transfer will take about 190 seconds. The file name is the month, day, hour and minute of the first block of data in the file.

Once the hard drive is almost full, the files can be copied to a tape streamer and the files deleted from the hard drive.

The program CONVERT.EXE is used to read the raw data files on the PC, uncompresses the data, and converts the data to an ASCII file.

Chapter 4

Triggering

One of the major problems with seismic field stations is the data storage capacity. This storage should be large enough so that the equipment can be left in the field for enough time to make the station a practical system. The required data storage would depend very much on the survey and survey conditions, but even with a 6 Mbyte memory (or 3 Mword) this only translates to approximately 9 hours of storage at a sampling rate of 32Hz for 3 channels of data. This would be impractical for most surveys and unless some backup form of storage (tape streamer, PC) is available, continuous recording is not viable. The alternative is to use a triggering method (external or internal) which allows the unit to only store data once an "event" has been detected. A "pre - event memory", simply a memory buffer, stores the pre-trigger data, while a simple long term, short term averaging (LTA/STA) algorithm is used for detecting a trigger. This works well for local events. For detecting the longer wavelengths of teleseismic events a much more sophisticated algorithm is neccessary. The simple LTA/STA approach, and some teleseismic trigger techniques were investigated. These techniques can be applied equally well in software or hardware. The software algorithms will be discussed, along with the hardware equivalent. The LTA/STA algorithm was implemented on a PC using Turbo Pascal(5.5). The teleseismic triggering algorithm would probably have to be implemented at a lower sampling frequency of 16Hz because of time constraints (This sampling rate is adequate for teleseismic studies).

4.1 Long Term Short Term Averaging

The discussion of simple averaging techniques is the same whether considering the software or hardware case. Using a simple 8-bit microprocessor it is essential that all the computations be done as efficiently as possible. If the length of the averages is kept to multiples of 2, all multiplies and divides can be achieved by either shifting the data to the left or right. The 6309 microprocessor has an 8-bit × 8-bit multiply feature, but as this is extremely slow it is to be avoided where possible. Time constraints dictate that all filtering, and processing be kept to the minimum. The actual data representation and deconvolution of the gain ranged data has been discussed in the software section. Only the specifics of the LTA/STA algorithms will be discussed here. The principle behind this method is to compare the output of a long period integrator (Long term average) with that of a short period integrator (Short term average). This method is well documented [Ambuter et al (1974), and Prothero (1980)], but will be dealt with briefly. A very simple LTA/STA scheme suitable for detecting short period local events is described by Ambuter et al (1974). Figure(4.1) shows the block diagram for the hardware event detector. The long term averager is a hardware integrator with a relatively long time constant, or a software routine averaging a large number of samples. The signal is first rectified before averaging. In general the LTA/STA time ratio is between 5-100:1. The time constant of the long term averager is much longer than the longest period of the seismic signal that must be recorded. The LTA therefore sets the level of the background noise, while the STA follows the envelope of the incoming signal. These are then compared and when the STA exceeds the LTA by a specified amount (the threshold) for a specified time (the duration), a trigger is recognised. The setting of the threshold and duration are critical, and a compromise must be made between the sensitivity of the algorithm to noise, and the sensitivity to real events. To rectify the signal it is unfortunately not as simple as taking the absolute value of the signal, due to possible DC offset. Initially it was thought that any offset would weight the LTA and STA equally, but after careful consideration it was decided that this was not the case. It is therefore necessary to calculate the signal offset by taking a true long term

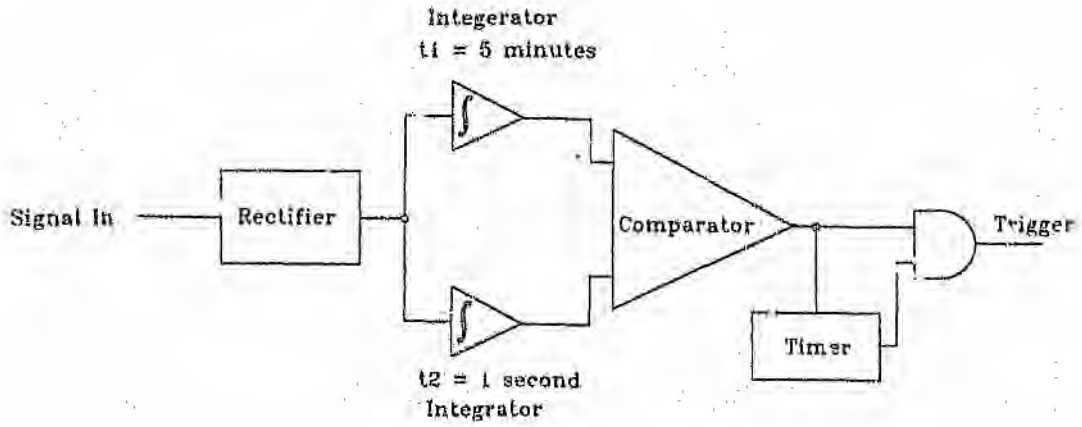


Figure 4.1: LTA/STA hardware - Using two integrators

average of the signal. The offset must be calculated from a true long term average significantly longer than the LTA. A value of 16384 samples (or 16 × the STA) was adopted. This means a setup delay of 1 minute at a sampling rate of 256Hz, and a setup delay of 9 minutes for a 32Hz sampling rate. For the LTA/STA filters, Prothero (1980) adopted a recursive filter of the form

$$y(i) = \frac{[x(i) + y(i-1)(N-1)]}{N}$$

where N is the number of samples in the filter, y(i) is the filter output, x(i) is the current sample and y(i-1) is the previous filter output. The transfer function of this filter is given by:

$$H(f) = \frac{1}{N - (N-1)e^{-j2\pi FT}}$$

where T is the sample interval.
See Appendix(B).

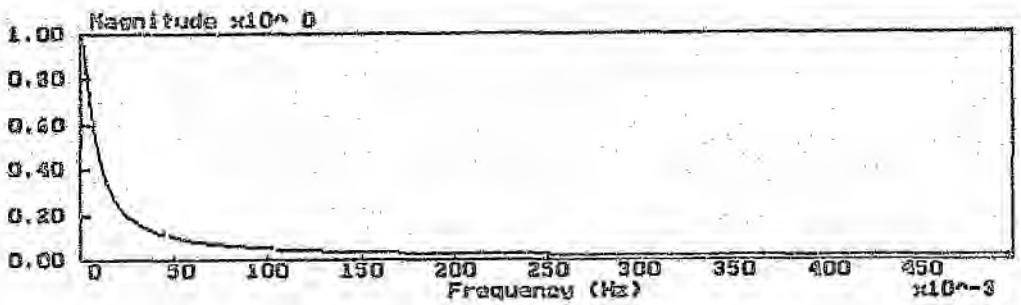


Figure 4.2: Magnitude response for 32 sample filter

The gain of the filter is given by

$$|H(f)| = \frac{1}{\sqrt{N(1+(\frac{N-1}{N})^2 - 2\frac{(N-1)}{N}\cos(2\pi FT)}}$$

Figure (4.2) shows the magnitude response of the filter (for a 32 sample filter, normalized to 1Hz sample interval). Figure(4.3) shows the response of the filter to a step input. As can be seen from the figure, the decay time of the filter is substantially larger than the attack time. This gets worse for filters with larger values of N. For a pulse input and a filter length of 1024 samples, the average takes 2 minutes to return to half its maximum value. Prothero gets round this problem by noting that if we rewrite the difference equation in the form

$$y(i) = \frac{y(i-1) + [x(i) - y(i-1)]}{N}$$

considering $[x(i) - y(i-1)]$ as a correction term, and if the output of the filter is decreasing, this factor can be weighted to speed the recovery. This scheme has the advantage that it is computationally simple, we only need to know the value of the latest sample and the value of the previous average. The disadvantages are the problems with the decay part of the curve.

A simpler solution is to use a true "running average" where the latest term

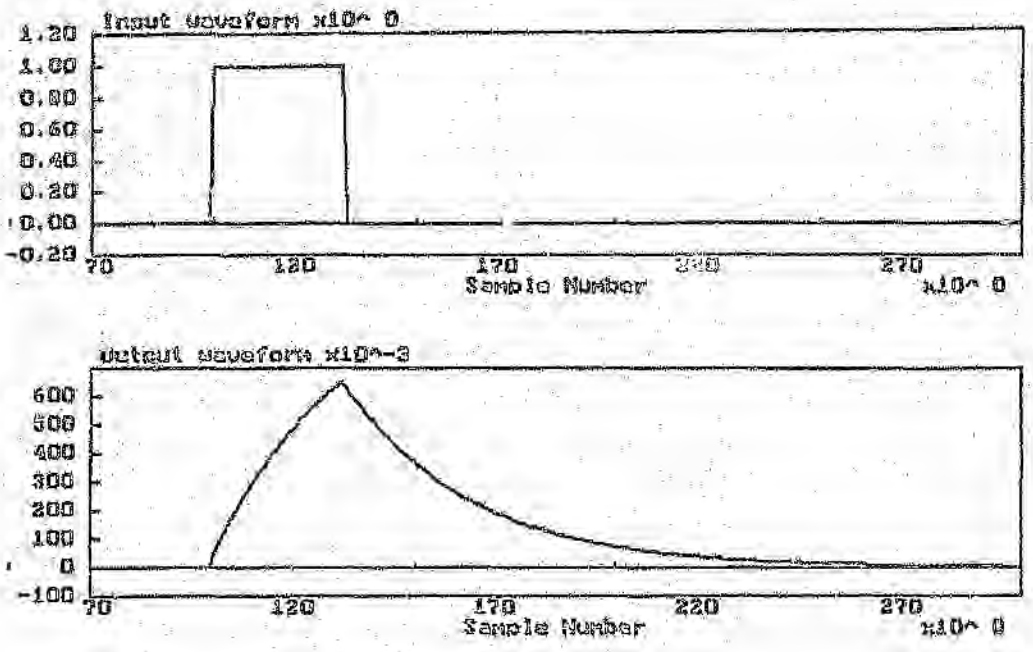


Figure 4.3: Response of filter to a step input

is added and the first term is deleted from the previous average.

$$y(i) = \frac{x(i) + y(i-1)N - x(i-N)}{N}$$

This has the disadvantage of having to set up a pointer scheme to keep track of the first and last element in the filter. The transfer function of the filter is given by

$$H(f) = \frac{1 - e^{-jN2\pi fT}}{N - Ne^{-j2\pi fT}}$$

See appendix(B) The magnitude response is shown in figure(4.4) while the response to a rectangular pulse is shown in figure(4.5). The magnitude response is similar to the filter suggested by Prothero, and it dosen't suffer from the recovery problems experienced with the previous filter. The extra upkeep involved in calculating the average is balanced by the fact that no weighting is required to correct the decay of the filter.

The values for the LTA/STA was chosen to be 16:1 (or 1024:64). The threshold and duration values are dependent on the noise level. With a sampling

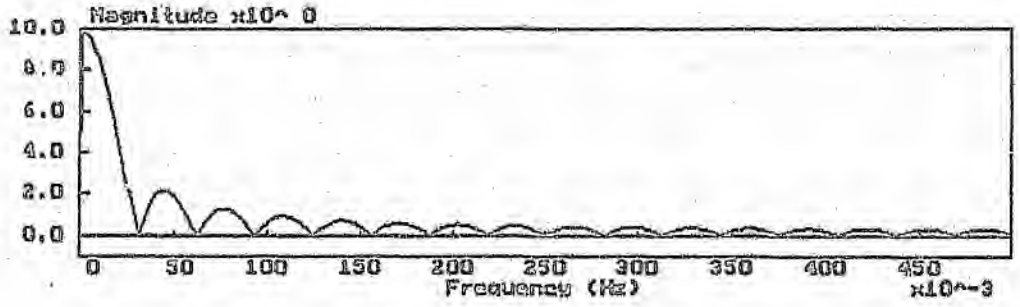


Figure 4.4: The magnitude response of the true averaging filter

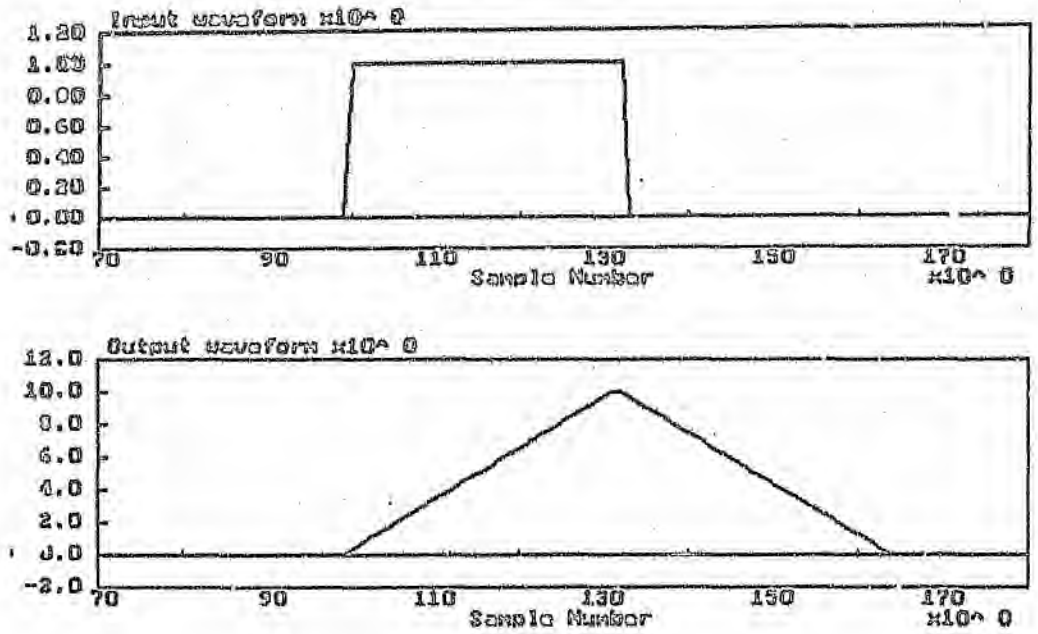


Figure 4.5: The response of the true averaging filter to a rectangular pulse

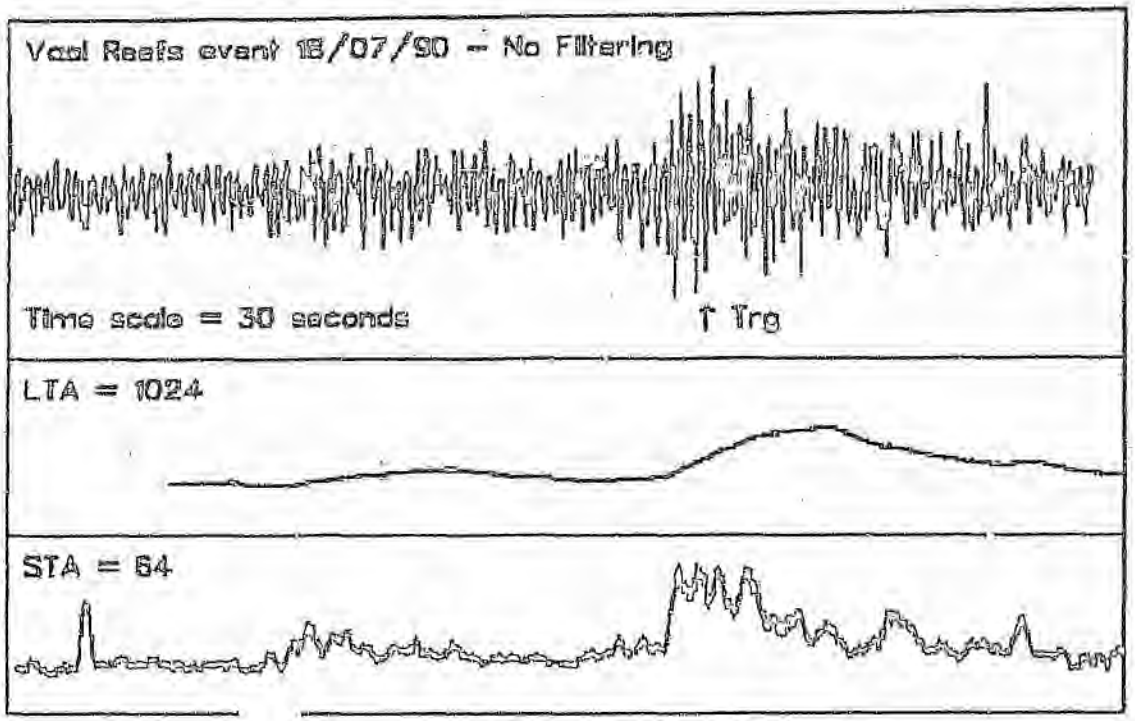


Figure 4.6: A local event showing the behaviour of the STA and LTA values

rate of 128Hz, LTA = 1024 samples and STA = 64 samples, a value of 30 (rectified data is an integer value from 0 to 32767) for the threshold and 50 samples for the delay was found to provide reasonable rejection of spikes, while small events should be detected. With a higher noise level present, the algorithm would have to be made more insensitive by increasing the threshold and increasing the delay. The response of the LTA and STA to a local event is shown in figure (4.6). The "↑" in the diagram indicates that a trigger has been recognised. A duration of 50 samples in this case is about 0,5 of a second. This was sufficient to preclude most high frequency spikes.

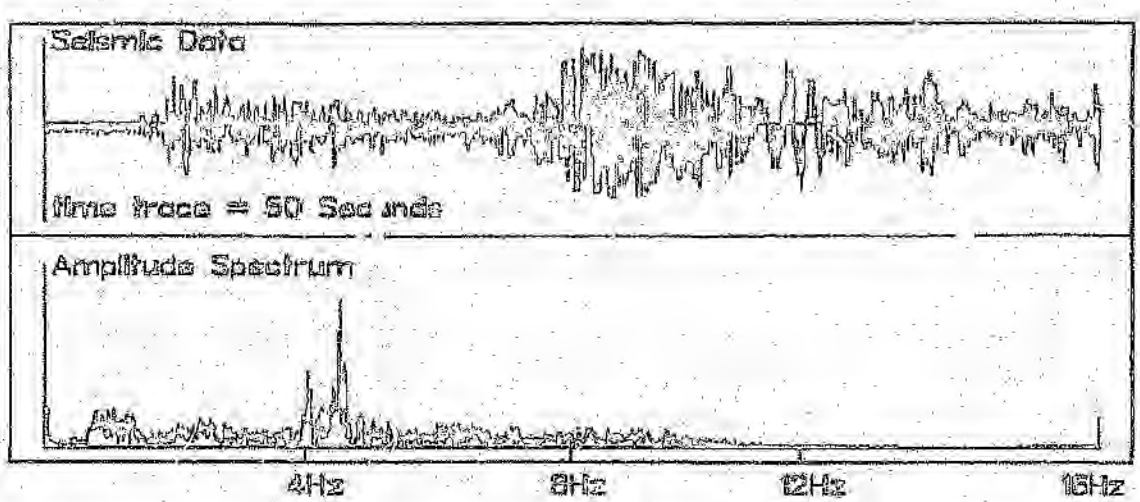


Figure 4.7: A typical local event with it's power spectral density. The small P wave train is followed by the larger S waves

4.2 Long period event detection

The triggering of teleseisms from a single station is much more of a problem. The P wave on a long period instrument may only provide a few signal swings on which to trigger. Also in a tectonically active region, or one with a great deal of mining activity, tremors can be numerous enough to preclude using the simple schemes already discussed. Apart from the fact that these tremors will cause a trigger, making the algorithms sensitive enough to detect teleseisms will also make them extremely sensitive to noise. Prothero (1980) suggested a good starting point for triggering teleseisms is to look for a LTA/STA trigger without a high frequency signal (>3Hz) present. Evans and Allen (1983) have expanded on this idea and have proposed a teleseism specific algorithm. The algorithm adopted by Evans and Allen reliably detected major phases from earthquakes more than 3° from the source while rejecting most noise events and earthquakes closer than 3°. From their studies of data in the Western United States, they found that events occurring more than 3° from

the stations lose most of their energy above 2Hz. The frequency spectrum of a local earthquake contains energy up to at least 10Hz. As can be seen from figure (4.7), this is true for a typical local event. Typical cultural noise such as vehicles and other natural noise such as wind tend to have energy extending well into the 20Hz range.

Evans and Allen suggest the use of a low pass filter with a corner frequency of 2Hz and a LTA/STA scheme to obtain an idea of the PSD at the lower frequencies, and a similar scheme but with a 3 - 8Hz band pass filter for the higher frequencies. The detection of a teleseism is than the occurrence of a low frequency trigger without a concurrent high frequen trigger. Figure(4.8) shows the logical operation of the modified Evans and Allen algorithm. The algorithm shown is a simplified version of the Evans/Allen algorithm. The original program will also detect very large local events (or large teleseismic events which may clip, causing a concurrent high frequency output). The original also prevents retriggering of the algorithm by bursts of teleseismic signals. The algorithm in figure(4.8) is the basic teleseism detection algorithm. The signals from the band pass and low pass filters are rectified and passed to the LTA/STA routine. This routine would be a similar scheme to that discussed for local events. If a high frequency trigger is detected, this retriggers a 3 second timer which inhibits any teleseismic triggers. Suitable digital filters were designed (see Appendix(D)) for the low pass and band pass filters, however the algorithm has not been tested. It was initially thought that teleseismic data from the 8-bit system could be used to test the algorithm, but unfortunately the data is already low pass filtered at 3 Hz. Also no teleseismic data has been recorded on the 16-bit system.

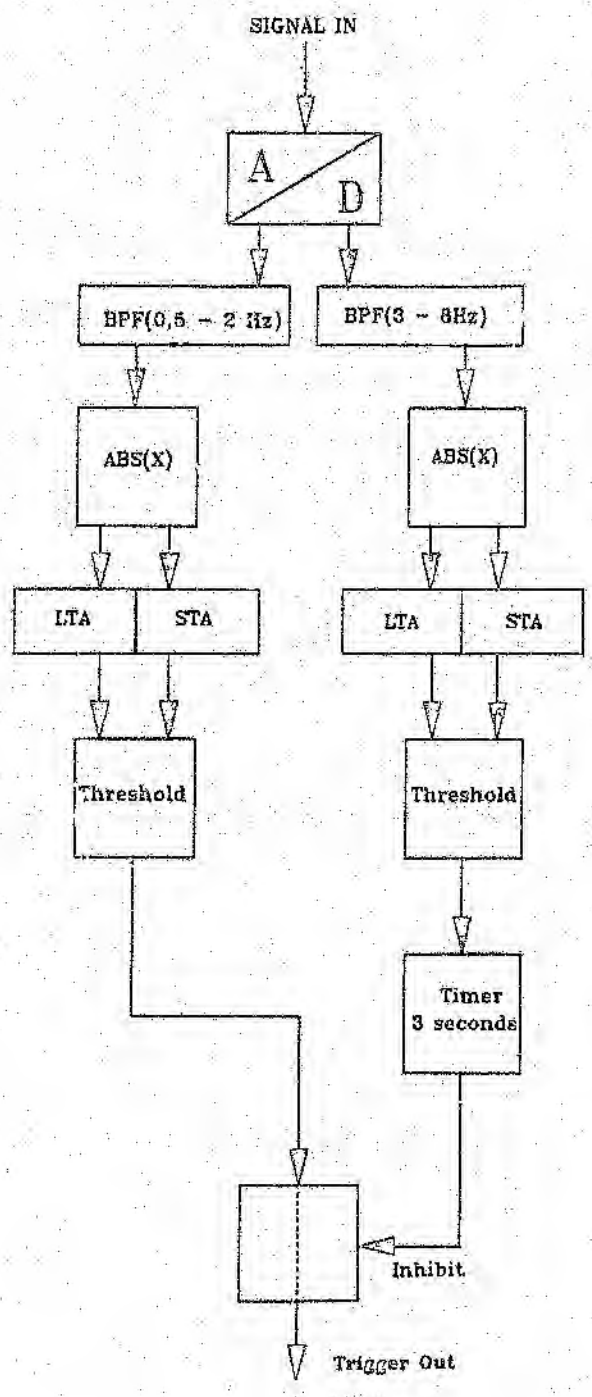


Figure 4.8: ... w chart of Evans/Allen algorithm

Chapter 5

Results

The system has not as yet been field tested. Various problems were encountered with the prototype in the laboratory, and most of the problems have been addressed. One of the major problems was high frequency noise entering the analogue stages from the 2 MHz microprocessor clock frequency. Noise was also introduced by the follow and hold gates. Most of the noise problems were sorted out, however the data given in the examples was recorded on the prototype and is rather noisy. Software filtering was employed on the data to remove most of the aliased high frequency noise.

5.1 Noise Problems

In the original design, the anti-alias filter, comparators and gain switching amplifier were all on one board. Three of these boards were needed for a three component system.This approach however led to all sorts of noise problems. Initially the latch for the comparators was "free running" at 1 MHz (derived from the real time clock). It would latch on the downgoing edge of the 1 Mhz signal while the A/D board logic was sensitive to the upgoing edge. This was to ensure that the gain bits could not change while the A/D was latching them. The presence of the 1 MHz signal, close to the analogue circuitry led to major noise problems as the 1 MHz noise was being induced into the

analogue line. Filtering of the power lines (with ferrite beads and tantalum capacitors) did not lower the noise. The design was then changed so that the gain was latched at the sample rate. A delay was introduced between the latching of the gain and the follow and hold operation to ensure that the gain ranging amplifier and control circuitry had settled down before reading the data. This technique helped but the signal was still noisy. An interesting aspect of the noise (and one that with retrospect can be a problem with this sort of gain ranging situation) was that the signal to noise ratio gets worse, the *lower* the gain ! This is caused by interference after the gain switching amplifier and the level is constant for all gain steps. This means that if a large signal is present, when the wave is reconstructed, the noise in the high gain stages relative to the size of the signal is negligible. This noise is however also present in the low gain stages and it now becomes significant. This produces a very strange signal where the high level input (ie. low gain steps) is dominated by noise. There was no simple solution to this problem. Even if the interfering signal is a very high frequency (relative) to the signal, as in this case, it could not be filtered out. The major difficulty was that any filtering after the gain ranging amplifier would produce large slew rate errors and "noise" would be introduced when reconstructing the signal.

After many attempts at reducing the noise, two main causes of this noise were isolated.
(1) - The presence of the switching comparators and digital circuitry (latches, decoders) near to the analogue circuitry. (80μV)
(2) - Bad layout of the A/D board, where the 1MHz line used in decoding, multiplexing and by the A/D chip, was inducing high frequency noise into the analogue circuitry. (20mV)

The solution to problem (1) was to split the original board into three separate boards with three channels on each board. So the system now comprised of, 1 anti-alias filter board, 1 comparator board, and 1 gain ranging board, each with 3 channels. This had a dramatic effect on the noise level and the noise dropped to around the 1μV level. This noise is due to switching noise and not due to the amplifier noise.

The solution to problem (2) was to change the layout of the A/D board and again a dramatic improvement occurred. The 1MHz line was kept as short as possible and the digital circuitry kept physically as far away as possible from the analogue circuitry. Particular attention was paid to the "starring" of the analogue and digital grounds, all having their common connection at the A/D itself. The specified noise on the A/D should have been about 1 bit or 1,2 mV (for a 12 bit + sign A/D). This was not quite achieved and the noise is closer to 2 bits or 3,6 mV. This is however significantly better than the 20mV or 4bits of noise achieved with the previous layout.

5.2 Dynamic Range

The overall gain of the analogue circuitry is $\times 1024$. So the smallest signal that is above the noise is say $5\mu V$ p-p. The largest signal is 10V p-p. So the dynamic range is

$$20\log(\frac{10}{5\times 10^{-6}}) = 126dB$$

If we could resolve down to the 1 bit level or $1{,}2\mu V$ 134dB dynamic range would be possible. Our limiting factor is the switching noise on the A/D board. The 126 dB is within the 120 dB to 140 dB stated as our goal.

5.3 Distortion due to gain ranging

There was no noticeable distortion due to gain ranging once the signal has been reconstructed. Care must be taken when setting up the switching points of the comparators (by adjusting the potentiometers controlling the rectification of the incoming signal) because if the output for one of the gain stages slightly over-ranges, spikes appear in the data. It is easy to see this problem when the signal is badly over-range, but if it is only slightly over-range then random spikes occur in the data. Because the gain is only latched at the sampling rate it is also not that easy to see with an oscilloscope, because the spike is only seen if the sample point corresponds to the amplifier over-

ranging. For this reason it is easier to set up the comparators when the latching of the gain is " free running" at 1 MHz. There is a jumper on the back plane where the sampling rate or the 1 MHz signal can be selected if problems are encountered in setting up. An alternative is to set up the gain ranging system with a DC signal

5.4 Power consumption

The 5V digital supply draws 100 mA, and the ±15 V supply draws 60 mA per rail. This is 2,3 W or 192 mA at 12 V. The power supply using DC-DC convertors was found to be typically less than 75% efficient. The total continuous current was approximately 300 mA. A more efficient power supply would have to be considered for the field system.

5.5 Time accuracy

The RTC can accept the input either from a crystal or a clock source. The short term stability of a simple crystal oscillator used is in the region of 1 in 10^6. This means an error of 3,6mS per hour. At 256 Hz sampling rate one sample corresponds to 3,9 mS. So unless the time update was updating the clock every hour errors would occur after 1 hour. At 32 Hz one sample corresponds to 31 mS. This means 5 hours before errors would be noticed. This would be just acceptable if the radio time synchronisation system is used. If a TCXO (Temperature compensated crystal oscillator) were used an accuracy of 1 in 10^7 could be expected. This is the case with the present 8-bit system where correlation between stations using the time update system is better than 10mS. The expense of a TCXO was prohibitive for the prototype, but would have to be incorporated in a practical field system.

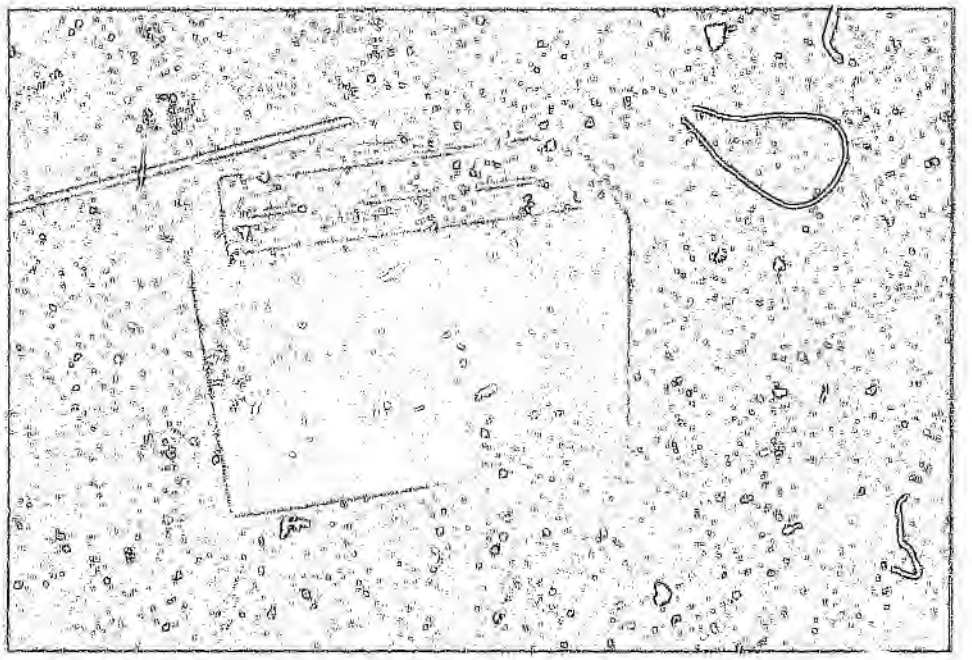


Figure 5.1: Weatherproof enclosure

5.6 Summary of technical specifications

- 3 Channels
- sampling rate 32Hz or 256Hz
- Anti-alias filters 8Hz and 52Hz (-3dB cut)
- Dynamic range = 126dB
- Noise figure 5μV
- Memory 384KByte expandable to 6,144 MByte
- Power Consumption 2,4W
- Time accuracy = 1 part in 10^6

5.7 Enclosure

A weatherproof enclosure, with the system rack mounted is shown in figure(5.1). The PC would also be mounted inside. (A small motherboard and hard drive can easily be accomodated in the enclosure). The batteries can

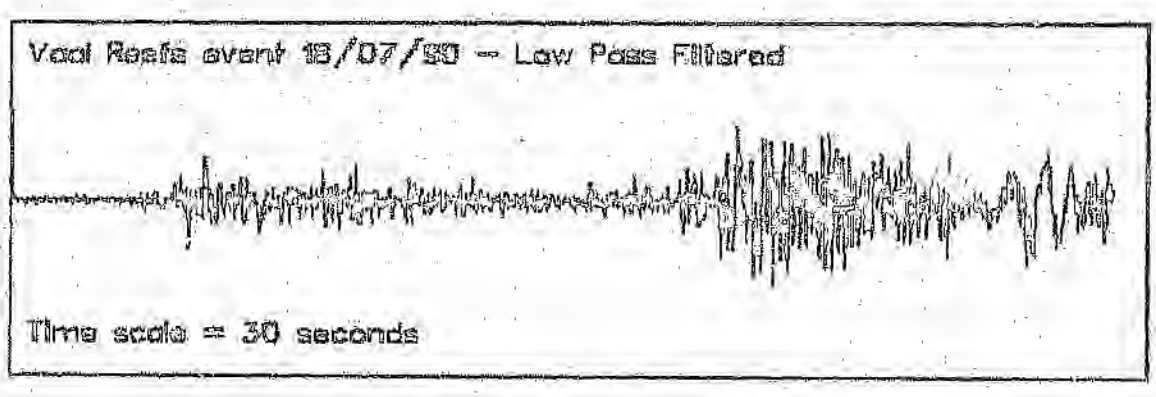


Figure 5.2: Event at Vaal Reefs 18/07/90

also be included in the enclosure, small sealed batteries being adequate.

5.8 Data recording and triggering

The prototype consisted of the acquisition system and 1 memory board (ie. 384 KByte - 1 file of 4 blocks could be stored). The software was the same as for the field system, except that only the one memory board was used. It requires only minor software modifications to adapt the system to work with more memory boards. Figure(5.2) shows a local event recorded by the prototype on 18/07/1990. Unfortunately there was some unexplained resonance at 21Hz and also some 50Hz power line noise present. (See the PSD plot figure(5.3) of the noise no signal present) An 8Hz software filter was applied to the data to eliminate the noise. This particular event was a large event (magnitude 3,9) at Vaal Reefs and the algorithm triggered on the S arrival. Figure (5.4) shows the entire event together with the trigger levels. This was the first successfully recorded event and the triggering scheme had not been optimised. Because of the large noise level present the triggering scheme was set to a fairly insensitive level and as a consequence it did not

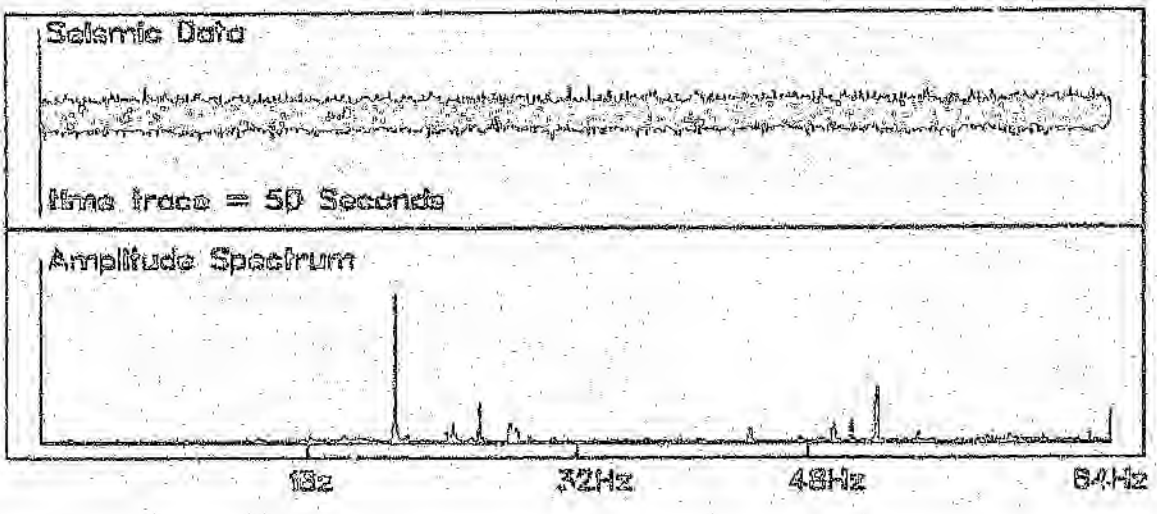


Figure 5.3: Noise together with its PSD

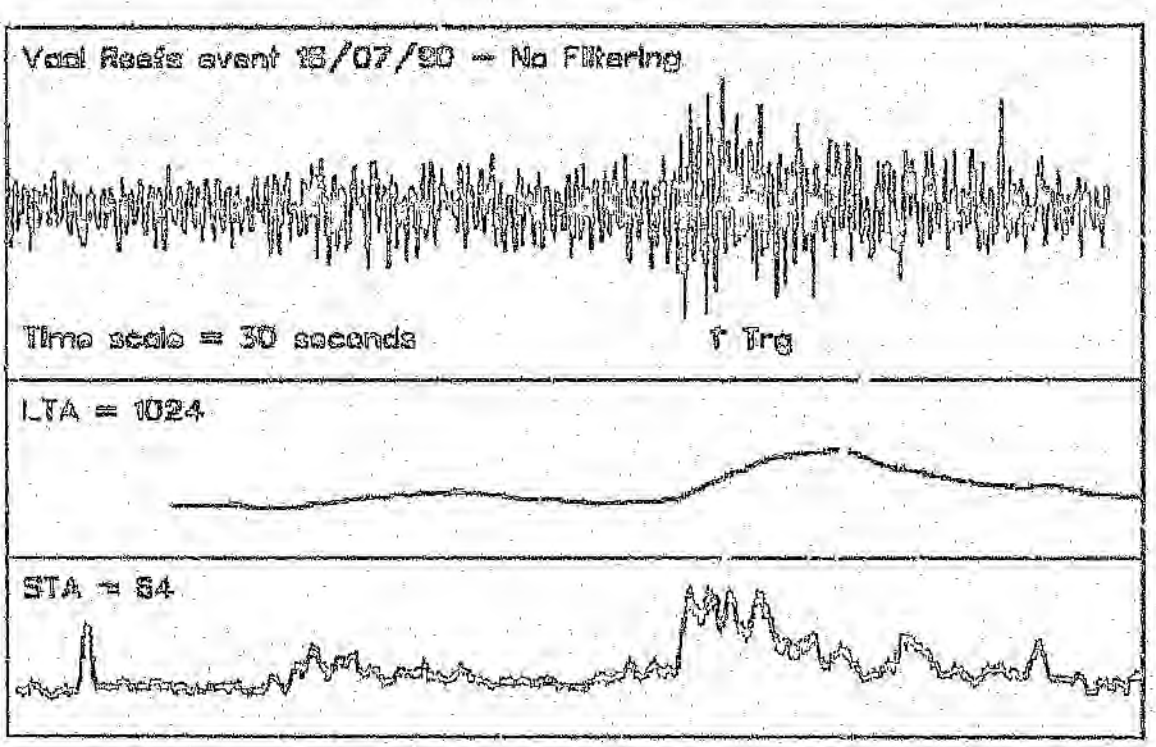


Figure 5.4: The event showing the trigger criterion

trigger on the P arrivals. The triggering algorithm was being run on the PC, a single channel of data being recorded at 128Hz (in this case the vertical channel) and dumped to the PC. The PC then analyses the block for a trigger. This particular event was triggered using the simple LTA/STA technique (LTA = 1024), (STA = 64), delay = 100 samples and the threshold = 30. This gave reasonable rejection against spikes but was not particularly sensitive as can be seen from the example. Reducing the value of the threshold makes the system more sensitive, while reducing the delay makes the algorithm more susceptible to spikes.

The highest sampling rate possible is dependent on the conversion time for the analogue to digital converter, for the ADC1225 it is 128μS. This is 7812 Hz for a single channel, or 2604 Hz for three channels. There is just enough time, without triggering to process three channels at 128 Hz. With triggering, assuming an average of 12 machine cycles per instruction, and 50 instructions for the algorithm, a sampling rate of 1Khz would just be possible (if the algorithm was only implemented for one channel). This may be a bit optimistic but a sampling rate of 512Hz with triggering should be possible.

Chapter 6

Conclusions

A microprocessor based acquisition system gives flexibility for possible future modifications. At the sampling rates used in seismology a simple microprocessor system is adequate for data storage and simple signal processing. Unfortunately a large portion of time was spent building and debugging the system, and as yet it has not been field tested. Only the time synchronisation system, using the FM broadcast stations has been extensively tested, using the present 8 bit field stations.

6.1 Gain ranging

A dynamic range of 138 dB should be possible using these techniques. Noise problems meant that this was not quite achieved (126 dB). Most of the noise was induced into the A/D analogue circuitry. Presumably microprocessor switching noise and follow and hold noise. The gain ranging system also proved rather difficult to set up and if not correctly adjusted can introduce spikes into the data. Once correctly adjusted however, the distortion due to the gain ranging was acceptable. Top compression could be just as easily incorporated into the design and the use of separate gain ranging boards means that these could quite easily be interchanged.

6.2 Triggering

Prothero (1980) has found that data compression for local events (relative to continuous recording) of 100 or greater (depending on the character of the noise) can be achieved using simple LTA/STA algorithms. Unfortunately not enough data has been recorded to substantiate this claim, however similar compression should be possible. The Pascal program used to trigger the events, showed promise, even in the presence of a large high frequency noise envelope. Large local events were successfully triggered, although with the presence of the noise, setting up of the triggering criterion (duration and threshold) was critical.

The problem of teleseismic triggering and possibly failing to record an event has not been fully covered, further work needs to be done and possibly a separate board for detecting triggers could be designed. The external trigger line to the DAC board would be used to initialise data transfer. At present it is probably better to record all data rather than risk not recording an event (with a number of field stations you may get one station that misses the trigger and important data lost). There was also not sufficient time to explore the triggering algorithms, but the Evans and Allen algorithm (Evans[83]) should provide a starting point for further study. Also suitable digital filters for the bandpass and low pass filters required for the algorithm are described in Appendix(C).

6.3 Power Considerations

The continuous power drawn (2,4 W) is 3× less than the 8-bit system (This is the actual power drawn by the boards, not including the DC-DC converters which were found to be very inefficient). With the possibility of using a much larger buffer memory (up to 6 Mbytes) the PC is also powered up less often. This means that one solar panel should be sufficient for the power needs of the system.

6.4 Time Accuracy

A simple crystal oscillator is not sufficiently stable for a practical field station. At a 16 Hz sampling rate, with a time update once a day, the accuracy would be just within tolerance, however at higher sampling rates (256 Hz) it is totally impractical. (It was also found that with the time decoder units if the radio becomes slightly detuned, the updates fail to operate - so cannot necesarily rely on daily time updates). A TCXO (temperature controlled crystal oscillator) such as the unit used in the 8-bit system could be used, but is unfortunately extremely expensive. Also using the time update system, and assuming a reasonable amount of updates (once per day) that sort of accuracy is not necessary. A cheaper alternative is to design a circuit using positive and negative temperature coefficient capacitors to improve the temperature stability, which should provide acceptable accuracy (1 part in 10^7).

6.5 Anti-Alias filters

Using relays to switch capacitors on the filter board to select different frequencies made the construction of the filter PC-board difficult. The prototype only has options for two cut frequencies, if more were needed the construction of the boards would be extremely difficult. To overcome this problem, the system could be run at its highest sampling rate (with an anti-alias filter for that frequency) and for lower sampling rates the extra samples summed and averaged to avoid aliasing. (The averaging operation is in fact a low pass filter as shown in section(4)). The algorithm can be implemented with adds and shifts and will not intefere seriously with the processing time. The large anti-alias filter boards and the associated switching circuitry can be done away with. The power consumption will however be increased at the higher sampling rate.

Appendix A

Calculation of comparator switching voltages

The comparator circuit of figure(A.1) can be simplified to the circuit as shown in figure(A.2).

Then we can write

$$V_{on} = R_g(I_i + I_o) \tag{A.1}$$

$$V_o^+ = I_i(R_g + (R_f + R_g))I_o \tag{A.2}$$

$$V_r = I_i(R_s + R_g) + I_o R_g \tag{A.3}$$

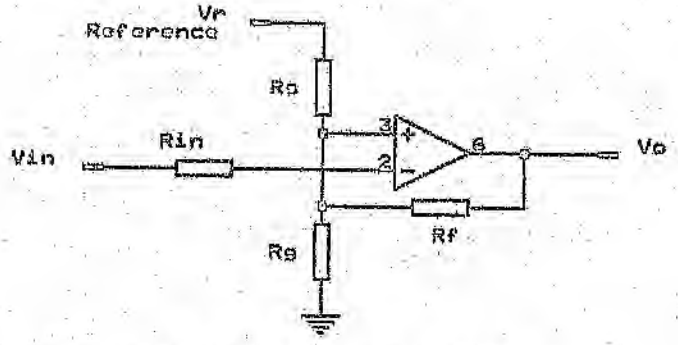


Figure A.1: The comparator circuit

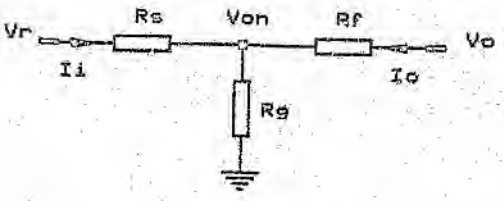


Figure A.2: simplified circuit

Where V_o^+ is the output for a positive swing.

from [A.1]

$$I_0 = \frac{V_{on} - I_i R_g}{R_g} \tag{A.4}$$

substituting in [A.3] we get

$$I_i = \frac{V_r - V_{on}}{R_s} \tag{A.5}$$

substituting [A.4] in [A.2] gives

$$V_o^+ Rg = (R_f + R_g)V_{on} - I_i(R_f R_g)$$

substituting [A.5] into the above equation and after some manipulation gives

$$V_{on} = \frac{V_o^+ R_g R_s + V_r R_f R_g}{R_f R_g + R_f R_s + R_g R_s}$$

similarly for the case of V_{off}, V_o^- (ie. a swing to the negative rail) we get

$$V_{off} = \frac{V_r R_g R_f - V_o^- R_g R_s}{R_s R_f + R_g R_f + R_g R_s}$$

Appendix B

Calculation of transfer functions for averaging filters

For a difference equation of the form (Stanley (84))

$$y(n) = \sum_{i=0}^{k} a_i x(n-i) - \sum_{i=1}^{k} b_i y(n-i)$$

the transfer function is given as :

$$H(z) = \frac{\sum_{i=0}^{k} a_i z^{-i}}{1 + \sum_{i=1}^{k} b_i z^{-i}} \tag{B.1}$$

For a first order filter this becomes

$$= \frac{a_0}{1 + bZ^{-1}} \tag{B.2}$$

Now for the Prothero [80] filter of the form

$$y(i) = \frac{[x(i) + y(i-1)(N-1)]}{N}$$

So $a_0 = \frac{1}{N}$ and $b_1 = \frac{1-N}{N}$ substituting in (B.2) gives

$$H(f) = \frac{1}{N - (N-1)e^{-j\pi 2\ldots}}$$

Similarly for the 'real averaging filter'

$$y[i] = \frac{x(i) + y(i-1)N - x(i-N)}{N}$$

So $a_0 = \frac{1}{N}$ and $a_N = \frac{1}{N}$ and $b_0 = 1$ substituting in (B.1) gives

$$H(f) = \frac{1 - e^{-jN2\pi fT}}{N - Ne^{-j2\pi fT}}$$

Appendix C

Jumper Settings for 8255 I/O Card

The jumpers on the I/O card can be changed, thus changing the address to ensure compatibility with the PC in use. Figure(C.1shows the table of settings, and the physical position of the jumpers on the card)

	J1/J2	J3/J4	J5/J6	Port Address
1	J1	J3	J5	1B4 - 1B7
2	J1	J3	J6	3B4 - 3B7
3	J1	J4	J5	1F4 - 1F7
4	J1	J4	J6	3F4 - 3F7
5	J2	J3	J5	1BC - 1BF
6	J2	J3	J6	3BC - 3BF
7	J2	J4	J5	1FC - 1FF
8	J2	J4	J6	3FC - 3FF

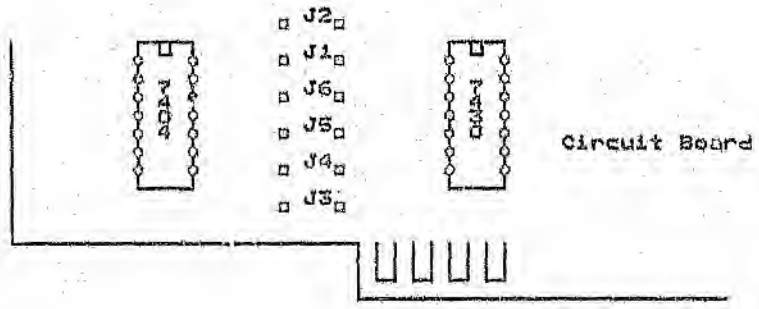


Figure C.1: Jumper settings for the 8255 I/O card

Appendix D

Filter calculations for the triggering algorithm

Suitable filters can be designed using IIR (infinite impulse response) filter techniques (Stanley [84]). The bilinear Z transform (BZT) can be applied to standard low pass analogue filters, to obtain the transfer function in the Z domain. A Bandpass or highpass transformation can then be applied for the desired response.

It can be shown (Stanley [84] pp. 89-90) that the transfer function of the form

$$H(z) = \frac{\sum_{i=0}^{k} a_i z^{-i}}{1 + \sum_{i=1}^{k} b_i z^{-i}}$$

can be represented in the discrete time domain by the difference equation

$$y(n) = \sum_{i=0}^{k} a_i x(n-i) - \sum_{i=1}^{k} b_i y(n-i)$$

It is a simple matter then to apply the bilinear Z transform to an analogue prototype and to obtain a difference equation which can be applied to the seis-

mic data. A standard 2 pole butterworth response is used as the analaogue prototype, but any other response or order filter can be substituted.

- The Bandpass filter

The standard form for a 2 pole Butterworth filter is as follows.

$$G(s) = \frac{1}{1 + \sqrt{2}s + s^2} \tag{D.1}$$

where

$$s = D\frac{1 - EZ^{-1} + Z^{-2}}{1 + Z^{-2}}$$

- Bandpass filter transformation

where

$$D = \lambda_r \cot(\frac{\pi}{2}(\nu_3 - \nu_1))$$

where ν_1 and ν_3 are the normalised bandpass cut frequencies and λ_r is the analogue low pass reference radian frequency (assumed to be = 1).

Now for a sampling rate of 16Hz

$f_s/2 = 8Hz$

so $\nu_1 = \frac{3}{8} = 0{,}375$
and $\nu_3 = \frac{8}{8} = 1$

Then D = 0,66817864 and E = -2,00000002

After manipulation of (1) we get

$$G(Z) = \frac{0{,}41816334(1 - Z^{-2})^2}{1 + 2{,}63667195Z^{-1} + 0{,}28382933Z^{-2} - 0{,}04350731Z^{-3} + 0{,}20971536Z^{-4}}$$

Note from the 2 pole original, we get a 4 pole band pass filter.

The real values would also have to be scaled to integer values for operations with the microprocessor.

So for this filter :

coefficient	*Real variable*	*integer variable*
a_0	0,41816334	5197
a_1	0	0
a_2	-0,083632668	-10393
a_3	0	0
a_4	0,41816334	5197
b_1	2,63667195	32767
b_2	0,28383933	3527
b_3	-0.04350731	-541
b_4	0,20971536	2606

- Low pass filter

Again the low pass butterworth response is given as

$$G(s) = \frac{1}{1 + \sqrt{2}s + s^2} \qquad (n\,?)$$

and for the BZT

$$S = Cfrac(1 - Z^{-1})(1 + Z^{-1})$$

where $C = \lambda_r \cot \frac{\pi}{2}\nu_r$

and ν_r is the normalised cut frequency.

So for $\frac{f_s}{2} = 8$

$\nu_r = \frac{3}{8}$

After manipulation of [2]

$$G(z) = \frac{0,18669433(1 + 2Z^{-1} + Z-2)}{1 - 0,23146901Z-1 + 0,20971536Z^{-2}}$$

Again the real values must be scaled to integer values.

So for the low pass filter :

coefficient	*Real variable*	*integer variable*
a_0	0,18669433	16384
a_1	0,37338866	32767
a_2	0,18669433	16384
b_1	-0,23146901	-20313
b_2	0,20971536	18404

References

[1] Allen, R.V. (1978) *Automatic earthquake recognition and timing from single traces*, Bull. seism. soc. Am. ,vol.68, no.5, pp1521-1532.

[2] Ambuter, B.P. and Solomon, S.C. (1974) *An event recording system for monitoring small earthquakes*, Bull. seism. soc. Am., vol.64, pp1181-1188.

[3] Beauchamp, K.G. (1975) *Walsh functions and their application*, Techniques of physics series number 3, Academic Press, New York.

[4] Evans, J.R. and Allen, R.V. (1983) *A teleseism-specific detection algorithm for single traces*, Bull. seism. soc. Am., vol.73, pp1173- 1186.

[5] Goforth, T. and Herrin, E. (1981) *An automatic detection algorithm based on the Walsh transform*, Bull. seism. soc. Am., vol.71, pp1351- 1360.

[6] Green, R. W. E. (1979) *A data acquisition and processing system, applications to seismological problems in Southern Africa.* Part1 & 2 - Thesis

[7] Green, R. W. E. (1984) *Design Considerations for an underground seismic network*, The South African Institute of

mining and metallurgy symposium series no.6, Rockbursts and seismicity in mines pp67-74.

[8] Kuo, F.F. (1966) *Network analysis and synthesis* Second edition, John Wiley and Sons.

[9] Horowitz, P. and Hill, W. (1983) *The Art of Electronics.* Cambridge University Press

[10] Johnson D.E., Johnson J.R. and Moore H.P. (1980) *A handbook of active filters*, Prentice Hall.

[11] Morris Mano M. (1979)*Digital logic and computer design* Prentice Hall.

[12] Prothero, W. A. Jr. (1980) *Earthquake signal processing and logging with a battery powered microcomputer*, Bull. seism. soc. Am., vol.70, no.6, pp2275-2290.

[13] Robinson E.A. and Durrani T.S. (1986) *Geophysical signal processing*, Prentice Hall.

[14] Stanley, W.D., Dougherty G.R. and Dougherty R. (1984) *Digital signal processing*, Restor publishing company, Inc.

[15] Walker, A.J. *Structured information processing system design.* February 1984.

[16] *Hitachi Microcomputer Data Book*, pp478-508

[17] *MC6809-MC68C9E Microprocessor programming manual* MOTOROLA INC

ADDENDUM TO DISSERTATION

'DESIGN OF A SEISMIC DATA ACQUISITION SYSTEM AND AUTOMATIC TRIGGERING SOFTWARE'

Robert Sidney John Cole

Johannesburg 1991

ADDENDUM

The addendum contains the programs and data sheets for the seismic data aquisition system described in the dissertation.

- CONTROL program listing
- DAC program listing
- MEMORY program listing
- RD_SEIS program listing
- CONVERT program listing
- List of 6809 (6309) assembler nmemonics
- Data sheet for 6321 I/O port
- Data sheet for 146818 RTC

The programs listed here are the prototype programs and will still have to modified further for field use. The programs are structured as described in the PDL descriptions. General comments are used throughout however, to allow people not familiar with assembler programming to modify the programs.

CONTROL Program

```
;-------------------------------------------------------------------------
;
;  LIB1.DCL  (Global Declarations for Control Software)
;
;  Constants and variables for routines in CTRL and LIB1.INC
;  LIB1.INC
;
;-------------------------------------------------------------------------
;
;  MEMORY MAP FOR CNTRL BOARD
;  --------------------------
;
;  FFF0 - FFFF  =  Interrupt Table
;  FE00 - FFEF  =  Constant Data Area ( Messages etc.)
;  FD60 - FDFF  =  Interupt Routines
;  F960 - FD5F  =  Subroutines for main program
;  F800 - F95F  =  Program
;
;
;  1000 - 2000  =  Variable Data Area
;  0800 - 0FFF  =  User Stack
;  0000 - 07FF  =  System Stack
;
;-------------------------------------------------------------------------
;
; GENERAL CONSTANTS
; -----------------
;
INT_TABLE:      EQU  0FFF6h     ; Addr of int table
START_ADDR:     EQU  0F800h     ; Start Of 2716 EPROM
CONST_ADDR:     EQU  0FE00h     ; Addr of Constants
VAR_ADDR:       EQU  01000h     ; Addr of variables
FIRQ_ADDR:      EQU  0FDF0h     ; Fast Int proc addr
IRQ_ADDR:       EQU  00000h     ; Not used
SWI_ADDR:       EQU  00000h     ; Not used
NMI_ADDR:       EQU  0FD60h     ; NMI Addr
PROC_ADDR:      EQU  0F960h     ; Proc Start Addr
;
STACK:          EQU  007FFh     ; Hardware Stack address
USER_STACK:     EQU  00FFFh     ; User stack address
;
EN_FIRQ:        EQU  0BFh       ; Enable fast interrupt
DIS_FIRQ:       EQU  040h       ; Disable fast interrupt
;
ORG CONST_ADDR
;
KBD_TABLE:      DFB 0D8h,0C3h,0C8h,0D3h,0D8h,0C0h,0C8h,0D0h,
                    0D9h,0C1h,0C9h,0D1h,0D2h,0CAh,0C2h,0DAh
; Keyboard look up table
;
; Screen Messages
;
MSG1:           DFB "Station Number ? $"
MSG2:           DFB 'Sampling Rate ? $"
```

```
MSG3:           DFB "Input Year : $"
MSG4:           DFB "Input Month : $"
MSG5:           DFB "Input Day : $"
MSG6:           DFB "Input Hour : $"
MSG7:           DFB "Input Minutes : $"
MSG8:           DFB "Press any key : $"
MSG9a:          DFB "Press any key to $"
MSG9b:          DFB "start sampling $"
;
;-------------------------------------------------------------------------
;
; DISPLAY CONSTANTS
; -----------------
;
DISPLAY:        EQU  04001h       ; Display port addr
DISP_CTRL:      EQU  04000h       ; Display control addr
;
CLS:            EQU  001h         ; Clear the display
HOME:           EQU  002h         ; Sets cursor to home
;
DDRAM:          EQU  080h         ; Internal ASCII set
;
INIT_VAL:       EQU  030h         ; Software init value
;
; Display and Cursor Controls
;
DSP_ON:         EQU  00Ch
CUR_ON:         EQU  00Ah
BLNK_ON:        EQU  009h
DSP_OFF:        EQU  008h
CUR_OFF:        EQU  008h
BLNK_OFF:       EQU  003h
;
; Defined Cursor types : 1   = Blinking Cursor
;                        2   = Standard Cursor
;                        No  = No Cursor
;
CURSOR_TYPE1:   EQU  (DSP_ON) OR (CUR_ON) OR (BLNK_ON)
CURSOR_TYPE2:   EQU  (DSP_ON) OR (CUR_ON) OR (BLNK_OFF)
NO_CURSOR:      EQU  (DSP_ON) OR (CUR_OFF) OR (BLNK_OFF)
DISPLAY_OFF:    EQU  DSP_OFF
;
; Entry Mode Set
; -------- -----
;
INC_SHIFT:      EQU  007h         ; Inc and shift disp
DEC_SHIFT:      EQU  005h         ; Dec and shift disp
INC_NO_SHIFT:   EQU  006h         ; Increment without shift
DEC_NO_SHIFT:   EQU  004h         ; Decrement without shift
;
;
; Cursor Display Shift
;
CURSOR_R:       EQU  014h
```

```
CURSOR_L:       EQU  010h
DISPLAY_R:      EQU  01Ch
DISPLAY_L:      EQU  018h
;
;
; Function Set
;
; (no. of bits) (duty cycle) (dots in LCD matrix)
;
SET_FUNC1:      EQU  03Ch          ; (8), (1/16), (5*10)
SET_FUNC2:      EQU  038h          ; (8), (1/16), (5*7 )
;
;
; KEYBOARD CONSTANTS
;
FIRQ:           EQU  0BFh          ; FIRQ sel for CWAI
;
KEYBOARD:       EQU  02001h        ; Keyboard Read Addr
RESET_KBD:      EQU  0A001h        ; Reset keyboard interrupt
;
A_KEY:          EQU  0FAh          ; Val of alpha-chars
B_KEY:          EQU  0FBh
C_KEY:          EQU  0FCh
D_KEY:          EQU  0FDh
E_KEY:          EQU  0FEh
F_KEY:          EQU  0FFh
;
;-------------------------------------------------------------------------------
;
; RTC CONSTANTS
; -------------
;
RTC:            EQU  08000h     ; Memory mapped address for RTC
RTC1:           EQU  08001h
;
;
; RTC registers
;
SEC_REG:        EQU  000h
MIN_REG:        EQU  002h
HRS_REG:        EQU  004h
DAY_REG:        EQU  007h
MTH_REG:        EQU  00Bh
YRS_REG:        EQU  009h
REG_A:          EQU  00Ah
REG_B:          EQU  00Bh
REG C:          EQU  00Ch
REG_D:          EQU  00Dh
;
EN_RTC:         EQU  00Ah
DIS_RTC:        EQU  08Ah
;
RTC_SEL_32HZ:   EQU  00Bh
RTC_SEL_256HZ:  EQU  008h
```

```
;
STR_32:         DFB  "32 Hz  $"
STR_256:        DFB  "256 Hz $"
;
TIME_MASK:      DFB "00:00:00$"
DATE_MASK:      DFB "00-00-00$"
;
;-------------------------------------------------------------------------
;
; I/O PORT CONSTANTS
; ------------------
;
;
; Memory mapped Port address (Comms port - comms with other board)
;                            (IO port - control of anti-alias filter,
;                             data direction, time update, start
;                             sampling)
;
COMMS_PORT:     EQU  06000h
COMMS_CONTROL:  EQU  06001h
IO_PORT:        EQU  06002h
IO_CONTROL:     EQU  06003h
;
;
; Select data direction on IO_PORT
;
DAC_MEM:        EQU  10000000b    ; DAC to MEM
CTRL_DAC:       EQU  00100000b    ; CNTRL to DAC
CTRL_MEM:       EQU  01000000b    ; CNTRL to MEM
CLEAR_TFR:      EQU  00011111b    ; All tri-state
;
;
; Set anti-alias filter on IO_PORT
;
AA_32_SEL:      EQU  00001000b
AA_128_SEL:     EQU  00010000b
;
;
; Time update bit on IO_PORT
;
TME_UPDTE:      EQU  00000100b
;
;
; Start sampling line on IO_PORT
;
SMPL_LINE_ON:   EQU  00000001b
SMPL_LINE_OFF:  EQU  11111110b
;
;
; I/O modes for I/O ports
;
IO_MASK:        EQU  11111101b    ; one input 7 outputs
ALL_OUTPUT:     EQU  0FFh
ALL_INPUT:      EQU  000h
```

```
NO_HSHAKE:      EQU   004h
OUT_HSHAKE:     EQU   024h        ; Use CA/CB lines for handshake
IN_HSHAKE:      EQU   02Dh
;
DDR_SET:        EQU   000h        ; Data direction set
;
;
CHECK_STATUS:   EQU   080h
;
;
;-------------------------------------------------------------------------
;
ORG   VAR_ADDR
;
; GENERAL VARIABLES
; -----------------
;
sta_num:          DFS 1
sample_rate:      DFS 1
year:             DFS 1
month:            DFS 1
day:              DFS 1
hour:             DFS 1
min:              DFS 1
sec:              DFS 1
old_hour          DFS 1
temp_val:         DFS 1
time:             DFS 9
date:             DFS 9
latch_tme_dta:    DFS 6
old_sec:          DFS 1
;
; DISPLAY VARIABLES
; -----------------
;
disp_data:        DFS  1
dec_num:          DFS  1
;
;
; KEYBOARD VARIABLES
; ------------------
;
;
keypressed:       DFS  1
;
;
; I/O PORT VARIABLES
; ------------------
;
nmi_count:        DFS  1
nmi_flag:         DFS  1
io_reg:           DFS  1
;
;-------------------------------------------------------------------------
```

```
CPU "6809.tbl"
HOF "int16"
;
; {-------------------------------------------------------}
; { Program CONTROL                                       }
; {                                                       }
; { This program is written using the X16 cross assembler.}
; { The program controls the input of data from the       }
; { keyboard, the RTC, the LCD display and data direction }
; { and transfer from the three uP boards DAC, CNTRL and  }
; { MEM.                                                  }
; {                                                       }
; {-------------------------------------------------------}
;
; (Include constant and variable declarations)
;
INCL "LIB1.DCL"
;
; (Setup interrupt table)
;
ORG INT_TABLE
            DWM FIRQ_ADDR       ; FIRQ Procedure Start Address
            DWM IRQ_ADDR        ; IRQ Procedure Start Address (not used)
            DWM SWI_ADDR        ; SWI Procedure Start Address (not used)
            DWM NMI_ADDR        ; NMI Procedure Start Address
            DWM START_ADDR      ; 2716 Eprom Start Addr
;
;-------------------------------------------------------------------
;
; Procedure Read_Keyboard (FIRQ)
;
; {-------------------------------------------------------}
; {  Procedure Read_Keyboard (FIRQ)                       }
; {                                                       }
; {  Fast Interrupt request procedure - The interrupt is  }
; {  reset and no further action taken. All keyboard      }
; {  interaction is done using CWAI operations - (wait    }
; {  for interrupt operations)                            }
; {-------------------------------------------------------}
;
ORG FIRQ_ADDR
;
            LDA RESET_KBD
;           (Reset Interrupt)
            RTI
;           (Return from interrupt)
;
;-------------------------------------------------------------------
;
; {-------------------------------------------------------}
; { Procedure Latch_Transfer_Time (NMI)                   }
; {                                                       }
; { Non-maskable interrupt procedure, NMI on uP Reset is  }
; { ignored. First Reset latches the Time, the Second     }
```

```
; { transfers the data                                              }
; {                                                                 }
; {-----------------------------------------------------------------}
;
ORG NMI_ADDR
;
;  (Check nmi_flag to see if NMI is due to reset ie. ignore first
;   NMI after a reset)

                LDA  nmi_flag
                CMPA #0FFh
;  (Check nmi_flag)
;
                BEQ  START_NMI
;  (If nmi_flag Then Branch to START_NMI)
;
                LDA  #0FFh
                STA  nmi_flag
;  (Else nmi_flag := True)
;
                RTI
;  (Return from interrupt)
;
START_NMI:
; (Latch time on first NMI, Transfer Time on the second NMI)
;
                LDA  nmi_count
                CMPA #00h
; (Check if first or second NMI)
;
                BNE  XFER
; (If second NMI branch to XFER)

                JSR  Latch_Time
; (Call procedure Latch_Time)
;
                INCA
                STA  nmi_count
                BRA  RET
; (Set nmi_count and Return)
;
XFER:
                JSR  Init_Ctrl_Mem
; (Setup transfer Ctrl-Dac)
;
                JSR  Transfer_Dta
; (Call Procedure Transfer_Dta

                LDA  #00h
                STA  nmi_count
; ( Reset nmi_count )
;
                JSR  INIT_DAC_MEM
; (Setup transfer Dac-Mem)
```

```
;
RET:            RTI
; (Return from interrupt)
;
;
;
;-------------------------------------------------------------------------
;
; Include the library Procedures LIB1.INC
;
ORG PROC_ADDR
INCL "LIB1.INC"
;
;
; BEGIN (Main Program)
;
ORG START_ADDR
;
                LDS   #STACK
                LDU   #USER_STACK
; (Setup hardware and user stacks)
;
                LDA   #00h
                STA   nmi_flag
                STA   nmi_count
; (Initialise variables)
;
                JSR   Init_Ports
                JSR   Init_Display
; (Call initialisation procdures for Ports and Display)
;
                ANDCC #EN_FIRQ
; (Enable Fast interrupt requests)

                LDX   #MSG1
                JSR   Write_Line
; (Write MSG1 - Input station number to display)
;
                LDX   #sta_num
                JSR   Read_String
; (Read two digit variable from keyboard into sta_num)
;
                JSR   Clear
                JSR   Cur_Home
; (Clear Screen and Cursor Home)
;
                LDX   #MSG2
                JSR   Write_Line
; ( Write MSG2 - Input sampling Rate)
;
GET_SAMPLE_RATE:
;
                LDX   #STR_32
                JSR   Write_String
```

```
; (Write Message - 32 Hz )
;
                LDB   #RTC_SEL_32HZ
; (B_Reg := 32 Hz selection for RTC)
;
                JSR   Read_Char
                LDA   keypressed
                CMPA #":"
; (If keyboard character pressed = A then selected Else continue)
                BEQ   SELECTED
;
                LDA   #07h
                STA   dec_num
                JSR   Dec_Display
; (Decrement the display by 7 characters - length of message)
;
                LDX   #STR_256
                JSR   Write_String
; (Write message - 256 Hz)
;
                LDB   #RTC_SEL_256HZ
; (B_Reg := 256 Hz selection for RTC)
;
                JSR   Read_Char
                LDA   keypressed
                CMPA #":"
; (If keyboard character pressed = A then selected Else continue)
;
                BEQ   SELECTED
;
                LDA   #07h
                STA   dec_num
                JSR   Dec_Display
; (Decrement the display by 7 characters - length of message)
;
                BRA   GET_SAMPLE_RATE
; (Repeat Until keypressed = A)
;
SELECTED:
                STB   sample_rate
; (Store selected sample rate)
;
                JSR   Sel_Filter
; (Select the anti-alias filter)
;
                JSR   Init_Ctrl_Dac
                LDA   sample_rate
                JSR   Xfer_AReg
; (Transfer sample_rate to DAC-board)
;
                JSR   Init_Dac_Mem
; (Setup data transfer DAC_MEM)
```

```
;
                JSR   Clear
                JSR   Cur_Home
; (Clear Display)
;
                LDA   #MSG3
                JSR   Write_Line
                LDX   #year
                JSR   Read_String
; (Input Year)
;
                JSR   Clear
                JSR   Cur_Home
                LDX   #MSG4
                JSR   Write_Line
                LDX   #month
                JSR   Read_String
; (Input Month)
;
                JSR   Clear
                JSR   Cur_Home
                LDX   #MSG5
                JSR   Write_Line
                LDX   #day
                JSR   Read_String
; (Input day)
;
                JSR   Clear
                JSR   Cur_Home
                LDX   #MSG6
                JSR   Write_Line
                LDX   #hour
                JSR   Read_String
; (Input hour)
;
                JSR   Clear
                JSR   Cur_Home
                LDX   #MSG7
                JSR   Write_Line
                LDX   #min
                JSR   Read_String
; (Input Minutes)
;
                JSR   Clear
                JSR   Cur_Home
                LDX   #MSG8
                JSR   Write_Line
                JSR   Read_Char
; (Start Clock when key is pressed with selected time)
;
                JSR   Init_Clock
; (Initialise RTC)
;
                JSR   Set_Sel
```

```
; (Set selected time and date)
;
                JSR  Start_Clock
; (Start the RTC running)
;
                JSR  Clear
                JSR  Cur_Home
                LDX  #MSG9a
                JSR  Write_Line
                LDX  #MSG9b
                JSR  Write_Line
; Start sampling after keypressed and transition of a second
;
                JSR  READ_CHAR
;
                LDA  #SEC_REG
                JSR  RTC_READ
                STB  old_sec
; (Read seconds from RTC and store in old_sec)
;
START_SEC:      JSR  RTC_READ
                CMPB old_sec
                BEQ  START_SEC
; (Read seconds from RTC and compare with old_sec)
; (Continue on transition of a second)
;
                LDA  io_reg
                ORA  #SMPL_LINE_ON
                STA  io_reg
                STA  IO_PORT
; (Or SMPL_LINE_ON with the io_reg (input/output register) To
;  enable the A/D)
;
              JSR  Clear
              JSR  Cur_Home
              LDA  #NO_CURSOR
              STA  DISP_CTRL
; (Clear Display and hide cursor)
;
; (Main loop checks for time updates, and writes time data to displa
;
MAIN_LOOP:
            JSR  Check_Update
; (Call Check_Update Procedure to check for time sync pulses from
;  the time-sync card)
;
            LDX  #year
            JSR  Read_Time
            JSR  Write_Time
            JSR  Write_Date
; (Read time & date and output to display)
;
            BRA  MAIN_LOOP
; (Repeat MAIN_LOOP forever)
```

```
;
END
;
```

```
;-------------------------------------------------------------------------------
;
;  LIB1.INC
;
;  Included are :
;  - Display Routines for the 6809 uP for driving the
;    Optrex LCD Display DMC16249.
;  - Routines for Reading the Keyboard
;  - Routines for the the MC146818 real time clock
;
;  The file LIB1.DCL, constant and variable
;  declarations for the routines given here must also
;  be included in the source file.
;
;-------------------------------------------------------------------------------
;
;
;
;===============================================================================
;
; Description of Display Procedures
; ---------------------------------
;
; Procdure Init_Display
;    { Performs the necessary software initialisation
;      for the OPTREX display }
;
; Procedure Clear
;    { Clears the display   =  JSR Clear }
;
; Procedure Cur_Home
;    { Moves Cursor to home position  =  JSR Home }
;
; Procedure Dec_Display
;    { The display is deleted from the cursor
;      by the ammount specified in the variable
;      dec_num }
;
; Procedure Write_String
;    { The address of the start of the string is
;    loaded into the X register, and the procedure
;    called. The string must be
;      terminated with a $ }
;
; Procedure Write_Line
;    { The address of the start of the string is
;    loaded into the X register, and the procedure
;    called. The string must be terminated with a $.
;    The program inserts up to 40 blanks }
;
; Procedure Delay1
;  { 'Long delay'}
;
; Procedure Delay2
```

```
;  {'Short delay'}                                                    ;
;                                                                     ;
;=====================================================================;
;_____________________________________________________________________
;
;Procedure Init_Display
;
Init_Display:
;
; Software Init procedure - See OPTREX Manual
;
                JSR   Delay1
                LDA   #INIT_VAL   ; [DISP_CTRL] .= INIT_VAL
                STA   DISP_CTRL   ; Wait more than 4.1mS
                JSR   Delay1
;
                LDA   #INIT_VAL   ; [DISP_CTRL] := INIT_VAL
                STA   DISP_CTRL   ; Wait more than 100uS
                JSR   Delay1
;
                LDA   #INIT_VAL   ; [DISP_CTRL] := INIT_VAL
                STA   DISP_CTRL   ; Wait more than 100uS
                JSR   Delay1
;
                LDA   #SET_FUNC2  ; 8, 1/16 duty, 5*10 dots
                STA   DISP_CTRL
                JSR   Delay1
;
                LDA   #CUR_OFF        ; Switch Cursor Off
                STA   DISP_CTRL
                JSR   Delay2
;
                JSR   CLEAR               ; Clear Display
;
                LDA   #INC_NO_SHIFT       ; Inc no shift
                STA   DISP_CTRL
                JSR   Delay2
;
; * End of Software Initialisation procedure *
;
                LDA   #DDRAM              ; Use internal ASCII
                STA   DISP_CTRL
                JSR   Delay2
;
                JSR   CLEAR               ; Clear Display
;
                LDA   #INC_NO_SHIFT       ; Inc and shift
                STA   DISP_CTRL
                JSR   Delay2
;
                JSR   CUR_HOME            ; Cursor Home
;
                LDA   #CURSOR_TYPE1       ; Sel flashing
                STA   DISP_CTRL
```

```
                JSR  Delay2
;
                RTS                     ; Return
;
;-------------------------------------------------------------------------
;
; Procedure Clear
;
Clear:          PSHU A
                LDA  #CLS               ; A_Reg := CLS
                STA  DISP_CTRL          ; [DISP_CTRL] := A_Reg
                JSR  Delay2
                PULU A
                RTS                     ; Return
;
;   End  { Clear }
;
;-------------------------------------------------------------------------
;
; Procedure Cur_Home
;
CUR_HOME:       PSHU A
                LDA  #HOME              ; A_Reg := HOME
                STA  DISP_CTRL          ; [DISP_CTRL] := A_Reg
                JSR  Delay2
        PULU A
                RTS                     ; Return
;
;   End { Home }
;
;-------------------------------------------------------------------------
;
; Procedure Dec_Display
;
Dec_Display:
                PSHU B
;
                LDA  #DEC_NO_SHIFT
                STA  DISP_CTRL                  ; Set Display for decrement
                JSR  Delay2
                LDA  #CURSOR_L                  ; Set cursor left
                LDB  dec_num                    ; B_Reg = dec_num
DELETE_CHAR:
                STA  DISP_CTRL                  ; backspace
                JSR  Delay2
                DECB                            ; dec_num := dec_num - 1
                CMPB #00h
                BNE  DELETE_CHAR                ; Until dec_num = 0
;
                LDA  #INC_NO_SHIFT
                STA  DISP_CTRL                  ; Reset display increment
                JSR  Delay2
;
                PULU B
```

```
                RTS                         ; Return
;
;-------------------------------------------------------------------------
;
; Procedure Write_String
;
Write_String:   PSHU A
                LDA   ,X+                   ; (A_Reg) := [(Reg_X)]
                                            ; (Reg_X) := (Reg_X) + 1
                CMPA #"$"                   ; ((Reg_A) - "$")
                BEQ   END_W_ST              ; (If zero : END_W_S,)
                STA   DISPLAY               ; [DISPLAY] := (Reg_A)
                JSR   Delay2                ; Call Delay2
                BRA   Write_String
END_W_ST:
                PULU A
                RTS
;   End
;-------------------------------------------------------------------------
;
; Procedure Write_Line
;
Write_Line:     PSHU A
                PSHU B

                LDB   #00 .
AAA:            LDA   ,X+                   ; A Reg := [Reg_X]  (inc X_Reg)
                CMPA #"$"                   ; C .eck for end of string
                BEQ   CHECK_LINE_LEN        ; Branch if end of string
                STA   DISPLAY               ; Write char. :ter to display
                JSR   Delay2
                INCB
                BRA   AAA
CHECK_LINE_LEN:
                LDA   #" "          ; fill with blanks
                STA   DISPLAY
                JSR   Delay2
                CMPB #27h           ; Leng.h of line (40 charcters)
                BEQ   END_WR_LINE
                INCB
                BRA   CHECK_LINE_LEN
END_WR_LINE:
                PULU B
                PLLU A
                RTS                 ; Return
;
;       End;
;
;-------------------------------------------------------------------------
;
; Procedure Delay1
;
Delay1:
                PSHU A
```

```
                PSHU B
                LDA  #00h
LOOP2:
                LDB  #00h
LOOP1:          INCB
                CMPB #040h
                BNE LOOP1
                INCA
                CMPA #0FFh
                BNE LOOP2
                PULU B
                PULU A
                RTS
;
;   End
;
;-------------------------------------------------------------------------
;
DELAY2:
                PSHU A
                LDA #00h
LOOP3:           INCA
                CMPA #0FFh
                BNE LOOP3
                PULU A
                RTS
;
;-------------------------------------------------------------------------
;
;=========================================== ==================
;
;  Description of Keyboard Procedures
;  ----------------------------------
;
; Procedure Read_and_Echo
;    { Procedure waits for a key to be pressed, and
;    echoes the character to the display. The lower
;    nybble of the variable key_pressed contains the
;    hex value of the input key. The upper nybble
;    should be anded out for safety sake }
;
; Procedure Read_Char
;    { Procedure waits for a key to be pressed, and
;    the value written to key_pressed as in
;    Read_and_Echo. The value is not however echoed
;    to the display }
;
; Procedure Read_String
;    { Reads two digit number from the keyboard and
;      stores them in a one byte variable noited to
;      by the X_Reg in BCD format }
;
;=============================================================
;
```

```
;_____________________________________________________________________
;
; Procedure Read_And_Echo
;
Read_And_Echo:
                PSHU A
                PSHU B
                PSHU X

                CWAI #FIRQ              ; Wait for Fast Interrupt
                LDA  KEYBOARD           ; Read keyboard latch
                LDX  #KBD_TABLE
                LDA  #00h
NEXT:
                LDB  ,X+
                CMPB KEYBOARD           ; Compare chacter with
                BEQ  FOUND              ; Lookup table
                INCA
                BRA  NEXT
FOUND:
                ADDA #30h               ; Make value Ascii
                STA  keypressed         ; store value in keypressed

                STA  DISPLAY            ; Write character to display
                JSR  Delay2
                PULU X
                PULU B
                PULU A
                RTS                     ; Return
;
; End (Read_and_Echo)
;_____________________________________________________________________
;
; Procedure Read_Char
;
Read_Char:      PSHU A
                PSHU B
                PSHU X

                CWAI #FIRQ              ; Wait for interrupt
                LDA  KEYBOARD           ; Read keyboard latch
                LDX  #KBD_TABLE
                LDA  #00h
NEXT1:
                LDB  ,X+                ; Compare value with lookup table
                CMPB KEYBOARD
                BEQ  FOUND1
                INCA
                BRA  NEXT1
FOUND1:
                ADDA #30h
                STA  keypressed         ; Make ascii and store value

                PULU X
```

```
                PULU B
                PULU A
                RTS                     ; Return

;
; End {Read_Char}
;
;-------------------------------------------------------------------------
;
; Procedure Read_String
;
Read_String:
                JSR  Read_and_Echo      ; Read Keyboard
                LDA  KeyPressed         ; A_Reg := Keypressed
                JSR  Read_and_Echo      ; Read Keyboard
                LDB  KeyPressed         ; B_Reg := Keypressed
                SUBA #30h
                SUBB #30h               ; Make Ascii
                ROLA
                ROLA                    ; Shift value in A_Reg to
                ROLA                    ; high byte
                ROLA
                STA  temp_val
                ADDB temp_val           ; Make BCD value of the two byte
                STB ,X+                 ; Write to variable [X_Reg]
                JSR  DELAY1
                RTS                     ; Return
;
; End {Read_String}
;-------------------------------------------------------------------------
;=========================================================;
;                                                         ;
; Description of Real Time Clock (RTC) Procedures         ;
;---------------------------------------------------      ;
;                                                         ;
; Procedure RTC_Write                                     ;
;   { Procedure to write data to the RTC. Reg_A           ;
;   must be loaded with the address of the RTC            ;
;   register, Reg_B with the data }                       ;
;                                                         ;
; Procedure RTC_Read                                      ;
;    { Procedure to read data from the RTC. Reg_A         ;
;    must be loaded with the address of the RTC           ;
;    register, Reg_B will contain the data }              ;
;                                                         ;
; Procedure Init_Clock                                    ;
;    { Initialise 24 hour, BCD format, &                  ;
;       sampling rate }                                   ;
;                                                         ;
; Procedure Start_Clock                                   ;
;    { Enable start bit }                                 ;
;                                                         ;
; Procedure Set_Sel                                       ;
;    { Set All the time & date regiters to selected       ;
```

```
;     values }
;
; Procedure Read_Time
;    { Read Time into Time and date variables }
;
; Procedure Update_Time
;    { Update the time variables (year.... seconds)
;      with the present time from RTC }
;
; Procedure Write_Time
;    { Write time to display }
;
; Procedure Write_Date
;    { Write date to display }
;
; Procedure Split_Write
;   {The one byte BCD variable passed in A_Reg is
;    split into two ascii characters and the two
;    ascii characters are stored in the variable
;    pointed to by X_Reg }
;
; Procedure Latch_Time
;   { The date and time data is read into the
;     variable latch_tme_dta }
;
; Procedure Check_Update
;  { Procedure checks a 10 second window about the
;    hour for a sync signal, resets the minutes and
;    seconds counters, and updates the hour if
;    necessary. (The procedure ignores any updates
;    around 00h00.) }
;
; Procedure Set_Hour
;   {used by Check_Update to reset minutes and
;    seconds and to update the hour }
;
;=============================================================
;
;-------------------------------------------------------------
;
; Procedure RTC_ Read
;
RTC_Read:  STA  RTC                    ; RTC Register in A_Reg
           LDB RTC1                    ; data in B_reg
           RTS
;
; End {RTC_Read}
;
;-------------------------------------------------------------
;
; Procedure RTC_Write
;
RTC_Write: STA  RTC                    ; RTC Register in A_Reg
           STB RTC1                    ; data in B_Reg
```

```
            RTS
;
; End (RTC_Write)
;
;-------------------------------------------------------------------------
;
; Procedure Init_Clock
;
Init_Clock:
        LDX #TIME_MASK
        LDY #TIME
HHH:    LDA ,X+                     ; Write the time mask to variable time
        STA ,Y+
        CMPA #"$"
        BNE HHH

        LDX #DATE_MASK
        LDY #date
HOH:    LDA ,X+                     ; Write the date mask to variable date
        STA ,Y+
        CMPA #"$"
        BNE HOH
;
        LDB  sample_rate
        LDA  #REG_A
        JSR  RTC_Write              ; Select the sample rate
        LDA  #REG_B
        LDB  #DIS_RTC
        JSR  RTC_Write              ; RTC disabled
        RTS                         ; Return
;
; End (Init_Clock)
;
;-------------------------------------------------------------------------
;
; Procedure Start_Clock
;
START_CLOCK:
                LDA  #REG_B
                LDB  #EN_RTC
                JSR  RTC_Write          ; Enable RTC
                RTS
;
; End (Start Clock)
;
;-------------------------------------------------------------------------
;
; Procedure Set_Sel
;
SET_SEL:
                LDA  #YRS_REG
                LDB  year
                JSR  RTC_Write              ; Write years to RTC
;
```

```
                LDA   #MTH_REG
                LDB   month
                JSR   RTC_Write             ; Write month to RTC
;
                LDA   #DAY_REG
                LDB   day
                JSR   RTC_Write             ; Write days to RTC
;
                LDA   #HRS_REG
                LDB   hour
                JSR   RTC_Write             ; Write hours to RTC
;
                LDA   #MIN_REG
                LDB   min
                JSR   RTC_Write             ; Write minutes to RTC
;
                LDA   #SEC_REG
                LDB   #00h
                JSR   RTC_Write             ; Write seconds to RTC
;
                RTS                         ; Return
;
; End {Set_Sel}
;
;-------------------------------------------------------------------------------
;
; Procedure Read_Time
;
; Updates the variable pointed to by X_Reg
;
Read_Time:
                PSHU A
                PSHU B
;
                LDA   #YRS_REG
                JSR   Updte_Time
;
                LDA   #MTH_REG
                JSR   Updte_Time
;
                LDA   #DAY_REG
                JSR   Updte_Time
;
                LDA   #HRS_REG
                JSR   Updte_Time
;
                LDA   #MIN_REG
                JSR   Updte_Time
;
                LDA   #SEC_REG
                JSR   Updte_Time
;
                PULU B
                PULU A
```

```
            RTS
;
; End {Read_Time}
;
;-------------------------------------------------------------------------
;
; Procedure Updte_Time
;
Updte_Time:
;
          JSR  RTC_Read
          STB  ,X+
          RTS
;
; End {Updte_Time}
;
;-------------------------------------------------------------------------
;
; Procedure Write_Time
;
Write_Time:
          PSHU A
          PSHU B
          PSHU X
          LDX  #time
          LDA  hour
          JSR  Split_Write        ; Write hour in ASCII to time
          LDA  ,X+
          LDA  min
          JSR  Split_Write        ; Write minutes in ASCII to time
          LDA  ,X+
          LDA  sec
          JSR  Split_Write        ; Write seconds in ASCII to time
          LDX  #time
          JSR  WRITE_LINE         ; Write time to display
          PULU X
          PULU B
          PULU A
          RTS                     ; Return
;
; End {Write_Time}
;
;-------------------------------------------------------------------------
;
; Procedure Write_Date
;
;  Same funtion as for Write_Time (See above)
;
Write_Date:
          PSHU A
          PSHU B
          PSHU X
          LDX  #date
          LDA  year
```

```
            JSR   Split_Write
            LDA   ,X+
            LDA   month
            JSR   Split_Write
            LDA   ,X+
            LDA   day
            JSR   Split_Write
            LDX   #date
            JSR   WRITE_LINE
            PULU  X
            PULU  B
            PULU  A
            RTS
;
; End {Write_Date}
;-------------------------------------------------------------------------
;
;Procedure Split_Write
;
Split_Write:
                  TFR   A,B               ; Split byte into two variables
                  ANDB  #00Fh             ; and make into BCD
                  ADDB  #30h
                  ANDA  #0F0h
                  RORA
                  RORA
                  RORA
                  RORA
                  ADDA  #30h
                  STA   ,X+               ; Store ASCII values in [X_Reg]
                  STB   ,X+               ; and [X_Reg + 1]
                  RTS                     ; Return
;
; End {Split_Write}
;-------------------------------------------------------------------------
;
; Procedure Latch_Time
;
Latch_Time:
                  LDX   #latch_tme_dta
                  JSR   Read_Time
                  RTS
;
; End {Latch_Time}
;-------------------------------------------------------------------------
;
; Procedure Check_Update
;
Check_Update:
                LDX   #year
                JSR   Read_Time
                LDA   hour
                CMPA  #23h
                BEQ   UPDATE_RET          ; If time = 23h00 then Return
```

```
                STA   old_hour
                LDA   min
                CMPA  #59h
                BNE   UPDATE_RET
                LDA   sec
                CMPA  #55h
                BLT   UPDATE_RET
CHECK:                                  ; If time = ??h59m55s Then continu
                LDA   #SEC_REG
                JSR   RTC_Read
                CMPB  #06               ; Wait for 10 seconds then Return
                BEQ   UPDATE_RET
                LDA   IO_PORT
                ANDA  #TME_UPDTE
                CMPA  #TME_UPDTE        ; Check port for update
                BEQ   CHECK
                LDA   io_reg            ; If update Then
                ANDA  #11111011b        ;   toggle line to NMI on DAC
                STA   IO_PORT
                JSR   Set_Hour          ; Set hours, mins, secs
                LDA   io_reg
                ORA   #00000100b
                STA   IO_PORT
UPDATE_RET:
                RTS                     ; Return
;
; End {Check_Update}
;-------------------------------------------------------------------------
;
; Procedure Set_Hour
;
Set_Hour:
                LDA   #MIN_REG           ; Set hours, minutes, seconds
                LDB   #00h
                JSR   RTC_Write
                LDA   #SEC_REG
                LDB   #00h
                JSR   RTC_Write
                LDA   #HRS_REG
                LDB   old_hour
                INCB
                JSR   RTC_Write
                RTS
;
; End {Set_Hour}
;
;-------------------------------------------------------------------------
;
;  ==============================================;
;                                                 ;
; Description of I/O Port Procedures              ;
;-------------------------------                  ;
;                                                 ;
; Procedure Init_Ports                            ;
```

```
;    { Setup I/O and control ports }                                    ;
;                                                                       ;
; Procedure Sel_Filter                                                  ;
;    { Setup anti-alias filter depending on selected                    ;
;      sampling rate }                                                  ;
;                                                                       ;
; Procedure Transfer_Dta                                                ;
;    { Transfer the latch_tme_dta to the MEM board }                    ;
;                                                                       ;
; Procedure Xfer_AReg                                                   ;
;    { Transfer the contents of A_Reg to comms port }                   ;
;                                                                       ;
; Procedure Init_Ctrl_Dac                                               ;
;    { Initialise data transfer CNTRL to DAC }                          ;
;                                                                       ;
; Procedure Init_Dac_Mem                                                ;
;    { Initialise data transfer DAC to MEM }                            ;
;                                                                       ;
; Procedure Init_Ctrl_Mem                                               ;
;    { Initialise data transfer CNTRL to MEM }                          ;
;                                                                       ;
; Procedure Init_Mem_Ctrl                                               ;
;    { Initialise data transfer MEM to CNTRL }                          ;
;                                                                       ;
;=======================================================================;

;-----------------------------------------------------------------------
;
; Procedure Init_Ports
;
Init_Ports:
            LDA #IO_MASK
            STA IO_PORT          ; Setup ; outputs 1 input - I/O port
            LDA #NO_HSHAKE
            STA IO_CONTROL       ; no handshake - I/O port
;
            LDA  #00000100b
            STA  io_reg          ; I/O register
            STA  IO_PORT
            RTS                  ; Return
;
; End {Init_Ports}
;-----------------------------------------------------------------------
;
; Procedure Sel_Filter
;
Sel_Filter:
            LDA  sample_rate
            CMPA #0Bh
            BNE  SET_128
            LDA  #AA_32_SEL
            ORA  io_reg
            BRA  FIL_SET
SET_128:
```

```
            LDA   #AA_128_SEL
            ORA   io_reg
FIL_SET:
            STA   IO_PORT          ; switch on aa_fil line
            LDB   #00h
            JSR   DELAY1
            LDA   #11100111b
            ANDA  io_reg
            STA   IO_PORT          ; switch off line
;
            RTS                    ; Return
;
; End {Sel_Filter}
;-------------------------------------------------------------------------
;
; Procedure Transfer_Dta
;
Transfer_Dta:
                LDA   sta_num
                JSR   Xfer_AReg          ; Transfer station number
                LDX   #latch_tme_dta
                LDB   #00h
CONT:                                    ; Transfer latch_tme_dta
                LDA   ,X+
                JSR   Xfer_AReg
                INCB
                CMPB  #06h
                BNE   CONT
                LDA   sample_rate
                JSR   Xfer_AReg          ; Transfer sample rate
;
                RTS                      ; Return
;
; End {Transfer_Dta}
;-------------------------------------------------------------------------
;
; Procedure Xfer_AReg
;
Xfer_AReg:
                STA   COMMS_PORT         ; Transfer A_Reg
                LDA   COMMS_PORT         ; Dummy read to
                                         ; toggle handshake
CHECK_TFR:
                LDA   COMMS_CONTROL
                ANDA  #CHECK_STATUS
                CMPA  #CHECK_STATUS      ; Check handshake line
                BNE   CHECK_TFR
;
                RTS                      ; Return
;
; End {Xfer AReg}
;-------------------------------------------------------------------------
;
; Procedure {Init_Ctrl_Dac}
```

```
;
Init_Ctrl_Dac:
                PSHU A
                LDA  io_reg
                ANDA #CLEAR_TFR
                ORA  #CTRL_DAC
                STA  io_reg
                STA  IO_PORT            ; hshake ctrl-dac
;
                LDA  #DDR_SET
                STA  COMMS_CONTROL
                LDA  #ALL_OUTPUT
                STA  COMMS_PORT
                LDA  #OUT_HSHAKE
                STA  COMMS_CONTROL
;
                PULU A
                RTS
;
; End {Init_Ctrl_Dac}
;-------------------------------------------------------------------------------
;
; Procedure Init_Dac_Mem
;
Init_Dac_Mem:
                PSHU A
                LDA  io_reg
                ANDA #CLEAR_TFR
                ORA  #DAC_MEM
                STA  io_reg
                STA  IO_PORT            ; hshake dac-mem
;
                LDA  #DDR_SET
                STA  COMMS_CONTROL
                LDA  #ALL_INPUT
                STA  COMMS_PORT
                LDA  #NO_HSHAKE
                STA  COMMS_CONTROL
;
                PULU A
                RTS
;
; End {Init_Dac_Mem}
;-------------------------------------------------------------------------------
;
; Procedure Init_Ctrl_Mem
;
Init_Ctrl_Mem:
                PSHU A
                LDA  io_reg
                ANDA #CLEAR_TFR
                ORA  #CTRL_MEM
                STA  io_reg
                STA  IO_PORT            ; hshake ctrl-mem
```

```
;
                LDA   #DDR_SET
                STA   COMMS_CONTROL
                LDA   #ALL_OUTPUT
                STA   COMMS_PORT
                LDA   #OUT_HSHAKE
                STA   COMMS_CONTROL
;
                PULU A
                RTS
;
; End {Init_Ctrl_Mem}
;-------------------------------------------------------------------------
;
; Procedure Init_Mem_Ctrl
;
;Init_Mem_Ctrl:
                PSHU A
                LDA   io_reg
                ANDA #CLEAR_TFR
                ORA   #CTRL_MEM
                STA   io_reg
                STA   IO_PORT           ; hshake mem-ctrl
;
                LDA   #DDR_SET
                STA   COMMS_CONTROL
                LDA   #ALL_INPUT
                STA   COMMS_PORT
                LDA   #IN_HSHAKE
                STA   COMMS_CONTROL
;
                PULU A
                RTS
;
; End {Init_Mem_Ctrl}
;-------------------------------------------------------------------------
```

DAC Program

```
;-------------------------------------------------------------------------
;
;  LIB2.DCL  {Global declarations for DAC software}
;
;  Constants and variables for routines in DAC and LIB2.INC
;
;-------------------------------------------------------------------------
;
;  MEMURY MAP FOR DAC BOARD
;  ------------------------
;
;  FFF0 - FFFF  =  Interrupt Table
;  FE00 - FFEF  =  Constant Data Area
;  FC00 - FDFF  =  Interupt Routines
;  FA00 - FBFF  =  Subroutines for main program
;  F800 - F9FF  =  Program
;
;
;  1000 - 2000  =  Variable Data Area
;  0800 - 0FFF  =  User Stack
;  0000 - 07FF  =  System Stack
;
;-------------------------------------------------------------------------
;
; GENERAL CONSTANTS
; -----------------
;
INT_TABLE:      EQU  0FFF0h     ; Addr of int table
START_ADDR:     EQU  0F800h     ; Start Of 2716 EPROM
CONST_ADDR:     EQU  0FE00h     ; Start Addr of Consts
VAR_ADDR:       EQU  01000h     ; Start Addr of Var
FIRQ_ADDR:      EQU  0FD00h     ; Fast Interrupt proc
IRQ_ADDR:       EQU  0FC00h     ; Interrupt proc
SWI_ADDR:       EQU  00000h
NMI_ADDR:       EQU  0FC80h     ; Time update
PROC_ADDR:      EQU  0FA00h     ; Procs Start Address
;
STACK:          EQU  007FFh     ; Hardware stack address
USER_STACK:     EQU  00FFFh     ; User stack address
;
EN_FIRQ:        EQU  0BFh       ; Enable fast int request
EN_IRQ:         EQU  0EFh       ; Enable int request
DIS_FIRQ:       EQU  040h       ; Disable FIRQ
DIS_IRQ:        EQU  010h       ; Disable IRQ
;
FIRQ:           EQU  0BFh         ; FIRQ sel for CWAT
IRQ:            EQU  0EFh
;
;
;
; I/O PORT CONSTANTS
; ------------------
;
; Memory maoped Port Address ( Comms port = comms with other boards
```

```
;                                   IO-port = misc control functions )
;
COMMS_PORT:     EQU  0CFFCh
COMMS_CONTRDL.: EQU  0CFFDh
IO_PORT:        EQU  0CFFEh
IO_CONTROL:     EQU  0CFFFh
;
; I/O modes for ports
;
ALL_OUTPUT:     EQU  0FFh
ALL_INPUT:      EQU  000h
NO_HSHAKE:      EQU  004h
OUT_HSHAKE:     EQU  024h
IN_HSHAKE:      EQU  02Dh
;
DDR_SET:        EQU  000h
;
CHECK_STATUS:   EQU  080h
;
;
; I/O Lines for paging memory (channels 1-3)
;
MEM_CH1:        EQU  020h OR NMI_OFF
MEM_CH2:        EQU  040h OR NMI_OFF
MEM_CH3:        EQU  080h OR NMI_OFF
;
;
; I/O line for NMI to CONTROL (latch/transfer time)
;
NMI_OFF:        EQU  010h
NMI_ON:         EQU  0EFh
;
;
;
;
BUF_FULL:       EQU  06FFFh
BUFOFS:         EQU  04000h
MAX_BUF:        EQU  07FFFh
;
AD_HB:          EQU  02001h
AD_LB:          EQU  02002h
RESET_INT:      EQU  02004h
;
;-------------------------------------------------------------------------------
;
ORG  VAR_ADDR
;
; GENERAL VARIABLES
; -----------------
;
sample_rate:    DFS  2
sample_count:   DFS  2
dragon_count:   DFS  2
x_counter:      DFS  2
```

```
y_counter:      DFS  2
bottom:         DFS  2
sel_chnl:       DFS  1
channel:        DFS  1
check_int:      DFS  1
top:            DFS  2
buf_flag:       DFS  1
num_loops:      DFS  1
temp:           DFS  1
;
; I/O PORT VARIABLES
; ------------------
;
io_reg:         DFS 1
;
;_____________________________________________________________________
```

```
CPU "6809.tbl"
HOF "int16"
;
; {-------------------------------------------------------------}
; { Program DAC                                                 }
; {                                                             }
; { This program is written for the X16 cross assembler. The    }
; { program reads three channels of data from the A/D and       }
; { stores them in three 32K buffers. Once the buffers are      }
; { the data is transferred to the MEMORY board.                }
; {                                                             }
; {-------------------------------------------------------------}
;
INCL "LIB2.DCL"
;
; (Setup Interrupt table)
;
ORG INT_TABLE
            DWM FIRQ_ADDR       ; FIRQ Procedure Start Address
            DWM IRQ_ADDR        ; IRQ procedure start address
            DWM SWI_ADDR        ; SWI start address (not used)
            DWM NMI_ADDR        ; NMI procedure start address
            DWM START_ADDR      ; 2716 Eprom Start Addr
;
;-------------------------------------------------------------------
;
; Procedure Comms
;
; {-------------------------------------------------------------}
; { Procedure Comms (IRQ)                                       }
; {                                                             }
; { The IRQ line together with the CA/CB lines on the 6321      }
; { I/O port are used for data transfer. See data sheet for     }
; { control of CA/CB lines under interrupt.                     }
; {                                                             }
; {-------------------------------------------------------------}
;
ORG IRQ_ADDR
;
            LDA  COMMS_PORT     ; A_Reg := Comms_Port
            STA  temp
;
            RTI                 ; Return
;
;-------------------------------------------------------------------
;
; Procedure Read_AD
;
; {-------------------------------------------------------------}
; { Procedure Read_AD (FIRQ)                                    }
; {                                                             }
; { When a FIRQ is received from the A/D board the data is      }
; { read into one of three memory buffers (buffers all at the   }
; { same address - and are paged)                               }
```

```
; {                                                                    }
; {--------------------------------------------------------------------}
;
ORG   FIRQ_ADDR
;
                PSHU A
                PSHU B
;
                LDA   num_loops     ; Case num_loops Of
                CMPA  #00h          ;  0 :(Reg_B) = MEM_CH1
                BNE   CHANNEL2      ;  1 :(Reg_B) = MEM_CH2
                LDB   #MEM_CH1      ;  3 :(Reg_B) = MEM_CH3
                BRA   go
;
CHANNEL2:
                CMPA  #01h
                BNE   CHANNEL3
                LDB   #MEM_CH2
                BRA   go
;
CHANNEL3:
                LDB   #MEM_CH3

GO:             STB    )_PORT       ; (IO_PORT) := (Reg_B)

                INCA
                STA   num_loops     ; num_loops=num_loops+1
;
                LDA   AD_HB           ; (A_Reg) := (AD_HB)
                LDB   AD_LB           ; (B_Reg) := (AD_LB)
                STD   BUFOFS,X        ; Buffer[(X_Reg)] :=
                                      ;             (Reg_D)
;
;
;
                LDA   num_loops
                CMPA  #03h
                BNE   RESTORE         ; If num_loops = 3
                                      ;  Then Begin
;                                     ;    num_loops := 0
;                                     ;    (X_Reg)=(X_Reg)+2
                LDA   #00h            ;    If (X_Reg) = 8000h
                                      ;     Then
                STA   num_loops       ;       (X_Reg) := 0;
;                                     ;   x_count=x_count+2
                LDB   #02h            ;    s_count=s_count+1
                ABX                   ;    If s_count =
;                                       sample_rate Then
                CMPX  #8000h          ;      sample_count := 0
                BNE   CCOUNT          ;    (IO_PORT)=sel_chnl
                LDX   #0000h          ;    * Reset Interupt *
;                                     ;  End; {RTI}
CCOUNT:         LDD   x_counter
                ADDD  #0002h
```

```
                STD   x_counter
;
                LDD   sample_count
                ADDD  #0001h
                STD   sample_count
                CMPD  sample_rate
                BNE   RESTORE
                LDD   #0000h
                STD   sample_count
;
;
RESTORE:
                LDA   sel_chnl
                STA   IO_PORT
;
                LDA   RESET_INT       ; Reset FIRQ interrupts
;
                PULU B
                PULU A
;
                RTI                   ; Return
;     End;
;
;-------------------------------------------------------------------------
;
; Procedure Reset_Dragon
;
; {-------------------------------------------------------------}
; { Procedure Reset_Dragon (NMI)                                }
; {                                                             }
; { The sample_count counter is reset on an NMI                 }
; {                                                             }
; {-------------------------------------------------------------}
;
ORG NMI_ADDR
;
        LDD   #00h
        STD   sample_count            ; Reset sample counter
        RTI
;
;-------------------------------------------------------------------------
;
ORG PROC_ADDR
;
; (Include procedures from LIB2.INC)
;
INCL "LIB2.INC"
;
; BEGIN (Main Program)
;
ORG START_ADDR

                LDS   #STACK
                LDU   #USER_STACK
```

```
; (Setup hardware and user stacks)
;
                JSR  Delay1
; (Wait for other boards to initialise)
;
                JSR  Init_Ports
                JSR  Delay2
; (Initialise I/O & Comms ports)

                LDA  #MEM_CH1
                STA  IO_PORT
                STA  sel_chnl
; (Setup memory buffer 1)
;
                JSR  Read_SRate
; (Get sample rate selected from CNTRL board)
;
                LDX  #0000h
                STX  x_counter
                STX  bottom
                STX  top
                STX  sample_count
                LDA  #00h
                STA  num_loops
; (Initialise counters)

;
                ANDCC #EN_FIRQ
; (Enable fast interrupt requests)
;
; ( The main loop waits until the buffers are full (each - 28K) then
;   transfers the buffers to the MEMORY card)
;
MAIN_LOOP:
                LDD  x_counter
                CMPD #7000h          ; Wait for 28K of data
                BNE  MAIN_LOOP
;
                STX  top
                LDD  #0000h
                STD  x_counter
;
                LDD  sample_count
                STD  dragon_count       ; Store counters
;
                JSR  Switch_NMI         ; Latch time on CNTRL board
;
                LDA  #00h
                STA  channel
;
                JSR  Init_Comms_Out
;
TXFER:
                LDA  channel                ; Setup memory buffer 1,2 or 3
```

```
                LDY   bottom
;
                CMPA  #00h
                BNE   MEM2
                LDB   #MEM_CH1
                BRA   SLCTED
;
MEM2:
                CMPA  #01h
                BNE   MEM3
                LDB   #MEM_CH2
                BRA   SLCTED
;
MEM3:
                LDB   #MEM_CH3
;
SLCTED:
                STB   sel_chnl
                STB   IO_PORT
;
TXLOOP:
                LDA   BUFOFS,Y
                JSR   Xfer_AReg             ; Transfer one byte of data
                TFR   Y,D
                ADDD  #0001h
                TFR   D,Y
;
                CMPY  #8000h
                BNE   CHECK_FINISHED
                LDY   #0000h
;
CHECK_FINISHED:                             ; Check if buffer is finished
                CMPY  top
                BNE   TXLOOP
                LDA   channel
                INCA
                STA   channel
                CMPA  #03h                  ; Check for 3 times
                BNE   TXFER
;
                LDD   dragon_count
                JSR   Xfer_AReg
                TFR   B,A
                JSR   Xfer_AReg             ; Transfer dragon count to MEMORY
;
                JSR   Tri_State
                LDD   top
                STD   bottom
                JSR   Switch_NMI            ; Tell CNTRL board OK to transfer
;                                             latched time
                JMP   MAIN_LOOP

;-------------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------
;
;  LIB2.INC { DAC Routines }
;
;-------------------------------------------------------------------------
;=========================================================================
;
; Description of Procedures
;--------------------------
;
; Procedure Del_v1
;   { 'Long Dela,' }
;
; Procedure Delay2
;   { 'Short Delay' }
;
; Procedure Init_Ports
;   { Initialise the 6321 I/O port - Comms Port
;     for comms with other boards, Io port for
;     control of memory paging }
;
; Procedure Xfer_AReg
;   { Transfer the contents of the A_Reg to the
;     Comms port - and wait for handshake }
;
; Procedure Init_Comms_Out
;   { Setup handshaking and comms port as output }
;
; Procedure Init_Comms_In
;   { Setup handshaking and comms port as input }
;
; Procedure Tri_State
;   { Set comms port as input, no handshake }
;
; Procedure Switch_NMI
;   { Switch line D4 on IO-port to toggle
;     latch/transfer time on CONTROL card (NMI
;     sequence) }
;
; Procedure Read_SRate
;   { Read sample rate code from CONTROL board
;     calculate the sample rate and store in
;     variable sample_rate }
;
;=========================================================================
;-------------------------------------------------------------------------
;
; Procedure Delay1
;
Delay1:
                PSHU A
                PSHU B
```

```
                LDA   #00h
LOOP2:
                LDB   #00h
LOOP1:          INCB
                CMPB #040h
                BNE   LOOP1
                INCA
                CMPA #0FFh
                BNE   LOOP2
                PULU B
                PULU A
                RTS
;
;    End {Delay1}
;
;-------------------------------------------------------------------------------
;
; Procedure Delay2
;
Delay2:
                PSHU A
                LDA   #00h
LOOP3:          INCA
                CMPA #0FFh
                BNE   LOOP3
                PULU A
                RTS
;
; End {Delay2}
;-------------------------------------------------------------------------------
;
; Procedure Init_Ports
;
Init_Ports:
              LDA #0FFh
              STA IO_PORT
              LDA #NO_HSHAKE
              STA IO_CONTROL
;
              LDA   #00H
              STA   io_reg          ; register
              STA   IO_PORT
              RTS
;
; End {Init_Ports}
;-------------------------------------------------------------------------------
;
; Procedure Xfer_Areg
;
Xfer_Areg:
                STA   COMMS_PORT
                LDA   COMMS_PORT      ; Dummy read to
                                      ; toggle handshake
CHECK_TFR:
```

```
                LDA   COMMS_CONTROL
                ANDA  #CHECK_STATUS
                CMPA  #CHECK_STATUS   ; Check for handshake
                BNE   CHECK_TFR
;
                RTS                   ; Return
;
; End {Xfer_AReg}
;-------------------------------------------------------------------------
;
; Procedure Init_Comms_Out
;
Init_Comms_Out:
;
                PSHU A
;
                LDA   #DDR_SET
                STA   COMMS_CONTROL
                LDA   #ALL_OUTPUT
                STA   COMMS_PORT
                LDA   #OUT_HSHAKE
                STA   COMMS_CONTROL
;
                PULU A
                RTS
;
; End {Init_Comms_Out}
;-------------------------------------------------------------------------
;
; Procedure Tri_State
;
Tri_State:
                PSHU A
;
                LDA   #DDR_SET
                STA   COMMS_CONTROL
                LDA   #ALL_INPUT
                STA   COMMS_PORT
                LDA   #NO_HSHAKE
                STA   COMMS_CONTROL
;
                PULU A
                RTS
;
; End {Tri_State}
;-------------------------------------------------------------------------
;
; Procedure Init_Comms_In
;
Init_Comms_In:
;
                PSHU A
;
                LDA   #DDR_SET
```

```
                STA   COMMS_CONTROL
                LDA   #ALL_INPUT
                STA   COMMS_PORT
                LDA   #IN_HSHAKE
                STA   COMMS_CONTROL
;
                PULU A
                RTS
;
; End {Init_Comms_Out}
;-------------------------------------------------------------------------
;
; Procedure Switch_NMI
;
Switch_NMI:
;
                LDA   #NMI_ON
                ANDA  sel_chnl          ; leave memory select bits intact
                STA   IO_PORT           ; switch line on
                JSR   DELAY2
                LDA   #NMI_OFF
                ORA   sel_chnl          ; switch line off
                STA   IO_PORT
                RTS
;
; End {Switch_NMI}
;-------------------------------------------------------------------------
;
; Procedure Read_SRate:
;
Read_SRate:     JSR Init_Comms_In
;
                ANDCC #EN_IRQ
                CWAI  #IRQ              ; wait for data from CNTRL
                ORCC  #DIS_IRQ
;
                LDA   temp
                CMPA  #0Bh
                BNE   LBL_1
                LDD   #20h
                STD   sample_rate
                BRA   END_SRATE
;
LBL_1:          LDD   #100h
                STD   sample_rate
;
END_SRATE:      JSR   Tri_State
                RTS
;
; End {Read_SRate}
;-------------------------------------------------------------------------
```

MEMORY Program

```
;-------------------------------------------------------------------------
;
;  LIB3.DCL  (Global declarations for MEMORY software)
;
;  Constants and variables for routines in MEMORY and LIB3.INC
;
;-------------------------------------------------------------------------
;
;  MEMORY MAP FOR MEMORY BOARD
;  ---------------------------
;
;  FFF0 - FFFF  =  Interrupt Table
;  FE00 - FFEF  =  Constant Data Area
;  FB00 - FDFF  =  Interupt Routines
;  FC00 - FBFF  =  Subroutines for main program
;  F800 - F9FF  =  Program
;
;
;  D000 - E000  =  Variable Data Area
;  C800 - CFFF  =  User Stack
;  C000 - C7FF  =  System Stack
;
;-------------------------------------------------------------------------
;
;  GENERAL CONSTANTS
;  -----------------
;
INT_TABLE:      EQU   0FFF6h      ; Addr of Int table
START_ADDR:     EQU   0F800h      ; Start of 2716 EPROM
CONST_ADDR:     EQU   0FE00h      ; Addr of Constants
VAR_ADDR:       EQU   0D000h      ; Addr of variables
FIRQ_ADDR:      EQU   0FA00h      ; Fast int proc addr
IRQ_ADDR:       EQU   0FB00h      ; IRQ int addr
SWI_ADDR:       EQU   00000h      ; not used
NMI_ADDR:       EQU   00000h      ; not used
PROC_ADDR:      EQU   0FC00h      ; Proc Start Addr
;
STACK:          EQU  0C7FFh       ; Hardware stack address
USER_STACK:     EQU  0CFFFh       ; User stack address
;
EN_FIRQ:        EQU  0BFh         ; Enable fast int request
EN_IRQ:         EQU  0EFh         ; Enable int request
DIS_FIRQ:       EQU  040h         ; Disable FIRQ
DIS_IRQ:        EQU  010h         ; Disable IRQ
;
FIRQ:           EQU  0BFh         ; FIRQ sel for CWAI
IRQ:            EQU  0EFh         ; IRQ sel for CWAI
;
;
;
;  I/O PORT CONSTANTS
;  ------------------
;
;  Memory mapped Port Address ( Comms port = comms with other boards
```

```
;         IO-port = misc control functions )
;
COMMS_PORT:       EQU   0A000h    ; comms port address
COMMS_CONTROL:    EQU   0A001h    ; cntrl port for comms
PC_PORT:          EQU   0A002h    ; PC port address
PC_CNTRL:         EQU   0A003h    ; cntrl port for PC
;
; Port A and B for control of MEMORY
MEM_SELECT_A:     EQU   08000h    ; D0 select ext board
                                  ; D1 = low for output
MEM_CNTRL_A:      EQU   08001h    ;
MEM_SELECT_B:     EQU   08002h    ;
MEM_CNTRL_B:      EQU   08003h    ;
;
; I/O modes for Ports
ALL_OUTPUT:       EQU   0FFh
ALL_INPUT:        EQU   000h
NO_HSHAKE:        EQU   004h
OUT_HSHAKE:       EQU   024h
IN_HSHAKE:        EQU   02Dh
;
DDR_SET:          EQU   000h
;
CHECK_STATUS:     EQU   080h
;
MAX_BUF:          EQU    07FFFh   ; End of buffer
MAX_LAST_BUF:     EQU   03FC0h    ; End of last buffer
;
PC_ON:            EQU   00001000b ; OR line to switch on PC
PC_OFF:           EQU   11110111b ; AND line to switch off PC
;
;-------------------------------------------------------------------------------
;
; GENERAL VARIABLES
; -----------------
;
ORG VAR_ADDR      ; Start of data area
;
memory_bank:      DFS   1
chip_num:         DFS   1
full:             DFS   2
temp:             DFS   1
ctrl_reg:         DFS   1
;
;___  ----------------------------------------------------------------------------
```

```
CPU "6809.tbl"
HOF "int16"
;
; {-------------------------------------------------------------}
; { Program MEM                                                 }
; {                                                             }
; { This program is written for the X16 cross assembler. The    }
; { program is written for only one RAM board, and would have   }
; { to be modified for more boards. It operates by filling the  }
; { memory sequentially, four dumps from the DAC board can be   }
; { read. The data is then transferred to the PC. It is assumed }
; { that the data is transferred to the PC before the next      }
; { dump from the DAC board. This is the case at 32Hz but       }
; { program may need to be modified for 256Hz.                  }
; {                                                             }
; {-------------------------------------------------------------}
;
INCL "LIB3.DCL"
;
; (Setup Interrupt table)
;
ORG INT_TABLE
            DWM FIRQ_ADDR       ; FIRQ Procedure Start Address
            DWM IRQ_ADDR        ; IRQ procedure start address
            DWM SWI_ADDR        ; SWI start address (not used)
            DWM NMI_ADDR        ; NMI procedure start address
            DWM START_ADDR      ; 2716 Eprom Start Addr
;
;-------------------------------------------------------------------
;
; Procedure Comms
;
; {-------------------------------------------------------------}
; { Procedure Comms (IRQ)                                       }
; {                                                             }
; { The IRQ line together with the CA/CB lines on the 6321      }
; { I/O port are used for data transfer. See data sheet for     }
; { control of CA/CB lines under interrupt.                     }
; {                                                             }
; {-------------------------------------------------------------}
;
ORG IRQ_ADDR
;
            LDX  count
            LDB  chip_num       ; Set chip on the RAM board
            ROLB
            ROLB
            ROLB
            ROLB
            ORB  #0Fh
            ANDB memory_bank
```

```
                STB   MEM_SELECT_B

                LDA   COMMS_PORT    ; A_Reg := Comms_Port
                STA   ,X+           ; [X_Reg] := A_Reg

                CMPX  #MAX_BUF      ; See if 32K is reached
                BLT   RETURN
                LDB   chip_num      ; Next chip if last chip = 32K
                INCB
                STB   chip_num
                LDX   #0000h        ; Reset X_counter

RETURN:         LDA   memory_bank   ; Restore memory bank in case
                STA   MEM_SELECT_B  ; transfer to PC interrupted
                STX   count
;
                RTI                 ; Return
;
;-------------------------------------------------------------------------------
;
ORG PROC_ADDR
;
; (Include procedures from LIB3.INC)
;
INCL "LIB3.INC"
;
; BEGIN (Main Program)
;
ORG START_ADDR

                LDS   #STACK
                LDU   #USER_STACK
; (Setup hardware and user stacks)
;
                JSR   Delay1
; (Wait for other boards to initialise)
;
                JSR   Init_Ports
                JSR   Delay2
; (Ir tialise I/O & Comms ports)
;
                LDB   #00h
                LDX   #0000h
                STB   chip_num
                STX   count
; (Setup counters)
;
;
; (Check for 11th chip in bank, then start the xfer to PC)
;
MAIN_LOOP:
```

```
                LDA   chip_num
                CMPA  #0Ah
                BNE   MAIN_LOOP
;
; (Use Y_Reg as the counter)
; (B_Reg counts the memory chips)
;
                LDY   #0000h
                LDD   #MAX_BUF
                STD   full
                LDB   #0Fh
                ANDB  memory_bank
                STB   memory_bank
                STB   MEM_SELECT_B    ; (Select memory chip)
;
                JSR   Switch_On_PC
;
XFER:           LDA   ,Y+
                JSR   Xfer_AReg
                CMPY  full
                BNE   XFER            ; 32k chip full ?
;
                INCB                  ; Select next chip
                PSHU  B
                ROLB
                ROLB
                ROLB
                ROLB
                LDA   #0Fh
                ANDA  memory_bank
                STA   memory_bank
                ORB   memory_bank
                STB   memory_bank
                STB   MEM_SELECT_B
                PULU  B
;
                CMPB  #0Ah            , If its the last (11th) chip
                BNE   CHECK_LAST      ; then stop when MAX_LAST_BUF is
                PSHU  B               ; reached
                LDD   #MAX_LAST_BUF
                STD   full
                PULU  B
                BRA   XFER
;
CHECK_LAST:     CMPB  #0Bh
                BNE   XFER
;
                LDX   #0000h          ; Finished then reset counters
                LDA   #00h
                STA   chip_num
;
```

```
; ** The PC will hold the handshake line after the last dump
;    until it has finished all disk access. It will then release
;    the line and the PC can be switched off. **
;
                JSR  Switch_Off_PC  ; switch off PC
;
                BRA  MAIN_LOOP      ; Continue
;
END.
;_____________________________________________________________________
```

```
;-------------------------------------------------------------
;
;  LIB3.INC ( MEMORY Routines )
;
;-------------------------------------------------------------
;
;=============================================================;
;                                                             ;
; Description of Procedures                                   ;
;-----------------------------                                ;
;                                                             ;
; Procedure Delay1                                            ;
;   { 'Long Delay' }                                          ;
;                                                             ;
; Procedure Delay2                                            ;
;   { 'Short Delay' }                                         ;
;                                                             ;
; Procedure Init_Ports                                        ;
;   { Initialise the 6321 I/O port - Comms Port               ;
;     for comms with other boards, Io port for                ;
;     control of memory paging }                              ;
;                                                             ;
; Procedure Xfer_AReg                                         ;
;   { Transfers the                                           ;
;     contents of the A_Reg to the PC port }                  ;
;                                                             ;
; Procedure Switch_On_PC                                      ;
;   { Enable line to turn on PC }                             ;
;                                                             ;
; Procedure Switch_Off_PC                                     ;
;   { Disable line to PC }                                    ;
;=============================================================;
;
;-------------------------------------------------------------
;
; Proced  e Delay1
;
Delay1:
                PSHU A
                PSHU B
                LDA  #00h
LOOP2:
                LDB  #00h
LOOP1:          INCB
                CMPB #040h
                BNE  LOOP1
                INCA
                CMPA #0FFh
                BNE  LOOP2
                PULU B
                PULU A
```

```
;_____________________________________________________
;
;  LIB3.INC ( MEMORY Routines )
;
;_____________________________________________________
;
;=====================================================;
;                                                     ;
; Description of Procedures                           ;
;---------------------------                          ;
;                                                     ;
; Procedure Delay1                                    ;
;   { 'Long Delay' }                                  ;
;                                                     ;
; Procedure Delay2                                    ;
;   { 'Short Delay' }                                 ;
;                                                     ;
; Procedure Init_Ports                                ;
;   { Initialise the 6321 I/O port - Comms Port       ;
;     for comms with other boards, Io port for        ;
;     control of memory paging }                      ;
;                                                     ;
; Procedure Xfer_AReg                                 ;
;   { Transfers the                                   ;
;     contents of the A_Reg to the PC port }          ;
;                                                     ;
; Procedure Switch_On_PC                              ;
;   { Enable line to turn on PC }                     ;
;                                                     ;
; Procedure Switch_Off_PC                             ;
;   { Disable line to PC }                            ;
;=====================================================;
;
;_____________________________________________________
;
; Procedure Delay1
;
Delay1:
                PSHU A
                PSHU B
                LDA  #00h
LOOP2:
                LDB  #00h
LOOP1:          INCB
                CMPB #040h
                BNE  LOOP1
                INCA
                CMPA #0FFh
                BNE  LOOP2
                PULU B
                PULU A
```

```
                RTS
;
;   End {Delay1}
;
;-------------------------------------------------------------------------
;
; Procedure Delay2
;
Delay2:
                PSHU A
                LDA  #00h
LOOP3:          INCA
                CMPA #0FFh
                BNE  LOOP3
                PULU A
                RTS
;
; End {Delay2}
;-------------------------------------------------------------------------
;
; Procedure Init_Ports
;
Init_Ports:
                LDA  #ALL_OUTPUT
                STA  PC_PORT
                LDA  #OUT_HSHAKE
                STA  PC_CNTRL

                LDA  #ALL_INPUT
                STA  COMMS_PORT
                LDA  #IN_HSHAKE
                STA  COMMS_CONTROL

                LDA  #ALL_OUTPUT
                LDB  #ND_HSHAKE
                STA  MEM_SELECT_A
                STB  MEM_CNTRL_A
                STA  MEM_SELECT_B
                STB  MEM_CNTRL_B

                LDA  #00h               ; Only using one RAM board
                STA  memory_bank        ; select board address 0

                STA  ctrl_reg
                STA  MEM_SELECT_A        ; Select RAM board for writing

                RTS
;
; End {Init_Ports}
;-------------------------------------------------------------------------
;
```

```
; Procedure Xfer_Areg
;
Xfer_Areg:
;
                STA   PC_PORT
                LDA   PC_PORT       ; Dummy read to toggle handshake
CHECK_TFR:
                LDA   PC_CNTRL
                ANDA  #CHECK_STATUS
                CMPA  #CHECK_STATUS  ; Check for handshake
                BNE   CHECK_TFR
;
;
                RTS                  ; Return
;
; End {Wait_Xfer_AReg}
;-------------------------------------------------------------------------------
;
; Procedure Switch_On_PC
;
Switch_On_PC:
                LDA   #PC_ON
                ORA   ctrl_reg
                STA   ctrl_reg
                STA   MEM_ SELECT_A
                RTS
;
; End {Switch_On_PC}
;-------------------------------------------------------------------------------
;
; Procedure Switch_Off_PC
;
Switch_Off_PC:
                LDA   #PC_OFF
                ANDA  ctrl_reg
                STA   ctrl_reg
                STA   MEM_SELECT_A
                RTS
;
; End {Switch_Off_PC}
;-------------------------------------------------------------------------------
```

Program RD_SEIS

```
{-------------------------------------------------------------}
{ Program READ_SEIS                                           }
{                                                             }
{ Program reads data from the data acquisition system         }
{ MEMORY board - 3 channels, and stores the data to disk.     }
{ Four blocks of 3-channel data are read at a time.           }
{ On power up the PC will initiate data transfer and four     }
{ dumps will be read from the MEMORY board. Once data has     }
{ been stored the program signals the MEMORY board to         }
{ switch off the PC. Data is stored in binary format          }
{ Because the program allocates memory using absolute         }
{ arrays, dont use this program with any TSR's otherwise      }
{ strange things will happen !!!          }
{                                                             }
{-------------------------------------------------------------}


USES Crt;

CONST
   Port_A       = $1b4 ;     { Addresses of 8255 I/O ports }
   Port_B       = $1b5 ;
   Port_C       = $1b6 ;
   Control_Port = $1b7 ;

TYPE
   Dat_Array = Array[0..$3FFF] Of Byte ;


VAR

   dragon, loops, channel,
   count                        : Integer;

   status                       : Byte;

   val, val2                    : String[1];

   header                       : String[8];

   dat_file                     : File;

   HB_array_1                   : Dat_Array Absolute  $A000:$0000 ;
   LB_array_1                   : Dat_Array Absolute  $A000:$4000 ;
   HB_array_2                   : Dat_Array Absolute  $A000:$8000 ;
   LB_array_2                   : Dat_Array Absolute  $A000:$C000 ;
   HB_array_3                   : Dat_Array Absolute  $8000:$0000 ;
   LB_array_3                   : Dat_Array Absolute  $8000:$4000 ;

   dat_arr                      : Array[1..16] Of Byte;
   {Data in the form :  dat_arr[1]  = HByte dragon count
                        [2]  = LByte dragon count
                        [3]  = station number
                        [4]  = year BCD
```

```
                                    [5]  = month BCD
                                    [6]  = day BCD
                                    [7]  = hours BCD
                                    [8]  = minutes BCD
                                    [9]  = seconds BCD
                                    [10] = sample rate
                                    [12] = unused
                                    [13] = unused
                                    [14] = unused
                                    [15] = unused
                                    [16] = unused}

{-------------------------------------------------------------------}

   { Procedure Check_Status
     Checks for handshake from the MEMORY CARD }

   Procedure Check_Status( Var status : Byte);
      Begin
         status := Port[Port_C];
         status := status AND $80;
      End;

{-------------------------------------------------------------------}

   { Procedure Toggle_Handshake_On
     Toggles the handshake line to the MEMORY card - data received}

   Procedure Toggle_Handshake_On;
      Begin
         Port[Port_C] := $FE;
      End;

{-------------------------------------------------------------------}

   { Procedure Toggle_Handshake_Off
     Toggles handshake line off }

   Procedure Toggle_Handshake_Off;
      Begin
         Port[Port_C] := $FF;
      End;

{-------------------------------------------------------------------}

   { Procedure Get_Data (HB, LB : Dat_Array;)
     Reads the blocks of hi-byte and lo-byte data from the MEMORY
     board and stores in the HB, LB arrays}

   Procedure Get_Data( Var HB_Array, LB_Array : Dat_Array );

      Var
         status     : Byte;
         count      : Integer;
```

```
      Begin
         count := 0;
         Repeat
            Check_Status(status);
            If status = 0 Then
            Begin
               Toggle_Handshake_Off;
               HB_Array[count] := Port[Port_A];  { Get Hi-byte}
               Toggle_Handshake_on;
               Repeat
                  Check_Status(status);
                  If status = 0 Then
                  Begin
                     Toggle_Handshake_Off;
                     LB_Array[count] := Port[Port_A]; { Get Lo-byte }
                     Toggle_Handshake_On;
                     count := count + 1;
                  End;
               Until status = 0;
               Toggle_Handshake_Off;
            End; {If}
         Until count = $3800;
      End; {Get_Data}

{-------------------------------------------------------------}

BEGIN

   ClrScr;
   Writeln;
   Writeln('Read Seismic Data');
   Writeln('-----------------');
   Writeln;
   Writeln('Initialising....');
   Port[CONTROL_PORT] := $9A ;          { Setup inputs and outputs }

   Toggle_Handshake_On;                 { Initiate Handshaking     }

   For loops := 1 To 4 Do
   Begin
      { Wait for Response from MEMORY Board }
      Writeln('Ready to receive file(',loops,')');
      Repeat
         check_status(status);          { Wait for handshake }
         If status = 0 Then
            Writeln('  Receiving Data ......');
      Until status = 0;

      { Read Data From Memory Board}
      For channel := 1 To 3 Do
      Begin
      Writeln('    Reading channel(',channel,')');
         Case channel Of
```

```
            1: Get_Data(HB_Array_1, LB_Array_1);
            2: Get_Data(HB_Array_2, LB_Array_2);
            3: Get_Data(HB_Array_3, LB_Array_3);
         End; {Case}
      End; {For}

      { Get time and date data }

      For count := 1 To 10 Do
      Begin
         Toggle_Handshake_Off;
         Repeat
            Check_Status(status);
         Until status = 0;
         dat_arr[count] := Port[Port_A];
         Toggle_Handshake_On;
      End; {For}
      Toggle_Handshake_Off;
      header := '';
      For count := 6 to 9 Do    { Make header from BCD time data }
      Begin
         Str(dat_arr[count] And $0F, val1);
         Str((dat_arr[count] And $F0 ) Shr 4, val2);
         header := header + val2 + val1 ;
      End; {For}
      Writeln('    finished file : ',header);

      { Open file and write the data }
      Writeln('    Dumping to disk...');
      Assign(dat_file,header+'.dat');
      ReWrite(dat_file,16);
      BlockWrite(dat_file,dat_arr,1);
      BlockWrite(dat_file,LB_Array_1,896);
      BlockWrite(dat_file,HB_Array_1,896);  { Write channel 1 }
      BlockWrite(dat_file,LB_Array_2,896);
      BlockWrite(dat_file,HB_Array_2,896);  { Write channel 2 }
      BlockWrite(dat_file,LB_Array_3,896);
      BlockWrite(dat_file,HB_Array_3,896);  { Write channel 3 }
      Close(dat_file);
   End; {For}

   { Signal MEMORY board - transfer complete, can turn off PC.
     It is not a problem if the PC is switched off before the
     program has terminated, as all data is stored and files
     are closed. }

   Toggle_Handshake_On;
   Delay(100);
   Toggle_Handshake_Off;

END.
```

Program CONVERT

```
{-------------------------------------------------------------}
{ Program CONVERT                                             }
{                                                             }
{ Program reads in binary files generated by READ_SEIS and    }
{ decompresses the data and stores in an ASCII text file      }
{ the data has been reduced to 16 bits - rather than the      }
{ uncompressed 21/22 bits for simplicity sake. This is not    }
{ a restriction on the data, the original 21/22 bits can      }
{ be retrieved they were not however needed for test          }
{ purposes. To make 22 bits variable y_val must be made       }
{ type LongInt and all constants in the Case structure        }
{ must be multiplied by 8.                                    }
{                                                             }
{-------------------------------------------------------------}

USES crt;

TYPE
   Dat_Array = Array[0..$3FFF] Of Byte ;

VAR
   in_file                  : File;
   txt_file                 : Text;
   HB_array                 : Dat_Array;
   LB_array                 : Dat_Array;
   gain, count,
   HB_val, LB_val           : Byte;
   y_val, x                 : Integer;  { Data is reduced to 16 bits}
   st, header               : String[8];
   dat_arr                  : Array[1..16] Of Byte;


   {Data in the form :  dat_arr[1]  = HByte dragon count
                        [2]  = LByte dragon count
                        [3]  = station number
                        [4]  = year BCD
                        [5]  = month BCD
                        [6]  = day BCD
                        [7]  = hours BCD
                        [8]  = minutes BCD
                        [9]  = seconds BCD
                        [10] = sample rate
                        [12] = unused
                        [13] = unused
                        [14] = unused
                        [15] = unused
                        [16] = unused}

BEGIN
   ClrScr;
   Writeln;
   Writeln('File conversion utility');
   Writeln('-----------------------');
   Writeln;
```

```
    Writeln('Files are decompressed and converted to ASCII text files')
    Writeln;
    Write('File to be converted : ');
    Readln(header);
    Writeln;
    Writeln('  Reading File....');
    Assign(in_file,header + '.dat');
    Reset(in_file, 16);
    Assign(txt_file,header + '.txt');
    Rewrite(txt_file);
    BlockRead(in_file, dat_arr, 1);   { Read the header }

    { The header is written to first 16 charcters in file, followed
      by a newline }
    For x := 1 To 16 Do
    Begin
       Str(dat_arr[x], st);
       Write(txt_file, st,' ');       { Write the header }
    End;
    Writeln(txt_file);

    {Read the 3 channels decompress and write to text file }
    Writeln('    Converting and Writing to text file ....');
    For count := 1 To 3 Do
    Begin
       BlockRead(in_file, LB_array, 896);  { Read the seismic data }
       Blockread(in_file, HB_array, 896);
       For x := 0 To $3FFF Do
       Begin
          HB_Val := HB_Array[(x)];
          LB_Val := LB_Array[(x)];
          gain   := HB_val And $E0;          { Strip gain bits }
          HB_val := HB_val And $1F;
          y_val  := HB_val * 256 + LB_Val;
          If y_val > 4096 Then
             y_val := y_val - 8192 ;       { Make value -8192 < X < 8192
          Case gain Of
             $0  :  y_val := y_val div 16;  { Calculate multiplication }
             $20 :  y_val := y_val div 8;   { factor depending on the }
             $40 :  y_val := y_val div 2;   { gain bits }
             $60 :  y_val := y_val * 2;
             $80 :  y_val := y_val * 8;
             $A0 :  y_val := y_val * 32;
             $C0 :  y_val := y_val * 64;
          End;

          Str(y_val,st);
          Write(txt_file,st,' ');     { Write decompressed ASCII value }
       End;
    End;
    Close(txt_file);
    Close(in_file);
END.
```

List of 6309 assembler nmemonics

8-BIT ACCUMULATOR AND MEMORY INSTRUCTIONS

Mnemonic(s)	Operation
ADCA, ADCB	Add memory to accumulator with carry
ADDA, ADDB	Add memory to accumulator
ANDA, ANDB	And memory with accumulator
ASL, ASLA, ASLB	Arithmetic shift of accumulator or memory left
ASR, ASRA, ASRB	Arithmetic shift of accumulator or memory right
BITA, BITB	Bit test memory with accumulator
CLR, CLRA, CLRB	Clear accumulator or memory location
CMPA, CMPB	Compare memory from accumulator
COM, COMA, COMB	Complement accumulator or memory location
DAA	Decimal adjust A accumulator
DEC, DECA, DECB	Decrement accumulator or memory location
EORA, EORB	Exclusive or memory with accumulator
EXG R1, R2	Exchange R1 with R2 (R1, R2 = A, B, CC, DP)
INC, INCA, INCB	Increment accumulator or memory location
LDA, LDB	Load accumulator from memory
LSL, LSLA, LSLB	Logical shift left accumulator or memory location
LSR, LSRA, LSRB	Logical shift right accumulator or memory location
MUL	Unsigned multiply (A × B → D)
NEG, NEGA, NEGB	Negate accumulator or memory
ORA, ORB	Or memory with accumulator
ROL, ROLA, ROLB	Rotate accumulator or memory left
ROR, RORA, RORB	Rotate accumulator or memory right
SBCA, SBCB	Subtract memory from accumulator with borrow
STA, STB	Store accumulator to memory
SUBA, SUBB	Subtract memory from accumulator
TST, TSTA, TSTB	Test accumulator or memory location
TFR R1, R2	Transfer R1 to R2 (R1, R2 = A, B, CC, DP)

NOTE: A, B, CC or DP may be pushed to (pulled from) either stack with PSHS, PSHU (PULS, PULU) instructions.

16-BIT ACCUMULATOR AND MEMORY INSTRUCTIONS

Mnemonic(s)	Operation
ADDD	Add memory to D accumulator
CMPD	Compare memory from D accumulator
EXG D, R	Exchange D with X, Y, S, U or PC
LDD	Load D accumulator from memory
SEX	Sign Extend B accumulator into A accumulator
STD	Store D accumulator to memory
SUBD	Subtract memory from D accumulator
TFR D, R	Transfer D to X, Y, S, U or PC
TFR R, D	Transfer X, Y, S, U or PC to D

NOTE: D may be pushed (pulled) to either stack with PSHS, PSHU (PULS, PULU) instructions.

INDEX REGISTER/STACK POINTER INSTRUCTIONS

Mnemonic(s)	Operation
CMPS, CMPU	Compare memory from stack pointer
CMPX, CMPY	Compare memory from index register
EXG R1, R2	Exchange D, X, Y, S, U or PC with D, X, Y, S, U or PC
LEAS, LEAU	Load effective address into stack pointer
LEAX, LEAY	Load effective address into index register
LDS, LDU	Load stack pointer from memory
LDX, LDY	Load index register from memory
PSHS	Push A, B, CC, DP, D, X, Y, U, or PC onto hardware stack
PSHU	Push A, B, CC, DP, D, X, Y, S, or PC onto user stack
PULS	Pull A, B, CC, DP, D, X, Y, U or PC from hardware stack
PULU	Pull A, B, CC, DP, D, X, Y, S or PC from hardware stack
STS, STU	Store stack pointer to memory
STX, STY	Store index register to memory
TFR R1, R2	Transfer D, X, Y, S, U or PC to D, X, Y, S, U or PC
ABX	Add B accumulator to X (unsigned)

BRANCH INSTRUCTIONS

Mnemonic'	Operation
	SIMPLE BRANCHES
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BMI, LBMI	Branch if minus
BPL, LBPL	Branch if plus
BCS, LBCS	Branch if carry set
BCC, LBCC	Branch if carry clear
BVS, LBVS	Branch if overflow set
BVC, LBVC	Branch if overflow clear
	SIGNED BRANCHES
BGT, LBGT	Branch if greater (signed)
BGE, LBGE	Branch if greater than or equal (signed)
BEQ, LBEQ	Branch if equal
BLE, LBLE	Branch if less than or equal (signed)
BLT, LBLT	Branch if less than (signed)
	UNSIGNED BRANCHES
BHI, LBHI	Branch if higher (unsigned)
BHS, LBHS	Branch if higher or same (unsigned)
BEQ, LBEQ	Branch if equal
BLS, LBLS	Branch if lower or same (unsigned)
BLO, LBLO	Branch if lower (unsigned)
	OTHER BRANCHES
BSR, LBSR	Branch to subroutine
BRA, LBRA	Branch always
BRN, LBRN	Branch never

MISCELLANEOUS INSTRUCTIONS

Mnemonic(s)	Operation
ANDCC	AND condition code register
CWAI	AND condition code register, then wait for interrupt
NOP	No operation
ORCC	OR condition code register
JMP	Jump
JSR	Jump to subroutine
RTI	Return from interrupt
RTS	Return from subroutine
SWI, SWI2, SWI3	Software interrupt (absolute indirect)
SYNC	Synchronize with interrupt line

Pseudo mnemonics (X-16 Assembler Directives)

DFB - Define byte. Define the storage area on a byte by byte basis.

DFS - Define storage. Reserve a section of memory with unspecified contents.

EQU - Equate. Used to assign an integer value to a label.

INCL - Include source file.

ORG - Origin. Specifies the value of the program counter during assembly.

Data sheet for 6321 I/O port

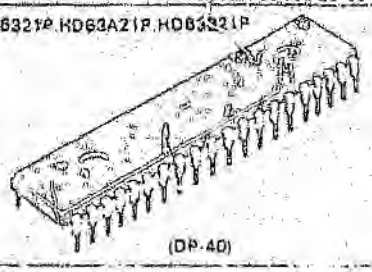
HD6321,HD63A21,HD63B21——
CMOS PIA (Peripheral Interface Adapter)

—PRELIMINARY—

The HD6321 is a CMOS Peripheral Interface Adapter provides the universal means of interfacing peripheral equipment to the HD6500 Microprocessing Unit (MPU). This device is capable of interfacing the MPU to peripherals through two 8-bit bi-directional peripheral data buses and four control lines. No external logic is required for interfacing to most peripheral devices.

The functional configuration of the PIA is programmed by the MPU during system initialization. Each of the peripheral data lines can be programmed to act as an input or output, and each of the four control/interrupt lines may be programmed for one of several control mode. This allows a high degree of flexibility in the over-all operation of the interface. Exceeding Low power dissipation is realized due to adopting CMOS process.

HD6321P, HD63A21P, HD63B21P



(DP-40)

HD6321FP, HD63A21FP, HD63B21FP

(FP-54)

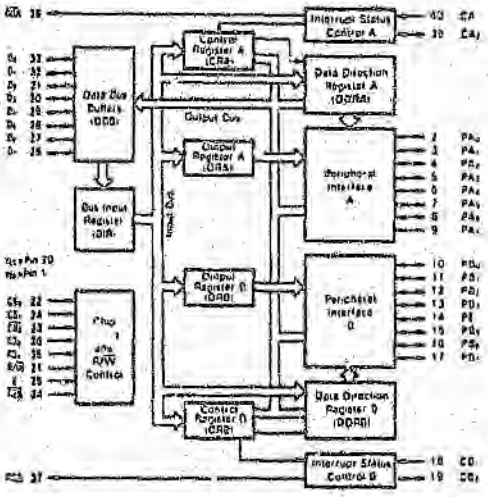
▮ FEATURES

- Low-Power, High-Speed, High-Density CMOS
- Compatible with NMOS PIA (HD6821) (Refer to Electrical Specifications as to Minor difference.)
- Two Bi-directional 8-Bit Peripheral Data Bus for interface to Peripheral devices
- High-Impedance 3-State and Direct Transistor Drive Peripheral Lines
- Two TTL Drive Capability on All A and B Side Buffers
- Handshake Control Logic for Input and Output Peripheral Operation

  CA_1, CA_2 ..... Port A ($PA_0 \sim PA_7$)

  CB_1, CB_2 ..... Port b ($PB_0 \sim PB_7$)
- Two Programmable Control Registers (CRA, CRB)
- Two Programmable Data Direction Registers (DDRA, DDRB)

▮ BLOCK DIAGRAM



PIA INTERFACE SIGNALS FOR MPU

The PIA interfaces to the HD6800 MPU with an eight-bit bidirectional data bus, three chip select lines, two register select lines, two interrupt request lines, read/write line, enable line and reset line. These signals, in conjunction with the HD6800 VMA output, permit the MPU to have complete control over the PIA. VMA should be utilized in conjunction with an MPU address line into a chip select of the PIA.

- Bi-Directional Data (D_0~D_7)

The bi-directional data lines (D_0~D_7) allow the transfer of data between the MPU and the PIA. The data bus output drivers are three-state devices that remain in the high-impedance (off) state except when the MPU performs a PIA read operation. The R/W line is in the Read ("High") state when the PIA is selected for a Read operation.

- Enable (E)

The enable pulse, E, is the only timing signal that is supplied to the PIA. Timing of all other signals is referenced to the leading and trailing edges of the E pulse. This signal will normally be a derivative of the MC6800 System ϕ_2 Clock. This signal must be a continuous clock pulse.

- Read/Write (R/W)

This signal is generated by the MPU to control the direction of data transfers on the Data Bus. A "Low" state on the PIA input line enables the input buffers and data is transferred from the MPU to the PIA on the E signal if the device has been selected. A "High" on the R/W line sets up the PIA for a transfer of data to the bus. The PIA output buffers are enabled when the proper device address and the enable pulse E are present.

- Reset ($\overline{RES}$)

The active "Low" $\overline{RES}$ line is used to reset all register bits in the PIA to a logical zero "Low". This line can be used as a power-on reset and as a master reset during system operation.

- Chip Select (CS_0, CS_1 and $\overline{CS_2}$)

These three input signals are used to select the PIA. CS_0 and CS_1 must be "High" and $\overline{CS_2}$ must be "Low" for selection of the device. Data transfers are then performed under the control of the E and R/W signals. The chip select lines must be stable for the duration of the E pulse. The device is deselected when any of the chip selects are in the inactive state.

- Register Select (RS_0 and RS_1)

The two register select lines are used to select the various registers inside the PIA. These two lines are used in conjunction with internal Control Registers to select a particular register that is to be written or read.

The register and chip select lines should be stable for the duration of the E pulse while in the read or write cycle.

- Interrupt Request ($\overline{IRQA}$ and $\overline{IRQB}$)

The active "Low" Interrupt Request lines ($\overline{IRQA}$ and $\overline{IRQB}$) act to interrupt the MPU either directly or through interrupt priority circuitry. These lines are "open drain" (no load device on the chip). This permits all interrupt request lines to be tied together in a wire-OR configuration.

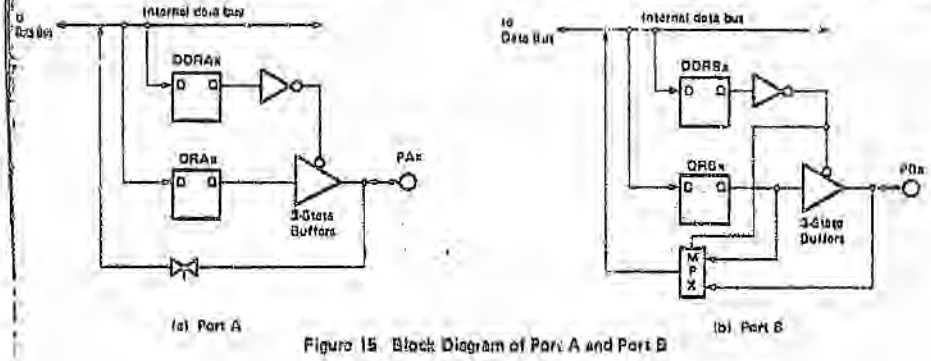
Each $\overline{IRQ}$ line has two internal interrupt flag bits that can cause the $\overline{IRQ}$ line to go "Low". Each flag bit is associated with a particular peripheral interrupt line. Also four interrupt enable bits are provided in the PIA which may be used to inhibit a particular interrupt from a peripheral device.

Servicing an interrupt by the MPU may be accomplished by a software routine that, on a prioritized basis, sequentially reads and tests the two control registers in each PIA for interrupt flag bits that are set.

The interrupt flags are cleared (zeroed) as a result of an MPU Read Peripheral Data Operation of the corresponding data register. After being cleared, the interrupt flag bit cannot be enabled to be set until the PIA is deselected during an E pulse. The E pulse is used to condition the interrupt control lines (CB_1, CB_2). When these lines are used as interrupt lines the E pulse must occur from the inactive edge to the active edge of the interrupt input signal to condition the transition when the interrupt flag has been enabled and has been properly conditioned, the interrupt flag will be set by the next active transition of the

CE LINES

Port A and Port B each have two interrupt control lines and two sets of 8-bit Bi-directional peripheral data bus for interfacing to input/output devices. Fig. 15 shows the block diagram of Port A and Port B. The output drivers of Port A and Port B consist of three-state drivers, allowing them to enter a High-impedance state when the peripheral data line is used as an input. Port A and Port B have the same output buffer. But the circuit configuration is slightly different and this makes the difference on data flow when MPU reads Port A and Port B in the case each Port is specified as output. As shown in Fig. 15, the output of the peripheral data A is transferred to internal data bus when used as output. On the other hand, in the case of Port B the contents of output register (ORB) is directly transferred to internal data bus through the multiplexor.



(a) Port A (b) Port B

Figure 15 Block Diagram of Port A and Port B

- Port A Peripheral Data (PA_0 ~ PA-)

Each of the peripheral data lines can be programmed to act as an input or output. This is accomplished by setting a "1" in the corresponding Data Direction Register bit for those lines which are to be outputs. A "0" in a bit of the Data Direction Register causes the corresponding peripheral data line to act as an input. During an MPU Read Peripheral Data Operation, the data on peripheral lines programmed to act as inputs appears directly on the corresponding MPU Data Bus lines.

The data in Output Register A will appear on the data lines that are programmed to be outputs. A logical "1" written into this register will cause a "High" on the corresponding data line while a "0" results in a "Low". Data in Output Register A may be read by an MPU "Read Peripheral Data A" operation when the corresponding lines are programmed as outputs.

- Port B Peripheral Data (PB_0 ~ PB_7)

Each of the Port B peripheral data bus can be programmed to act as an input or output like PA_0 ~ PA_7.

PB_0 ~ PB_7 are in High-impedance condition because they are three-state outputs just like PA_0 ~ PB_0 when the peripheral buses are used as inputs, when programmed as outputs. MPU read of Port B make it possible to read the output register regardless of PB_0 ~ PB_7 loads.

- Interrupt Input (CA_1 and CB_1)

Peripheral input lines CA_1 and CB_1 are input only lines that set the interrupt flags of the control registers. The active transition for these signals is also programmed by the two control registers.

- Peripheral Control (CA_2)

The peripheral control line CA_2 can be programmed to act as an interrupt input or as a peripheral control output.

The function of this signal is programmed by the Control Register A. When used as an input, this signal is in High-impedance state

- Peripheral Control (CB_2)

Peripheral Control line CB_2 may also be programmed to act as an interrupt input or peripheral control output.

This line is programmed by Control Register B.

When used as an input, this signal is in High-impedance.

(NOTE) 1. Pulse width of interrupt inputs CA_1, CA_2, CB_1 and CB_2 shall be greater than a E cycle time. In the case that "High" time of E signal is not contained in interrupt pulse, an interrupt flag may not be set.

- INTERNAL CONTROLS

There are six locations within the PIA accessible to the MPU data bus: two Peripheral Registers, two Data Direction Registers, and two Control Registers. Selection of these locations is controlled by the RS_0 and RS_1 inputs together with bit 2 in the Control Register, as shown in Table 1.

Table 1 Internal Addressing

RS_1	RS_0	Control Register Bit CRA2	Control Register Bit CRB2	Location Selected
0	0	1	x	Peripheral Register A*
0	0	0	x	Data Direction Register A
0	1	x	x	Control Register A
1	0	x	1	Peripheral Register B*
1	0	x	0	Data Direction Register B
1	1	x	x	Control Register B

x = Don't Care

* Peripheral interface register is a generic term concerning peripheral data bus and output register.

- Initialization

A "Low" reset line has the effect of zeroing all PIA registers. This will set PA_0~PA_7, PB_0~PB_7, CA_2 and CB_2 as inputs, and all interrupts disabled. The PIA must be configured during the restart program which follows the reset.

Details of possible configurations of the Data Direction and Control Register are as follows.

- Data Direction Registers (DDRA and DDRB)

The two Data Direction Registers allow the MPU to control the direction of data through each corresponding peripheral data line. A Data Direction Register bit set at "0" configures the corresponding peripheral data line as an input; a "1" results in an output.

- Control Registers (CRA and CRB)

The two Control Registers (CRA and CRB) allow the MPU to control the operation of the four peripheral control lines CA_1, CA_2, CB_1 and CB_2. In addition they allow the MPU to enable the interrupt lines and monitor the status of the interrupt flags. Bits 0 through 5 of the two registers may be written or read by the MPU when the proper chip select and register select signals are applied. Bits 6 and 7 of the two registers are read only and are modified by external interrupts occurring on control lines CA_1, CA_2, CB_1 or CB_2. The format of the control words is shown in Table 2.

Table 2 Control Word Format

	7	6	5	4	3	2	1	0
CRA	IRQA1	IRQA2	CA_2 Control			DDRA Access	CA_1 Control	
CRB	IRQB1	IRQB2	CB_2 Control			DDRB Access	CB_1 Control	

Data Direction Access Control Bit (CRA2 and CRB2)

Bit 2 in each Control register (CRA and CRB) allows selection of either a Peripheral Interface Register or the Data Direction Register when the proper register select signals are applied to RS_0 and RS_1.

Interrupt Flags (CRA6, CRA7, CRB6, and CRB7)

The four interrupt flag bits are set by active transitions of signals on the four Interrupt and Peripheral Control lines when these lines are programmed to be inputs. These bits cannot be set directly from the MPU Data Bus and are reset indirectly by a Read Peripheral Data Operation on the appropriate section.

Control of CA_1 and CB_1 Interrupt Lines (CRA0, CRB0, CRA1, and CRB1)

The two lowest order bits of the control registers are used to control the interrupt input lines CA_1 and CB_1. Bits CRA0 and CRB0 are used to enable the MPU interrupt signals $\overline{IRQA}$ and $\overline{IRQB}$, respectively. Bits CRA1 and CRB1 determine the active transition of the interrupt input signals CA_1 and CB_1 (Table 3).

Control of CA_2 and CB_2 Peripheral Control Lines (CRA3, CRA4, CRA5, CRB3, CRB4, and CRB5)

Bits 3, 4 and 5 of the two control registers are used to control the CA_2 and CB_2 Peripheral Control lines. These bits determine if the control lines will be an interrupt input or an output control signal. If bit CRA5 (CRB5) is "0" CA_2 (CB_2) is an interrupt input line similar to CA1 (CB1) (Table 4). When CRA5 (CRB5) is "1", CA_2 (CB_2) becomes an output signal that may be used to control peripheral data transfers. When in the output mode, CA_2 and CB_2 have slightly different characteristics (Table 5 and 6).

Table 3 Control of Interrupt Inputs CA_1 and CB_1

CRA1 (CRB1)	CRA0 (CRB0)	Interrupt Input CA_1 (CB_1)	Interrupt Flag CRA7 (CRB7)	MPU Interrupt Request $\overline{IRQA}$ ($\overline{IRQB}$)
0	0	↓ Active	Set "1" on ↓ of CA_1 (CB_1)	Disabled – $\overline{IRQ}$ remains "High"
0	1	↓ Active	Set "1" on ↓ of CA_1 (CB_1)	Goes "Low" when the interrupt flag bit CRA7 (CRB7) goes "1"
1	0	↑ Active	Set "1" on ↑ of CA_1 (CB_1)	Disabled – $\overline{IRQ}$ remains "High"
1	1	↑ Active	Set "1" on ↑ of CA_1 (CB_1)	Goes "Low" when the interrupt flag bit CRA7 (CRB7) goes "1"

(Notes)
1. ↑ indicates positive transition ("Low" to "High").
2. ↓ indicates negative transition ("High" to "Low").
3. The interrupt flag bit CRA7 is cleared by an MPU Read of the A Peripheral Register and CRB7 is cleared by an MPU Read of the B Peripheral Register.
4. If CRA0 (CRB0) is "0" when an interrupt occurs (interrupt disabled) and it later brought "1", $\overline{IRQA}$ ($\overline{IRQB}$) occurs after CRA0 (CRB0) is written to a "1".

Table 4 Control of CA_2 and CB_2 as Interrupt Inputs – CRA5 (CRB5) is "0"

CRA5 (CRB5)	CRA4 (CRB4)	CRA3 (CRB3)	Interrupt Input CA_2 (CB_2)	Interrupt Flag CRA6 (CRB6)	MPU Interrupt Request $\overline{IRQA}$ ($\overline{IRQB}$)
0	0	0	↓ Active	Set "1" on ↓ of CA_2 (CB_2)	Disabled – $\overline{IRQ}$ remains "High"
0	0	1	↓ Active	Set "1" on ↓ of CA_2 (CB_2)	Goes "Low" when the interrupt flag bit CRA6 (CRB6) goes "1"
0	1	0	↑ Active	Set "1" on ↑ of CA_2 (CB_2)	Disabled – $\overline{IRQ}$ remains "High"
0	1	1	↑ Active	Set "1" on ↑ of CA_2 (CB_2)	Goes "Low" when the interrupt flag bit CRA6 (CRB6) goes "1"

(Notes)
1. ↑ indicates positive transition ("Low" to "High").
2. ↓ indicates negative transition ("High" to "Low").
3. The interrupt flag bit CRA6 is cleared by an MPU Read of the A Peripheral Register and CRB6 is cleared by an MPU Read of the B Peripheral Register.
4. If CRA3 (CRB3) is "0" when an interrupt occurs (interrupt disabled) and it later brought "1", $\overline{IRQA}$ ($\overline{IRQB}$) occurs after CRA3 (CRB3) is written to a "1".

Table 5 Control of CB_2 as an Output – CRB5 is "1"

CRB6	CRB4	CRB3	CB_2	
			Cleared	Set
1	0	0	"Low" on the positive transition of the first E pulse after MPU Write "B" Data Register operation.	"High" when the interrupt flag bit CRB7 is set by an active transition of the CB_1 signal. (See Figure 16)
1	0	1	"Low" on the positive transition of the first E pulse after an MPU Write "B" Data Register operation.	"High" on the positive edge of the first "E" pulse following an "E" pulse which occurred while the part was deselected. (See Figure 16)
1	1	0	"Low" (The content of CRB3 is output on CB_2)	
1	1	1	"High" (The content of CRB3 is output on CB_2)	

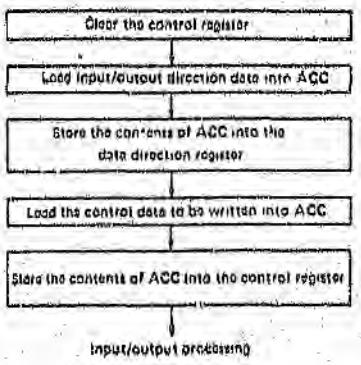
Table 6 Control of CA_2 as an Output – CRA5 is "1"

CRA5	CRA4	CRA3	CA_2	
			Cleared	Set
1	0	0	"Low" on negative transition of E after an MPU Read "A" Data Operation.	"High" when the interrupt flag bit CRA7 is set by an active transition of the CA_1 signal. (See Figure 16)
1	0	1	"Low" on negative transition of E after an MPU Read "A" Data operation.	"High" on the negative edge of the first "E" pulse which occurs during a deselect. (See Figure 16)
1	1	0	"Low" (The content of CRA3 is output on CA_2)	
1	1	1	"High" (The content of CRA3 is output on CA_2)	

■ PIA OPERATION

● Initialization

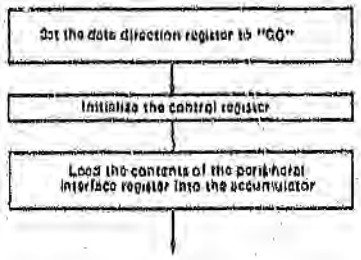
When the external reset input $\overline{RES}$ goes "Low", all internal registers are cleared to "0". Peripheral data port (PA_0~PA_7, PB_0~PB_7) is defined to be input and control lines (CA_1, CA_2, CB_1 and CB_2) are defined to be the interrupt input lines. PIA is also initialized by software sequence as follows.



- Program the data direction register access bit of the control register to "0" to allow to access the data direction register.
- The data of the control line function is set into the accumulator, of which Data Direction Register Access Bit shall be programmed to "1".
- Transfer the control data from the accumulator into the control register.

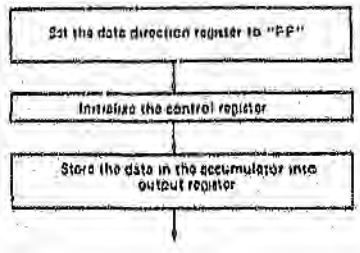
● Read/Write Operation Not Using Control Lines

<Read Operation>




```
CLR   CRA     ; Clear the DDRA access bit of the control register to "0".
CLR   DDRA    ; Clear all bits of the data direction register.
LDAA  #$04    ; Set DDRA access bit of the control register to "1" to allow
STAA  CRA     ;   to access the peripheral interface register.
      :
LDAA  PIRA
```

<Write Operation>



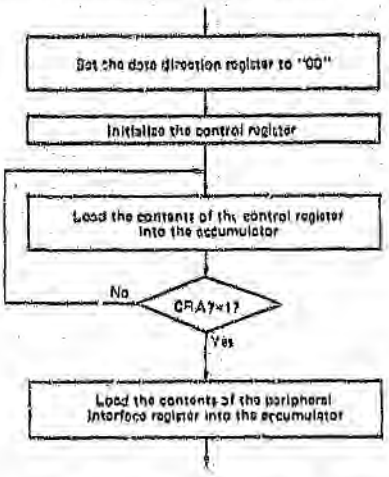

```
CLR   CRA     ; Set DDRB access bit of the control register to "0".
LDAA  #$FF    ; Set all bits of the data direction register to "FF".
STAA  DDRB
LDAA  #$04    ; Set DDRB access bit of the control register to "1" to allow to
STAA  CRB     ;   access the peripheral interface register.
      :
LDAA  DATA    ; Write the data into the peripheral interface register.
STAA  PIRB
```

• Read/Write Operating Using Control Lines

Read/write request from peripherals shall be put into the control lines as an interrupt signal, and then MPU reads or writes after detecting interrupt request.

< Read >

The following case is that Port A is used and that the rising edge of CA_1 indicates the request for read from peripherals.



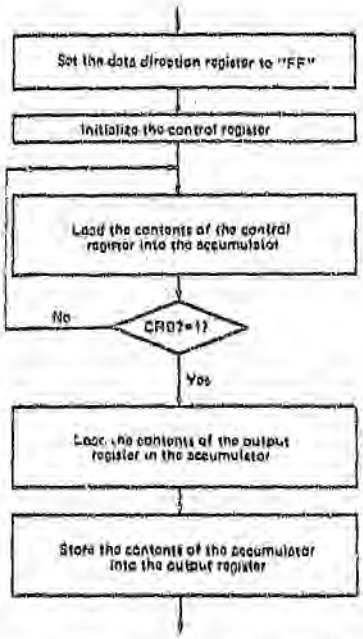

```
CLR  CRA   • Set the DDRA access bit to "0".
CLR  DDRA  • Set all bits of the data direction register to "0".

LDAA #0C0B • Program the rising edge of CA1 to be active. IRQA is masked
STAA CRA     and DDRA access bit is set to 1.

LOOP LDAA CRA
     BPL LOOP   • Check whether the read request comes from peripherals
                  or not.

LDAA PIRA  • Load the data from the peripheral interface register into the
             accumulator. CRA flag is reset after this read operation.
```

To read the peripheral data, the data is directly transfered to the data buses D_0~D_7 through PA_0~PA_7 or PB_0~PB_7 and they are not latched in the PIA. If necessary, the data should be held in the external latch until MPU completes reading it.

When initializing the control register, interrupt flag bit (CRA7, CRA6, CRB7, CRB6) cannot be written from MPU. If necessary the interrupt flag must be reset by dummy read of Peripheral Register A and B.

<Write>

Write operation using the interrupt signal is as follows. In this case, B port is used and interrupt request is input to CB_1. And the $\overline{IRQ}$ flag is set at the rising edge of CB_1.




```
CLR   CRB      ; Set DDRB access bit to "0"
LDAA  #$FF     ; Set all bits of DDRB to output "1".
STAA  DDRB
LDAA  #$06     ; Program the rising edge of CB1 to be active. IRQB is masked
STAA  CRB      ; and DDRB access bit is set to "1".
LOOP  LDAA  CRB    ; Check whether the write request comes from peripherals
      BPL   LOOP   ; or not.
LDAA  PIRB     ; Reset the CRB7 flag by the dummy read of the
               ; peripheral interface register.
STAB  PIRB     ; Store the data of the accumulator B to the peripheral
               ; interface register.
```

Interrupt request flag bits (CRA7, CRA6, CRB7 and CRB6) cannot be written, and they cannot be also reset by write operation to the peripheral interface register. So dummy read of peripheral interface register is needed to reset the flags.

To accept the next interrupt, it is essential to reset indirectly the interrupt flag by dummy read of peripheral interface register.

Software polling method mentioned above requires MPU to continuously monitor the control register to detect the read/write request from peripherals. So other programs cannot run at the same time. To avoid this problem, hardware interrupt may be used. The MPU is interrupted by $\overline{IRQA}$ or $\overline{IRQB}$ when the read/write request is occurred from peripherals and then MPU analyzes cause of the interrupt request during interrupt processing.

- Handshake Mode

The functions of CRA and CRB are similar but not identical in the hand-shake modes. Port A is used for read hand-shake operation and Port B is used for write hand-shake mode.

CA_1 and CB_1 are used for interrupt input requests and CA_2 and CB_2 are control outputs (answer) in hand-shake mode.

Fig. 16, Fig. 17 and Fig. 18 show the timing of hand-shake mode.

<Read Hand-shake Mode>
CRA5="1", CRA4="0" and CRA3="0"

① A peripheral device puts the 8-bit data on the peripheral data lines after the control output CA_2 goes "Low".

② The peripheral requests MPU to read the data by using CA_1 input.

③ CRA7 flag is set and CA_2 becomes "High" (CA_2 automatically becomes "High" by the interrupt CA_1). This indicates the peripheral to maintain the current data and not to transfer the next data.

④ MPU accepts the read request by $\overline{IRQA}$ hardware interrupt or CRA read. Then MPU reads the peripheral register A.

⑤ CA_2 goes "Low" on the following edge of read Enable pulse. This informs that the peripheral can set the next data to port A.

<Write Hand-shake>
CRB5 = "1", CRB4 = "0" and CRB3 = "0"

① A peripheral device requests MPU to write the data by using CB_1 input. CB_2 output remains "High" until MPU write data to the peripheral interface register.

② CRB7 flag is set and MPU accepts the write request

③ MPU reads the peripheral interface register to reset CRB7 (dummy read).

④ Then MPU write data to the peripheral interface register. The data is output to port B through the output register.

⑤ CB_2 automatically becomes "Low" to tell the peripheral that new data is on port B.

⑥ The peripheral read the data on Port B peripheral data lines and set CB_1 to "Low" to tell MPU that the data on the peripheral data lines has been taken and that next data can be written to the peripheral interface register.

<Pulse mode>
CRA5 = "1", CRA4 = "0" and CRA3 = "1"
CRB5 = "1", CRB4 = "0" and CRB3 = "1"

o SUMMARY OF CONTROL REGISTERS CRA AND CRB

Control registers CRA and CRB have total control of CA_1, CA_2, CB_1, and CB_2 lines. The status of eight bits of the control registers may be read into the MPU. However, the MPU can only write into Bit 0 through Bit 5 (6 bits), since Bit 6 and Bit 7 are set only by CA_1, CA_2, CB_1, or CB_2.

o Addressing PIAs

Before addressing PIAs, the data direction (DDR) *must first* be loaded with the bit pattern that defines how each line is to function, i.e., as an input or an output. A logic "1" in the data direction register defines the corresponding line as an output while a logic "0" defines the corresponding line as an input. Since the DDR and the peripheral interface register have the same address, the control register bit 2 determines which register is being addressed. If Bit 2 in the control register is a logic "0", then the DDR is addressed. If Bit 2 in the control register is logic "1", the peripheral interface register is addressed. Therefore, it is essential that the DDR be loaded first before setting Bit 2 of the control register.

<Example>

Given a PIA with an address of 4004, 4005, 4006, and 4007. 4004 is the address of the A side peripheral interface register. 4005 is the address of the A side control register. 4006 is the address of the B side peripheral interface register. 4007 is the address of the B side control register. On the A side, Bits 0, 1, 2, and 3 will be defined as inputs, while Bits 4, 5, 6, and 7 will be used as outputs. On the B side, all lines will be used as outputs.

```
PIA1AD = 4004      (DDRA, PIRA)
PIA1AC = 4005      (CRA)
PIA1BD = 4006      (DDRB, PIRB)
PIA1BC = 4007      (CRB)

1.  LDA A #%11110000    (4 outputs, 4 inputs)
2.  STA A   PIA1AD      (Loads A DDR)
3.  LDA A #%11111111    (All outputs)
4.  STA A   PIA1BD      (Loads B DDR)
5.  LDA A #%00000100    (Sets Bit 2)
6.  STA A   PIA1AC      (Bit 2 set in A control register)
7.  STA A   PIA1BC      (Bit 2 set in B control register)
```

Statement 2 addresses the DDR, since the control register (Bit 2) has not been loaded. Statements 6 and 7 load the control registers with Bit 2 set, so addressing PIA1AD or PIA1BD accesses the peripheral interface register.

o PIA Programming Via The Index Register

The program shown in the previous section can be accomplished using the Index Register.

```
1.  LDX   #$F004
2.  STX   PIA1AD      $F0→PIA1AD, $04→PIA1AC
3.  LDX   #$FF04
4.  STX   PIA1BD      $FF→PIA1BD, $04→PIA1BC
```

Using the index register in this example has saved six bytes of program memory as compared to the program shown in the previous section.

o Active Low Outputs

When all the outputs of given PIA port are to be active "Low" (True ≦ 0.4 volts), the following procedure should be used.

a) Set Bit 2 in the control register.
b) Store all 1s ($FF) in the peripheral interface register.
c) Clear Bit 2 in the control register.
d) Store all 1s ($FF) in the data direction register.
e) Store control word (Bit 2 = 1) in control register.

<Example>

B side of PIA1 is set up to have all active low outputs. CB_1 and CB_2 are set up to allow interrupts in the HANDSHAKE MODE and CB_1 will respond to positive edges ("Low"-to-"High" transitions). Assume reset conditions. Addresses are set up and equated to the same labels as previous example.

```
1.  LDA A #4
2.  STA A PIA1BC      Set Bit 2 in PIA1BC (control register)
3.  LDA B #$FF
4.  STA B PIA1BD      All 1s in peripheral interface register
5.  CLR PIA1BC        Clear Bit 2
6.  STA B PIA1BD      All 1s in data direction register
7.  LDA A #$27
8.  STA A PIA1BC      00100111 → control register
```

The above procedure is required in order to avoid outputs going "Low", to the active "Low" TRUE STATE, when all 1s are stored to the data direction register as would be the case if normal configuration procedure were followed.

Interchanging RS_0 And RS_1

Some system applications may require movement of 16 bits of data to or from the "outside world" via two PIA ports (A side + B side). When this is the case it is an advantage to interconnect RS_1 and RS_0 as follows.

RS_0 to A1 (Address Line A1)
RS_1 to A0 (Address Line A0)

This will place the peripheral interface registers and control registers side by side in the memory map as follows.

Table	Example Address	
PIA1AD	$4004	(DDRA, PIRA)
PIA1BD	$4005	(DDRB, PIRB)
PIA1AC	$4006	(CRA)
PIA1BC	$4007	(CRB)

The index register or stackpointer may be used to move the 16-bit data in two 8-bit bytes with one instruction. As an example

LDX PIA1AD PIA1AD → IX[1], PIA1BD → IX[0]

o PIA — After Reset

When the $\overline{RES}$ (Reset Line) has been held "Low" for a minimum of one microsecond, all registers in the PIA will be cleared.

Because of the reset conditions, the PIA has been defined as

follows.

1. All I/O lines to the "outside world" have been defined as inputs.
2. CA_1, CA_2, CB_1, and CB_2 have been defined as interrupt input lines that are negative edge sensitive.
3. All the interrupts on the control lines are masked. Setting of interrupt flag bits will not cause $\overline{IRQA}$ or $\overline{IRQB}$ to go "Low".

SUMMARY OF CA_1-CB_1 PROGRAMMING

Bits 1 and 0 of the respective control registers are used to program the interrupt input control lines CA_1 and CB_1.

b1	b0	
0	0	b1 = Edge (0 = −, 1 = +)
0	1	b0 = Mask (0 = Mask, 1 = Allow)
1	0	
1	1	

Note that this is the same logic as Bits 4 and 3 for CA_2-CB_2 when CA_2-CB_2 are programmed as inputs.

SUMMARY OF CA_2-CB_2 PROGRAMMING

Bits 5, 4, and 3 of the control registers are used to program the operation of CA_2-CB_2.

	b5	b4	b3		
CA_2-CB_2 Input Mode →	0	0(−)	0	(Mask)	CA_2-CB_2 Input Mode
	0	0(−)	1	(Allow)	b4 = Edge (0 = −, 1 = +)
	0	1(+)	0	(Mask)	b3 = Mask (0 = Mask,
	0	1(+)	1	(Allow)	1 = Allow)
CA_2-CB_2 Output Mode →	1	0	0	0 = Handshake Mode	
	1	0	1	1 = Pulse Mode	
	1	1	0	b3 Following Mode	
	1	1	1		

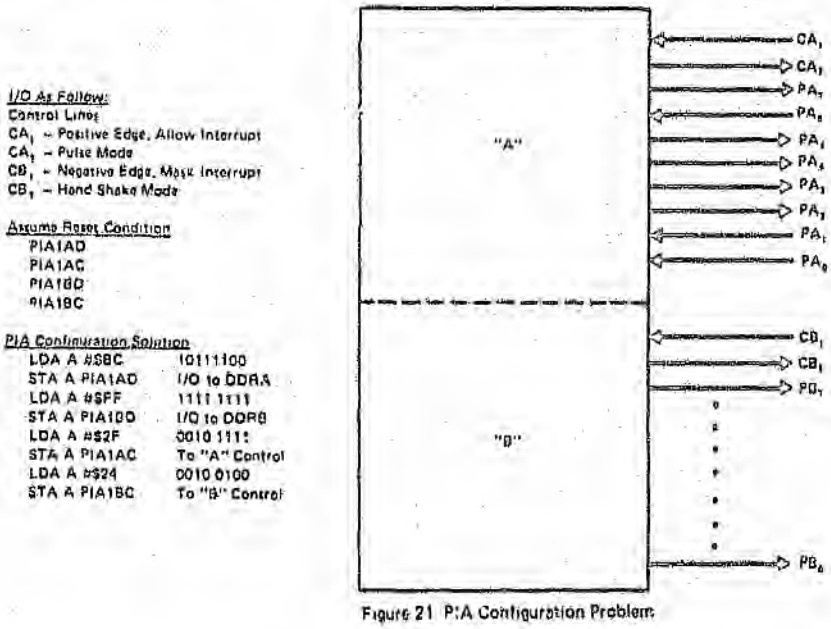


Figure 21 PIA Configuration Problem

Data sheet for 146818 Real Time Clock

MOTOROLA

MC146818

Advance Information

CMOS

(HIGH-PERFORMANCE SILICON-GATE COMPLEMENTARY MOS)

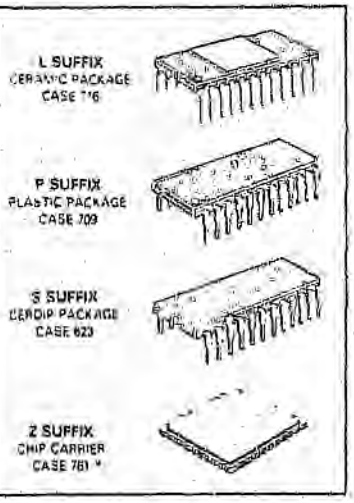
REAL-TIME CLOCK PLUS RAM

REAL-TIME CLOCK PLUS RAM (RTC)

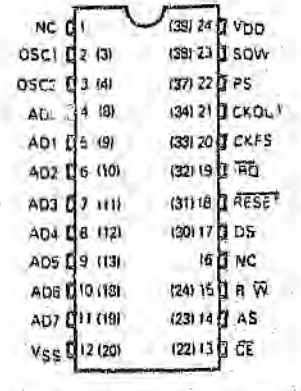
The MC146818 Real-Time Clock plus RAM is a peripheral device which includes the unique MOTEL concept for use with various microprocessors, microcomputers, and larger computers. This part combines three unique features: a complete time-of-day clock with alarm and one hundred year calendar, a programmable periodic interrupt and square-wave generator, and 50 bytes of low-power static RAM. The MC146818 uses high-speed CMOS technology to interface with 1 MHz processor buses, while consuming very little power.

The Real-Time Clock plus RAM has two distinct uses. First, it is designed as a battery powered CMOS part (in an otherwise NMOS/TTL system) including all the common battery backed-up functions such as RAM, time, and calendar. Secondly, the MC146818 may be used with a CMOS microprocessor to relieve the software of the timekeeping workload and to extend the available RAM of an MPU such as the MC146805E2.

- Low-Power, High-Speed, High-Density CMOS
- Internal Time Base and Oscillator
- Counts Seconds, Minutes, and Hours of the Day
- Counts Days of the Week, Date, Month, and Year
- 3 V to 6 V Operation
- Time Base Input Options: 4.194304 MHz, 1.048 76 MHz, or 32.768 kHz
- Time Base Oscillator for Parallel Resonant Crystals
- 40 to 200 µW Typical Operating Power at Low Frequency Time Base
- 4.0 to 20 mW Typical Operating Power at High Frequency Time Base
- Binary or BCD Representation of Time, Calendar, and Alarm
- 12- or 24-Hour Clock with AM and PM in 12-Hour Mode
- Daylight Savings Time Option
- Automatic End of Month Recognition
- Automatic Leap Year Compensation
- Microprocessor Bus Compatible
- MOTEL Circuit for Bus Universality
- Multiplexed Bus for Pin Efficiency
- Interfaced with Software as 64 RAM Locations
- 14 Bytes of Clock and Control Registers
- 50 Bytes of General Purpose RAM
- Status Bit Indicates Data Integrity
- Bus Compatible Interrupt Signals ($\overline{IRQ}$)
- Three Interrupts are Separately Software Maskable and Testable
  - Time-of-Day Alarm, Once-per-Second to Once-per-Day
  - Periodic Rates from 30.5 µs to 500 ms
  - End-of-Clock Update Cycle
- Programmable Square-Wave Output Signal
- Clock Output May Be Used as Microprocessor Clock Input
  - At Time Base Frequency ÷1 or ÷4
- 24-Pin Dual-In-Line Package
- Chip Carrier Also Available

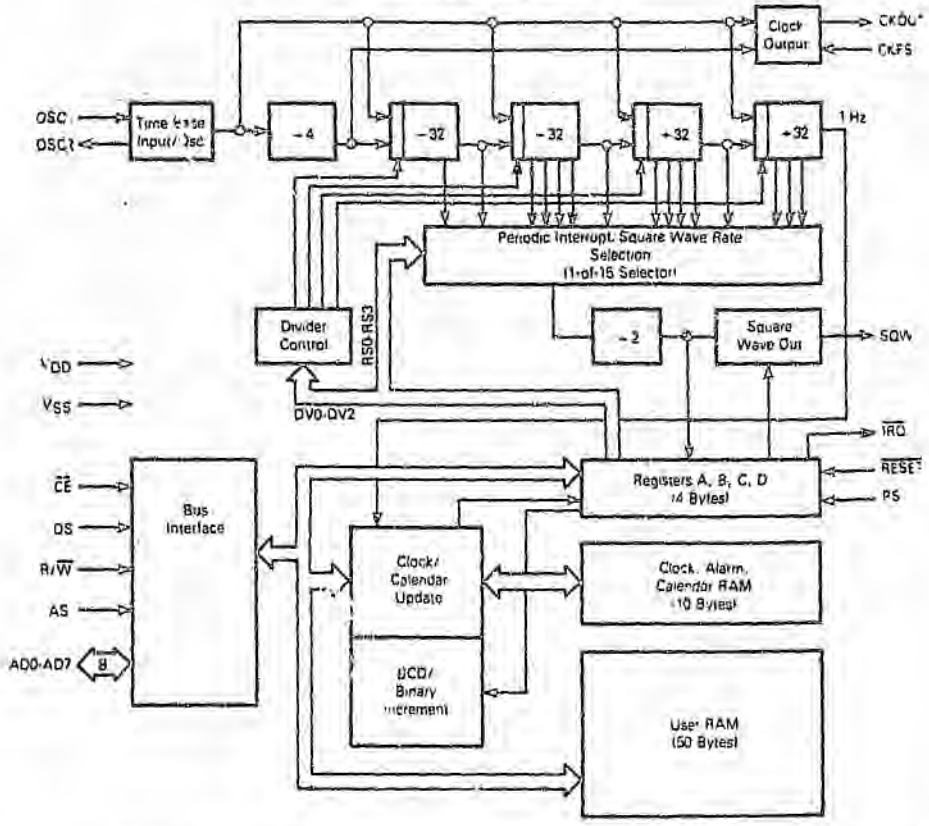


PIN ASSIGNMENT



Pin numbers in parentheses represent equivalent Z suffix chip carrier pins. Pins that have not been designated for the chip carrier are not connected.

FIGURE 1 – BLOCK DIAGRAM



MAXIMUM RATINGS (Voltages referenced to V_{SS})

Ratings	Symbol	Value	Unit
Supply Voltage	V_{DD}	−0.3 to +6.0	V
All Input Voltages Except OSC1	V_{in}	−0.5 to V_{DD} + 0.5	V
Current Drain per Pin Excluding V_{DD} and V_{SS}	I	10	mA
Operating Temperature Range MC146818 MC146818C (V_{DD} = 3.0 to 5.5 V operation)	T_A	T_L to T_H 0 to 70 −40 to 85	°C
Storage Temperature Range	T_{stg}	−55 to +150	°C

THERMAL CHARACTERISTICS

Characteristic	Symbol	Value	Unit
Thermal Resistance Plastic Cerdip Ceramic	θ_{JA}	120 65 50	°C/W

This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum rated voltages to this high-impedance circuit. For proper operation it is recommended that V_{in} and V_{out} be constrained to the range $V_{SS} \le (V_{in}$ or $V_{out}) \le V_{DD}$. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage level (e.g., either V_{SS} or V_{DD}).

MOTEL

The MOTEL circuit is a new concept that permits the MC146818 to be directly interfaced with many types of microprocessors. No external logic is needed to adapt to the differences in bus control signals from common multiplexed bus microprocessors.

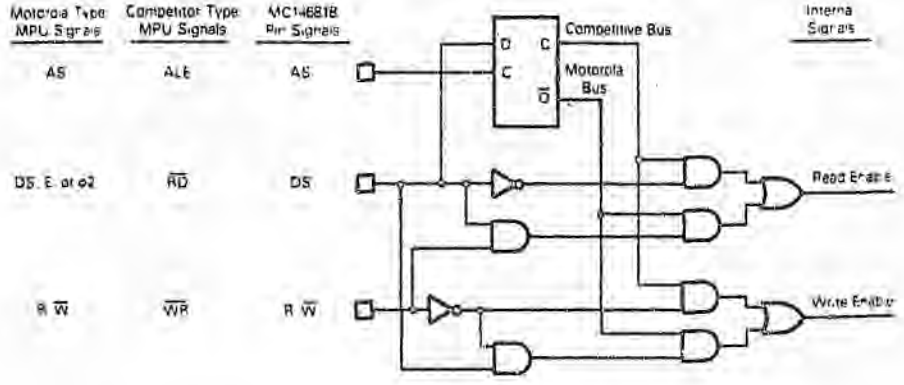
Practically all microprocessors interface with one of two synchronous bus structures. One bus was originated by the Motorola MC6800 and the other by the Intel 8080 and its companion part, the 8228.

The MOTEL circuit (for MOTorola or Intel bus compatibility) is built into peripheral and memory ICs to permit direct connection to either type of bus. An industry standard bus structure is now available. The MOTEL concept is shown logically in Figure 9.

MOTEL selects one of two interpretations of two pins. In the Motorola case, DS and R/$\overline{W}$ are gated together to produce the internal read enable. The internal write enable is a similar gating of the inverse of R/$\overline{W}$. With competitor buses, the inversion of $\overline{RD}$ and $\overline{WR}$ create functionally identical internal read and write enable signals.

The MC146818 automatically selects the processor type by using AS/ALE to latch the state of the DS/$\overline{RD}$ pin. Since DS is always low and $\overline{RD}$ is always high during AS and ALE, the latch automatically indicates which processor type is connected.

FIGURE 9 – FUNCTIONAL DIAGRAM OF MOTEL CIRCUIT



SIGNAL DESCRIPTIONS

The block diagram in Figure 1, shows the pin connection with the major internal functions of the MC146818 Real-Time Clock plus RAM. The following paragraphs describe the function of each pin.

V_{DD}, V_{SS}

DC power is provided to the part on these two pins, V_{DD} being the more positive voltage. The minimum and maximum voltages are listed in the Electrical Characteristics tables.

OSC1, OSC2 – TIME BASE, INPUTS

The time base for the time functions may be an external signal or the crystal oscillator. External square waves at 4.194304 MHz, 1.048576 MHz, or 32.768 kHz may be connected to OSC1 as shown in Figure 10. The internal time-base frequency to be used is chosen in Register A.

The on-chip oscillator is designed for a parallel resonant AT cut crystal at 4.194304 MHz or 1.048576 MHz frequencies. The crystal connections are shown in Figure 11 and the crystal characteristics in Figure 12.

CKOUT – CLOCK OUT, OUTPUT

The CKOUT pin is an output at the time-base frequency divided by 1 or 4. A major use for CKOUT is as the input clock to the microprocessor, thereby saving the cost of a second crystal. The frequency of CKOUT depends upon the time-base frequency and the state of the CKFS pin as shown in Table 2.

CKFS – CLOCK OUT FREQUENCY SELECT, INPUT

When the CKFS pin is tied to V_{DD} it causes CKOUT to be the same frequency as the time base at the OSC1 pin. When CKFS is tied to V_{SS}, CKOUT is the OSC1 time-base frequency divided by four. Table 2 summarizes the effect of CKFS.

FIGURE 10 – EXTERNAL TIME-BASE CONNECTION

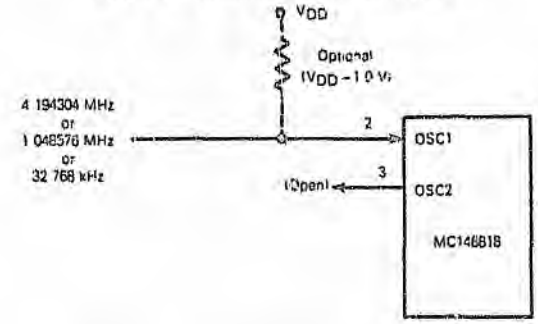
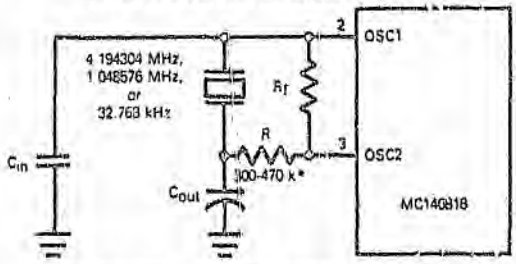


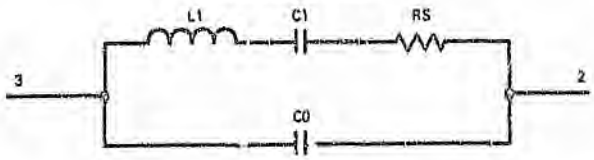
FIGURE 11 – CRYSTAL OSCILLATOR CONNECTION

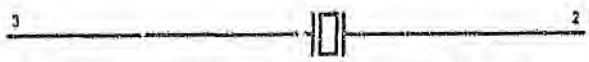


*32.768 kHz Only – Consult Crystal Manufacturer's Specification

FIGURE 12 – CRYSTAL PARAMETERS

Crystal Equivalent Circuit





f_{osc}	4.194304 MHz	1.048576 MHz	32.768 kHz
RS (Maximum)	75 Ω	700 Ω	50 k
C0 (Maximum)	7 pF	5 pF	1.7 pF
C1	0.012 pF	0.009 pF	0.003 pF
Q	50 k	35 k	30 k
C_{in}, C_{out}	15-30 pF	15-40 pF	10-22 pF
R	–	–	300-470 k
R_f	10 M	10 M	22 M

TABLE 2 — CLOCK OUTPUT FREQUENCIES

Time Base (OSC1) Frequency	Clock Frequency Select Pin (CKFS)	Clock Frequency Output Pin (CKOUT)
4.194304 MHz	High	4.194304 MHz
4.194304 MHz	Low	1.048576 MHz
1.048576 MHz	High	1.048576 MHz
1.048576 MHz	Low	262.144 kHz
32.768 kHz	High	32.768 kHz
32.768 kHz	Low	8.192 kHz

SQW — SQUARE WAVE, OUTPUT

The SQW pin can output a signal from one of the 15 taps provided by the 22 internal-divider stages. The frequency of the SQW may be altered by programming Register A, as shown in Table 5. The SQW signal may be turned on and off using the SQWE bit in Register B.

AD0-AD7 — MULTIPLEXED BIDIRECTIONAL ADDRESS/DATA BUS

Multiplexed bus processors save pins by presenting the address during the first portion of the bus cycle and using the same pins during the second portion for data. Address-then-data multiplexing does not slow the access time of the MC146818 since the bus reversal from address to data is occurring during the internal RAM access time.

The address must be valid just prior to the fall of AS/ALE at which time the MC146818 latches the address from AD0 to AD5. Valid write data must be presented and held stable during the latter portion of the DS or $\overline{WR}$ pulses. In a read cycle, the MC146818 outputs eight bits of data during the latter portion of the DS or $\overline{RD}$ pulses, then ceases driving the bus (returns the output drivers to the high-impedance state) when DS falls (in the Motorola case of MOTEL) or $\overline{RD}$ rises (in the other case).

AS — MULTIPLEXED ADDRESS STROBE, INPUT

A positive going multiplexed address strobe pulse serves to demultiplex the bus. The falling edge of AS or ALE causes the address to be latched within the MC146818. The automatic MOTEL circuit in the MC146818 also latches the state of the DS pin with the falling edge of AS or ALE.

DS — DATA STROBE OR READ, INPUT

The DS pin has two interpretations via the MOTEL circuit. When emanating from a Motorola type processor, DS is a positive pulse during the latter portion of the bus cycle, and is variously called DS (data strobe), E (enable), and φ2 (φ2 clock). During read cycles, DS signifies the time that the RTC is to drive the bidirectional bus. In write cycles, the trailing edge of DS causes the Real Time Clock plus RAM to latch the written data.

The second MOTEL interpretation of DS is that of $\overline{RD}$, $\overline{MEMR}$, or $\overline{I/OR}$ emanating from the competitor type processor. In this case, DS identifies the time period when the real-time clock plus RAM drives the bus with read data. This interpretation of DS is also the same as an output-enable signal on a typical memory.

The MOTEL circuit, within the MC146818, latches the state of the DS pin on the falling edge of AS/ALE. When the Motorola mode of MOTEL is desired DS must be low during AS/ALE, which is the case with the Motorola multiplexed bus processors. To ensure the competitor mode of MOTEL, the DS pin must remain high during the time AS/ALE is high.

R/$\overline{W}$ — READ/WRITE, INPUT

The MOTEL circuit treats the R/$\overline{W}$ pin in one of two ways. When a Motorola type processor is connected, R/$\overline{W}$ is a level which indicates whether the current cycle is a read or write. A read cycle is indicated with a high level on R/$\overline{W}$ while DS is high, whereas a write cycle is a low on R/$\overline{W}$ during DS.

The second interpretation of R/$\overline{W}$ is as a negative write pulse, $\overline{WR}$, $\overline{MEMW}$, and $\overline{IOW}$ from competitor type processors. The MOTEL circuit in this mode gives R/$\overline{W}$ pin the same meaning as the write ($\overline{W}$) pulse on many generic RAMs.

$\overline{CE}$ — CHIP ENABLE, INPUT

The chip-enable ($\overline{CE}$) signal must be asserted (low) for a bus cycle in which the MC146818 is to be accessed. $\overline{CE}$ is not latched and must be stable during DS and AS (Motorola case of MOTEL) and during $\overline{RD}$ and $\overline{WR}$ (in the other MOTEL case). Bus cycles which take place without asserting $\overline{CE}$ cause no actions to take place within the MC146818. *When $\overline{CE}$ is high, the multiplexed bus output is in a high impedance state.*

When $\overline{CE}$ is high, all address, data, DS, and R/$\overline{W}$ inputs from the processor are disconnected within the MC146818. This permits the MC146818 to be isolated from a powered-down processor. When $\overline{CE}$ is held high, an unpowered device cannot receive power through the input pins from the real-time clock power source. Battery power consumption can thus be reduced by using a pullup resistor or active clamp on $\overline{CE}$ when the main power is off. When $\overline{CE}$ is not used, it should be grounded.

$\overline{IRQ}$ — INTERRUPT REQUEST, OUTPUT

The $\overline{IRQ}$ pin is an active low output of the MC146818 that may be used as an interrupt input to a processor. The $\overline{IRQ}$ output remains low as long as the status bit causing the interrupt is present and the corresponding interrupt-enable bit is set. To clear the $\overline{IRQ}$ pin, the processor program normally reads Register C. The $\overline{RESET}$ pin also clears pending interrupts.

When no interrupt conditions are present, the $\overline{IRQ}$ level is in the high-impedance state. Multiple interrupting devices may thus be connected to an $\overline{IRQ}$ bus with one pullup at the processor.

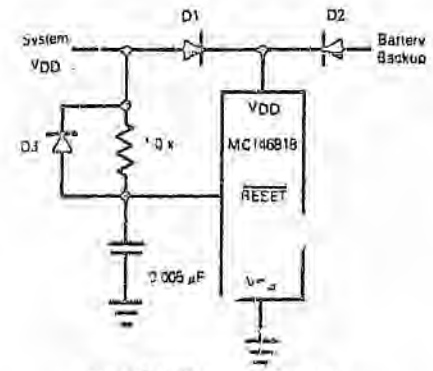
$\overline{RESET}$ — RESET, INPUT

The $\overline{RESET}$ pin does not affect the clock, calendar, or RAM functions. On powerup, the $\overline{RESET}$ pin must be held low for the specified time, t_{RLH}, in order to allow the power supply to stabilize. Figure 13 shows a typical representation of the $\overline{RESET}$ pin circuit.

When $\overline{RESET}$ is low the following occurs:

a) Periodic Interrupt Enable (PIE) bit is cleared to zero.
b) Alarm Interrupt Enable (AIE) bit is cleared to zero.
c) Update ended Interrupt Enable (UIE) bit is cleared to zero.
d) Update ended Interrupt Flag (UF) bit is cleared to zero.
e) Interrupt Request status Flag (IRQF) bit is cleared to zero.
f) Periodic Interrupt Flag (PF) bit is cleared to zero.
g) The part is not accessible.

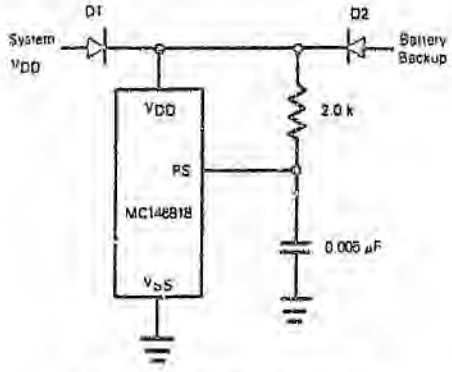
FIGURE 13 – TYPICAL POWERUP DELAY CIRCUIT FOR $\overline{RESET}$



D1 = MBD701 (Schottky) or Equivalent
D2 = D3 = 1N4148 or Equivalent

Note: If the RTC is isolated from the MPU or MCU power by a diode drop, care must be taken to meet V_{IH} requirements.

FIGURE 14 – TYPICAL POWERUP DELAY CIRCUIT FOR POWER SENSE



D1 = MBD701 (Schottky) or Equivalent
D2 = 1N4148 or Equivalent

g) Alarm Interrupt Flag (AF) bit is cleared to zero,
h) $\overline{IRQ}$ pin is in high-impedance state, and
i) Square Wave output Enable (SQWE) bit is cleared to zero.

PS – POWER SENSE, INPUT

The power-sense pin is used in the control of the valid RAM and time (VRT) bit in Register D. When the PS pin is low the VRT bit is cleared to zero.

When using the VRT feature during powerup, the PS pin must be externally held low for the specified t_{PLH} time. As power is applied, the VRT bit remains low indicating that the contents of the RAM, time registers, and calendar are not guaranteed. PS must go high after powerup to allow the VRT bit to be set by a read of register D.

POWER-DOWN CONSIDERATIONS

In most systems, the MC146818 must continue to ke p time when system power is removed. In such systems, a conversion from system power to an alternate power supply, usually a battery, must be made. During the transition from system to battery power, the designer of a battery backed-up RTC system must protect data integrity, minimize power consumption, and ensure hardware reliability.

The chip enable ($\overline{CE}$) pin controls all bus inputs (R/$\overline{W}$, DS, AS, AD0-AD7). $\overline{CE}$, when negated, disallows any unintended modification of the RTC data by the bus. $\overline{CE}$ also re: uces power consumption by reducing the number of transitions seen internally.

Power consumption may be further reduced by removing resistive and capacitive loads from the clock out (CKOUT) pin and the squarewave (SQW) pin.

During and after the power source conversion, the V_{IN} maximum specification must never be exceeded. Failure to meet the V_{IN} maximum specification can cause a virtual SCR to appear which may result in excessive current drain and destruction of the part.

ADDRESS MAP

Figure 15 shows the address map of the MC146818. The memory consists of 50 general purpose RAM bytes, 10 RAM bytes which normally contain the time, calendar, and alarm data, and four control and status bytes. All 64 bytes are directly readable and writable by the processor program except for the following: 1) Registers C and D are read only, 2) bit 7 of Register A is read only, and 3) the high-order bit of the seconds byte is read only. The contents of four control and status registers (A, B, C, and D) are described in REGISTERS.

TIME, CALENDAR, AND ALARM LOCATIONS

The processor program obtains time and calendar information by reading the appropriate locations. The program may initialize the time, calendar, and alarm by writing to these RAM locations. The contents of the 10 time, calendar and alarm bytes may be either binary or binary-coded decimal (BCD).

before initializing the internal registers, the SET bit in Register B should be set to a "1" to prevent time/calendar updates from occurring. The program initializes the 10 locations in the selected format (binary or BCD), then indicates the format in the data mode (DM) bit of Register B. All 10 time, calendar, and alarm bytes must use the same data mode, either binary or BCD. The SET bit may now be cleared to allow updates. Once initialized the real-time clock makes all updates in the selected data mode. The data mode cannot be changed without reinitializing the 10 data bytes.

Table 3 shows the binary and BCD formats of the 10 time, calendar, and alarm locations. The 24/12 bit in Register B establishes whether the hour locations represent 1-to-12 or 0-to-23. The 24/12 bit cannot be changed without reinitializing the hour locations. When the 12-hour format is selected the high-order bit of the hours byte represents PM when it is a "1".

The time, calendar, and alarm bytes are not always accessable by the processor program. Once-per-second the 10 bytes are switched to the update logic to be advanced by one second and to check for an alarm condition. If any of the 10 bytes are read at this time, the data outputs are undefined. The update lockout time is 248 µs at the 4.194304 MHz and 1.048567 MHz time bases and 1948 µs for the 32.768 kHz time base. The Update Cycle section shows how to accommodate the update cycle in the processor program.

FIGURE 15 — ADDRESS MAP

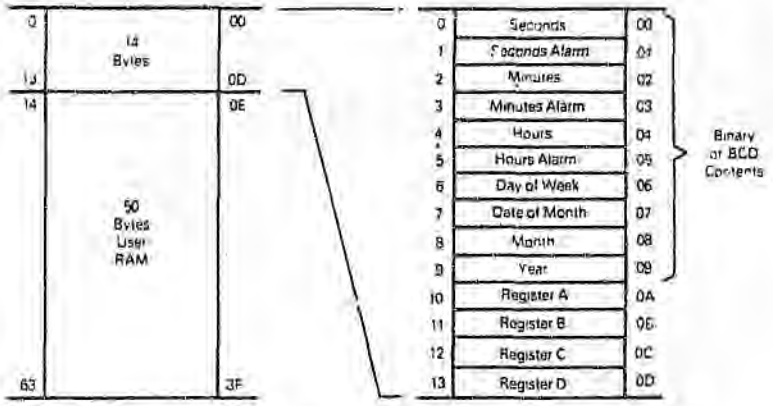


TABLE 3 — TIME, CALENDAR, AND ALARM DATA MODES

Address Location	Function	Decimal Range	Range		Example*	
			Binary Data Mode	BCD Data Mode	Binary Data Mode	BCD Data Mode
0	Seconds	0-59	$00-$3B	$00-$59	15	21
1	Seconds Alarm	0-59	$00-$3B	$00-$59	15	21
2	Minutes	0-59	$00-$3B	$00-$59	3A	58
3	Minutes Alarm	0-59	$00-$3B	$00-$59	3A	58
4	Hours (12 Hour Mode)	1-12	$01-$0C (AM) and $81-$8C (PM)	$01-$12 (AM) and $81-$92 (PM)	06	05
	Hours (24 Hour Mode)	0-23	$00-$17	$00-$23	05	05
5	Hours Alarm (12 Hour Mode)	1-12	$01-$0C (AM) and $81-$8C (PM)	$01-$12 (AM) and $81-$92 (PM)	05	05
	Hours Alarm (24 Hour Mode)	0-23	$00-$17	$00-23	05	05
6	Day of the Week Sunday = 1	1-7	$01-$07	$01-$07	05	05
7	Date of the Month	1-31	$01-$1F	$01-$31	0F	15
8	Month	1-12	$01-$0C	$01-$12	02	02
9	Year	0-99	$00-$63	$00-$99	4F	79

*Example: 5:58:21 Thursday 15 February 1979 (time is AM)

The three alarm bytes may be used in two ways. First, when the program inserts an alarm time in the appropriate hours, minutes, and seconds alarm locations, the alarm interrupt is initiated at the specified time each day if the alarm enable bit is high. The second usage is to insert a "don't care" state in one or more of three alarm bytes. The "don't care" code is any hexadecimal byte from C0 to FF. That is, the two most significant bits of each byte, when set to "1", create a "don't care" situation. An alarm interrupt each hour is created with a "don't care" code in the hours alarm location. Similarly, an alarm is generated every minute with "don't care" codes in the hours and minutes alarm bytes. The "don't care" codes in all three alarm bytes create an interrupt every second.

STATIC CMOS RAM

The 50 general purpose RAM bytes are not dedicated within the MC146818. They can be used by the processor program, and are fully available during the update cycle.

When time and calendar information must use battery back up, very frequently there is other non-volatile data that must be retained when main power is removed. The 50 user RAM bytes serve the need for low-power CMOS battery-backed storage, and extend the RAM available to the program.

When further CMOS RAM is needed, additional MC146818s may be included in the system. The time/calendar functions may be disabled by holding the DV0-DV2 dividers, in Register A, in the reset state by setting the SET bit in Register B or by removing the oscillator. Holding the dividers in reset prevents interrupts or SQW output from operating while setting the SET bit allows these functions to occur. With the dividers clear, the available user RAM is extended to 59 bytes. The high-order bit of the seconds byte, bit 7 of Register A, and all bits of Registers C and D cannot effectively be used as general purpose RAM.

INTERRUPTS

The RTC plus includes three separate fully automatic sources of interrupts to the processor. The alarm interrupt may be programmed to occur at rates from once-per-second to one-a-day. The periodic interrupt may be selected for rates from half-a-second to 30.517 μs. The update-ended interrupt may be used to indicate to the program that an update cycle is completed. Each of these independent interrupt conditions are described in greater detail in other sections.

The processor program selects which interrupts, if any, it wishes to receive. Three bits in Register B enable the three interrupts. Writing a "1" to a interrupt-enable bit permits that interrupt to be initiated when the event occurs. A "0" in the interrupt-enable bit prohibits the IRQ pin from being asserted due to the interrupt cause.

If an interrupt flag is already set when the interrupt becomes enabled, the $\overline{IRQ}$ pin is immediately activated, though the interrupt initiating the event may have occurred much earlier. Thus, there are cases where the program should clear such earlier initiated interrupts before first enabling new interrupts.

When an interrupt event occurs a flag bit is set to a "1" in Register C. Each of the three interrupt sources have separate flag bits in Register C, which are set independent of the state of the corresponding enable bits in Register B. The flag bit may be used with or without enabling the corresponding enable bits.

In the software scanned case, the program does not enable the interrupt. The "interrupt" flag bit becomes a status bit, which the software interrogates, when it wishes. When the software detects that the flag is set, it is an indication to software that the "interrupt" event occurred since the bit was last read.

However, there is one precaution. The flag bits in Register C are cleared (record of the interrupt event is erased) when Register C is read. Double latching is included with Register C so the bits which are set are stable throughout the read cycle. All bits which are high when read by the program are cleared, and new interrupts (on any bits) are held until after the read cycle. One, two, or three flag bits may be found to be set when Register C is read. The program should inspect all utilized flag bits every time Register C is read to insure that no interrupts are lost.

The second flag bit usage method is with fully enabled interrupts. When an interrupt-flag bit is set and the corresponding interrupt-enable bit is also set, the $\overline{IRQ}$ pin is asserted low. $\overline{IRQ}$ is asserted as long as at least one of the three interrupt sources has its flag and enable bits both set. The IRQF bit in Register C is a "1" whenever the $\overline{IRQ}$ pin is being driven low.

The processor program can determine that the RTC initiated the interrupt by reading Register C. A "1" in bit 7 (IRQF bit) indicates that one or more interrupts have been initiated by the part. The act of reading Register C clears all the then-active flag bits, plus the IRQF bit. When the program finds IRQF set, it should look at each of the individual flag bits in the same byte which have the corresponding interrupt-mask bits set and service each interrupt which is set. Again, more than one interrupt-flag bit may be set.

DIVIDER STAGES

The MC146818 has 22 binary-divider stages following the time base as shown in Figure 1. The output of the dividers is a 1 Hz signal to the update-cycle logic. The dividers are controller by three divider bits (DV2, DV1, and DV0) in Register A.

DIVIDER CONTROL

The divider-control bits have three uses, as shown in Table 4. Three usable operating time bases may be selected (4.194304 MHz, 1.048576 MHz, or 32.768 kHz). The divider chain may be held reset, which allows precision setting of the time. When the divider is changed from reset to an operating time base, the first update cycle is one-half second later. The divider-control bits are also used to facilitate testing the MC146818.

TABLE 4 — DIVIDER CONFIGURATIONS

Time-Base Frequency	Divider Bits Register A			Operation Mode	Divider Reset	Bypass First N-Divider Bits
	DV2	DV1	DV0			
4.194304 MHz	0	0	0	Yes		N = 0
1.048576 MHz	0	0	1	Yes	–	N = 2
32.768 kHz	0	1	0	Yes	–	N = 7
Any	1	1	0	No	Yes	–
Any	1	1	1	No	Yes	–

Note: Other combinations of divider bits are used for test purposes only.

SQUARE-WAVE OUTPUT SELECTION

Fifteen of the 22 divider taps are made available to a 1-of-15 selector as shown in Figure 1. The first purpose of selecting a divider tap is to generate a square-wave output signal at the SQW pin. The RS0-RS3 bits in Register A establish the square-wave frequency as listed in Table 5. The SQW frequency selection shares the 1-of-15 selector with periodic interrupts.

Once the frequency is selected, the output of the SQW pin may be turned on and off under program control with the square-wave enable (SQWE) bit in Register B. Altering the divider, square-wave output selection bits, or the SQWE output-enable bit may generate an asymmetrical waveform at the time of execution. The square-wave output pin has a number of potential uses. For example, it can serve as a frequency standard for external use, a frequency synthesizer, or could be used to generate one or more audio tones under program control.

PERIODIC INTERRUPT SELECTION

The periodic interrupt allows the $\overline{IRQ}$ pin to be triggered from once every 500 ms to once every 30.517 µs. The periodic interrupt is separate from the alarm interrupt which may be output from once-per-second to once-per-day.

Table 5 shows that the periodic interrupt rate is selected with the same register A bits which select the square-wave frequency. Changing one also changes the other. But each function may be separately enabled so that a program could switch between the two features or use both. The SQW pin is enabled by the SQWE bit in Register B. Similarly the periodic interrupt is enabled by the PIE bit in Register B.

Periodic interrupt is usable by practically all real-time systems. It can be used to scan for all forms of inputs from contact closures to serial receive bits or bytes. It can be used in multiplexing displays or with software counters to measure inputs, create output intervals, or await the next needed software function.

TABLE 5 — PERIODIC INTERRUPT RATE AND SQUARE WAVE OUTPUT FREQUENCY

Select Bits Register A				4.194304 or 1.048576 MHz Time Base		32.768 kHz Time Base	
RS3	RS2	RS1	RS0	Periodic Interrupt Rate (PI)	SQW Output Frequency	Periodic Interrupt Rate (PI)	SQW Output Frequency
0	0	0	0	None	None	None	None
0	0	0	1	30.517 µs	32.768 kHz	3.90625 ms	256 Hz
0	0	1	0	61.035 µs	16.384 kHz	7.8125 ms	128 Hz
0	0	1	1	122.070 µs	8.192 kHz	122.070 µs	8.192 kHz
0	1	0	0	244.141 µs	4.096 kHz	244.141 µs	4.096 kHz
0	1	0	1	488.281 µs	2.048 kHz	488.281 µs	2.048 kHz
0	1	1	0	976.562 µs	1.024 kHz	976.562 µs	1.024 kHz
0	1	1	1	1.953125 ms	512 Hz	1.953125 ms	512 Hz
1	0	0	0	3.90625 ms	256 Hz	3.90625 ms	256 Hz
1	0	0	1	7.8125 ms	128 Hz	7.8125 ms	128 Hz
1	0	1	0	15.625 ms	64 Hz	15.625 ms	64 Hz
1	0	1	1	31.25 ms	32 Hz	31.25 ms	32 Hz
1	1	0	0	62.5 ms	16 Hz	62.5 ms	16 Hz
1	1	0	1	125 ms	8 Hz	125 ms	8 Hz
1	1	1	0	250 ms	4 Hz	250 ms	4 Hz
1	1	1	1	500 ms	2 Hz	500 ms	2 Hz

UPDATE CYCLE

The MC146818 executes an update cycle once-per-second, assuming one of the proper time bases is in place, the DV0-DV2 divider is not clear, and the SET bit in Register B is clear. The SET bit in the "1" state permits the program to initialize the time and calendar bytes by stopping an existing update and preventing a new one from occurring.

The primary function of the update cycle is to increment the seconds byte, check for overflow, increment the minutes byte when appropriate and so forth through to the year of the century byte. The update cycle also compares each alarm byte with the corresponding time byte and issues an alarm if a match or if a "don't care" code (11XXXXXX) is present in all three positions.

With a 4.194304 MHz or 1.048576 MHz time base the update cycle takes 248 µs while a 32.768 kHz time base update cycle takes 1984 µs. During the update cycle, the time, calendar, and alarm bytes are not accessable by the processor program. The MC146818 protects the program from reading inaccurate data. This protection is provided by switching the time, calendar, and alarm portion of the RAM off the microprocessor bus during the entire update cycle. If the processor reads these RAM locations before the update is complete the output will be undefined. The update in progress (UIP) status bit is set during the interval.

A program which randomly accesses the time and date information finds data unavailable statistically once every 4032 attempts. Three methods of accommodating nonavailability during update are usable by the program. In discussing the three methods it is assumed that at random points user programs are able to call a subroutine to obtain the time of day.

The first method of avoiding the update cycle uses the update-ended interrupt. If enabled, an interrupt occurs after every update cycle which indicates that over 999 ms are available to read valid time and date information. During this time a display could be updated or the information could be transfered to continuously available RAM. Before leaving the interrupt service routine, the IRQF bit in Register C should be cleared.

The second method uses the update-in-progress bit (UIP) in Register A to determine if the update cycle is in progress or not. The UIP bit will pulse once-per-second. Statistically, the UIP bit will indicate that time and date information is unavailable once every 2032 attempts. After the UIP bit goes high, the update cycle begins 244 µs later. Therefore, if a low is read on the UIP bit, the user has at least 244 µs before the time/calendar data will be changed. If a "1" is read in the UIP bit, the time/calendar data may not be valid. The user should avoid interrupt service routines that would cause the time needed to read valid time/calendar data to exceed 244 µs.

The third method uses a periodic interrupt to determine if an update cycle is in progress. The UIP bit in Register A is set high between the setting of the PF bit in Register C (see Figure 16). Periodic interrupts that occur at a rate of greater than $t_{BUC} + t_{UC}$ allow valid time and date information to be read at each occurrence of the periodic interrupt. The reads should be completed within $(t_{PI} - 2) - t_{BUC}$ to ensure that data is not read during the update cycle.

To properly setup the internal counters for daylight savings time operation, the user must set the time at least two seconds before the rollover will occur. Likewise, the time must be set at least two seconds before the end of the 29th or 30th day of the month.

REGISTERS

The MC146818 has four registers which are accessible to the processor program. The four registers are also fully accessible during the update cycle.

REGISTER A ($0A)

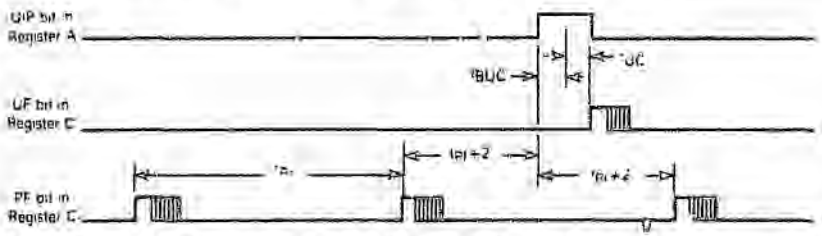
MSB							LSB	Read/Write
b7	b6	b5	b4	b3	b2	b1	b0	Register
UIP	DV2	DV1	DV0	RS3	RS2	RS1	RS0	except UIP

UIP – The update in progress (UIP) bit is a status flag that may be monitored by the program. When UIP is a "1" the update cycle is in progress or will begin. When UIP is a "0" the update cycle is not in progress and will not be for at least 244 µs (for all time bases). This is detailed in Table 6. The time, calendar, and alarm information in RAM is fully available to the program when the UIP bit is zero – it is not in transition. The UIP bit is a read only bit, and is not affected by Reset. Writing the SET bit in Register B to a "1" inhibit any update cycle and then clear the UIP status bit.

TABLE 6 – UPDATE CYCLE TIMES

UIP Bit	Time Base (OSC1)	Update Cycle Time (t_{UC})	Minimum Time Before Update Cycle (t_{BUC})
1	4.194304 MHz	248 µs	–
1	1.048576 MHz	248 µs	–
1	32.768 kHz	1984 µs	–
0	4.194304 MHz	–	244 µs
0	1.048576 MHz	–	244 µs
0	32.768 kHz	–	244 µs

FIGURE 16 – UPDATE-ENDED AND PERIODIC INTERRUPT RELATIONSHIPS



t_{PI} = Periodic Interrupt Time Interval (500 ms, 250 ms, 125 ms, 62.5 ms, etc. per Table 5)
t_{UC} = Update Cycle Time (248 µs or 1984 µs)
t_{BUC} = Delay Time Before Update Cycle (244 µs)

DV2, DV1, DV0 – Three bits are used to permit the program to select various conditions of the 22-stage divider chain. The divider selection bits identify which of the three time-base frequencies is in use. Table 4 shows that time bases of 4.194304 MHz, 1.048576 MHz, and 32.768 kHz may be used. The divider selection bits are also used to reset the divider chain. When the time/calendar is first initialized, the program may start the divider at the precise time stored in the RAM. When the divider reset is removed the first update cycle begins one-half second later. These three read/write bits are not affected by $\overline{\text{RESET}}$.

RS3, RS2, RS1, RS0 – The four rate selection bits select one of 15 taps on the 22-stage divider, or disable the divider output. The tap selected may be used to generate an output square wave ($\overline{\text{SQW}}$ pin) and/or a periodic interrupt. The program may do one of the following: 1) enable the interrupt with the PIE bit, 2) enable the $\overline{\text{SQW}}$ output pin with the SQWE bit, 3) enable both at the same time at the same rate, or 4) enable neither. Table 5 lists the periodic interrupt rates and the square-wave frequencies that may be chosen with the RS bits. These four bits are read/write bits which are not affected by $\overline{\text{RESET}}$.

REGISTER B (\$0B)

MSB							LSB	
b7	b6	b5	b4	b3	b2	b1	b0	Read/Write Register
SET	PIE	AIE	UIE	SQWE	DM	24/12	DSE	

SET – When the SET bit is a "0", the update cycle functions normally by advancing the counts once per second. When the SET bit is written to a "1", any update cycle in progress is aborted and the program may initialize the time and calendar bytes without an update occurring in the midst of initializing. SET is a read/write bit which is not modified by $\overline{\text{RESET}}$ or internal functions of the MC146818.

PIE – The periodic interrupt enable (PIE) bit is a read/write bit which allows the periodic-interrupt flag (PF) bit in Register C to cause the $\overline{\text{IRQ}}$ pin to be driven low. A program writes a "1" to the PIE bit in order to receive periodic interrupts at the rate specified by the RS3, RS2, RS1, and RS0 bits in Register A. A zero in PIE blocks $\overline{\text{IRQ}}$ from being initiated by a periodic interrupt, but the periodic flag (PF) bit is still set at the periodic rate. PIE is not modified by any internal MC146818 functions, but is cleared to "0" by a $\overline{\text{RESET}}$.

AIE – The alarm interrupt enable (AIE) bit is a read/write bit which when set to a "1" permits the alarm flag (AF) bit in Register C to assert $\overline{\text{IRQ}}$. An alarm interrupt occurs for each second that the three time bytes equal the three alarm bytes (including a "don't care" alarm code of binary 11XXXXXX). When the AIE bit is a "0", the AF bit does not initiate an $\overline{\text{IRQ}}$ signal. The $\overline{\text{RESET}}$ pin clears AIE to "0". The internal functions do not affect the AIE bit.

UIE – The UIE (update-ended interrupt enable) bit is a read/write bit which enables the update-end flag (UF) bit in Register C to assert $\overline{\text{IRQ}}$. The $\overline{\text{RESET}}$ pin going low or the SET bit going high clears the UIE bit.

SQWE – When the square-wave enable (SQWE) bit is set to a "1" by the program, a square-wave signal at the frequency specified in the rate selection bits (RS3 to RS0) appears on the $\overline{\text{SQW}}$ pin. When the SQWE bit is set to a zero, the $\overline{\text{SQW}}$ pin is held low. The state of SQWE is cleared by the $\overline{\text{RESET}}$ pin. SQWE is a read/write bit.

DM – The data mode (DM) bit indicates whether time and calendar updates are to use binary or BCD formats. The DM bit is written by the processor program and may be read by the program, but is not modified by any internal functions or $\overline{\text{RESET}}$. A "1" in DM signifies binary data, while a "0" in DM specifies binary-coded-decimal (BCD) data.

24/12 – The 24/12 control bit establishes the format of the hours bytes as either the 24-hour mode (a "1") or the 12-hour mode (a "0"). This is a read/write bit, which is affected only by software.

DSE – The daylight savings enable (DSE) bit is a read/write bit which allows the program to enable two special updates (when DSE is a "1"). On the last Sunday in April the time increments from 1:59:59 AM to 3:00:00 AM. On the last Sunday in October, when the time first reaches 1:59:59 AM it changes to 1:00:00 AM. These special updates do not occur when the DSE bit is a "0". DSE is not changed by any internal operations or reset.

REGISTER C (\$0C)

MSB							LSB	
b7	b6	b5	b4	b3	b2	b1	b0	Read Only Register
IRQF	PF	AF	UF	0	0	0	0	

IRQF – The interrupt request flag (IRQF) is set to a "1" when one or more of the following are true:

PF = PIE = "1"

AF = AIE = "1"

UF = UIE = "1"

i.e., IRQF = PF•PIE + AF•AIE + UF•UIE

Any time the IRQF bit is a "1", the $\overline{\text{IRQ}}$ pin is driven low. All flag bits are cleared after Register C is read by the program or when the $\overline{\text{RESET}}$ pin is low.

PF – The periodic interrupt flag (PF) is a read-only bit which is set to a "1" when a particular edge is detected on the selected tap of the divider chain. The RS3 to RS0 bits establish the periodic rate. PF is set to a "1" independent of the state of the PIE bit. PF being a "1" initiates an $\overline{\text{IRQ}}$ signal and sets the IRQF bit when PIE is also a "1". The PF bit is cleared by a $\overline{\text{RESET}}$ or a software read of Register C.

AF – A "1" in the AF (alarm interrupt flag) bit indicates that the current time has matched the alarm time. A "1" in the AF causes the $\overline{\text{IRQ}}$ pin to go low, and a "1" to appear in the IRQF bit, when the AIE bit also is a "1". A $\overline{\text{RESET}}$ or a read of Register C clears AF.

UF – The update-ended interrupt flag (UF) bit is set after each update cycle. When the UIE bit is a "1", the "1" in UF causes the IRQF bit to be a "1", asserting $\overline{\text{IRQ}}$. UF is cleared by a Register C read or a $\overline{\text{RESET}}$.

b3 TO b0 – The unused bits of Status Register 1 are read as "0's". They can not be written.

REGISTER D ($0D)

MSB							LSB	
b7	b6	b5	b4	b3	b2	b1	b0	Read Only Register
VRT	0	0	0	0	0	0	0	

VRT – The valid RAM and time (VRT) bit indicates the condition of the contents of the RAM, provided the power sense (PS) pin is satisfactorily connected. A "0" appears in the VRT bit when the power-sense pin is low. The processor program can set the VRT bit when the time and calendar are initialized to indicate that the RAM and time are valid. The VRT is a read only bit which is not modified by the $\overline{\text{RESET}}$ pin. The VRT bit can only be set by reading Register D.

b6 TO b0 – The remaining bits of Register D are unused. They cannot be written, but are always read as "0's."

TYPICAL INTERFACING

The MC146818 is best suited for use with microprocessors which generate an address-then-data multiplexed bus. Figures 17 and 18 show typical interfaces to bus-compatible processors. These interfaces assume that the address decoding can be done quickly. However, if standard metal gate CMOS gates are used the $\overline{\text{CE}}$ setup time may be violated. Figure 19 illustrates an alternative method of chip selection which will accommodate such slower decoding.

The MC146818 can be interfaced to single-chip microcomputers (MCU) by using eleven port lines as shown in Figure 20. Non-multiplexed bus microprocessors can be interfaced with additional support.

There is one method of using the multiplexed bus MC146818 with non-multiplexed bus processors. The interface uses available bus control signals to multiplex the address and data bus together.

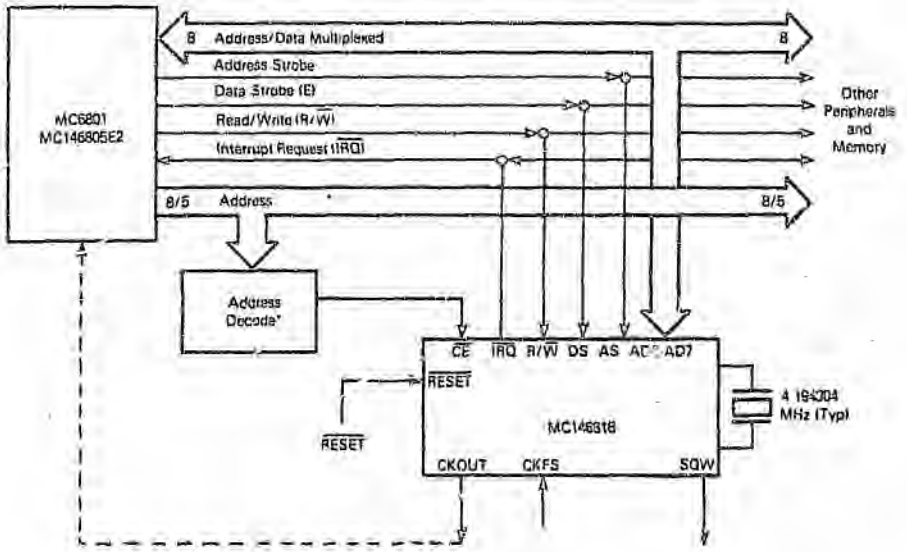
An example using either the Motorola MC6800, MC6802, MC6808, or MC6809 microprocessor is shown in Figure 21.

Figure 22 illustrates the subroutines which may be used for data transfers in a non-multiplexed system. The subroutines should be entered with the registers containing the following data.

Accumulator A: The address of the RTC to be accessed.
Accumulator B: Write: The data to be written.
Read: The data read from the RTC.

The RTC is mapped to two consecutive memory locations – RTC and RTC + 1 as shown in Figure 21.

FIGURE 17 – MC146818 INTERFACED WITH MOTOROLA COMPATIBLE MULTIPLEXED BUS MICROPROCESSORS



*High Speed Silicon-Gate CMOS or TTL Address Decoding