Report 8, 1992

COMPUTER INTERFACE FOR DIGITAL DATA ACQUISITION IN GEOTHERMAL WELL LOGGING

Alexander M. Lacanilao,
UNU Geothermal Training Programme,
Orkustofnun - National Energy Authority,
Grensasvegur 9,
108 Reykjavik,
ICELAND

Permanent address:
PNOC - Energy Development Corporation,
PNPC Complex, Merritt Road,
Fort Bonifacio, Makati,
Metro Manila,
PHILIPPINES

ABSTRACT

A project was undertaken to design and build a digital data acquisition system suitable for use by PNOC-EDC well logging units. The design was built around the utilization of personal computers in digital data acquisition employing the use of an internal bus board with counter/timer circuits to convert the frequency signals of the well logging tools. A computer program was written to drive the data acquisition system. Preliminary tests of the completed design were done with promising results. The study established the suitability of counter/timer circuits in converting the frequency signals to digital format.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This report is written in culmination of a six-month fellowship training awarded to the author by the United Nations University (UNU) under its Geothermal Training Programme. The training was conducted by Orkustofnun, the National Energy Authority of Iceland, under the auspices of the United Nations University. The training was held in Reykjavik from April to October, 1992.

The main focus of the specialized training attended by the author is in the field of Borehole Geophysics and Reservoir Engineering. The author works as a well logging engineer under the Reservoir Engineering Section of the Philippine National Oil Company - Energy Development Corporation (PNOC-EDC) Geothermal Division. PNOC-EDC is actively engaged in the development and operation of high temperature geothermal fields in the Philippines.

The well logging group of the PNOC-EDC Reservoir Engineering Section operates two electronic well logging units. The logging tools in use by PNOC-EDC are identical to those used by Orkustofnun. While Orkustofnun employs digital data gathering in well logging, the PNOC-EDC system gathers data in analog form. The author decided to undertake a project to study and get involved in designing and building a suitable model for a digital data acquisition system that may be used with the PNOC-EDC well logging units. Since the digital data acquisition system of Orkustofnun was built in-house by the local staff, a wealth of knowledge in studying a design for a digital data gathering system can be drawn from their experience.

The popularity of the modern personal computer (PC) makes it possible for almost everyone to take advantage of the power and flexibility of computerized data acquisition. Today's PCs are relatively lower priced and of compact design, small enough to fit the confined space of a well logging truck. Hence, this project focused on the use of a hardware interface for a personal computer in digital data acquisition, particularly, the application of counter/timer circuits to convert the frequency variation signal coming from most downhole logging tools. A commercially available prototype board for an IBM PC was used in the design of the system. The design of the system was conceived by Josef Holmjarn of Orkustofnun who introduced the author to basic electronics and its applications in well logging. The author was largely involved in writing the computer programs to drive the data acquisition system. The mechanics involved in the building and operation of this data acquisition system are discussed in this report.

2. WELL LOGGING ASPECTS

2.1 Parameters

Generally, well logging denotes any operation wherein some characteristic data of the formation penetrated by a borehole are recorded in terms of depth (Fertl and Overton, 1982). Such data are gathered by lowering a probe into the well and moving either up or down while physical measurements are performed. Data can be gathered electronically or mechanically depending on the type of instrument used. In either case, a sequential record of measurement is produced usually as a function of depth. These records are called well logs or simply logs. Figure 1 is an example of a well log.

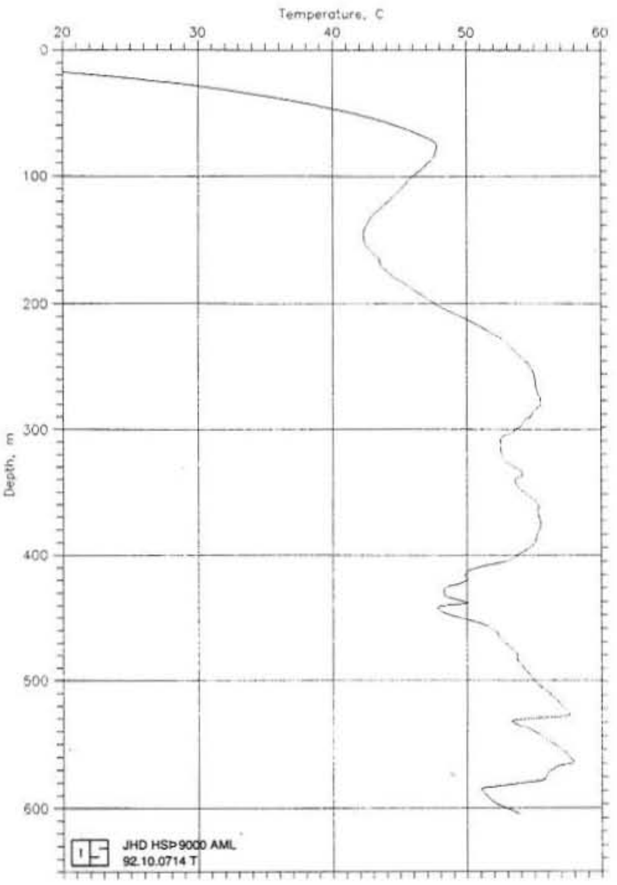


FIGURE 1: Example of well log

Geothermal well logging techniques were largely evolved from the experiences and development in the petroleum sector. However, the very high temperature and hostile borehole conditions prevalent in most geothermal wells restrict the direct application of existing petroleum logging tools in geothermal wells. Standard well logging instruments in the petroleum industry are capable of working at temperatures up to 150-180°C, too low for most of geothermal logging in which temperatures up to 350°C have been encountered (Stefansson and Steingrimsson, 1991). More recently though, several well logging companies have been developing new tools and temperature-upgrading existing ones for application in a geothermal environment.

Well logs give information on structure, physical properties and performance of the geothermal system penetrated by the borehole. Hence, well logging is vital in the measurement of key parameters identified by the Geothermal Measurement Workshop (Baker et al., 1975) as necessary in the evaluation of most geothermal resources. Following is a list of the parameters arranged to indicate ranking and importance, although priorities may vary depending on resource types.

1. Temperature
2. Formation pressure
3. Flow rate
4. Fracture system (location, orientation, permeability, etc.)
5. Fluid compositions (pH, dissolved solids and gases, redox potential)
6. Permeability
7. Porosity (interconnected and isolated)
8. Formation depth and thickness

Most of the above parameters can be measured directly or indirectly by well logging. Temperature and pressure measurements are of prime importance in the management of geothermal systems; these are the most common logging jobs done in geothermal wells. Other parameters such as fracture systems, permeability and porosity are indirectly determined, i.e. measuring the attenuation of gamma radiation or the slowing down of neutron by the formation can lead to information regarding specific gravity and porosity. In any case, the well log either recorded on paper, analog chart or magnetic media serves as a valuable source of information.

2.2 Operation

Mechanical logging tools are still in use today in geothermal well logging, however, discussion in this paper will be confined to electronic methods of well logging. In general, all well logging equipment consists of three parts , viz. the downhole sonde, transmission line and the registration unit (Stefansson and Steingrimsson, 1991). A specific probe or sonde is necessary to obtain information on the parameters mentioned above. The appropriate sonde or probe must be connected to the cable via the cable head and lowered into the well. The logging cable which is composed of one or several insulated conductors inside a steel armor transmits the power from the surface to the downhole tool and at the same time conveys the tool signal to the surface instruments. Continuous contact between the cable and the surface instruments is achieved by connecting the cable to a slip ring assembly on the cable drum. The slip ring assembly provides good electrical contact between the cable and the surface instruments, while the cable drum rotates, which provides power for the tool and feeds the signal to the surface instruments. Figure 2a shows a simplified diagram of the mechanical portion of a typical well logging operation.

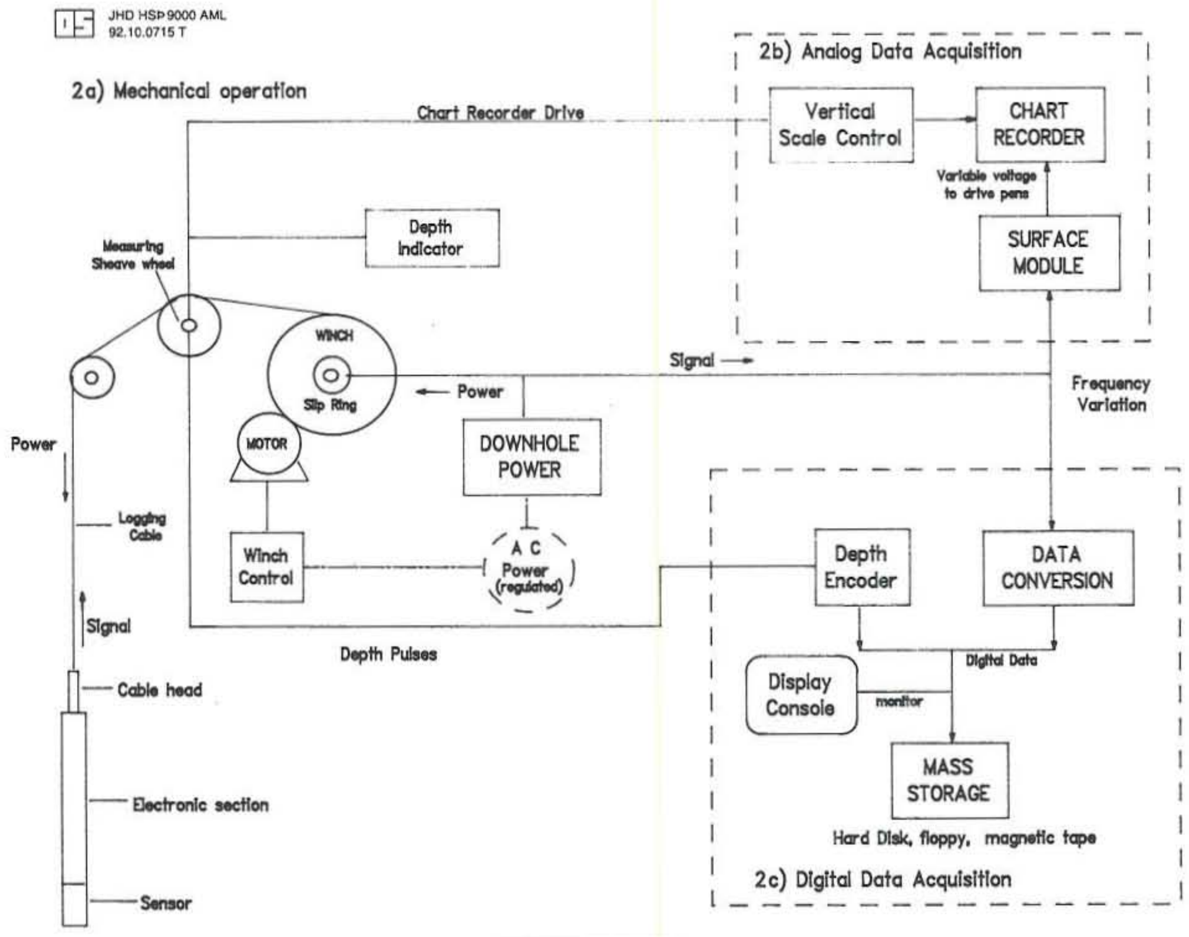


FIGURE 2: Simplified diagram of a typical well logging operation

In most cases, it is necessary to log at constant speed such that the cable drum must be motorized to gain control of the speed and direction of rotation. For logging wells with depth in excess of 1000 m, a hydraulic draw-work is practical. To establish the location of the measuring sonde inside the well, a sheave measures the line feed and intake of the cable drum. A depth indicator which could be preset to any desired value is connected to the cable measuring sheave. This keeps track of the depth of the sonde inside the well while logging either up or down.

2.3 Analog data registration

While registration of data (data acquisition) is an integral part of well logging it can be treated separately from the actual data measurement. Data acquisition may be analog, i.e. data graphically stored in charts, or digital, data numerically stored.

Conventional analog data acquisition makes use of the signal given by the tool and transmitted to the surface through the logging cable and then recorded on an analog chart. Figure 2b shows the simplified schematic diagram of analog data registration. The line is usually of considerable length and could cause problems in sending the signal to the surface. In most cases the downhole tool transforms the signal to frequency variations before it is fed to the line. Frequency signals can be transmitted over long distances relatively free of noise, further the effects of cable resistance and capacitance are greatly reduced. The measured parameter carried by the frequency signal is converted into DC (direct current) voltage by the surface modules. Different probes usually have different power requirements and in most cases different sets of surface modules are necessary to process the signal transmitted by the probe. In the case of the PNOC-EDC well logging system as well as the system in use by Orkustofnun, these modules are adapted with the use of the NIMS (Nuclear Instrument Module System) bin. This is an international trade standard with a 19" bin and an integrated power supply. NIMS modules can be easily plugged into the bin which provides the interconnection between the modules and the logging cable. The recorders are plugged to the specific module.

The DC voltage coming from the module is fed to a chart recorder which in turn drives the recorder pens to move along the chart at a magnitude proportional to the signal given by the downhole tool. The chart recorder which is coupled mechanically with the depthometer moves in step with the logging cable. Event markers mark the depth of measurement along the chart, in some instances depth are manually written on the chart based on the depth reflected by the depthometer. The parameter measured by the tool is continuously recorded in this manner.

The end product is a graph of the measured values as a function of depth. With all these parameters to be measured, a voluminous amount of data will be produced giving rise to problems in data handling and interpretation. Comparison of each parameter measured for a given well is not straightforward as usually the data is plotted on a chart of different scale. With analog charts, interpretation of results takes time and is prone to error unless extra care in data handling is observed by the analyst. Computer processing and cross-plotting of analog data can be done only after the data is digitized which entails additional operational overhead.

2.4 Digital data registration

In today's technology, modern computers offer high speed, reliability, consistency, adaptability and mass memory at a lower cost than it was a decade ago. Personal computers (PCs) offer portability which make them suitable for digital data acquisition. Data can be stored in floppy disks or the computer's mass storage, processed and plotted in relatively shorter time.

Parameters measured in geothermal well logging can be digitally recorded using microprocessors or personal computers. The signals transmitted by the tool can be converted to digital values by processing it using an analog-to-digital converter (ADC) which is interfaced to a computer. Depth values can also be encoded with the data by addition of either an optoelectronic or magnetic device on the measuring sheave wheel and a sensor to generate a pulse whenever movement of the wheel is detected. Figure 2c is a simplified schematic diagram of digital data registration in well logging. In this case, the tool must be properly calibrated and its measurement constants known. For example, a resistivity thermometer changes its resistance proportionately with a corresponding change in temperature. This proportionality can be used to translate the measured resistance to temperature. Most of today's computers are fast enough to read the data, translate to real value and write the data to a storage device. Depending on the software used, it is also possible to plot the gathered data on the computer screen while well logging is in progress.

Digital data recording offers several advantages over the analog system. Recorded data can be plotted in any scale and cross-plotting with other measured parameters is simple. In most cases, processing and interpretation of the gathered data can be done on-site using the same computer, thus saving time. Depending on the amount of mass storage available, a relatively large volume of data can be efficiently handled.

By using computer interface, it is relatively simple to convert the analog system of data acquisition to the more efficient digital system. There are, however, several factors to be considered in choosing the interface to be used. The details of these factors and the necessary peripherals needed to successfully interface an analog recording system to a computer will be discussed in the following sections.

3. CONSIDERATIONS IN DIGITAL DATA ACQUISITION

3.1 Measurement parameters

In order to design a digital data gathering system adaptable to the conventional analog system for well logging currently in use by PNOC-EDC well logging unit, some measurement parameters must first be known. The PNOC-EDC well logging group conducts four (4) types of electronic logging that will benefit mostly from digital data registration. These are

1. Temperature logs
2. Caliper logs
3. Flowmeter logs
4. Cement Bond Logs (CBL)

The logging tools used in these jobs translate the signal to either one of two forms: voltage variation or frequency variation. In the analog system, either form of signal sent by the tool is processed by the instrument modules and converted to voltage signal to drive the recorder pens.

The Gearhart-Owen (GO) temperature tool in use by PNOC-EDC makes use of a sensing element whose resistance varies with temperature. The wireline which is in excess of 3000 m has resistance and capacitance that may mask or destroy the unprocessed signal given by the tool. To circumvent the problem, the downhole tool using a voltage-controlled oscillator (vco), converts the resistance variation into frequency variation which is then transmitted to the surface. This makes the signal independent of the resistance and capacitance of the cable.

The caliper tool, used to measure borehole diameter, works on almost the same principle. In this case the resistance varies with the opening of the caliper arms in proportion to the diameter of the borehole. Likewise, the resistance variation is converted to frequency variation before feeding the signal to the wireline.

The flowmeter is used in the direct downhole measurement of flow rate. The flowmeter in use by PNOC-EDC generates a single pulse for one complete revolution of the impeller. Attached to the impeller is a small magnet which causes a reedswitch in the flowmeter shaft body to close whenever they are aligned. The speed of impeller rotation is proportional to the velocity of fluid passing through the impeller; at the surface this is recorded as the number of pulses per unit time or frequency.

Cement Bond Log tools are used to check for the quality of cementing behind the casing. The tool works primarily on the principle of sonic wave attenuation. Sonic signals are more complex as these involve timing as well as voltage amplitudes. The first arrival signal, however, is available as an integrated output from the surface processor and could be taken as a voltage amplitude signal (Maceda, 1983). This also holds true for ΔT or the travel time of the first arrival signal. In this case, a voltage to frequency converter is suitable in processing the output signal of the surface processor (module). A voltage to frequency converter is an electronic circuit that converts an input voltage into a train of digital output pulses at a rate that is directly proportional to the input (Zuch, 1980).

With the frequency output of the well logging tools mentioned above, conversion of the signal to digital format can be done by using counters and timers. Programmable counter/timers (C/T) are optimized for pulse applications including frequency measurement and time-base generation. The suitability of using counter/timers to convert the tool generated frequency signal to digital data will be considered.

3.2 Accuracy of digital data conversion

In analog systems wherein values are graphically recorded on paper while logging, measurements are done continuously with depth. Such a system of recording is not possible with digital systems. Analog to digital conversion is basically a two-step process: quantizing and coding. The conversion requires a small but significant amount of time to perform the quantizing and coding operations. Quantizing is the process of transforming a continuous analog signal into a set of discrete output states. Coding is the process of assigning a digital code word to each of the output states (Zuch, 1980). Thus an analog signal must be sampled at an interval discrete enough to be able to recreate the analog input. With most of the well logging tools mentioned earlier, the first process of analog to digital conversion is already done by the tool itself. That is, the voltage variation is quantized or converted to frequency variation which is the front end of digital data conversion. All that is needed is to convert the frequency count to its digital equivalent and record the data. The voltage to frequency conversion process and its accuracy as carried out by the logging tool will not be dealt with here, therefore the digital data accuracy referred to in this report pertains to how accurately the counter/timers can convert and digitally represent the frequency variation.

Frequency measurements using counter/timer circuits can be accomplished in different ways. Basically, the input signal (from the tool) is fed to the counter circuit which accumulates the number of input pulses. Counting can be started from a defined initial value and the counter can be configured to automatically reset after it has been read (Burr-Brown Corp., 1987). The timer portion of the circuit is made of a rate generator which provides a precise time base for accurate data gathering. The counter value (accumulated input pulses) divided by the time it took to accumulate the pulses is the digital equivalent of the input at that sampling point. Commands to read the counter/timer, perform the computation and record the data are controlled by the computer and can be tailored to fit the user's requirements.

For counter/timer circuits, the accuracy of measurement is often related to the resolution of the counter used. Most systems available today use 16-bit counters which means up to 65,536 (2^{16}) events or counts can be accumulated before the counter overflows. The resolution of the 16-bit counter is

$$\textit{Counter Resolution} = \frac{FSR}{2^{16}} \tag{1}$$

where FSR is the full scale range of the measured parameter.

The 16-bit counter can fully cover the requirements in the proposed data acquisition system for well logging. However, the measurement required for well logging does not always use the full 16 bits of the counter. The counter resolution alone does not really determine the accuracy of the expected measurement. In most cases, measurements are done in a time frame related to logging speed.

The accuracy of the counts largely affects the accuracy of digital data conversion. For an electronic counter used in frequency measurement, the major sources of error are

- the ± 1 count error
- the time base error

The ± 1 count error becomes dominant for input frequency less than 1 MHz but is masked by the time base error for input frequency higher than 1 MHz (Hewlett-Packard, 1978). For this specific application the expected input frequency of the tool is around 6000 Hz, therefore the ±1 count error is more dominant and will be given more weight in the discussion. A ±1 count ambiguity can exist in the least significant digit when a counter makes a measurement as illustrated in Figure 3. In this figure, the gate is open for the same time t_m in both cases; the gating of the counter to read the input signal can cause two valid measurements which are 1 for case #1 and 2 for case #2. In absolute terms, the error introduced is ±1 out of the total accumulated count.

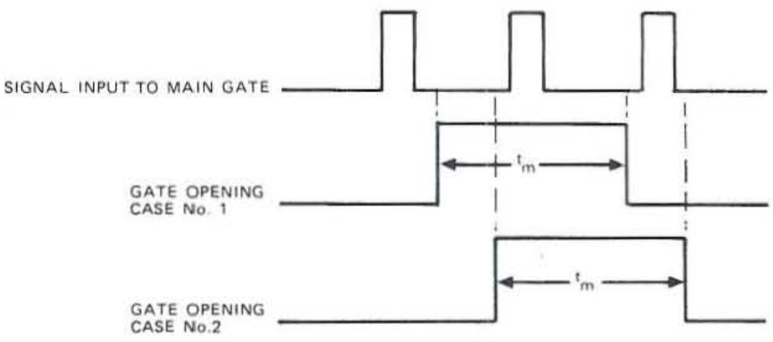


FIGURE 3: Illustration of the ±1 count ambiguity (Hewlett-Packard, 1978)

The ±1 count error applies to the counts of the tool input signal as well as the counts on the time base. For this specific application, the time base is supplied by a 1 MHz oscillator whose output is also counted by the counter. The tool *pulse count* and the *timer count* are used in the calculation of the measured frequency. In this case, the maximum frequency due to ±1 count error will result when the *time count* is 1 less (TC-1) and the *pulse count* is 1 more (PC+1) and the minimum, when the situation is reversed. Table 1 shows the frequency (columns *e* and *f*) and temperature (columns *g* and *h*) variation resulting from the ±1 count ambiguity for input frequency of 1000 Hz. Table 2 shows the same for input frequency of 6000 Hz. Note that the frequency variation is virtually unchanged for both input frequencies. The temperatures listed in Tables 1 and 2 were computed using the frequency-to-temperature characteristics (Equation 2) of the tool in use by Orkustofnun.

$$f = 36\,T + 640 \tag{2}$$

TABLE 1: Frequency and temperature variation due to ±1 count ambiguity at measured frequency of 1000 Hz; sampling interval 10 cm

Logging speed	Time frame	Expected counts		Frequency variation		Temperature variation		Difference*
m/min	*sec*	Timer	Tool	Max., *Hz*	Min., *Hz*	Max., °*C*	Min., °*C*	°*C*
(a)	*(b)*	*(c)*	*(d)*	*(e)*	*(f)*	*(g)*	*(h)*	*(i)*
0	6.000	6000000	6000	1000	1000	10.005	9.995	±0.005
5	1.200	1200000	1200	1001	999	10.023	9.977	±0.023
10	0.600	600000	600	1002	998	10.046	9.954	±0.046
15	0.400	400000	400	1003	997	10.070	9.930	±0.070
20	0.300	300000	300	1003	997	10.093	9.907	±0.093
25	0.240	240000	240	1004	996	10.116	9.884	±0.116
30	0.200	200000	200	1005	995	10.139	9.861	±0.139
35	0.171	171429	171	1006	994	10.162	9.838	±0.162
40	0.150	150000	150	1007	993	10.185	9.815	±0.185
45	0.133	133333	133	1008	992	10.209	9.791	±0.209
50	0.120	120000	120	1008	992	10.232	9.786	±0.232
55	0.109	109091	109	1009	991	10.255	9.745	±0.255
60	0.100	100000	100	1010	990	10.278	9.722	±0.278

* True temperature at 1000 Hz is 10.0°C.

TABLE 2: Frequency and temperature variation due to ±1 count ambiguity at measured frequency of 6000 Hz; sampling interval 10 cm

Logging speed	Time frame	Expected counts		Frequency variation		Temperature variation		Difference*
m/min (*a*)	*sec* (*b*)	Timer (*c*)	Tool (*d*)	Max, *Hz* (*e*)	Min, *Hz* (*f*)	Max, *°C* (*g*)	Min, *°C* (*h*)	*°C* (*i*)
0	6.000	6000000	36000	6000	6000	148.894	148.884	±0.005
5	1.200	1200000	7200	6001	5999	148.912	148.866	±0.023
10	0.600	600000	3600	6002	5998	148.935	148.842	±0.047
15	0.400	400000	2400	6003	5997	148.959	148.819	±0.070
20	0.300	300000	1800	6003	5997	148.982	148.796	±0.093
25	0.240	240000	1440	6004	5996	149.005	148.772	±0.117
30	0.200	200000	1200	6005	5995	149.029	148.749	±0.140
35	0.171	171429	1029	6006	5994	149.052	148.726	±0.163
40	0.150	150000	900	6007	5993	149.075	148.703	±0.186
45	0.133	133333	800	6008	5992	149.098	148.679	±0.210
50	0.120	120000	720	6008	5992	149.122	148.656	±0.233
55	0.109	109091	655	6009	5991	149.145	148.633	±0.256
60	0.100	100000	600	6010	5990	149.168	148.609	±0.280

* True temperature at 6000 Hz is 148.889°C.

In Tables 1 and 2, column *a* is the logging speed in m/min, and column *b* is the time frame of measurement based on a sampling interval of 10 centimetres. Column *c* lists the expected counts from the timer and *d* from the tool. Note that as the time frame of measurement becomes longer and the expected counts become larger, the temperature comes closer to the true value. Longer time frames mean slower logging speed. The ±1 count error on the *pulse count* largely affects the accuracy of measurement, count error on the *timer count* becomes negligible in the computation of measured frequency. Figures 4a and 4b show the plot of temperature variation (jitter) for measured frequencies of 1000 Hz and 6000 Hz, respectively. The jitter is a straight line function of the logging speed and varies very little for both measured frequencies. Because the counts increase proportionately with the time frame of measurement, the observable jitter of the system due to ±1 count error can be expressed as a function of the logging speed. For temperature measurements, this is

$$T_j = 1.274x10^{-3} \pm 4.604x10^{-3} v \tag{3}$$

where T_j = temperature jitter, °C;
v = logging speed, m/min.

In general, for frequency measurements, the jitter is expressed as

$$f_j = 0.0458 \pm 0.1657 v \tag{4}$$

where f_j = frequency jitter, Hz.

The ±1 count error is random, therefore, the temperature difference as listed in column *i* of both tables represents the maximum variation (jitter) in the measurements for each respective logging speed. Without correction for the tool signal *pulse count*, the system has good accuracy up to a logging speed of 25 m/min (tool accuracy is ±0.1°C). To improve the accuracy of the system, the counters should be gated exactly on the first high-to-low transition of the tool signal pulse immediately after the 10-cm depth pulse is detected. This will reduce the frequency jitter to a minimum.

3.3 Data interval and resolution

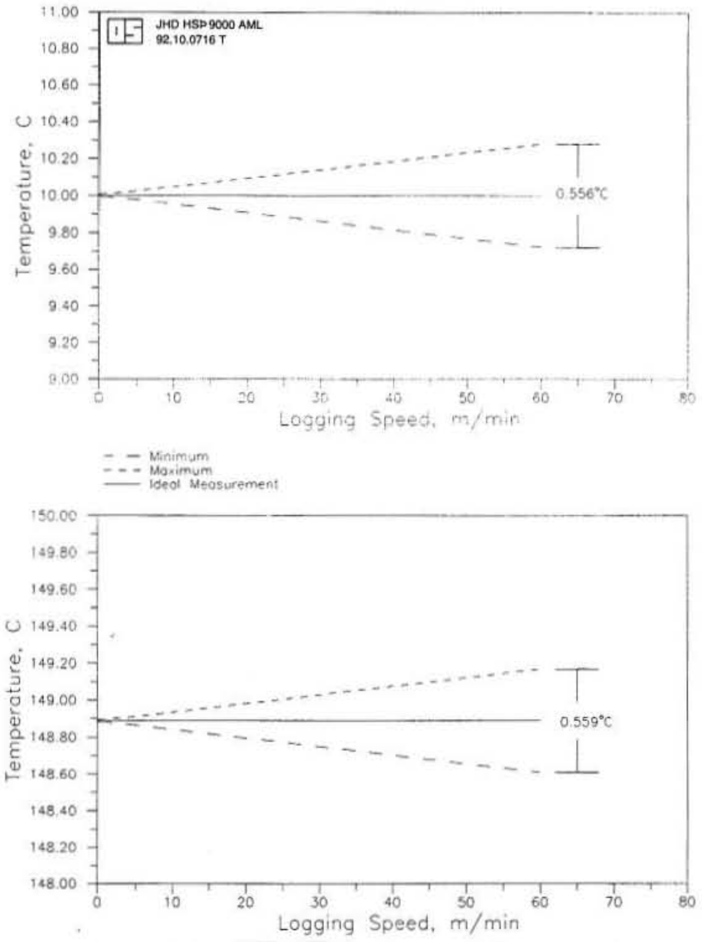


FIGURE 4: Plot of temperature jitter against logging speed at different measured frequencies

Several cases must be considered before arriving at a decision on how often to read and record the incoming data. Analog systems record data continuously as a function of depth, on the other hand, the digital system can only sample the incoming data at a certain interval to approximate the appearance of a continuous log when plotted.

In well logging, the sampling of data can be tied up to either depth or time. Incoming data may be read and recorded at a predetermined depth interval of say 10, 20 or 50 centimetres depending on the requirement. For stationary logs, data is read according to the time interval. The logging speed affects the sampling rate and data resolution. Consider a sampling interval of 10 cm. A logging speed of 10 m/min will produce new data every 600 msec while a logging speed of 60 m/min at the same sampling interval means new data is available every 100 msec. Figure 5 shows the time lapse between sampling of different intervals plotted against logging speed. Coarser sampling intervals mean a longer time lapse between data. The counter accumulates the input pulse for the duration of the time lapse, therefore, rapid changes in the input pulse frequency will be averaged out. While this is desirable in some cases due to the smoother data produced, it may not be practical where details are needed. In conditions where the measured parameter changes gradually, a coarser sampling rate can be used with a higher degree of confidence. Better data resolution can be achieved with finer sampling intervals and should be available whenever detail is necessary.

Rapid data conversion is easily handled by today's microcomputers but doing so without appropriate control will result in the accumulation of a large volume of data. Although computers today have much bigger mass storage capacity than those available a decade ago, it is still a good practice to optimize the sampling rate in order to best represent a continuous log record.

At 10 cm sampling interval, logging a well 3000 m deep will produce 30,000 records. Consider three parameters including the depth to be recorded; if 5 bytes are to be used for each parameter (Microsoft QuickBASIC uses an ASCII character to represent each digit), the file will need 450,000 bytes of computer storage. Files this size can be handled easily by personal computers available today, still such an amount of data is too large. The programs that drive the digital data acquisition system usually offer options on when to read data. With some programming knowhow it is easy to write a code that will enable data gathering at variable depth and time intervals with respect to how data changes. The flexibility offered by the data gathering application should make it possible to exercise control over the data resolution in the depth domain, limiting the size of stored digital files to a more manageable level.

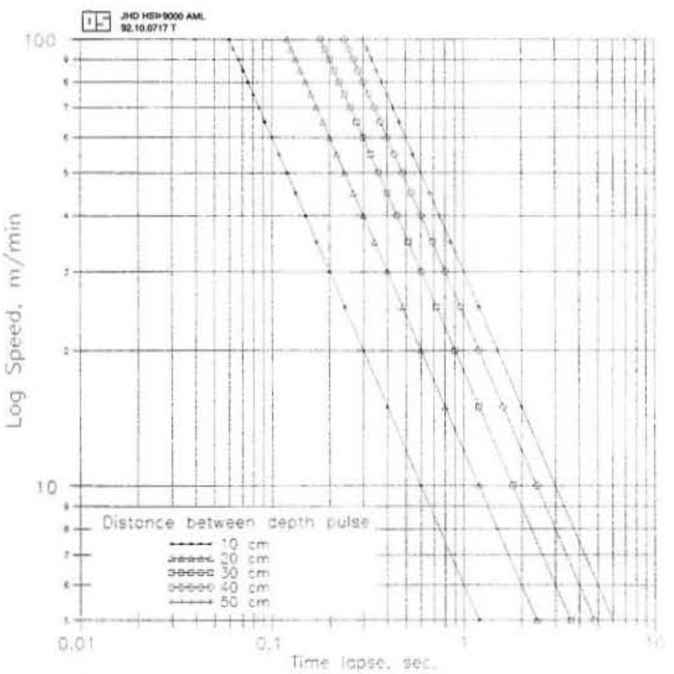


FIGURE 5: Time lapse between sampling at different log speed

3.4 Depth control

Well logging data should be depth controlled to be useful. To control the registration of depth, a depth encoder should be incorporated into the system. In the design envisioned for this data acquisition project, the depth encoder must be able to send depth pulses at fixed interval, preferably every 10 cm. A logic circuit to tell the direction of logging must also be included in the system. The depth encoder discussed in the report by Maceda (1983) can be used in this system.

Depth encoders and logic circuits are commercially available from instrumentation companies. Hence, the principle involved in decoding the up/down logic will not be discussed in detail. Basically, such a depth encoder is a magnetic or optoelectronic device. An optoelectronic device employs infrared lights and photosensitive detectors. A code wheel passing through a slot which houses the infrared emitter on one side and the detector on the other, interrupts the light path. Interruption of the light path causes a signal pulse on the detector. Figure 6a shows an opto-emitter and detector on a single package, Figure 6b shows the code wheel and the two opto-detectors mounted 90 electrical degrees out of phase with each other. The code wheel is mechanically linked to the measuring sheave, in the same line that drives the paper chart of the analog recorder during logging operations. Depth counts can be updated by the data acquisition software every time a depth pulse is detected from the depth encoder.

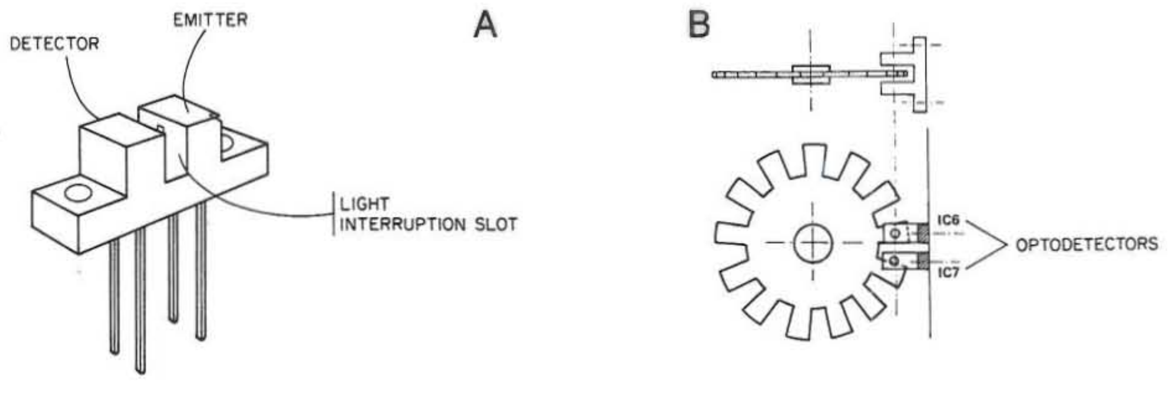


FIGURE 6: A. Opto-emitter and detector on a single package;
B. Code wheel and two opto-detectors mounted 90 electrical degrees out of phase with each other (Maceda, 1983)

4. INTERFACE TYPES

4.1 External bus system

The digital data acquisition system of an external bus set-up has its own internal microprocessor which transmits the data to the host computer via the communication cable. Figure 7 shows a simplified block diagram of an external bus data acquisition system. Parallel interfacing of the data acquisition system and the host computer is possible over a short cable distance. However in most external bus systems, the connection to the host computer is done through the serial port via a standard communication channel such as RS-232, RS-422 or the IEEE-488; these are common protocols for serial communication. Most PCs have one or two serial communication ports which allow transmission of data over long distances. A limitation of serial communication is that data transfer rates should be low, also a serial port can communicate with only one device. RS-232 and RS-422 serial communication protocols are mostly used to interface simple instruments in remote data acquisition subsystems. The IEEE-488 is a standard set by the Institute of Electrical and Electronics Engineers, to interface programmable instruments to computers. It has the advantage of interfacing several instruments together with high transfer rates of data, thus it is mostly used to interface sophisticated data gathering equipment to the host computer.

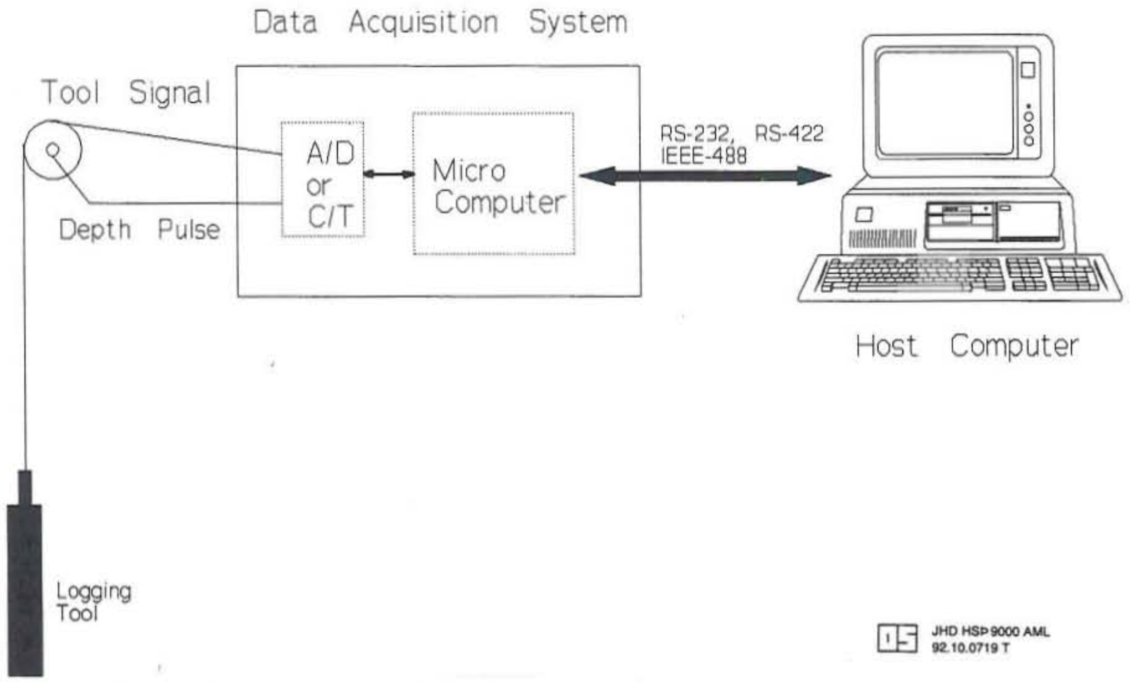


FIGURE 7: Simplified diagram of an external bus data acquisition system

In this system, the local microcomputer helps reduce the load on the host computer (PC) and it also facilitates remote operation. The whole data acquisition system can be packaged separately to include the microcomputer, analog to digital converter and its own power supply. This system can be very powerful and offers the following advantages (Burr-Brown Corp.,1987):

1. The data acquisition system can off-load some of the data collection tasks from the host computer enabling it to do other tasks while the local microcomputer is gathering data.
2. It is easy to configure any size system with this set-up.
3. The data acquisition system can be placed near to the signal source remote from the host computer.
4. The system can be interfaced to virtually any type of computer.

5. A large number of parameters can be monitored from several locations far from each other.

The digital data acquisition system in use by Orkustofnun is of this type. The system makes use of a Rockwell R6502 CPU and a General Purpose Input/Output (GPIO) and Timer Module. The data acquisition system itself is a self-contained unit dedicated to data gathering. Gathered data can be buffered by the system using a 2K RAM, leaving the host computer free to do other jobs while well logging is in progress. Data transfer to the host computer is through a serial communication line using RS-232 protocol. Since data are buffered by the independent data acquisition system, using the slower serial communication line presents no problem to the system.

4.2 Internal bus system

In the internal bus set-up, the data acquisition system is directly connected inside the host computer itself. This set-up is very popular in PC-driven systems since the data acquisition circuitry comes as plug-in boards which can be readily integrated into the PC's system bus. Power for the data acquisition boards come from the PC itself thus the set-up can be very compact. Figure 8 shows a simplified diagram of an internal bus data acquisition set-up.

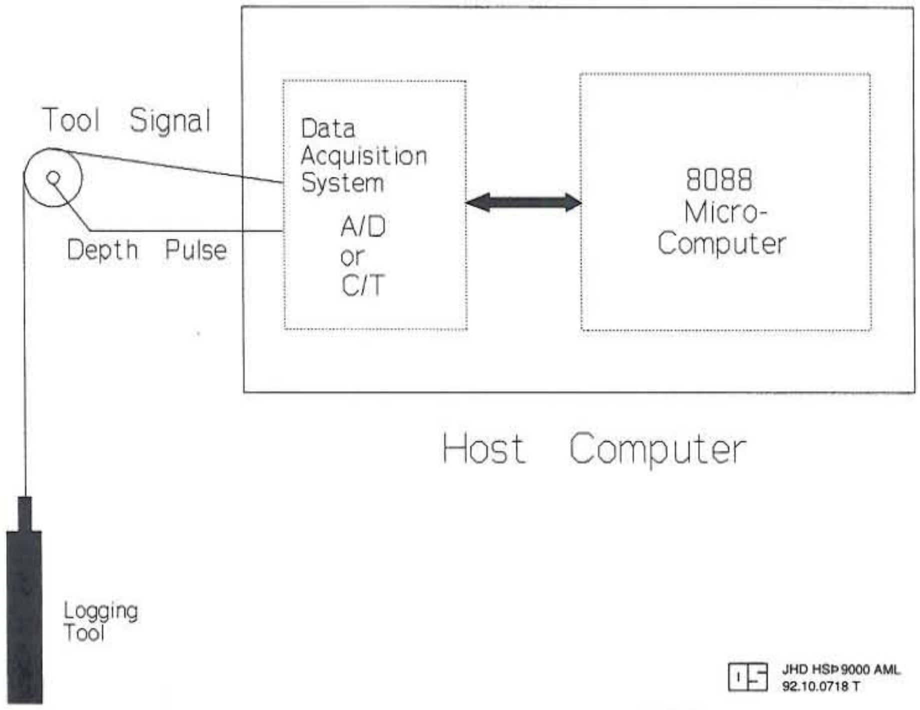


FIGURE 8: Simplified diagram of an internal bus data acquisition system

Because the data acquisition board is directly connected to the PC bus, it offers high speed at relatively lower cost. Some direct PC bus products can take data faster than 100,000 channels per second, whereas data transmitted via the serial port using RS-232 protocol is limited to about 20 analog channels per second. However, the type of PC used in the data acquisition system determines the overall processing speed. High-frequency signals requiring real-time processing will need a high-speed, 32-bit processor with its accompanying co-processor or a dedicated plug-in processor such as a digital signal processing (DSP) board. For slower applications which only acquire and scale data once or twice a second, a low-end PC may be suitable.

The relatively small size of the PC containing the internal bus data acquisition system made it ideal for well logging applications as, in most instances, the well logging truck has very limited working space. As mentioned earlier, counter/timer circuits can be used to digitize the frequency signal sent by the well logging tools. A variety of counter/timer circuits suitable for well logging requirements are available in plug-in boards.

The internal bus digital data acquisition system was chosen as the set-up to be used in this project. With the availability of plug-in boards containing counter/timer circuitry the system can be set up in relatively shorter time and less work is required in interfacing the hardware to the computer. Parameters to be monitored in a well logging application includes the tool signal, the depth pulse and the up/down logic signal of the depth counter which does not require a sophisticated signal monitoring hardware. Figure 9 shows the block diagram of how the well logging tool, hardware and software will interact in the basic design of the system to be built for this project.

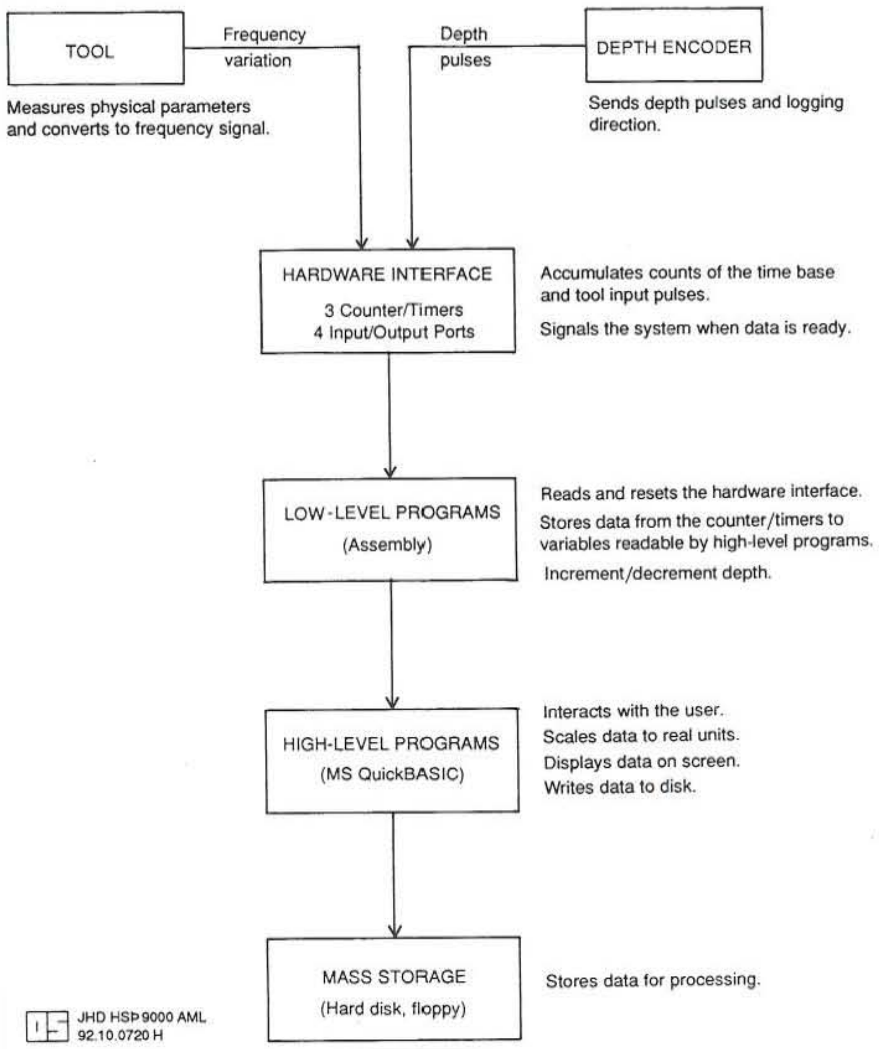

TOOL
Frequency variation
Depth pulses
DEPTH ENCODER
Measures physical parameters and converts to frequency signal.
Sends depth pulses and logging direction.
HARDWARE INTERFACE
3 Counter/Timers
4 Input/Output Ports
Accumulates counts of the time base and tool input pulses.
Signals the system when data is ready.
LOW-LEVEL PROGRAMS
(Assembly)
Reads and resets the hardware interface.
Stores data from the counter/timers to variables readable by high-level programs.
Increment/decrement depth.
HIGH-LEVEL PROGRAMS
(MS QuickBASIC)
Interacts with the user.
Scales data to real units.
Displays data on screen.
Writes data to disk.
MASS STORAGE
(Hard disk, floppy)
Stores data for processing.
JHD HSP 9000 AML
92.10.0720 H

FIGURE 9: Block diagram of hardware/software interaction for the planned digital data acquisition system

5. INTERFACING WITH THE PERSONAL COMPUTER

5.1 The personal computer (PC) system bus

It is now feasible to tailor an efficient solution to unique applications such as data gathering in well logging. However, to take advantage of the power offered by the PCs, some understanding of its architecture and operation is required. There are several architectures associated with personal computers, differentiated mainly by the microcomputer chip used. The IBM PC architecture is widely accepted as de facto standard in most technical and scientific applications (Jiu An et al., 1988). IBM PCs were built around the Intel 8088 microprocessor, later versions use faster microprocessors (80286, 80386 or i486).

Inside the PC enclosure are expansion slots, these are physical and electrical space set aside for the connection of accessory hardware such as a data acquisition board. The expansion slot, also known as the system bus, is an extension of the Intel 8088 microprocessor bus. Through the system bus, the microcomputer can address the plug-in board as either input/output port or as memory locations. Table 3 shows the I/O map of the IBM PC. All signals are transistor-transistor logic (TTL). The slot also provides power (+5 VDC) and ground (GND) for the plug-in boards.

TABLE 3: I/O map in IBM PC

Hex Range	Usage
000 - 00F	DMA controller (8237A-5)
020 - 03F	Interrupt controller (8259A)
040 - 043	Timer (8253)
060 - 063	PPI (8255)
080 - 083	DMA page registers (74LS612)
0Ax	NMI mask register
0Cx	Reserved
0Ex	Reserved
100 - 1FF	Not usable
200 - 20F	Game control (joystick)
210 - 217	Expansion unit
220 - 24F	Reserved
278 - 27F	Reserved
2F0 - 2F7	Reserved
2F8 - 2FF	Serial port (secondary)
300 - 31F	**Prototype card**
320 - 32F	Fixed disk
378 - 37F	Printer
380 - 38C	SDLC communications
380 - 389	Binary synchronous communications (2)
3A0 - 3A9	Binary synchronous communications (1)
3B0 - 3BF	Monochrome adapter/printer
3C0 - 3CF	Reserved
3D0 - 3DF	Colour/graphics adapter
3E0 - 3F7	Reserved
3F0 - 3F7	Diskette
3F8 - 3FF	Serial port (primary)

The PC to be used in interfacing the data acquisition board should have at least one free expansion slot to accommodate the board. This project was designed around an IBM PC compatible. However, the data acquisition system can be used with other PC architecture such as the IBM PC-AT which uses a faster microprocessor. The PC used in testing the project is an IBM compatible with the following configuration:

Processor	8088
Video Adapter	VGA
Mathcoprocessor	8087
Memory (RAM)	640 KB
Clock Speed	10 MHz
Operating System	MS-DOS 5.0
Hard Disk	(1) 20 MB
Floppy Drive	(1) 360 KB

The mathcoprocessor helps in speeding the operation of the system, especially for fast data input rates. A video adapter is needed if the application uses graphics to produce screen display of gathered data. A hard disk is necessary for quick data storage and a floppy drive to be able to take the data out of the host computer for off-site processing or for storage to a database.

5.1.1 Controlling data input

To communicate with the outside world, the PC uses the I/O hardware and its corresponding software to provide the interface to transfer data between the computer and any peripheral device (Jiu An et al., 1988). A data acquisition board once plugged into the PC can be addressed as an I/O board, this serves as the link between the tool sensor (transducer) and the PC. Data transfer can be initiated and controlled by

1 Program-controlled I/O
2. Interrupt-service-routine-controlled I/O
3. Hardware-controlled I/O or Direct Memory Access (DMA)

An important factor considered in designing the system is how to read the data from the board to the computer. The computer must be able to read and process the data as it comes in. Program-controlled input/output are of two types, conditional and unconditional. An unconditional transfer conveys data to or from an I/O port without determining if the port is ready to receive or transmit the data. If the input device sends data at a rate faster than the processor can receive it, data will be lost. On the other hand, redundant data will result if the processor accesses the input device faster than it could provide data. To avoid this problem, the processor can periodically access (poll) a device to determine whether it requires attention, this conditional transfer of data is known as **polling**.

Data transfer using the polling technique requires the processor to check the device periodically by reading one or more status registers. This could be a memory or I/O locations whose values allow the processor to determine whether the device needs servicing. If the device needs attention, a flag is set and an appropriate task (subroutine) is executed. Otherwise, the processor goes around the loop. Computer programs using polling to initiate data transfer are easy to write as they can be written in any programming language. Such programs are relatively easy to understand and debug.

Polling routines are very simple, however, this method has some disadvantages. The processor must be able to execute the entire loop fast enough to be able to keep up with the input/output requirements of the device. Long polling loops may not allow the checking of the timing source often enough to insure adequate data. For the purposes of this project, the system anticipates a pulse every 10 cm from the depth indicator. This means that the reading given by the tool must be read and recorded with the depth. Refer to Figure 5 for a plot of time lapse between sampling at different intervals. Note that increasing logging speed shortens the time lapse between samplings. If the data is to be read, scaled and written to a file, the polling routine must be tight enough to eliminate the possibility of losing data.

On the other hand, the device can be set up to **interrupt** the processor when it needs attention. The interrupt structure is one of the important features of a computer. Its principal task is to provide an efficient way for the microprocessor to respond quickly to unpredictable events . It allows peripheral devices to request service from the microprocessor when they need it instead of requiring the microprocessor to poll the peripheral devices for service requests, thereby increasing the throughput of the computer.

Interrupt processing can service device requests by temporarily halting the current process to execute the interrupt service routine, then it returns to the state previous to the interrupt request. To do this, the microprocessor automatically saves the program counter and the processor status when the interrupt is acknowledged. The interrupt service routine must save all microprocessor registers that will be changed and restore these registers before returning the program counter and processor status to their previous states. The system stack is commonly used for saving the program counter, processor status and registers. This stack is maintained by the stack pointer register for the Intel 8088 microprocessor (Jiu An et al., 1988).

An interrupt handler procedure must be written to make use of interrupts; these handler routines are usually written in assembly language. The address of these procedures is then placed in a special location in memory called the *interrupt vector table* so that it can be executed when an interrupt occurs. The interrupt routine proposed to be used in this project using counter/timers will read the frequency pulse and time counts when an interrupt occurs, then return control to the main program. Scaling the digital counts to real values will be handled by the main program. Programs using interrupt routines increase the efficiency of the processor, however, mistakes in handling the computer's interrupt system can result in program failures which makes debugging the application extremely difficult.

Hardware-controlled input/output or Direct Memory Access (DMA) is a fast and efficient way to get large quantities of data into a personal computer. However, it is rarely used in sensor interfacing because of its limitations. This technique can only transfer one type of data per DMA channel, which limits its versatility and will not be discussed further. It is worth mentioning this technique since the process is being improved to suit data acquisition requirements and may see wider use in the future.

5.1.2 Data storage

Personal computers typically have at least one floppy disk drive to facilitate input and output of data. For the data acquisition system it is recommended that the PC to be used have a hard disk and at least one floppy drive. The data acquisition program that will drive the system can be installed and launched from the hard disk. Writing the digital data to the hard disk presents some advantages, generally the access to the disk is faster than the floppy drives. This helps improve the efficiency of the system. Proper management of the hard disk space will insure there is enough storage space for the gathered data.

The storage capacity of the installed hard disk depends on the user's requirements. Typical hard disk capacities are 40 MB, 80 MB or higher. If the PC will be dedicated to data acquisition, a smaller disk space will suffice. But it is typical for well logging applications to have the data analysis and graphics software on-site at the same computer to facilitate quick data interpretation to help formulate critical decisions whenever necessary. In this case, the storage space left after all the software is loaded must be able to handle the largest possible amount of data to be gathered. If the available storage space is limited, the expected size of the gathered data must first be considered before logging to avoid losing valuable time and data.

One of the advantages of digital data is that it can be stored in a centralized database for later use or reference. This is the reason why a floppy drive is necessary, data can be copied from the system through portable diskettes for storage to a database. This also helps in the maintenance of the disk storage space, since older data that were already transferred to a database can be deleted out of the system. It is sometimes desirable to have most of the data available on site for comparison with previous data, in this case back up data may be kept in diskettes to be used as needed.

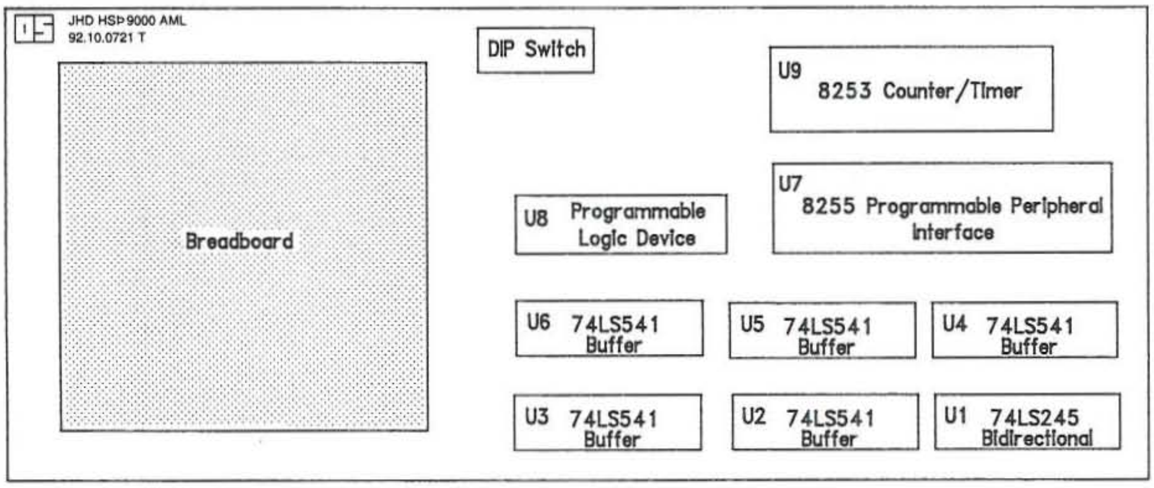


FIGURE 10: Lay-out of the PDS-601 prototype board

5.2 The plug-in data acquisition board

The internal bus plug-in board for the data acquisition system used in this project is a PDS-601 breadboard prototype card (manufactured by JDR Microdevices, San Jose, CA) designed for use in IBM PCs, PC-ATs and compatibles. The PDS-601 makes use of the 8-bit slots found in PCs, ATs and compatibles. The card is designed for a computer hobbyist, thus it affords the user some degree of flexibility to test and design the board to a specific application. The PDS-601 circuitry, as shipped, is composed of nine (9) integrated circuits (IC), a 5-position dual in-line package (DIP) switch, 5 resistors, several capacitors and about 100 signal connection sockets. Figure 10 shows the layout of the board. The ICs on the board are:

one (1) 74LS245 IC labelled U1, this is a bi-directional octal transceiver. It determines the direction of input/output operations and connects the board to the data bus of the PC.

five (5) 74LS541 IC labelled U2 to U6, these are octal buffers (input buffers) connecting the address bus to the board

one (1) Programmable Logic Device labelled U8, which detects the signals that combine to create board selection

one (1) 8255 Programmable Peripheral Interface labelled U7, this has three input/output ports

one (1) 8253 Programmable counter/timer labelled U9, contains three independent software-configurable clocks. They may be set up to send pulses or count events.

The DIP switch on the PDS-601 board is used to set the address of the board so it will be known to the computer during the hardware initialization stage. The 5-position DIP switch was set to 10000 (1=on, 0=off), the 5 bits correspond to pins A4 to A8 of the PC system bus. Address bit 9 is always on, thus the PDS-601 prototype board will be known to the computer as address 300 hex (refer to Table 3 for the I/O map of the IBM PC). The board must be properly wired according to the intended operational logic to make it function as a data gathering system. To complete the system, a 1 MHz crystal oscillator, a NAND gate and a flip-flop were added and wired to the board. Figure 11 shows the wiring diagram of the completed system. It is wired such that it can send interrupt request to the IRQ2 line of the PC bus when either the depth pulse or the timer goes high. For this, an interrupt handler routine is necessary. It is also possible to use a polling routine to check for incoming data, in this case, the interrupt line to IRQ2 must be disconnected if it is used by other peripherals, such as a mouse.

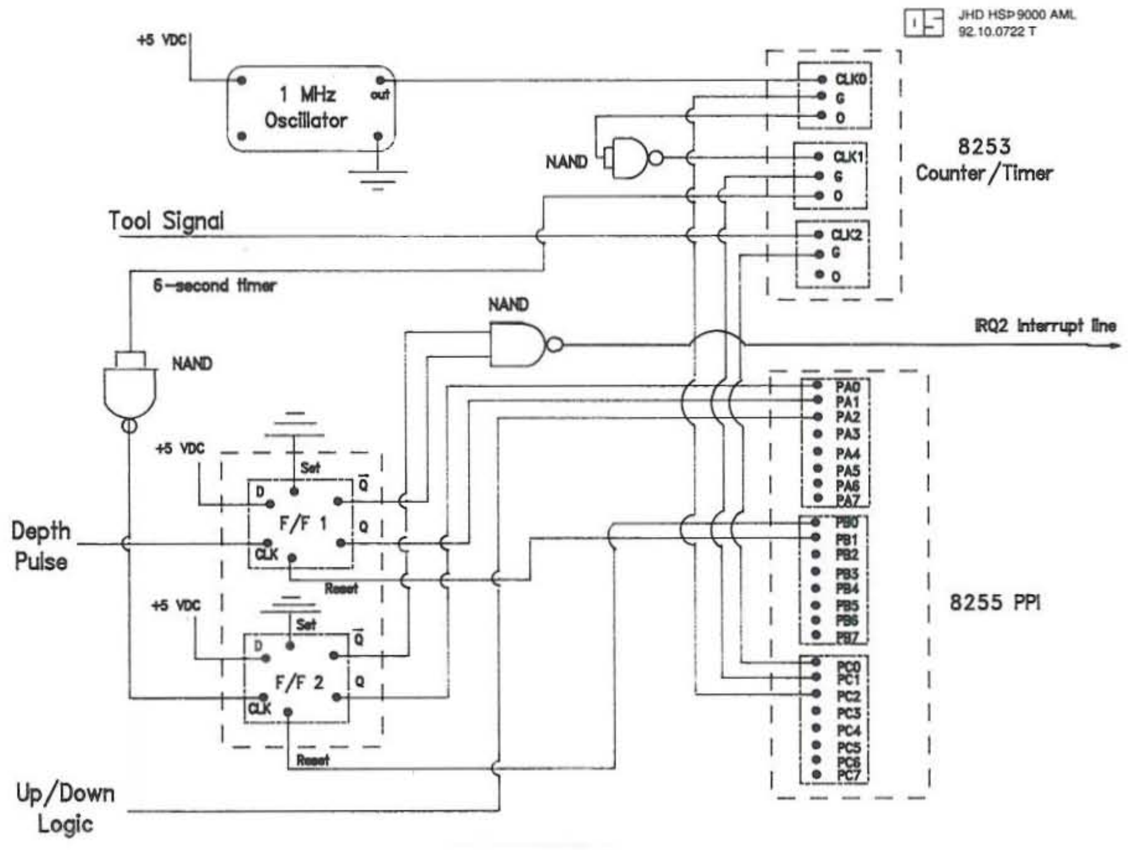


FIGURE 11: Wiring connections of the completed data acquisition card

The design of the system and its basic operation are described as follows:

1. The crystal oscillator which gives a precise square-wave pulse every 1 μsec will serve as the rate generator. This is connected to counter 0 on the 8253 counter/timer
2. Counter 0 is cascaded to counter 1 to produce a 32-bit counter. An initial count of 30,000 will be loaded to counter 0 (8253 counters are down counting) and 200 to counter 1. These counters will be used as the *time base* to compute for frequency. Every time counter 0 rolls over, counter 1 will be decremented by 1, thus the counters are configured to count 6 seconds.
3. The output signal of the logging tool is connected to counter 2. Counter 2 will be used to count the output pulse of the tool; this is the *accumulator*.
4. The depth pulse source is inputted to flip-flop 1 which outputs to pin PA1 of the 8255 PPI and the NAND gate. Every time a depth pulse (every 10 cm) is detected PA1 goes high, and an interrupt request will be sent to the IRQ2 line if it is connected.
5. The up/down logic of the depth signal is connected to pin PA2 of the 8255 PPI. This sets the value of PA2, high (1) means logging up, low (0) means logging down.
6. Every time counter 1 rolls over (6 seconds), it will set pin PA0 to high and send an interrupt request to the IRQ2 line if it is connected. Counter 1 output is connected to flip-flop 2.
7. Flip-flops 1 and 2 are connected to port A of the 8255 PPI (timer=PA0, depth=PA1) and the NAND gate. If connected, the NAND gate sends the interrupt request when either flip-flop goes high.
8. Each time a flag is set or an interrupt request received, the software that drives the system takes over and interacts with the hardware. The following steps take place:
   a) Read and store the value of Port A to a flag variable.
   b) Port C will be loaded with 0, this will simultaneously turn off the gates to all counters. All counters stop.
   c) The computer reads the counter registers, each bit of the counter will have a value of either 1 or 0. The values of counters 0 and 1 (time pulse count) are stored to a variable (*time base*), the reading of counter 2 (tool output count) is also stored to another variable (*accumulator*).
   d) Load high value (hex FF or 11111111 in binary) to Port C, this simultaneously turns on the gates to all counters. All counters reset and restart.
   e) Load high value to Port B then again load Port B with 0. This clears the flip-flops which in turn clears the flag/interrupt line and prepares for the next signal. The value of Port A is now 0.
   f) Evaluate the flag variable to determine from where the signal came from, a high first bit means the signal is from the 6-second timer, a high second bit means the signal is due to the 10-cm depth pulse. The third bit of the flag variable gives the direction of logging.
   g) If the signal is due to depth pulse, the current logging depth is decremented or incremented depending on the value of the third bit of the flag variable.
   h) If the signal is due to the 6-second timer, the current depth is not changed.
   i) Calculate frequency, divide the *accumulator* by the *time base* in seconds, to get frequency in Hz.
   j) Translate the frequency to real value using the tool constant (Equation 2 for temperature logs).
   k) Write depth and the real value (measured parameter) to disk (or memory).
   l) Write data to screen, in numeric format.
   m) Return to the loop to keep on checking the flag.

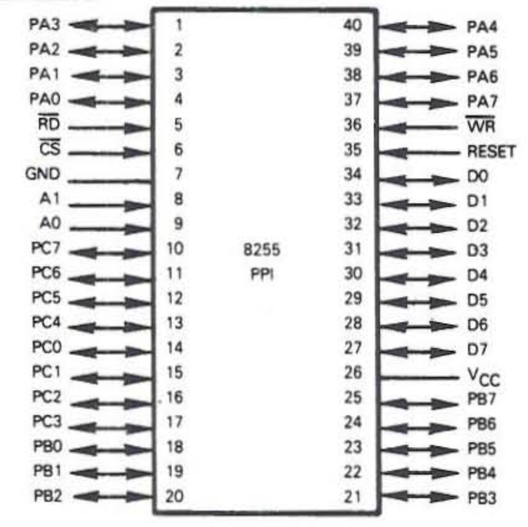
The present design of this data acquisition board does not provide for an analog-to-digital converter, thus this system is not designed for data acquisition in CBL logging. However, it would be a relatively simple task to add a voltage-to-frequency converter or an analog-to-digital converter in the future, when further work is carried out. A description of each of the relevant components of the system will be presented in the following sections.

5.2.1 The Intel 8255 programmable peripheral interface

Programmable I/O ports are the most versatile general-purpose parallel I/O devices. They allow groups of data to be specified as input or output under program control (Kane and Osborne, 1978). The Intel 8255 is a single-chip programmable peripheral interface (PPI) with three programmable I/O ports. Figure 12 shows the block diagram of the Intel 8255 signals and pin assignments. The 8255 PPI is packaged as a 40-pin DIP, it contains a control register and three 8-bit I/O ports named A, B and C. Port C is actually two separately programmable ports; C-upper and C-lower. Port assignment to define each port is written as control word to the control register, it is only possible to write to this register, its contents cannot be read.

When selected, the 8255 PPI appears to the CPU either as four I/O ports or as four memory locations. I/O assignments are made by writing a control word to the control port address (control register). Figure 13 shows how the control word is interpreted. There are three operation modes definable for the 8255 PPI:

1. Mode 0: basic I/O
2. Mode 1: strobed I/O
3. Mode 2: bidirectional bus



PIN NAME	DESCRIPTION	TYPE
D0 - D7	Bidirectional Data Bus	Bidirectional
PA0 - PA7	Eight I/O pins, designated as Port A	Bidirectional
PB0 - PB7	Eight I/O pins, designated as Port B	Bidirectional
PC0 - PC7	Eight I/O pins, designated as Port C upper and Port C lower	Bidirectional
$\overline{RD}$	Read from device control	Input
$\overline{WR}$	Write to device control	Input
RESET	System reset	Input
$\overline{CS}$	Device select	Input
A0,A1	I/O port select	Input
V_{CC},GND	Power and Ground	

FIGURE 12: Block diagram of the Intel 8255 PPI signals and pin assignments (Kane and Osborne, 1978)

Mode 0 provides two 8-bit ports , A and B, and two 4-bit ports, C-upper and C-lower. In this mode, any port can be programmed as an input or an output port. There are 16 possible I/O configurations in this mode.

Mode 1 allows ports A and B to be defined as 8-bit I/O ports, the two 4-bit ports (C-upper and C-lower) provide the handshaking lines for ports A and B. Port C directs the flow of data transfer. In input mode, the processor

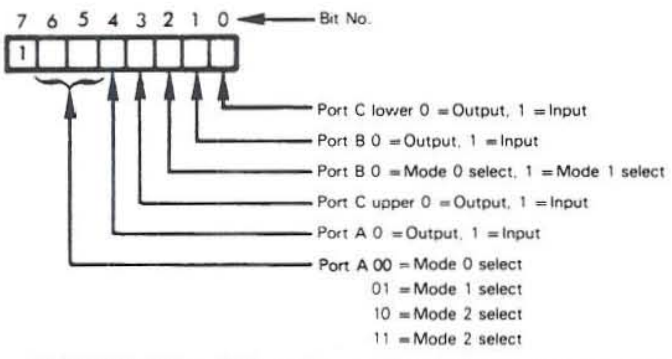

7 6 5 4 3 2 1 0
Bit No.
1
Port C lower 0 = Output, 1 = Input
Port B 0 = Output, 1 = Input
Port B 0 = Mode 0 select, 1 = Mode 1 select
Port C upper 0 = Output, 1 = Input
Port A 0 = Output, 1 = Input
Port A 00 = Mode 0 select
01 = Mode 1 select
10 = Mode 2 select
11 = Mode 2 select

FIGURE 13: Bit assignment of the control word for the Intel 8255 PPI (Kane and Osborne, 1978)

reads port C and determines whether data is available, if it is (logic 1), the processor reads port A or B then resets to signal that data is read. In the output mode, the processor writes data to A or B and the buffer-full flag to indicate data is ready. The output device monitors the flag and acknowledges acceptance of the data by bringing the acknowledge signal to low thus clearing the output buffer-full flag.

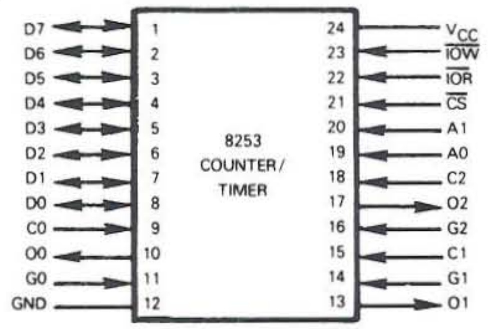
Mode 2 provides a single 8-bit bidirectional bus on port A. Bidirectional handshaking control signals are provided by I/O port C. Bidirectional I/O control signals are a simple combination of the Mode 1 input and output control signals.

Various combinations of mode operation are possible, ports A and C-upper may be used in mode 0 operation while ports B and C-lower may be used in mode 1. The 8255 PPI used in this project is set to Mode 0 operation with port A as input and ports B and C as output.

5.2.2 The Intel 8253 programmable counter/timer

The Intel 8253 chip is a programmable device which contains three sets of timing logic. Each set of timing logic can be programmed independently as an interval timer, a one-shot, a clock signal generator or an event counter; the 8253 can be used with any microprocessor (Kane and Osborne, 1978). This counter/timer is a very significant device in any digital logic oriented microcomputer application. Figure 14 shows the 8253 counter/timer signals and pin assignments, all pins are TTL-level compatible.

The 8253 has three independent 16-bit down counters. Each counter has three control signals, a clock (C), output (O) and gate (G). A diagram is shown on Figure 15. The counter will decrement by one on each falling edge of the clock signal C. The signal maybe synchronous, in which case it functions as a traditional clock signal and the timer is being used to measure time intervals. If the clock signal is asynchronous, then the timer is functioning as an event counter, that is, it counts the number of times the clock signal input has made a high-to-low transition. When the counter counts out, it gives a signal via the output signal O, which may be used as a one-shot pulse, interrupt request or simple control signal.



PIN NAME	DESCRIPTION	TYPE
D0 - D7	Data Bus	Tristate, Bidirectional
C0	Timer 0 clock	Input
O0	Timer 0 output	Output
G0	Timer 0 gate	Input
C1	Timer 1 clock	Input
O1	Timer 1 output	Output
G1	Timer 1 gate	Input
C2	Timer 2 clock	Input
O2	Timer 2 output	Output
G2	Timer 2 gate	Input
A0, A1	Register select	Input
$\overline{CS}$	Chip select	Input
$\overline{IOR}$	Read control	Input
$\overline{IOW}$	Write control	Input
V_{CC}, GND	Power and Ground	

FIGURE 14: The Intel 8253 counter/timer signals and pin assignments (Kane and Osborne, 1978)

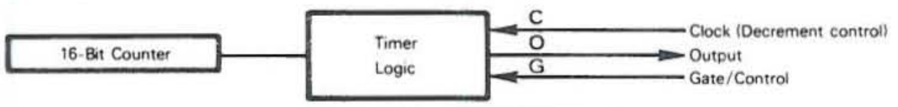
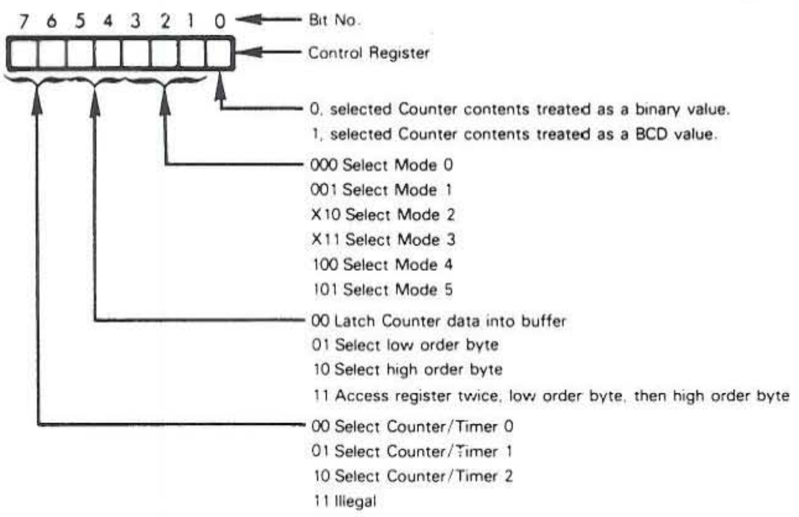


FIGURE 15: Control signals of a 16-bit counter in the Intel 8253 chip (Kane and Osborne, 1978)



7 6 5 4 3 2 1 0
Bit No.
Control Register
0, selected Counter contents treated as a binary value.
1, selected Counter contents treated as a BCD value.
000 Select Mode 0
001 Select Mode 1
X10 Select Mode 2
X11 Select Mode 3
100 Select Mode 4
101 Select Mode 5
00 Latch Counter data into buffer
01 Select low order byte
10 Select high order byte
11 Access register twice, low order byte, then high order byte
00 Select Counter/Timer 0
01 Select Counter/Timer 1
10 Select Counter/Timer 2
11 Illegal

FIGURE 16: Bit assignment of the control word for the Intel 8253 counter/timer (Kane and Osborne, 1978)

Clocks of any frequency up to 2 MHz may be used with the 8253, such that it can be used as an event counter. The triggering may be done by either hardware or software. This makes the 8253 an effective counter of frequency pulses from outside sources. The output may be used to cause interrupts, be polled or clock other devices. The initial count and mode of operation may be loaded by the user through software control. To initialize the counter/timer, a control word must be written to the control register. Figure 16 shows the bit assignments of the control word that must be sent to the 8253 to initialize a specific counter. Bits 6 and 7, the two most significant bits, designate the counter to be set.

The counters may be preset to count in either binary or BCD format. Also, an initial value may be loaded from which a counter can count down. There are six different modes of operation to which the 8253 can be set, as summarized below (Jiu An et al., 1988):

Mode 0. Upon terminal count the output goes high. A low level on the gate stops counting and a high level enables counting.

Mode 1. This is a programmable one-shot. The output is low only while the clock is counting. A rising edge on the gate enables counting and resets the count after the next clock.

Mode 2. This mode produces a string of pulses. The output is high while the clock is counting and goes low for one clock cycle upon terminal count. This process repeats itself as long as the gate is high. A low level on the gate disables counting and forces the output high if it is not already. A rising edge on the gate initiates counting. This is the mode used for all the counters in this data acquisition project. The mode of operation is illustrated in Figure 17.

Mode 3. This mode is similar to mode 2, but a square wave is produced at the output. If the count *n* is even, the output is high for *n*/2 clock cycles and low for *n*/2 clock cycles. If the count is odd, the output is high for $(n + 1)/2$ counts and low for $(n - 1)/2$ counts. The gating is the same as for mode 2.

Mode 4. The output is normally high in this mode and it goes low for one clock cycle upon terminal count. A low on the gate disables counting and a high enables counting.

Mode 5. The output is the same as mode 4 except that the counting is enabled on the rising edge of the gate.

A 16-bit counter can count up to 65,536 events or pulses before it rolls over. However, one or more counters may be cascaded to form a 32-bit counter for counting requirements in excess of 65,536. In this project, two counters of the 8253 chip were cascaded to count up to 6 million microseconds (6 seconds). This will be used as a time base in converting the frequency pulses to digital data. The other counter was used to count the frequency pulses coming from the tool.

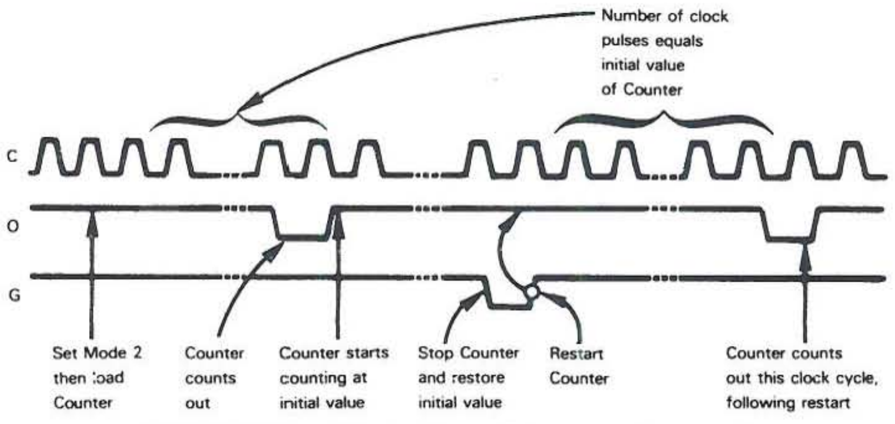

Number of clock pulses equals initial value of Counter
C
O
G
Set Mode 2 then load Counter
Counter counts out
Counter starts counting at initial value
Stop Counter and restore initial value
Restart Counter
Counter counts out this clock cycle, following restart

FIGURE 17: Illustration of the operation mode for all counters used (Kane and Osborne, 1978

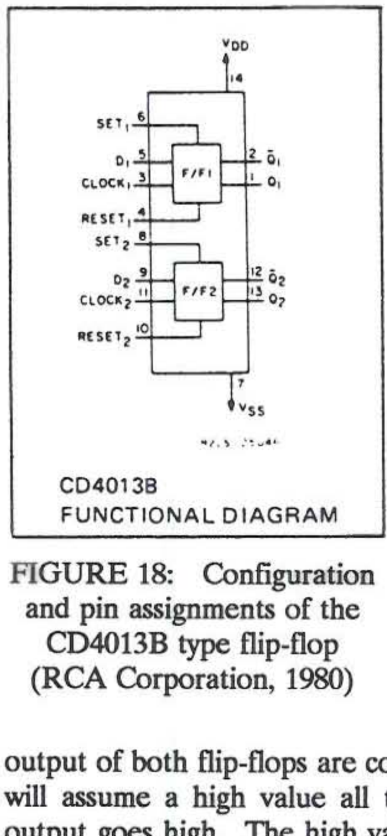


FIGURE 18: Configuration and pin assignments of the CD4013B type flip-flop (RCA Corporation, 1980)

5.2.3 The flip-flop

The flip-flop used is a CD4013B dual D-type flip-flop. Flip-flops are used for the temporary storage of data and in shift registers (Loveday, 1986). Figure 18 shows the configuration and pin assignment of the CD4013B type flip-flop. It has one input called D and the state of the data on D is transferred to the Q output when the clock goes high. The flip-flop will be used to detect the 10 cm depth pulse. As shown on Figure 11, the D-pin of both flip-flops are connected to the +5 VDC line, so it will always have a high value. The depth pulse is connected to the clock (CLK) pin of flip-flop 1 and the counter 1 output to CLK pin of flip-flop 2. Every time a pulse is detected a high value will be output to Q on either flip-flop. $\overline{Q}$ is the complement output, and will assume the opposite value of Q. The $\overline{Q}$ output of both flip-flops are connected to the NAND gate and will assume a high value all the time except when either Q output goes high. The high value of Q is used to set the flag, while $\overline{Q}$ triggers the interrupt signal. Both flag and interrupt are reset by the serving routine.

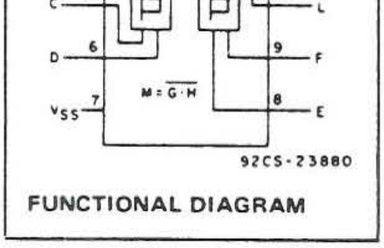

$J = \overline{A \cdot B}$
$K = \overline{C \cdot D}$
$L = \overline{E \cdot F}$
$M = \overline{G \cdot H}$
VDD
VSS
92CS-23880
FUNCTIONAL DIAGRAM

FIGURE 19: Configuration and pin assignments of the CD4093B type NAND gate (RCA Corporation, 1980)

5.2.4 The NAND gate

The CD4093B type NAND gate contains four 2-input Schmitt trigger circuits. Figure 19 shows the configuration and pin assignments of the NAND gate. This gate allows the transmission of a signal when an appropriate switching input is applied. The type of gate gets its name from the logic used to determine its output. NAND means Not AND from Boolean algebraic logic. A NAND gate will give a logic 0 output only when all of its inputs are at logic 1 (high). Note from Figure 11 that the NAND gate gets its input from the $\overline{Q}$ output of the flip-flops. This will force the NAND gate to take a value of 0 except when any of the flip-flops detect a pulse. In which case, one of the inputs to the NAND will go low, causing the NAND gate to output high giving an interrupt signal. The NAND gate output is connected to the interrupt line (IRQ2) of the PC.

5.3 Data capture by polling routine

A program was written by the author to capture data from the data acquisition system employing a polling routine (a complete listing of the program is found in Appendix A). The program was written entirely in Microsoft QuickBASIC (1988). This programming language is relatively easy to use and affords the user sufficient means to control and access the PDS-601 prototype card.

The data acquisition program was structured such that it executes from top to bottom, making it easy to understand and debug. Figure 20 shows a generalized flow chart of the program featuring the subroutines used. The actual polling loop reading port A of the 8255 PPI, which serves as the flag, is located in subroutine LOGGER. A detailed flowchart of subroutine LOGGER is shown on Figure 21.

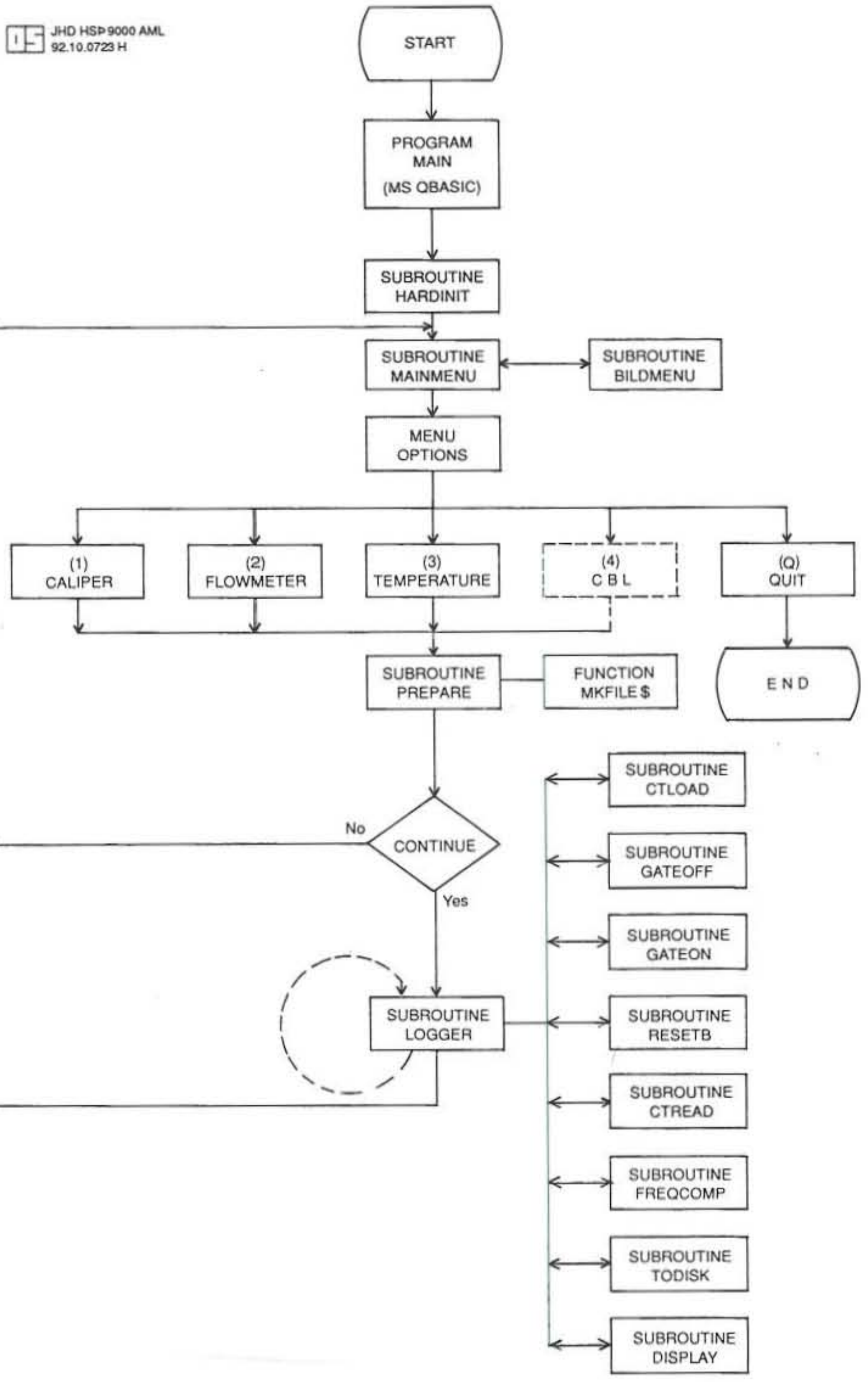


FIGURE 20: Generalized flowchart of the computer program for data acquisition using a polling routine

JHD HSÞ 9000 AML
92.10.0724 H

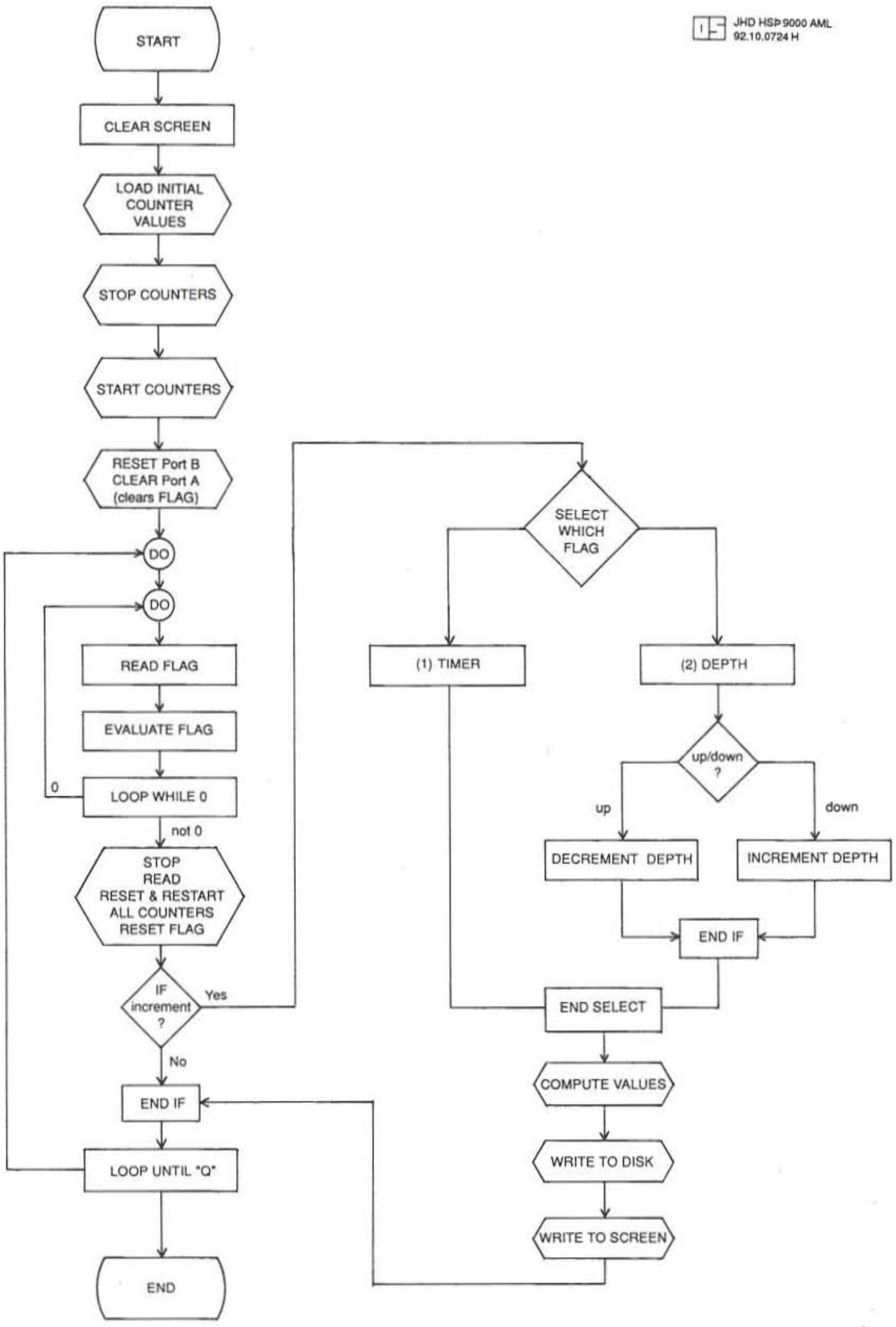


FIGURE 21: Detailed flowchart of subroutine LOGGER

The data gathering process, that is, reading the time base and accumulator is triggered by the depth pulse or whenever the timer counts 6 seconds (for stationary logs). Every time a flag is detected (high value in port A) the counters are simultaneously stopped. After reading the counts, the counters are then simultaneously reset and restarted by deactivating and activating a gate. All these processes are under program control. Incrementing or decrementing the depth is also done by the software.

Gathered data are displayed on the screen as numbers. No graphics routine was written to display the data graphically on the screen as this will slow down the processor thus extending the time needed by the routine to return to the polling loop. The polling loop was designed to be as short as possible.

5.4 Data capture using interrupt

A code to handle interrupt from the PDS-601 data acquisition board was written (by Josef Holmjarn) using Assembly language. A complete listing of the interrupt code written in Assembly language is found in Appendix B. Interrupts can be triggered by the depth pulse or the 6-second timer. Upon interrupt signal from the data acquisition system, whatever program is running saves what it is currently doing. Then counters are stopped, read and restarted, then the program returns to its previous state. Data read from the counters are stored in registers that can be accessed by the complementary program that drives the system. The codes to actually process the data can be written in any high-level language such as QuickBASIC.

Codes written in Assembly are quite fast and use very little computer time, thus speeding up the process. Reading and resetting the timers can be done by the interrupt handler routine leaving the computer free to do other things. However, interrupt routines can be difficult to write and unpredictable results can happen to the computer system when interrupts are not properly handled. The use of the interrupt routine is necessary for rapid data gathering while logging at a speed in excess of what the polling routine can handle. Appendix C lists a program code written in Microsoft QuickBASIC. This code was used to test the data acquisition system in conjunction with the interrupt handler listed in Appendix B.

5.5 Plotting the data

The output of the data gathering program is written to a file in a comma-delimited ASCII format and can be exported to most graphics programs. The data can be plotted using commercially available graphics software, thus no plotting program was written. A graphics software called GRAPHER (from Golden Software Inc., CO) is used by Orkustofnun to plot log data; the same software is used in PNOC-EDC.

6. TESTING THE SYSTEM

6.1 Simulated tests

A simulated test was done to determine the validity of the logic used in wiring the system. The tests also afforded the chance to debug the software that will drive the system before it is used in actual well logging. The tests were done using rate generators to simulate the pulses given by the tool and the depth encoder. The rate generators are capable of supplying variable frequency to the system, thus a wide range of frequency may be used. It is also possible to test the system at different simulated logging speeds by varying the pulse rate of the signal connected to the flip-flop which senses the depth pulse. Logging direction can be set and changed by a switch on the control panel of the testing circuit. The simulated test was done for temperature logging using the frequency-to-temperature relationship shown in Equation 2.

6.2 Actual field tests

Actual temperature logging to test the completed data acquisition system was conducted at well R-33, a low-temperature geothermal well outside of Reykjavik. The electronic well logging unit and the Gearhart-Owen temperature tool of Orkustofnun were used in the field test of the system. The hardware interface worked as intended; no changes in the original wiring of the system as shown in Figure 11 were necessary. The inputs were filtered before being fed into the system. This is necessary to protect the system and to ensure smooth input signals. The newly completed data acquisition system was connected to the same lines that feed information to Orkustofnun's data gathering system. Thus, it is possible for the two systems to log simultaneously using the same inputs. This provided the opportunity to compare the results to be obtained from the newly completed system to that of Orkustofnun's. The results from Orkustofnun's system will be used as base data.

6.3 Test results

The preliminary tests verified the validity and the applicability of the logic used in designing the data acquisition system using counter/timers. By inserting time measurement control in the polling program used in the simulated data acquisition tests, the time the loop takes to read, scale and write the data to memory was determined to be 0.06 sec. The data acquisition loop is fast enough and may be used in logging speeds of 60 m/min at 10-cm sampling interval without losing data. Faster logging speeds mean shorter time for the software to return to the loop that checks the flag. If the software misses a depth flag then a reading will be missed and the depth will be offset by one interval (10 cm) because incrementing/decrementing the depth is done by the software. The tests also showed that the ± 1 count error discussed in Section 3.2 of this report becomes pronounced at higher logging speed. At the present set-up, no correction for the ± 1 count error is applied. A plot of the gathered data in testing the system using simulated signals is shown in Figure 22. The graphics software GRAPHER was used to plot the data. The data was gathered by varying the frequency input to the system at a simulated speed of 60 m/min.

In the actual field tests of the system, both the polling program and the interrupt handler routine used to drive the system were tested. The system was tested up to a logging speed of 160 m/min; the hardware interface coped with the measurements and no problems concerning the hardware were observed. Figure 23 shows the plots of the temperature logs gathered from well RV-33 during the actual field tests and Figure 24 an enlargement of a small part of Figure 23. Figures

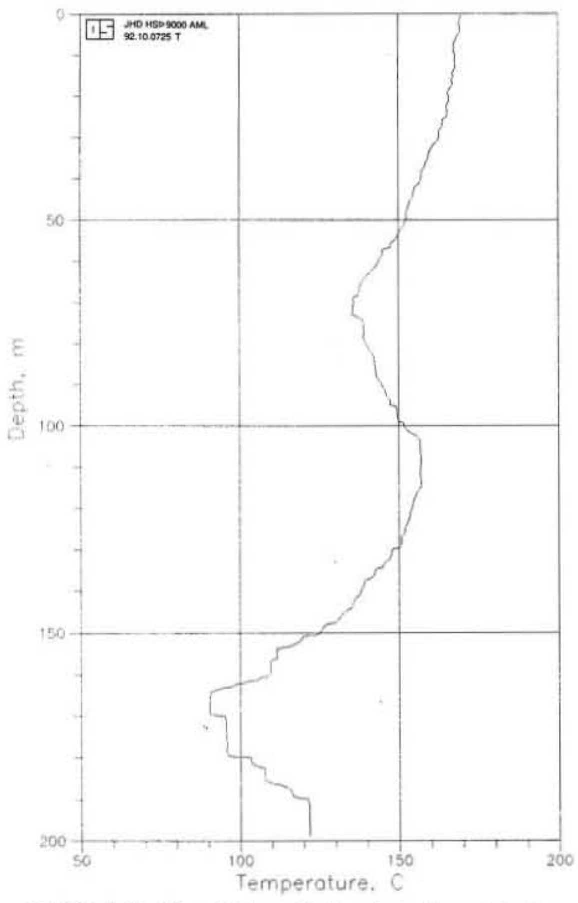


FIGURE 22: Plot of simulated test data

23a and 24a show a plot of data recorded using Orkustofnun's digital data gathering system. This will be used as base data for comparison. Figure 23b is a plot of the temperature data recorded on the system being tested using an interrupt handler routine. The data were gathered every 10 cm while logging at 30 m/min. The results compare very well with Figure 23a, however, there is a noticeable jitter in the data as plotted in Figure 23b, as Figure 24b shows clearly. This is due to the ±1 count error. The jitter is reduced at lower logging speeds or at longer sampling intervals. Figures 23c and 24c show a plot of temperature data gathered using the polling routine. The data were recorded at 50-cm interval while logging at an average speed of 30 m/min. In this case the jitter was reduced. The results of the tests plotted in Figures 23b and 23c compare very well with the base data plotted on Figure 23a. The tests showed that the effect of the ±1 count error can be reduced without significant loss of detail by gathering data at longer intervals. Whenever detail is necessary, higher accuracy is obtainable by logging slowly.

From the tests, it was observed that the time (0.06 sec) the loop takes can actually be used in logging speeds of 90 m/min. However, most PCs have

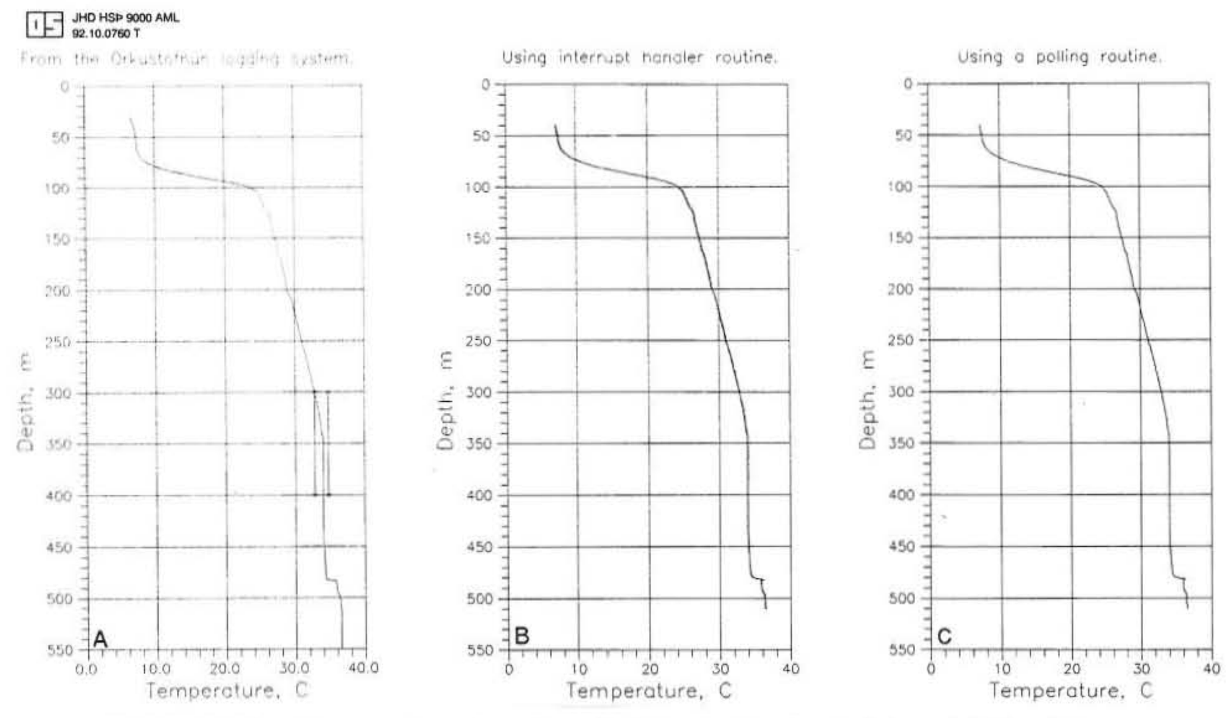


FIGURE 23: Temperature logs from field tests; the depth interval 300 - 400 m, marked on A, is for all tests enlarged in Figure 24

built-in internal interrupts like the system clock and the keyboard which may at times slow down the system. Accessing the disk to write the gathered data extends the time necessary for the program to return to the loop. This may cause loss of data. A way to get around this problem is to specify a larger file buffer in opening the output file on the polling program. When using the interrupt handler routine, a small buffer specified on the low-level program can be used to hold the data while the main program is accessing the disk; this will prevent loss of data. Currently, the polling program can handle logging speeds higher than 60 m/min, reading data every 50 cm for 1000 m depth without losing data. For logging applications requiring a higher speed, a program using an interrupt handler should be used. Tests of the QuickBASIC program listed in Appendix C which makes use of the interrupt handler listed in Appendix B showed that a logging speed of 160 m/min is attainable without losing data. This is far in excess of most well logging requirements. For applications in well logging, a data gathering software using an interrupt handler routine offers more flexibility.

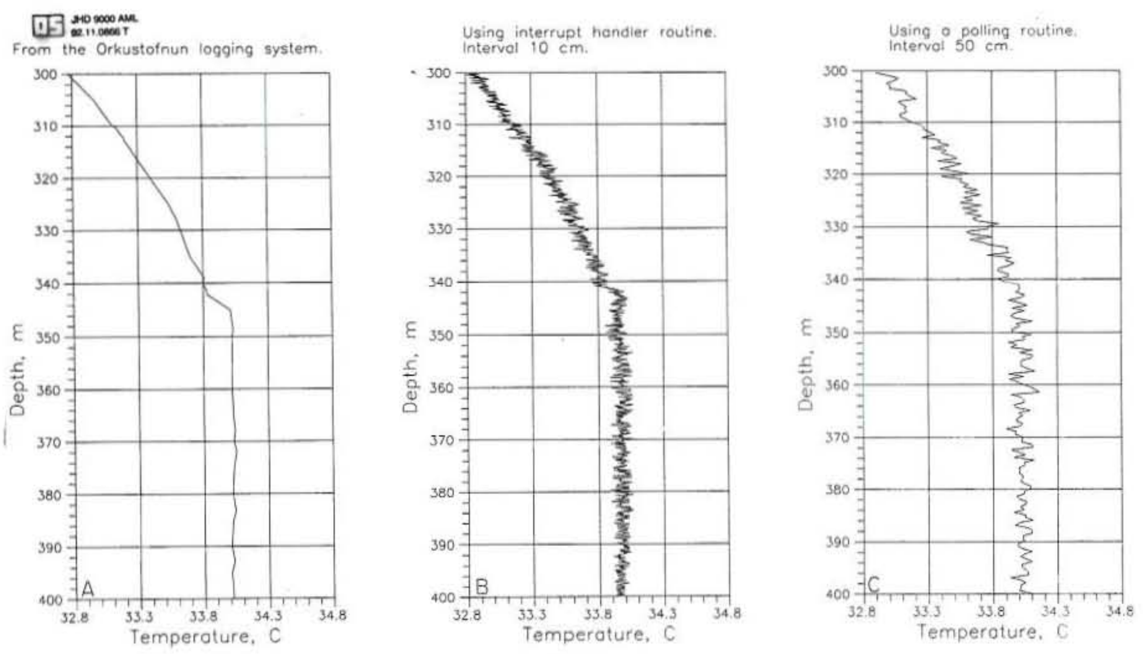


FIGURE 24: Temperature logs from field tests, enlarged for the depth interval 300-400 m

7. CONCLUSIONS

The power and versatility offered by personal computers simplified the process of digital data acquisition to a large extent. Interfacing sensors to the PC's can be done by using the correct hardware interface. Using the internal bus system of data acquisition for well logging has some advantages over the external bus system. The plug-in boards used in the internal bus system are easy to install, compact, require less work and in most cases are relatively cheaper. For the purposes of well logging using the tools mentioned in this report, the applicability of counter timer circuitry is well suited.

The prototype board used in this study may be used as a jumping point to a fully workable digital data acquisition system suitable for well logging requirements. In this design, the three counter/timer circuits in the prototype board are fully utilized leaving no room for expansion. A board with more counter/timer is preferable if further expansion to the system is anticipated. Computer software to control the data acquisition system using counter/timers and polling loops is relatively easy to write and maintain. The system is working as intended, but due to the limited time in the preparation of this project, the software and the hardware used are not thoroughly tested. Further improvements to the system must be done by running more comprehensive tests to weed out all possible problems. However, the exercise of designing and building this digital data acquisition system prepared the author to undertake a similar project for PNOC-EDC.

There is a wide variety of data acquisition boards featuring counter/timer circuits commercially available from instrumentation companies at affordable prices. Such as DT2819 from Data Translation, Inc., MCN-8 from Keithley Metrabyte and PC-TIO-10 from National Instruments, Inc. These data acquisition systems usually come in complete packages with the necessary computer software to drive and tailor the system to one's requirements. This is an alternative to building a complete data acquisition system from scratch.

ACKNOWLEDGMENTS

I wish to extend my sincerest gratitude to everybody who made this training an enjoyable learning experience. Most particularly to my key adviser, Josef Holmjarn, without whose guidance and patient attention, this project could not have been as successful as it was. He was the driving force behind this endeavour. Thank you Josef, for seeing me through this project. My thanks also to Hilmar Sigvaldason, for reviewing this paper and for his valuable advice and suggestions towards its improvement.

My thanks to the director of the UNU Geothermal Training Program, Dr. Ingvar B. Fridleifsson, for affording me the opportunity to undertake this training. It was such a thoroughly planned, well-executed program, worth all the time. Thanks also to Ludvik S. Georgsson for his attention and expert organization of the day-to-day activities, but most of all, thank you, Ludvik for making our life away from home a lot more bearable.

I would like to give credit to all our lecturers, especially the nice people of the Reservoir Engineering Section of Orkustofnun. Those lectures will surely be remembered and put to good use. Special thanks to Benedikt Steingrimsson who has been so helpful and accommodating.

Back home, I extend my gratitude to Zammy Sarmiento and Bing Buñing of the PNOC-EDC Reservoir Engineering Section for their support and encouragement. Lastly, I would like to give due credit to my wife Flor, who, after giving birth to our son Archibald, while I was attending this course, still found the will to encourage me to finish what I had set out to do.

REFERENCES

Baker, L.E., Baker, R.P., and Hughen, R.L., 1975: Report of the geophysical measurements in geothermal wells workshop. Sandia Laboratories, USA, report SAND 75-0608, 76 pp.

Burr-Brown Corp., 1987: The handbook of personal computer instrumentation. Burr-Brown Corp. and Intelligent Instrumentation Inc., Tuczon, Arizona, 1-1 to 9-32.

Fertl, W., and Overton, H., 1982: Formation Evaluation. In: Edwards, L.M., Chilingar, G.V., Rieke III, H.H., and Fertl, W.H.: Handbook of geothermal energy. Gulf Publishing Co., Houston, Texas, 326-425.

Hewlett-Packard, 1978: Fundamentals of the electronic counters; Application note 200. Hewlett-Packard, CA, USA, 44 pp.

Jiu An, Doumas, T.A., and Yorkey, T.J., 1988: Interfacing to the IBM PC bus. In: Tompkins, W.J., and Webster, J.G. (editors): Interfacing sensors to the IBM PC. Prentice-Hall, New Jersey, 59-106.

Kane, J., and Osborne, A., 1978: An introduction to microcomputers, Vol. 3. Osborne and Associates Inc., Berkeley, CA, B3-1 to B3-10, F2-1 to F2-10.

Loveday, G., 1986: Electronics sourcebook for engineers. Pitman Publishing, London, 293 pp.

Maceda, N.S., 1983: On digital data acquisition and processing in geothermal well logging. UNU Geothermal Training Program, Iceland, Report 11, 80 pp.

Microsoft Corp., 1988: Microsoft QuickBASIC - Programming in BASIC. Microsoft Corp., USA, 458 pp.

RCA Corporation, 1980: COS/MOS integrated circuits. RCA Corporation, USA, 688 pp.

Stefansson, V., and Steingrimsson, B., 1990: Geothermal logging 1; An introduction to techniques and interpretation. Orkustofnun, Iceland, report OS-80017/JHD-09, Second edition, 117 pp.

Zuch, E.L., 1980: Data acquisition and conversion handbook. Datel-Intersil Inc., Massachusetts, 242 pp.

```
' Program MAIN for digital data acquisition written entirely in Microsoft QuickBASIC
' by A. M. Lacanilao - September 24, 1992
' UNU-GTP Reykjavik, Iceland
'********************************************
DECLARE SUB FREQCOMP ()
DECLARE SUB MAINMENU ()
DECLARE SUB HARDINIT ()
DECLARE SUB CTREAD ()
DECLARE SUB CTLOAD ()
DECLARE SUB GATEOFF ()
DECLARE SUB GATEON ()
DECLARE SUB RESETB ()
DECLARE SUB DISPLAY ()
DECLARE SUB TODISK ()
DECLARE SUB LOGGER ()
DECLARE SUB BILDMENU ()
DECLARE SUB PREPARE (log$, code$)
DECLARE FUNCTION MKFILE$ (wn$)

DEFINT A-Z
COMMON SHARED depth AS SINGLE, value AS SINGLE, FREQ AS SINGLE
COMMON SHARED tcount AS LONG, pcount AS LONG
COMMON SHARED time AS LONG, pulse AS LONG
COMMON SHARED PortA, PortB, PortC, PortR, CT0, CT1, CT2, CTReg AS INTEGER
COMMON SHARED kount, inc AS INTEGER
COMMON SHARED choice AS STRING

TYPE ltype                    'Data for types of log
   label AS STRING * 20
   code AS STRING * 3
END TYPE

HARDINIT             'call hardware initialization routine

READ num
DIM logs(1 TO num) AS ltype
kount = 0
FOR j = 1 TO num                  'Read log type data
 READ logs(j).label, logs(j).code
NEXT j

MAINMENU             'call Main Menu routine

END           'End MAIN

'This following data number should equal log type count
'To add items to main menu, edit these DATA lines
DATA 4
DATA "Caliper Log","CAL","Flowmeter  Log","FM","Temperature Log","TMP"
DATA "CBL Log","CBL"
```

```
' This procedure will build the Main Menu
SUB BILDMENU
  SHARED logs() AS ltype
  CLS
  LOCATE 6, 17: COLOR 1, 15
  PRINT "WELL LOGGING - DIGITAL DATA ACQUISITION"
  LOCATE 7, 17: COLOR 15, 0
  PRINT "=============MAIN MENU==============="
  row = 9
  col = 28
  FOR i = 1 TO UBOUND(logs)                   'Build Menu
    COLOR 15: LOCATE row, col
    PRINT i
    COLOR 7: LOCATE row, col + 4
    PRINT logs(i).label
    row = row + 1
  NEXT i
  COLOR 4, 15: LOCATE row + 1, col + 1
  PRINT "Q"
  COLOR 15, 0: LOCATE row + 1, col + 4
  PRINT "QUIT"
  COLOR 7: LOCATE row + 3, col
  PRINT "Select Option ==> "
  LOCATE row + 3, col + 19
END SUB


'Function to load all counters  with initial values
SUB CTLOAD
 OUT CT0, &H30        'load counter 0 LSB, load 30,000
 OUT CT0, &H75        'load counter 0 MSB
 OUT CT1, &HC8        'load counter 1 LSB, load 200
 OUT CT2, &HFF        'load counter 2 LSB, load 65,535
 OUT CT2, &HFF        'load counter 2 MSB
END SUB


'Subroutine to read and reset counters
SUB CTREAD
' following commands store counter values to variables
  GATEOFF             'freeze counters
  low0% = INP(CT0)     'read counter 0 LSB to variable
  high0% = INP(CT0)    'read counter 0 MSB to variable
  low1% = INP(CT1)     'read counter 1 LSB to variable
  low2% = INP(CT2)     'read counter 2 LSB to variable
  high2% = INP(CT2)    'read counter 2 MSB to variable
'following commands reset and restart counters
  GATEON
  RESETB
'evaluate counts
  tc0& = 30000 - ((high0% * 256&) + low0%)
  tc1& = (201& - low1%) * 30000&
  tcount& = tc1& + tc0&
  pcount& = 65536 - (high2% * 256& + low2%)
END SUB
```

```
'Subroutine to write readings on screen while logging
SUB DISPLAY
  LOCATE 10, 1
  PRINT depth!, CINT(value! * 10) / 10
  PRINT time&, pulse&
  'PRINT CINT(inc * 1000000 / time& * 60) / 10
  time& = 0              'reset counts
  pulse& = 0
END SUB


'Subroutine to compute for frequency
SUB FREQCOMP
   IF pulse& = 0 THEN
     FREQ! = 0
   ELSE
     FREQ! = pulse& / time& * 1000000
   END IF

   SELECT CASE choice$

     CASE "1"        'caliper logging
     'substitute the correct frequency-holesize relationship here
     value! = FREQ! / 200     'this equation is just for the purpose
                              'of testing the system
     CASE "2"        'flowmeter logging
     value! = FREQ!           'flowmeter readings are usually given in
                              'terms of frequency
     CASE "3"        'temperature logging
     value! = .02777778# * FREQ! - 17.777778#   'equation in use by Orkustofnun

     CASE ELSE
      PRINT "Error!  Check program logic."
   END SELECT
END SUB


'Subroutine to disable and reset all counters to low, reset count
SUB GATEOFF
 OUT PortC, &H0         'set gates of all counters to low
END SUB


SUB GATEON
'Subroutine to enable all counters, set to high
  OUT PortC, &HFF       'set gate of all counters to high
END SUB


' Hardware initialization routine
' Variable names are common to MAIN
SUB HARDINIT
' Variable names used for 8255 PPI ports
```

```
  PortA = &H310
  PortB = PortA + 1
  PortC = PortA + 2
  PortR = PortA + 3          'PIO control register
' Variable names used for 8253 counter/timer
  CT0 = &H314
  CT1 = CT0 + 1
  CT2 = CT0 + 2
  CTReg = CT0 + 3            'C/T control register
' Load control words to registers
  OUT PortR, &H90       'control word 10010000
                        'A=input, B=output, C=output
  OUT CTReg, &H34       'Counter 0=mode 2, binary counting, 00110100 2 regs
  OUT CTReg, &H54       'Counter 1=mode 2, binary counting, 01010100 1 reg
  OUT CTReg, &HB4       'Counter 2=mode 2, binary counting, 10110100 2 regs
END SUB


' This procedure sets the stage for logging by polling
SUB LOGGER
 CLS
LOCATE 2, 10
PRINT "Data acquisition in progress.  Press 'Q' to terminate."
'initialize all counters, this will begin count immediately
 dum% = CINT(depth * 10)
 CTLOAD
 GATEOFF              'freezes all counters
 GATEON               'starts all counters
 RESETB               'prepares port A for input
     '****** actual well logging is now on ******

 DO
   DO       'this is the polling loop, exits when r% <> 0
    t% = INP(PortA)
    r% = t% AND &H3
   LOOP WHILE r% = 0

  CTREAD             'read counters

  SELECT CASE r%
   CASE 1          'signal from timer
  'do not increment depth
   time& = tcount&
   pulse& = pcount&
   FREQCOMP
   TODISK
   DISPLAY

   CASE 2          'signal from depth
   kount = kount + 1
    'test for increment
    IF kount = inc THEN
     time& = time& + tcount&
     pulse& = pulse& + pcount&
     SELECT CASE (t% AND &H4)
       CASE 0          'increment depth & write data to disk, log down
         dum% = dum% + inc

       CASE 4          'decrement depth & write data to disk, log up
         dum% = dum% - inc

     END SELECT
     depth! = dum% / 10
     kount = 0
     FREQCOMP
     TODISK
     DISPLAY

    ELSE
     time& = time& + tcount&          'totalize counts
     pulse& = pulse& + pcount&
    END IF

   CASE ELSE
    BEEP

   END SELECT
   r = 0
  LOOP UNTIL UCASE$(INKEY$) = "Q"
 CLOSE #1
 END SUB


SUB MAINMENU
 SHARED logs() AS ltype
DO
 BILDMENU                           'call to build main menu
 choice$ = UCASE$(INPUT$(1))
 PRINT choice$                      'Select option
   SELECT CASE choice$
     CASE "1"
     'call to execute caliper log routine
     CALL PREPARE(logs(VAL(choice$)).label, logs(VAL(choice$)).code)
     LOGGER
     GATEOFF

     CASE "2"
     'call to execute Flowmeter log routine
     CALL PREPARE(logs(VAL(choice$)).label, logs(VAL(choice$)).code)
     LOGGER
     GATEOFF
```

```
     CASE "3"
     'call to execute Temperature log routine
     CALL PREPARE(logs(VAL(choice$)).label, logs(VAL(choice$)).code)
     LOGGER
     GATEOFF

     CASE "4"
     'call to execute CBL log routine
     'CALL PREPARE(logs(VAL(choice$)).label, logs(VAL(choice$)).code)
     CLS : LOCATE 10, 15
     PRINT "SORRY!!! This option is not yet operational."
     LOCATE 12, 15: PRINT "Hit any key to return to Main menu."
     DO
     LOOP UNTIL INKEY$ <> ""

     CASE "Q"             'Only option to quit
     CLS
     SYSTEM

     CASE ELSE
     BEEP
   END SELECT
LOOP
END SUB


'Function to create output file from well name
FUNCTION MKFILE$ (wn$) STATIC
LOCATE 15, 10
PRINT "Please wait ... creating output file!"
wkount = 1
wnlen = LEN(wn$)
wfile$ = "testfile.xxx"
testf$ = UCASE$(wn$) + LTRIM$(STR$(wkount)) + "." + "DAT"
SHELL "dir" + " > " + wfile$
 OPEN wfile$ FOR INPUT AS #3
 DO UNTIL EOF(3)
   LINE INPUT #3, line$
   IF line$ <> "" THEN
     f1$ = RTRIM$(LEFT$(line$, 8))
     f2$ = RTRIM$(MID$(line$, 10, 3))
     wtest$ = f1$ + "." + f2$
     IF testf$ = wtest$ THEN
       wkount = wkount + 1
       testf$ = UCASE$(wn$) + LTRIM$(STR$(wkount)) + "." + "DAT"
     END IF
   END IF
 LOOP
CLOSE #3
KILL wfile$
MKFILE$ = testf$
LOCATE 15, 10
PRINT SPACE$(37)
END FUNCTION
```

```
' Subroutine to create output data file
' filename will be the well name + a number as identifier
' then set the stage for actual well logging.
SUB PREPARE (log$, code$) STATIC
CLS
LOCATE 5, 5
PRINT "Ready to run " + RTRIM$(log$) + " (" + DATE$ + ")"
LOCATE 7, 5: INPUT "                    Please enter well name ==> ", f$
LOCATE 9, 5: INPUT "Enter data gathering increment (1=10 cm) ==> ", inc
LOCATE 11, 25: INPUT "Enter starting depth ==> ", depth!
  depth! = ABS(depth!)
  IF inc = 0 THEN
    inc = 1                    'Default increment value of 1
  END IF
  IF f$ <> "" THEN
   outfile$ = MKFILE$(f$)        'Call function to create filename
  ELSE
   outfile$ = RTRIM$(code$) + MID$(DATE$, 1, 2) + MID$(DATE$, 4, 2) + ".DAT"
  END IF
  BEEP
LOCATE 15, 15: PRINT "Output data filename will be " + outfile$
LOCATE 22, 5: PRINT "Press 'Y' to run " + log$
LOCATE 23, 10: PRINT "Any other key to return to Main Menu ..."
LOCATE 23, 51: decis$ = UCASE$(INPUT$(1))

  IF decis$ = "Y" THEN
    ' open outfile and write first line header (well name, logtype, date)
    OPEN outfile$ FOR OUTPUT AS #1 LEN = 30000
    WRITE #1, SPACE$(20), UCASE$(f$), log$, DATE$
  ELSE
    MAINMENU
  END IF
END SUB


' Subroutine to reset Port B to prepare for signal from depth & timer
SUB RESETB
  OUT PortB, &HFF       'high to clear flip-flops (positive logic)
  OUT PortB, &H0        'low to prepare for signal from depth or timer
END SUB


'Subroutine to write data to disk
SUB TODISK
  WRITE #1, depth!, CINT(value! * 10) / 10
END SUB
```

APPENDIX B: Program listing of the interrupt code written in Assembly language callable by high-level programs to drive the data acquisition system

```
;*This is a skeleton for a assembly program to act as
;* core for borehole logging program.
;* Started: 12/09/92          Latest revision: 02/10/92
;* Author: Jósef Hólmjárn

;* First some MASM stuff to locate segments etc.

     .model      medium, basic
     extrn  SETUEVENT:       far
;****????     extrn flag:word
     .code
;* Start by defining the Port Numbers for the hardware

CT     EQU    0314H              ; base address for CounterTimer
CTC     EQU    CT+3              ; Control register
CT1    EQU    CT           ; Counter 1
CT2    EQU    CT+1              ; Counter 2
CT3    EQU    CT+2              ; Counter 3

PIO    EQU    0310h         ;8255
porta  EQU    PIO
PORTB  EQU     PIO+1
PORTC  EQU     PIO+2
PIOC   EQU    PIO+3

;* We are going to use Hardware Interrupt 2 (IRQ2) which is mapped
;* as # 00Ah. First we have to get and save the interrupt vector (address)
;* of the old routine supposed to serve this interrupt (to be polite and SAVE)
;* The safest way to change interrupt vectors is by DOS int 21 # 25h and 35h

init   proc                          ;
     jmp     SHORT start

;* Put the data into the code segment for technical reasons
public  time
public  count
public timeo_flag
OLDVECTOR_0A    dd     0
old_mask       db     0
up_d_flag      dw     0
timeo_flag     dw     0
time          dd    0
count         dw    0
timeH          dw    0
timeL         dw    0
flag        db    0
depth         dw     0
event         dw     0

start:  push    ds
     mov   ax,350Ah                     ;Get old vector
```

```
     int  21h                          ;and
     mov    word ptr cs:OldVector_0A,bx     ;save it
     mov    word ptr cs:OldVector_0A +2,es
     push  cs                    ;Put our code segment into ds
     pop   ds                    ;and our handler address in dx
     lea    dx,IRQ2Handler          ;=new vector in ds:dx
     mov   ax,250Ah              ;put it in place
     int  21h                   ;for IRQ2

;************
;* Now we must set up the hardware we are going to use.
;**
;* Set up PIO
     mov     dx,PIOC
     mov     al,10010000B
     out     dx,al
     mov     dx,portB
     mov     al,00fh
     out     dx,al
     inc     dx            ; port C
     out     dx,al

;**
;* Set up counters
     mov     dx,ctc          ;select counter control register
     mov     al,00111100B   ; C0 mode 2
     out     dx,al
     mov     al,01011100B   ; C1 mode 2
     out     dx,al
     mov     al,10111100B   ; C2 mode 2
     out     dx,al

; *************    loadcnt
     dec     dx             ; select C3
     mov     al,0FFh         ; load C3 w/ FFFF
     out     dx,al
     out     dx,al

;***************   loadtime
     dec     dx
     mov     al,200          ;200 dec in C2
     out     dx,al
     dec     dx
     mov     al,LOW 30000      ;30 mS in C1
     out     dx,al
     mov     al,HIGH 30000
     out     dx,al
      mov     dx,portb
      mov     al,0FFh        ;Counter gates on, reset flip-flops
      out     dx,al
      mov     al,000h
      out    dx,al
      inc    dx          ;port C
      mov    al,000h
      out    dx,al
      mov    al,0ffh
      out    dx,al

;**
;* Set up interrupt controller 8559 to allow IRQ2
      cli                ;forbid interrupts
      in   al,21h
      and  al,11111011b        ;mask for IRQ2
      out  21h,al
      pop    ds
      sti                ;allow interrupts
      ret
init  endp

;*******************************************************
;* Interrupt handler checks if timeout (6 sec) or Cableint(10cm)
;* reads the counters and puts the results in variables BASIC can
;* read and then calls SETUEVENT to inform BASIC

public  IRQ2Handler
IRQ2Handler proc
      push   ds
      push   ax
      push   bx
      push   cx
      push   dx
      push   es

      sti            ;enable interrupts

;*****
;* Check if time or cable and then if up or down
      mov     dx,porta
      in      al,dx
      mov     flag,al
      test    al,01B
      jnz     timeout
      test    al,10B
      jz      exint
cable:
      call    clcaflg
      mov     al,flag
      test    al,100B
      jz      up
      add     depth,1
      jnz     mesure
```

```
up:    sub     depth,1
       jmp     SHORT mesure
timeout:
       call    cltflg
mesure:
       mov     dx,portc
       mov     al,00          ; GateOff
       out     dx,al          ;stops counters
       call    GetTime
       call    getcnt
;      call    setmode
       call    loadcnt
       call    loadtime
       mov     dx,portc
       mov     al,0FFh        ; GateOn
       out     dx,al          ;start counters
;***************
;call   SETUEVENT ;Let BASIC know of UEVENT
exint:
       mov     ax,0FFFFh            ; Set flag for BASIC
       mov     event,ax             ; to poll w/ GetEvent

;***************
;* enable hardware interrupts. Location of this code
;* may have to change
       cli
       mov     al,20h
       out     20h,al
       sti

       pop     es
       pop     dx
       pop     cx
       pop     bx
       pop     ax
       pop   ds
       iret                 ; Return from interrupt
;***********************************************************
; Following subs are local to this procedure
;***
cltflg:
       mov     dx,portb
       in      al,dx
       or      al,00000001B
       out     dx,al
       dec     al
       out     dx,al
       retn
clcaflg:
       mov     dx,portb
       in      al,dx
```

```
        or      al,00000010B
        out     dx,al
        dec     al
        dec     al
        out     dx,al
        retn
setmode:
        mov     dx,ctc          ;select counter control register
        mov     al,00111100B    ; C0 mode 2
        out     dx,al
        mov     al,01011100B    ; C1 mode 2
        out     dx,al
;       mov     al,10111100B    ; C2 mode 2
;       out     dx,al
        retn
loadcnt:                        ;***** Loads Counter
        mov     dx,ct3
        mov     al,0FFh         ; load C2 w/ FFFF
        out     dx,al
        out     dx,al
        retn
getcnt:                         ;***** Reads Counter
        mov     dx,ctc
        mov     al,10000000B
        out     dx,al
        mov     dx,ct3
        in      al,dx
        xchg    ah,al
        in      al,dx
        xchg    ah,al
        mov     count,ax
        retn
loadtime:                       ;*** Loads Timer
        mov     dx,ct2
        mov     al,200          ;200 dec in C2
        out     dx,al
        dec     dx
        mov     al,LOW 30000    ;30 mS in C1
        out     dx,al
        mov     al,HIGH 30000
        out     dx,al
        retn
gettime:                        ;***** Reads Timer
        mov     dx,ctc
        mov     al,0
        out     dx,al
        mov     dx,ct1
        xor     ax,ax
        in      al,dx
        xchg    ah,al
        in      al,dx
```

```
        xchg    ah,al
        mov     timel,ax
        mov     dx,ctc
        mov     al,01000000B
        out     dx,al
        mov     dx,ct2
        xor     ax,ax
        in      al,dx
        mov     timeh,ax
        retn

IRQ2Handler endp
; **********************************************************
;* Here Comes the Public Subrutines Callable from BASIC
;* or whatever
;* Basic expects integers to be in AX at return and long
;* integers to be in DX (high) and AX (low)
;***________________________________________________*****
Public ReadCount
ReadCount       proc            ;returns count value
        mov     ax,count
        ret

ReadCount       endp
;*******************


Public  ReadTimel
ReadTimel  proc         ; returns time value low word
        mov     ax,timel
        ret

ReadTimel       endp
;********************
public  ReadTimeH
ReadTimeH       proc            ; returns high byte of time
        mov     ax,timeh
        ret
ReadTimeH       endp
;********************
Public  ReadDepth
ReadDepth       proc
        mov     ax,depth
        ret
ReadDepth       endp
;**********************


public  ResEvent
ResEvent        proc
        mov     ax,event
        mov     event,00
        ret
ResEvent        endp
```

```
;*********************
Public  SetDept
SetDept Proc
        push    bp
        mov     bp,sp
        mov     bx,[bp+6]
        mov     ax,[bx]
        mov     depth,ax
        pop     bp
        ret     2

SetDept endp

;******************
Public  GetEvent
GetEvent        proc
        mov     ax,event
        ret
GetEvent        endp

;********************
Public  ResInt
ResInt          proc            ; resets old interrupt vectors and
        cli


        mov     al,cs:old_mask
        out     21h,al
        push    ds              ; cleans up
        lds     dx,cs:OldVector_0A
        mov     ax,250Ah
        int     21h
        pop     ds
        sti
        ret
ResInt          endp

END
```

APPENDIX C: Program code written in MS QuickBASIC which gathers data using the interrupt handler listed in Appendix B.

```
DECLARE SUB CreateFile ()
' Microsoft QuickBASIC program that demonstrates the use of the CORE driver
' To use : load BASIC with "QB/lalex", this loads ALEX.QLB quick library
' To compile, alex.lib must be in the same directory of the directory
'  specified in LIB.
'EnvVar
DECLARE SUB mesure ()
DECLARE SUB init
DECLARE SUB ResInt
DECLARE SUB ResEvent
DECLARE FUNCTION ReadCount% ()
DECLARE FUNCTION ReadTimeH% ()
DECLARE FUNCTION ReadTimeL% ()
DECLARE FUNCTION ReadDepth% ()
DECLARE FUNCTION GetEvent% ()
DECLARE FUNCTION SetDept% (Depth%)
DECLARE SUB handler

REM $INCLUDE: 'qb.bi'
ON ERROR GOTO errorh
DIM regs AS RegType
CALL init
CreateFile
CLS
INPUT "Please enter starting depth ==> "; Depthm!
 Depth% = CINT(Depthm! * 10)
 Dummy% = SetDept(Depth%)

'ON UEVENT GOSUB UserEvent
'UEVENT ON

DO
  DO
  LOOP UNTIL GetEvent% <> 0
   mesure
LOOP UNTIL UCASE$(INKEY$) = "Q"
'  CALL Interrupt(&HA, regs, regs)
  PRINT "Stopped by user"
  CALL ResInt
END


errorh:
      PRINT "timl%"; timl%; "Timh%"
      PRINT Timh%; "TimeHigh&"; TimeHigh&
      PRINT "tim&"; Tim&; " F#"; F#
      PRINT "Count%"; count%; "Tidni!"; Tidni!
RESUME NEXT
```

```
SUB CreateFile
 t$ = TIME$
 tfile$ = "TEST" + RTRIM$(LEFT$(t$, 2)) + RTRIM$(MID$(t$, 4, 2)) + ".DAT"
 OPEN tfile$ FOR OUTPUT AS #1 LEN = 30000
END SUB


SUB mesure
      count% = ReadCount%
       IF count% < 0 THEN
             lcount& = count% + 65536
       ELSE
             lcount& = count%
       END IF
      count% = CINT(65536 - lcount&)

      timl% = ReadTimeL%
      Timh% = ReadTimeH%
      timlt% = 30000 - timl%
      TimeHigh& = ((201& - Timh%) * 30000&)' + 30000
      Tim& = timlt% + TimeHigh&
      Depthm! = ReadDepth% / 10
      F! = (CLNG(count%) * 1000) / (Tim& / 1000)
      Tidni! = F!
      Value! = .02777778# * F! - 17.777778#
      WRITE #1, Depthm!, Tim&, count%, CINT(Value! * 10) / 10
      PRINT
      PRINT USING "##########.#"; Depthm!; CINT(Value! * 10) / 10;
      PRINT Tim&; count%; F!;
      CALL ResEvent

END SUB
```