To the Graduate Council:

I am submitting herewith a thesis written by William Bruce Jatko entitled "A Microcomputer-Based Averaging Flowmeter Using FORTH Programming Language." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Donald W. Bouldin

Donald W. Bouldin, Major Professor

We have read this thesis
and recommend its acceptance:

Robert W. Rochelle

J. M. Youngquist

Accepted for the Council:

Vice Provost
and Dean of the Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature William Bruce Jatho

Date May 9 1985

A MICROCOMPUTER-BASED AVERAGING FLOWMETER USING FORTH PROGRAMMING LANGUAGE

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

William Bruce Jatko

June 1985

ACKNOWLEDGEMENTS

The author wishes to thank the members of his Committee, Drs. J. M. Googe and R. W. Rochelle, for their help. Special thanks goes to Major Advisor Dr. D. W. Bouldin for his patience and invaluable advice, which has been greatly appreciated.

Thanks are also due to members of the Oak Ridge National Laboratory staff for their consideration. Particularly helpful during this effort were L. H. Thacker and C. A. Mossman. Appreciation is also given to Dr. T. V. Blalock for his help and guidance.

Finally, thanks must be given to the author's wife, Beverly, and to children Stacey and Brandon, for their patience and encouragement. Without their cooperation this undertaking would not have been possible.

ABSTRACT

A flowmeter was designed and constructed to measure the average fluid flow over a ten-minute interval. The instrument used a bubbler system and differential pressure sensors to measure the density-compensated level in a sealed vessel. A microcomputer was used to record a fluid-level histogram and to calculate the slope of the histogram through a linear least-squares regression. The average flow is proportional to the slope of the level histogram. The instrument met or exceeded all requirements.

TABLE OF CONTENTS

CHAPTER		PAGE
I.	INTRODUCTION	1
	Background Information	1
	System Specifications	2
	Alternate Methods	3
	Summary	3
II.	DEVELOPMENT OF MODEL	5
	Bubbler System	5
	Vessel	7
	Differential Pressure Cells	8
	Current-to-Voltage Conversion	9
	Analog-to-Digital Conversion	11
	Computer Flow Equation	11
	Error Estimates	14
III.	HARDWARE DEVELOPMENT	18
	System Requirements	18
	Power Supply Frame	19
	Single-Board Computer	19
	Analog I/O Board	23
	Alarm Interface	25
	Display Interface	25
	Reset-Analog Interface Board	27

IV. SOFTWARE DEVELOPMENT 31
Preliminary Considerations 33
Development of Least-Squares Fit 34
Program Development 42
System Initialization 43
Data Array 44
RESET Service Routine 45
TIMER Service Routine 46
V. EXPERIMENTAL RESULTS 51
Analog Input Filter 51
Verification of Model 51
Scale Factor 52
Least-Squares Algorithm 54
Statistical Sampling 55
Operational Tests 56
Transient Response 59
Long-Term Stability 59
VI. SUMMARY AND CONCLUSIONS 60
LIST OF REFERENCES 62
APPENDICES 64
APPENDIX A 65
APPENDIX B 73
VITA 90

LIST OF FIGURES

FIGURE		PAGE
1.	Diagram of containment vessel	6
2.	Flowmeter block diagram	20
3.	Single-board computer block diagram	22
4.	Analog-to-digital converter block diagram	24
5.	Alarm interface block diagram	26
6.	Display interface block diagram	27
7.	Reset interface diagram	28
8.	Analog input filter for typical channel	30
9.	Arbitrary Collection of Data Points Illustrating a Least-Squares Fit	36
10.	Arbitrary data Points after translating origin	39
11.	Data points for function y = 2x + 5	41
12.	Flowmeter Software Block Diagram	43
13.	System Initialization Flow Diagram	44
14.	Memory map of Channel One data array	46
15.	RESET interrupt service flow diagram	47
16.	TIMER interrupt service flow diagram	48
17.	Effect of analog filter on input signal	52
18.	Diagram of experimental apparatus	53
19.	Frequency distribution histogram for data-sampling process	56
20.	Deviations of calculated flow from measured flow	58

CHAPTER I

INTRODUCTION

Background Information

The instrumentation of a chemical process that uses toxic material poses a difficult problem for the instrument engineer. The engineer is charged with measuring various phases of the process reliably and accurately. When the chemical process involves a radioactive substance, the process must be adequately shielded and isolated from human operators. This usually requires complete containment of the process within several inches of lead protection. While this method provides protection for the operators, it places additional constraints on the instrumentation. Any measurement sensor located inside the containment vessel must be reliable since access into the vessel is limited. The sensors must also be immune to radioactive bombardment which might deteriorate performance. The author was asked to design an instrument which would measure average fluid flow within a sealed, radioactive vessel.

A chemical process was being developed at the Oak Ridge National Laboratory to properly dispose of nuclear power reactor waste products. The waste product was a radioactive fluid left over from a commercial nuclear power facility. In the decontamination process, the fluid solution is solidified into metal cylinders. After the cylinders have been filled with the solidified fluid, the end caps are welded in place and the cylinders can be stored safely.

In order for solidification to occur, the chemical reaction rate must be tightly controlled. This is done by controlling the mass flow of reactants into the solidification vessel. The solution is pumped from a large reservoir into the solidification vessels. Since the reservoir was to be equipped with a bubbler system to measure the fluid level, a method of measuring the flow rate was devised using these fluid-level sensors. The bubbler system has the desired immunity to radioactive attack and is both accurate and reliable. The bubbler system could also be used to measure the density of the fluid. This is important since the fluid density, although uniform throughout the reservoir, would be variable with a specific gravity between 1 and 2. The density must be taken into account in controlling the mass flow.

The method chosen uses a single-board computer and a data-acquisition board to measure and chart trends in the reservoir level. By calculating the level changes over a period of time, an estimate of average mass flow over the time period can be made.

System Specifications

The flow measurement should have an accuracy of ±5%. The nominal flow is 9.0 milliliters per minute (mL/min) ±5 mL/min from a 7500-mL reservoir. It is desired to measure 9.0 mL/min ±0.4 mL/min. There are four independent channels, with each channel having a separate reset, digital flow indicator, and over/under flow alarms. The flow rate display should have a 0 to 9.9-mL/min range with a 0.1-mL/min resolution.

Alternate Methods

Although many methods of measuring the instantaneous flow rate would be more precise, they were deemed unsatisfactory since they would require electrical sensors be placed in the restricted access area. Because maintenance must be minimized, the bubbler system was chosen as the best alternative. This method permits all electrical components to be mounted outside the restricted area for maintenance access.

Summary

The method chosen uses a bubbler system with differential pressure transducers to measure the fluid level and density within a sealed containment reservoir. A single-board computer using a resident FORTH programming language was selected. The computer is a P-FORTH model made by Peopleware Systems, Inc. It is an STD bus design using a 6801 microprocessor and a 6522 peripheral interface adapter. A 16-channel analog-to-digital converter, Model RTI-1260 from Analog Devices, was selected for the data-acquisition board. The single P-FORTH computer monitors four channels of flow by time multiplexing. Level and density data are taken at 6-second intervals and stored in a data array. The computer performs a linear least-squares regression on the level/density points to estimate an average flow rate.

The chemical process was slow enough to allow the flow rate to be averaged over a 10-min period. The long measurement interval is

required to allow statistical accumulation of data. By taking large amounts of data, precision can be improved. The software uses 100 data points per measurement, which increases precision by an order of magnitude over a single-point measurement.

CHAPTER II

DEVELOPMENT OF MODEL

The radioactive fluid being instrumented was waste material from a commercial nuclear power reactor being processed at the Oak Ridge National Laboratory. The solution was to be solidified within metal cylinders which could then be sealed and stored. The solidification process requires that the solution be pumped at a controlled rate from a containment reservoir vessel into the solidification vessel where the chemical reaction occurs. Figure 1 is a schematic diagram of the containment vessel.

Bubbler System

Two metal tubes deliver a small flow of air from the main air supply into the vessel. One tube extends to the bottom of the vessel and the other tube to within distance D of the bottom. Since the pressure in the bottom of the tube is just enough to force an air bubble from the tube, the pressure in the tube is nearly the same as the hydrostatic pressure of the fluid at the tubing outlet. Since the air velocity is small, pressure drops along the tubing are neglected. A differential pressure-to-current transducer generates a 4- to 20-mA current proportional to the pressure difference across the pressure sensor. One pressure cell is connected from the level bubbler tube to the top of the vessel. This cell measures the

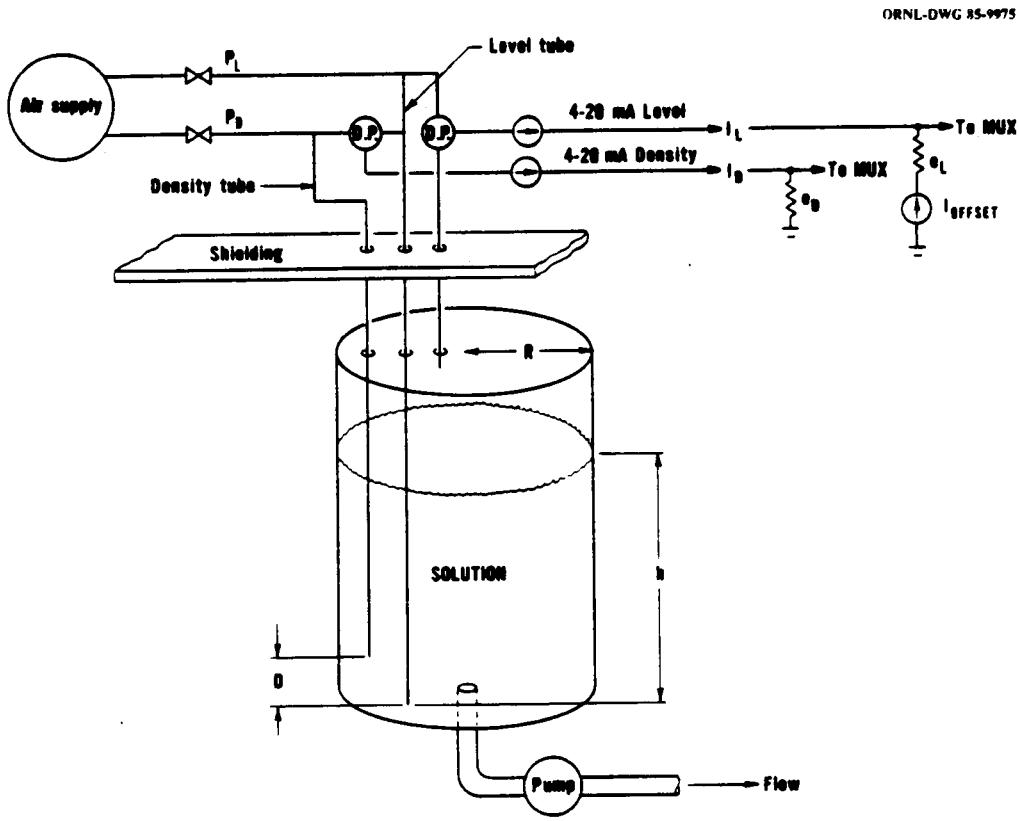


Figure 1. Diagram of containment vessel.

pressure of the fluid in the vessel and is proportional to the fluid height, h. The pressure equation for a static fluid is well known:

$$P = \rho g h \qquad (2\text{-}1)$$

where P is the pressure, ρ is the fluid density, and g is the gravitational constant. This leads to the equation for the level tube:

$$P_L = \rho g h \qquad (2\text{-}2)$$

The second pressure cell is connected between the level tube and the density tube and measures the pressure drop across distance D. Equation (2-1) still applies:

$$P = \rho g h$$

but h is known to be equal to D, and we can write

$$P_D = \rho g D \tag{2-3}$$

Since ρ and g are the same for both (2-2) and (2-3), we can divide and cancel the ρg terms. This yields

$$\frac{P_L}{P_D} = \frac{h}{D}$$

which can be written as

$$h = \frac{P_L \, D}{P_D} \tag{2-4}$$

Vessel

The fluid flow is the change in volume per time and can be written mathematically as

$$F = \frac{\partial}{\partial t} (\text{volume})_{\text{fluid}} \tag{2-5}$$

The volume of fluid in a cylinder is

$$V = \pi R^2 h \tag{2-6}$$

taking the derivative of (2-6) and assuming only time variance

$$\frac{\partial}{\partial t} V = \pi R^2 \frac{d}{dt} (h) \qquad (2\text{-}7)$$

Substituting (2-7) and (2-4) into (2-5)

$$F = \pi R^2 D \frac{d}{dt}\left(\frac{P_L}{P_D}\right). \qquad (2\text{-}8)$$

Differential Pressure Cells

The differential pressure cells convert pressure to current. The cell is designed to give a linear relationship between pressure and current in the form

$$I = mP + b \qquad (2\text{-}9)$$

where I is the output current, m is a scale factor, and b is an offset. Both m and b can be chosen to suit the application.

In this application, the pressure cell connected to the level probe is calibrated to provide 4 mA for zero pressure and 20 mA for the maximum head pressure, H_L. Since the solution has a maximum density twice that of water, H_L must be twice the vessel volume. Thus the 7.5-L vessel might have a potential fluid pressure equivalent to 15.0 L of water.

Using the proportionality

$$0 - H_L : 4 - 20$$

we can derive an equation for the pressure-to-current transfer function in the level probe

$$I_L = \left(\frac{16}{H_L}\right) P_L + 4 \qquad (2\text{-}10)$$

where I_L is in milliamperes.

The density probe transfer function is different from the level probe. The density probe would have a minimum pressure equal to the height of a column of water D inches high. With a fluid of specific gravity 2, the maximum pressure would be equivalent to a height of 2D. Thus, the density pressure cell has an operating range of D to 2D. This leads to the proportionality

$$D - 2D : 4 - 20$$

which can be written as

$$I_D = \frac{16}{D} P_D - 12 \qquad (2\text{-}11)$$

Current-to-Voltage Conversion

The analog-to-digital converter (ADC) has a 5-V full-scale range. The range of the ADC integrated circuit is 10 V full scale, but a variable-gain preamplifier is available to allow the use of the 5-V full-scale input.

Normally the current-to-voltage conversion is done by using a 250-ohm resistor in series with the current. This develops a signal

voltage from 1 to 5 V. However, this process does not utilize the full 12-bit dynamic range of the ADC, and consequently the accuracy of the measurement suffers. In many cases this reduced accuracy is acceptable due to the ease of implementation. In this case, it was decided to use the full 12 bits of accuracy available in the ADC board for the level measurement since it is the most critical, and to use the simpler, less accurate method for the density measurement. Therefore an analog circuit was designed to provide a 4-mA offset current and also provide low-pass filtering. This circuit allowed the 4- to 20-mA current to develop a 0- to 5-V signal at the ADC. The appropriate transfer equation is

$$e_L = \frac{5}{16} I_L - \frac{5}{4} \tag{2-12}$$

where e_L is in volts, I_L in milliamperes.

The density probe measurement was less critical, and the conventional 250-ohm resistor method previously described was used. This yields an equation for the density voltage.

$$e_D = I_D R_D$$

and since R_D = 250 ohms,

$$e_D = \frac{I_D}{4} \tag{2-13}$$

where e_D is in volts, I_D in milliamperes.

Analog-to-Digital Conversion

The final transfer equation is the analog-to-digital conversion. The ADC has 12 bits of accuracy. This allows 2^N or 4096 possible states. Since there is a 5-V full-scale range at the input, this gives

$$N^* = \frac{4096}{5} \quad e \tag{2-14}$$

where N^* is the digital number outcome of the conversion process. The asterisk is to remind us that N is a quantized integer and not a continuous number.

Computer Flow Equation

Rearranging (2-14) and substituting into (2-12) and (2-13) gives

$$\frac{5}{4096} \quad N_L^* = \frac{5}{16} \quad I_L - \frac{5}{4} \tag{2-15}$$

$$\frac{5}{4096} \quad N_D^* = (.25)I_D \tag{2-16}$$

substituting (2-10) and (2-11) into (2-15) and (2-16), respectively, gives

$$\frac{5}{4096} \quad N_L^* = \left(\frac{5}{16}\right)\left(\frac{16}{H_L} \quad P_L + 4\right) - \frac{5}{4} \tag{2-17}$$

$$\frac{5}{4096} \quad N_D^* = \left(\frac{1}{4}\right)\left(\frac{16}{D} \quad P_D - 12\right)$$

simplifying and rearranging

$$P_L = \frac{H_L \, N_L^*}{4096} \tag{2-18}$$

$$P_D = \left(\frac{D}{4}\right)\left(\frac{5 \, N_D^*}{4096} + 3\right) \tag{2-19}$$

Recalling the flow equation (2-8)

$$F = \pi R^2 D \; \frac{d}{dt} \; \frac{P_L}{P_D} \tag{2-20}$$

since the density pressure term is invariant within the measurement period, it can be moved outside the derivative operation

$$F = \frac{\pi \, R^2 \, D}{P_D} \; \frac{d}{dt} \; (P_L) \tag{2-21}$$

substituting (2-19) and (2-20) into (2-21)

$$F = \frac{\pi \, R^2 D}{\left(\frac{D}{4}\right)\left(\frac{5 \, N_D^*}{4096} + 3\right)} \; \frac{d}{dt} \; \frac{H_L \, N_L^*}{4096} \tag{2-22}$$

H_L and 4096 are constants and likewise can be removed from the derivative term:

$$F = \frac{\pi \, R^2 \, D \, H_L}{\left(\frac{D}{4}\right)\left(5 \, N_D^* + 3 \, (4096)\right)} \; \frac{d}{dt} \; (N_L^*) \tag{2-23}$$

simplifying

$$F = \frac{4 \pi R^2 H_L}{5 N_D^* + 12288} \frac{d}{dt} (N_L^*) \qquad (2\text{-}24)$$

Since the intended flow measurement is to be an average flow, we change the derivatives to delta and approximate

$$F \cong \frac{4 \pi R^2 H_L}{5 N_D^* + 12288} \frac{\Delta}{\Delta t} (N_L^*) \qquad (2\text{-}25)$$

This is the fundamental equation implemented in the FORTH software algorithm.

One modification of Eq. (2-25) is to place the denominator of the first term within the delta operator and to multiply and divide by 5. This was done to simplify the data acquisition software. This results in

$$F \cong \frac{4 \pi R^2 H_L}{5} \frac{\Delta}{\Delta t} \frac{5 N_L^*}{5 N_D^* + 12288} \qquad (2\text{-}26)$$

Since the first term is made of constants, they may be lumped together as a single constant K. We also replace the

$$\frac{5 N_L^*}{5 N_D^* + 12288} \qquad (2\text{-}27)$$

term by variable Q and write

$$F \simeq K \frac{\Delta Q}{\Delta t} \qquad (2\text{-}28)$$

Constant K is stored in the computer memory, and $\frac{\Delta Q}{\Delta t}$ is computed through the linear least-squares regression. The development of the least-squares algorithm is given in Chapter IV.

Error Estimates

The accuracy of the differential pressure cell is quoted by the manufacturer to be ±0.25% of full scale.[1] If the nominal vessel volume is considered to be 7500 mL, the bubbler system will measure the volume as 7500 mL ±18.75 mL. If we assume an anticipated flow rate of 9.0 mL/min, after 10 min there would be 7410 mL in the vessel. The bubbler system would measure this as 7410 mL ±18.75 mL. Considering the worst-case condition of the initial measurement at the upper limit of the error band, and the final measure at the lower limit of the error band, we would obtain measurements of 7518.75 mL and 7391.25 mL. The change of volume in the vessel would appear to be

$$\Delta V = 7518.75 - 7391.25$$

$$\Delta V = 127.5 \text{ mL}$$

when actually it was 90 mL. This would be a maximum error of

[1] Model GC-781 Moore Pneumatic to Electric Transducer, Data Sheet GC-781, Spring House, Pennsylvania: Moore Products Co. (1982).

$$\varepsilon = \frac{90 - 127.5}{90} \times 100$$

$$\varepsilon = 41.7\%$$

With an actual flow of 9.0 mL/min, this percent error is ±3.75 mL/min.

The 12-bit resolution of the ADC gives a quantizing uncertainty of ±0.5 least significant bit (LSB) where

$$LSB = \frac{7500}{2^{12}}$$

$$= 1.83 \text{ mL/min}$$

then 0.5 LSB = 0.62 mL/min.

Assuming the two errors to be uncorrelated, the total error from the two sources should add as the square root of the sum of squares.

$$\varepsilon = \sqrt{(3.75)^2 + (0.62)^2} \qquad (2\text{-}29)$$

$$= 3.8 \text{ mL/min}$$

$$= \pm 42\%$$

This shows that the bubbler system error dominates the total system error.

We consider the 3.8 mL/min error to be within 2 statistical standard deviations, which would include 95% of all occurrences. This means that one standard deviation would be

$$s = 1.9 \text{ mL/min}$$

The standard deviation of a sampled group is reduced by the square root of the number of samples[2]

$$s^* = \frac{s}{\sqrt{n}} \tag{2-30}$$

Where s* is the data group deviation and n is the number of data points in the sample group.

In order to reduce the 1.9 mL/min deviation to less than the desired 0.4 mL/min would require

$$n = \left(\frac{1.9}{0.4}\right)^2 \text{ samples}$$

$$= 22.6 \text{ samples per measurement.}$$

It was decided to use 100 samples in the measurement. This should give an estimation error of

$$\varepsilon = \frac{1.9}{\sqrt{100}}$$

$$= 0.19 \text{ mL/min}$$

which is within our desired error of ± 0.4 mL/min.

A physical system model has been developed and an estimation of the anticipated error has been made. The number of data points to use in the least-squares fit was chosen to be 100 in order to reduce

---

[2]Thornton C. Fry, Probability and Its Engineering Uses, 2nd edition, Princeton, New Jersey: D. Van Nostrand Co., p. 342 (1965).

the errors in measuring sensors. While additional errors from component tolerances, aging, and assumptions are also present, they are likely to be negligible as seen in Eq. (2-28).

CHAPTER III

HARDWARE DEVELOPMENT

System Requirements

In order to achieve the desired improvement in accuracy discussed in Chapter II, a method of taking and holding more than 100 data samples was required. A single-board computer implementation was chosen as the most practical way to acquire data from the pressure sensors and to calculate a slope from the sensor data history. Details of the slope calculations are considered in the chapter on software development.

An alternate hardware scheme using an analog operational amplifier with a hardware divider chip to perform the P_L/P_D division was suggested initially. This method would use a sample-hold circuit to store sample valves until the next sample was taken. Since the time between samples is relatively long (10 min), the sample-hold capacitor could contribute to sampling errors due to voltage discharge over the holding period. This architecture was rejected due to the high potential for sampling errors.

The flow to be measured is slow enough to permit sampling of the vessel level at 6-s intervals. This permits using one computer board to do the calculations and data acquisition for all channels by time-multiplexing.

Requirements for the system included four separate channels of flow measurement with a three-digit display, 120-V ac high/low flow alarms, and a reset for each channel.

A flow measurement system was assembled using a single-board computer, analog-to-digital conversion board, display interface board, alarm interface board, reset/analog interface board, and power supply frame. The system block diagram is shown in Figure 2.

Power Supply Frame

The power supply frame and card cage for the STD bus components is a ProLog Model 701B. This model was chosen because it has enough card slots to allow future expansion and comes in a 19-in. rack-mountable frame. This was desirable since the instrument would be installed in a process control room. The 19-in.-wide cabinet also provided enough front-panel space to mount digital display meters and reset buttons for all four channels. With this cabinet, the four channel flow monitoring system can be contained in one integral package. The ProLog cabinet also provided sufficient internal space to mount an auxiliary power supply for the pressure-to-current transmitters. The STD card cage holds 13 cards connected on the STD bus plane.

Single-Board Computer

The heart of the flow monitor is the P-FORTH card by Peopleware Systems, Inc. This card uses a Motorola MC 6801 microprocessor and a 6522 peripheral interface adapter (PIA). The P-FORTH card has 2K

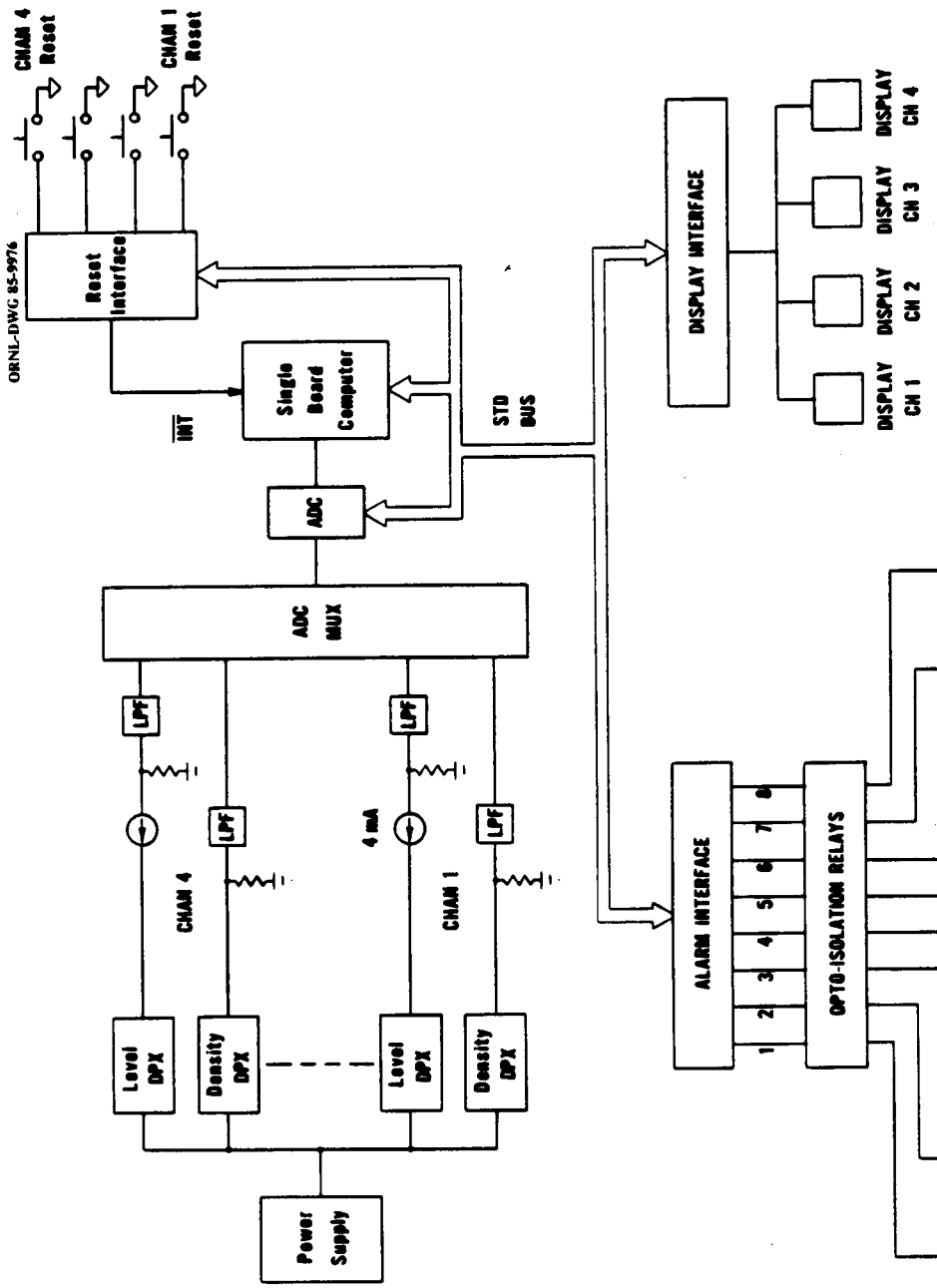


Figure 2. Flowmeter block diagram.

of random-access memory (RAM) and 6K of electrically erasable read-only memory (EEROM). The FORTH programming language is resident on the card in 8K of read-only memory (ROM). There are also 16 I/O lines available through the PIA and a serial RS-232 port for communications via a CRT terminal. Interrupts may be generated by the PIA in response to various external events. A block diagram of the single-board computer card is shown in Figure 3.[3]

The 6801 microprocessor uses a memory-mapped I/O scheme. However, the STD bus supports an I/O mapped region for use with Z-80 type microprocessors. The P-FORTH card handles this situation by decoding a block of memory from 0100-01FF hexadecimal as the I/O region. Memory accesses in this region are decoded by the P-FORTH circuitry as an I/O request. This allows 256 possible I/O locations in the system.

A convenient feature of the P-FORTH card is the use of the EEROM memory. When a FORTH application program is compiled, it is placed in the EEROM space. Once application software has been written and tested satisfactorily, the P-FORTH card can be switched from a development mode to a target application mode. The application software can be vectored to begin execution upon power-up. This allows the designer to use the same card as both a development station and a target machine.

---

[3]P-FORTH Users Manual, Minneapolis, Minnesota: Peopleware Systems, Inc. (1982).

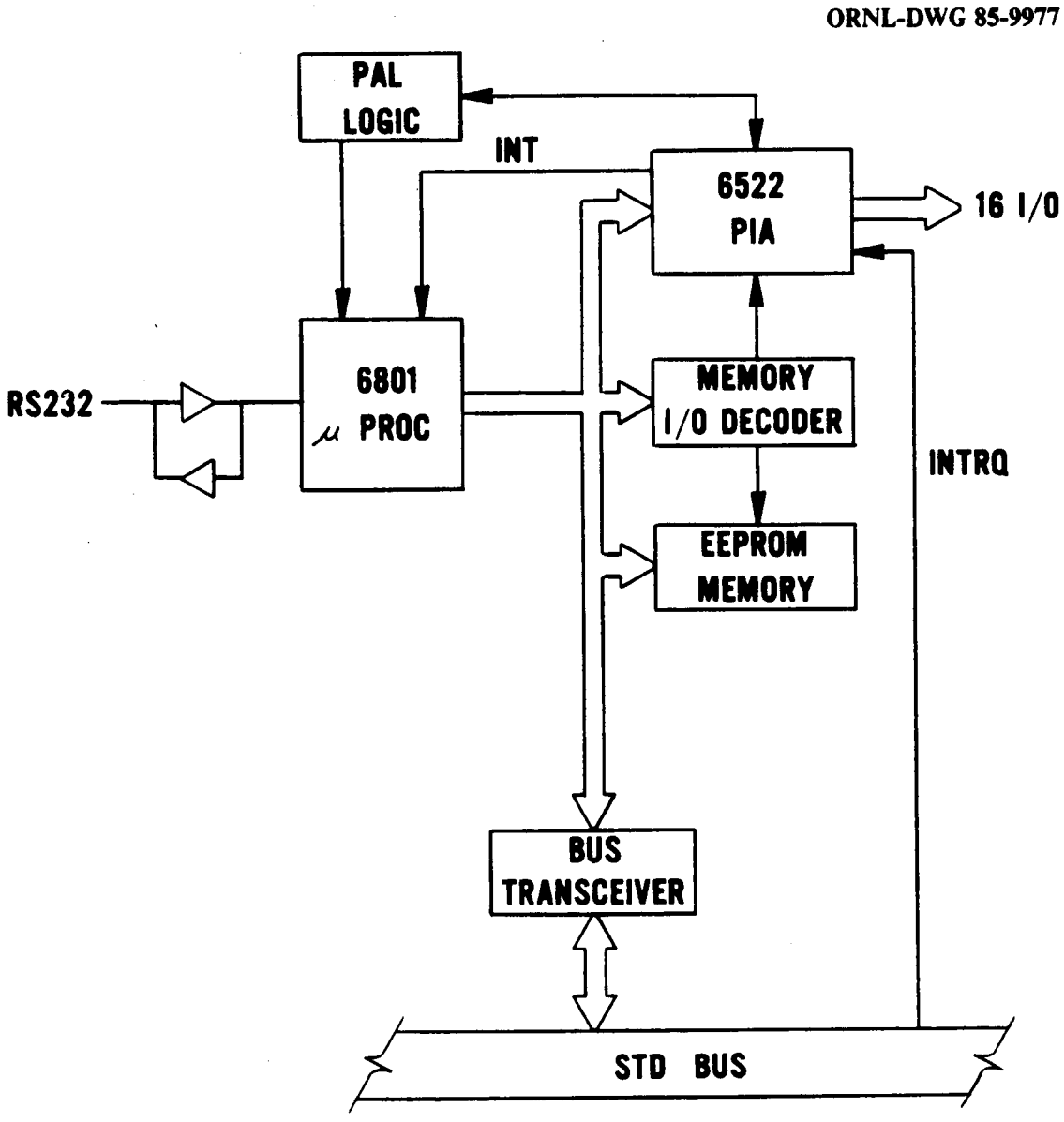


Figure 3. Single-board computer block diagram.

The flow monitor uses the two timers on the 6522 PIA to generate the sample command. The vessel volume is sampled every 6 s. Since 4 channels are time-multiplexed, a sample command is issued at 1.5-s intervals. A software algorithm maintains proper channel sequencing.

Another interrupt is generated on the reset/analog interface card. The reset interrupt indicates one or more channels are to be reset. The 6801 polls the 6522 interrupt register to determine if an interrupt request is from a reset command or a sampling command. The sampling command has the higher priority in the polling. More detailed consideration of the software is given in Chapter IV.

Analog I/O Board

The vessel volume and density is sampled and quantified on the analog I/O board, which is an Analog Device Model RTI-1260. The RTI-1260,[4] shown in Figure 4, provides 16 single-ended or 8 differential inputs which are converted to 12 bits. A programmable gain amplifier allows input signals from 10 mV to 10 V. The sample-hold circuit and AD574 analog-to-digital converter provides throughput rates of up to 25,000 channels per second. The analog-to-digital conversion time is 25 μs. The RTI-1260 has a dc-to-dc converter on board which allows it to be powered from the 5-V dc on the STD backplane.

[4] RTI-1260 Users Manual, No. AC1563. Norwood, Massachusetts: Analog Devices, Inc. (1981).

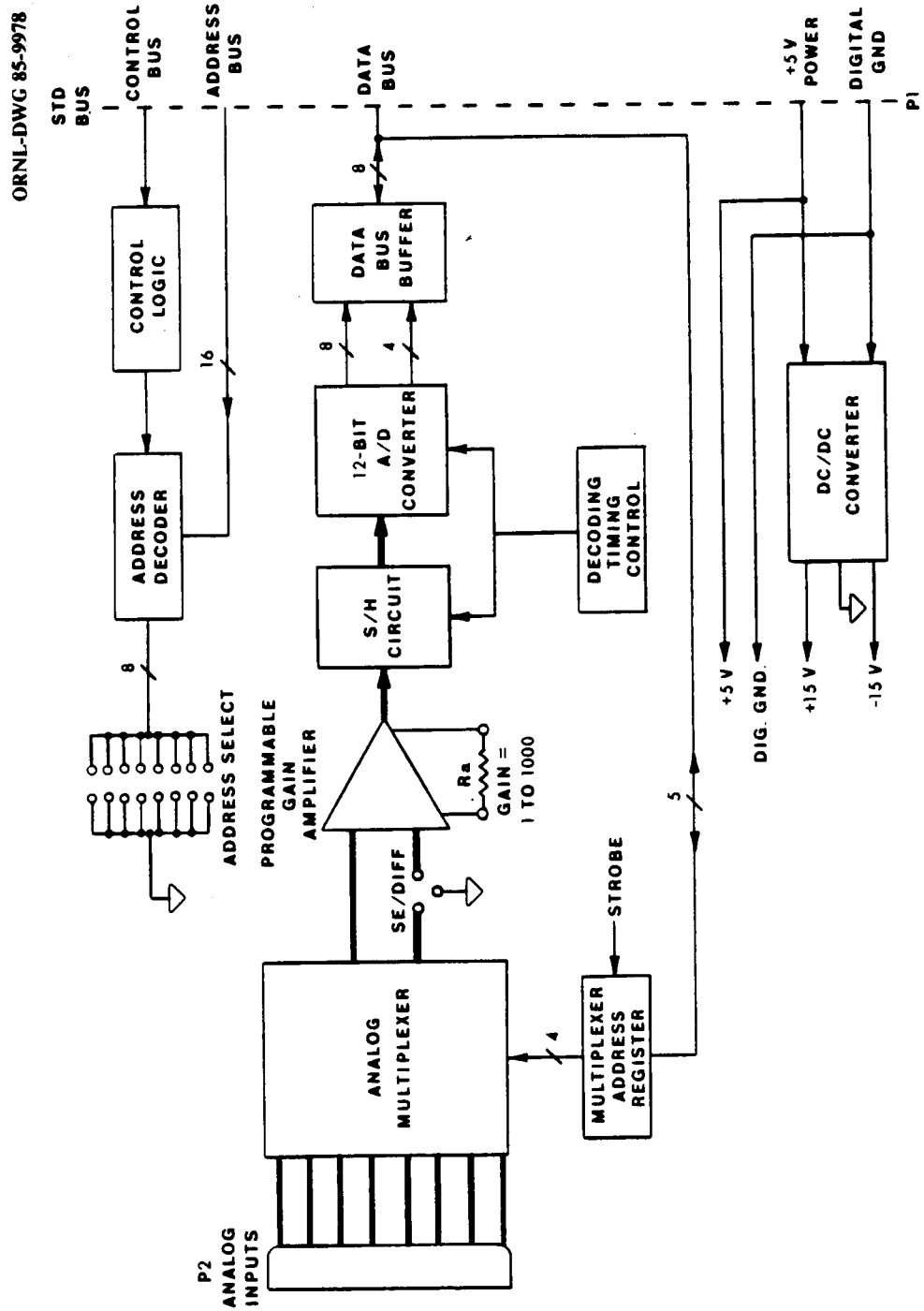


Figure 4. Analog-to-digital converter block diagram.

Alarm Interface

The flow monitor was required to provide contact closures to indicate flow levels above or below prescribed limits. The P-FORTH card compares the current flow rate with the upper/lower limits stored in memory.

A ProLog Model 7507 interface card is used to latch the status of each channel into the I/O region of memory. The 7507 card is interfaced to an OPTO-22 optical isolation relay card,[5] which holds eight optically isolated solid-state relays. Each channel has one relay to indicate overflow and underflow. The relay contacts activate supervisory warning lamps in the main control room. The eight alarms are mapped directly to the 8-bit data word written to the 7507. The selected alarm is activated by setting the corresponding bit in the data word high and writing to the 7507 input port. The 7507 has internal latches to hold the status until the alarm is cleared.[6] Figure 5 shows the 7507 diagram.

Display Interface

Each channel in the flow monitor has a 4-digit display to indicate average flow in mL/min. The displays use BCD inputs for each digit, so 16 input lines are required per channel. A ProLog

---

[5] OPTO-22 Data Book, 1st ed., Huntington Beach, California: OPTO-22, Inc. (1983).

[6] 7507 General Purpose I/O Interface Card Users Manual, Monterey, California: Pro-Log Corp. (1982).

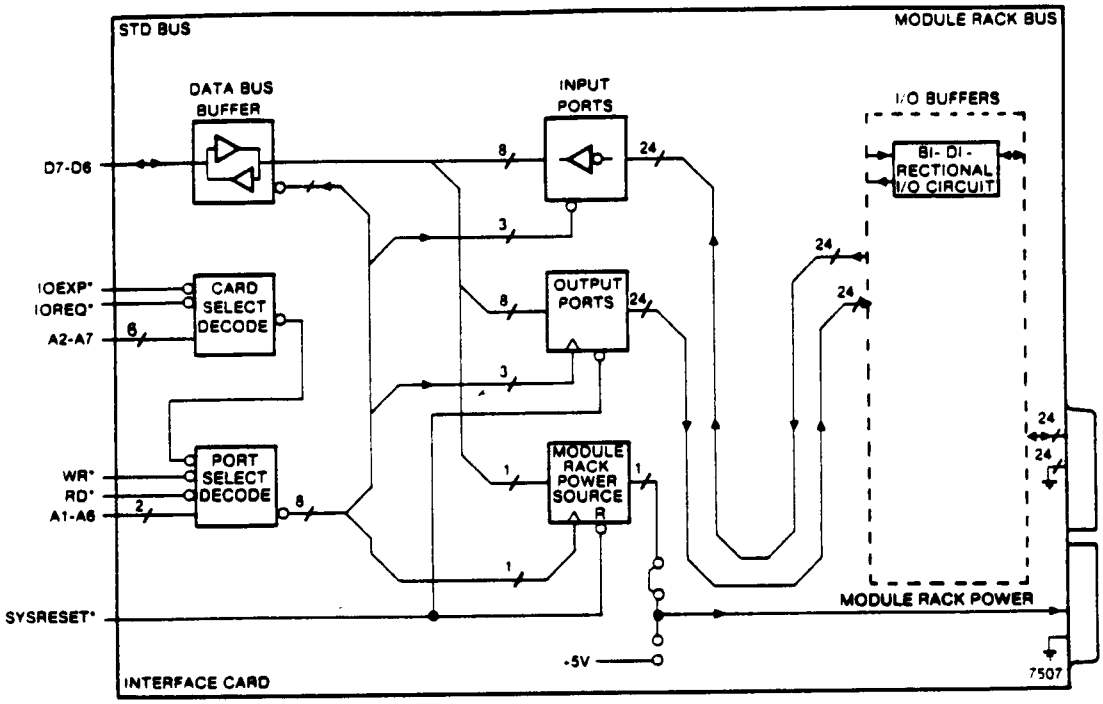


Figure 5. Alarm interface block diagram.

Model 7602 Output Port Card (shown in Figure 6) provides the interface between the P-FORTH card and the four display modules. The 7602 has 64 output lines and maps into the I/O region of the P-FORTH card.[7] The 7602 has eight output ports to allow interfacing to the outside world. Each display module is connected to two output ports on the 7602. This allows each display module to have an address in the I/O region.

Flow data from the P-FORTH card are converted to BCD format and then written to the proper channel display module. The 7602 latches the BCD values into the channel output ports, where the display value is held until the next update cycle.

---

[7] 7602 Output Port Card Users Manual, Monterey, California: Pro-Log Corp. (1981).

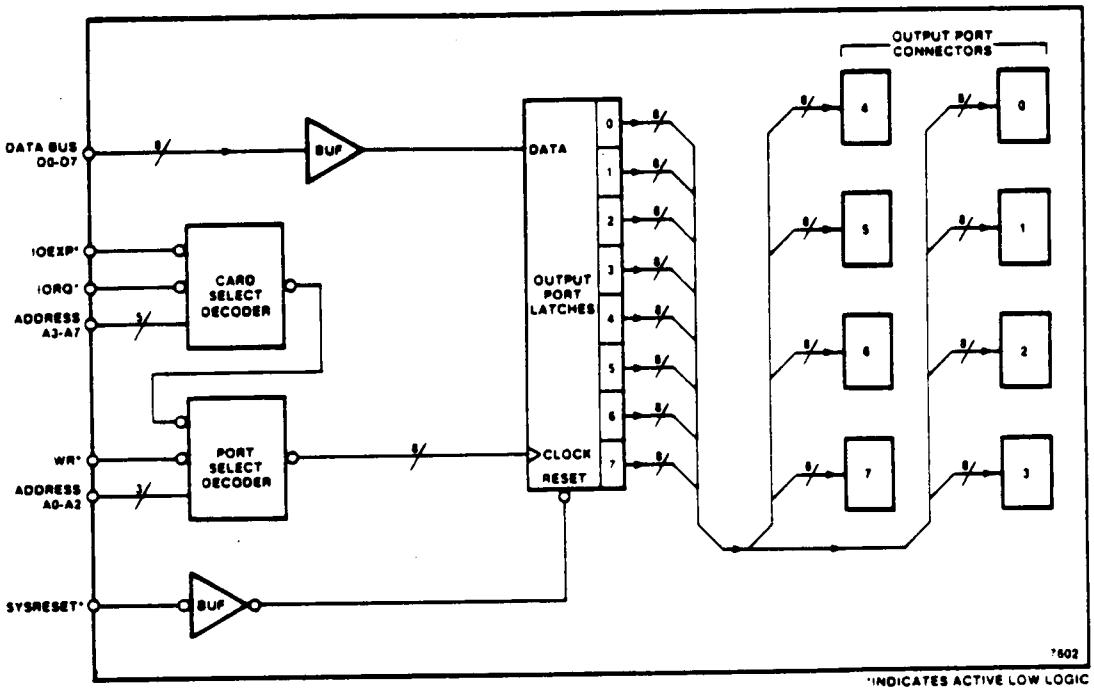


Figure 6. Display interface block diagram.

Reset-Analog Interface Board

The reset-analog interface card provides the auxiliary functions of analog filtering the signal currents from the pressure sensors and generating an interrupt request for a channel reset. The reset-analog card used is a ProLog Model 7904,[8] a blank wire-wrap card with an I/O decoder provided. This allows the user to design and fabricate any I/O function required and to use the on-board decoder and bus transceivers to access the STD bus.

The reset section of the card is shown in Figure 7. The front panel pushbuttons are used to trigger a 7474 latch after being

[8] 7904 Decoded Utility I/O Card Users Manual, Monterey, California: Pro-Log Corp. (1982).

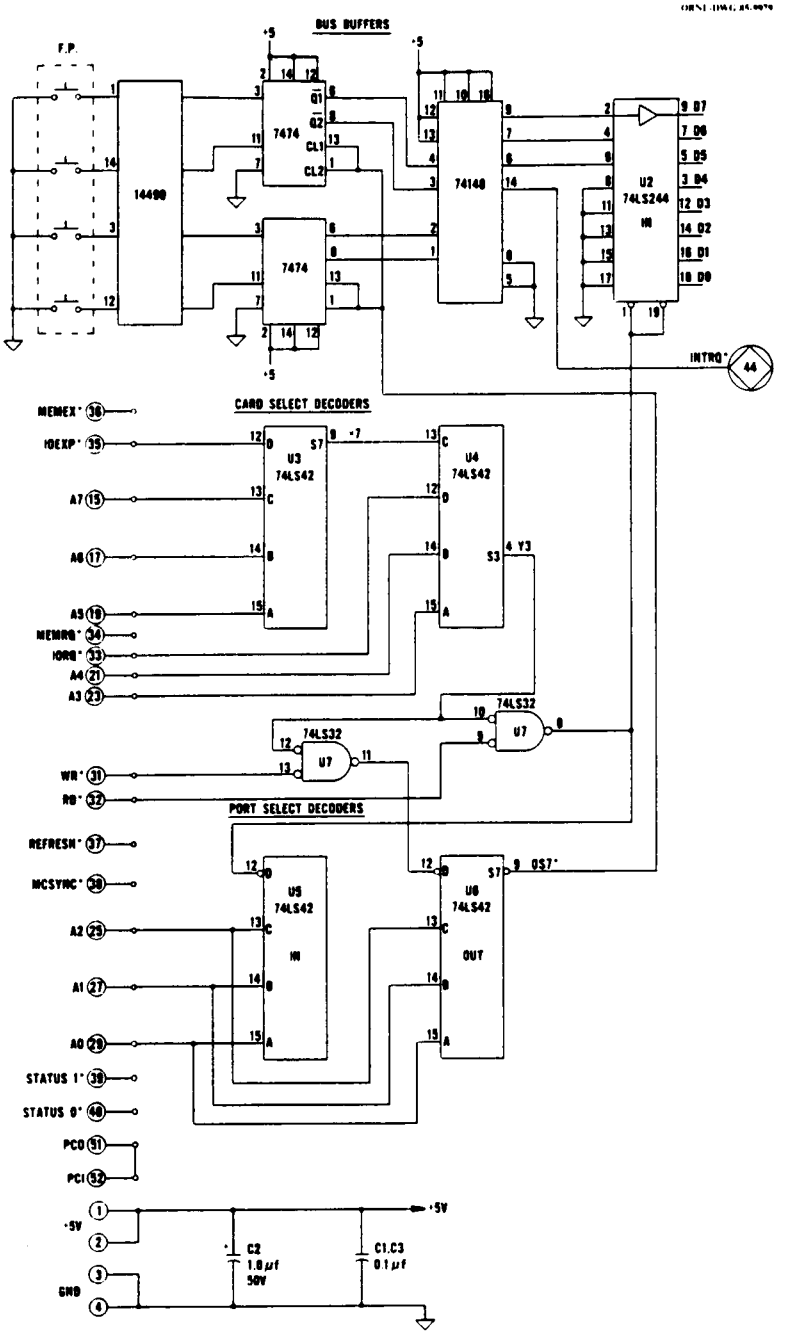


Figure 7. Reset interface diagram.

debounced by the MC14490. The latches are connected to an 8-to-3 encoder, which provides a binary output of the channel number to be reset and generates an interrupt request on the STD bus pin 44. The interrupt request line (INTRQ*) will set an interrupt flag in the 6522 PIA chip on the P-FORTH card. The 6801 microprocessor reads the reset I/O port to determine the proper channel to be cleared. The interrupt condition on the 7904 reset-analog card is cleared by writing a dummy word to the port. The analog section for a typical channel is shown in Figure 8.

The density signal is developed across a 250-ohm resistor. The capacitors in parallel form a 125-Hz low-pass filter. The low-pass filter acts to remove noise and other high-frequency artifacts from the input signal.

The level signal is filtered similarly and also has a 4-mA current source to provide an offset current. The offset current is used to translate the 4- to 20-mA signal to 0 to 16 mA. This is done to utilize the full dynamic range of the 12-bit analog-to-digital converter (ADC). The ADC input range is 0 to 5 V; the 4- to 20-mA current would develop a 1- to 5-V signal as seen in the density channel. This results in 1 V of the full range of the ADC not being utilized. This is undesirable in the level channel where the accuracy is most important. The density range of the fluid being measured is small enough to permit the loss of dynamic range in conversion.

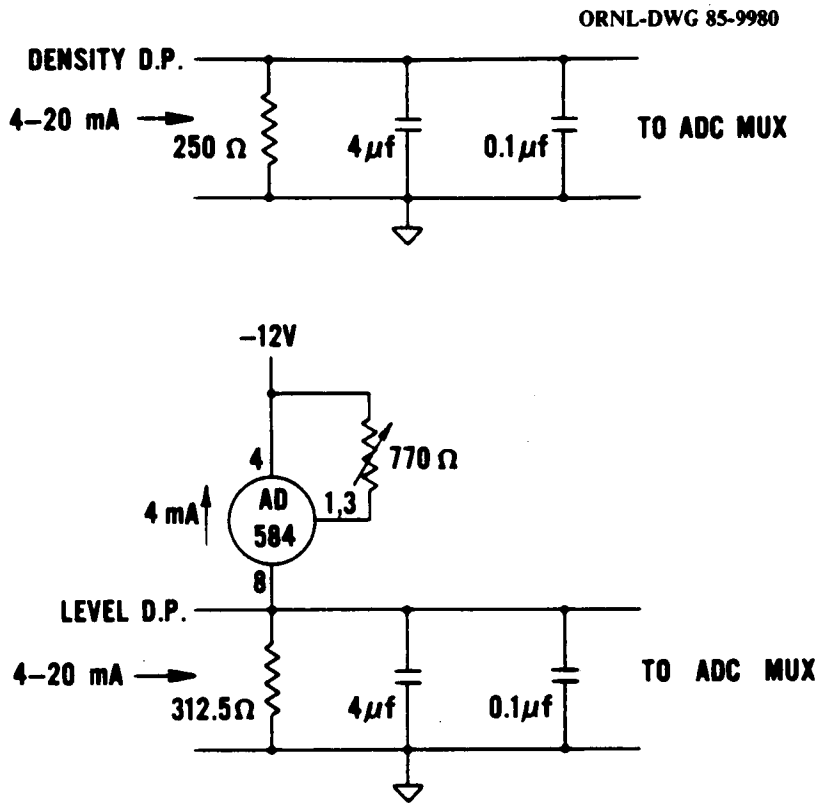


Figure 8. Analog input filter for typical channel.

CHAPTER IV

SOFTWARE DEVELOPMENT

The FORTH programming language was chosen for this application due to its advantages in real-time instrument systems. FORTH is a modularized, structured, extensible programming environment originally designed for real-time process control applications. The primary advantages of FORTH are its interactive nature, flexibility, and efficient use of hardware resources.

The FORTH language is organized as a dictionary in memory. The dictionary is comprised of "words," which are the fundamental unit of the language. A word is any string of characters separated by spaces. Each word can be called from the terminal and executed in an interactive mode or can be executed as a part of a compiled algorithm. Because FORTH is a stack-oriented language, all parameters and data are passed through the stack between the words of the program.

A new word can be created and defined in terms of previously defined words and added to the dictionary. Once in the dictionary, the newly defined word is available for use by the programmer. The capability to define new words and add them to the language gives FORTH its extensibility.

The extensibility of FORTH allows the programmer to create a language uniquely designed for each application. FORTH has a precompiled kernel of rudimentary words that handle such tasks as

stack manipulation, program control, terminal I/O, and other functions. The applications programmer typically begins creating "application words" from predefined words in the FORTH kernel. The dictionary is extended with the creation of each new word. As more application words are created, they may be combined to perform more complex functions. This process can be continued until the application program is a single word.

As mentioned previously, FORTH has an interpreter which can be used to test newly created words from the terminal. This is of tremendous value to the programmer as it allows each word to be tested and debugged in an interactive mode. As an example, the use of an ADC requires a series of commands from a controller, which instructs it to perform a conversion and return the results of that conversion. A FORTH word can be defined that reads a value from the ADC and prints the results on the CRT screen. The word can then be called from the terminal. If the conversion is not performed correctly, the word can be changed and retried until it executes in the desired fashion. The immediacy of the trial-and-error process can reduce program development time as much as a factor of 10 over conventional languages.[9]

One feature of FORTH which makes it attractive in small instruments is the minimal amount of memory it requires. The typical FORTH system will include a compiler, assembler, text

---

[9]E. D. Rather and C. H. Moore, FORTH High-Level Programming Technique on Microprocessors, Manhattan Beach, California: FORTH, Inc. (1976).

editor, and utility functions, yet occupy less than 6K bytes of memory. FORTH applications require less memory than the equivalent assembly language code with only a 25% speed reduction. The programmer may elect to use the built-in assembler to write at the code level for time-critical areas of the program.

Preliminary Considerations

The average flowrate was developed in Chapter II. Equation (2-28) shows that the average flowrate is equal to the product of a scaling constant and the time derivative of the vessel volume. The scaling constant, K, is determined by the physical parameters of the vessel and bubbler system. The scaling factor will remain constant for any given vessel. The time derivative of the vessel volume is dynamic and must be known in order to determine the average flowrate.

The time derivative of vessel volume can be estimated by several methods in the computer. One method is to subtract the final level from the initial level and divide by the time interval. This would give a $\Delta V/\Delta t$ over the interval T. While this method is easily understood, it lacks precision due to the uncertainty of the volume measurement. As shown in Chapter II, the accuracy of any volume sample is within ±18 mL. A randomness of 36 mL could lead to variations of up to 50% in the calculated flowrate. In order to minimize the effect of data-point uncertainty, an array is allocated in the computer memory. The array is used to store vessel volume data for later use. As a new data point is taken, it is added to

the array. A time interval of 10 min was selected to allow a sufficiently measurable volume change. This is necessary because the nominal flowrates in the solidification process are small. Consequently, the vessel volume change is imperceptible without a long collection interval. As shown previously, the accuracy of the flowrate can be improved by statistical averaging. The discussion in Chapter II shows that by using 100 samples of data, the precision can be improved by a factor of 10. The method chosen to implement all of the afore-mentioned concepts was a linear regression of the data array.

In order to get 100 data points over the desired 10-min interval, it is necessary to acquire a new sample at 6-s intervals. As each new sample is acquired, it is added to the array and a linear regression is applied to the array. Since each data point is a vessel volume sample and the sampling interval is fixed, the array is a two-dimensional graph of volume versus time.

By using a linear regression on the entire array, the best straight-line fit is found. The straight line is defined to be the average flowrate over the array time interval. The linear regression is implemented by minimizing the error squared or least-squares fit. Development of the least-squares fit algorithm is explained in the following section.

Development of Least-Squares Fit

The slope of the normalized level versus time graph is determined through a least-squares regression. The least-squares

regression forces a straight line to fit a collection of points by minimizing the deviations between the generated straight line and the points. Figure 9 shows an arbitrary set of points which represent the sampled fluid level, y. The horizontal axis is the time in minutes, t. Let the desired straight line be F where F is of the form:

$$F = mt + b \tag{4-1}$$

The error between the straight line and any y is

$$E = F - y$$
$$E = mt + b - y \tag{4-2}$$

Since the errors may be positive or negative, we square the error term to constrain E to positive values. This gives

$$E^2 = (mt + b - y)^2 \tag{4-3}$$

We wish to find values for m and b which will minimize the total errors from F. We do this by taking the partial derivative of the sum of all E^2 terms and setting equal to zero.

The sum of all error terms is

$$\sum E^2 = \sum (mt_i + b - y_i)^2 \tag{4-4}$$

where the subscript i is to indicate discrete values for t and y.

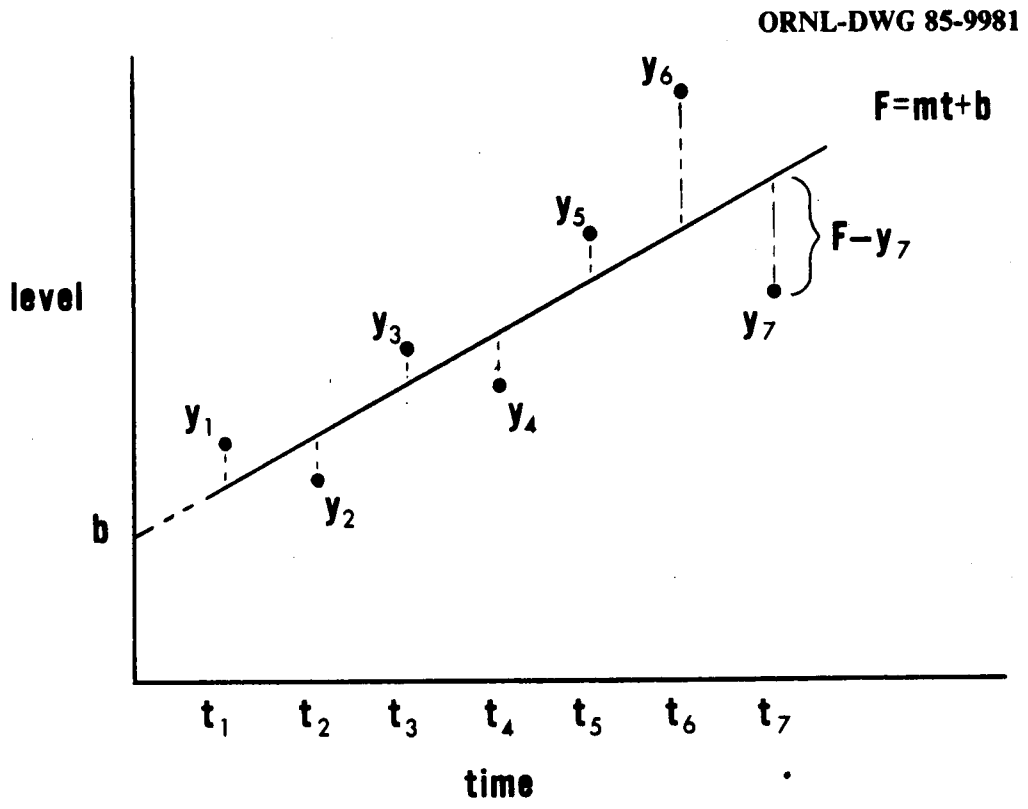


Figure 9. Arbitrary collection of data points illustrating a least-squares fit.

Taking the partial with respect to m and b

$$\frac{\partial \sum E^2}{\partial m} = 2 \sum (mt_i + b - y_i)(t_i) \qquad (4\text{-}5)$$

$$\frac{\partial \sum E^2}{\partial b} = 2 \sum (mt_i + b - y_i) \qquad (4\text{-}6)$$

Setting these two terms to zero

$$\sum (mt_i^2 + bt_i - y_i t_i) = 0, \qquad (4\text{-}7)$$

$$\sum (mt_i + b - y_i) = 0, \qquad (4\text{-}8)$$

Expanding out and recognizing that m is a constant

$$m \sum t_i^2 + \sum bt_i = \sum y_i t_i \tag{4-9}$$

$$m \sum t_i + \sum b = \sum y_i \tag{4-10}$$

If there are n number of points, the summation of b is just nb and we can write

$$m \sum t_i^2 + \sum bt_i = \sum y_i t_i \tag{4-11}$$

$$m \sum t_i + nb = \sum y_i \tag{4-12}$$

We can now separate terms to find expressions for m and b:

$$m = \frac{\sum y_i t_i - \sum bt_i}{\sum t_i^2} \tag{4-13}$$

$$b = \frac{\sum y_i - m \sum t_i}{n} \tag{4-14}$$

If we impose a few physical constraints, the above expressions can be simplified considerably and be easily implemented by numerical techniques.

Since the start of time is arbitrary, the calculation of m is unaffected by the placement of the origin along the time axis. If we force the array to have only an odd number of points, the origin can be placed in the center of the array. Thus if there were seven points, we could choose to define the fourth point t = o. This would mean that points 1, 2, and 3 would have been taken during

negative time in a mathematical sense, but this will not affect the results. This is shown in Figure 10.

It can be seen from the two figures that the data points have not been changed, just the location of the origin. The advantage of this mathematical sleight-of-hand can be seen by re-examining Eq. (4-13).

$$m = \frac{\sum y_i t_i - \sum b t_i}{\sum t_i^2}$$

since t_i represents the sampling interval which is constant, t_i is a series of integers in both directions. This means the sum of t_i is the sum of a positive series of integers and an equal negative series.

$$\sum t_i = \sum t_i + \sum t_{-i}$$

$$= \sum t_i - \sum t_i$$

$$\sum t_i = 0$$

Recognizing this, we can eliminate the $\sum t_i$ terms in Eqs. (4-13) and (4-14). This results in

$$m = \frac{\sum y_i t_i}{\sum t_i^2} \qquad (4\text{-}15)$$

$$b = \frac{\sum y_i}{n} \qquad (4\text{-}16)$$

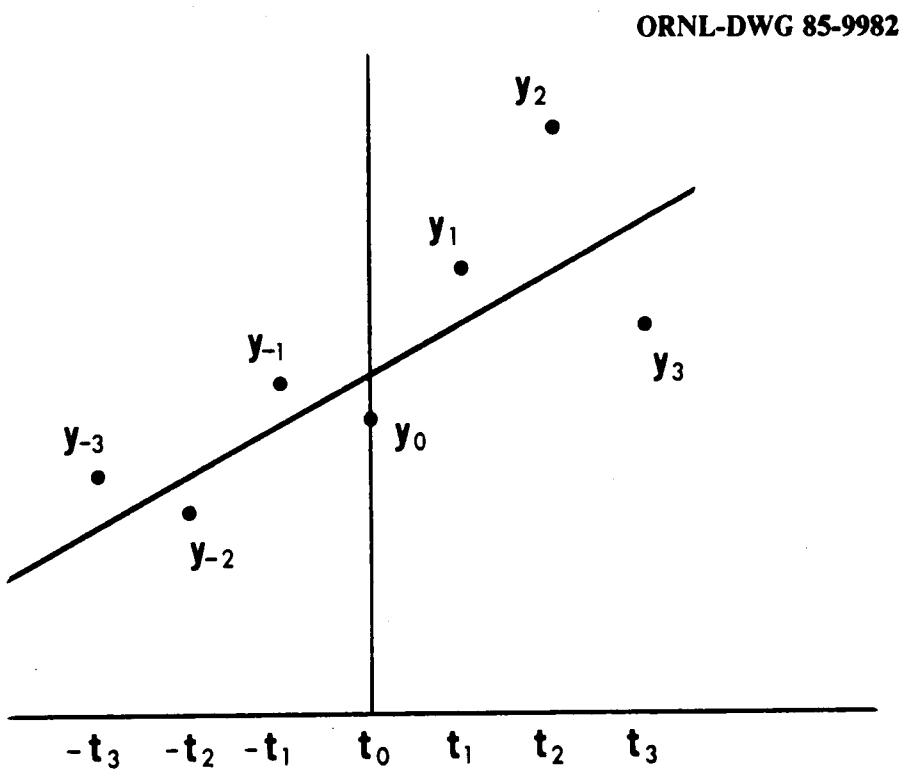


Figure 10. Arbitrary data points after translating origin.

The right side of (4-16) can be seen to be the arithmetic average of the data points.

$$b = \bar{y}$$

The numerical algorithm for m is developed by recalling that the sampling rate for this application is 0.1 min. This lets us rearrange the numerator in Eq. (4-15).

$$\sum y_i t_i = y_{-3}\, t_{-3} + y_{-2}\, t_{-2} + y_{-1}\, t_{-1}$$

$$+ y_0\, t_0 + y_1\, t_1 + y_2\, t_2 + y_3\, t_3$$

Since $t_0 = 0$, the middle term on the right-hand side may be eliminated. Using the numerical values for t,

$$\sum y_i t_i = (y_{-3})(-.3)+(y_{-2})(-.2)+(y_{-1})(-.1)+(y_1)(.1)+(y_2)(.2)+(y_3)(.3)$$

This can be factored to give

$$\sum y_i t_i = (.3)(y_3-y_{-3})+(.2)(y_2-y_{-2})+(.1)(y_1-y_{-1})$$

The denominator of Eq. (4-15) is seen to be the sum of negative terms squared and positive terms squared or simply twice the positive terms squared.

$$\sum t^2 = 2(t_1^2 + t_2^2 + t_3^2) \qquad (4\text{-}17)$$

This gives the final form for calculating the slope of the data points using a numerical least-squares fit.

$$m = \frac{\sum t_i(y_i - y_{-i})}{2 \sum t_i^2} \qquad (4\text{-}18)$$

where $i = 1, 2, 3.....\text{INT}(\frac{n}{2} + 1)$

The effectiveness of this algorithm may be shown by illustration. Consider the data points shown in Figure 11, which have the functional form

$$y = 2x + 5.$$

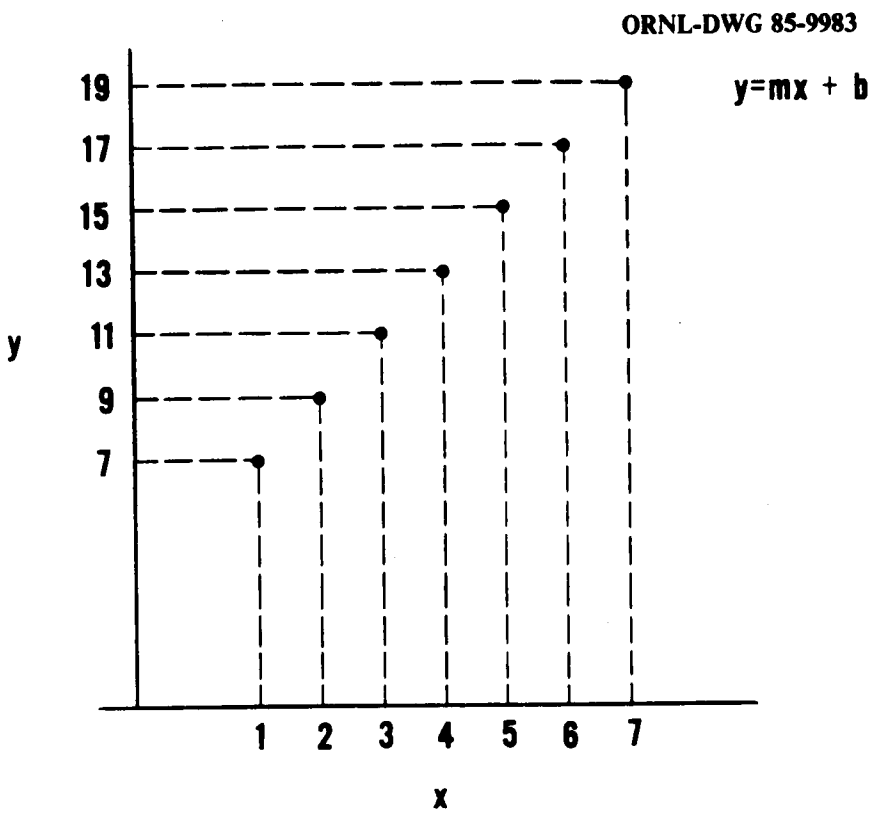


Figure 11. Data points for function y = 2x + 5.

We first translate the horizontal reference frame so that the center data point becomes the new origin. This gives y_0 at x = 4, y_1 at x = 5, y_{-1} at x = 3, etc.

Using these values in Eq. (4-18) gives:

$$m = \frac{(1)(15 - 11) + (2)(17 - 9) + (3)(19 - 7)}{1^2 + 2^2 + 3^2}$$

$$= \frac{4 + 16 + 36}{28}$$

$$m = 2$$

$$b = \frac{7 + 9 + 11 + 13 + 15 + 17 + 19}{7}$$

$$b = 13$$

The value for b is that of y_0 in the numerical computation or y_4 in the physical model. The value of the y intercept can be easily found from b. However, for the flow measurement, only the slope of the density-compensated level is required and b may be neglected.

Program Development

The implementation of Eq. (2-28) to calculate the average flowrate is described in the following section. Since all channels are similiar, only one channel will be discussed.

An overall flow diagram of the software architecture is shown in Figure 12. Since the program is resident in EEPROM, the system begins operation from power-up, when power is applied. The system is initialized and the P FORTH card waits for an interrupt. There are two possible interrupt conditions in this system. An interrupt is generated from a front-panel reset button or from a hardware timer. The front-panel reset button is used to reset an individual channel without affecting the other channels, and the hardware timer is used to generate a 6-s sample interval signal.

When the P-FORTH microprocessor receives an interrupt, it interrogates the system to determine the source of the interrupt. The microprocessor will then execute either a TIMER service routine or a RESET service routine.

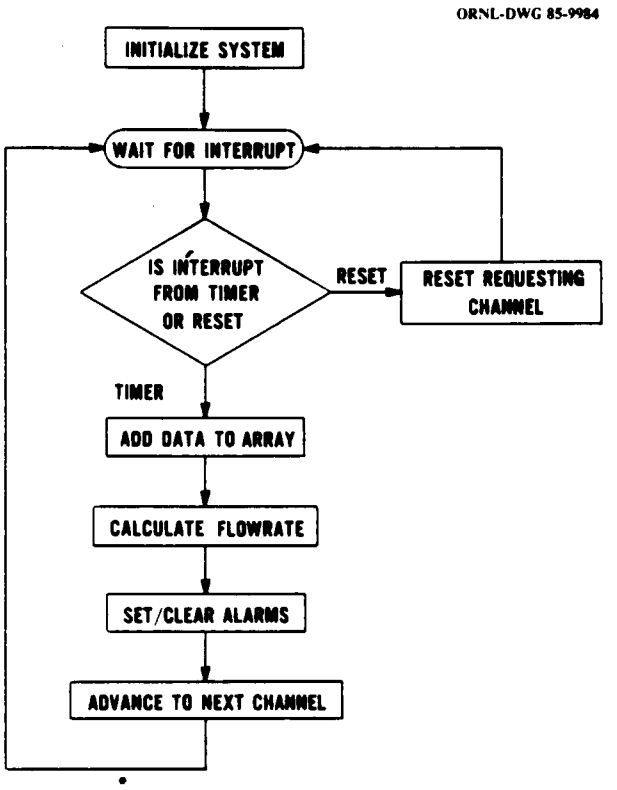


Figure 12. Flowmeter software block diagram.

The TIMER service routine is executed by the FORTH word TIMER and the RESET service routine by the word RESET. This is an example of the self-documenting nature of a well-written FORTH program. These words are easily remembered and their functions are self-explanatory. A complete listing of the FORTH application VOCABULARY and a description of the GLOSSARY are included in the Appendix.

System Initialization

Upon power-up, a reset vector will execute a system initialization as shown in Figure 13 by the the FORTH word RESTART.

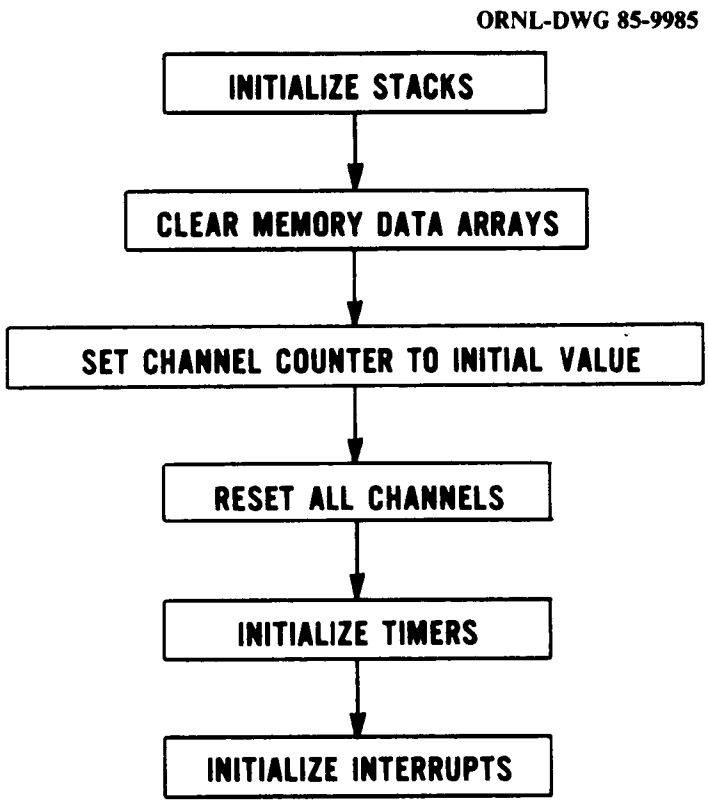


Figure 13. System initialization flow diagram.

RESTART will clear the data stack and the four data arrays. It also resets the channel counter to the first channel and clears each display. RESTART then initializes the hardware timers of the 6522 PIA integrated circuit. It then sets a vector to the interrupt service routine and enables the interrupts. After RESTART has initialized the system, the program goes to the FORTH monitor to await an interrupt condition.

Data Array

The primary element of each channel is the data array in random-access memory (RAM). Each channel is allocated 250 bytes for

storage of the vessel volume samples; 202 bytes are set aside for 101 volume data-points, and the remainder is used for holding channel information such as flow, density, and array size. A map of a typical array is shown in Figure 14.

The array can contain up to 101 volume samples. The data counter register contains the number of data points in the array, and is used by the program to calculate the least-squares fit of the data points.

RESET Service Routine

If the interrupt source was from a front-panel reset, the microprocessor will reset that channel if it has finished the TIMER service routine. The RESET service routine is shown in Figure 15. The RESET/ANALOG board latches a channel reset command and generates an interrupt signal. The microprocessor then reads the requesting channel from the RESET/ANALOG board.

After determining the requesting channel, the microprocessor uses the FORTH word RESET to execute a channel reset. RESET expects the number of the requesting channel to be on the data stack. RESET will clear the data counter, flow register, density register, and the front-panel display of the requesting channel. It then takes an initial density sample and stores it into the density register. RESET then returns to the monitor.

The TIMER service routine has highest priority, so the microprocessor will not acknowledge a RESET interrupt until it has completed the TIMER service.

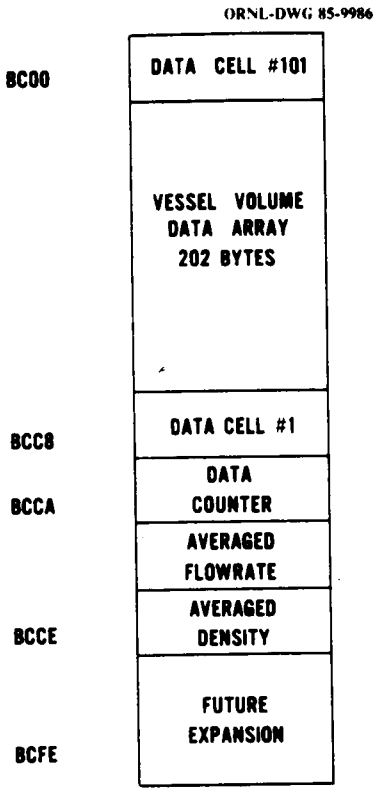


Figure 14. Memory map of Channel One data array.

TIMER Service Routine

If an interrupt is generated from the hardware timer, the program executes a TIMER service routine as shown in Figure 16. A timer-generated interrupt is issued at 1.5-s intervals. The program executes TIMER for the active channel, then advances to the next channel and waits for the next timer interrupt. Since there are 4 channels, any channel is updated every 6 s. All channels use the same service routine, with incoming and outgoing data being vectored to the appropriate channel locations in the memory array.

The first action of TIMER is to reset the hardware timer on the 6522 PIA. Then a sample of the density-compensated level is taken

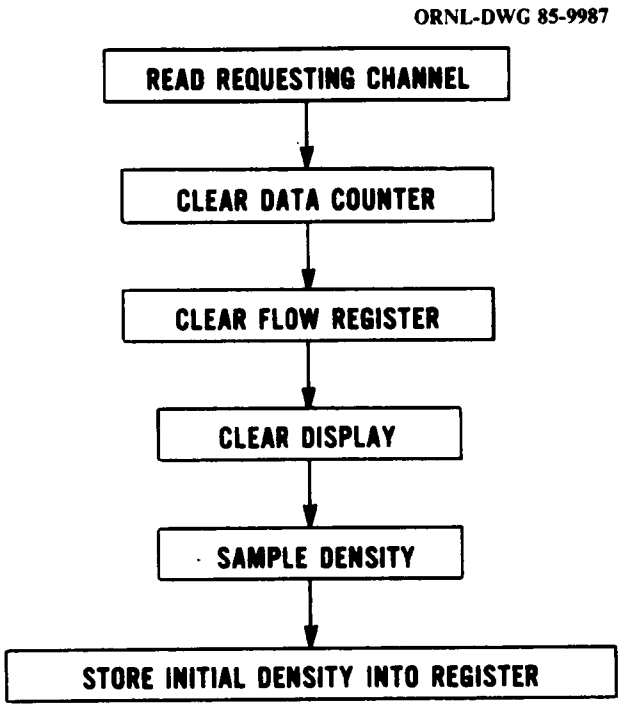


Figure 15. RESET interrupt service flow diagram.

from the containment vessel. The vessel level is taken by making an analog-to-digital conversion from the differential pressure transmitter connected to the bubbler system level tube. The level is sampled 100 times and averaged to improve measurement precision and reduce the effects of bubbles and artifacts on the measurement.

The density sample is taken similiarly except that only 20 samples are averaged. This density value is averaged with the previous density value and stored in the data array for averaging with the next sample. The recursion uses equal parts of the old and new densities.

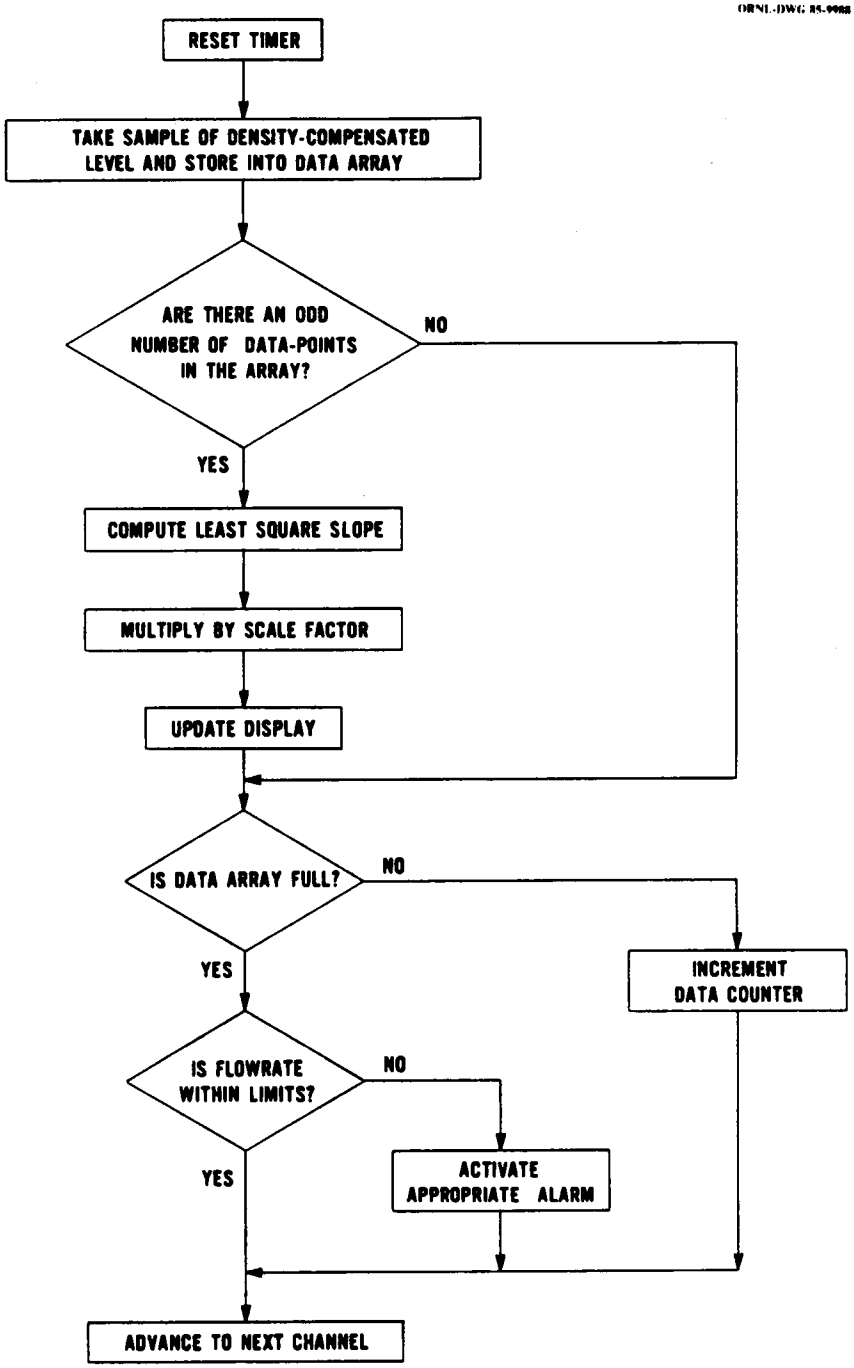


Figure 16. TIMER interrupt service flow diagram.

The average level is multiplied by 5 and divided by 5 times the density plus a constant. This is the density-compensated level or the Q term from Eq. (2-27).

The memory array is shifted by one cell to allow room for the newly acquired sample. Data cell #1 moves to cell #2, cell #2 moves to cell #3, and so on. The most recently acquired sample is written into cell #1 and the data counter is checked to see if there is an odd number of samples in the array. If the number of data samples in the array is odd, the program computes the average flowrate and updates the display. If there is an even number of data samples, then a least-squares fit cannot be done on the array.

The program computes the average flowrate by calculating the best fit slope of the points in the array. The slope is multiplied by a scaling factor and averaged with the previous flow. The recursion is such that a change of 0.1 mL/min is seen on the display if a flow increase of 0.1 mL/min is maintained for three consecutive samples. This prevents slight perturbations in the data array from affecting the displayed flowrate. It is desirable to stabilize the display so as not to create ambiguous readings that might mislead a control room operator. The slope of the array is found using the FORTH word LSFIT. LSFIT takes the data counter value, indexes into the center of the array, and calculates the slope $\Delta Q/\Delta t$ by the previously discussed least-squares fit algorithm.

The averaged flowrate is rounded off to the nearest 0.1 mL/min and written to the appropriate digital front-panel display.

The data counter is checked to see if the array contains 101 datapoints, indicating that the array is full. If so, the flowrate is compared with upper and lower flow limits to ascertain if the flow is within the acceptable range. If the flowrate is lower than 4.0 mL/min, an alarm is activated which closes a solid-state relay. The relay closure illuminates an UNDERFLOW warning lamp for the corresponding channel. A flow above 14.0 mL/min will similiarly cause an OVERFLOW to illuminate. No action is taken if the flow is within tolerance.

The flowrate check is not performed unless the array is full. This is to allow start-up transients to decay before the alarms are enabled. If the data counter shows the array has less than 101 datapoints, the data counter is incremented.

The final action by the TIMER service routine is to increment the channel counter. This advances the program to the next channel and controls the sequential multiplexing of the four channels. The active channel is stored in RAM in the variable CHAN.

After the interrupt service routines have been completed, the program returns to the FORTH monitor. The monitor continuously scans the input for the next interrupt, whereupon the sequence is repeated.

CHAPTER V

EXPERIMENTAL RESULTS

A series of tests were conducted to verify the performance of the flow monitor. Tests were run which proved the goodness of fit of the least-squares algorithm and established the actual operational characteristics of the device under field conditions.

Analog Input Filter

The use of the 125-Hz analog filter in the analog-reset board was justified by comparing data acquired with the filter and without. The analog filter is located in the input signal line in series with the differential-pressure-to-current converter and the analog-to-digital converter. Data taken without the filter are shown in the upper trace of Figure 17. The reduction of artifacts is readily apparent in the lower trace.

Verification of Model

A test vessel was constructed to simulate the actual containment vessel. The test vessel had an inside diameter of 6 13/32 in. A bubbler system shown in Figure 18 was installed with the density separation, D, adjusted to 10 in. The differential-pressure-to-current converter, P_L, had a range of 32 in. of water and P_D had a range of 20 in. of water. The test vessel had an outlet on the bottom of the tank with a fluid rotameter and needle valve to measure and adjust outgoing flow.

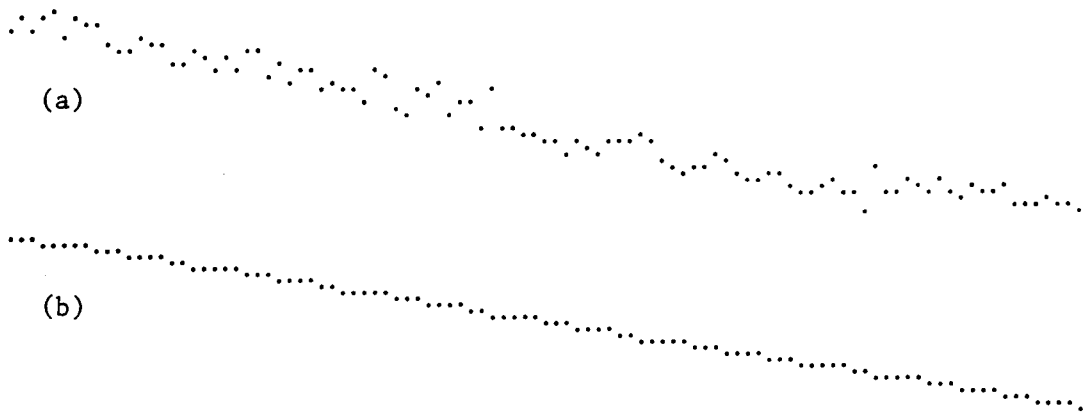


Figure 17. Effect of analog filter on input signal. (a) Input signal without 125-Hz low-pass filter; (b) input signal with filter.

The tank was filled with water, and the flow was collected and measured in the graduated cylinder. The collected water volume was compared with the flow monitor readings.

Scale Factor

The scaling factor was tested by comparing the theoretically derived value with an experimentally determined value. Since the scale factor is

$$K = \frac{4}{5} \pi R^2 H_L$$

from Eq. (2-26), by inserting the appropriate parameters from the test vessel

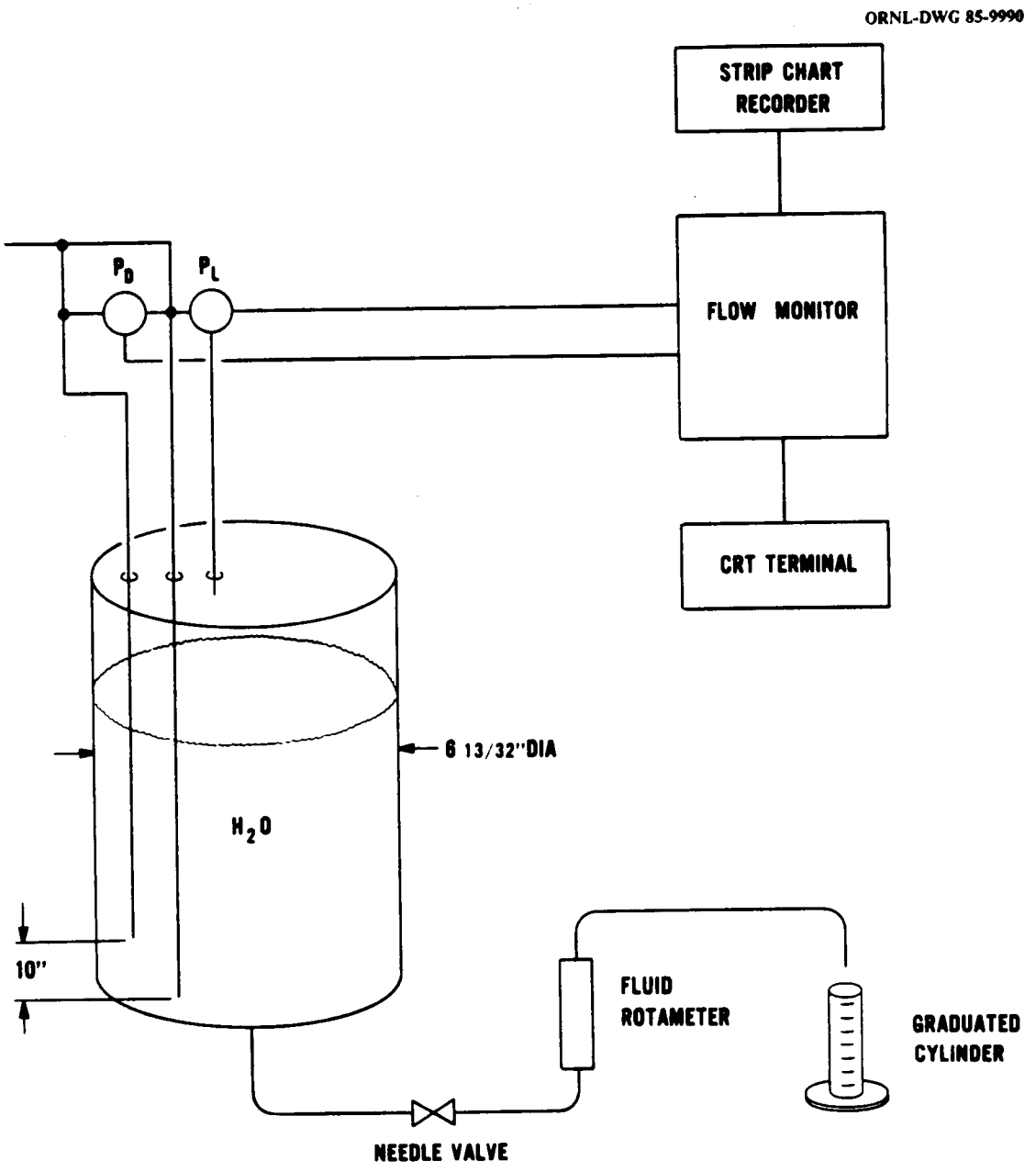


Figure 18. Diagram of experimental apparatus.

$$K = \left(\frac{4\pi}{5}\right)\ (32\ \text{in.})\ \left(\frac{6.41}{2}\text{in.}\right)^2\ \left(2.54\ \frac{\text{cm}}{\text{in.}}\right)^3,$$

$$K = 13538\ \text{mL}$$

The scale factor was determined experimentally by sampling the vessel volume with the ADC and recording the result. The needle valve of the vessel was opened and 100 mL was collected in the graduated cylinder. The needle valve was then closed and the vessel volume was measured. The difference between the volumes is divided into the volume collected to determine the scale factor. This procedure was used with the vessel nearly full, half full, and nearly empty. The results were averaged to get an experimental scale factor:

$$K = 13898\ \text{mL}$$

The theoretical scale factor differs by:

$$\text{Error} = \frac{13898 - 13538}{13898} \times 100$$

$$\text{Error} = 2.6\%$$

This is within acceptable limits of the measurement precision.

Least-Squares Algorithm

The accuracy of the least-squares-fit algorithm was tested by loading the array with a predetermined sequence of numbers. This sequence of numbers was calculated from the mathematical slope that

would correspond to an arbitrary flowrate. The least-squares routine was run, and the results of the least-squares fit were compared with the mathematical flowrates. The test was run for 60 slopes within the range of anticipated flowrates. The results of this experiment showed the least-squares-fit algorithm to be within 0.2 mL of the mathematical flow. All deviations from the theoretical flow were negative.

Statistical Sampling

The sampling technique used in the flow monitor reduces sampling error through averaging. This is done by making many individual analog-to-digital conversions and averaging the results. A frequency-of-occurrence distribution graph for a density measurement is shown in Figure 19. The vertical axis is the number of occurrences, and the horizontal axis is a normalized digital value. The distribution on the left is from a single-sample process. The distribution on the right is similiar except that 20 single samples were averaged together. A statistical comparison of the two distributions is shown below.

	Mean	Standard deviation
Single sample	2.7	0.84
Averaged sample	2.6	0.65

It was found through repeated tests that the standard deviation of the sampling distribution was reduced through averaging. No

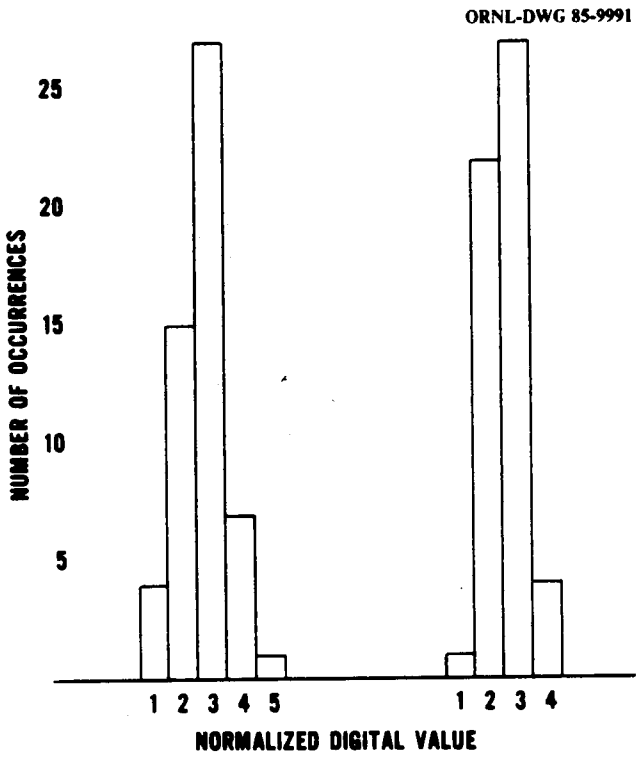


Figure 19. Frequency distribution histogram for data-sampling process.

correlation could be established between the number of samples averaged and the amount of reduction. This is probably an indication that the sampling process is not random. The average standard deviation for a single-sample measurement was 1.1, while the average deviation for the multi-sample measurement was 0.8.

Operational Tests

Operational tests were run on the instrument to verify accuracy under field conditions. The needle valve was set to an arbitrary flowrate within the anticipated normal operating range. The water output was collected in a graduated cylinder for 10 min. The

cylinder was then removed and the collected volume recorded. The collected volume was then compared with the flow monitor. Tests were conducted from a start-up condition with the array initially empty and under steady-state conditions. In the start-up tests, the collected volume was compared with the results of a least-squares-fit slope and with an end-point-derived average slope. The results of this group of tests are shown in Figure 20. The three traces show the deviations of the flowrate monitor from the actual flowrate. The actual flowrate is defined as the collected volume divided by the collection period.

The upper trace shows the deviations in mL/min of the end-point derived slope. This slope was calculated as the difference between the initial array entry and the final array entry. The middle trace is the deviation of the least-squares slope from a start-up condition. The lower trace shows the deviation of the least-squares slope under steady-state conditions that would be the normal operating mode of the instrument. The standard deviations of the conditions are shown below.

Method	Standard Deviation
End-point transient	0.586
Least-squares transient	0.18
Least-squares steady-state	0.098

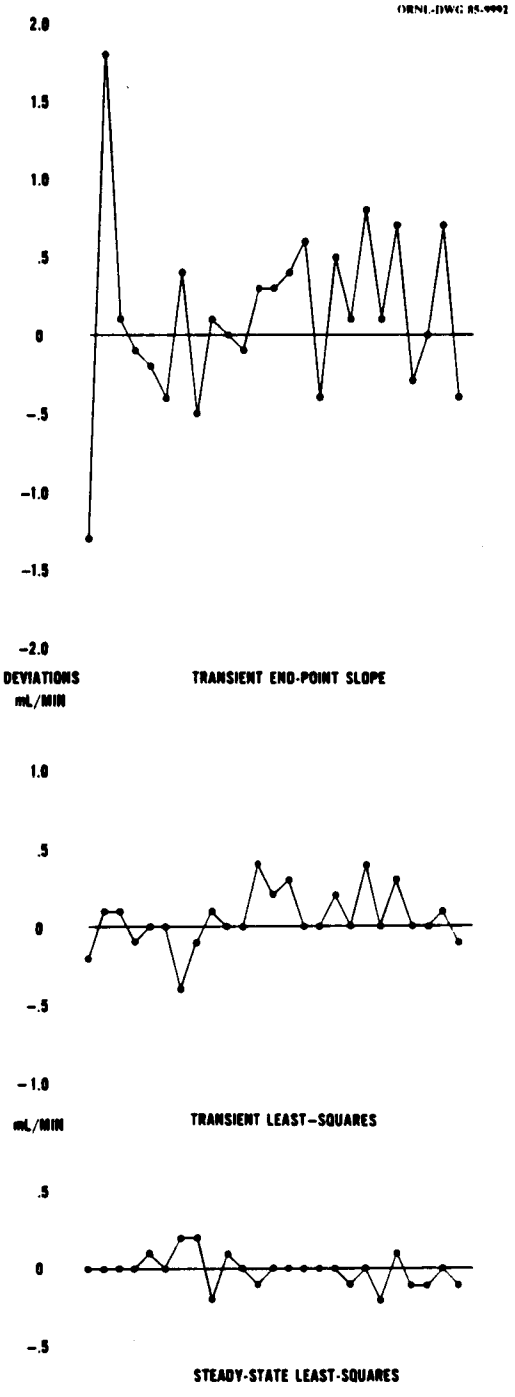


Figure 20. Deviations of calculated flow from measured flow.

The standard deviation of the steady-state, least-squares method is seen to be nearly 1/6 that of the end-point measurement. This differs from the expected reduction of 1/10 predicted in Chapter II. This difference probably is due to the end-point measurement not deviating as much as expected. It is likely that the pressure sensor accuracy was better than the rated 0.25%.

Transient Response

The transient response of the monitor was approximately second order. The overshoot was variable with a maximum value of 200%. The high overshoots occur when the array has few data points and quickly subside as more data are taken. The typical 5% settling time was 5 min.

Long-term Stability

A measure of the long-term stability of the instrument was made by closing the needle valve and recording the flowrate on a chart recorder. Any deviation from zero flow would be an inaccuracy of the system. The maximum deviation seen over an 18-h period was 0.4 mL/min.

CHAPTER VI

SUMMARY AND CONCLUSIONS

A flowrate monitor was developed to measure the average flow of a variable-density fluid from a cylindrical holding tank. The flowrate monitor uses an analog-to-digital converter to sample the level and density of the fluid. A single-board STD bus computer normalizes these signals to a density-compensated level. The density-compensated level sample is stored in a data array in the computer memory. At 6-s intervals a new sample is taken and added to the array. As each new sample is added, the oldest sample is dropped. The array contains the history of density-compensated level for the most recent 10-min period. A linear regression is run on the array to determine the best fit of a straight-line slope of level versus time. The slope is defined as the average change of level in the holding tank. The slope is multiplied by a dimensional constant and averaged with the two preceeding slopes. This number is displayed as the average flowrate from the holding tank. An alarm is activated if the flowrate exceeds prescribed limits of operation.

The intended function of the flowrate monitor is to provide supervisory monitoring of a chemical process. Deviations from normal are reported to a control room operator via an annunciator panel. The flowrate monitor could easily be adapted to provide process control input to allow automatic regulation of flow.

The monitor met or exceeded all required specifications. It has been reliable in long-term tests conducted to date. An improved sensor would increase the instrument accuracy. The pressure-to-current converter used for testing had an accuracy of 0.25%. P/I converters with 0.1% accuracy are available at higher cost.

Future work might include checking the standard deviation of the array. It is possible to make a numerical calculation of the statistical characteristics of the array. The instrument could respond with an error condition if the standard deviation exceeded a preset limit. Upper and lower limits could be established based on the flow range and sensor accuracy. Samples that fell outside these limits could indicate a defective component in the measurement process.

Computational speed is the limiting factor for additional improvements because numerical statistics require large amounts of data crunching. Increasing the workload of the microprocessor must be paid for by increasing the clock speed. This becomes more feasible as faster and more powerful microprocessors become available.

LIST OF REFERENCES

LIST OF REFERENCES

1. Model GC-781 Moore Pneumatic to Electric Transducer, Data Sheet GC-781, Spring House, Pennsylvania: Moore Products Co. (1982).

2. Thornton C. Fry, Probability and Its Engineering Uses, 2nd edition, Princeton, New Jersey: D. Van Nostrand Co., p. 342 (1965).

3. P-FORTH Users Manual, Minneapolis, Minnesota: Peopleware Systems, Inc. (1982).

4. RTI-1260 Users Manual, No. AC1563. Norwood, Massachusetts: Analog Devices, Inc. (1981).

5. OPTO-22 Data Book, 1st ed., Huntington Beach, California: OPTO-22, Inc. (1983).

6. 7507 General Purpose I/O Interface Card Users Manual, Monterey, California: Pro-Log Corp. (1982).

7. 7602 Output Port Card Users Manual, Monterey, California: Pro-Log Corp. (1981).

8. 7904 Decoded Utility I/O Card Users Manual, Monterey, California: Pro-Log Corp. (1982).

9. E. D. Rather and C. H. Moore, FORTH High-Level Programming Technique on Microprocessors, Manhattan Beach, California: FORTH, Inc. (1976).

APPENDICES

APPENDIX A

FORTH APPLICATION PROGRAM LISTING

```
2 VARIABLE CHAN

2 VARIABLE ISUM

0152H CONSTANT ALARM

01FFH CONSTANT RPORT

004DH CONSTANT IFR

0040 CONSTANT LOWER

0140 CONSTANT UPPER

CODE CONV
                PULA
                PULB            ;  MOVE STACK TO ACCUMULATOR
     010BH      STAB            ;  SELECT MUX CHANNEL
                BEGIN
     010D       LDAA
                MI              ;  IS CONVERSION COMPLETE?
                NOT UNTIL          REPEAT UNTIL COMPLETE
     010DH      LDAA            ;  READ UPPER BYTE
     010C       LDAB            ;  READ LOWER BYTE
                PSHB
                PSHA            ;  POP TO STACK
                NEXT JMP        ;  RETURN TO FORTH
                END-CODE

CODE J

     008EH      LDX             ;  LOAD RETURN STACK ADDRESS INTO
                                   INDEX
                INX
                INX
                INX
                INX
                INX
                INX             ;  ADD 6 TO LOCATE 3RD ITEM
     C012H      JMP             ;  MOVE TO STACK, RETURN TO FORTH
                END-CODE

SELECT                  <BUILDS   DOES>  CHAN @
                        2  *  +   @  ;
```

```
SELECT      'ARRAY

                    BCC8, BDC8, BEC8, BFC8,

SELECT      MASK

                    FC, F3, CF, 3F,

SELECT      SCALOR

                    14098, 14098, 14098, 14098,

:SHIFT          'ARRAY 200 - DUP  2 - 202 CMOVE;

:COUNTER        'ARRAY 2 + ;

:FLOW           'ARRAY 4 + ;

:DENS           'ARRAY 6 + ;

:ROUND          10 /MOD SWAP 4 > IF 1+ THEN ;

:MEMCLR         BBC8H 0437H ERASE ;

:LCON           CHAN @ 2 * CONV ;

:DCON           CHAN @ 2 * 1+ CONV ;

:S/             U/ SWAP DROP ;

:SCALE          SCALOR 1000 * / ;

:DENSITY        00 00 20 0 DO DCON 0 D+ LOOP
                2 S/ 0 DENS @ 0 D+
                2 S/ DUP DENS ! ROUND ;
:LEVEL          00 00 100 0 DO LCON 0 D+ LOOP 100 S/ ;

:ADCON          LEVEL 5 * 10,000 DENSITY 5 *
                12288 + */ ;

:LSFIT          'ARRAY OVER 1 - - >R
                0  ISUM ! 1+ 2 /
                0 0 ROT 1 DO
                I I * ISUM +!
                J I 2 * + @
                J I 2 * - @ -
                100 I * M* D+ LOOP
                R> DROP DABS ISUM @ S/ 2 / ;
```

```
:DISPLAY        DUP 1000 < IF 100 /MOD
                256 * SWAP 10 /MOD 16 * + +
                ELSE -1 THEN
                CHAN @ 2 * 256 + ! ;

:LIM-BIN        1 CHAN @ - DUP IF 0 DO 4 * LOOP THEN;

:WITHIN         FLOW @ DUP LOWER < SWAP UPPER >

                2 * + ;

:AVG            FLOW @ SWAP OVER - 3 / +
                DUP FLOW ! ;

:?LIMIT         WITHIN LIM-BIN * ALARM C@
                MASK AND OR ALARM C! ;

:FLOWRATE       COUNTER @ LSFIT SCALE
                AVG ROUND DISPLAY ;

:TIMER          0049H C! SHIFT ADCON 'ARRAY !
                COUNTER @ 2 MOD IF FLOWRATE THEN
                COUNTER @ 65H = IF ?LIMIT ELSE 1 COUNTER +!
                THEN CHAN @ 3 = IF 0 CHAN !
                ELSE 1 CHAN +! THEN ;

:RESET          CHAN @ SWAP CHAN ! 0 COUNTER !
                0 FLOW ! 0 DISPLAY
                DCON 10 * DENS !
                CHAN ! ;

:CHAN?          40H @ DROP RPORT C@
                0 RPORT C! ;

:RESET-ALL      4 0 DO I RESET LOOP ;

:INTR           OFFIRQ IFR C@ 20H AND IF
                TIMER THEN IFR C@ 01 AND IF
                CHAN? RESET 01 IFR C! THEN
                ONIRQ QUIT ;

CODE INTLINK
                BA6D #LDS,      ; clear data stack
                NEXT # LDX,
                PSHX,           ; load NEXT onto data stack
                1 INTR
                2 - #LDX,       ; put address of INTR in index register
                8C STX,         ; move INTR address to I register
                RTS,            ; return from subroutine
                END-CODE        ; return to FORTH and execute INTR
```

```
:RESTART        SP! SETIRQ INTLINK 0
MEMCLR          0 CHAN ! RESET-ALL
E0 4B C!
BF 42 C!
FF 43 C!
627F 44 !
A1 4E C!
16 48 C!
00 49 C!
02 4C C!
CHAN? DROP DECIMAL ONIRQ ;
AUTO RESTART
```

FORTH APPLICATION GLOSSARY

CHAN - 2-byte variable which contains the currently active channel.

ISUM - 2-byte variable used in LSFIT to accumulate the sum of time periods squared.

ALARM - 0152 HEX constant of the ALARM card I/O port.

RPORT - 01FF HEX constant of the RESET card I/O port.

IFR - 04D HEX constant of the 6522 interrupt flag register.

CONV - (n--n) assembly language routine to perform an analog-to-digital conversion. Expects the channel number to be on the stack and leaves the results on the stack.

J - Used during a DO loop to copy the third item on the return stack to the parameter stack.

SELECT - Defining word which allows creation of memory arrays. When words created by SELECT are executed, elements in the array of the active channel are readily available.

ARRAY - (--n) An array created by SELECT which holds the addresses of data cell #1 for each channel. When 'ARRAY is executed, it reads the active channel from CHAN and leaves the corresponding channel address on the stack.

MASK - (--n) An array created by SELECT which contains a bit pattern to mask out the alarm functions of inactive channels. Used in ?LIMIT.

SCALOR - (--n) An array created by SELECT which returns the scaling factor for the active channel. Used in FLOWRATE.

SHIFT - Moves the elements in the data array down 2 bytes. Cell #1 moves to cell #2, cell #2 moves to #3 etc. Begins with cell #100 and works back to cell #1. Cell #101 is eliminated.

COUNTER - Register in data array which holds the size of the array.

FLOW - Register in data array which contains the current flowrate of that channel.

DENS - Register in data array which holds the channel density.

MEMCLR - Used in initialization to clear all data arrays.

LCON - (--n) Performs an analog-to-digital conversion for the level of the active channel and leaves result on stack.

DCON - (--n) Same as above except does conversion of density of active channel.

S/ - (dn--n) Divides a double-precision number by a single-precision quotient on the stack.

SCALE - (n--n) Multiplies number on stack by the scale factor of active channel and leaves results on stack.

ROUND - (n--n) Takes number on stack, rounds off to nearest tenth, and leaves results on stack.

DENSITY - (--n) Makes 20 density measurements of active channel. Takes the average of the 20 measurements and then averages with the previous sampling period density.

LEVEL - (--n) Makes 100 level measurements and leaves the average on the stack.

ADCON - (--n) Takes 5 times DENSITY and adds 12288, then divides into 5 times LEVEL. This is the value stored in the data array as the density-compensated level.

LSFIT - (n--n) Takes the array size on the stack and does a least-squares fit on the data array. Leaves the slope on the stack.

AVG - (n--n) Takes 1/3 of flow from stack and adds to 2/3 of value in FLOW. Places new value into FLOW and leaves a copy on stack.

DISPLAY - (n--) Takes number from stack and performs a decimal-to-hex conversion. Then writes value to the active channel display.

LIM-BIN - (--n) Generates a binary limit for the active channel. Used to vector the correct alarm location in ?LIMIT.

WITHIN - (n--f) Takes a number from stack and checks to see if the number is within the fixed upper and lower limits. Leaves a flag 0, 1, or 2 to indicate normal, under, or over condition.

?LIMIT - (n--) Takes flowrate from stack and compares to upper/lower limits. Activates the appropriate alarm if necessary.

FLOWRATE - (--n) Computes the flowrate of the active channel using LSFIT, SCALE, AVG, and ROUND. Updates display with flowrate and leaves copy on stack.

TIMER - TIMER interrupt service routine which resets hardware timer, takes density-compensated level sample, and calculates flowrate.

RESET - (n--) RESET interrupt service routine. Clears data counter and flow register and sets density register to initial value.

CHAN? - (--n) Returns channel number when a front-panel reset button has been depressed.

RESET-ALL- Used in initialization to RESET all channels.

INTR - Interrupt service routine which polls the interrupt flag register to determine if a RESET or TIMER interrupt has occurred.

INTLINK - Assembly language word which vectors to INTR upon receipt of an interrupt.

RESTART - Initialization word which clears the system and sets all operating parameters to their initial values. Executes automatically from power-up to initialize system and begin execution of flowrate program.

APPENDIX B

Portions of the FIG-* FORTH Glossary Used in the P-FORTH Model

The FIG-* FORTH Glossary was developed by the
Forth Interest Group, San Carlos, Calif.

P-FORTH GLOSSARY

Explanation of Glossary

In this glossary, word names are listed in two sub-sequences:

1. The main sub-sequence is for all those word names that include letters or numbers. Within this group word names are in order of their alphanumeric content. (When more than one word name has the same alphanumeric content, the ASCII sequence for the rest of the name determines the relative order.)

2. The other sub-sequence is for those names that do not contain letters or numbers. This sub-sequence is in ASCII order.

   The non-alphanumeric sub-sequence comes first in the Glossary. Thus, the following words appear in the Glossary in this order:

   ! # / ? [ ] 0< 0= ABORT <BUILDS C! CFA XOR

The first line of each entry shows a symbolic description of the action of the procedure on the parameter stack. The symbols indicate the order in which input parameters have been placed on the stack. Three dashes "---" indicate the execution point; any parameters left on the stack are listed. In this notation, the top of the stack is to the right.

The symbols include:

addr	memory address
b	8-bit byte (i.e., hi 8 bits zero)
c	7-bit ASCII character (hi 9 bits zero)
d	32-bit signed double integer, most significant portion with sign on top of stack.
f	boolean flag. 0=false, non-zero=true
ff	boolean false flag = 0
n	16-bit signed integer number
u	16-bit unsigned integer
tf	boolean true flag = non-zero
tp	true part
fp	false part

The capital letters on the right show definition characteristics:

C	May be used only within a colon definition. A digit indicates number of memory addresses used, if other than one.
E	Intended for execution only.
I	Indicates that the word is immediate and will execute even when compiling unless special action is taken.
U	A user variable.

Unless otherwise noted, all references to numbers are for 16-bit signed integers. The high byte of a number is on top of the stack, with the sign in the leftmost bit. For 32-bit signed double numbers, the most significant part (with the sign) is on top.

All arithmetic is implicitly 16-bit signed integer math, with error and underflow indication unspecified.

P-FORTH Glossary

! n addr ---

Store 16 bits of n at address. Pronounced "store."

' --- addr I

Used in the form: ' nnn
Leaves the parameter field address of dictionary word nnn. As a compiler directive, executes in a colon-definition to compile the address as a literal. If the word is not found after a vocabulary search, an appropriate error message is given. Pronounced "tick."

* n1 n2 --- prod

Leave the signed product of two signed numbers.

*/ n1 n2 n3 --- n4

Leave the ratio n4 = n1*n2/n3 where all are signed numbers. Retention of an intermediate 31-bit product permits greater accuracy than would be available with the sequence
n1 n2 * n3 /

+ n1 n2 --- sum

Leave the sum of n1+n2.

+! n addr ---

Add n to the value at the address. Pronounced "plus-store."

, n ---

Store n into the next available dictionary memory cell, advancing the dictionary pointer. Pronounced "comma."

- n1 n2 --- diff

Leave the difference of n1-n2.

/ n1 n2 --- quot

Leave the signed quotient of n1/n2.

: I,E

Used in the form called a colon-definition: : cccc ... ;

Creates a dictionary entry defining cccc as equivalent to the following sequence of FORTH word definitions '...' until the next ';' or ';CODE'. The compiling process is done by the text interpreter as long as STATE is non-zero.

; --- I,C

Terminate a colon-definition and stop further compilation.

< n1 n2 --- f

Leave a true flag if n1 is less than n2; otherwise leave a false flag.

= n1 n2 --- f

Leave a true flag if n1=n2; otherwise leave a false flag.

> n1 n2 --- f

Leave a true flag if n1 is greater than n2; otherwise leave a false flag.

@ addr --- n

Leave the 16-bit contents of address. Pronounced "fetch."

0= n --- f

Leave a true flag if the number is equal to zero, otherwise leave a false flag.

1+ n1 --- n2

Increment n1 by 1.

2+ n1 --- n2

Leave n1 incremented by 2.

ABS n --- u

Leave the absolute value of n as u.

AND n1 n2 --- n3

Leave the bitwise logical AND of n1 and n2 as n3.

AUTO ---

Used in the form: AUTO <Name> to initialize a pointer in EEROM. On power-up if target switch is on <Name> is executed immediately.

BEGIN --- addr n (compiling) I

Occurs in a colon-definition in the form:

```
BEGIN ... UNTIL
BEGIN ... AGAIN
BEGIN ... WHILE ... REPEAT
```

At run time, BEGIN marks the start of a sequence that may be executed repetitively. It serves as a return point from the corresponding UNTIL, AGAIN, or REPEAT. When executing UNTIL, a return to BEGIN will occur if the top of the stack is false; for AGAIN and REPEAT a return to BEGIN always occurs.

At compile time, BEGIN leaves its return address and n for compiler error checking.

<BUILDS C

Used within a colon-definition: : cccc <BUILDS ...
DOES> ... ;

Each time cccc is executed, <BUILDS defines a new word with a high-level execution procedure. Executing cccc in the form: cccc nnnn uses <BUILDS to create a dictionary entry for nnnn with a call to the DOES> part for nnnn. When nnnn is later executed, it has the address of its parameter area on the stack and executes the words after DOES> in cccc. <BUILDS and DOES> allow run-time procedures to be written in high-level rather than assembler code (as required by ;CODE).

C! b addr ---

Store 8 bits at address. Pronounced "c-store."

C, b ---

Store 8 bits of b in the next available dictionary byte, advancing the dictionary pointer.

C@ addr --- b

Leave the 8-bit contents of memory address. Pronounced "c-fetch".

CMOVE from to count ---

Move the specified quantity of bytes beginning at address from to address to. The contents of address from is moved first, proceeding toward high memory.

(;CODE) --- C

The run-time procedure, compiled by ;CODE , that rewrites the code field of the most recently defined word to point to the following machine code sequence. See ;CODE. .

;CODE I,C,

Used in the form: : ccc .... ;CODE (Assembly mnemonics.)

Stop compilation and terminate a new defining word cccc by compiling (;CODE). Set the CONTEXT vocabulary to ASSEMBLER, assembling to machine code the following mnemonics.

When cccc later executes in the form: cccc nnnn, the word nnnn will be created with its execution procedure given by the machine code following cccc. That is, when nnnn is executed, it does so by jumping to the code after nnnn. An existing defining word must exist in cccc prior to ;CODE.

CODE I

A defining word to define words in assembly language.

CONSTANT n ---

A defining word used in the form: n CONSTANT cccc to create word cccc, with its parameter field containing n. When cccc is later executed, it will push the value of n to the stack.

CONTEXT --- addr U

A user variable containing a pointer to the vocabulary within which dictionary searches will first begin.

!CSP

Save the stack position in CSP. Used as part of the compiler security.

?CSP

Issue error message if stack position differs from value saved in CSP.

CSP --- addr U

A user variable temporarily storing the stack pointer position, for compilation error checking.

CURRENT --- addr U

A user variable containing a pointer to the vocabulary to which new definitions are compiled.

D+ d1 d2 --- dsum

Leave the double number sum of two double numbers.

DABS d --- ud

Leave the absolute value ud of a double number.

(DO) C

The run-time procedure compiled by DO which removes the loop control parameters to the return stack. See DO.

DO n1 n2 --- (execute)
addr n --- (compile) I,C2

Occurs in a colon-definition in form:

DO ... LOOP

At run-time, DO begins a sequence with repetitive execution controlled by a loop limit n1 and an index with initial value n2. DO removes these from the stack. Upon reaching LOOP, the index is incremented by one. Until the new index equals or exceeds the limit, execution loops back to just after DO; otherwise the loop parameters are discarded and execution continues ahead. Both n1 and n2 are determined at run-time and may be the result of other operations. Within a loop, 'I' will copy the current value of the index to the stack. See I, LOOP, LEAVE.

DOES>

A word which defines the run-time action within a high-level defining word. DOES> alters the code field and first parameter of the new word to execute the sequence of compiled word addresses following DOES>. Used in combination with <BUILDS. When the DOES> part executes it begins with the address of the first parameter of the new word on the stack. This allows

interpretation using this area or its contents. Typical uses include the FORTH assembler, multidimensional arrays, and compiler generation.

DROP n ---

Drop the number from the stack.

-DUP n1 --- n1 (if zero)
n1 --- n1 n1 (non-zero)

Reproduces n1 only if it is non-zero. This is usually used to copy a value just before IF, to eliminate the need for an ELSE part to drop it.

DUP n --- n n

Duplicate the value on the stack.

ELSE addr1 n1 --- addr2 n2
(compiling) I,C2

Occurs within a colon-definition in the form: IF ... ELSE ... THEN

At run time, ELSE executes after the true part following IF. ELSE forces execution to skip over the following false part and resumes execution after the THEN. It has no stack effect.

At compile time, ELSE emplaces BRANCH reserving a branch offset, and leaves the address addr2 and n2 for error testing. ELSE also resolves the pending forward branch from IF by calculating the offset from addr1 to HERE and storing at addr1.

ERASE addr n ---

Clear a region of memory to zero from addr over n addresses.

FORTH I

The name of the primary vocabulary. Execution makes FORTH the CONTEXT vocabulary. Until additional user vocabularies are defined, new user definitions become a part of FORTH. FORTH is immediate, so it will execute during the creation of a colon-definition, to select this vocabulary at compile time.

I --- n C

Used within a DO-LOOP to copy the loop index to the stack. See R.

IF f --- (run-time)
--- addr n (compile) I,C2

Occurs in a colon-definition in form:

IF (tp) ... THEN
IF (tp) ... ELSE (fp) ... THEN

At run time, IF selects execution based on a boolean flag. If f is true (non-zero), execution continues through the true part. If f is false (zero), execution skips till just after ELSE to execute the false part. After either part, execution resumes after THEN. ELSE and its false part are optional; if missing, false execution skips to just after THEN.

(LOOP) C2

The run-time procedure compiled by LOOP which increments the loop index and tests for loop completion. SEE LOOP.

LOOP addr n --- (compiling) I,C2

Occurs in a colon-definition in the form: DO ... LOOP

At run time, LOOP selectively controls branching back to the corresponding DO based on the loop index and limit. The loop index is incremented by one and compared to the limit. The branch back to DO occurs until the index equals or exceeds the limit; at that time, the parameters are discarded and execution continues.

M* n1 n2 --- d

A mixed magnitude math operation that leaves the double-number signed product of two signed numbers.

/MOD n1 n2 --- rem quot

Leave the remainder and signed quotient of n1/n2. The remainder has the sign of the dividend.

OFFIRQ . ---

Disables interrupts.

ONIRQ ---

Enables interrupts.

OR n1 n2 --- n3

Leave the bit-wise logical OR of two 16-bit values.

OVER n1 n2 --- n1 n2 n1

Copy the second stack value, placing it as the new top.

QUIT

Clear the return stack, stop compilation, and return control to the operator's terminal. No message is given.

>R n --- C

Remove a number from the computation stack and place as the most accessible on the return stack. Use should be balanced with R> in the same definition.

R> --- n

Remove the top value from the return stack and leave it on the computation stack. See >R.

ROT n1 n2 n3 --- n2 n3 n1

Rotate the top three values on the stack, bringing the third to the top.

RP!

Initialize the return stack pointer from silent user variable RO.

SETIRQ --- I,C

Used only inside a colon definition in the form SETIRQ <NAME> n which will set the nth IRQ vector to point to the routine <name>, <name> must be a CODE word.

SP!

Initialize the stack pointer from silent user variable SO.

SWAP n1 n2 --- n2 n1

Exchange the top two values on the stack.

THEN I,CO

Occurs in a colon definition in form:

```
IF ... THEN
IF ... ELSE ... THEN
```

At run time, THEN serves only as the destination of a forward branch from IF or ELSE. It marks the conclusion of the conditional structure. See also IF.

U* u1 u2 --- ud

Leave the unsigned double-number product of two unsigned numbers.

U/ ud u1 --- u2 u3

Leave the unsigned remainder u2 and unsigned quotient u3 from the unsigned double dividend ud and unsigned divisor u1.

UNTIL f --- (run time)
addr n --- (compile) I,C2

Occurs within a colon definition in the form BEGIN ... UNTIL

At run time, UNTIL controls the conditional branch back to the corresponding BEGIN. If f is false, execution returns to just after BEGIN; if true, execution continues.

VARIABLE n ---

A defining word used in the form n VARIABLE <name> to create a dictionary entry for <name> and assign n bytes for storage in P-FORTH's variable storage area in RAM, which starts at location $BA6F. The application must initialize the stored value. When <name> is later executed, it will place the address of the first byte of the assigned storage on the stack. Note that this differs from versions of FORTH that build their dictionary in RAM.

ASSEMBLER DESCRIPTION AND GLOSSARY

Introduction. P-FORTH supplies a FORTH-type assembler that supports user macros, literal values expressed in any numeric base, expressions using any resident computation capability, and nested control structures without labels and with error control.

This assembler is used to create execution procedures that would be time inefficient if written as colon definitions. Functions may be written first in high level, then tested, and then recoded into assembly with a minimum of restructuring.

Using the Assembler. Invoking the assembler causes CONTEXT to be switched to the ASSEMBLER vocabulary. Each word in the input steam will be matched according to the FORTH practice of searching CONTEXT first, then CURRENT.

The ASSEMBLER words in a CODE definition specify operands, address modes, and op-codes. At the conclusion of a CODE definition, a final error check verifies correct completion by "unsmudging" the definition's name, making it available for dictionary searches.

Run Time, Assembly Time. One must be careful to understand at what time a particular word definition executes. During assembly, each assembler word interpreted executes. Its function at that instant is called 'assembling' or 'assembly time'. This function may involve op-code generation, address calculation, mode selection, etc.

The later execution of the generated code id called 'run-time'. This distinction is particularly important with the conditionals. At assembly-time each such word (i.e., IF, UNTIL, BEGIN, etc.) itself 'runs' to produce machine code, which will later execute at what is labeled 'run-time' when its named code definition is used.

Op Codes. The ASSEMBLER vocabulary includes a dictionary entry for each 6801 op-code (BSR and SWI are not included). These entries end in ",". The significance of this is:

1. The comma shows the conclusion of a logical grouping that would be one line of classical assembly source code.

2. "," compiles into the dictionary; thus a comma implies the point at which code is generated.

3. The "," distinguishes op-codes from possible hex numbers (e.g., ADDA and ADDB).

NEXT. FORTH executes user word definitions under control of the address interpreter, named NEXT. This short code routine moves execution from one definition to the next. At the end of code definition, the user must return control to NEXT or else to code which returns to NEXT. NEXT is a constant that specifies the machine address of FORTH's address interpreter. It is the operand for JMP,. As JMP, executes, it assembles a machine code jump to the address of NEXT from the assembly time stack value.

PUSHBA is the other location to which a JMP may be made. PUSHBA will push the two accumulators on the data stack and continue to NEXT.

Security. Numerous tests are made within the ASSEMBLER for user errors:

1. All parameters used in CODE definitions must be removed.

2. Conditionals must be properly nested and paired.

3. Address modes and operands must be allowable for the op-codes.

These tests are accomplished by checking the stack position (in CSP) at the creation of the definition name and comparing it with the position at END-CODE. The legality of address modes and operands is insured by means of a bit mask associated with each operand.

Remember that if an error occurs during assembly, END-CODE never executes. The result is that the "smudged" condition of the definition name remains, and it will not be found during dictionary searches.

The user should be aware that one error not trapped is referencing a definition in the wrong vocabulary:

i.e., NOT of ASSEMBLER when you want
NOT of FORTH

Explanation of Assembler Glossary

The first line of each entry shows a symbolic description of the action of the procedure on the parameter stack. The symbols indicate the order in which input parameters have been placed in the stack. Three dashes "---" indicate the execution point; any parameters left on the stack are listed. In this notation, the top of the stack is to the right.

The symbols include:

addr	memory address
b	8-bit byte (i.e., hi 8 bits zero)
f	boolean flag. 0=false, non-zero=true
ff	boolean false flag - 0
n	16-bit signed integer number
u	16-bit unsigned integer
tf	boolean true flag - non-zero
cc	condition specifier.

In addition to the entries in this glossary, the ASSEMBLER includes one word for each 6801 mnemonic in the form:

ABX, where ABX is a standard 6801 mnemonic

The , suggests that code is compiled into the dictionary at this point.

Two mnemonics are not implemented:

SWI Since this is used by the system.

BSR Because the offset to a sub-routine is not under the usual control and sub-routines are seldom used.

Assembler Glossary

Specify immediate addressing mode for the next op-code generated.

BEGIN, UNTIL,

(Because of their close relationship, these words are covered in one glossary entry.)

They occur in a CODE definition in the form:

BEGIN, ... cc UNTIL,

At run time, BEGIN, simply marks the beginning of a sequence of code that is executed repeatedly. The corresponding UNTIL, assembles code that branches back to the "BEGIN, point" if the processor condition code register does not satisfy the condition specified by cc. On the other hand, when the condition code register does satisfy cc no branch is taken at the "UNTIL, point" and execution proceeds to the following code.

CODE

A defining word used in the form: CODE <name> ... END-CODE

Creates a "smudged" dictionary header for <name>, sets the CONTEXT vocabulary to ASSEMBLER and executes !CSP. See also END-CODE. CODE is in the FORTH vocabulary.

END-CODE ---

Exit the assembler by making the CONTEXT vocabulary equal to the CURRENT vocabulary. Perform error checking with ?CSP. If successful, "unsmudge" the code word being defined.

MI --- n

A condition specifier: Minus.

NEXT --- addr

A constant. The address of the inner interpreter. CODE routines usually end by a jumping to NEXT (or PUSHBA).

NOT cc --- $\overline{cc}$

"Inverts" the condition code which preceeds it. Thus EQ NOT IF, will execute the code after IF, if the zero flag in the condition code register is not set.

PUSHBA --- addr

A constant. The address of a routine that pushes the contents of accumulators B and A on the stack and jumps to NEXT.

UNTIL,

See BEGIN,

,X ---

Specify indexed addressing mode for the next op-code generated.

VITA

William Bruce Jatko was born in Selma, Alabama, on September 19, 1949. After graduating from Albert G. Parrish High School in 1967, he attended W. R. King State Technical School. He received an Associate Degree in Electronic Servicing in 1969.

He moved to Knoxville, Tennessee, in 1970, where he worked at WTVK Television as a broadcast technician until 1976. He then accepted the position of chief engineer at Cinetel Television Productions in Knoxville. In 1978 he enrolled at the University of Tennessee in Knoxville and began study toward a Bachelor of Science degree in Electrical Engineering, while continuing as chief engineer at Cinetel. He received the BSEE degree in 1982. Since 1982 he has been employed as a development engineer at the Oak Ridge National Laboratory.

He enrolled in graduate school at The University of Tennessee in June 1982 and began work toward the Master of Science degree in Electrical Engineering.

He is a member of Phi Eta Sigma, Eta Kappa Nu, Tau Beta Pi, Phi Kappa Phi, and Golden Key Academic Fraternities. He also belongs to the Society of Motion Picture and Television Engineers.