2

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

CONSTRUCTION AND TESTING OF AN 80C86 BASED COMMUNICATIONS CONTROLLER FOR THE PETITE AMATEUR NAVY SATELLITE (PANSAT)

by

Stephen M. Tobin

December, 1990

Thesis Advisor: Mitchell L. Cotton

Approved for public release; distribution is unlimited.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Distribution Statement A
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) Naval Postgraduate School	5 MONITORING ORGANIZATION REPORT NUMBER(S) Naval Postgraduate School

6a. NAME OF PERFORMING ORGANIZATION	6b OFFICE SYMBOL (If applicable) Code 33	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000		7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS (City, State, and ZIP Code)

10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification) CONSTRUCTION AND TESTING OF AN 80C86 BASED COMMUNICATIONS CONTROLLER FOR THE PETITE AMATEUR NAVY SATELLITE (PANSAT)

12 PERSONAL AUTHOR(S) Tobin, Stephen M.

13a TYPE OF REPORT Master's Thesis	13b TIME COVERED FROM ________ TO ______	14. DATE OF REPORT (Year, Month, Day) December 1990	15 PAGE COUNT 105

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	communications, satellite microprocessors, satellite

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis describes the testing of a prototype of a satellite computer communications controller, based on the 80C86 microprocessor, which is to be placed on the Petite Amateur Navy Satellite (PANSAT) for launch in 1993 on a two year mission. First, the background of the justification for PANSAT is described. PANSAT will serve primarily as an inexpensive store-and-forward orbiting "mailbox", and secondarily it will serve as a teaching and learning vehicle for NPS faculty and students. Then there are reviews of the requirements and design concepts involved in the initial paper design by a previous thesis student. A reliability analysis is done, validating the reliability of the design. Finally, concepts considered in the wire-wrapped prototype construction are explained, followed by an extensive description of initial circuit testing and development of various machine language circuit test programs.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL Mitchell L. Cotton	22b TELEPHONE (Include Area Code) (408) 646-2377	22c OFFICE SYMBOL Code EC/Cc

Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603

UNCLASSIFIED

Construction and Testing of an 80C86 Based Communications Controller for the Petite Amateur Navy Satellite (PANSAT)

by

Stephen M. Tobin
Captain, United States Army
B.S., United States Military Academy, 1980

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING SCIENCE (COMPUTER SYSTEMS)

from the

NAVAL POSTGRADUATE SCHOOL
December 1990

Author: Stephen M Tobin
Stephen M. Tobin

Approved by: Mitchell L. Cotton
Mitchell L. Cotton, Thesis Advisor

Chin-Hwa Lee, Second Reader

Michael A. Morgan
Michael A. Morgan, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

This thesis describes the testing of a prototype of a satellite computer communications controller, based on the 80C86 microprocessor, which is to be placed on the Petite Amateur Navy Satellite (PANSAT) for launch in 1993 on a two year mission. First, the background of the justification for PANSAT is described. PANSAT will serve primarily as an inexpensive store-and-forward orbiting "mailbox", and secondarily it will serve as a teaching and learning vehicle for NPS faculty and students. Then there are reviews of the requirements and design concepts involved in the initial paper design by a previous thesis student. A reliability analysis is done, validating the reliability of the design. Finally, concepts considered in the wire-wrapped prototype construction are explained, followed by an extensive description of initial circuit testing and development of various machine language circuit test programs.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

This thesis builds upon a principal previous thesis [Ref. 1]. That previous author's purpose was to derive an initial design for a small, inexpensive, reliable, on-board, satellite communications controller for the Petite Amateur Navy Satellite (PANSAT). Hence, much of the form and content of this introduction comes from that author's thesis.

A. PURPOSE AND SCOPE OF THESIS

The purpose of this effort was to verify the utility and validity of the cited paper design of the on-board processor for the Petite Amateur Navy Satellite (PANSAT). The establishment of the processor requirements and the processor design were the subjects of that previous thesis [Ref. 1]. The requirement was to design and build a small, inexpensive, and reliable computer controller with enough memory and capability to act as a store-and-forward "mailbox" for amateur packet-radio messages using the AX.25 protocol. In addition, of course, the controller had to be able to withstand the extremes of radiation, heat, and cold in outer space. The design was a challenging task.

This current thesis is a continuation of only a portion of the required steps towards implementation of the actual flight-ready processor based on the initial design. This effort concentrated on the building of a wire-wrapped breadboard prototype and the incremental testing of the hardware using relatively simple machine language programs. These programs were written to incrementally test some of the basic operations which will be required of the controller in its final flight-ready form. Correct operation of all the system's components within the system has been verified.

A 100 MHz dual-trace oscilloscope was the tool used to test for the presence of predicted signals. As the hardware and software of the system incrementally grew in complexity, it became increasingly difficult to verify correct circuit operation with just the oscilloscope, e.g., see the discussion of the third incremental test of the interrupt controller. Was the problem with the oscilloscope not being able to lock on the increasingly longer period, less stable signal, or was the problem with the program's complexity exceeding the circuit's capabilities? A better monitoring system will have to be constructed to carry on from this point. This possible (unlikely) circuit failure could indicate the necessity of some sort of redesign, assuming no flaws in the components or the prototype construction itself. Once a keyboard, monitor, and assembler/compiler are constructed for the system, the next step will be to find out if the system is in fact operating at the edge of its design parameters, and if so, what can be done about it.

B. PANSAT BACKGROUND

The Petite Amateur Navy Satellite (PANSAT) is a small, simple, and inexpensive satellite currently being designed at the Naval Postgraduate School. It is a precursor to the NPS ORION satellite project.

1. PANSAT Concept

PANSAT is intended to be a space-based communications experiment that provides students with hands-on experience in satellite design and operations. It will accomplish three objectives. First, it will serve as an educational tool for NPS officer students, offering them experience in satellite design and operations. Second, it will provide world-wide digital communications using spread spectrum in the amateur band. Third, it will serve as a low-cost, space-based platform for small experiments. [Ref. 2:p. 2]

Additionally, it is an important milestone in achieving for the Space Systems Academic group its ultimate goal of producing the ORION satellite [Ref. 3]. It is a simpler and less capable satellite than ORION. Because PANSAT is less than half of ORION's size and is not attitude stabilized it can be produced for a fraction of the cost of the final version of ORION. Simplicity will help minimize the risks inherent in a first design. Simplicity will also result in reduced cost [Ref. 1:p. 1]. A tentative launch date has been set for 1993 [Ref. 4].

2. Mission and Objectives

The primary mission of PANSAT is to conduct a space-based communications experiment which will provide students with experience in design and operation of such a system. The desired implementation is a store-and-forward message system. In effect, the satellite would be a (delayed) transponder or mailbox. This would allow an authorized user to send a message to the satellite while the satellite was overhead. At a later time, another authorized user could retrieve the message. Outdated or retrieved messages could be (and would eventually have to be) deleted [Ref. 1:pp. 1-2].

In addition, several secondary missions are being considered if volumes and weights permit. These would probably require various analog or digital sensors which would amass various data, e.g., solar cell efficiency or degradation data. This telemetry data could then be periodically collected on-board by the computer and stored as messages. Various programs could be loaded into the satellite processor from the ground, and then tested. This could give NPS students, and others, experience in writing software for satellite control or experiments. One could also write monitor programs to monitor memory errors over time, or to mcnitor power usage by memory and processor components over time. This would allow students to evaluate the effects on memory circuits and/or semiconductor components of exposure to increased radiation and a harsh environment [Ref. 1:p. 2].

3. PANSAT Design

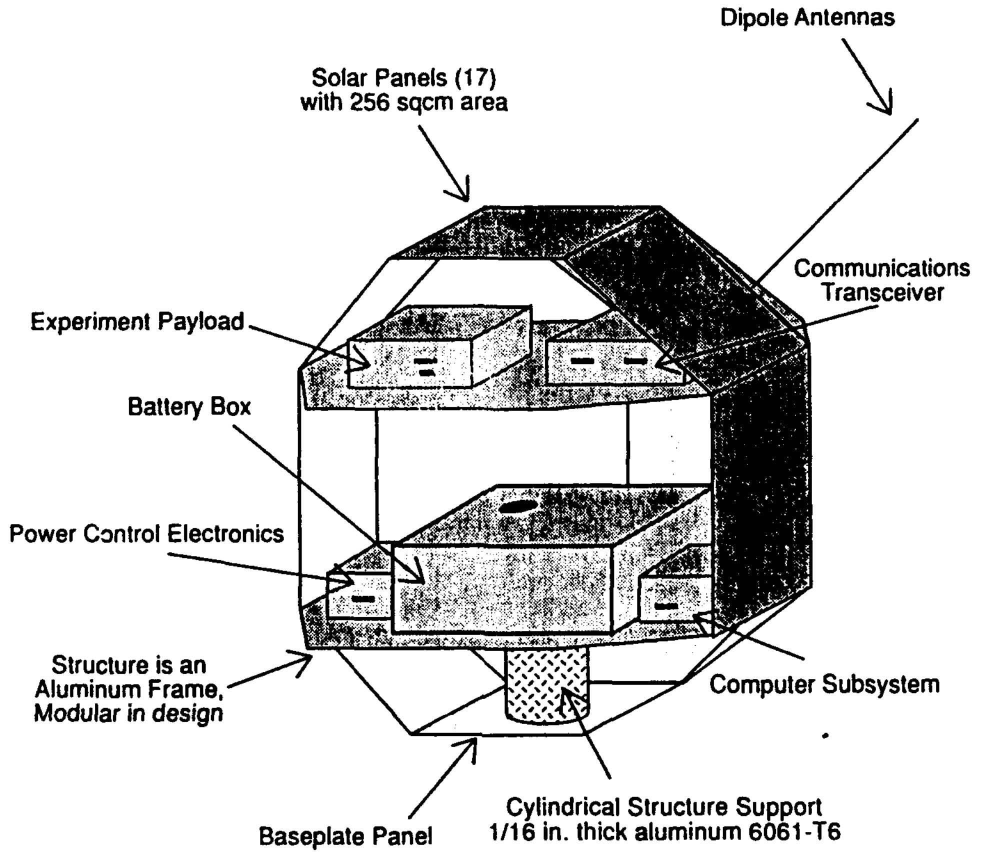
The following are the working design constraints that impact on the processor system design.

a. Orbit

The first PANSAT is planned for launch from the space shuttle in a Get Away Special (GAS) canister or expendable launch vehicle. This constrains the satellite to a typical low earth orbit of approximately 480 km inclined at 28.5 degrees [Ref. 5:p. 2]. The actual orbit will depend on shuttle parameters of the particular mission that launches the PANSAT. Typical orbits have a 90 minute period. The orbit will also determine specific communication opportunities with the satellite. A typical orbit will provide only two or three, greater than ten-minute communications windows per day at the latitude of NPS for any particular ground station [Ref. 6:pp. 7-10].

b. Size

The Get Away Special canister size limits the physical size of the satellite. If a regular size canister is used, this limits satellite size to approximately 19 inches in diameter. Working within these limits, an octagonal cross section design is planned to maximize solar collector area. (See Figure 1.) The satellite volume reserved for the processor and its memory is also shown in



PANSAT diameter is approximately 19"

Figure 1. Pansat Size and Structure [Ref. 5]

the figure.

c. Stabilization

The PANSAT will not be stabilized and will not have any station keeping ability. This means that the processor does not require the capability to monitor any attitude sensors or perform any stabilization calculations. Another advantage to the lack of stabilization in relation to the processor is the potential to leaven the effect of solar heating on the daylight portion of the orbit which constitutes 61.20% of the total equatorial orbit based on strictly line of sight calculations with an earth radius of 6.4×10^6 meters and a satellite elevation of 480 kilometers. (See Figure 2, on the next page.) Refraction due to the atmosphere will undoubtedly extend that percentage somewhat. Lack of stabilization will dictate the use of an omnidirectional antenna(s) which will have a negative effect on the signal-to-noise ratio and on the communications power budget. These problems will be primary concerns of the communications and power supply designers.

d. Communications

The radio communications link with the PANSAT is currently planned to have a 437.25 MHz center frequency with a 960 kHz bandwidth. It will utilize the AX.25 amateur packet-radio communications protocol in half-duplex mode at 1200 bps [Ref. 5:p 3]. The reason for the low 1200 bps data rate is two-fold. First, a low data rate will conserve power on the satellite because less bandwidth is required. Second, it will reduce the probability of bit error thereby possibly making forward error correction (FEC) and its attendant hardware and/or software less important and possibly not required.

((2 x 21.53) + 180)/360 x 100 = 63.29% in sunlight

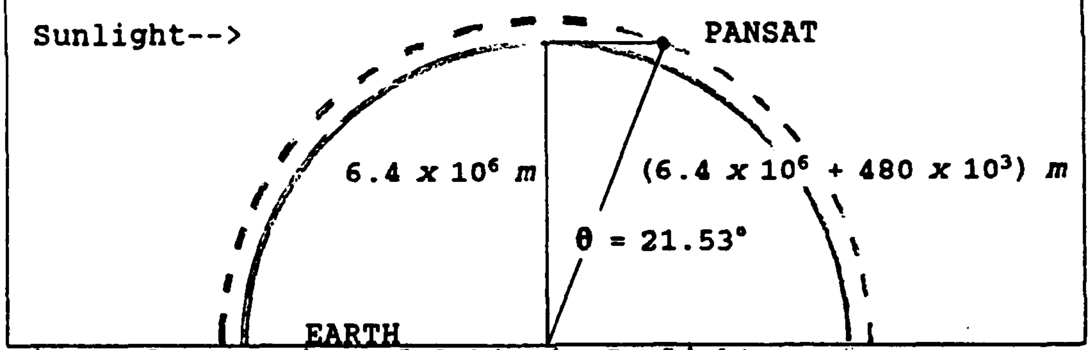


Figure 2. Portion of Orbit in Sunlight

The AX.25 protocol utilizes automatic repeat request (ARQ) at the frame level for error correction. With too high a bit error rate (BER) due to a high data transmission rate this could be too time consuming. But with only a 10 minute or less communications window there is a tradeoff with the desire to keep the data transmission rate low to reduce the BER and the need to get all the data transmitted during the short window. There is room here for further data transmission rate optimization studies which is primarily dependent on the signal-to-noise (S/N) ratio of the PANSAT/ground antenna/transmitter/receiver system and its chosen modulation method. [Ref. 7:p. 28]

The current communications subsystem design forecasts a bit error rate of 10^{-5} [Ref. 6:p. 19]. This is a probability of one character being incorrect every 2.6 pages. (A page is assumed to be 60 x 80 = 4800 bytes = 38400 bits.) Since there will be a maximum amount of available RAM for telemetry/messages of about 800 kilobytes, it will take about 800 kilobytes/1200 sec. = 90 minutes to completely load or unload the RAM. But any message of a page or less (about 4800 bytes) will only take about 4 seconds or less to transmit or receive. Most messages will surely be on the order of a page or less.

e. Power

The satellite will be powered by an array of 15 volt solar cells mounted on the exterior. These will charge two 10.5 volt lead-acid batteries in parallel. The batteries each consist of five 2.1 volt, 5 ampere-hour cells. The batteries are required, of course, for the eclipsed portions of the orbit. A breadboard version of the power system has been designed and constructed. The nominal power requirements of the PANSAT are not yet known because not all the systems have been designed and/or built but the breadboarded power system design estimated that 11.0 watts average power will be required, of which the processor's share is 2.6 watts 100% of the time. [Ref. 8]

f. Durability

The satellite will be subjected to high vibration during launch and orbital injection. The overall root mean squared vibration level is 12.9 g's for 40 seconds [Ref. 9:p. 57]. The processor must be able to withstand these stresses without failure.

g. Lifetime

The processor must be able to function properly during the satellite's design lifetime of one and one half years. The design should be such that system failure can be avoided. If a fault occurs, the design should minimize the impact on the mission by redundancy (appropriate to the relative simplicity of the satellite) or by allowing the processor to work around the fault. [Ref. 1:p. 6]

C. PANSAT COMPUTER FUNCTIONS

The following functions are to be performed by the processor:

1. Communications

The satellite will act as an orbital store-and-forward message service. The satellite will be capable of receiving messages via a communications link, storing the messages, and transmitting them upon request. There should be a way for the processor to transmit a list of its current messages, upon request. Also, messages will have to be periodically erased to free up memory space.

2. Telemetry

The processor may store telemetry data from on-board sensors and the satellite will transmit the data upon request.

3. Housekeeping

The processor must manage housekeeping functions in support of the mission functions. These housekeeping functions could include, but are not limited to:

a. Power management and monitoring of power supply voltage, battery charging current, and battery current draw.

b. Generation and formatting of status messages.

c. Reception, decoding, and execution of commands from the ground control station.

d. On-board fault detection and recovery. (Message fault detection and recovery is a function of the AX.25 protocol.)

e. Ability to update or change programming.

4. Transmitter Control

A major concern for getting the PANSAT design approved for launch is demonstrating positive control over the transmitter. If the satellite has a malfunction, there must still exist means to secure the transmitter from the ground station. Legally, telecommand capability is necessary: "...to turn off a malfunctioning transmitter that might conceivably cause harmful interference to important radio services worldwide." [Ref. 10:p. 12-2]

Transmitter control will be a joint responsibility between the processor and the communications subsystem. During normal operation, the processor will turn the transmitter on and off, but if the processor fails during transmission, the communications subsystem must have a way to secure the transmitter. If the receiver and/or the serial communications controller and/or the processor or software fail with the transmitter keyed on so that the receiver has no way to secure the transmitter, a watchdog countdown timer is provided with the processor to time out and reset the processor, thereby securing the transmitter. Then, if the fault is not correctable, either from the ground or on the satellite, the transmitter is left turned off. Figure 3, on the next page, is the overall computer system diagram. It clearly shows the implementation of a watchdog timer to effect positive transmitter control.

D. DESIGN CONSTRAINTS AND CONSIDERATIONS

Other microprocessors or microcontrollers available in a low power, radiation hardened version could have been used for the initial design, e.g., the 8085, Z80, MC68000, 8096 (microcontroller), and others, but the 80C86 is a good choice

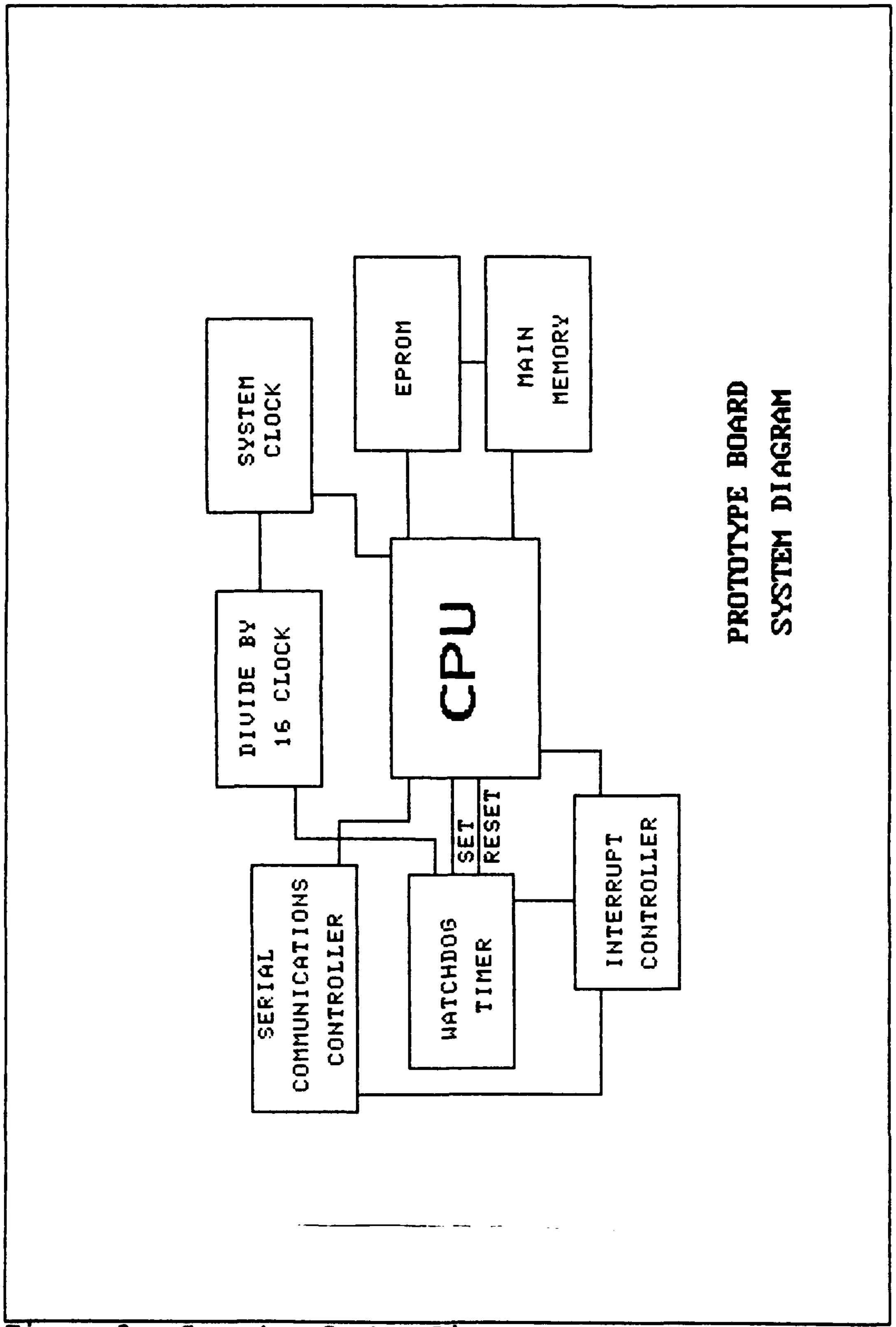


Figure 3. Computer System Diagram

because it is probably better known, with commonly available compilers, assemblers, and software. Most of the following items were considered in the paper design of the processor system in the predecessor's thesis [Ref. 1].

1. Commonality

The processor should be similar to one commonly available. This will simplify program development and allow increased educational benefits. Program development is simplified by the larger number of software packages (i.e., compilers and assemblers) available for a common processor. It is very desirable that the processor have high level language compilers available to allow programming the processor in C or an equivalent language. The educational benefit is enhanced by allowing students to develop a program on a ground-based computer. Once debugged and verified a program can be uploaded and tested on an actual satellite. The 80C86 microprocessor fits this consideration nicely.

2. Upward Compatibility

The PANSAT processor should be one which can be easily expanded to meet the greater demands of ORION. The 80C86 is a less powerful member of the Intel 80x86 family of microprocessors. A more powerful processor, e.g., the 80386, could be substituted for the 8086 with increased speed, memory addressing capability, and multi-processor configuration capability, without the need for much modification of existing software or support tools which will have been developed and debugged for use in the PANSAT program.

3. Reliability

Reliability is one of the most important, if not the most important consideration in the design of a satellite-borne computer system because on-station satellite repair is so expensive. Computer system reliability is a function of design simplicity, operational requirements, the environment and its effect on durability, individual component reliability, and error detection and/or correction capability. Error detection and correction capability is a tradeoff with simplicity of design. The harsh effects of the space environment can be mollified by the use of radiation hardened parts, shielding, thermal control, and (for launch) shock absorbing and vibration dampening mounts. For the design of this simple system one can ensure component reliability by using less than state-of-the-art parts from a reputable company, such parts being well understood and tested with a reliability track record. This again argues for the use of a system based on a common processor like the 80C86 with its attendant memory and commonly used peripheral support chips.

The reference cited below lists some things a designer can incorporate into a spacecraft system to increase its reliability (only that portion of the list applying to the computer subsystem is shown):

...

- Data communications

  ...

  - Digital error detection and correction techniques

  ...

- Command and Control
  - Hardware testing of parity, illegal instruction, memory addresses
  - Sanity checks
  - Memory checksums
  - Task completion timed
  - Watchdog timers
  - Memory write protection
  - Reassemble and reload memory to map around the memory failures [Ref. 11:p. 342]

Some, or all of these could be used in the final design. The watchdog timer is already incorporated in the design.

4. Estimate of System Reliability [Ref. 12:p. 669-674]

To keep the system simple, there is minimal redundant circuitry. Partial protection from software failure is taken care of by the watchdog timer. The only accommodation to redundancy may be in the inclusion of a duplicate bootstrap EPROM. Will this system, as it stands, be reliable enough? As stated earlier, per diem and travel expenses are prohibitively costly for repairmen to do maintenance on a small, inexpensive low-earth orbit satellite such as PANSAT. However, it is possible and very easy to get a ballpark figure for the overall system reliability to assure oneself that the computer system will be reliable enough. The cited reference explains that reliability, as a function of time, *R(t)*, is simply the probability that the system still works correctly at time *t*. The way this function could be derived is to build lots of systems and then plot the number of systems still working against time, *t*. Typically, the resulting graph will be an exponentially decreasing curve. See Figure 4.

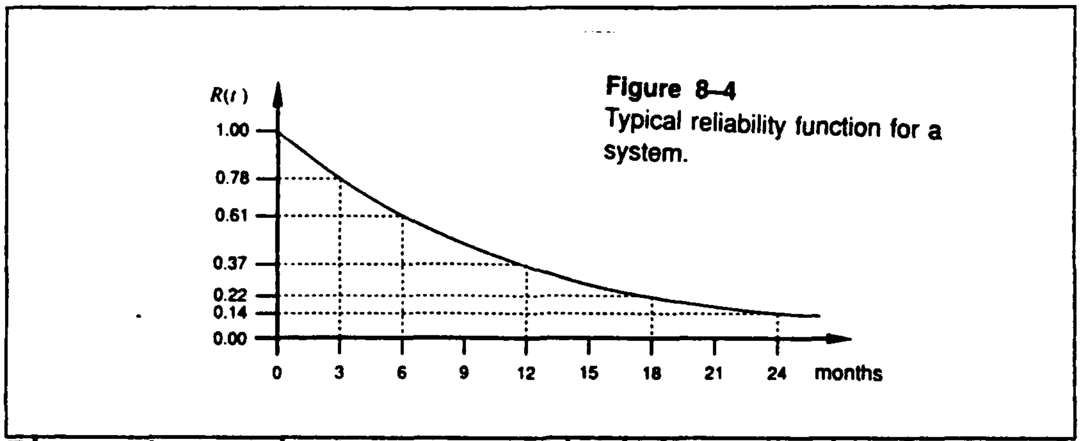


Figure 4. Typical System Reliability Function [Ref. 12]

However, it is impractical and expensive to derive the value for the failure rate by actually building many systems and testing them over a long period of real time. Instead, one takes the reliability information from the specification sheets for the system's individual components. Then one derives the overall system reliability from that information, using a simple mathematical model, to be described below.

Typically, reliability information for electronic components are available from the manufacturer in the form of a failure rate per unit time. This failure rate is usually expressed by the Greek letter λ. This λ is determined by the manufacturer as follows:

$$\lambda = \frac{\textit{number of chip failures}}{\textit{number of chips tested} \cdot \textit{number of hours tested}} \, . \quad (1)$$

Therefore, the reliability of a component can be expressed as

$$R(t) = e^{\lambda t}. \quad (2)$$

Because the failure rates for electronic components are generally very small, these failure rates are often expressed in a scaled unit called a FIT where

$$1 \; FIT = 1 \; failure/10^{9} \; hours. \quad (3)$$

These component failure rates are usually not constant, however. They typically display a bathtub shaped curve when plotted against time. See the figure (Figure 5) on the next page.

The high failure rates during the infant mortality portion of the curve are usually eliminated by the manufacturer's burn-in procedure. If a component is going to fail due to improper or borderline manufacture, the defect should reveal itself early on. The manufacturer only ships those components which have survived this initial testing period. Most components never make it to the wear-out stage. The systems containing them typically become obsolete or

Figure 8–5
The "bathtub curve" for electronic-component failure rates.

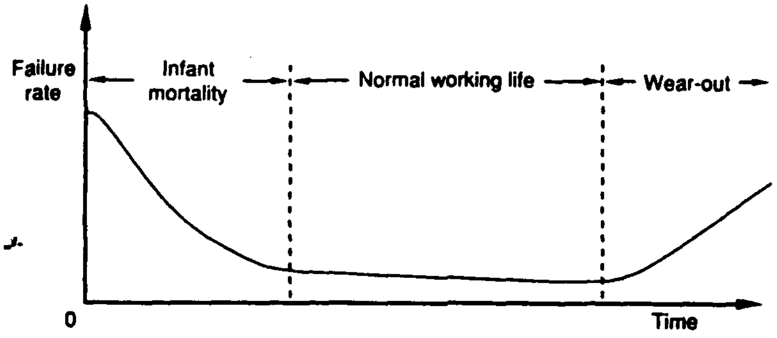


Figure 5. "Bathtub Curve" [Ref. 12]

their physical structure or housing wears out first and the system is discarded. For example, how many people own PCs or VCRs which are more than ten years old? The PANSAT computer system only has to last for up to two years.

There are several factors which can influence the baseline failure rate of components in a real system during its working life. Some of them include temperature, humidity, shock, vibration, radiation, and power cycling. For terrestrial components, temperature is usually the most significant of these factors. The failure rate approximately doubles for every 10°C rise in temperature. This is good news for the PANSAT computer, since it will be operating in the cold environment of space. However, this advantage must be balanced with the lack of convection cooling to get rid of internally generated heat due to the lack of air.

With known component failure rates from the manufacturer, one can calculate the overall system rate by simply adding the individual component failure rates. This assumes all components are necessary for the system to work, the failure rates are independent of each other, and the failure rate of the component interconnections is minimal.

Specific failure rates are not readily available for specific components. However, failure rates of certain batches or runs of a particular manufacturing process are available for some of the radiation hardened parts [Ref. 13:pp. 14-16--14-20]. For other non-radiation hardened parts, such as the serial communications controller (SCC) or the SRAMs, one can make a rough estimate using Wakerly's text [Ref. 12:p. 674]. The EPROMs, SCC, and SRAMs will be treated as LSI parts. A FIT of 250 will be used for them. For the remaining chips, a FIT of 50 should be conservative enough. Finally, a FIT of 15 will be used for the 50 or so resistors, capacitors, and the crystal. One can calculate the tentative failure rate for the prototype PANSAT computer system, λ_{sys}, as follows:

$$\lambda_{sys} = (9 \ x \ 250) + (15 \ x \ 50) + (50 \ x \ 15) =$$

$$3750 \ failures/10^9 \ hours \ . \quad (4)$$

The mean time between failures is $1/\lambda_{sys}$, or about 30 years.

5. Radiation Hardening

Another advantage of using an 80C86 based system, alluded to above, is that almost all the required parts are available in a radiation hardened version. This greatly increases the system reliability (as well as the expense). The low earth orbit that PANSAT will occupy does not strictly require the use of radiation hardened components [Ref. 14:pp. 34,80]. However, reliability vis-à-vis solar and extrasolar radiation (and manmade nuclear EMP) is enhanced by the use of radiation hardened parts. If money is no object they can be used with this design.

6. Environment

Consideration of the space environment impacted the design primarily in the selection of components which were available in a radiation hardened version.

a. Launch

Launching parameters had no direct influence on the design.

b. Orbit

The orbit has not been determined. But the use of radiation hardened parts will almost certainly gain a reliability advantage regardless of the final orbital parameters. This again argued for the use of the 80C86 or any other processor available in a radiation hardened version. The orbit also has a direct effect on the frequency and limits of temperature extremes. Because PANSAT is small and inexpensive, the only thermal stabilization will come from the satellite structure design and subsystems placement, and possibly reflective or absorptive coatings. The processor and support chips are available which will operate reliably in the range of approximately -55°*C* to +125°*C*.

7. Power

Available power on a satellite is limited. Static power consumption was a consideration in choosing the 8086, because it is available in a CMOS version, the 80C86. CMOS consumes much less quiescent power, although dynamic power consumption is a linear function of operating frequency.

8. Communications Protocol

The tentative communications protocol to be used is the AX.25 amateur packet-radio link-layer protocol. This protocol follows, in principle, the CCITT X.25 protocol, with the exception of an extended address field and the addition of an Unnumbered Information (UI) frame. It was designed to be a mechanism for the reliable transport of data between two signaling terminals, in this case between a satellite and a ground station. [Ref. 15:p. 1]

a. Error Detection

Both the sender and receiver calculate a frame-check sequence to make sure a frame was not corrupted by the medium. In addition, there are several link-layer errors which are recoverable without terminating the connection due to things like the I frame information field being too long, or the reception of a frame with an invalid Send Sequence Number (N(S)).

b. Error Correction

The protocol handles communications errors by retransmission. Errors due to transients in the medium or the hardware can be fixed this way. A permanent hardware fault in a ground station, once detected by the protocol, can be fixed. There are limited options for error correction if a hardware fault occurs on the satellite. Hardware redundancy with automatic switching upon error detection, or switching commanded from the ground are options, but to keep the design simple hardware redundancy and error correction have been forgone. The emphasis has been placed on hardware reliability, negating the requirement for (hardware) error correction.

E. PREVIOUS FLIGHT DESIGNS

At least three other satellites have been launched with successful missions similar enough to that of PANSAT to provide useful information and insights. Taken together, they show that amateur packet radio store-and-forward message service with the AX.25 communications protocol is feasible. It can be done because it has been done before.

1. UoSAT-2

The UoSAT-2 was designed to investigate the technical feasibility of store-and-forward communications via inexpensive satellites. It was launched in 1985 into an 800 kilometer polar orbit and it was designed to operate for more than three years. It utilized a NSC-800 processor running at 0.9 MHz. The communications link operated at 1200 bps, using a custom message protocol (MSG2). The kernel implementing the protocol was written in Z-80 assembler code and occupied 2.5 kilobytes of error detection and correction protected (EDAC) 12 bit wide RAM. This satellite was the first civilian demonstration of an operational store-and-forward communication system. Its on-board EDAC hardware provided invaluable data on the frequency of hard and soft radiation induced errors. It found that about 6 single-event upsets (SEUs) per day would occur in a 1 megabyte commercial grade CMOS memory in that orbit. Of course, the SEU rate depends on: [Ref. 16]

- Memory device manufacturing technique (radiation hardening)
- Device geometry
- Shielding
- Satellite orbit
- Satellite attitude
- Solar activity
- Geomagnetic activity

2. UoSAT-D

The UoSAT-D was a packet communications experiment that built on UoSAT-2. It implemented the AX.25 packet radio protocol operating at 9600 bps in full duplex mode. It utilized an 80C186 processor running at 8 MHz. It mission was to support amateur packet-radio communications using low earth orbit satellites and to study the orbital radiation environment and its effects on semiconductors. [Ref. 17]

3. FO-12

The FO-12 was the first Japanese amateur satellite, launched in August of 1986,to serve as a store-and-forward digital mailbox. It implemented the AX.25 protocol with four uplink channels and one downlink channel, all operating at 1200 bps. No ROM was used. Only DRAM was used. Initial program load was accomplished via hard-wired logic. The FO-12 also used a NSC800 processor. [Ref. 18]

II. ORIGINAL DESIGN CHOICES/PROTOTYPE CHANGES

Changes have been made to the original design primarily for three reasons--expense, advance in the state of the art, and expediency. For the prototype, military standard or commercial grade were substituted because they are functionally equivalent and much less expensive. The wire-wrapped breadboard and flight model components may also have to be changed in response to changing mission requirements.

A. PROCESSOR SELECTED

The prototype uses an 80C86 microprocessor, commercial grade, instead of the 80C86RH radiation hardened version because they are functionally equivalent and the 80C86 is much less expensive. This expense was the main reason for chip substitution for all the other chips, with the exception of the Zilog serial communications controller (SCC). See below.

B. PROCESSOR MODE

The prototype stayed with minimum mode instead of maximum mode because this is still a small stand alone system.

C. BUS DEMULTIPLEXING

The wire-wrapped prototype still used the 54HC573 octal D-type address latches and the 54HC245 octal tri-state data bus transceivers because there was no need to change them. They are available in radiation hardened versions and are fast enough to operate at 5 MHz.

D. CLOCK AND PERIPHERALS

1. Analog to Digital Converter

The A/D converter is not implemented, but it would be straightforward to implement once the telemetry/sensor requirements are determined.

2. Parallel Port

For the same reason, the parallel port is not implemented.

3. HDLC

The CMOS 85C30 was substituted for the "TTL only" 8273 HDLC because the anticipated orbit was changed to a lower one and radiation hardening was determined to be not as critical. The 8273 was not available in a CMOS version at the time of the original design. The TTL 8273's radiation immunity was traded for the reduction in power consumption by using the CMOS 85C30 serial communications controller.

4. Timer

The 82C54 programmable interval timer is still used. Its primary function is to serve as the transmitter control watchdog timer. For some of the testing, a functionally equivalent subset, the 82C53, was substituted.

5. Interrupt Controller

The 82C59 programmable interrupt controller is still used.

6. I/O Device Chip Selection

The 54HC138 3-line to 8-line decoder/multiplexer is still used. However, the address selectors were changed to A7 through A10, instead of A3 through A5, for added flexibility [Ref. 19].

7. Clock

The 82C85 static clock controller/generator and the 74C161 synchronous binary counter are still used.

8. I/O Device Timing

There has been no significant change.

9. Reset

The 82C54 still acts as the watchdog countdown timer. If the processor is operating properly, the watchdog timer will be reset before it has a chance to time out and reset the processor. However, in the original design, if the timer timed out, a flip-flop would switch to an identical ROM and the system would reinitialize. This would provide for the correction of a hard fault in the first ROM. In the interest of expediency, the actual flip-flop and extra ROM was not built. It would be easy enough to include it in the flight model, however.

10. Miscellaneous changes

A reset switch has been added to the prototype for an on-board reset capability and an LM7805C 5 volt voltage regulator chip have been added to the board between the external power and the rest of the board. Also, an MC1488 line driver chip and an MC1489A line receiver chip have been added to the board for conversion of the RS-232C voltages of +4 volts and -3 volts to the CMOS logic voltages of 0 and +5 volts, respectively. The RS-232C connection should be used to download program code or test messages during the next phase of the board's testing.

E. MEMORY SYSTEM

1. (E)PROM

The original design used two 2k x 8 bit PROMs in tandem. The breadboard prototype uses two 128k x 8 bit TMS27C010 PROMs in tandem. This is certainly more than is required for the on-board operating system, or at least the boot-strap kernel, but it made the construction symmetrical and gives plenty of room for experimentation.

2. Vital RAM

The wire-wrapped prototype doesn't have a vital memory section. The actual size of protected memory required can not be determined without a firmer idea of how much telemetry or other vital data will be required. One could go back to using the current 128k x 8 PROM memory space and splitting it into a much smaller actual PROM portion with the balance of the 128 k x 8 space being taken up with radiation hardened/shielded redundant SRAM. Perhaps this address space could be occupied by special 12 bit EDAC memory as in the UoSAT-2.

3. Bulk RAM

The original design used 32k x 8 bit SRAM chips with extra 54HC138 multiplexers and 54HC32 OR gates for addressing. Due to rapid advances in the state-of-the-art, 128k x 8 bit memory chips were available for the wire-wrapped prototype, dividing the RAM memory chip count by four and eliminating the need for five 138 decoders and eight 32 hex OR gate chips.

F. LOGIC DEVICE FAMILY SELECTION

1. CMOS v. TTL

The original design was done with CMOS instead of TTL because, as can be seen in Figure 6 on the next page [Ref. 20:p. 569], all the CMOS logic families use less power than their TTL counterparts and most CMOS families are fast enough to operate at 5 MHz (With the exception of the C family.)

In addition to reliability, low power consumption is a primary goal in the design of satellite circuits. The following three differences dictated the use of CMOS versus TTL components in the design. Factors which determined the particular family of CMOS selected are also discussed.

a. Speed

As stated, for the breadboard prototype, CMOS was

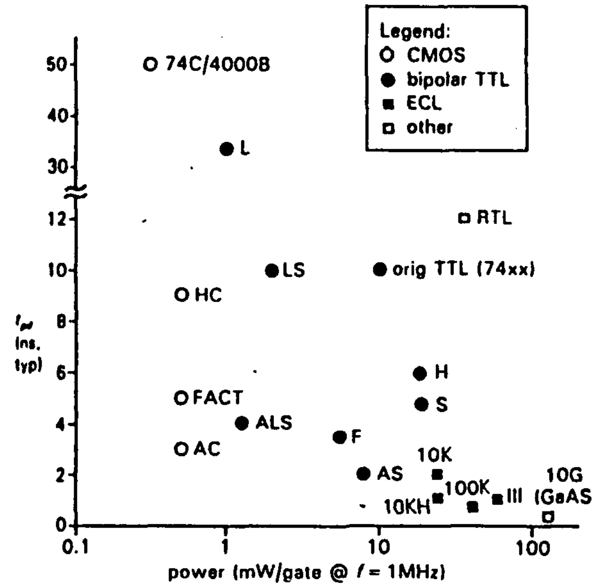


Figure 9.2. Speed versus power for various logic families.

Figure 6. CMOS v. TTL Relative Power Consumption & Speed [Ref. 20]

used because at 5 MHz CMOS dissipates less power than TTL although many TTL families are as fast or faster. But this is not a high speed design. The CMOS HC family was specifically selected for the design, instead of the C family because "...The speed range of CMOS goes from about 2 MHz (4000B/74C at 5V) to about 100MHz (for AC/ACT)." [Ref. 20:p. 486] Because the processor is only being driven at 5 MHz, or less, speed considerations argue for the use of CMOS HC family parts or faster. The 82CXX family of parts, used because of the ease of conversion to a nearly totally radiation hardened circuit, all operate to at least 5 MHz or better.

b. Power

One disadvantage of CMOS is that, as the operating frequency increases, so does the power dissipation. Even so, the total power dissipated at 5 MHz will still be well below that of TTL. [Ref. 20:p. 487] Figure 7 shows this clearly.

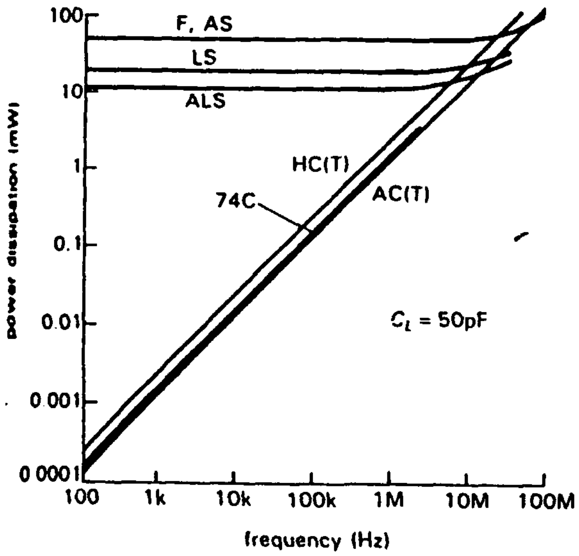


Figure 8.18. Gate power dissipation versus frequency.

Figure 7. CMOS Power Dissipation v. Frequency [Ref. 20]

c. Noise immunity

CMOS has slightly better noise immunity than TTL, for most circuit configurations. HC and AC family CMOS have the best noise immunity of all the CMOS families. [Ref. 20:p. 569]

2. Compatibility

Any CMOS family, operating at 5 volts, is compatible with any other CMOS family [Ref. 20:p. 569]. It is probably best to have all the parts of a circuit be of the same family, which is the HC family in this case.

3. MIL-SPECS

Finally, for the parts other than 82CXX, the 54HCXX versions should be used for the final flight version of the circuit. The 54 prefix means that the component meets more rigorous temperature and reliability standards than the commercial grade series, 74. For example, a 54 series component will be able to operate in a temperature range of -55°*C* to +125°*C*, and can withstand + or - 10% power supply variations whereas a 74 series component can only operate from 0°*C* to 70°*C* and can only withstand + or - 5% power supply variations [Ref. 21:pp. 244,247]. That extra temperature margin, especially, could be important in the harsh environment of space. (Some schematic components are designated as 74 series.)

III. WIRE-WRAP CONSTRUCTION AND TESTING

A. BOARD LAYOUT

The layout of the board with its components is shown in Figure 8 on the next page. One can see that the chips are as close together as possible, consistent with having enough room to install filter caps and/or current limiting or pull up resistors if or when needed. An attempt was also made to arrange the components so as to minimize wire lengths while taking into account the required circuit interconnections.

1. Wire Size(s)

The wire size used was AWG 30. One of the references, however [Ref. 21:p. 119], recommended that AWG 20 be used for all IC power and ground lines. Unfortunately, the wire wrap tool is designed for AWG 30. One solution would be to make double runs of the AWG 30 wire to simulate AWG 20. However, all the chips drew so little current that voltage drop was not a problem. The wall power supply current meter never read more than about 80 milliamps.

2. Heat Dissipation

With CMOS, heat dissipation is not really much of a concern. On the prototype board the chips have stayed cool to the touch, even during extended periods of testing.

3. Component Placement to Minimize Wire Length

The components were not just arbitrarily placed on

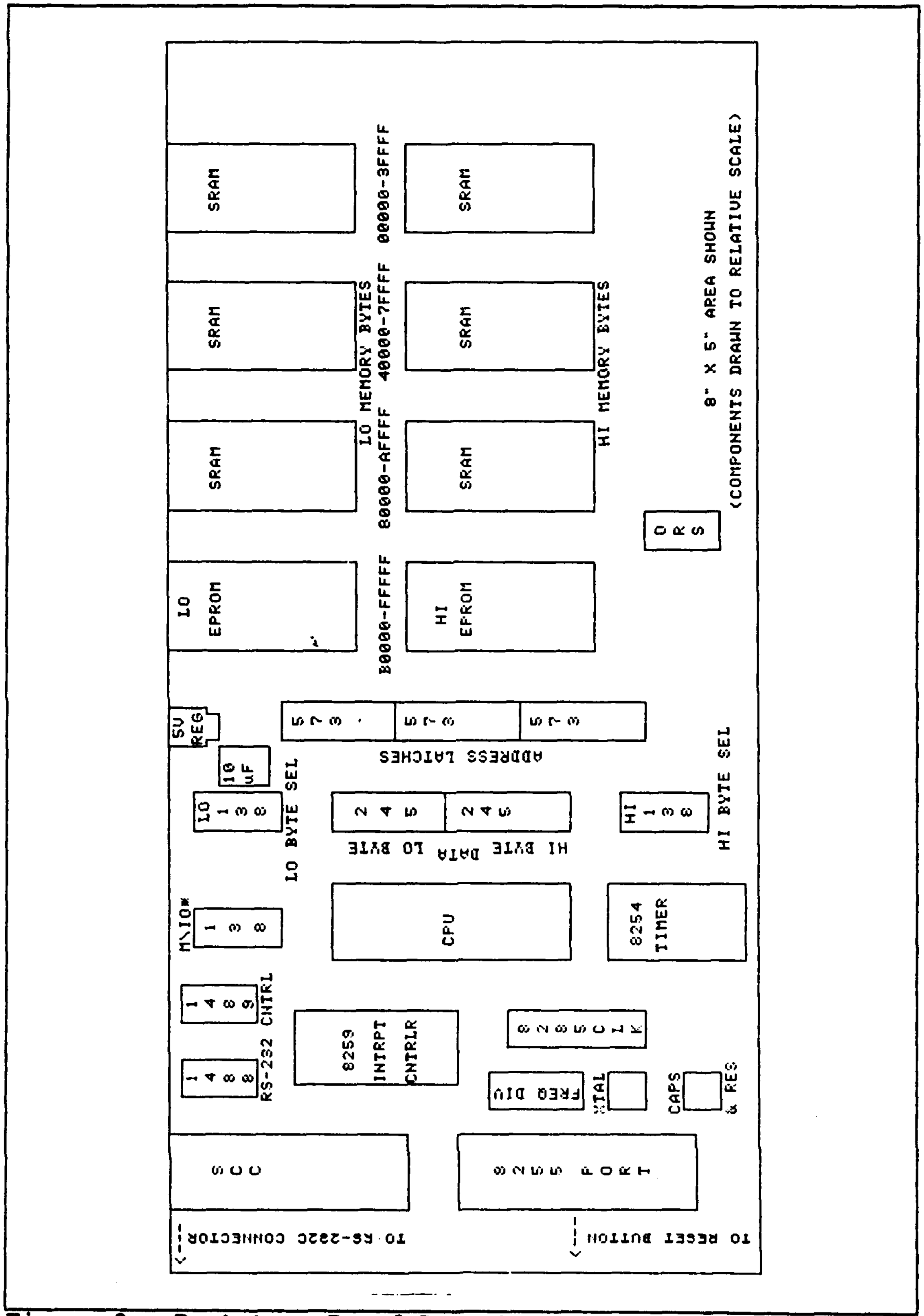


Figure 8. Prototype Board Layout

the wire-wrapped board. The components were placed so as to minimize wire lengths. Some of the reasons for minimizing wire lengths are described below.

a. Stray capacitance

Stray capacitance's effect on a digital circuit is to act as an AC load. An AC load is undesirable because it can slow down transition times. There are at least three things which can cause stray capacitance in a digital circuit:

- Output circuits--typically several pF
- Input circuits--typically from 2 to 15 pF
- The wiring that connects outputs to other inputs--typically 1 pF per inch, or more. [Ref. 12:p. 74]

The first and second items are inherent to the I.C. packages themselves. The only factor which the wire-wrap circuit builder has control over is the length of the wiring. Fortunately, the circuit is only being driven at 5 MHz, but even so, a conscious effort was made to minimize wire length. Rise-times on any of the tested signals were not observed to be a problem. There were, however, significant voltage spikes on many rising and falling signal edges.

b. Signal Degradation and Power Drop

Standard wire-wrap size wire, 30 AWG, was used. This seemed to be at odds with the reference [Ref. 21:p. 119] which recommended at least 20 AWG or larger for all logic IC power and return lines. (Although this reference is presumably more concerned with higher speed circuits.) Wires with a smaller cross-section are more prone to noise pickup and power drop per unit length. However, because CMOS was used with a low current draw and the computer could fit all on one board in a 5" by 8" area, the 30 AWG wire seemed to work. Double wire runs were started on some of the power and ground lines, to increase the effective AWG, but the effort was abandoned when initial testing seemed to indicate that noise and voltage drop were not a problem.

c. Propagation Delay

Another reason one should try to keep the wires as short as possible is to minimize the effects of propagation delay. Too much propagation delay could have a deleterious effect on the circuit. Such things as clock skew or missed "gates" or "windows" could cause circuit malfunction if one or more lines are particularly long compared to others. [Ref. 22:p. 119]

4. Filter Capacitors

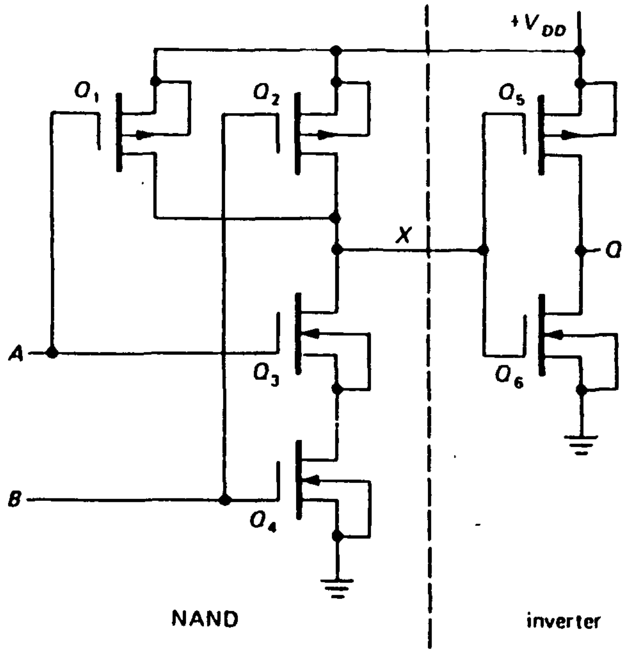
In accordance with the *Logic Designer's Manual* [Ref. 22:pp. 117-118], a 10 microfarad capacitor was placed in parallel with a 0.01 microfarad capacitor across the incoming 5 volts from the wall power supply. A filter capacitor is used to help reduce noise. This is recommended whenever power supply lines run an "appreciable" distance to a board. However, an LM7805 5 Volt voltage regulator was later added to the board negating the requirement. Strictly speaking, these capacitors are only recommended for circuits operating above 10 MHz anyway. However, with the fast switching times of the HC chips, harmonics may be generated which may be partially shunted by these two capacitors. It shouldn't hurt to leave them on the board. Additionally, a 0.1 microfarad bypass capacitor was placed across the power and ground connection of each chip. The reference recommends a value of 1 to 10 microfarads, but the experienced Space Systems electronics technician [Ref. 23] recommended the 0.1 microfarad capacitors be used instead.

B. CMOS POTENTIAL PROBLEMS/CONSIDERATIONS

1. Unused Inputs

One slight disadvantage of CMOS versus TTL is the requirement to tie all unused CMOS inputs high or low, as appropriate, even for unused gate sections in the same package. With TTL, one can ignore unused sections of a single chip, as well as unused input lines within a particular circuit on a chip. This is not true with CMOS. If one leaves any input line unconnected, the input level could float up to the logic threshold. This could result in both MOS output transistors conducting, resulting in excessive current draw. [Ref. 20:p. 580]

Refer to Figure 9 [Ref. 20:p. 485] on the next page, which is a standard CMOS AND gate. Suppose, for example, inputs A and B were unused but not tied off. One can see



B

Figure 8.17.

Figure 9. CMOS AND Gate [Ref. 20]

that if both of these inputs floated high enough to bias Q3 and Q4 on, but not high enough to bias Q1 and Q2 off, the resulting current flow through the low resistance path from +Vdd to ground could be excessive. This could cause latch-up (described below) resulting in temporary or permanent damage to the part.

2. Latch-up

"All CMOS devices are instrinsically susceptible to latch-up, which occurs when internal parasitic silicon-controlled rectifiers (SCRs), structures that are inherent in CMOS integrated circuits, are triggered on." [Ref. 21:p.57] The three small figures directly below (Figure 10) help to

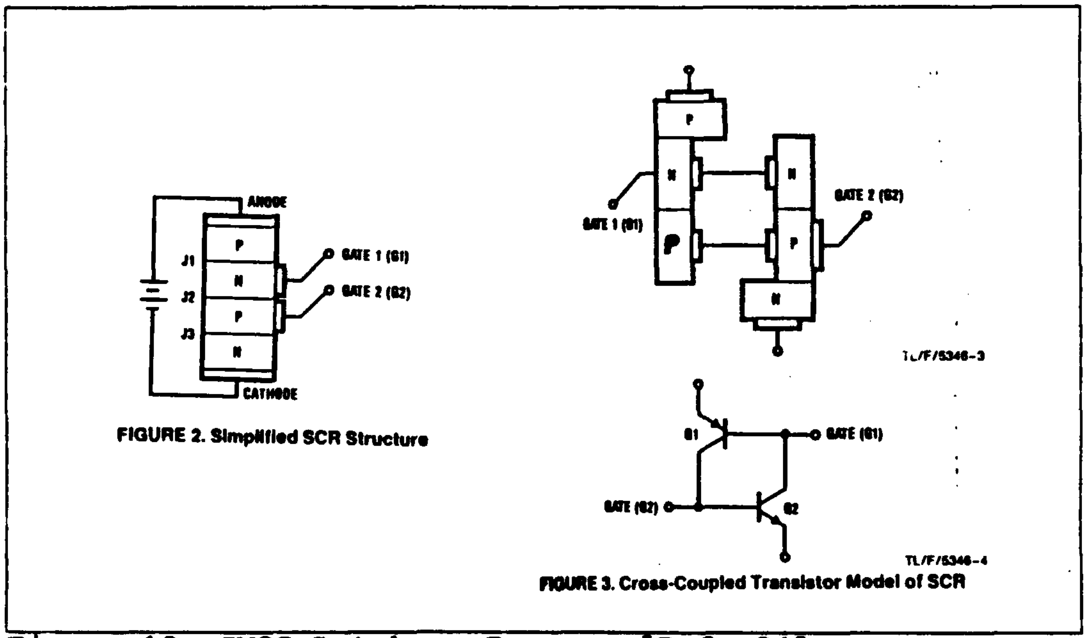


FIGURE 2. Simplified SCR Structure

FIGURE 3. Cross-Coupled Transistor Model of SCR

Figure 10. CMOS Latch-up Process [Ref. 24]

explain the inherent problem of CMOS latch-up.

The small figures show the simplified SCR structure inherent in all CMOS devices, the effective cross-coupled transistor operation of the junctions during latch-up, and a simple circuit model of the two "transistors" involved. Normally, gate 1 and gate 2 will be at the same potential. But suppose a large positive voltage is placed across the anode and the cathode of the SCR structure on Q1. The process of latch-up can be described as follows: If enough leakage current can flow from Q1's anode to its cathode, thereby biasing Q2 on, then more current will be on gate 1, biasing Q1 on even more. This is a runaway situation which rapidly leads to saturation. [Ref. 24:pp. 2-112--2-113]

However, latch-up should not be a problem anymore with the current production methods used to manufacture the

54HCxxxx parts [Ref. 24:p. 2-112]. The latch-up problem has been eliminated with the Harris radiation hardened parts as well [Ref. 25:p. 2-11]. These are the most likely components to be used in the final design.

3. Pull up Resistors

Pull ups (or pull downs) are only required if one is driving a CMOS input with a device that has an open state [Ref. 20:p. 579]. Pull downs can be used for CMOS but they are not effective for TTL. For CMOS, noise immunity is the same with either pull ups or pull downs and there is no current draw through the resistor for either choice when the driving device is open.

There is no need for pull ups in this design. The only three state devices used in this design are the 573 octal latches and the 245 octal transceivers. The 573s are permanently enabled and any device driven by the 245s has a *chip select* signal. For example, in between *data enable* assertions, the outputs of the 245s are in a high impedance state. This is not a problem because none of the devices driven by this input, e.g., the interrupt controller or the SRAMs, will be chip selected at this time. Therefore, one will not load the devices with unpredictable data and no pull ups are required.

4. Fan out

Because all the components are CMOS (with the exception of the RS-232 drivers, to be used for testing, which are NMOS), fan out should not be a problem with the design, with the possible exception of the *read* signal. The *read* signal is the worst case. Any of the outputs should be able to easily drive at least 10 other inputs [Ref. 20:p. 487]. The *read* signal goes to the eight memory chips, the timer, the interrupt controller, and the SCC, for a total fan out of 11. This should still be acceptable.

5. Fast Rise-times and Harmonics

These are a concern with the newer CMOS logic families [Ref. 21:p. 5] such as HC used in this design. The fast rise-times have two major deleterious effects: Faster rise-times generate higher harmonic frequencies, and faster rise-times result in greater transient voltage swings.

The bandwidth of the interconnection network on a wire-wrap board will limit the amplitude of the higher frequency components and the inductance will result in transient voltage spikes. This was observed on the signal traces in the lab and already mentioned above. The data bus signals, especially, which are interconnected with many more wires than most other signals, clearly showed variations in voltage levels. Also, voltage spikes of about 0.75 volts (or more) were observed on the system clock, for example. These problems were minimized by paying attention to the component layout, where the shorter all the interconnecting wires were, the better, as recommended in the reference [Ref. 21:p. 84].

C. INCREMENTAL TESTING

1. Clock and Clock Frequency Divider

First all unused inputs to the 82C85 and the 54HC161 were checked to make sure that they were tied either *high* or *low*. Then they were plugged into their respective chip holders and power was applied to the board. The 82C85 was putting out a 5MHz, zero to five volts clock on pin 2. The 161 was putting out a 0.3125 MHz clock (5MHz divided by 16) on pin 11 . The clocks had noise of approximately 0.75 volts peak-to-peak centered around either the zero or five volt reference level. See Figure 11 on the next page.

2. CPU, EPROM, Latches, and Data Transceivers

After verifying that the 82C85 was putting out an acceptable clock, the next step was to verify that the CPU and EPROM, with their integral latches and data transceivers, were at least nominally functioning. This was done by hand coding the simplest program possible (five bytes) and burning it into the EPROMs. (Appendix A has the 8086 machine language code and the program code is in Appendix B.) The EPROMs come from the manufacturer initially "programmed" with 128K bytes of "1"s, or "FF"s. (Programming or burning the EPROM changes appropriate bits to "0"s.) The program is simply one statement--an unconditional jump to itself.

A 100 MHz oscilloscope was used to verify that the EPROM *chip select* signal (*CS**) was being asserted at the same time as the CPU *read* signal (*RD**) on pins 22 and 24,

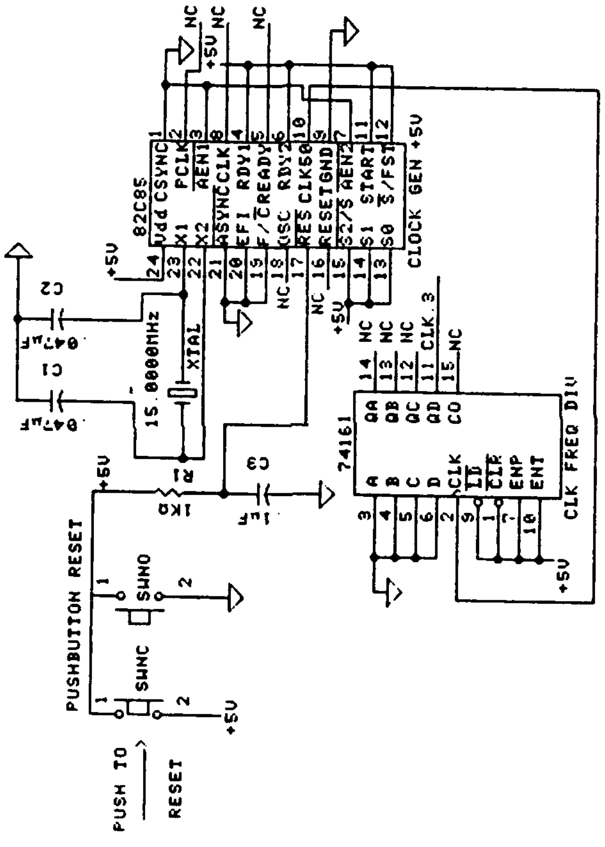


Figure 11. Clock Circuit Schematic

respectively, of the EPROM. Also, the address and data lines on the 573 address latches and the 245 data transceivers were checked to verify that all but A3 through A1 stayed high and that the 16 data bits changed appropriately to reflect the data transfer of "00EA", "FF00", and "(FF)FF". The relevant parts of the board for this CPU loop test are shown on the next two pages in Figures 12 and 13.

3. Memory

Next, to do an initial verification test of SRAM, a 25 byte hand coded machine language program was burned into the EPROMs for testing. (See Appendix C.) The program would jump to the start of a routine. The routine would set up the data segment register for a segment located in memory. Then a memory location (byte) would be loaded with the arbitrary number "05". Then there is a NOP (no operation) in the code. Then the same memory location is read and the value there (should be "05") is loaded into the DL register. Finally, the routine jumps to the initial jump command and the program repeats.

As above, an oscilloscope was used to verify that the address and data lines were what was expected, as well as verifying that the 2 *chip select* signal assertions and the *read* and *write* signal assertions happened at the right times with respect to each other. They did.

Figure 14 follows Figures 12 and 13. It shows the complete memory schematic used for this and subsequent tests.

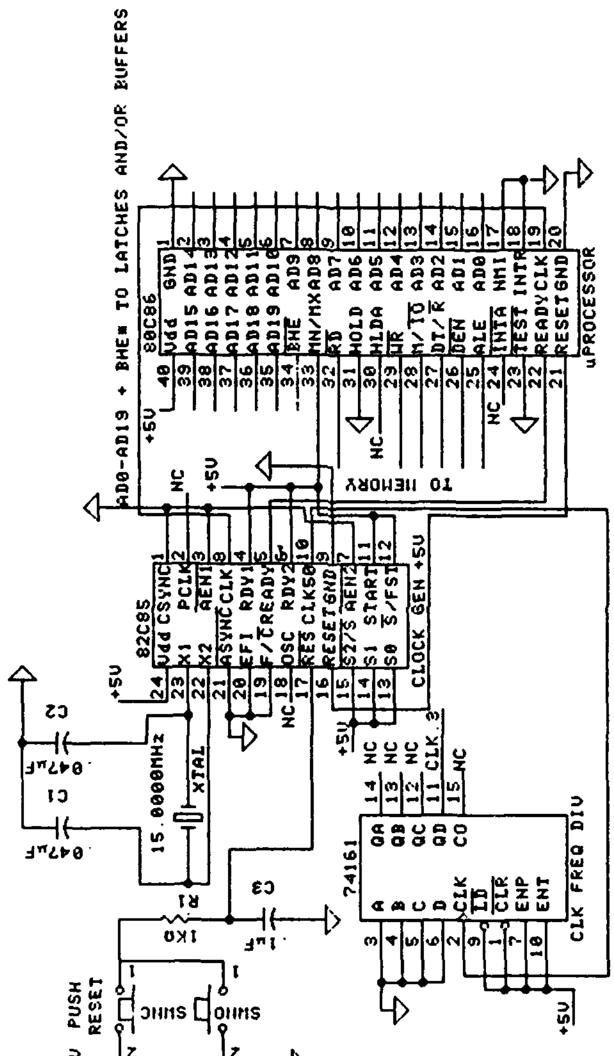


Figure 12. Loop Test Schematic--Clock and CPU

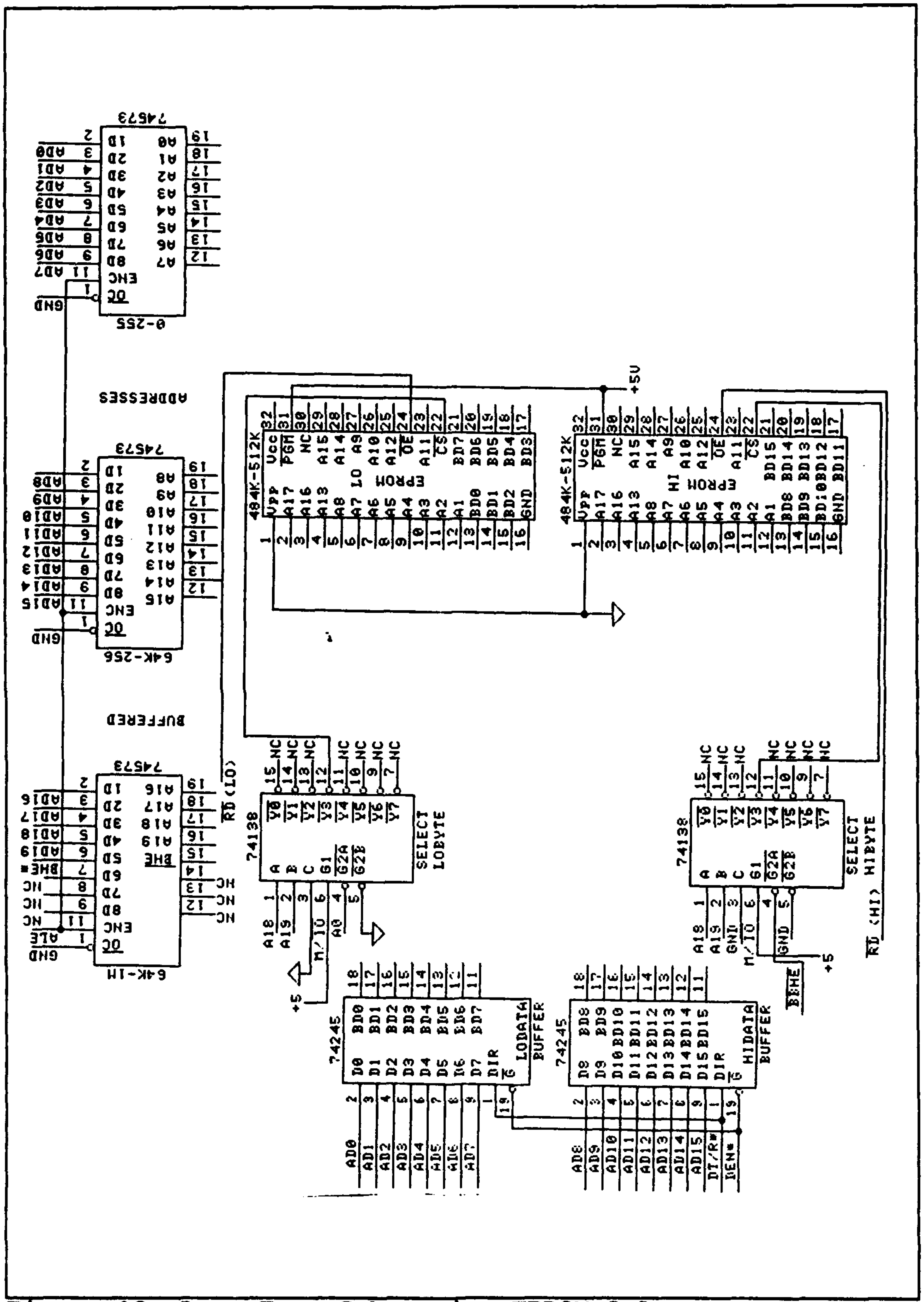


Figure 13. Loop Test Schematic--EPROM Only

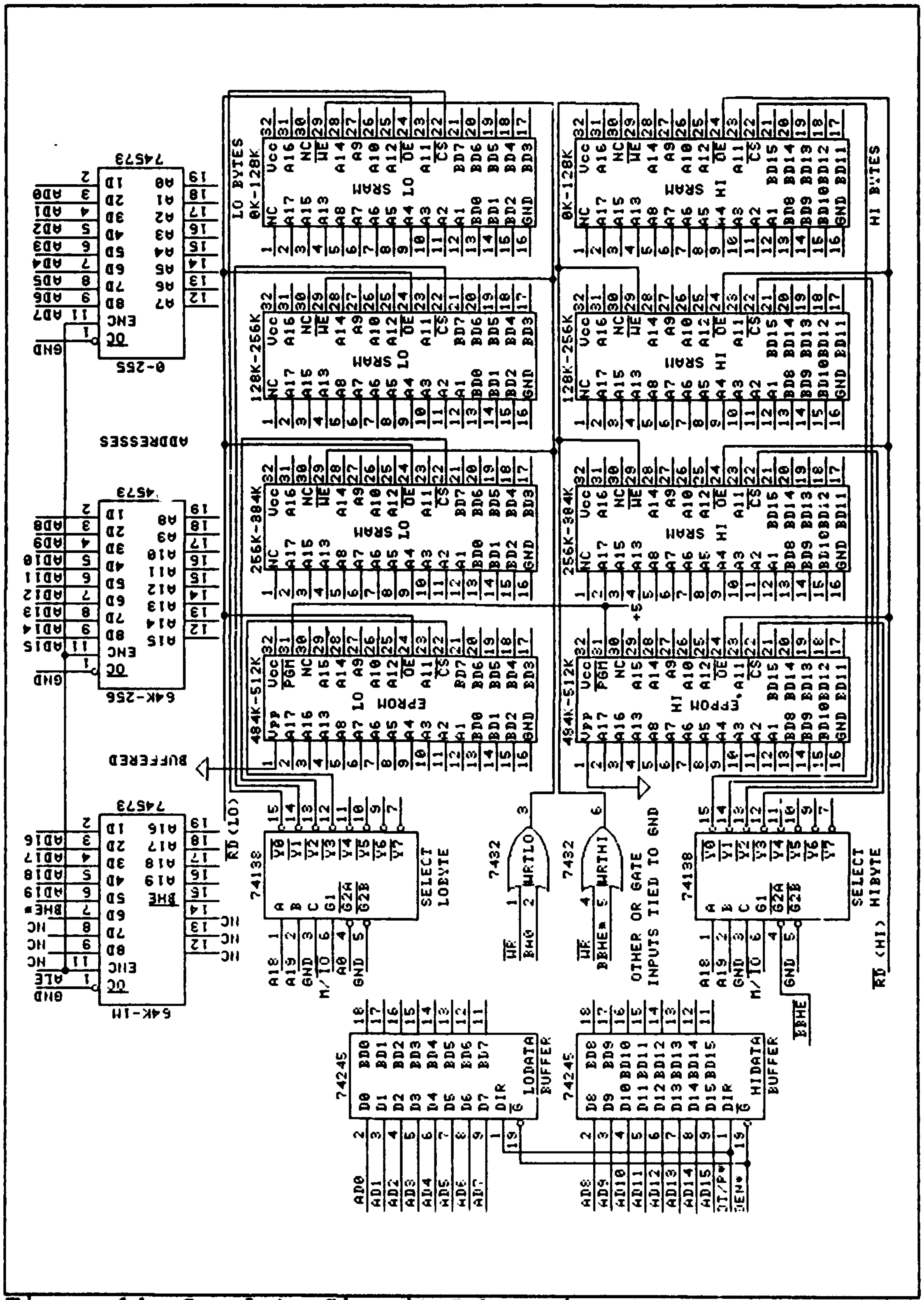


Figure 14. Complete Circuit Schematic--Memory

4. The Need for a Monitor

But now there were 15 reads and 1 write per program "cycle." It was becoming difficult to unambiguously verify correct circuit operation. It was becoming evident that a better way to monitor circuit operation than sticking an oscilloscope probe on various chip pins would need to be found. Also, the capacitance of the oscilloscope probe, itself, has the potential to change the circuit parameters (for HC CMOS, at least) just enough to make a circuit work which otherwise would not, or vice versa [Ref. 20:p. 575].

In fact, this phenomenon was observed in the lab. Often, the oscilloscope would indicate the circuit was not operating properly and would have to be reset, as one went about the circuit with the probe checking inputs or outputs on various pins. However, if the probe(s) was (were) clamped onto the circuit and a correct oscilloscope trace was obtained, the trace would remain unchanged for two hours or more. This was observed several times.

This also argues for a more reliable and less invasive means of monitoring the circuit performance. So the next goal, after completion of the complete incremental system check out, should be to set the board up with an RS-232C connector. Then the board could be hooked to a CRT monitor. The Space Systems technician had a monitor program available, written in assembly, which would allow the target board to be hooked up to a PC host. If this monitor program and hardware were installed, then one could write assembly, or higher level compiled programs, which could then be downloaded to the target RAM. This would eliminate the requirement for the tedious bother of machine language level hand coding and burning of the EPROMS.

But the monitor program requires extensive modification for application to the board's particular components and configuration. This has been left for the student(s) who will follow on. There was not enough time to sidetrack into the extensive set up of the complete monitoring system.

5. Timer

Before testing the timer directly a short program was written to test the M/IO* selection circuitry. This was done by running a program which would run a continuous loop which would select the unused pin number 10 (Y5 on the schematic) on the 54HC138 decoder with an appropriate OUT statement. (Register DX already having been loaded and register AL being all *don't cares* in this case.) (See Appendix D.) A periodic high pulse was observed when the IO command OUT was executed and pin number 10 on the 138 IO decoder was periodically selected. Figure 15 on the next page shows the pertinent circuitry.

Now that the M/IO* selection circuitry was shown to be working, the next step was to write a simple test program for direct testing of the 82C54 programmable interface timer.

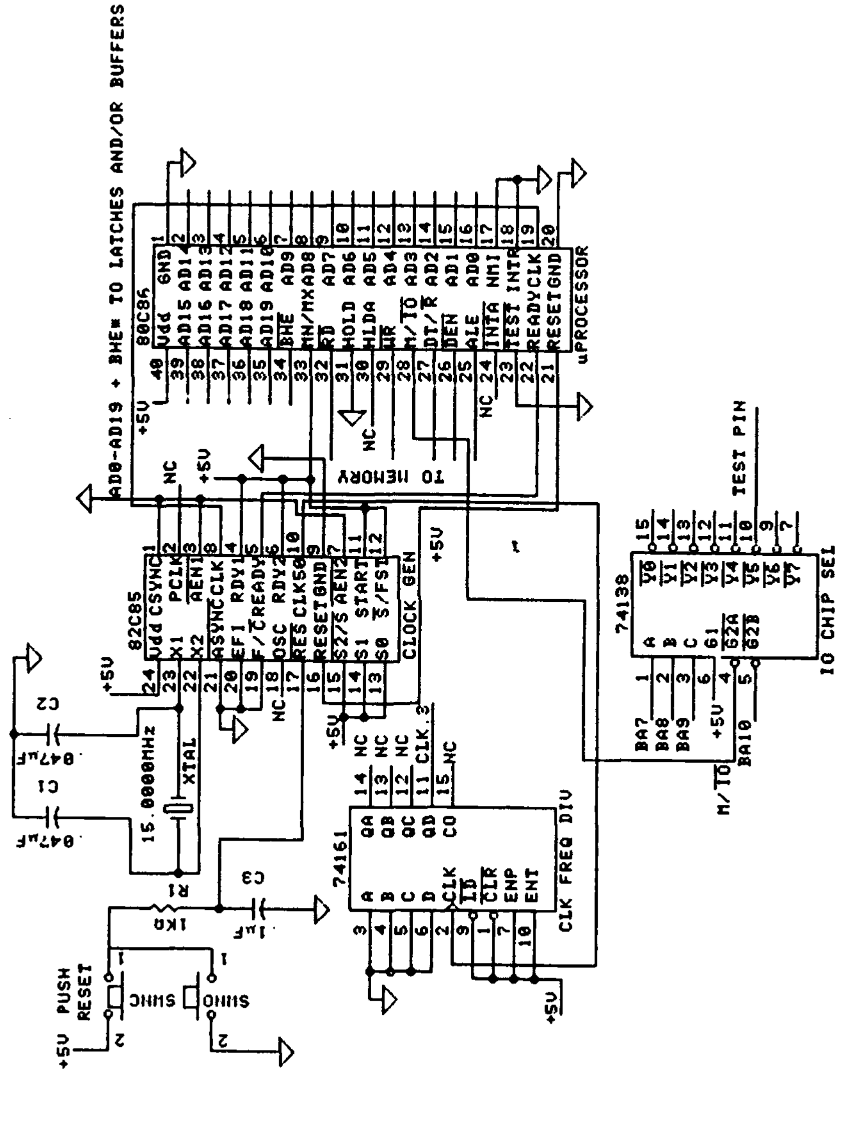


Figure 15. Memory/IO Test Schematic

Counter 2, Mode 3, was selected with an initial count of four. This should have resulted in a square wave on the *OUT2* output with a period of four 82C54 clocks. That is what was observed. (See Appendix E.)

6. Interrupt controller

Originally, this was to be the final portion of the piecemeal testing of the hardware components. It was also the most difficult. Initially, a program was written to test both the 8259 and the SCC at the same time. The timer was used in Mode 0 to provide a periodic interrupt signal. (In Mode 0, the timer stays low until it times out; then it goes high.) Part of the interrupt service routine (ISR) was to reset the timer as well as read a previously stored character from memory and write it to the SCC which would put the character out at 9600 baud.

But this turned out to be too ambitious for one program (i.e.,it would not work), so the program was split into parts. Thus, the SCC was left out of its holder and the program would just put out the stored character to the SCC port address. It was hoped that the data's periodic presence on the SCC chip holder data pins would be observable. This would verify proper operation of everything except the SCC itself.

The SCC could be tested separately, without using the timer or the interrupt controller, by having the CPU poll the SCC until its transmit buffer was empty and it would then be ready to accept another character from the CPU. Then, after all the components were verified as functioning correctly, they could all be tested as a complete system much more effectively after implementing the suggested monitor program with the RS-232C protocol.

This test of just the timer and the interrupt controller could not be made to work, either. It appeared as though the timer was not being reset, because *OUT2* stayed high and the CPU was cycling through low memory. Evidently, the CPU was putting the return address on the stack and then it was being immediately interrupted again, whereupon it again put the return address on the stack, and so on. The act of cycling through low memory indicated that the stack segment and the stack pointer had not been initialized.

But even so, the program still should have worked. The return address would just be located at 00000, 0FFFF, 0FFFE, and 0FFFD.

So the interrupt controller testing program was again broken into smaller parts with the first part being a test of the timer, alone, in Mode 0. (See Appendix F.) This first incremental program was finally made to work, but only after discovering that the 8254 data sheet's functional description was wrong.

But first, it was thought that it might be a problem with the timer taking too long to go low after being reset. Since the 8259 is clocked 8 times faster than the timer and the CPU is clocked 16 times faster than the timer, perhaps the 8259 reset was happening too quickly upon the heels of the timer reset? This is why there are delay loops between the timer reset and the 8259 reset. After the real problem was discovered, it was realized the delay loops are almost certainly not necessary, but it doesn't hurt to have them there to remove all doubt.

The first increment of the 8259 testing program shows what was later realized to be probably the main or only reason why the original program did not work. The data sheet on the 8254 timer is wrong. The data sheet says that once the timer times out in Mode 0, one needs only to reload the control word (CW) or reload the count and the timer will resume counting. Evidently, one needs to reload both the CW and the initial count for the counter to resume counting. The timer stayed high if one reloaded the CW, only. When one ran the program which resets both the CW and the initial count, suddenly the address and data lines and the timing relationships were all as expected. Figure 16 on the next page shows the circuit for the testing of the 82C54 (or 82C53) countdown timer.

The second increment of the interrupt controller testing used the timer in Mode 0 to provide the periodic interrupt. (See Appendix G.) *OUT2* is used as a signal to the 8259 on *IR3* that there is an interrupt to be serviced. The 8259 alerts the CPU with the *INT* line. The CPU replies with

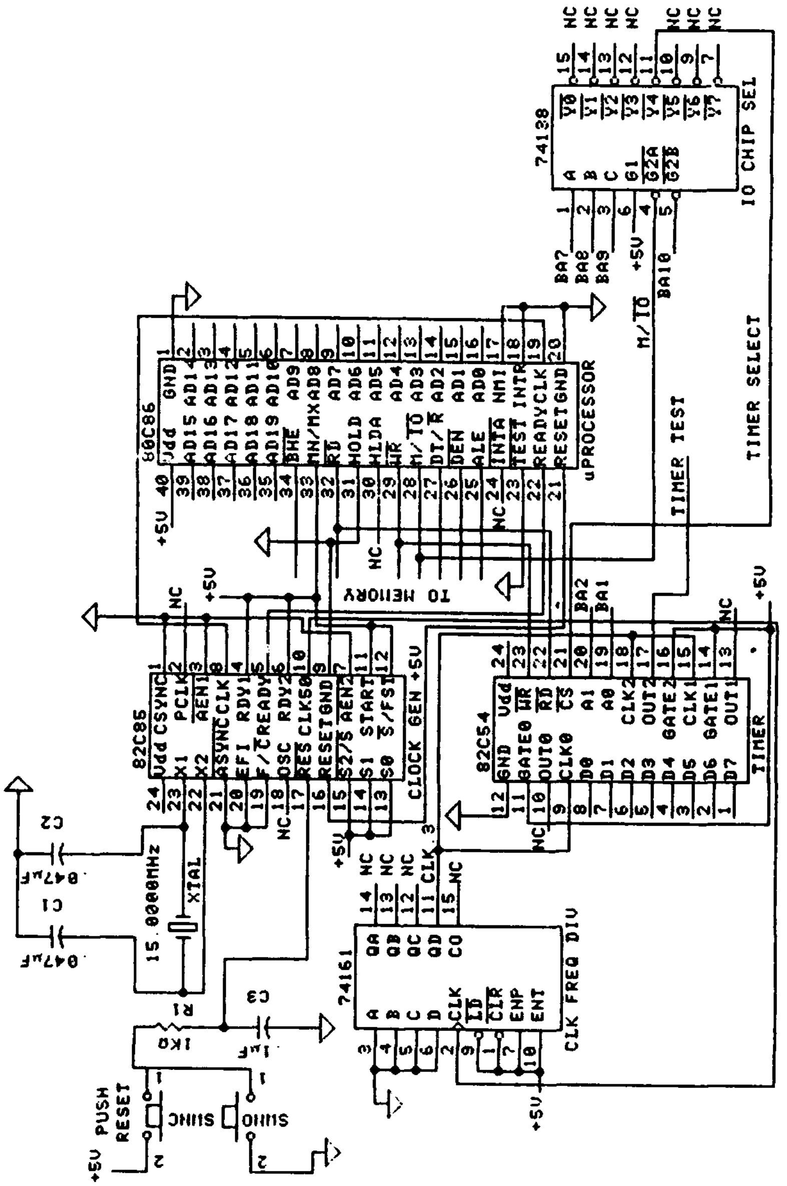


Figure 16. Countdown Timer Test Schematic

*INTA** and the 8259 gives the CPU the address of where to look for the address of the interrupt service routine (ISR). (The ISR address and the ISR itself must have first been loaded into memory.) The ISR just resets the timer. One must be careful not to enable interrupts until the timer actually goes low again. Otherwise, the CPU will keep interrupting itself and the ISR to reset the timer will never be executed and *OUT2* will stay high (causing interrupt upon interrupt) and the CPU will cycle through memory doing nothing but continually putting the return address on the stack, as already discussed above. (See Figure 17, next page.)

Also for the reasons already discussed above, it was very difficult to verify proper circuit operation for this increment. The only way to verify the circuit now was to trigger a trace on the oscilloscope with something like one of the ISR SRAM chip selects and check to see if the cycle time was what would be expected if the program were operating properly. Sometimes the reset button had to be pressed several times to get a steady trace with the correct cycle time. It was not clear, though, if the instability was an artifact of improper circuit operation or of a problem with the scope triggering mechanism.

Finally a steady trace was obtained from the *chip select* signal on the stack segment low byte SRAM (memory locations 40000 to 7FFFF). On the 0.02 msec per division scale, the cycle was 8.5 divisions long. This is 0.00017

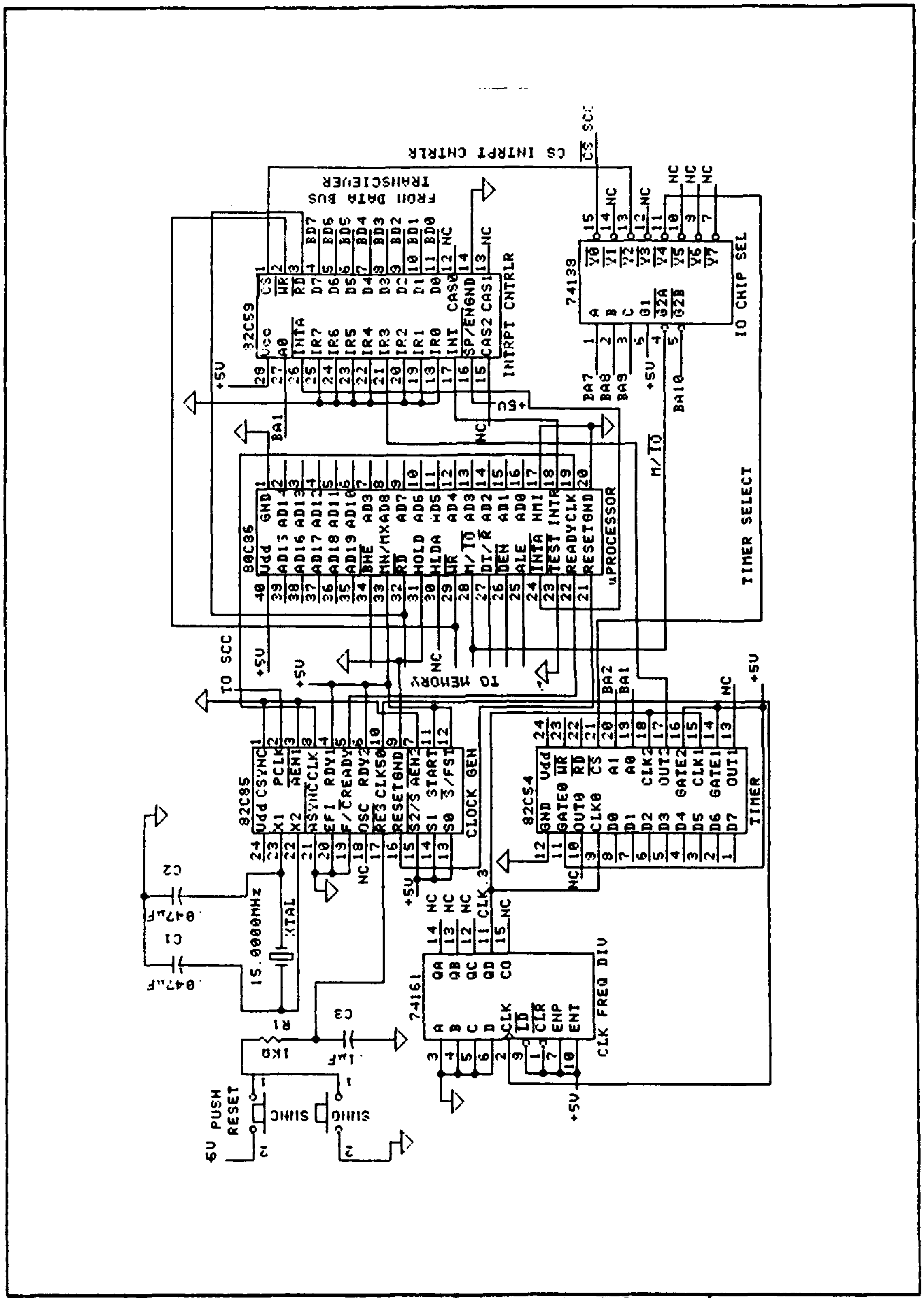


Figure 17. Interrupt Controller Test Schematic

seconds per cycle. Since the counter is loaded with 2Fh (= 47d), it should take 47d x 1/(5 MHz/16) = 0.0001504 seconds to complete. But of course, there is overhead time in reinitializing the counter when it times out. The return address has to be placed on the stack and the vector address of the ISR has to be loaded. Then the DS and DI registers have to be initialized. Finally, the CW and the initial count can be written to the timer and the timer will count down for 0.001504 seconds again. This accounts for the difference of the 98 or so CPU clock cycles between 0.000154 seconds and 0.00017 seconds.

Figure 18 (Page 60) is a copy of a Polaroid picture of the oscilloscope trace of *chip select* on the 40000-7FFFF, low byte memory SRAM. It shows the previously mentioned edge spikes very clearly. It also shows one cycle of a repeating pattern. There are 12 chip selects per cycle. This is expected. If one looks at the ISR code in the appendix which is run from memory location 40000h to 4001Dh, one will see there are 12 separate reads of code (a word at a time) between op code executions, i.e.:

```
•     BA 06                                          1
      02 B0
      (BA 06 02)-->LOAD DX with 0206
•     B0 EE                                          2
      (B0 B0)----->Load AL with B0
      (EE)-------->Write AL to counter 2
•     BA 04                                          3
      02 B8
      (BA 02 04)-->LOAD DX with 0204
•     2F 00                                          4
      (B8 2F 00)-->Load AX with 002F
```

```
•    EF B0                                       5
     (EF)-------->Write to counter 2 with 002F
•    08 48                                       6
     (B0 08)----->Load AL with 8
     (48)-------->Decrement AX
•    3C 00                                       7
     (3C 00)----->Compare AL with 0
•    EB 49                                       8
     (EB 49)----->Jump back 7 to code 48
•    74 02                                       9
     (74 02)----->Jump to next code if AL = 0
•    BA 00                                      10
     01 B0
     (BA 00 01)-->Load DX with 0100
•    20 EE                                      11
     (B0 20)----->Load AL with 20
     (EE)-------->Write AL to 8259
•    CF FB                                      12
     (FB)-------->Enable interrupts
     (CF)-------->Return from ISR
```

These 12 separate reads correspond to the 12 chip selects shown in the figure. (Probably not in this order.) This gives an example of the kind of procedure(s) followed to verify proper circuit operation.

Finally, for the third increment, a program was again written to repetitively output a stored character directly to the SCC. (See Appendix H.) This program was tried and it could not be made to work. The circuit, or the oscilloscope triggering, was just too unstable. Perhaps the program was incorrect (not likely because only a very small new part was added), or perhaps it was a function of some kind of performance margin being exceeded with respect to the SRAM memory. This problem would take too much time to solve without more advanced monitoring tools. The implementation of the host and target monitor system already mentioned

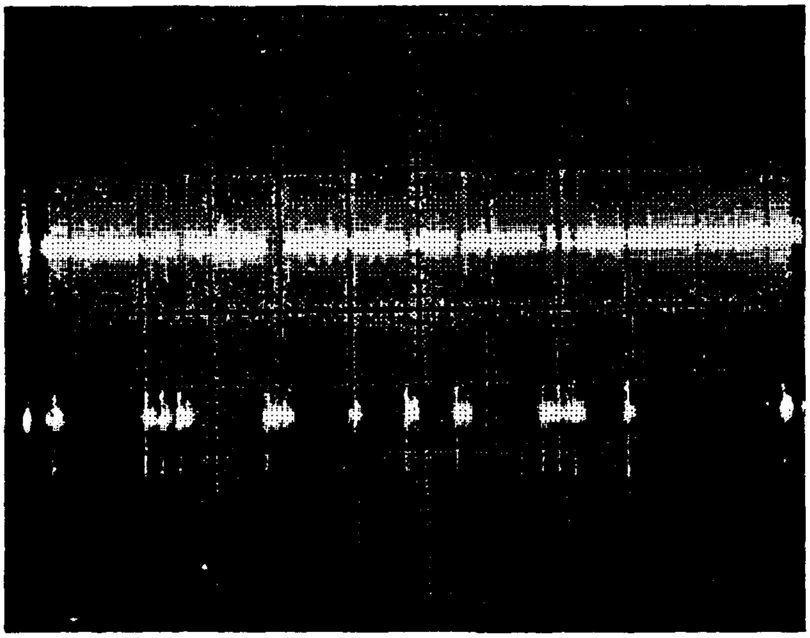


Figure 18. Interrupt Controller Test--Increment 2

should settle the issue. However, the interrupt controller was already shown to work, so a program was written to test just the SCC by itself to complete the hardware verification.

7. Serial Communications Controller (SCC)

The last program, to test the SCC, is shown in Appendix I. The CPU sends a character to the SCC's transmit buffer and the SCC is (or should be) programmed to put the character out at 9600 baud. For the first part of the program, the SCC is initialized to transmit the character at 9600 baud. Then, after the CPU reads the character from memory and loads it into the SCC's transmit buffer, the CPU keeps polling the SCC until it finds the SCC transmit buffer empty. When the CPU finds the buffer empty, it sends the SCC the same character from memory again and the process repeats.

One should detect a single character stream from the SCC at 9600 baud. A CRT was procured to hook up to the RS-232 port so the character could possibly be seen on the CRT screen. It was not clear from the data sheets available on the 85C30 SCC in the component catalog [Ref. 26] exactly how to set all the bits in the various control registe⁻⁻. The program in the appendix could not be made to work. (A Zilog technical representative was contacted and he explained it was very difficult to program the SCC with only the component description in the component catalog. He said he would be mailing the proper information. [Ref. 27])

Then the error was discovered. The final "EE" in the write registers initialization sequence had been inadvertently left off, so of course the baud rate generator was not enabled. This was corrected, and then one could see the *serial data out* pin going high and low at some changing frequency. One could only get a flash of the pulse as is periodically traversed the oscilloscope screen. This inability to lock on the signal is most likely caused by the character being put out at slightly different times on each pass. The CPU is so fast in executing the Tx buffer empty check loop that an inconsistent number of loops is probably performed each time before the next character is fetched and loaded in the SCC transmit buffer. Figures 19 and 20 on the

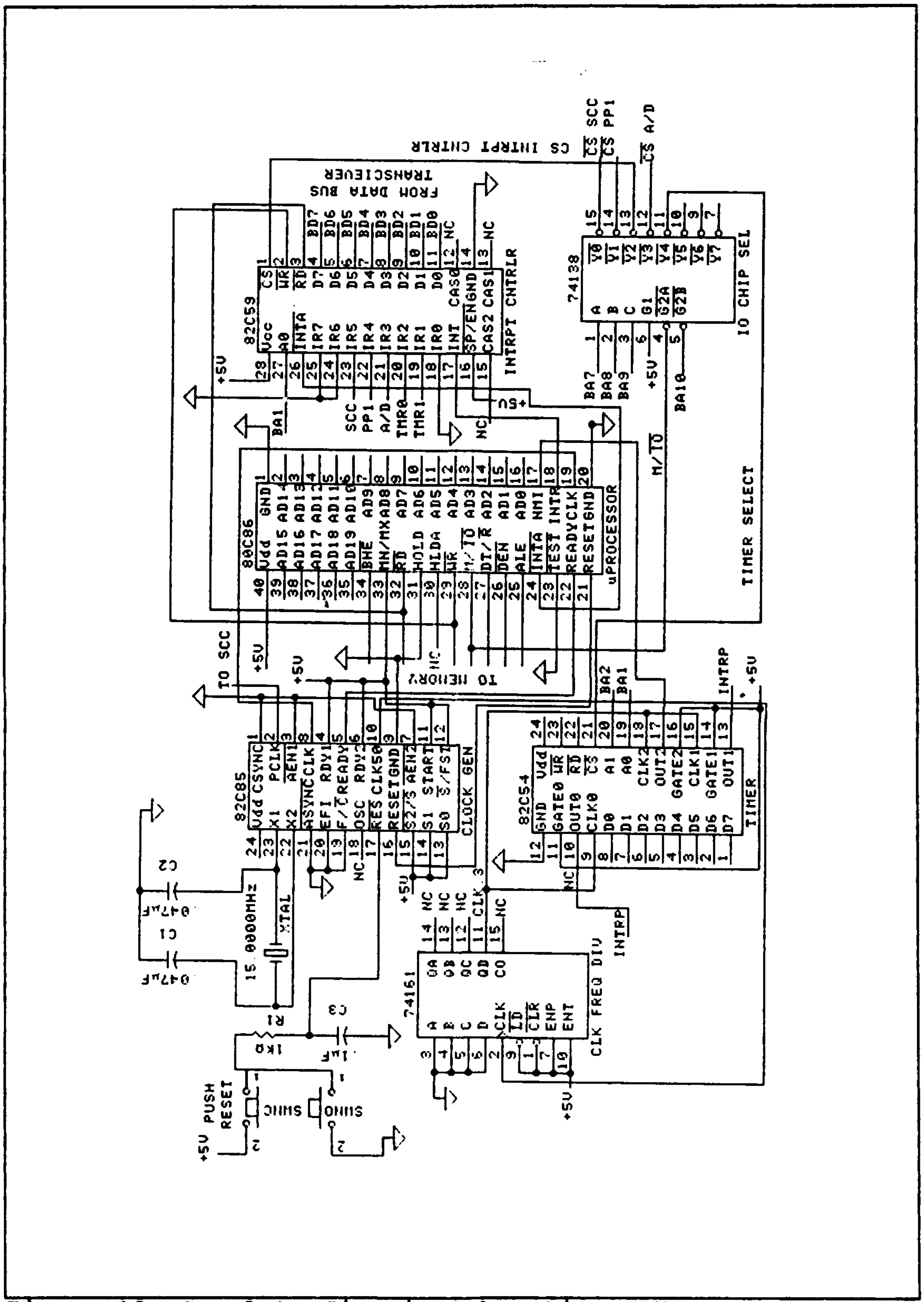


Figure 19. Complete Circuit Schematic--CPU

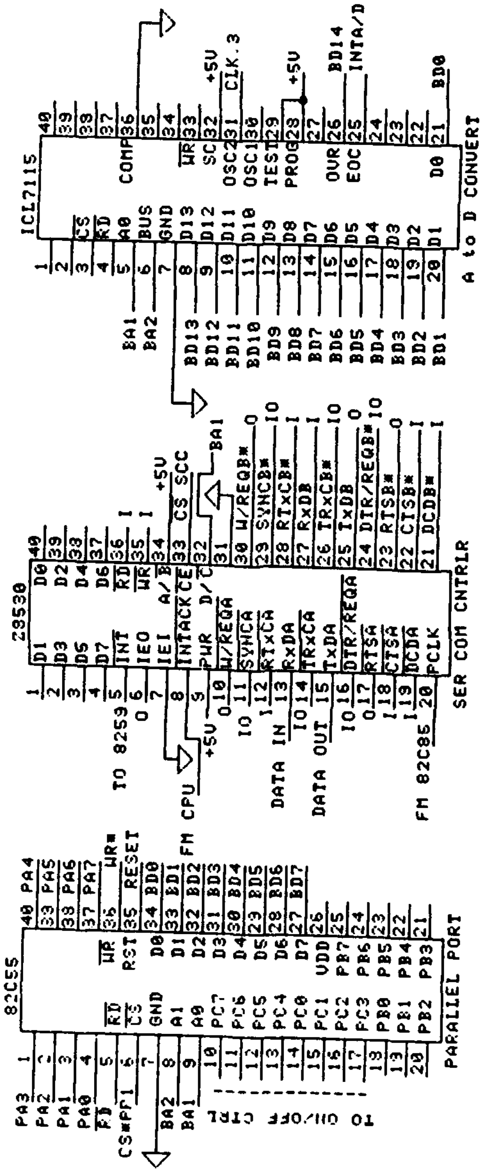


Figure 20. Complete Circuit Schematic--Peripherals

last two pages, in conjunction with Figure 14, constitute the entire computer system in schematic form.

All the components of the system have been shown to work as expected. The oscilloscope has been used to verify correct circuit operation for the complete system.

IV. RECOMMENDATIONS FOR FURTHER WORK

A. WRITE A CHARACTER TO THE CRT

This should be the next step, to verify the capability of correct operation of the SCC within the circuit. The program in Appendix I should give the next person to continue with this open ended project a good base with which to begin.

The following recommended actions are not necessarily in temporal or priority order. However, implementation of this CRT step will be instrumental in the next recommended step.

B. INSTALL HOST-TO-TARGET MONITOR PROGRAM

This will make the job of writing test programs much easier. An assembler or compiler can be used and the program code can be loaded directly to SRAM or into a file which can then be more quickly burned into the EPROM. An assembler could have been used for the present effort, but it was felt that the programs at this level were simple enough that the extra time it would have taken to implement an assembler could be better spent in coding the short programs directly.

C. WRITE OPERATING SYSTEM BOOTSTRAP KERNEL

This assumes that the operating system and application programs should be capable of being loaded from the ground. It is possible, though not likely, that a decision could be made that all programs will reside in ROM. If so, then SRAM will be used only for messages (and possibly telemetry data).

D. IMPLEMENT AX.25 PROTOCOL

Whether the AX.25 protocol resides in ROM or SRAM, its programming and validation will be an extensive project. There will probably have to be close coordination with the person overseeing the PANSAT communications subsystem at that time.

E. IMPLEMENT RESET

Experience in the laboratory has shown that this is one of the most important required external commands. If the computer system gets hit by a cosmic ray in the wrong place, or otherwise locks up or goes into an invalid state, there must be a way to force the system back to a known state. Hence the need for reset.

F. IDENTIFY OTHER REQUIREMENTS AND BEGIN HARDWARE OR SOFTWARE MODIFICATIONS

A partial list of other possible required tasks on the road to a fully functional flight-ready communications controller and processor are:

- Identify and implement any other required external commands
- Identify and incorporate need, if any, for backup batteries
- Identify any time critical housekeeping tasks
- Identify need for A/D converter and/or parallel port

APPENDIX A

Instruction Set Summary

8086 INSTRUCTION SET

CONTROL TRANSFER

CALL = Call:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Direct within segment	1 1 1 0 1 0 0 0	disp-low	disp-high
Indirect within segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m	
Direct intersegment	1 0 0 1 1 0 1 0	offset-low	offset-high
		seg-low	seg-high
Indirect intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m	

JMP = Unconditional Jump:			
Direct within segment	1 1 1 0 1 0 0 1	disp-low	disp-high
Direct within segment-short	1 1 1 0 1 0 1 1	disp	
Indirect within segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m	
Direct intersegment	1 1 1 0 1 0 1 0	offset-low	offset-high
		seg-low	seg-high
Indirect intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m	

RET = Return from CALL:			
Within segment	1 1 0 0 0 0 1 1		
Within seg. adding immed to SP	1 1 0 0 0 0 1 0	data-low	data-high
Intersegment	1 1 0 0 1 0 1 1		
Intersegment, adding immediate to SP	1 1 0 0 1 0 1 0	data-low	data-high
JE/JZ = Jump on equal/zero	0 1 1 1 0 1 0 0	disp	
JL/JNGE = Jump on less/not greater or equal	0 1 1 1 1 1 0 0	disp	
JLE/JNG = Jump on less or equal/not greater	0 1 1 1 1 1 1 0	disp	
JB/JNAE = Jump on below/not above or equal	0 1 1 1 0 0 1 0	disp	
JBE/JNA = Jump on below or equal/not above	0 1 1 1 0 1 1 0	disp	
JP/JPE = Jump on parity/parity even	0 1 1 1 1 0 1 0	disp	
JO = Jump on overflow	0 1 1 1 0 0 0 0	disp	
JS = Jump on sign	0 1 1 1 1 0 0 0	disp	
JNE/JNZ = Jump on not equal/not zero	0 1 1 1 0 1 0 1	disp	
JNL/JGE = Jump on not less/greater or equal	0 1 1 1 1 1 0 1	disp	
JNLE/JG = Jump on not less or equal/greater	0 1 1 1 1 1 1 1	disp	

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
JNB/JAE = Jump on not below/above or equal	0 1 1 1 0 0 1 1	disp
JNBE/JA = Jump on not below or equal/above	0 1 1 1 0 1 1 1	disp
JNP/JPO = Jump on not par/par odd	0 1 1 1 1 0 1 1	disp
JNO = Jump on not overflow	0 1 1 1 0 0 0 1	disp
JNS = Jump on not sign	0 1 1 1 1 0 0 1	disp
LOOP = Loop CX times	1 1 1 0 0 0 1 0	disp
LOOPZ/LOOPE = Loop while zero/equal	1 1 1 0 0 0 0 1	disp
LOOPNZ/LOOPNE = Loop while not zero/equal	1 1 1 0 0 0 0 0	disp
JCXZ = Jump on CX zero	1 1 1 0 0 0 1 1	disp

INT = Interrupt		
Type specified	1 1 0 0 1 1 0 1	type
Type 3	1 1 0 0 1 1 0 0	
INTO = Interrupt on overflow	1 1 0 0 1 1 1 0	
IRET = Interrupt return	1 1 0 0 1 1 1 1	

PROCESSOR CONTROL

CLC = Clear carry	1 1 1 1 1 0 0 0	
CMC = Complement carry	1 1 1 1 0 1 0 1	
STC = Set carry	1 1 1 1 1 0 0 1	
CLD = Clear direction	1 1 1 1 1 1 0 0	
STD = Set direction	1 1 1 1 1 1 0 1	
CLI = Clear interrupt	1 1 1 1 1 0 1 0	
STI = Set interrupt	1 1 1 1 1 0 1 1	
HLT = Halt	1 1 1 1 0 1 0 0	
WAIT = Wait	1 0 0 1 1 0 1 1	
ESC = Escape (to external device)	1 1 0 1 1 x x x	mod x x x r/m
LOCK = Bus lock prefix	1 1 1 1 0 0 0 0	

Footnotes:

AL = 8-bit accumulator
AX = 16-bit accumulator
CX = Count register
DS = Data segment
ES = Extra segment
Above/below refers to unsigned value
Greater = more positive;
Less = less positive (more negative) signed values
if d = 1 then "to" reg; if d = 0 then "from" reg
if w = 1 then word instruction; if w = 0 then byte instruction

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low

if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP
DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low

Mnemonics © Intel, 1978

if s w = 01 then 16 bits of immediate data form the operand
if s w = 11 then an immediate data byte is sign extended to form the 16-bit operand
if v = 0 then "count" = 1; if v = 1 then "count" in (CL)
x = don't care
z is used for string primitives for comparison with ZF FLAG

SEGMENT OVERRIDE PREFIX

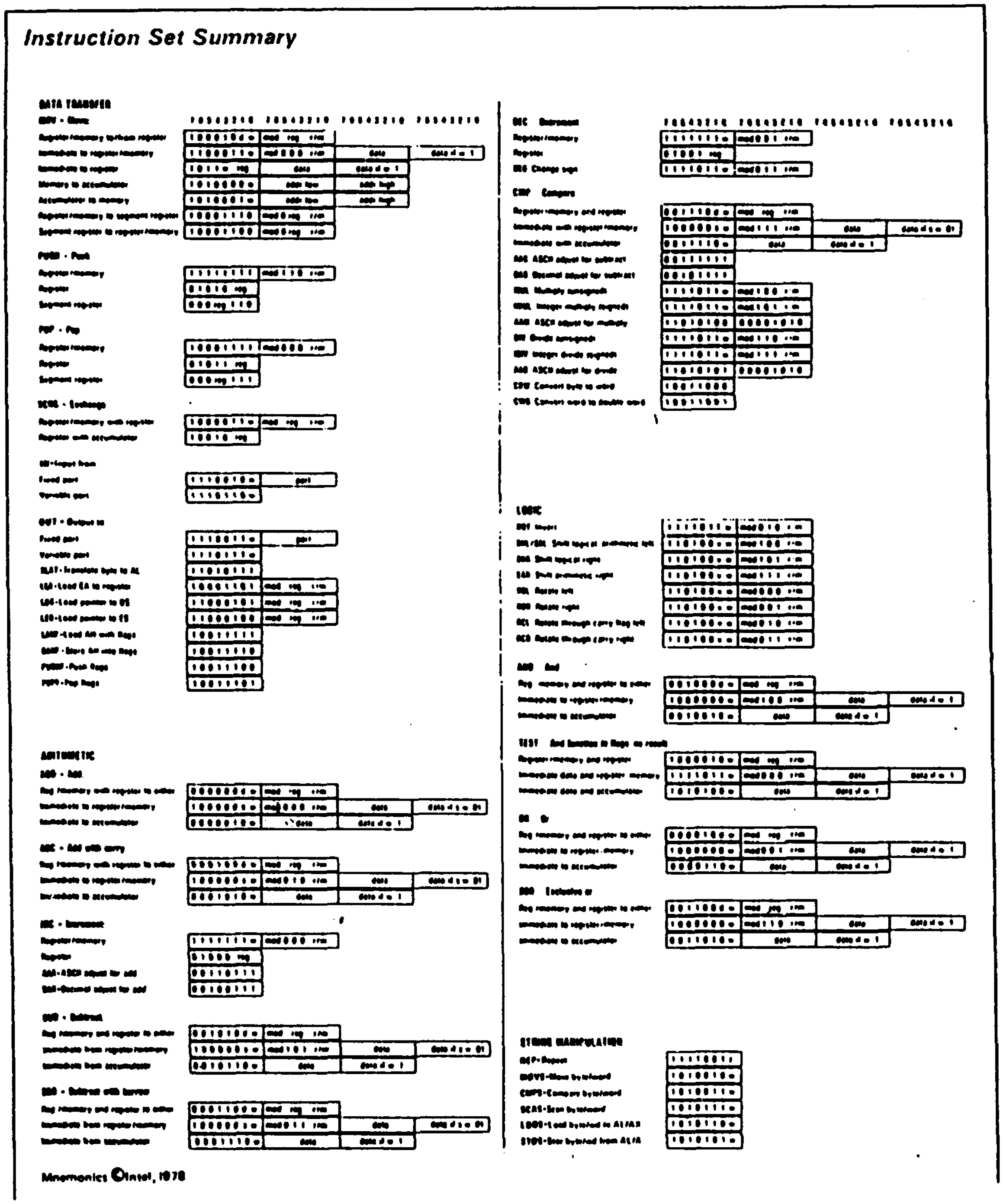
0 0 1 reg 1 1 0

REG is assigned according to the following table:

16-Bit (w = 1)		8-Bit (w = 0)		Segment	
000	AX	000	AL	00	ES
001	CX	001	CL	01	CS
010	DX	010	DL	10	SS
011	BX	011	BL	11	DS
100	SP	100	AH		
101	BP	101	CH		
110	SI	110	DH		
111	DI	111	BH		

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:

FLAGS = X:X:X:X:(OF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF)



Instruction Set Summary

DATA TRANSFER	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
MOV - Move:				
Register/memory to/from register	1 0 0 0 1 0 d w	mod reg r/m		
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1
Immediate to register	1 0 1 1 w reg	data	data if w = 1	
Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high	
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high	
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 reg r/m		
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 reg r/m		
PUSH - Push:				
Register/memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m		
Register	0 1 0 1 0 reg			
Segment register	0 0 0 reg 1 1 0			
POP - Pop:				
Register/memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m		
Register	0 1 0 1 1 reg			
Segment register	0 0 0 reg 1 1 1			
XCHG - Exchange:				
Register/memory with register	1 0 0 0 0 1 1 w	mod reg r/m		
Register with accumulator	1 0 0 1 0 reg			
IN=Input from:				
Fixed port	1 1 1 0 0 1 0 w	port		
Variable port	1 1 1 0 1 1 0 w			
OUT - Output to:				
Fixed port	1 1 1 0 0 1 1 w	port		
Variable port	1 1 1 0 1 1 1 w			
XLAT=Translate byte to AL	1 1 0 1 0 1 1 1			
LEA=Load EA to register	1 0 0 0 1 1 0 1	mod reg r/m		
LDS=Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m		
LES=Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m		
LAHF=Load AH with flags	1 0 0 1 1 1 1 1			
SAHF=Store AH into flags	1 0 0 1 1 1 1 0			
PUSHF=Push flags	1 0 0 1 1 1 0 0			
POPF=Pop flags	1 0 0 1 1 1 0 1			

ARITHMETIC				
ADD - Add:				
Reg./memory with register to either	0 0 0 0 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s:w = 01
Immediate to accumulator	0 0 0 0 0 1 0 w	data	data if w = 1	
ADC - Add with carry:				
Reg./memory with register to either	0 0 0 1 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s:w = 01
Immediate to accumulator	0 0 0 1 0 1 0 w	data	data if w = 1	
INC - Increment:				
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m		
Register	0 1 0 0 0 reg			
AAA=ASCII adjust for add	0 0 1 1 0 1 1 1			
DAA=Decimal adjust for add	0 0 1 0 0 1 1 1			
SUB - Subtract:				
Reg./memory and register to either	0 0 1 0 1 0 d w	mod reg r/m		
Immediate from register/memory	1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s:w = 01
Immediate from accumulator	0 0 1 0 1 1 0 w	data	data if w = 1	
SBB - Subtract with borrow:				
Reg./memory and register to either	0 0 0 1 1 0 d w	mod reg r/m		
Immediate from register/memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s:w = 01
Immediate from accumulator	0 0 0 1 1 1 0 w	data	data if w = 1	

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
DEC Decrement:				
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m		
Register	0 1 0 0 1 reg			
NEG Change sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m		
CMP Compare:				
Register/memory and register	0 0 1 1 1 0 d w	mod reg r/m		
Immediate with register/memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s:w = 01
Immediate with accumulator	0 0 1 1 1 1 0 w	data	data if w = 1	
AAS ASCII adjust for subtract	0 0 1 1 1 1 1 1			
DAS Decimal adjust for subtract	0 0 1 0 1 1 1 1			
MUL Multiply (unsigned)	1 1 1 1 0 1 1 w	mod 1 0 0 r/m		
IMUL Integer multiply (signed)	1 1 1 1 0 1 1 w	mod 1 0 1 r/m		
AAM ASCII adjust for multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0		
DIV Divide (unsigned)	1 1 1 1 0 1 1 w	mod 1 1 0 r/m		
IDIV Integer divide (signed)	1 1 1 1 0 1 1 w	mod 1 1 1 r/m		
AAD ASCII adjust for divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0		
CBW Convert byte to word	1 0 0 1 1 0 0 0			
CWD Convert word to double word	1 0 0 1 1 0 0 1			

LOGIC				
NOT Invert	1 1 1 1 0 1 1 w	mod 0 1 0 r/m		
SHL/SAL Shift logical/arithmetic left	1 1 0 1 0 0 v w	mod 1 0 0 r/m		
SHR Shift logical right	1 1 0 1 0 0 v w	mod 1 0 1 r/m		
SAR Shift arithmetic right	1 1 0 1 0 0 v w	mod 1 1 1 r/m		
ROL Rotate left	1 1 0 1 0 0 v w	mod 0 0 0 r/m		
ROR Rotate right	1 1 0 1 0 0 v w	mod 0 0 1 r/m		
RCL Rotate through carry flag left	1 1 0 1 0 0 v w	mod 0 1 0 r/m		
RCR Rotate through carry right	1 1 0 1 0 0 v w	mod 0 1 1 r/m		
AND And:				
Reg./memory and register to either	0 0 1 0 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w = 1
Immediate to accumulator	0 0 1 0 0 1 0 w	data	data if w = 1	
TEST And function to flags no result:				
Register/memory and register	1 0 0 0 0 1 0 w	mod reg r/m		
Immediate data and register/memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1
Immediate data and accumulator	1 0 1 0 1 0 0 w	data	data if w = 1	
OR Or:				
Reg./memory and register to either	0 0 0 0 1 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w = 1
Immediate to accumulator	0 0 0 0 1 1 0 w	data	data if w = 1	
XOR Exclusive or:				
Reg./memory and register to either	0 0 1 1 0 0 d w	mod reg r/m		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w = 1
Immediate to accumulator	0 0 1 1 0 1 0 w	data	data if w = 1	

STRING MANIPULATION	
REP=Repeat	1 1 1 1 0 0 1 z
MOVS=Move byte/word	1 0 1 0 0 1 0 w
CMPS=Compare byte/word	1 0 1 0 0 1 1 w
SCAS=Scan byte/word	1 0 1 0 1 1 1 w
LODS=Load byte/wd to AL/AX	1 0 1 0 1 1 0 w
STOS=Stor byte/wd from AL/A	1 0 1 0 1 0 1 w

Mnemonics ©Intel, 1978

[Ref. 25:pp. 4-76--4-77]

APPENDIX B

INITIAL CIRCUIT TEST PROGRAM

The program continuously jumps to itself at starting address FFFF0h.

```
8086            LO      HI
ADDR            BYTE    BYTE
                EPROM   EPROM
                ADDR    ADDR
                             _____
FFFF4  FF       1FFFA             ;Jmp to FFFF0
FFFF3      FF           1FFF9     ;
FFFF2  00       1FFF9             ;
FFFF1      00           1FFF8     ;
FFFF0  EA       1FFF8        _____; <---(Pwr on rst jmp)
```

APPENDIX C

INITIAL MEMORY TEST PROGRAM

Program continuously writes the arbitrary number "5" to arbitrary SRAM memory location BFFF0.

```
8086                LO       HI
ADDR                BYTE     BYTE
                    EPROM    EPROM
                    ADDR     ADDR
                                  ______
FFFF4  FF           1FFFA             ;Jmp to FFFDC
FFFF3       FD               1FFF9    ;
FFFF2  00           1FFF9             ;
FFFF1       0C               1FFF8    ;
FFFF0  EA           1FFF8       ______; <---(Pwr on rst jmp)
FFFEF       FF               1FFF7    ;Jmp to FFFF0
FFFEE  FF           1FFF7             ;
FFFED       00               1FFF6    ;
FFFEC  00           1FFF6             ;
FFFEB       EA               1FFF5______;
FFFEA  14           1FFF5             ;Mov B fm M[BFFF0] to DL
                                      ;register
FFFE9       8A               1FFF4______;
FFFE8  90           1FFF4       ______;NOP
FFFE7       05               1FFF3    ;Mov B immed #05 to
                                      ;M[BFFF0]
FFFE6  04           1FFF3             ;(M[BFFFx4 + 0 = BFFF0])
FFFE5       C6               1FFF2______;
FFFE4  00           1FFF2             ;Mov W immed #0000 to SI
                                      ;register
FFFE3       00               1FFF1    ;
FFFE2  BE           1FFF1       ______;
FFFE1       DE               1FFF0    ;Mov W SI reg to DS seg
                                      ;register
FFFE0  8E           1FFF0       ______;
FFFDF       BF               1FFEF    ;Mov W immed #BFFF to SI
                                      ;register
FFFDE  FF           1FFEF             ;
FFFDD       C6               1FFEE    ;
FFFDC  C7           1FFEE       ______;
```

APPENDIX D

MEMORY/IO TEST PROGRAM

Program to verify that the CPU asserts IO* of the *M/IO** signal when writing to a variable port--in this case pin 10 (Y5) on the IO chip select Decoder. It was written because the program to load the 82C53 (cheap prototype replacement for the 82C54) with a control word (Mode 3) and an initial count (04) didn't appear to be working. To eliminate the *M/IO** select circuitry as the culprit, this program tests to make sure IO* is asserted when OUT writes to a variable port. The proper *chip select* line is asserted.

```
8086            LO      HI
ADDR            BYTE    BYTE
                EPROM   EPROM
                ADDR    ADDR
                             _____
FFFF4  FF       1FFFA             ;Jmp to FFFE8
FFFF3      F0           1FFF9     ;
FFFF2  00       1FFF9             ;
FFFF1      E8           1FFF8     ;
FFFF0  EA       1FFF8        _____; <---(Pwr on rst jmp)
FFFEF      EE           1FFF7_____;OUT to [port DX] with
                                  ;(AL)
FFFEE  FF       1FFF7             ;Load AL with #FF
FFFED      C0           1FFF6     ;
FFFEC  C6       1FFF6        _____;
FFFEB      02           1FFF5     ;Load DX with #0280 (Port
                                  ;"5")
FFFEA  80       1FFF5           ;A11A10A9A8A7A6A5A4A3A2A1A0
FFFE9      C2           1FFF4   ;  X  0 1 0 1 X X X X 0 0 X
                                  ;        ¦       ¦
                                  ;    2       8       0
FFFE8  C7       1FFF4        _____;
```

Line Y5 is asserted, as expected, when IO* of *M/IO** is asserted.

APPENDIX E

INITIAL 82C54 TIMER TEST PROGRAM

The program loads the 82C54 (or '53) timer (counter 2) with a control word (byte) indicating mode 3 (square wave generator) with a period loaded as an initial count, in this case "4". The timer's cycle, therefore, is 1/4 of the input clock's, or 1/4 of 0.3125 MHz = 78.125 kHz.

```
8086                LO      HI
ADDR                BYTE    BYTE
                    EPROM   EPROM
                    ADDR    ADDR
                                 _____
FFFF4  FF           1FFFA             ;Jmp to FFFDB
FFFF3      F0               1FFF9     ;
FFFF2  00           1FFF9             ;
FFFF1      DB               1FFF8     ;
FFFF0  EA           1FFF8        _____; <---(Pwr on rst jmp)
FFFEF      FF               1FFF7     ;Jmp to FFFEB-->
                                      ;Continuous loop--
FFFEE  F0           1FFF7             ;timer set
FFFED      00               1FFF6     ;(Jmp to FFFDB-keep
                                      ;setting the timer,
FFFEC  EB (DB)      1FFF6             ;over and over)
FFFEB      EA               1FFF5_____;
FFFEA  EE           1FFF5             ;OUT to [DX port] with
                                      ;(AL=04)--port addr
                                      ;is Y4 (Timer)
                                 _____;Load cntr 2 with init
                                      ;count
FFFE9      02               1FFF4     ;Load DX with #027C*
FFFE8  7C           1FFF4           ;A11A10A9A8A7A6A5A4A3A2A1A0
FFFE7      C2               1FFF3   ;  X  0 1 0 0 X X X X 1 0 X
                                      ;         !       !
                                      ;     2       7       C
                                      ;Port Y4, load into cntr
                                      ;2 init value (in AL)
FFFE6  C7           1FFF3        _____;
FFFE5      04               1FFF2     ;Load AL with #04*
FFFE4  C0           1FFF2             ;(Initial counter value)
FFFE3      C6               1FFF1_____;
FFFE2  EE           1FFF1             ;OUT to [DX port] with
                                      ;(AL = 96)--Control word
                                      ;(AL) selects cntr 2,
                                 _____;Mode 3
```

```
FFFE1      96           1FFF0     ;Load AL with #96*
FFFE0  C0       1FFF0             ;Cntr 2, LSB, Mode3,
                                  ;Binary cnt
FFFDF      C6           1FFEF_____;
FFFDE  02       1FFEF             ;Load DX with #027E*
FFFDD      7E           1FFEE   ;A11A10A9A8A7A6A5A4A3A2A1A0
FFFDC  C2       1FFEE           ;  0  0 1 0 0 X X X X 1 1 X
                                  ;        ¦       ¦
                                  ;    2       7       E
                                  ;Port Y4 (Timer), write
                                  ;cntrl "word" (to come
FFFDB      C7           1FFED_____;from AL)
```

26 bytes

*The 2 byte opcode loads of DX (C7 C2) and AL (C6 C0) can be replaced by the 1 byte opcodes BA and B0, respectively.

APPENDIX F

INTRPT CNTRLR TEST PROGRAM: INCREMENT 1

This program only tests the 8254 timer in mode 0 as a prelude to the full-blown interrupt controller test program (iteration 3).

First, it was verified that the timer could be programmed to time out and go high while the CPU was doing a loop for a certain number of iterations. The address and data lines and the timing relationships were as expected.

The data sheet on the 8254 timer is wrong. It said that once the timer times out in Mode 0, one needs only to reload the CW or reload the count and the timer will resume counting. Evidently, one needs to reload both the CW and the initial count, as in this program, for the counter to resume counting.

The code is read from bottom to top.

```
8086            LO      HI
ADDR            BYTE    BYTE
                EPROM   EPROM
                ADDR    ADDR
                             _____
FFFF4  FF       1FFFA             ;Jmp to FFFCA
FFFF3      F0           1FFF9     ;
FFFF2  00       1FFF9             ;
FFFF1      CA           1FFF8     ;
FFFF0  EA       1FFF8        _____; <---(Pwr on rst jmp)
                             _____
FFFEF      FF           1FFF7     ;Jmp to FFFD6
FFFEE  F0       1FFF7             ;(Reset timer after AX
FFFED      00           1FFF6     ;counts down from 127d)
FFFEC  D6       1FFF6             ;
FFFEB      EA           1FFF5_____;
FFFEA  EE       1FFF5        _____;Write init cnt (02) to
FFFE9      02           1FFF4     ;timer; Load AL with init
FFFE8  B0       1FFF4        _____;count = 02
FFFE7      02           1FFF3     ;Move #0204 to DX reg
FFFE6  04       1FFF3             ;Cntr 2 is to be loaded
FFFE5      BA           1FFF2_____;with initial count
FFFE4  EE       1FFF2        _____;Write cntrl W (90) to
                                  ;timer
FFFE3      90           1FFF1     ;Load AL with CW = 90
FFFE2  B0       1FFF1        _____;(Mode 0, LSB, cntr 2)
```

```
FFFE1           02                    1FFF0       ;Move #0206 to DX
                                                  ;register
FFFE0    06                 1FFF0                 ;The CW in AL is to be
FFFDF           BA                    1FFEF       ;written to the cntr 2 CW
                                           _______;register
FFFDE    F9                 1FFEF                 ;Else, Short Jump to
                                                  ;FFFD8 to decrement
FFFDD           EB                    1FFEE_______;AX again (F9 = -7d)
FFFDC    02                 1FFEE                 ;If AX = 0, jump 2
FFFDB           74                    1FFED_______;addresses, to FFFDF
FFFDA    00                 1FFED                 ;CMP AX with 0
FFFD9           3C                    1FFEC_______;
FFFD8    48                 1FFEC          _______;Decrement AX
FFFD7           7F                    1FFEB       ;Load AX with 127d
FFFD6    B0                 1FFEB          _______;
                                                  ;OUT initial count to
FFFD5           EE                    1FFEA_______;counter 2
FFFD4    02                 1FFEA                 ;Load initial count of 2
FFFD3           B0                    1FFE9_______;into AL
FFFD2    02                 1FFE9                 ;Load DX with #0204 for
FFFD1           04                    1FFE8       ;timer initial count
FFFD0    BA                 1FFE8          _______;(from AL to be loaded)
FFFCF           EE                    1FFE7_______;OUT the CW to the timer
FFFCE    90                 1FFE7                 ;Load the CW for Mode 0,
FFFCD           B0                    1FFE6_______;LSB, counter 2 into AL
FFFCC    02                 1FFE6                 ;Ld DX with #0206 for
FFFCB           06                    1FFE5       ;selecting timer, write
FFFCA    BA                 1FFE5          _______;CW (with CW in AL)

43 bytes
```

APPENDIX G

INTRPT CNTRLR TEST PROGRAM: INCREMENT 2

In step one, it was verified that the timer could be programmed to run in Mode 0. When *OUT2* on the 8254 went high it stayed high until the timer was reset. The timer was reset only after the CPU looped for a large number of iterations. When the CPU was done, it reset the timer and the process was continually repeated.

In this iteration, when the timer times out, *OUT2* is used as a signal to the 8259 on *IR3* that there is an interrupt to be serviced. The 8259 alerts the CPU with the *INT* line. The CPU replies with *INTA** and the 8259 gives the CPU the address of where to look for the address of the interrupt service routine (ISR). (The ISR address and the ISR itself have to have been first loaded into memory.) The ISR just resets the timer.

The code is read from bottom to top.

```
8086            LO      HI
ADDR            BYTE    BYTE
                EPROM   EPROM
                ADDR    ADDR

                             _____
FFFF4  FF       1FFFA             ;Jmp to FFF61
FFFF3      F0           1FFF9     ;
FFFF2  00       1FFF9             ;
FFFF1      61           1FFF8     ;
FFFF0  EA       1FFF8        _____; <---(Pwr on rst jmp)
                             _____
FFFEF      FF           1FFF7     ;Jmp to FFFEB
FFFEE  F0       1FFF7             ;The CPU stays here until
FFFED      00           1FFF6     ;it's interrupted
FFFEC  EB       1FFF6             ;
FFFEB      EA           1FFF5_____;
                             _____
FFFEA  FB       1FFF5        _____;Enable interrupts
```

Load the interrupt service routine. The ISR resets the timer and clears the highest priority ISR bit in the 8259 (there is only one in this case). Because the 8259 runs at 2.5 MHz and the timer only runs at .3125 MHz, there is a little delay loop to make sure the timer has gone low after being reset before the ISR bit is reset and interrupts are again enabled.

The delay loop is almost certainly not required, but it doesn't hurt to have it.

```
                              ______
FFFE9       CF           1FFF4      ;Code CF-->return from
FFFE8   FB       1FFF4              ;ISR; Code FB-->Enable
FFFE7       05           1FFF3      ;interrupts
FFFE6   C7       1FFF3        ______;
FFFE5       47           1FFF2______;
FFFE4   47       1FFF2        ______;
FFFE3       EE           1FFF1      ;Code EE-->OUT to reset
                                    ;ISR bit
FFFE2   20       1FFF1              ;Code B0 20-->Load AL
                                    ;with 20 ("20" for OCW2
FFFE1       05           1FFF0      ;will reset highest
FFFE0   C7       1FFF0        ______;priority ISR bit)
FFFDF       47           1FFEF______;
FFFDE   47       1FFEF        ______;
FFFDD       B0           1FFEE      ;Code BA 00 01-->Load DX
FFFDC   01       1FFEE              ;with addr to write OCW2
FFFDB       05           1FFED      ;to 8259 to reset ISR bit
FFFDA   C7       1FFED              ;for IR3; (OUT2 has had
                                    ;plenty of time to go low
                              ______;because AX counted down
FFFD9       47           1FFEC______;from 8)
FFFD8   47       1FFEC        ______;
FFFD7       00           1FFEB      ;
FFFD6   BA       1FFEB              ;
FFFD5       05           1FFEA      ;
FFFD4   C7       1FFEA        ______;
FFFD3       47           1FFE9______;
FFFD2   47       1FFE9        ______;
FFFD1       F9           1FFE8      ;Code EB 49-->Jump back 7
FFFD0   EB       1FFE8              ;counts to opcode 48 to
FFFCF       05           1FFE7      ;decrement AX again
FFFCE   C7       1FFE7        ______;
FFFCD       47           1FFE6______;
FFFCC   47       1FFE6        ______;
FFFCB       02           1FFE5      ;Code 74 02-->If AX was 0
FFFCA   74       1FFE5              ;then jump forward 2
FFFC9       05           1FFE4      ;counts to BA opcode;
FFFC8   C7       1FFE4        ______;else, go to next opcode:
FFFC7       47           1FFE3______;EB
FFFC6   47       1FFE3        ______;
FFFC5       00           1FFE2      ;Code 3C 00-->CMP AX with
FFFC4   3C       1FFE2              ;0
FFFC3       05           1FFE1      ;
FFFC2   C7       1FFE1        ______;
FFFC1       47           1FFE0______;
FFFC0   47       1FFE0        ______;
```

```
FFFBF      48           1FFDF     ;Code B0 08--Load AL with
FFFBE  08       1FFDF             ;8
FFFBD      05           1FFDE     ;Code 48-->Decrement AX
FFFBC  C7       1FFDE        _____;
FFFBB      47           1FFDD_____;
FFFBA  47       1FFDD        _____;
FFFB9      B0           1FFDC     ;Code EF-->OUT with
FFFB8  EF       1FFDC             ;initial count to
FFFB7      05           1FFDB     ;cntr 2
FFFB6  C7       1FFDB        _____;
FFFB5      47           1FFDA_____;
FFFB4  47       1FFDA        _____;
FFFB3      00           1FFD9     ;Code B8 2F 00-->load AX
FFFB2  2F       1FFD9             ;with 00 2F as initial
FFFB1      05           1FFD8     ;count
FFFB0  C7       1FFD8        _____;
FFFAF      47           1FFD7_____;
FFFAE  47       1FFD7        _____;
FFFAD      B8           1FFD6     ;Code BA 04 02-->Load DX
FFFAC  02       1FFD6             ;with addr to load the
FFFAB      05           1FFD5     ;cntr 2 with the initial
FFFAA  C7       1FFD5             ;word size count (from
                             _____;AX)
FFFA9      47           1FFD4_____;
FFFA8  47       1FFD4        _____;
FFFA7      04           1FFD3     ;
FFFA6  BA       1FFD3             ;
FFFA5      05           1FFD2     ;
FFFA4  C7       1FFD2        _____;
FFFA3      47           1FFD1_____;DI = 5
FFFA2  47       1FFD1        _____;
FFFA1      EE           1FFD0     ;Code B0 B0-->load AL
FFFA0  B0       1FFD0             ;with cntr 2 CW, Mode 0,
FFF9F      05           1FFCF     ;word size initial count
FFF9E  C7       1FFCF        _____;Code EE-->WR the CW B0
                                  ;to AL)
FFF9D      47           1FFCE_____;DI = 4
FFF9C  47       1FFCE        _____;
FFF9B      B0           1FFCD     ;Move code "02 B0" to
                                  ;M[DS + DI]
FFF9A  02       1FFCD             ; = M[40002]
FFF99      05           1FFCC     ;Code "BA 06 02"-->Load
FFF98  C7       1FFCC             ;DX with addr where to ld
                             _____;AL cntr 2 CW
FFF97      47           1FFCB_____;Increment DI-->DI = 2
FFF96  47       1FFCB        _____;Increment DI-->DI = 1
FFF95      06           1FFCA     ;Move code "BA 06" to [DS
FFF94  BA       1FFCA             ;+ DI = M[4000(0)]
FFF93      05           1FFC9     ;
FFF92  C7       1FFC9        _____;
FFF91      00           1FFC8     ;Move 00 00 TO DI reg
```

```
FFF90  00       1FFC8             ;
FFF8F      BF          1FFC7_____;
FFF8E  D8       1FFC7             ;Move (AX) to DS reg
FFF8D      8E          1FFC6_____;
FFF8C  40       1FFC6             ;Load AX with 40 00
FFF8B      00          1FFC5     ;
FFF8A  B8       1FFC5        _____;
```

Load the address of the ISR in the location for external interrupt number 3 (0008C to 0008F) and initialize the stack segment and the stack pointer.

```
                             _____
FFF89      FF          1FFC4     ;Move FF FF to SP
FFF88  FF       1FFC4             ;
FFF87      BC          1FFC3_____;
FFF86  D0       1FFC3             ;Move (AX) to SS reg
FFF85      8E          1FFC2_____;
FFF84  50       1FFC2             ;Load AX with 50 00
FFF83      00          1FFC1     ;
FFF82  B8       1FFC1        _____;
FFF81      40          1FFC0     ;Move 40 00 to EA =
FFF80  00       1FFC0             ;[DS + DI] = [0 + 8E]
FFF7F      05          1FFBF     ;(CS = 4000(0), IP =
FFF7E  C7       1FFBF        _____;0000; ISR addr loaded)
FFF7D      00          1FFBE     ;Move 00 8E to DI reg
FFF7C  8E       1FFBE             ;
FFF7B      BF          1FFBD_____;
FFF7A  00       1FFBD             ;Move 00 00 to EA =
FFF79      00          1FFBC     ;[DS + DI] = [0 + 8C]
FFF78  05       1FFBC             ;(IP for ISR = 0000)
FFF77      C7          1FFBB_____;
FFF76  00       1FFBB             ;Move 00 8C to DI reg
FFF75      8C          1FFBA     ;(ISR addr will be at
FFF74  BF       1FFBA        _____;0008C through 0008F)
FFF73      D8          1FFB9     ;Move AX to DS register
FFF72  8E       1FFB9        _____;
FFF71      00          1FFB8     ;Load AX with 00 00
FFF70  00       1FFB8             ;
FFF6F      B8          1FFB7_____;
```

Initialize the 82C54 timer for counter 2, Mode 0, with an initial count of FF FF. This should be plenty of time for the ISR add~ess and the ISR itself above to be completely loaded before the timer times out.

```
                             _____
FFF6E  EF       1FFB7        _____;LSB, then MSB to cntr 2
FFF6D      FF          1FFB6     ;Load initial count, FF
FFF6C  FF       1FFB6             ;FF, into AX
FFF6B      B8          1FFB5_____;
```

```
FFF6A  02       1FFB5             ;Load DX with address for
FFF69      04           1FFB4     ;cntr 2 initial count (to
FFF68  BA       1FFB4        _____;come from AX)
FFF67      EE           1FFB3_____;CW to cntr 2
FFF66  B0       1FFB3             ;Load CW in AL = B0
FFF65      B0           1FFB2     ;(Mode 0,16 bit cnt,
                             _____;cntr 2)
FFF63  02       1FFB2             ;Load DX with "address"
FFF62      06           1FFB1     ;for counter 2 CW (to
FFF61  BA       1FFB1        _____;come from AL)
```

148 bytes

APPENDIX H

INTRPT CNTRLR TEST PROGRAM: INCREMENT 3

(Correct operation of this program could not be validated.)

In this iteration, as in the second iteration, when the timer times out, *OUT2* is used as a signal to the 8259 on *IR3* that there is an interrupt to be serviced. The 8259 alerts the CPU with the *INT* line. The CPU replies with *INTA** and the 8259 gives the CPU the address of where to look for the address of the interrupt service routine (ISR). (The ISR address and the ISR itself have to have been first loaded into memory.)

The difference of this iteration is that the ISR first puts out the byte, previously stored at memory location 40000, to the serial communications controller (SCC). Next, the ISR resets the timer, loops for a short while to give the timer more than enough time to go low, and then resets the 8259. When the timer times out, the ISR is run again. One sees the letter "A" on the SCC data pins every 79, 8254 clock cycles. Each 8254 clock cycle takes 16 system or CPU clock cycles (5MHz) to complete. Thus, one sees the "A" approximately every 79 x 16 x 200 nanoseconds = 252.8 microseconds. Actually, it takes a little bit longer than this due to the overhead time required to reset the timer.

The code is read from bottom to top.

```
8086                LO      HI
ADDR                BYTE    BYTE
                    EPROM   EPROM
                    ADDR    ADDR
                                 _____
FFFF4   FF          1FFFA             ;Jmp to FFF30
FFFF3       F0              1FFF9     ;
FFFF2   00          1FFF9             ;
FFFF1       30              1FFF8     ;
FFFF0   EA          1FFF8      _____; <---(Pwr on rst jmp)
                                 _____
FFFEF       FF              1FFF7     ;Jmp to FFFEB
FFFEE   F0          1FFF7             ;The CPU stays here until
FFFED       00              1FFF6     ;it's interrupted
FFFEC   EB          1FFF6             ;
FFFEB       EA              1FFF5_____;
                                 _____
FFFEA   FB          1FFF5      _____;Enable interrupts
```

Load the interrupt service routine. The ISR first puts out the byte from memory location 40000 to the SCC. Then it resets the timer and clears the highest priority ISR bit in the 8259 (there is only one in this case). Because the 8259 runs at 2.5 MHz and the timer only runs at .3125 MHz, there is a little delay loop to make sure the timer has gone low after being reset before the ISR bit is reset and interrupts are again enabled. The delay loop is almost certainly not required, but it doesn't hurt to have it.

```
                             _____
FFFE9      CF         1FFF4     ;Code CF-->return from
                                  ;ISR
FFFE8  FB      1FFF4             ;Code FB-->Enable
FFFE7      05         1FFF3     ;interrupts
FFFE6  C7      1FFF3        _____;
FFFE5      47         1FFF2_____;
FFFE4  47      1FFF2        _____;
FFFE3      EE         1FFF1     ;Code EE-->OUT to reset
                                  ;ISR bit
FFFE2  20      1FFF1             ;Code B0 20-->Load AL
FFFE1      05         1FFF0     ;with 20 ("20" for OCW2
FFFE0  C7      1FFF0             ;will reset highest
                             _____;priority ISR bit)
FFFDF      47         1FFEF_____;
FFFDE  47      1FFEF        _____;
FFFDD      B0         1FFEE     ;Code BA 00 01-->Load DX
FFFDC  01      1FFEE             ;with addr to write OCW2
FFFDB      05         1FFED     ;to 8259 to reset ISR bit
FFFDA  C7      1FFED             ;for IR3; (OUT2 has had
                                  ;plenty of time to go low
                                  ;because AX counted down
                             _____;from 8)
FFFD9      47         1FFEC_____;
FFFD8  47      1FFEC        _____;
FFFD7      00         1FFEB     ;
FFFD6  BA      1FFEB             ;
FFFD5      05         1FFEA     ;
FFFD4  C7      1FFEA        _____;
FFFD3      47         1FFE9_____;
FFFD2  47      1FFE9        _____;
FFFD1      F9         1FFE8     ;Code EB 49-->Jump back 7
FFFD0  EB      1FFE8             ;counts to opcode 48 to
FFFCF      05         1FFE7     ;decrement AX again
FFFCE  C7      1FFE7        _____;
FFFCD      47         1FFE6_____;
FFFCC  47      1FFE6        _____;
FFFCB      02         1FFE5     ;Code 74 02-->If AX was 0
FFFCA  74      1FFE5             ;then jump forward 2
FFFC9      05         1FFE4     ;counts to BA opcode;
```

```
FFFC7         47                  1FFE3______;
FFFC6   47          1FFE3              ______;
FFFC5         00                  1FFE2      ;Code 3C 00-->CMP AX with
FFFC4   3C          1FFE2                    ;0
FFFC3         05                  1FFE1      ;
FFFC2   C7          1FFE1              ______;
FFFC1         47                  1FFE0______;
FFFC0   47          1FFE0              ______;
FFFBF         48                  1FFDF      ;Code B0 08--Load AX with
FFFBE   08          1FFDF                    ;8
FFFBD         05                  1FFDE      ;Code 48-->Decrement AX
FFFBC   C7          1FFDE              ______;
FFFBB         47                  1FFDD______;
FFFBA   47          1FFDD              ______;
FFFB9         B0                  1FFDC      ;Code EF-->OUT with
FFFB8   EF          1FFDC                    ;initial count to
FFFB7         05                  1FFDB      ;cntr 2
FFFB6   C7          1FFDB              ______;
FFFB5         47                  1FFDA______;
FFFB4   47          1FFDA              ______;
FFFB3         00                  1FFD9      ;Code B8 2F 00-->load AX
FFFB2   4F          1FFD9                    ;with 00 4F as initial
FFFB1         05                  1FFD8      ;count
FFFB0   C7          1FFD8              ______;
FFFAF         47                  1FFD7______;
FFFAE   47          1FFD7              ______;
FFFAD         B8                  1FFD6      ;Code BA 04 02-->Load DX
FFFAC   02          1FFD6                    ;with addr to load the
FFFAB         05                  1FFD5      ;cntr 2 with the initial
FFFAA   C7          1FFD5                    ;word size count (from
                                       ______;AX)
FFFA9         47                  1FFD4______;
FFFA8   47          1FFD4              ______;
FFFA7         04                  1FFD3      ;
FFFA6   BA          1FFD3                    ;
FFFA5         05                  1FFD2      ;
FFFA4   C7          1FFD2              ______;
FFFA3         47                  1FFD1______;
FFFA2   47          1FFD1              ______;
FFFA1         EE                  1FFD0      ;Code B0 B0-->load AL
FFFA0   B0          1FFD0                    ;with cntr 2 CW, Mode 0,
FFF9F         05                  1FFCF      ;word size initial count
FFF9E   C7          1FFCF                    ;Code EE-->WR the CW B0
                                       ______;to AL)
FFF9D         47                  1FFCE______;
FFF9C   47          1FFCE              ______;
FFF9B         B0                  1FFCD      ;Code BA 06 02-->Load DX
FFF9A   02          1FFCD                    ;with addr for cntr 2 CW
FFF99         05                  1FFCC      ;(from AL)
FFF98   C7          1FFCC              ______;
FFF97         47                  1FFCB______;
```

```
FFF96  47       1FFCB            _____;
FFF95      06           1FFCA         ;
FFF94  BA       1FFCA                 ;
FFF93      05           1FFC9         ;
FFF92  C7       1FFC9            _____;
FFF91      47           1FFC8_____;
FFF90  47       1FFC8            _____;
FFF8F      EE           1FFC7         ;Code BA 02 00-->
FFF8E  00       1FFC7                 ;Load DX with 0200--
FFF8D      05           1FFC6         ;"addr" of the SCC port
FFF8C  C7       1FFC6            _____;Code EE-->(AL) to SCC
FFF8B      47           1FFC5_____;
FFF8A  47       1FFC5            _____;
FFF89      02           1FFC4         ;
FFF88  BA       1FFC4                 ;
FFF87      05           1FFC3         ;
FFF86  C7       1FFC3            _____;
FFF85      47           1FFC2_____;
FFF84  47       1FFC2            _____;
FFF83      05           1FFC1         ;Code 8A 05-->
FFF82  8A       1FFC1                 ;Move "A" at M[40000]
FFF81      05           1FFC0         ;to AL register
FFF80  C7       1FFC0            _____;
FFF7F      47           1FFBF_____;
FFF7E  47       1FFBF            _____;
FFF7D      00           1FFBE         ;Code BF 00 00-->
FFF7C  00       1FFBE                 ;Load DI with 0000
FFF7B      05           1FFBD         ;
FFF7A  C7       1FFBD            _____;
FFF79      47           1FFBC_____;
FFF78  47       1FFBC            _____;
FFF77      BF           1FFBB         ;Move D8 BF to 40005
FFF76  D8       1FFBB                 ;Code 8E D8-->Move (AX)
FFF75      05           1FFBA         ;to DS register
FFF74  C7       1FFBA            _____;
FFF73      47           1FFB9_____;Inc DI
FFF72  47       1FFB9            _____;Inc DI
FFF71      8E           1FFB8         ;Move code "40 8E" to
FFF70  40       1FFB8                 ;M[40003];
FFF6F      05           1FFB7         ;Code B8 00 40-->Load
FFF6E  C7       1FFB7            _____;AX with 4000
FFF6D      47           1FFB6_____;Inc DI-->DI = 3
FFF6C  47       1FFB6            _____;Inc DI-->DI = 2
FFF6B      00           1FFB5         ;Move code "B8 00" to
FFF6A  B8       1FFB5                 ;M[4000x4 + 1] = 40001
FFF69      05           1FFB4         ;
FFF68  C7       1FFB4                 ;
FFF67      00           1FFB3         ;Move 00 01 TO DI reg
FFF66  01       1FFB3                 ;
FFF65      BF           1FFB2_____;
FFF64  D8       1FFBF                 ;Move (AX) to DS reg
```

```
FFF63      8E           1FFB1_____;
FFF62  40       1FFB1             ;Load AX with 40 00
FFF61      00           1FFB0     ;
FFF60  B8       1FFB0        _____;
```

Load the address of the ISR in the location for external interrupt number 3 (0008C to 0008F) and initialize the stack segment and the stack pointer.

```
                             _____
FFF5F      FF           1FFAF     ;Move FF FF to SP
FFF5E  FF       1FFAF             ;
FFF5D      BC           1FFAE_____;
FFF5C  D0       1FFAE             ;Move (AX) to SS reg
FFF5B      8E           1FFAD_____;
FFF5A  50       1FFAD             ;Load AX with 50 00
FFF59      00           1FFAC     ;
FFF58  B8       1FFAC        _____;
FFF57      40           1FFAB     ;Move 40 00 to EA =
FFF56  00       1FFAB             ;[DSx4 + DI] = [0 + 8E]
FFF55      05           1FFAA     ;(CS = 4000(0), IP =
FFF54  C7       1FFAA        _____;0001; ISR addr loaded)
FFF53      00           1FFA9     ;Move 00 8E to DI reg
FFF52  8E       1FFA9             ;
FFF51      BF           1FFA8_____;
FFF50  00       1FFA8             ;Move 01 00 to EA =
FFF4F      01           1FFA7     ;[DSx4 + DI] = [0 + 8C]
FFF4E  05       1FFA7             ;(IP for ISR = 0001)
FFF4D      C7           1FFA6_____;
FFF4C  00       1FFA6             ;Move 00 8C to DI reg
FFF4B      8C           1FFA5     ;(ISR addr will be at
FFF4A  BF       1FFA5        _____;0008C through 0008F)
FFF49      D8           1FFA4     ;Move AX to DS register
FFF48  8E       1FFA4        _____;
FFF47      00           1FFA3     ;Load AX with 00 00
FFF46  00       1FFA3             ;
FFF45      B8           1FFA2_____;
```

Initialize the 82C54 timer for counter 2, Mode 0, with an initial count of FF FF. This should be plenty of time for the ISR address and the ISR itself above to be completely loaded before the timer times out the first time.

```
                             _____
FFF44  EF       1FFA2        _____;LSB, then MSB to cntr 2
FFF43      FF           1FFA1     ;Load initial count, FF
FFF42  FF       1FFA1             ;FF, into AX
FFF41      B8           1FFA0_____;
FFF40  02       1FFA0             ;Load DX with address for
FFF3F      04           1FF9F     ;cntr 2 initial count (to
FFF3E  BA      1FF9F        _____;come from AX)
FFF3D      EE          1FF9E_____;CW to cntr 2
FFF3C  B0      1FF9E             ;Load CW in AL = B0
FFF3B      B0          1FF9D     ;(Mode 0,16 bit cnt,
                            _____;cntr 2)
FFF3A  02      1FF9D             ;Load DX with "address"
FFF39      06          1FF9C     ;for counter 2 CW (to
FFF38  BA      1FF9C        _____;come from AL)
```

Load the character "A" into memory location [40000].

```
                            _____
FFF37      C1          1FF9B     ;Write "A" to [DSx4 + DI
FFF36  05      1FF9B             ;= 40000] ("A" =
FFF35      C6          1FF9A_____;11000001)
FFF34  D8      1FF9A             ;Move AX to DS
FFF33      8E          1FF99_____;
FFF32  40      1FF99             ;Move #4000 to AX
FFF31      00          1FF98     ;
FFF30  B8      1FF98        _____;
```

197 bytes

APPENDIX I

SERIAL COMMUNICATIONS CONTROLLER TEST PGM

First, memory is loaded with the arbitrary character "A." Then the SCC is initialized. The CPU then goes into a loop where it puts out the "A" from memory to the SCC Tx buffer and checks to see if the Tx buffer is empty. It keeps polling the Tx buffer until it is empty. Then the CPU fetches the "A" from memory again and the process repeats. For this test, the timer and the 8259 are not used.

The code is read from bottom to top.

```
8086                LO      HI
ADDR                BYTE    BYTE
                    EPROM   EPROM
                    ADDR    ADDR
                                   ______
FFFF4   FF          1FFFA                ;Jmp to FFFA5
FFFF3       F0              1FFF9        ;to begin program
FFFF2   00          1FFF9                ;
FFFF1       A5              1FFF8        ;
FFFF0   EA          1FFF8          ______; <---(Pwr on·rst jmp)
                                   ______
FFFEF       FF              1FFF7        ;Jmp to read character
FFFEE   F0          1FFF7                ;from memory again
FFFED       00              1FFF6        ;(Tx buffer empty)
FFFEC   CD          1FFF6                ;
FFFEB       EA              1FFF5______;
FFFEA   FF          1FFF5                ;Jump to read RR0 again;
FFFE9       F0              1FFF4        ;Tx buffer not yet empty
FFFE8   00 (CD)         1FFF4            ;(CD-->Put out the char
FFFE7       DB              1FFF3        ;from memory, anyway)
FFFE6   EA          1FFF3          ______;
FFFE5       05              1FFF2        ;If Tx buffer empty, skip
FFFE4   74          1FFF2          ______;next JMP statement
FFFE3       44              1FFF1        ;Compare AL with #44
FFFE2   3C          1FFF1          ______;AL = 44 if Tx empty
FFFE1       EC              1FFF0______;Read byte in RR0-->(AL)
FFFE0   EE          1FFF0                ;Tell WR0 what reg to
                                   ______;read
FFFDF       00              1FFEF        ;Load AL with 00->The reg
FFFDE   B0          1FFEF          ______;to be read is RR0
FFFDD       00              1FFEE        ;Load DX for SCC CS* and
FFFDC   00          1FFEE                ;to read (or write) a CW
FFFDB       BA              1FFED______;
```

The character at [40000] is put out to the SCC to be automatically transmitted at 9600 baud.

```
                             _____
FFFDA  EE       1FFED             ;OUT the data ("A") to
                             _____;the SCC
FFFD9      00           1FFEC     ;Load #0002 into DX for
FFFD8  02       1FFEC             ;SCC CS* and indicating
FFFD7      BA           1FFEB     ;data to come instead of
                             _____;CW
FFFD6  05       1FFEB             ;Move the "A" at [40000]
FFFD5      8A           1FFEA_____;to AL
FFFD4  00       1FFEA             ;Load DI with 00 00
FFFD3      00           1FFE9     ;
FFFD2  BF       1FFE9        _____;
FFFD1      D8           1FFE8     ;Move AX to DS reg
FFFD0  8E       1FFE8        _____;
FFFCF      00           1FFE7     ;Move #4000 to AX
FFFCE  40       1FFE7             ;
FFFCD      B8           1FFE6_____;
```

The following code intitializes the SCC registers for the proper baud rate and protocols.

```
                             _____
FFFCC  EE       1FFE6        _____;
FFFCB      03           1FFE5     ;Enable baud rate gen
FFFCA  B0       1FFE5        _____;
FFFC9      EE           1FFE4_____;
FFFC8  0E       1FFE4             ;WR14
FFFC7      B0           1FFE3     ;
FFFC6  EE       1FFE3        _____;
FFFC5      06           1FFE2     ;TC = 06
FFFC4  B0       1FFE2        _____;
FFFC3      EE           1FFE1_____;
FFFC2  0C       1FFE1             ;WR12
FFFC1      B0           1FFE0_____;
FFFC0  EE       1FFE0        _____;
FFFBF      16           1FFDF     ;TRxC will have baud
FFFBE  B0       1FFDF        _____;rate out
FFFBD      EE           1FFDE_____;
FFFBC  0B       1FFDE             ;WR11
FFFBB      D0           1FFDD_____;
FFFBA  EE       1FFDD        _____;Write WR5 with (AL)
FFFB9      68           1FFDC     ;Load AL with 68
FFFB8  B0       1FFDC        _____;8 bits/char
                                  ;Tell WR0 that WR5 is
FFFB7      EE           1FFDB_____;next
FFFB6  05       1FFDB             ;Load AL with 05
FFFB5      B0           1FFDA_____;
```

```
FFFB4  EE       1FFDA        _____;Write WR4 with 4C (fm
FFFB3      4C           1FFD9     ;AL); Load AL with 4C-
FFFB2  B0       1FFD9        _____;x16 Mode,2 stop bits
FFFB1      EE           1FFD8_____;Load WR0 with 04 (fm AL)
FFFB0  04       1FFD8             ;Load AL with 04--will
FFFAF      B0           1FFD7_____;write Write Reg 4 (WR4)
FFFAE  00       1FFD7             ;Load DX for CS* of the
FFFAD      00           1FFD6     ;SCC to write a control
FFFAC  BA       1FFD6             ;word (CW) to Write Reg 0
                                  ;to tell the SCC which
                                  ;register CW will be
                             _____;written next
```

Load the character "A" into memory location [40000].

```
                             _____
FFFAB      C1           1FFD5     ;Write "A" to [DS + DI =
FFFAA  05       1FFD5             ;40000]
FFFA9      C6           1FFD4_____;
FFFA8  D8       1FFD4             ;Move AX to DS
FFFA7      8E           1FFD3_____;
FFFA6  40       1FFD3             ;Move #4000 to AX
FFFA5      00           1FFD2     ;
FFFA4  B8       1FFD2        _____;
```

81 bytes

APPENDIX J

LIST OF COMPONENTS

2	AM27C010--128K x 8 bit EPROM
8	MSM8128S-12--128K x 8 bit SRAM
3	CD54HC573F3A--Address latch (Octal D type latch)
2	CD54HC245F3--Data transceiver (Octal tri-state)
3	CD54138F3A--3 to 8 Decoder
1	MC74HC32N--quad 2 input OR gate
1	Z85C30--SCC serial communications controller
1	MD82C59A-5/B--Programmable interrupt controller
1	MD80C86-2/B--16 bit microprocessor
1	MD82C85/B--Static clock controller/generator
1	Xtal CTS MP150 15.000--15 MHz crystal (Have also used 4.0 MHz xtal)
1	MC7805C--5 volt voltage regulator
1	MM54HC161J/883C--Synchronous binary counter with asynchronous clear
1	M82C53-5--Programmable interval timer
1	MC1488--quad line driver (+, - 9v)
1	MC1489A--quad line receiver(+5v)
1	82C55--programmable Peripheral interface
1	DB 25 P RS-232 connector Pin 2-Trans data Tx output Pin 3-Rcv data Rx input Pin 7-Sig Gnd Gnd common
1	SPDT Pushbutton switch

LIST OF REFERENCES

1. Hiser, James K., *Design of a Reliable Computing System for the Petite Amateur Navy Satellite (PANSAT)*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1989.

2. Sakoda, Daniel, Noble, Mike, and Paluszek, Steve(sic), "Preliminary Design of the Naval Postgraduate School Petite Amateur Navy Satellite (PANSAT) Electric Power and Communications Subsystems," briefing paper for the 4th Annual AIAA/Utah State University Conference on Small Satellites, August 28-31 1990.

3. DD Form 1721, Experiment Title: *Petite Amateur Navy Satellite*, Experiment No.: NPS-901, prepared 890130.

4. DD Form 1721, Experiment Title: *Petite Amateur Navy Satellite*, prepared 901210, Block 8b--Hardware Ready Flight Date; and Block 8c--Duration, Revision of DD Form 1721 with same title prepared 890130.

5. Sakoda, Daniel, "Petite Amateur Navy Satellite (PANSAT) A Proposal to the Defense Advance Research Projects Agency," proposal for research, by the Space Systems Academic group, Naval Postgraduate School, Monterey, California, October 1990.

6. Paluszek, Stephen E., *Spread Spectrum Communications for the Petite Amateur Navy Satellite (PANSAT)*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1990.

7. Couch, Leon W., *Digital and Analog Communication Systems*, 3rd Ed., Macmillan Publishing Company, 1990.

8. Noble, Michael A., *A Preliminary Design of the Electrical Power System for the Naval Postgraduate School's PANSAT*, Master's Thesis, Naval Postgraduate School, Monterey, California 1990.

9. *Get Away Special Small Self-Contained Payloads Experimenters Handbook*, National Aeronautics and Space Administration, 1984

10. Davidoff, Martin R., *The Satellite Experimenter's Handbook*, American Radio Relay League, Newington, Connecticut, 1985.

11. Siewiorek, Daniel P., and Swarz, Robert S., *The Theory and Practice of Reliable System Design*, Digital Equipment Corporation, 1982.

12. Wakerly, John F., *Digital Design Principles and Practices*, Prentice Hall, Englewood Cliffs, New Jersey, 1990.

13. *1988 Digital Product Data Book*, Harris Corporation, 1989.

14. *Reliability Handbook*, National Semiconductor Corporation, Santa Clara, California, 1987.

15. Fox, Terry L., *AX.25 Amateur Packet-Radio Link-Layer Protocol, Version 2.0*, American Radio Relay League, Newington, Connecticut, 1984.

16. Ward, J.W., and Price, H. E., "The UoSAT-2 Digital Communications Experiment," *Journal of the Institution of Electronic and Radio Engineers,(Australia)*, Vol. 57, No. 5(Supplement), pp. S163-S173, September, October 1987.

17. Bean, N.P., and others, "The UoSAT-C,D & E Technology Demonstration Satellites," *Proceedings of the Second Annual AIAA/Utah State University Conference on Small Satellites*, Utah State University, Logan, Utah, 1988.

18. Ohara, Moriyoshi, Masaya Fukasawa, and Youichi Kikukawa, "The FO-12 Mailbox System," *Proceedings of the AMSAT-NA Fifth Space Symposium and Annual Meeting*, pp. 57-61, American Radio Relay League, Newington, Connecticut, 1987.

19. Conversation between David Rigmaiden, Electronics technician and laboratory manager for Space Systems Academic group, NPS, and the Author, September 1990.

20. Horowitz, Paul, *The Art of Electronics*, 2nd Ed., Cambridge University Press, 1989.

21. Buchanan, James E., *CMOS/TTL Digital Systems Design*, McGraw Hill Publishing Company, 1990.

22. Lenk, John D., *Logic Designer's Manual*, Reston Publishing Company, 1977.

23. Conversation between David Rigmaiden, Electronics technician and laboratory manager for Space Systems Academic group, NPS, and the Author, October 1990.

24. Wakeman, Larry, "National's Process Enhancements Eliminate the CMOS SCR Latch-Up Problem in 54HC/74HC Logic," Application Note 339, *CMOS LOGIC DATABOOK*, National Semiconductor Corporation, pp. 2-112--2-119, 1988.

25. *RAD-HARD/HI-REL CICD DATA BOOK*, Harris Corporation, Custom Integrated Circuits Division, Revised June 1987.

26. *Z8000 Family Data Book*, Zilog, Inc., Campbell, California, pp. 222-252, November 1988.

27. Telephone Conversation between Mike Lewis, Zilog Corp. field service representative, and the Author, 5 December 1990.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   Cameron Station
   Alexandria, Virginia 22304-6145

2. Library, Code 52 2
   Naval Postgraduate School
   Monterey, California 93943-5002

3. Commander, 1101st Signal Brigade 1
   Bldg. 48, Fort McNair
   Washington, D.C. 20319-5050

4. LT John Quint 1
   SMC #1473
   Naval Postgraduate School
   Monterey, California 93943

5. Kadri Hekimogln 1
   DeGol Cadessi No = 1/5
   Tandogan
   Ankara, Turkey

6. CPT Stephen M. Tobin 3

7. Professor Mitchell L. Cotton, Code EC/Cc 2
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943

8. Professor Chin-Hwa Lee, Code EC/Le 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943

9. Chairman, Code EC 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5000

10. Chairman, Code SP 1
    Space Systems Academic Group
    Naval Postgraduate School
    Monterey, California 93943-5000