HERCULES ATTITUDE PROCESSOR (HAP)

by

Robert Francis Higgins Jr.

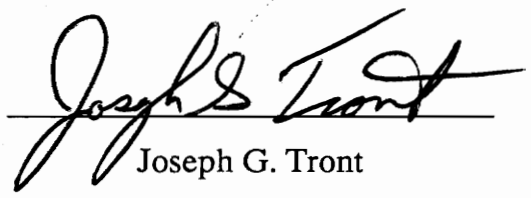
Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

APPROVED:



Joseph G. Tront

George E. Flach

Fred J. Ricci

May, 1992
Blacksburg, VA

HERCULES ATTITUDE PROCESSOR (HAP)

by

Robert Francis Higgins Jr.

Committee Chairman: Joseph G. Tront
Electrical Engineering

(ABSTRACT)

The design and analysis of a microprocessor-based gyro attitude data processing system used to geolocate natural phenomena from space was performed. Operational software was written and a HERCULES Attitude Processor (HAP) unit was built and tested.

Strict adherence to worst-case timing design criterion was a prime hardware design consideration. Weight, volume, and power requirements were also addressed. Redundancy was included for critical time maintenance functions. Hardware performance and accuracy was calculated and measured.

Operational software was written to control the functions of the HAP unit. Algorithms were written to accurately process the high speed gyro attitude data. Data communication between subsystems in the HERCULES system was controlled by the software. Subsystem configuration, operating modes, self-testing, and resource management was performed by the operational software.

Testing was performed on the HAP unit and operational software. Hardware and software performance was analyzed and is presented.

Acknowledgement

The HERCULES program was financed through U.S. Navy and U.S. Army finding. I would like to thank Dr. Howard for the opportunity to work on such an exciting project. I would like to thank the other members of the HERCULES program team for their support and hard work on the HERCULES system.

I would like to thank Dr. Tront and Dr. Ricci for their assistance and patience during the time I have spent on this thesis. I would like to thank George Flach, my technical mentor. Finally I would like to thank my family and especially my wife Kathi for their support, encouragement, and understanding during my masters studies.

TABLE OF CONTENTS

1.0 INTRODUCTION

The perspective from space allows a unique observation opportunity to study geological, oceanographic and environmental phenomenon. Landsat, a geological observation spacecraft [30], and other spacecraft are in orbit to take advantage of this viewpoint. These systems are tasked days or months in advance to observe specific Earth-based features. Most of the imagery is plagued by cloud cover or haze. The spacecraft will take the image regardless of the quality of the observation. Alternatively, an astronaut has the ability to recognize interesting phenomenon when the opportunity presents itself. Pictures are taken of the phenomenon for later analysis.

During the Mercury, Gemini, and Apollo programs astronauts took thousands of photographs using 70-mm Hasselblad film cameras [30,31]. Skylab also used the Hasselblad film cameras, but also used specially designed cameras for Earth observations.

Recently, astronauts have access to this unique observation viewpoint using the Space Transportation System (STS). STS missions are less than 14 days, and occur only several times a year. Efficient use of observation time is important during the mission.

Hundreds of photographs are taken out of any of the ten small windows on the flight deck during a typical mission. Cataloging and quantifying the photo-

graphs is an important part of the analysis process. The time and the location of the photograph on the Earth must be known to create the catalog.

Audio cassette tape recorders are used to record notes about the photographs that are taken. These notes provide general information about what has been photographed. Points of interest in the photograph are verbally recorded to capture the motivation for the picture. Time of day is coarsely recorded verbally on the cassette tape recorder. The approximate position of the photograph on the Earth is also verbally recorded.

If multiple photographs of the same phenomenon are taken over time, movement can be detected. Differences between photographs are used to determine the direction of the movement and the speed of the movement. Accurate time is needed to satisfactorily quantify the speed of the movement. Cloud growth, ocean currents and many other phenomenon are studied in this way.

Accurate determination of the latitude and longitude of the developed photographs is difficult. Distinguishing geology or landmarks in the photograph are used to identify the photograph's location. If there are no landmarks, the photograph cannot be readily associated with a latitude and longitude on the Earth. Sometimes the pictures can not even be associated with a continent. More than 71% of the Earth is covered by water, and there are few landmarks in the middle of an ocean. Locating images of oceanographic phenomenon is almost impossible. When a high power lens is used, the field of view is reduced, and the chance of recognizing a landmark is unlikely.

Development of the photographs takes place on the ground, so there is no method for the astronaut to view the pictures on orbit. If the pictures are not turning out properly, there is no way of knowing that on orbit. The record of the

phenomenon can be lost, and the opportunity to record the event is wasted.

An effort is underway to enhance the usefulness of the photographic observations made from the STS. The Naval Space Command and the Space and Strategic Technologies Office for Research, Development and Acquisition for the U.S. Army have been actively sponsoring work in this area.

Several objectives have been defined. Photographs must be geolocated on the Earth. That is, the latitude and longitude of the photograph must be known. The time of the photograph must also be known. On orbit viewing of images, capturing of images and storing of images in a digital format, are highly desirable features. Real-time transmission of images to the ground is also desirable.

In 1985, the National Aeronautics and Space Administration (NASA) Johnson Space Center (JSC) began the development of the Latitude and Longitude Locator (L-cubed) system. The intent of this system was to record the time, the latitude and longitude of the photographs that were taken by the astronauts aboard the STS. The system, described in Chapter 2, was flown aboard the STS twice, and was successful both times [1].

However, the L-cubed system had several undesirable features. A second generation L-cubed system was needed to enhance the performance and the ease of use of the system. This system is called HERCULES (Hand held, Earth oriented, Real-time, Cooperative, User friendly, Location, Targeting, and Environmental System). Time of the photographs and the latitude and longitude of the photographs will be determined in real-time and stored with the image. Images will be captured by an electronic camera which digitally records and stores the images. Images can be viewed on the STS and can be transmitted directly to the ground. The camera is being developed by NASA JSC. The latitude, longitude, and time determination components of the HERCULES system are being developed by the

Naval Center for Space Technology (NCST) at the Naval Research Laboratory (NRL). Integration and system testing of the entire HERCULES system is being performed by the NCST.

This thesis describes the work that I have done in the development and testing of the HERCULES Attitude Processor (HAP). I have designed and analyzed the electrical hardware and I have written the software to control the functions of the HAP unit.

Chapter 2 gives a description of the L-cubed system which is the predecessor to HERCULES. Chapter 3 provides an introduction to the HERCULES system and to HAP. Chapter 4 provides the hardware and software design requirements for HAP as well as the theory of operation for the attitude algorithms. HAP hardware that I have developed is described in Chapter 5. HAP software that I have developed is described in Chapter 6. Chapter 7 presents my analysis, performance and test data for the HAP. Areas requiring further work and study and the conclusion are provided in Chapter 8. Appendices are included at the end for schematics, hardware details, and software listings. A bibliography and list of acronyms is also provided.

2.0 BACKGROUND

In 1985, NASA developed a system that determined the latitude, longitude and time of pictures taken from the STS. This system was called the Latitude Longitude Locator (L-cubed) [2]. L-cubed was flown aboard the STS in August 1989 and again in January 1990.

The L-cubed system, depicted in Figure 2.1, consisted of three hardware subsystems: a modified Hasselblad 70mm camera, a Camera-Computer Interface Unit (CCIU), and a Graphic Retrieval and Information Display (GRiD) computer.

Photographs were taken using a modified Hasselblad 70mm film camera. A custom designed viewfinder contained both a fixed reticle and a movable reticle. The fixed reticle was aligned with the horizon and the vertical plane, while the movable reticle was aligned on the point of interest. Measurement of the angle between these two reticles was accomplished using a rotary encoder. When the camera shutter button was depressed, angle data was sent to the CCIU for processing.

When a photograph or data point was taken, angle and time data was formatted by the CCIU and sent to the GRiD computer. Time was tracked, in the CCIU, using a battery backed up clock that was set before launch. The CCIU and GRiD computer communicate using a RS232C port.

A GRiD computer was used as the input device for the system. Latitude,

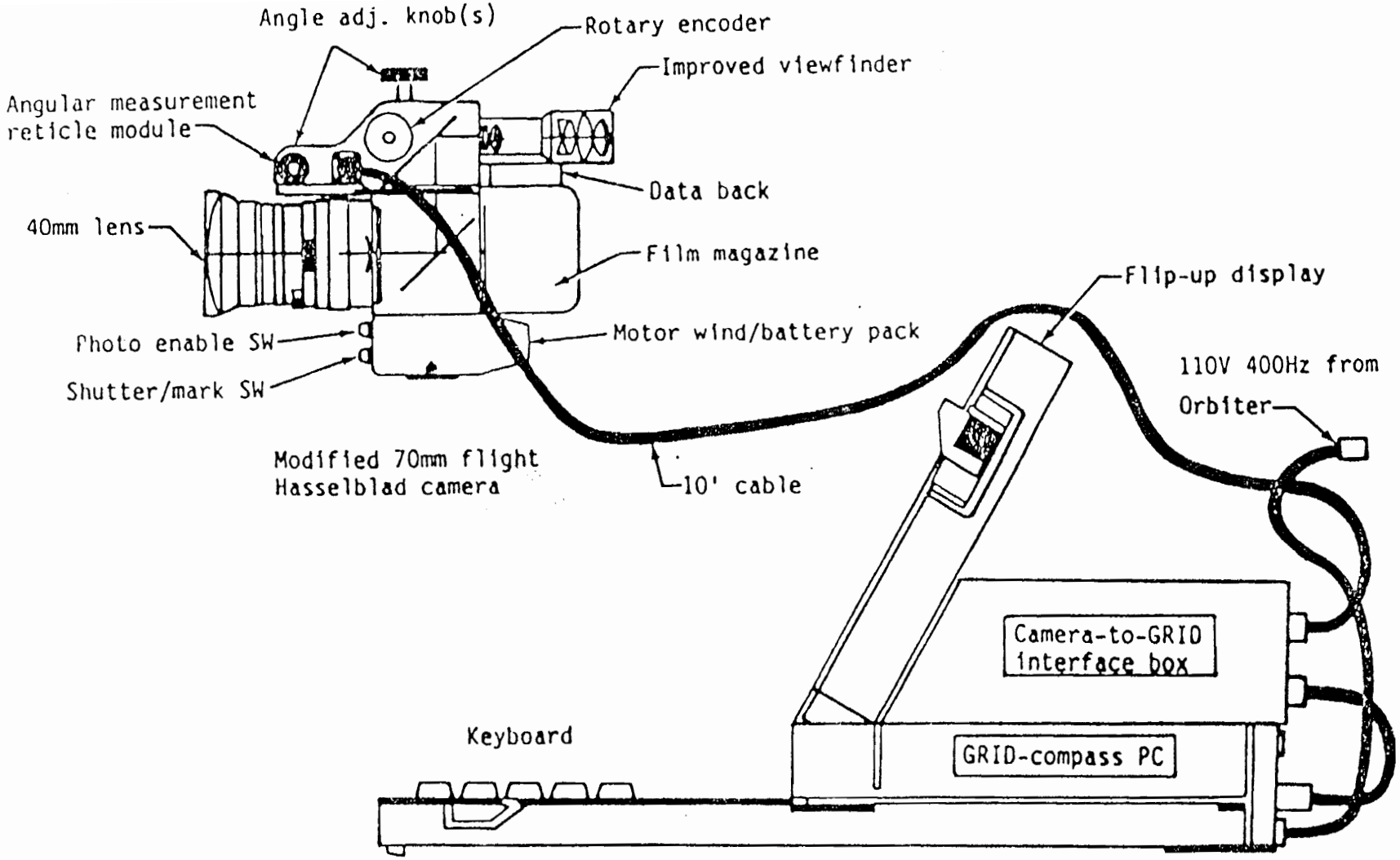


Figure 2.1 - L-cubed Hardware Configuration
Reprinted from [2]

longitude, and time of the photograph was calculated in the GRiD computer and stored on the GRiD computer's hard disk.

The L-cubed system functioned and was used as described below. Stored in a STS middeck locker for launch, the L-cubed was unpacked and assembled on the STS flight deck. STS position was determined by the GRiD computer, after the initial STS ephemeris was entered. The location of the point of interest, north or south of the ground track, was entered into the GRiD.

Between 2 and 18 data points were taken by the astronaut, for each point of interest. To take a data point, the astronaut pointed the camera at the point of interest and lined up the two reticles in the viewfinder. One reticle was aligned with the horizon, and the other reticle was lined up with the point of interest. When the shutter switch was pressed, the time and reticle angle data point was sent from the CCIU to the GRiD computer. Film was conserved using a shutter enable/disable switch. Data point time was used by the GRiD computer to propagate the ephemeris of the STS to the time of the photograph. Using the updated ephemeris, the altitude of the STS was determined and the distance from the STS ground track to the visible horizon was calculated. The visible horizon appeared as a circle on the Earth. The angle between the two reticles was used to calculate the distance from the STS ground track to the point of interest. This appeared as a concentric target circle on the Earth. A diagram detailing the configuration is shown in Figure 2.2. The point of interest was located at one of the two intersections of the target circles. The astronaut input, that identified the point of interest as north or south of the ground track, identified to the GRiD computer which intersection was the location of the point of interest. If more data points were taken, the location of the picture could be determined more accurately.

Several shortcomings were realized while using the L-cubed system.

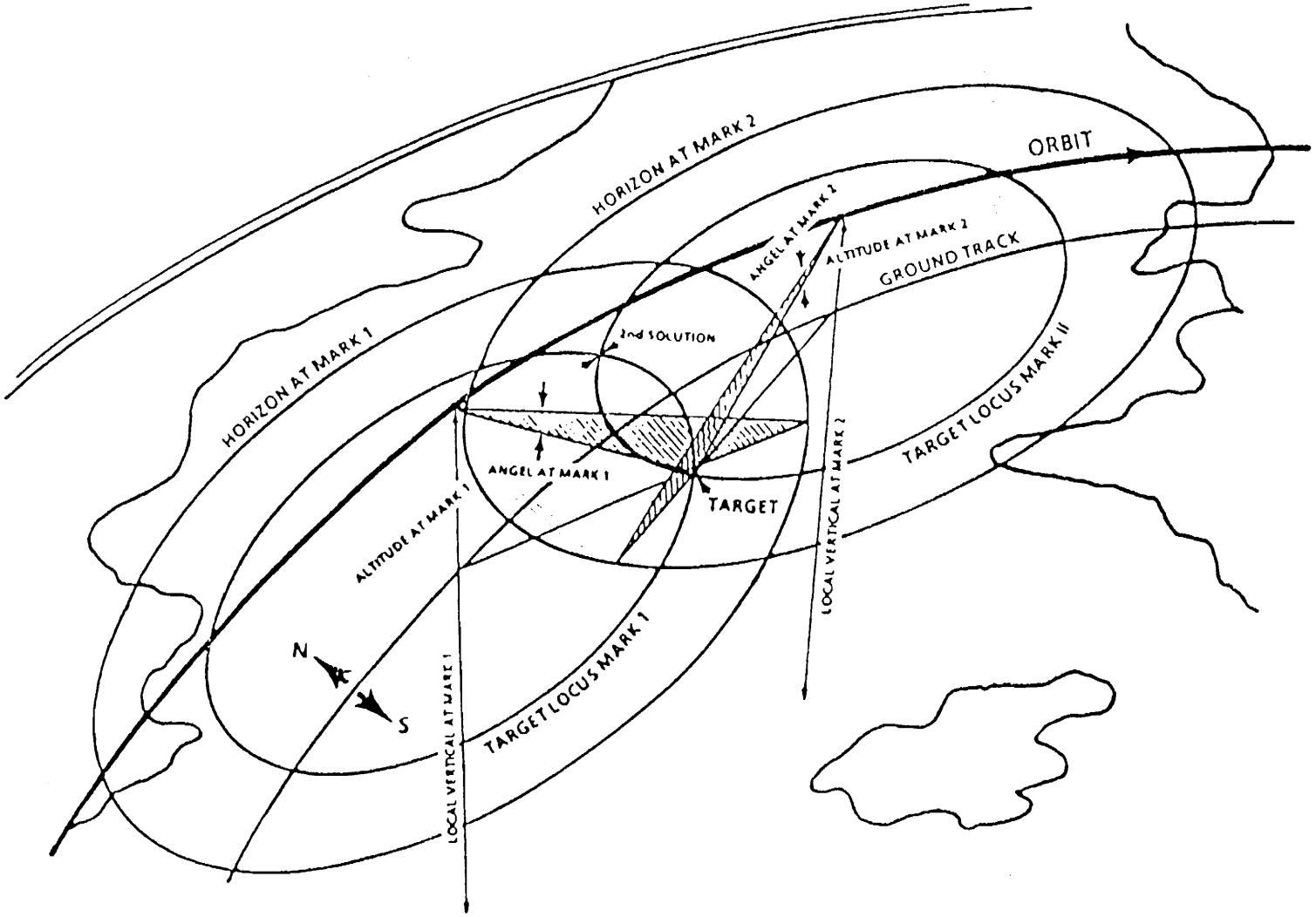


Figure 2.2 - L-cubed Operations Overview
Reprinted from [2]

L-cubed was awkward to point and difficult to use. It required that the horizon and the point of interest to be simultaneously visible. Astronauts found it difficult to align the two reticles simultaneously and accurately, especially when the point of interest was far from the horizon. It is hard to look two places simultaneously, and errors were induced because of this difficulty. The Earth horizon appears as a hazy region above the Earth surface, there is not a distinct edge. A judgment needed to be made by the astronaut, as to exactly where the horizon was. Errors were accumulated because the judgment needed to be consistent. Some of the astronauts were consistently able to judge the position of the horizon, while others were not.

Known landmarks were used as points of interest, so the L-cubed could be tested on orbit. The L-cubed was designed to locate the pictures in real-time with median accuracies of about 6 nautical miles. Results obtained in real-time aboard the STS provided location accuracies of between 5 and 300 nautical miles. After the data was returned to Earth, additional processing of the data provided location accuracies of between 1.5 and 30 nautical miles.

Maximum magnification was also limited by the L-cubed system. Earth horizon and the point of interest needed to be simultaneously visible, limiting the available magnification of the photograph to about one-half power. Higher power lenses are required to capture the desired phenomenon adequately.

Film was used to store the images in the L-cubed system. Pictures could not be reviewed in the STS for quality, or be transmitted to the Earth.

A second generation L-cubed system was needed to enhance the ease of use and performance of the system and to add the missing functionality.

3.0 HERCULES OVERVIEW

As previously stated, the second generation of L-cubed is called HERCULES. The HERCULES system will capture and digitally store the images. Images can be viewed on the STS and can be transmitted to the ground. Time of the photographs and the latitude and longitude of the photographs will be determined in real-time and stored with the image.

The HERCULES system, shown in Figure 3.1, consists of five hardware subsystems: an Electronic Still Camera (ESC), an Electronic Still Camera Electronics Box (ESCEB), an Inertial Measurement Unit (IMU) and associated Inertial Attitude Electronics (IAE), my thesis topic: a HERCULES Attitude Processor (HAP), and a Payload and General Support Computer (PGSC). Figure 3.2 shows an electrical block diagram of the HERCULES system. Basic components of the system are described below.

3.0.1 ELECTRONIC STILL CAMERA

The ESC is a Nikon F4 camera that has been modified to allow the addition of a Charge Coupled Device (CCD) image sensor in place of the film at the focal plane. CCD array, digitizers, and CCD controls are contained in a custom designed back for the ESC [3].

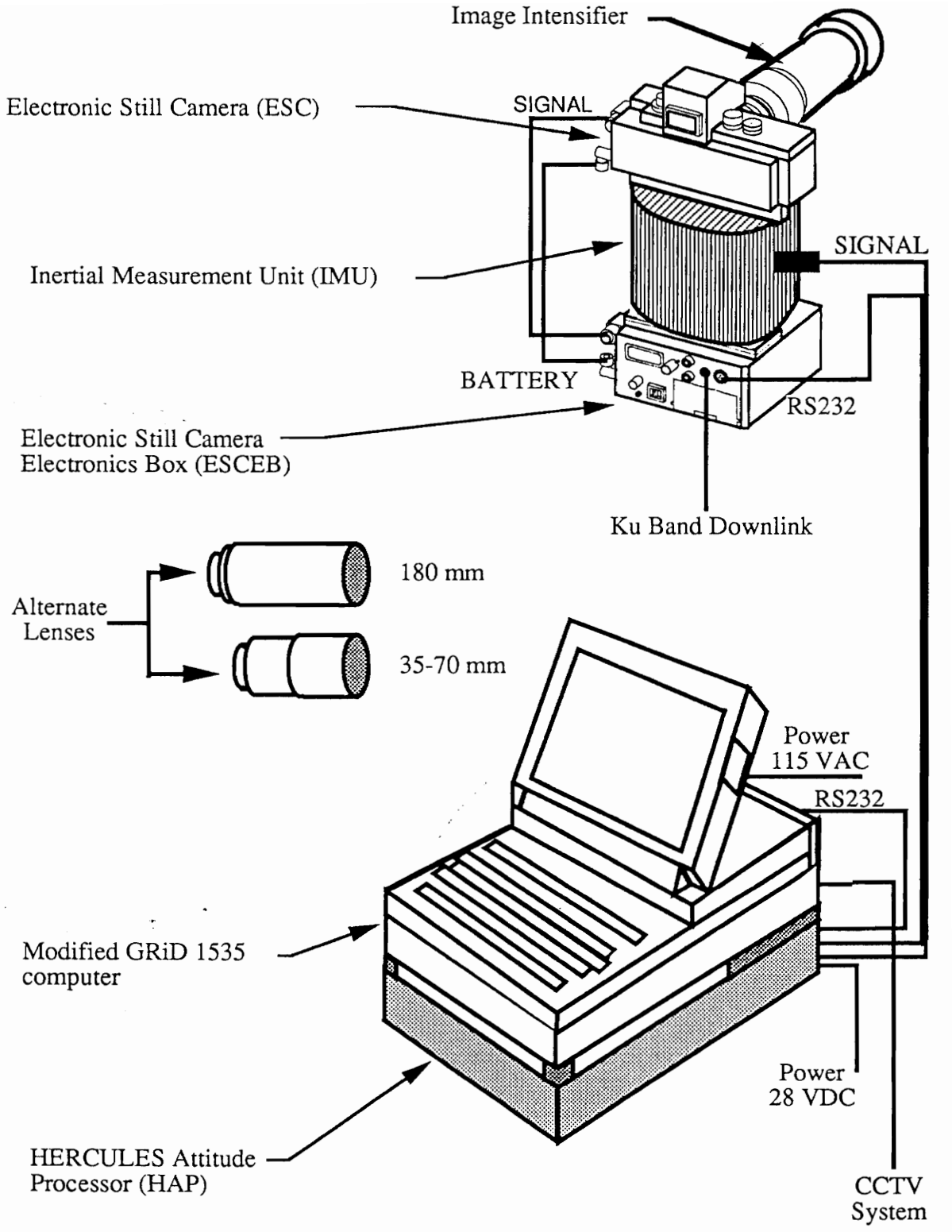


Figure 3.1 - HERCULES System Configuration

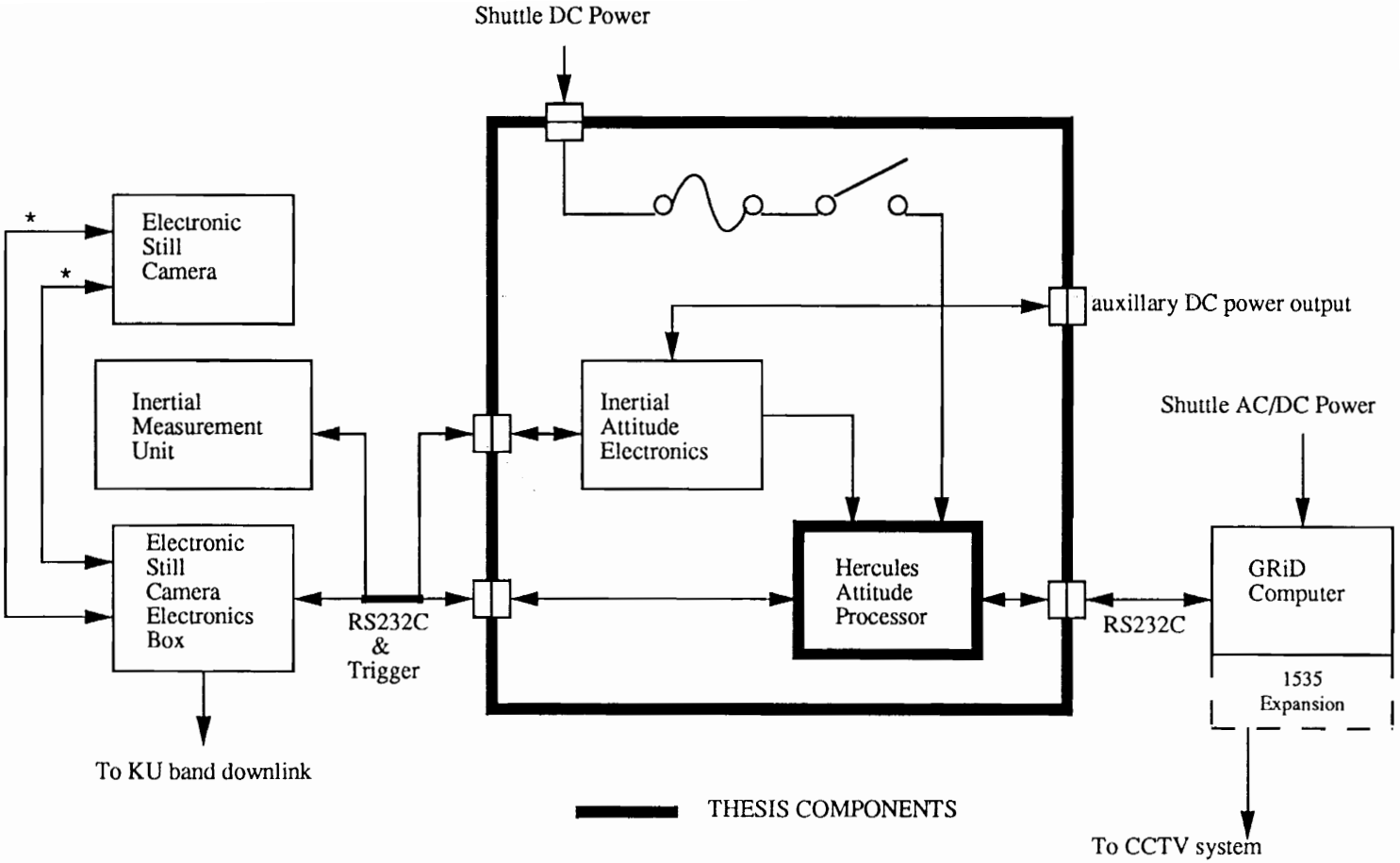


Figure 3.2 - Hercules System Block Diagram

3.0.2 ELECTRONIC STILL CAMERA ELECTRONICS BOX

ESC functions are controlled and monitored by the ESCEB. An IBM-PC compatible Wildcard computer is housed in the ESCEB. A mode control switch on the ESCEB determines which of the four modes the ESC and ESCEB will operate in. In mode one, the ESC will capture the image, the ESCEB will store the image on the hard disk, but the ESCEB will not downlink the image. In mode two, the ESC will capture an image, the ESCEB will store the image on the removable hard disk, and the ESCEB will downlink the image using the STS Ku band downlink. In mode three, the ESCEB will downlink the images that have been selected using the PGSC image processing software. Images will not be stored on the hard disk in mode three. In mode four, the ESC will capture an image, and the ESCEB will downlink the image. Images are not stored on the hard disk in mode four. Mode four can be used to save hard disk space when practicing star sightings or camera usage. In all four modes, an ESC shutter trigger pulse is sent through the ESCEB to HAP when an image is taken [4]. Thirty nine images can be stored on the removable hard disk. A full hard disk can be easily replaced with an empty hard disk. Hard disks are removed and installed using a thumbscrew mounted to the rear of the hard disk.

3.0.3 INERTIAL ATTITUDE UNIT and INERTIAL ATTITUDE ELECTRONICS

The IMU, manufactured by Honeywell, is directly mounted to the ESC. Internal to the IMU are three Helium-Neon (HeNe) ring laser gyros mounted orthogonal to each other. Inertial angular rates of change of three mutually orthogonal axes, are measured by the IMU. Necessary control electronics and interfaces for the IMU are provided by the Honeywell developed IAE. The IAE unit is

housed within the HAP enclosure. With the IMU attached directly to the ESC, the attitude of the ESC can be determined. IMU initialization is described in Chapter 3.1.

3.0.4 HERCULES ATTITUDE PROCESSOR

The HAP is electrically connected to the ESCEB, the IAE, and the PGSC. ESC attitude is propagated by HAP and a stable time source is included within HAP. When an ESC shutter trigger pulse is detected by HAP, the PGSC is sent a packet of information containing the time, the ESC attitude, and the status of the HAP at the time of the trigger pulse. HAP also sends packets of data, containing time and ESC attitude, to the PGSC at regular intervals.

When the HAP is powered on, it initializes the IMU and the HAP hardware. Once initialization is complete, the HAP reads, filters, compensates, and propagates the IAE data and reports the data to the PGSC. Propagation is another term for advancing the state of a system model from one time step to another. Attitude propagation means that the model of the attitude is being updated with rate data from the gyro. Packets of data are sent to the PGSC from HAP two times per second. This packet of data contains the ESC attitude, the time of that attitude sample, and the status of HAP, ESC, and IMU. ESC attitude is initialized, in software, to a predetermined attitude. This predetermined attitude needs to be related to the operational attitude through an on orbit alignment process that will be described later. IAE angle data is presented to HAP once every 0.5 milliseconds. Every set of angle data needs to be captured and processed to maintain knowledge of the ESC attitude. Angular rate of change, not attitude, is determined by the IMU and IAE. HAP needs to integrate every one of the angle data sets to

maintain knowledge of the ESC attitude. Angle data is propagated by HAP for as long as HAP is powered on, regardless of other operations that are occurring in HAP.

3.0.5 PAYLOAD AND GENERAL SUPPORT COMPUTER

The PGSC is a modified version of the commercially available GRiD 1530 portable IBM-AT compatible computer. NASA JSC modified the GRiD 1530 to be used as a general computer to support the astronauts. STS missions fly two PGSCs for general mission support on every mission. Additional PGSCs are provided to experimental payloads when a general computer is needed. HERCULES has taken advantage of the NASA JSC provided PGSC. Features of the PGSC include: an INTEL 80386 processor with the INTEL 80387 floating point co-processor, 4 Mbytes of Random Access Memory (RAM), a 1.44 Mbyte 3.5" floppy disk, a 40 Mbyte hard disk, +28 volt power supply, forced air cooling, and an electro-luminescent display [4]. Astronaut interface to the HERCULES software is through the PGSC.

When the PGSC is powered on, the HERCULES software is loaded from the 3.5" floppy disk. A menu system is used by the HERCULES system to facilitate parameter input and system control. HERCULES software provides a real-time display of pertinent information. Display information includes: current position of the STS, HERCULES status, time, and geolocation information. Information is updated every time a new value is calculated. Real-time display of this information allows the identification of predetermined points of interest, where the latitude and longitude is known, but the apparent position is not.

When the PGSC receives a packet of information from HAP, several operations are performed. Time contained within the HAP packet is used to propagate

the STS ephemeris forward to the current time. Latitude and longitude of the intersection of the boresight of the ESC and the Earth is calculated using the ESC attitude contained in the HAP packet. If the packet of information from HAP is associated with an image, the PGSC sends the image time and location information through HAP to be stored in the ESCEB along with the image. Regardless of the ESCEB mode control switch position, the PGSC attempts to send the image time and location information to the ESCEB.

An expansion chassis can be added to the PGSC. When this expansion chassis is added, two NASA JSC custom circuit cards can be installed. One of the cards interfaces to the ESCEB hard disk. The other card interfaces with the STS Closed Captioned Television System (CCTV). Using image processing software developed at NASA JSC, the images can be viewed on the CCTV system or can be marked for downlink on the KU band downlink. All images are downlinked using the ESCEB.

3.1 HERCULES OPERATIONAL DESCRIPTION

HERCULES is stored in a middeck locker during launch of the STS. After launch, HERCULES is removed from the locker and assembled on the STS flight deck. The STS middeck lockers are not large enough to hold the assembled HERCULES system. All HERCULES mechanical mounting interfaces are designed to facilitate assembly without the use of tools. Using thumbscrews, the IMU is mounted to the ESC and the ESCEB. The HAP is placed below the PGSC. Velcro is used to attach the HAP and the PGSC to a shelf on the flight deck. Power for HERCULES is provided by one of the three STS DC power connectors on the flight deck.

3.1.1 INITIALIZATION

After assembly, the astronaut must initialize the HERCULES system. When HERCULES is initially turned on, the orbital position of the STS is unknown, and the attitude of the ESC in relation to the earth is unknown.

Orbital position of the STS is determined by the STS mission control center on the Earth and faxed to the STS astronaut. The astronaut will then take this orbital position or state vector, and its associated time tag, and input it into the PGSC using the HERCULES menu system. Time, contained within the HAP data packet, is used by the PGSC to propagate the state vector from the initial state vector time tag to the current time received from the HAP. Every time the PGSC receives a packet of data from the HAP, the PGSC propagates the state vector to that time. HERCULES software in the PGSC will continue to track and update the orbital position of the STS with every data packet it receives from HAP. The state vector propagator is a 6 element state space representation of the STS orbit with an Earth entered inertial frame of reference [5].

Attitude of the ESC is initialized, on orbit, using a known star. The stars that can be used are contained in the U.S. Naval Observatory's FK5 star catalog. HERCULES software executing on the PGSC contains position information for selected FK5 stars. These selected stars are numbered and their positions are known by the astronauts.

Attitude initialization can begin after the astronaut has entered the STS state vector. Using the HERCULES menu system, the astronaut inputs, into the PGSC, the star number for the first star that will be used for the initialization. ESCEB mode 1 is selected using the ESCEB mode control switch. The astronaut then sights the chosen star through the ESC viewfinder. When the crosshairs are directly over the star, the astronaut will press the ESC shutter trigger button. ESCEB image storage begins after the ESC shutter has closed. HAP detects the shutter trigger pulse and sends the PGSC a trigger data packet. Trigger data packet information is in the same format as the regular data packet. The trigger data packet sets a flag to indicate that the data in the packet is associated with an image. Trigger data packet information represents the time and attitude of the ESC at the time of the shutter trigger pulse. HERCULES software receives the trigger packet and uses the time to propagate the STS state vector to the shutter trigger time. The same star is sighted again, with the ESC rotated approximately 90 degrees. When the crosshairs are directly over the rotated first star, the astronaut will press the ESC shutter trigger button. HAP will send the exact time of the image, and the ESC attitude to the PGSC. This second sighting of the same star allows the HERCULES software to determine the optical axis of the ESC. Another star sighting must be taken to establish the IMU three axis orientation. Using the HERCULES software menu system, the astronaut will choose another star. When the

crosshairs are directly over the new star, the astronaut will again press the ESC shutter trigger button. HAP will send another trigger data packet to the HERCULES software. HERCULES software will calculate a new STS position and the required ESC attitude to sight the chosen star from the STS location at the shutter trigger time. A reference orientation matrix is calculated by the HERCULES software using the required ESC attitude and the measured ESC attitude for the three star sightings. The reference orientation matrix is a 3 by 3 matrix that is used to perform a coordinate system translation. After initialization, every time HAP sends a packet of data to the HERCULES software, the reference orientation matrix is used to convert the data packet attitude to the Earth fixed frame of reference used by HERCULES.

All pertinent information to the location and initialization process is sent to the ESCEB to be stored with the image. This information will be used for post flight data analysis. Pertinent information includes: star numbers used for sightings, trigger data packet ESC attitudes for both star sightings, initial STS state vector, propagated STS state vector, time of sightings, HAP status, and the reference orientation matrix.

3.1.2 EARTH IMAGING OPERATIONS

Once initialized, HERCULES is available to capture and geolocate images of the Earth. To capture an image, the astronaut first places the ESCEB mode control switch in one of the four positions. Using the HERCULES menu system, the photograph and geolocate mode is selected. At the astronaut's option, annotation can be input which will be stored with the image. This annotation will assist the astronaut in documenting the motivation for the images. Annotation is not required by the HERCULES software. Cassette tape recorders will also be used to

take general notes during the imaging operations. Images are captured by pointing the ESC at the point of interest and depressing the shutter trigger button. At that point, the HAP will send the trigger time and ESC attitude to the PGSC. The PGSC will propagate the state vector to the time of the trigger, reference the HAP provided ESC attitude, and geolocate the center of the image. If the ESCEB mode control switch is in one of the storage positions, the ESCEB will store the image on the hard disk. Data is then sent to the ESCEB from the HERCULES software to be stored with the image.

All pertinent information for the location of the image is sent to the ESCEB to be stored with the image. This information will be used for post flight data analysis. Pertinent information includes: star numbers used for sightings, time of sightings, HAP status, reference orientation matrix, time of image, most recently input STS state vector update, propagated STS state vector value, latitude and longitude solution for the image boresight, and intermediate calculation results. If the image is also being transmitted on the downlink, the pertinent information listed above is also transmitted.

HERCULES can be used for several hours before initialization is required again. Re-initialization is needed because of errors that accumulate in the knowledge of the STS position. Inaccuracies are contained in the initial state vector value, and these inaccuracies cause increasing position knowledge error with time. Modeling of the STS orbit and drag are also not exact and add position errors with time. STS attitude is not used in the HERCULES software to determine drag; a mean attitude is used to determine drag. The STS attitude control system provides visible readouts of the STS attitude, but it is difficult to interface with the STS attitude control system for electronic readout. Assuming mean attitude introduces

errors in the drag modeling which leads to an error in the position knowledge of the STS. HERCULES will need to be reinitialized whenever the PGSC or the HAP are powered off and then on again, or reset. Because of the heat generated by the HERCULES system, the operating time will be limited by NASA to two hours. During this time, the ESC attitude knowledge will be adequate to meet system goals.

HERCULES has been designed for ease of use and accuracy. The astronaut only needs to assemble and initialize the system before images can be taken. Once initialized, images can be taken as they would be with a normal camera. The L-cubed system required the horizon and point of interest to be simultaneously in the field of view. HERCULES does not require the astronaut to view anything but the point of interest. This allows the astronaut to use a multitude of lenses with the ESC. There are no limitations for using high power lenses with the ESC. Images can be captured and geolocated of areas that do not contain distinguishing geography. HERCULES' first opportunity to be used on the STS will be aboard STS-53 in October 1992.

4.0 HAP DESIGN REQUIREMENTS AND THEORY OF OPERATION

This chapter presents the hardware and software requirements that I have designed HAP to operate under. A brief functional description of Ring Laser Gyros (RLG) is provided, and the HAP attitude algorithms that I implemented are presented.

4.1 HAP DESIGN REQUIREMENTS

Middeck payloads on the STS are stored in a small locker. To fit all of the HERCULES components into the allotted two lockers, the HAP enclosure must be no larger than 9.5" high by 17" wide and 20" deep. HAP is used only with the PGSC, and it is desirable that the HAP have the same width and depth so the HAP and PGSC can be stacked. Stacking will minimize the necessary shelf space required on the STS when HERCULES is operating.

HAP must operate in the STS space environment. HAP will operate in the pressurized flight deck, in a room temperature environment, 18 degrees Celsius to 27 degrees Celsius. The exterior surfaces of the unit must not exceed 45 degrees Celsius, to prevent a burn hazard. Launch vibration of 6.5 Grms and normal handling during operation must not damage any of the HAP components. Zero gravity must not effect HAP adversely. HAP must maintain critical functions in the

presence of the radiation environment of the STS orbit.

STS power feeds on the flight deck are limited to 5 Amps. Only one outlet can be used by HERCULES. Power is provided as 23-28 Volt D.C.. This single feed must power the IMU, the HAP, and the PGSC. Power drawn by the HAP must not exceed 30 Watts.

The HAP electronics must pass worst-case design analysis. The worst-case analysis consider parametric variances due to normal voltage variations, thermal variations, and production lot to lot differences. These sources of parametric variances are included into the minimum and maximum limits on the component data sheets. Digital circuits must have timing margin analysis performed. Analog and power components must not be over stressed or operated beyond maximum data sheet limits.

The HAP must interface with the IAE and process the gyro data using the algorithms given in Chapter 4.3. No data can be lost or buffered. Double precision math must be used process the gyro data to guarantee accuracy. Implementation of the algorithms must not degrade the overall performance of the gyro. The IMU can report a rate of up to 750 degrees per second. Scale factor linearity and stability are 10 parts per million. Bias stability is 0.1 degrees per hour.

Hap must provide a means of maintaining Universal Coordinated Time (UTC) for use by the PGSC. During the 18 days from the time the clock is set until the end of HERCULES operations, the clock must not drift more than 0.25 seconds. Time can not be set accurately on orbit, so safeguards must be taken to prevent loss of time.

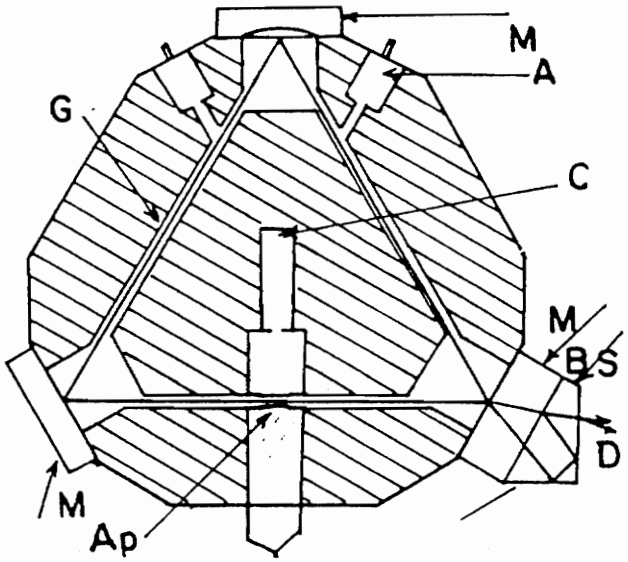
Hap must communicate with the PGSC and the ESC via RS232C interfaces at 2400 baud. Status and attitude data must be provided to the PGSC at regular

intervals of no slower than 1 time per second. HAP must provide a communication path between the ESCEB and the PGSC at the request of the PGSC. Commands sent by the PGSC must be recognized and performed by the HAP. HAP must provide the ability to dump HAP memory to the PGSC. As enhancements or corrections are made to the HAP software, the HAP must have the ability to accept new code from the PGSC and execute that code. IMU and ESCEB status must be monitored by the HAP and reported to the PGSC. ESC shutter trigger pulses must be detected by HAP. The time and attitude of the ESC at the trigger time must be determined by HAP and reported to the PGSC. Report of the trigger time must be within 0.01 seconds, as read from the UTC clock. Software must include means to recover from Single Event Upset (SEU) disruption of program flow and must recognize execution of data. Software must perform self tests of the system.

4.2 RING LASER GYRO FUNCTION DESCRIPTION

To understand the ESC attitude determination process, a general understanding of RLGs is required [29,32]. The RLG principle is illustrated schematically in Figure 4.1. A triangular block with three mirrors connected by cylindrical cavities is used for the laser resonator. The cavity is filled with a HeNe gas and sealed. The laser starts to oscillate between the three mirrors when the gas is sufficiently excited by an initial direct current discharge in the gas.

The laser travels in the closed path formed by the three mirrors and the cylindrical cavity. One of the two electromagnetic waves that form the beam circulates clockwise around the path, and the other circulates counterclockwise around the path. A standing wave is created by these two waves of the same frequency, as shown in Figure 4.2. Maxima of the standing wave are manifested as bright regions in the resonator and nulls of the standing wave are manifested as dark regions in



M-mirror; A-anode; C-cathode; BS-beamsplitter, D-to detector, G-gain tube; Ap-aperture.

Figure 4.1 - Ring Laser Gyro Block Diagram
Reprinted from [32]

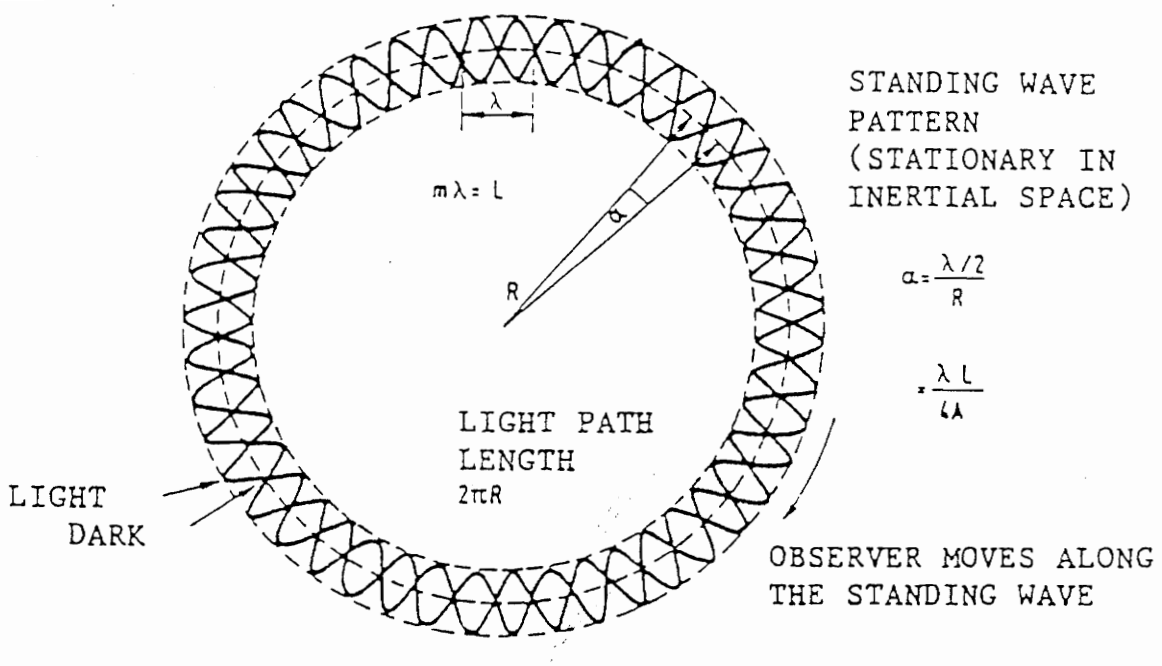


Figure 4.2 - Ring Laser Gyro Standing Wave Pattern
Reprinted from [29]

the resonator.

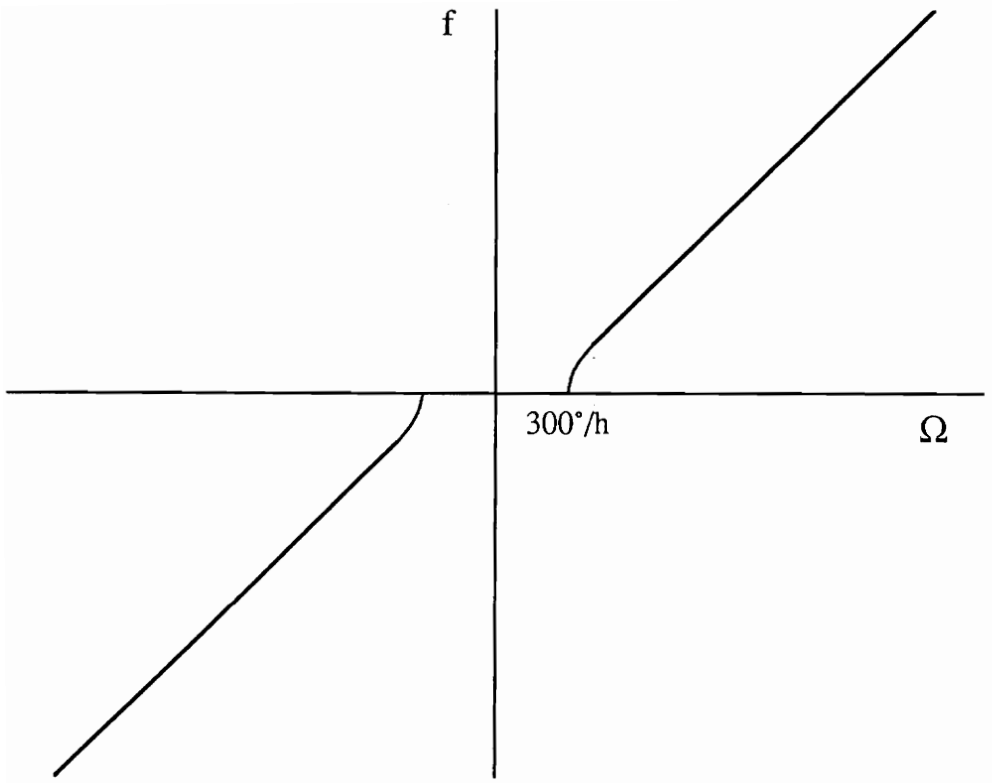
The standing wave pattern remains fixed in inertial space when the laser block is rotated rapidly. Using a readout prism, the bright and dark zones are detected using a photodiode. If the laser is rotated, then the interference fringes move past the photo-detector with a frequency proportional to the rate of rotation. The distance between adjacent nulls in the standing wave pattern is half of the wavelength. This distance gives the RLG a very accurate high resolution angular measurement capability.

For very low rotational rates, the standing wave pattern experiences an effect similar to static friction and moves with the rotating block. Known as the lock-in effect, this phenomenon will cause the RLG to have no output at low rates, as shown in Figure 4.3. Lock-in is caused is caused by a coupling between the two electromagnetic waves due to mirror backscatter. Even with superior mirrors, a RLG will not have good low-rate performance.

All RLGs currently use a dither technique to overcome the lock-in problem. The RLG is supported, by a spring element which allows the RLG to pivot about its center. Dithering motion is induced using a piezoelectric motor and the spring mass system is driven at its resonant frequency of 600Hz to 800Hz. Amplitude of the dithering motion is sensed and used in the control of the dither motor. Angular dither motion effectively averages over the transfer function discontinuity present for small rates. Because of the dithering, the lock-in effect occurs for an extremely short time, as the dither oscillation goes through its null.

The dithering motion induces a superimposed noise term on the RLG output signal. This noise results in a random walk of the angle signal. Errors induced by dithering, can be reduced by filtering out the dither frequencies.

Ideally, the spring element will only allow rotation of the block about the



Coupling of the light waves results in a dead zone. An output response is only produced for rates above the threshold.

Figure 4.3 - Lock-in Effect [29]

rotation axis and will be stiff about all transverse axes. Springs cannot be made to meet these requirements fully, and a small amount of the dithering contributes to tilting the block. When the three RLGs are mounted orthogonally to the sensor block, dithering couples between the RLGs. Angular oscillation of each RLG transfers a reaction moment to the sensor block and to the other RLGs. The physical relationship of the individual RLG oscillations to each other results in a conical sensor block motion and an associated drift. This motion can be compensated for, to achieve better performance. To prevent in-phase summing of these oscillations, each RLG is dithered at a slightly different frequency.

Despite the lock-in problem and the disadvantages of the dithering, the RLG has established itself for use in high accuracy systems. Basic advantages of the RLG include: digital output, no fast rotating parts or gyro run away, high accuracy, ability to measure high angular rates, and high reliability.

4.3 HAP ALGORITHMS

The algorithms required to maintain knowledge of the ESC attitude are presented below. Five stages of algorithms are needed to complete the process. Each of these stages is presented below.

4.3.1 FILTERING

Data filtering is the first stage of the algorithms. The IMU measures delta angles by counting the number of standing wave maxima that pass the photodiode. Delta angles are accumulated in the IMU, in a 16-bit up-down counter. This counter allows the IMU to maintain delta angle information that can support rates of up to 750 degrees/second. The delta angle values include noise that was generated by the dithering process. This noise needs to be filtered out.

Data is read from the IMU in a digital format. Filtering of this data will also be done in a digital format. The dithering motion causes noise in the 605Hz to 770Hz range. Several types of filters can be considered. An Infinite Impulse Response (IIR) filter was considered, however, IIR filters add non-linear phase shifts to the data [37]. These non-linear phase shifts will not provide acceptable data for attitude propagation, since angular velocities of different rates will be integrated at different times. Finite Impulse Response (FIR) filters have linear phase response[37]. A FIR filter has been chosen for this application.

The filter chosen uses four averaging filters in a cascaded configuration. The first filter performs a sum on two samples. Filters 2, 3, and 4 perform a sum on three samples. The gain from this summing is taken into account in the second stage algorithms. The delta angle accumulator in the IMU maintains a running total. Accumulator contents are not cleared after each read. This requires that a differencing be performed on the data to obtain the actual delta angle. The filter

algorithm is presented below.

IMU DATA FILTER

DEFINITIONS:

a(a,b) represents an array of numbers, as in FORTRAN. The first dimension represents one of the three axis that are being processed. The second dimension represents the filter stage number.

in(a) represents an array of the three axis of accumulator data from the IMU.

pin(a) represents an array of the previous values of the three axis of accumulator data from the IMU.

da(a) represents the delta angle of rotation since the last reading

fda(a) represents an array of filtered IMU accumulator data for each of the three axis.

Subscripts are used to designate time.

INITIAL CONDITIONS: a(1..3,1..15) = 0
in(1..3) = 0
pin(1..3) = 0

Perform the following algorithm for each of the three axis, where I specifies the axis:

(1)
IMU accumulator differencing

da(I) = in(I) - pin(I)

if (da(I) > 32768) da(I) = in(I) - pin(I) - 65536

if (da(I) < -32768) da(I) = in(I) - pin(I) + 65536

pin(i) = in(i)

(2)
FILTER 4, 3 Stage sum

$$fda_{t+1}(I) = a_t(I,5) + a_t(I,9) + a_t(I,10)$$

$$a_{t+1}(I,10) = a_t(I,9)$$

$$a_{t+1}(I,9) = a_t(I,5)$$

(3)
FILTER 3, 3 Stage sum.

$$a_{t+1}(I,5) = a_t(I,2) + a_t(I,6) + a_t(I,7)$$

$$a_{t+1}(I,7) = a_t(I,6)$$

$$a_{t+1}(I,6) = a_t(I,2)$$

(4)
FILTER 2, 3 Stage sum.

$$a_{t+1}(I,2) = a_t(I,0) + a_t(I,3) + a_t(I,4)$$

$$a_{t+1}(I,4) = a_t(I,3)$$

$$a_{t+1}(I,3) = a_t(I,0)$$

(5)
FILTER 1, 2 Stage sum.

$$a_{t+1}(I,0) = da_t(I) + a_t(I,1)$$

$$a_{t+1}(I,1) = da_t(I)$$

The magnitude response of this filter is shown in Figure 4.4. Latency of the filter is 1267.54 us.

4.2.2 CONING RECOVERY COMPENSATOR

Coning recovery compensation (CRC) is the second stage of the algorithms [35]. Double precision floating point arithmetic is used in this algorithm.

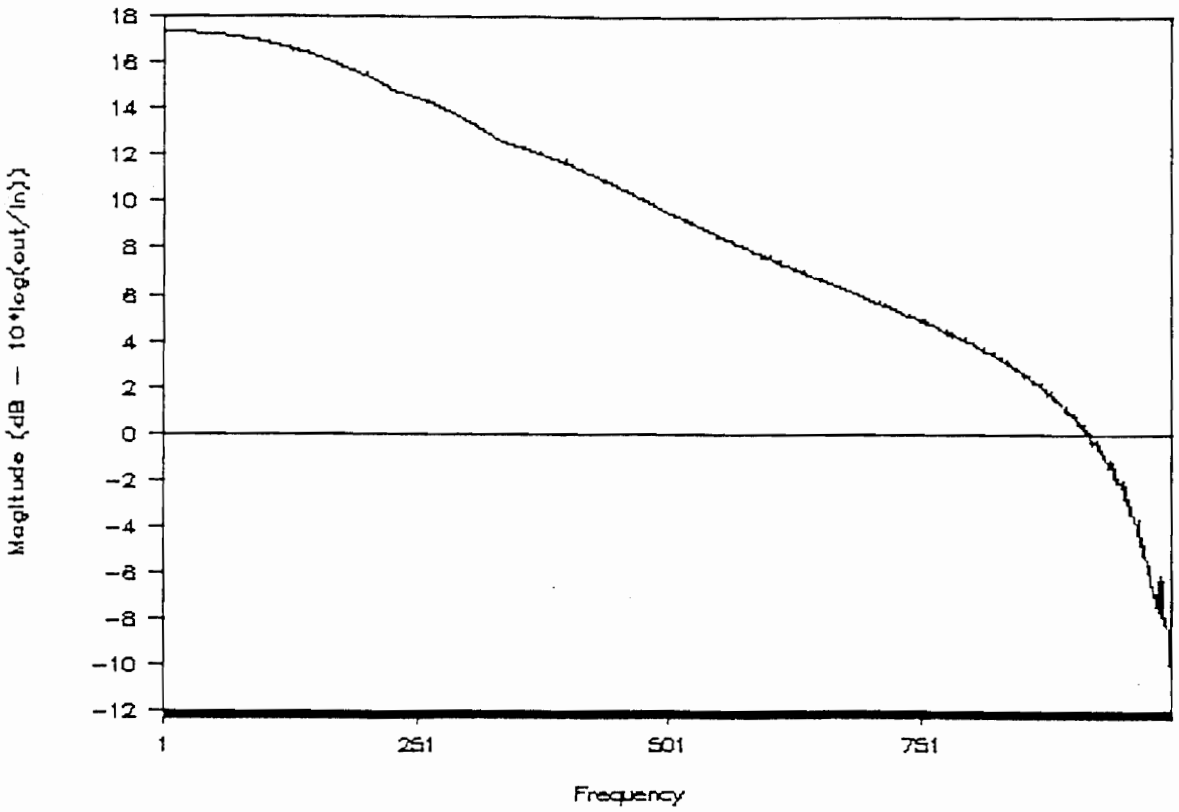


Figure 4.4 - Dither Filter Magnitude Response

CONING RECOVERY COMPENSATOR

DEFINITIONS:

fda(a) represents an array of filtered IMU accumulator data

pfda(a) represents an array of previous filtered IMU accumulator data

dfda(a) differenced filtered angles, angular speed

fc(a) represents an array of filtered and crc compensated delta angle data for each of the three axis.

g_{crc} is the coning compensator recovery gain provided by Honeywell.

Perform the following algorithm for each of the three axis, where I specifies the axis:

INITIAL CONDITIONS: pfda(1..3) = 0

(6)

$$dfda(I) = fda(I) - pfda(I)$$

$$pfda(I) = fda(I)$$

(7)

$$fc(I) = fda(I) + g_{crc} * dfda(I)$$

4.2.3 COMPENSATION

The third stage of the algorithm is used to compensate for scale factor, bias, gyro misalignment, and filter gain [35]. All of these quantities, except filter gain, are measured by the manufacturer and are provided as constants. The compensation algorithm is given below.

COMPENSATION

DEFINITIONS:

FC represents an array of filtered and crc compensated delta angle data for each of the three axis.

MG is a 3 by 3 matrix of constants provided to compensate for scale factor, filter gain and RLG misalignment.

B is a three element matrix of constants provided to compensate for gyro bias.

dt represents the sample time of the IAE.

FCGB represents an array of filtered, crc compensated, scale factor and misalignment compensated, bias compensated, and filter gain compensated delta angle data for each of the three IMU axes.

Uppercase bold characters represent vectors or arrays.

INITIAL CONDITIONS: **MG** given
B given
dt = 1/2000 = 0.0005 seconds

(8)

$$\mathbf{FCGB} = \mathbf{MG} * (\mathbf{FC} - \mathbf{B} * dt)$$

4.2.4 CONING COMPENSATOR

Coning compensation is the fourth stage of the algorithms [35]. This algorithm compensates for the coning motion induced by the dither motors. Double precision floating point arithmetic is used in this algorithm.

CONING COMPENSATOR

DEFINITIONS:

FCGB represents an array of filtered, crc compensated scale factor and misalignment compensated, bias compensated, and filter gain compensated delta angle data for each of the three IMU axes.

PFCGB represents an array of previously compensated delta angle data for each of the three IMU axes.

ANGLE represents an array of coning compensated angle data

Uppercase bold characters represent vectors or arrays.

(9)

$$\mathbf{ANGLE} = \mathbf{FCGB} + (1/12) * \mathbf{PFCGB} \times \mathbf{FCGB}$$

$$\mathbf{PFCGB} = \mathbf{FCGB}$$

4.2.4 ATTITUDE PROPAGATION

Once the data has been filtered and compensated, the delta angle values can be used to propagate the attitude [33]. Attitude is represented using the quaternion representation. The quaternion representation is preferred over Euler angle representation because of the analytical characteristics of the quaternion representation. A quaternion representation will not have problems with singularities.

The kinematic equations for the IMU can be written in differential form as:

(10)

$$\frac{d\mathbf{Q}}{dt} = \frac{1}{2} * \mathbf{OMEGA} * \mathbf{Q}$$

where **Q** is the quaternion and

(11)

$$\mathbf{OMEGA} = \begin{pmatrix} 0 & w_3 & -w_2 & w_1 \\ -w_3 & 0 & w_1 & w_2 \\ w_2 & -w_1 & 0 & w_3 \\ -w_1 & -w_2 & -w_3 & 0 \end{pmatrix}$$

w_1, w_2, and w_3 are angular velocities about each of the three orthogonal axes of the IMU.

If the angles are sampled at a fixed rate and the angular velocity is constant over the sampling interval, then a closed form solution to the kinematic equation is given by:

(12)
$$\mathbf{Q}(t_{n+1}) = e^{(0.5 * \mathbf{OMEGA} * T)} * \mathbf{Q}(t_n)$$

where T is the sampling interval.

In the HAP application T=0.0005 seconds. Samples are latched in the IAE electronics at this fixed rate. Assuming small angular velocities, compared to the sample rate, and a fixed sample rate, equation (12) has a convenient numerical solution of the form:

(13)
$$\mathbf{Q}(t_{n+1}) = [\cos (w * T / 2) * \mathbf{1} + (1 / w) * \sin (w * T / 2) * \mathbf{omega}] * \mathbf{Q}(t_n)$$

where $w = (w_1^2 + w_2^2 + w_3^2)^{1/2}$

This solution has been adapted to the IMU [5,36] as follows:

DEFINITIONS:

angle(a) represents an array of coning compensated angle data

th(a) represents an array of new angle values

tho(a) represents an array of previous angle values

q(a) represent the quaternion representation of attitude

INITIAL QUATERNION:

(14)
$$q(0) = 1$$

$$q(1..3) = 0$$

ATTITUDE PROPAGATION:

(15)
$$th(1) = 0.5 * (0.5 * angle(0) + tho(1))$$

$$th(2) = 0.5 * (0.5 * angle(1) + tho(2))$$

$$th(3) = 0.5 * (0.5 * angle(2) + tho(3))$$

$$th(0) = (th(1)^2 + th(2)^2 + th(3)^2)^{1/2}$$

(16)

if (th(0) = 0) then

$$cth = 1.0$$

$$sth = 1.0$$

else

$$cth = \cos (th(0))$$

$$sth = \sin (th(0)) / th(0)$$

endif

(17)

$$q_n(0) = q_{n-1}(0) * cth - sth * (th(1) * q_{n-1}(1) + th(2) * q_{n-1}(2) + th(3) * q_{n-1}(3))$$

$$q_n(1) = q_{n-1}(1) * cth + sth * (th(1) * q_{n-1}(0) + th(3) * q_{n-1}(2) - th(2) * q_{n-1}(3))$$

$$q_n(2) = q_{n-1}(2) * cth + sth * (th(2) * q_{n-1}(0) - th(3) * q_{n-1}(1) + th(1) * q_{n-1}(3))$$

$$q_n(3) = q_{n-1}(3) * cth + sth * (th(3) * q_{n-1}(0) + th(2) * q_{n-1}(1) - th(1) * q_{n-1}(2))$$

(18)

$$tho(1) = thn(1)$$

$$tho(2) = thn(2)$$

$$tho(3) = thn(3)$$

4.2.6 ALGORITHM USAGE

The algorithms presented above comprise the bulk of the processing that is required in HAP. Using these algorithms, HAP will gather IAE data, filter the data, compensate the data, and propagate the attitude. The algorithms presented will be completed for each sample of data taken from the IAE. Data is sampled at 2000 Hz. This sample rate is the maximum sample rate available from the IMU. The faster the samples are taken, the smaller the time steps are in the attitude propagation algorithms. As the time steps get smaller, the attitude propagation process becomes more accurate. Using these algorithms, the ESC camera attitude knowledge is maintained. The HAP hardware required to implement these algorithms is presented next.

5.0 HAP HARDWARE IMPLEMENTATION

The hardware for the HERCULES Attitude Processor, HAP, consists of several major components: microprocessor and support logic, PROM and RAM memories, serial communications, IAE interface, triple redundant Universal Time Correlated (UTC) clocks, camera interface, and power conditioning. A block diagram of the major HAP components is shown in Figure 5.1. Worst-case analysis of the HAP design is given in Chapter 7.

The microprocessor is the central controller and computer for the HAP. Management of the peripheral subsystems is controlled by the microprocessor. PROM and RAM are used by the microprocessor to execute programs and store data. The serial and IAE peripheral subsystems provide the necessary interfaces to the other HERCULES components. A clock peripheral subsystem provides accurate time for the HERCULES system. Each component of the block diagram is described below.

5.0.1 GENERAL DESIGN CONSIDERATIONS

Power and volume are constraining parameters in the design of HAP. NASA carefully preserves STS power for critical functions. Low power usage will reduce the effect of thermally dissipated power inside the STS. STS temperature regulation systems must operate under full capacity, and must maintain a

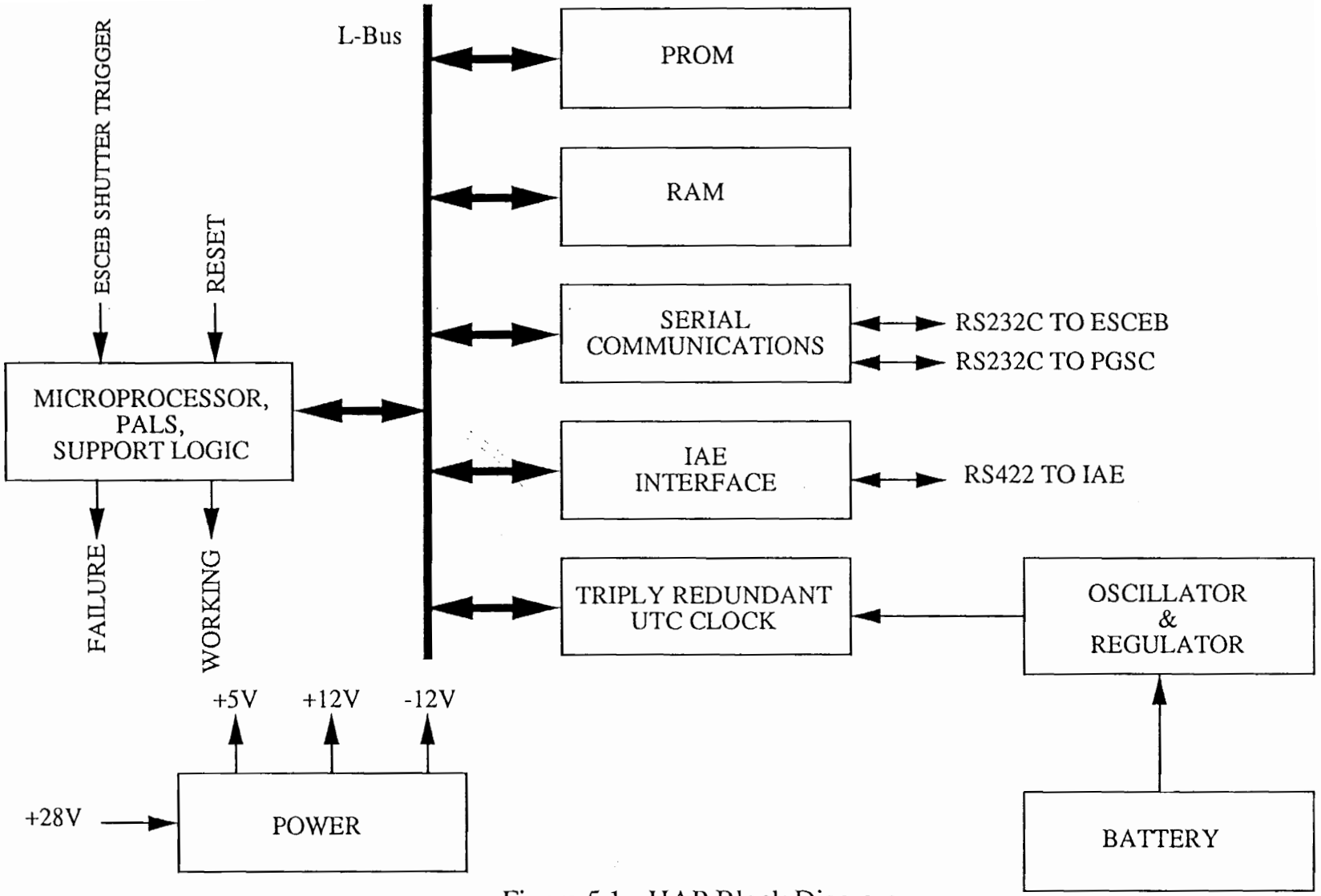


Figure 5.1 - HAP Block Diagram

comfortable temperature for the astronauts. Use of low power will also prevent heating of the exterior surfaces to temperatures that might burn the astronauts. To prevent major impacts to the STS functions, unregulated prime power required by the HAP has been limited by NASA to be below 30 watts. HAP's volume is limited by the STS locker size. A large middeck locker is 9.5" high by 17" wide and 20" deep [6].

High energy cosmic particles deposit charges on semiconductor devices. When a charge of sufficient value is deposited, an integrated circuit can change the state of a flip-flop or temporarily pulse a logic gate. The effect of the high energy particle is called a Single Event Upset (SEU). Since the SEU phenomenon is common in space, the design must be able to detect and recover from a SEU. SEUs usually manifest themselves as flipped bits in memory [16]. Protection from SEUs will be provided by both software and hardware. The microprocessor must detect execution of data and illegal opcodes to help guard against errant program flow. Watchdog timers, described in Chapter 5.4 are used to correct temporary upsets of microprocessor execution [39,40]. SEU immune Programmable Read Only Memory (PROM) has been used to prevent the loss of the operating software.

5.1 MICROPROCESSOR COMPONENT

The microprocessor block diagram component performs the following functions: HAP initialization and self-testing, detection of ESC shutter trigger pulses, generation of reset signals, system timing and control signal distribution, interrupt masking, and system data processing and interpretation.

Quick reaction to asynchronous events is required from the microprocessor. ESC shutter trigger pulses are initiated by the astronaut, and can occur at any time. HAP must accurately determine the time of the ESC shutter trigger pulse to within

0.01 seconds so the ESC attitude can be correctly correlated with the instant that the image is captured. IAE angle data is available once every 0.5 milliseconds, but this rate is not synchronous to the microprocessor. Every set of angle data needs to be captured and processed to maintain knowledge of the ESC attitude. Serial communications between the PGSC, ESCEB, and HAP are also asynchronous.

A high execution rate must be maintained by the microprocessor. IAE data filtering, IAE data compensation, and ESC attitude propagation require the largest number of computations in the HAP. IAE data filtering requires approximately 39 fixed point additions. Attitude propagation requires approximately 52 floating point multiplies, 34 floating point additions, and several floating point trigonometric functions. Since the IAE data needs to be read continuously, attitude propagation using the current set of data must be completed before the next set of data is available, 0.5 milliseconds later.

The microprocessor needs to react to asynchronous events quickly, in near real-time. A high performance microprocessor is required to perform the IAE data processing in the allotted 0.5 milliseconds. All of the microprocessor activities must be controlled within the HAP without human help. A real-time high performance embedded processor is needed to control the activity in HAP.

5.1.1 MICROPROCESSOR SELECTION

Several microprocessors were investigated to determine their suitability for the HAP application. The microprocessors investigated include: MIPS R3000, Motorola MC68040, and INTEL 80960MC. Important features studied include: word size, addressing modes and control, interrupt capabilities, performance, power requirements, design complexity for a minimal system, and volume required for a minimum system.

The MIPS R3000, manufactured by IDT and LSI Logic, is a Reduced Instruction Set Computer (RISC)[7,8,9]. Features of the R3000 are described below. Data and registers for the R3000 are 32-bits wide. A 4 Gbyte direct addressing range is provided by using the 32-bit addresses. Memory and peripherals share the same address space. The R3000 required memory management function are provided by an on-chip memory management unit. Up to 6 asynchronous interrupt sources can be connected to the on-chip interrupt controller. Pipelining is used internally to increase the instruction execution rate. Instruction and data caches are used to achieve high performance with this processor. Instruction and data cache memory controls are provided on-chip. Cache memories are not included on-chip and need to be added by the designer. Floating point arithmetic is supported using an off-chip floating point co-processor. The R3000 can execute up to 28 Million Instructions Per Second (MIPS).

Analysis given in Figure 5.2 shows that the R3000 can execute the HAP algorithms at the required rates. However, the R3000 does not use power and volume as efficiently as other microprocessors that were studied. The required functions in a R3000 design are not highly integrated onto a single chip. Co-processors are required to provide many of the required functions. Power required by the R3000 and its associated co-processor is approximately 8 watts. High complexity and high power will not help make the system more reliable. Use of co-processor and off-chip data and instruction cache memories increase system power and volume.

The Motorola MC68040 is a high performance, 32-bit microprocessor [10,11]. Data and registers are 32-bits wide. A 4 Gbyte direct addressing range is provided by using the 32-bit addresses. Memory and peripherals share the same

Instruction Type	Number Required	R3000 Cycles per Instruction	R3000 Execution Time (nsec)	MC68040 Cycles per Instruction	MC68040 Execution Time (nsec)	80960MC Cycles per Instruction	80960MC Execution Time (nsec)
Integer Addition	39	1	1950	1	1950	1	1950
Integer Move	51	1	2550	1	2550	1	2550
Floating Addition	23	2	2300	5	5750	20	23000
Floating Subtraction	11	2	1100	5	2750	20	11000
Floating Multiply	48	5	12000	5	12000	43	103200
Floating Divide	4	19	3800	38	7600	77	15400
Floating Square Root	1	150	7500	103	5150	104	5200
Floating SIN	1	450	22500	450	22500	441	22050
Floating COS	1	450	22500	450	22500	441	22050
Floating Move	35	1	1750	13	22750	7	12250
Time in Nanoseconds to Execute IAE Data Processing Algorithm			77950		105500		218650
Maximum Number of IAE Data Processing Algorithms per Second Possible			12828		9478		4573

Figure 5.2 - Microprocessor Execution Speed Comparison

address space. Memory management is provided using an on-chip memory management unit. Up to 8 encoded interrupt sources can be connected to the on-chip interrupt controller. A high degree of instruction execution parallelism is achieved by using multiple execution pipelines, and on-chip independent 4 Kbyte instruction and data caches. Floating point arithmetic is supported using an on-chip floating point execution unit. Because of the high level of on-chip integration, the MC68040 uses comparably little power and volume to complete a design. The MC68040 can execute up to 20 MIPS.

Analysis given in Figure 5.2 shows that the MC68040 can execute the HAP algorithms at the required rates. Unfortunately, the production and delivery schedule of the MC68040 did not make it available in time for the HAP application.

The INTEL 80960MC is a high performance 32-bit microprocessor [12,13,14]. Data and registers are 32-bits wide. A 4 Gbyte direct addressing range is provided by using the 32-bit addresses. Memory and peripherals share the same address space. Memory management is provided using an on-chip memory management unit. Up to 4 asynchronous interrupt sources can be connected to the on-chip interrupt controller. Using an external interrupt controller, 255 interrupts can be supported. Instruction pipelining, on-chip instruction cache, and parallel on-chip execution units are used to increase performance. Floating point arithmetic is supported using an on-chip floating point execution unit. At a 20 Mhz operating frequency, the processor can sustain an instruction execution rate of 7.5 MIPS and can burst execution to 20 MIPS.

Analysis given in Figure 5.2 shows that the 80960MC can execute the HAP algorithms at the required rates. On-chip floating point arithmetic unit, memory management unit, instruction cache, and interrupt unit reduce the power and

volume of a 80960MC design. The 80960MC was chosen for the HAP design.

The 80960MC satisfies the requirements for the HAP microprocessor. It is highly integrated, low power, processes the IMU data at the required rate, processes interrupts with low latency, and contains features useful for detecting program and hardware faults. The key features of the 80960MC are presented below.

5.1.2 INTEL 80960MC ARCHITECTURE DETAILS

Complex features of the 80960MC are implemented internally. Multiple processes are supported by a generous set of registers. In addition to a floating point register set, there are 5 sets of registers contained within the 80960MC. When a process switch is performed, all process registers are automatically cached on-chip or in system memory. The recovery of the process registers is also performed automatically. Interrupts are handled immediately or are automatically stored on a pending interrupt stack in user memory. No interrupts are lost or forgotten. Interrupts are handled based on their priority. The 80960MC has an automatic mechanism to handle faults. When arithmetic, floating point, or instruction faults occur, the appropriate fault handling routine is automatically called. The state of the microprocessor is automatically saved to enable efficient recovery from the fault.

After reset, the 80960MC automatically performs extensive internal self-tests. If the internal self-tests are completed successfully, the microprocessor reads the first 8 words from memory and calculates a checksum. If the checksum does not equal zero or if the internal self-test does not pass, the microprocessor asserts the *FAILURE signal line.

The 80960MC uses instruction pipelining and register scoreboarding to enhance the instruction execution rate. When a load instruction is encountered in

the code, a scoreboard bit is set in the destination register. When the destination register is loaded, the scoreboard bit is cleared. While the load is in progress, other instructions are allowed to execute as long as they do not use a register whose scoreboard bit is set. The register scoreboarding does not require special programming by the user. All of the scoreboarding tasks are handled by the 80960MC internally.

A 512 word instruction cache is included on-chip. Commonly used instructions execute in a single clock cycle. Floating point instructions are microcoded within the 80960MC and require multiple clock cycles to execute. The 80960MC floating point unit supports the IEEE standard for 754 for floating point arithmetic, exponential, logarithmic, and other transcendental functions [15].

The 80960MC interfaces to the support hardware through a high bandwidth, 32-bit multiplexed address and data, local bus (L-bus). Because of the large amount of caching, the L-bus supports burst transactions that transfer one to four 32-bit words of data. Transactions of 8-bits, 16-bits, or 32-bits are also supported. An address space of 4 Gbytes can be directly addressed. All memory and input and output (I/O) are mapped into this 4 Gbyte address space. Though the L-bus supports multiple bus masters, the HAP design does not use Direct Memory Access (DMA) peripherals or multiple processors, and the 80960MC is always the bus master.

Five states are defined for the L-bus: idle (Ti), address (Ta), data (Td), wait (Tw), and recovery (Tr). Each of the five bus states are at least one clock cycle long. When no transactions are in progress, the L-bus is in the idle bus state. The address bus state is one cycle long and occurs when the microprocessor outputs the address of the transaction. Address information is driven from the microprocessor's 32 multiplexed address and data lines. Immediately after the address bus

state, the L-bus enters the data bus state for one clock cycle. During this state, the data is transmitted to or from the microprocessor using the 32 multiplexed address and data lines. If the peripheral device is unable to store or provide the requested data by the end of the data bus state, the bus enters the wait bus state. When the peripheral device has either stored or provided the requested data the L-bus moves to the recovery bus state. The L-bus state diagram is shown in Figure 5.3.

L-bus signals are grouped into the address and data lines and the control lines. Address and data (LAD) lines consist of 32 bi-directional signal lines. These 32 signal lines are used to transmit address during the address bus state. Data is sent over the 32 signal lines during the data bus state and the wait bus state. During the idle bus state, the LAD lines are in their high impedance state. There are 12 signal lines that control the transfer of data. These 12 lines consist of *ALE, *ADS, DT/*R, *DEN, W/*R, *BE3 TO *BE0, *READY, *LOCK, and CACHE. Address is latched using the *ALE output signal during the address bus state. A read or a write transaction is identified using the W/*R output signal. The *BE3 to *BE0 byte enable output signals determine which bytes will be involved in the transaction. When the peripheral device has either stored or retrieved the data, the *READY input signal is driven low. Slow devices are disconnected from the LAD signals using the *DEN and DT/*R output signals and user provided bi-directional transceivers. The *LOCK output signal is used in multiple bus master configurations, to prevent bus mastership from switching during a flag test and set operation. The CACHE output signal specifies whether the output data cacheable.

Relations of these signals during a transaction are shown in Figures 5.4, 5.5, 5.6 and 5.7. Figure 5.4 shows a single read L-bus transaction. A single write L-bus transaction is shown in Figure 5.5. L-bus burst read and write transactions are shown in Figures 5.6 and Figure 5.7, respectively.

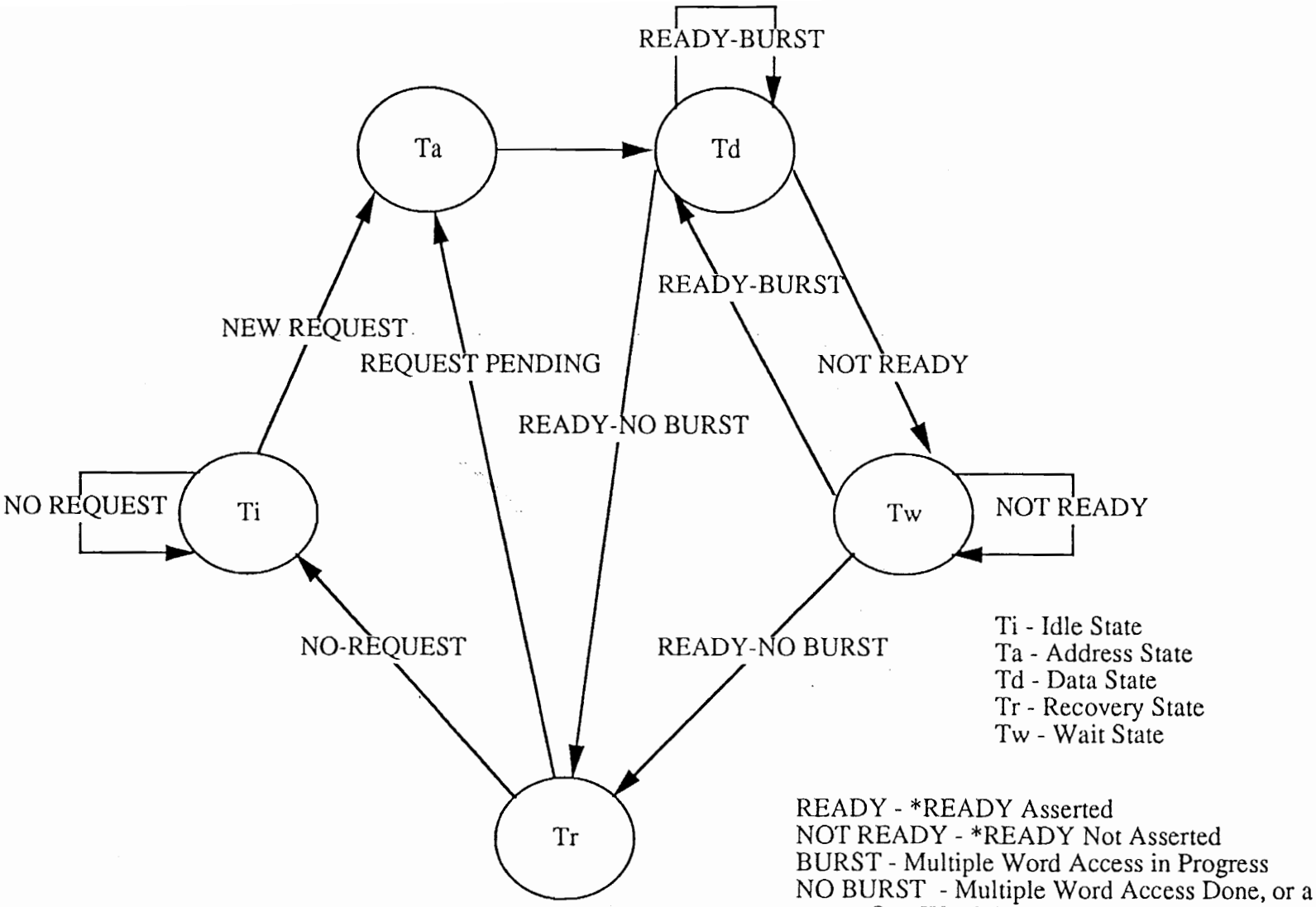


Figure 5.3 - Basic L- Bus States [12]

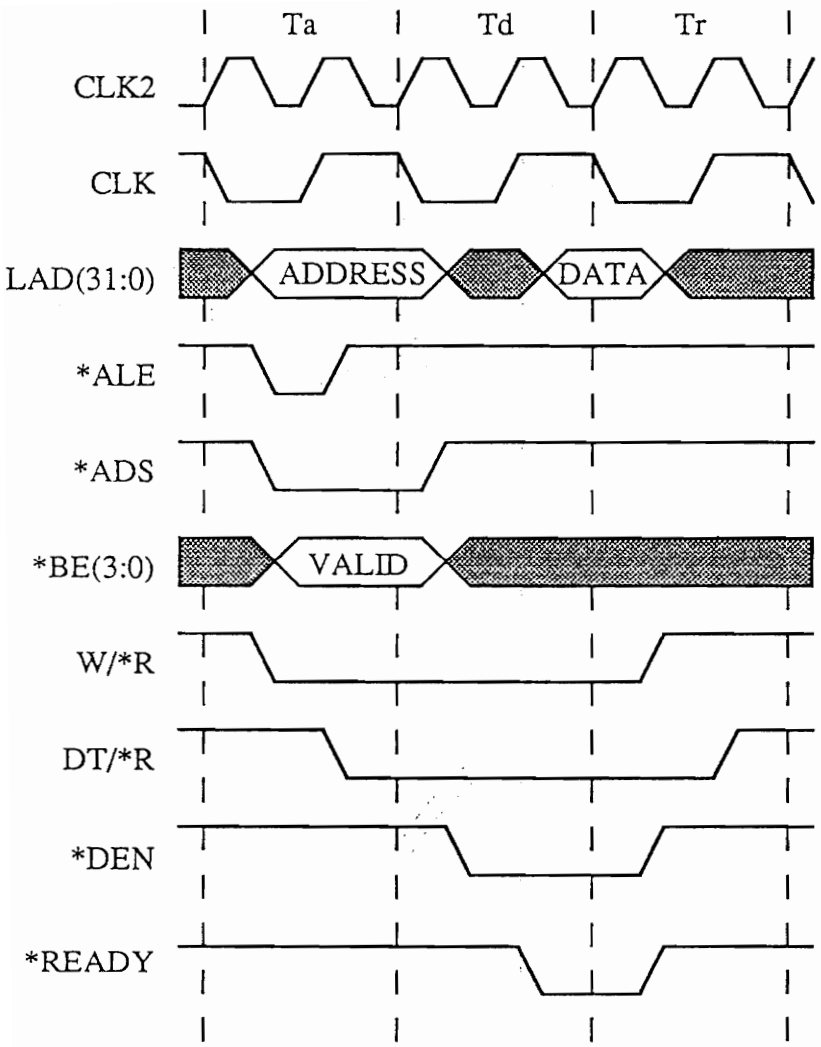


Figure 5.4 - Processor Read Transaction [12]

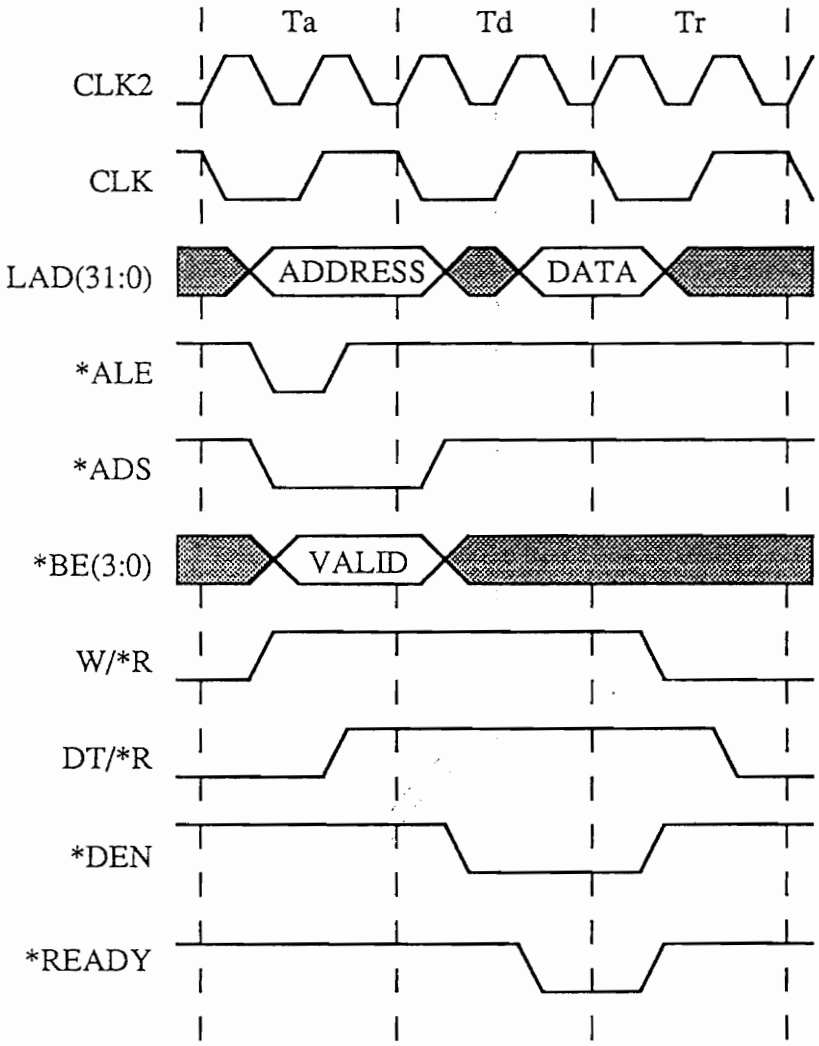


Figure 5.5 - Processor Write Transaction [12]

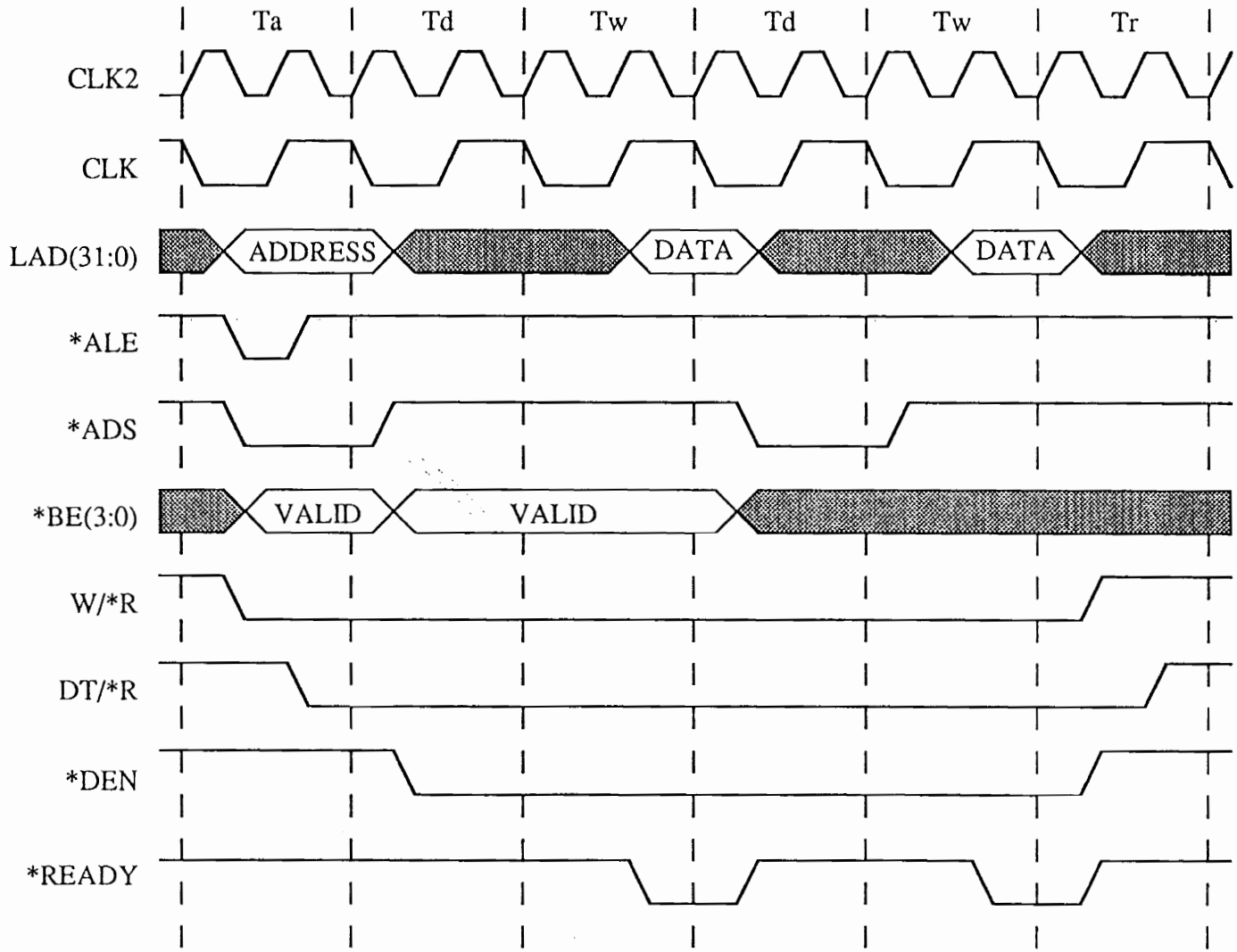


Figure 5.6 - Processor Burst Read Transaction [12]

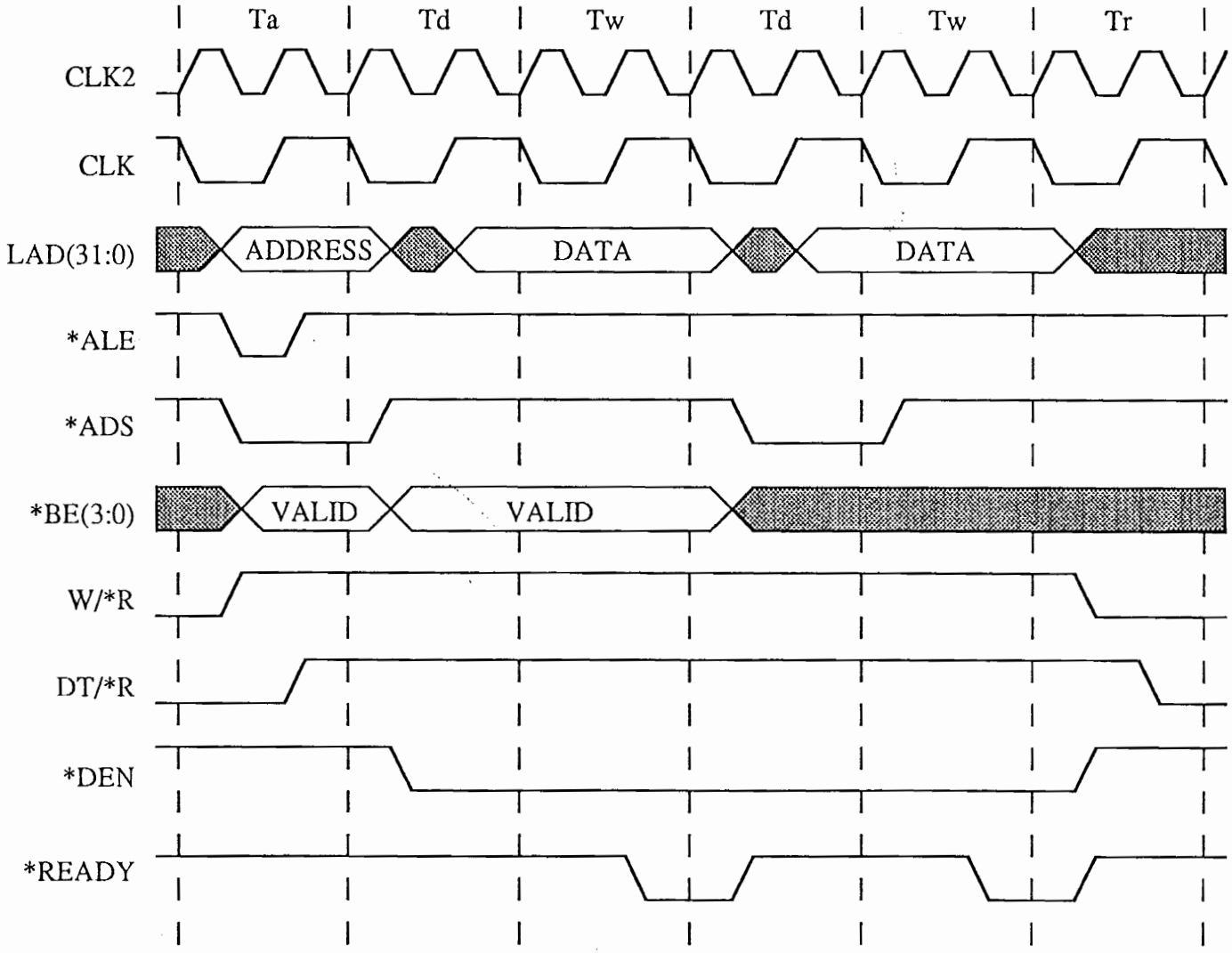


Figure 5.7 - Processor Burst Write Transaction [12]

5.1.3 HAP MICROPROCESSOR DESIGN

The schematic that I have drawn for the microprocessor block diagram component is shown in Appendix D, drawing HERCULES-0004, schematic sheet 3. Figures 5.4 through 5.7 were used in my design of the L-bus support hardware. Major components of the microprocessor block diagram component include: clock generation circuits, reset generation circuits, ESC shutter trigger pulse conditioning circuit, interrupt enable register, address latches and decoder, byte enable latch programmable array logic (PAL), burst logic control PAL, and wait state generator and peripheral device control signal generation PAL. A worst-case analysis for the microprocessor block diagram component is given in Chapter 7.4.

A crystal oscillator is used to generate the microprocessor 40MHz CLK2 input clock. CLK2 is divided into a two phase clock internal to the microprocessor. CLK2 is divided into a 20MHz CLK and a 10MHz CLK_BY_2 clock for use by the bus control logic and the Universal Asynchronous Receiver Transmitter (UART). Microprocessor internal clock phasing is synchronized to the external CLK using special reset timing.

The microprocessor requires the RESET signal to be at least 41 CLK cycles long. HAP guarantees the RESET signal width using the value of the capacitor C1 in the reset circuit and the hysteresis of the comparator. Two D-flip-flops are used to synchronize RESET to CLK so CLK2 and CLK are phased properly for the microprocessor. I have designed the reset circuits to generate the reset signal for any of the following three conditions: HAP power is turned on, the two front panel reset buttons are simultaneously depressed, or the second level of watchdog timers expire.

Four prioritized direct interrupt signals are provided on the 80960MC. Priority of each of the interrupts is specified by software. Interrupt 0 has been

allocated to the first level of the watchdog timer. Interrupt 1 has been allocated to the IAE data ready signal. Interrupt 2 has been allocated to the shutter trigger pulse. The ESC shutter trigger pulse is conditioned using a comparator with hysteresis. Interrupt 3 has been allocated to the serial communications peripheral. An 8-bit interrupt enable register has been added to the design. Upon reset, the interrupt enable register will be reset and will disable all interrupts to the microprocessor. After software has initialized the system, the software will enable the interrupts by writing ones to the interrupt enable register bits 7 through 3. Bit 0 of the interrupt enable register is an output that will be used as a 'heart beat' or working signal. This signal will drive a front panel light emitting diode (LED) which will blink when the system is working correctly. Bits 1 and 2 are unused general purpose unused outputs.

Address is latched during the L-bus address bus state by four 8-bit latch chips controlled by the inverted *ALE signal. System access time to the memory and peripherals is primarily determined by the speed of these latches. High speed Advanced Schottky latches were selected [34]. The output of these latches is sent to the address inputs of the peripheral devices. Address decoding is performed on the latched address. Eight 16 Mbyte sections are decoded by the single chip address decoder. Each peripheral is allocated one of the sections. The HAP processor memory map and peripheral allocations are shown in Figure 5.8.

As shown in Figures 5.4, 5.5, 5.6, and 5.7, the byte enables are not valid during the entire transaction. A PAL has been used to latch the byte enables so they are valid for the proper periods of the transaction. I have designed one state machine for each of the four byte enables, each state machine has the same state diagram. A PAL was used because discrete logic did not satisfy worst case timing requirements. A state diagram for the byte enable latch is shown in Figure 5.9.

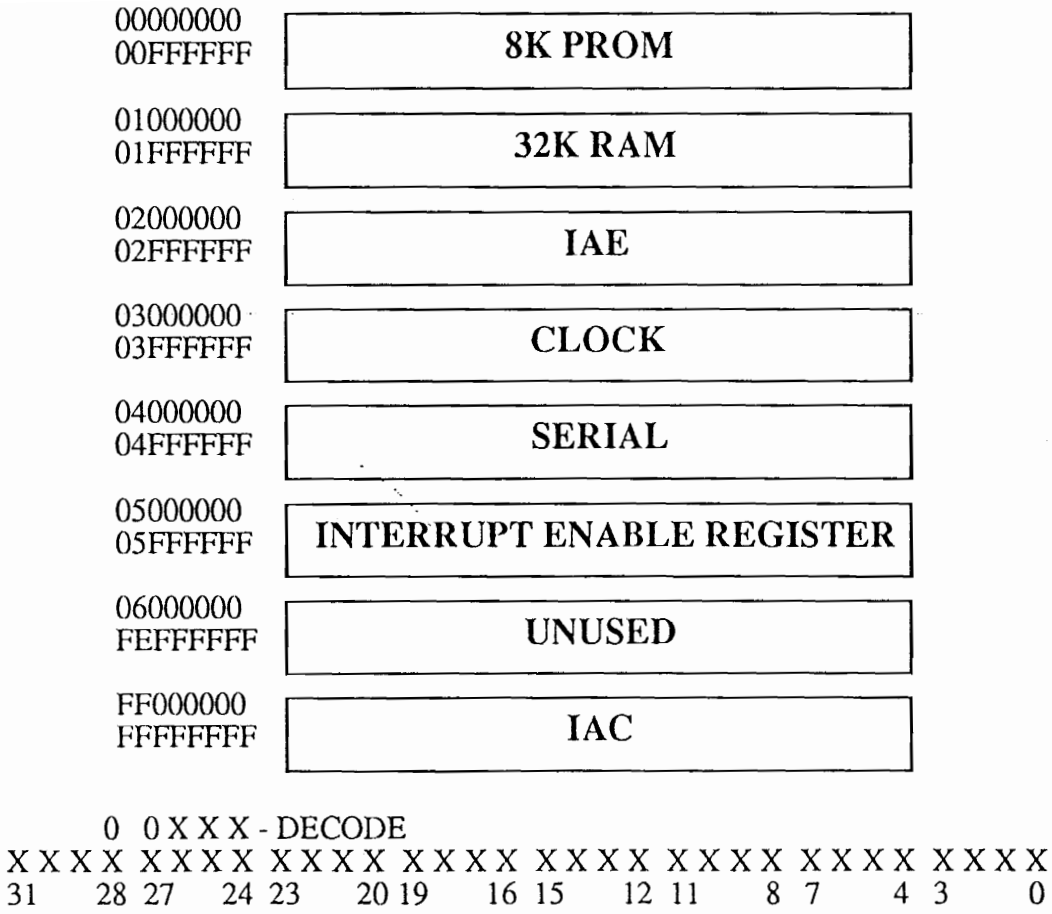
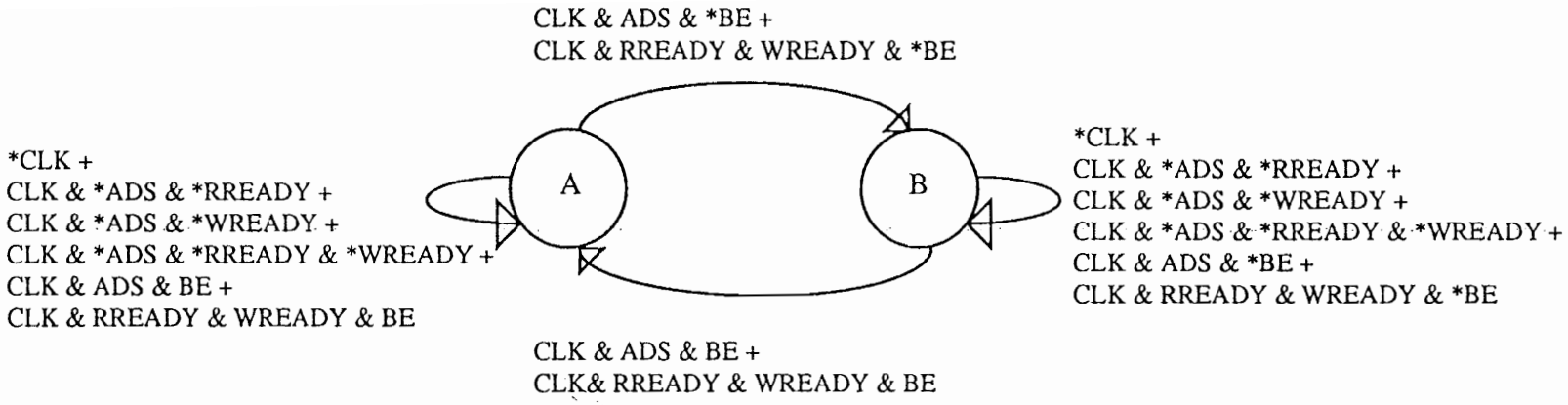


Figure 5.8 - HAP Memory Map



INPUTS:
CLK
*WREADY
*RREADY
*ADS
*BE

OUTPUT:
*LBE

A,B STATES OF *LBE
A -> 0
B -> 1

Figure 5.9 - Byte Enable Latch State Diagram

CLK2 is used to drive the state machine to minimize clock skew and output delay. Byte enables are latched during the L-bus address cycle and during the L-bus data cycle when the *READY signals are asserted. The PAL equations and test vectors that I have generated are included in Appendix A. Byte enables are used by the microprocessor to select which bytes are required for the transaction.

The HAP design must support burst transactions, as shown in Figures 5.6 and 5.7. Burst transactions are used automatically to fetch instructions and to maintain the microprocessor stacks and pointers. Application software can also specify burst transactions. A second PAL controls the burst transactions. I chose the PAL because of its flexibility, high speed, and small implementation size. To minimize clock skew and delays, my design uses CLK2 to drive the state machines. Two state machines have been used, one to control the address, and one to count the number of transfers. State diagrams for the two state machines are shown in Figures 5.10 and 5.11. Address bits 0 and 1 determine how many words will be transferred in the transaction, these bits are stored in the SIZE_OUT1 and SIZE_OUT0 flip-flops at the end of the L-bus address state. Address bits 2 and 3 determine the first address in the transaction, these bits are stored in the ADDRESS_OUT2 and ADDRESS_OUT3 flip-flops at the end of the L-bus address state. After each transfer is complete, if the SIZE_OUT is not zero, the SIZE_OUT is decremented, and the ADDRESS_OUT is incremented to point to the next address of the transfer. This is repeated until the SIZE_OUT is zero. When the SIZE_out is zero, both the SIZE and the ADDRESS_OUT do not change state until another L-bus address state occurs. The PAL equations and test vectors that I have written are included in Appendix B.

Worst-case timing analysis that I performed, showed that the *DEN signal available from the microprocessor would violate some of the timing conditions of

INPUTS:
CLK2
ADDRESS_IN = (LAD3,LAD2)
*ADS
*DEN
*READY
SIZE_OUT

OUTPUTS:
ADDRESS_OUT

LOAD FUNCTION:
IF (*ADS=0 AND *DEN=1) THEN
ADDRESS_OUT = ADDRESS_IN
ENDIF

A,B,C,D STATES OF ADDRESS_OUT
A -> ADDRESS_OUT = 00
B -> ADDRESS_OUT = 01
C -> ADDRESS_OUT = 10
D -> ADDRESS_OUT = 11

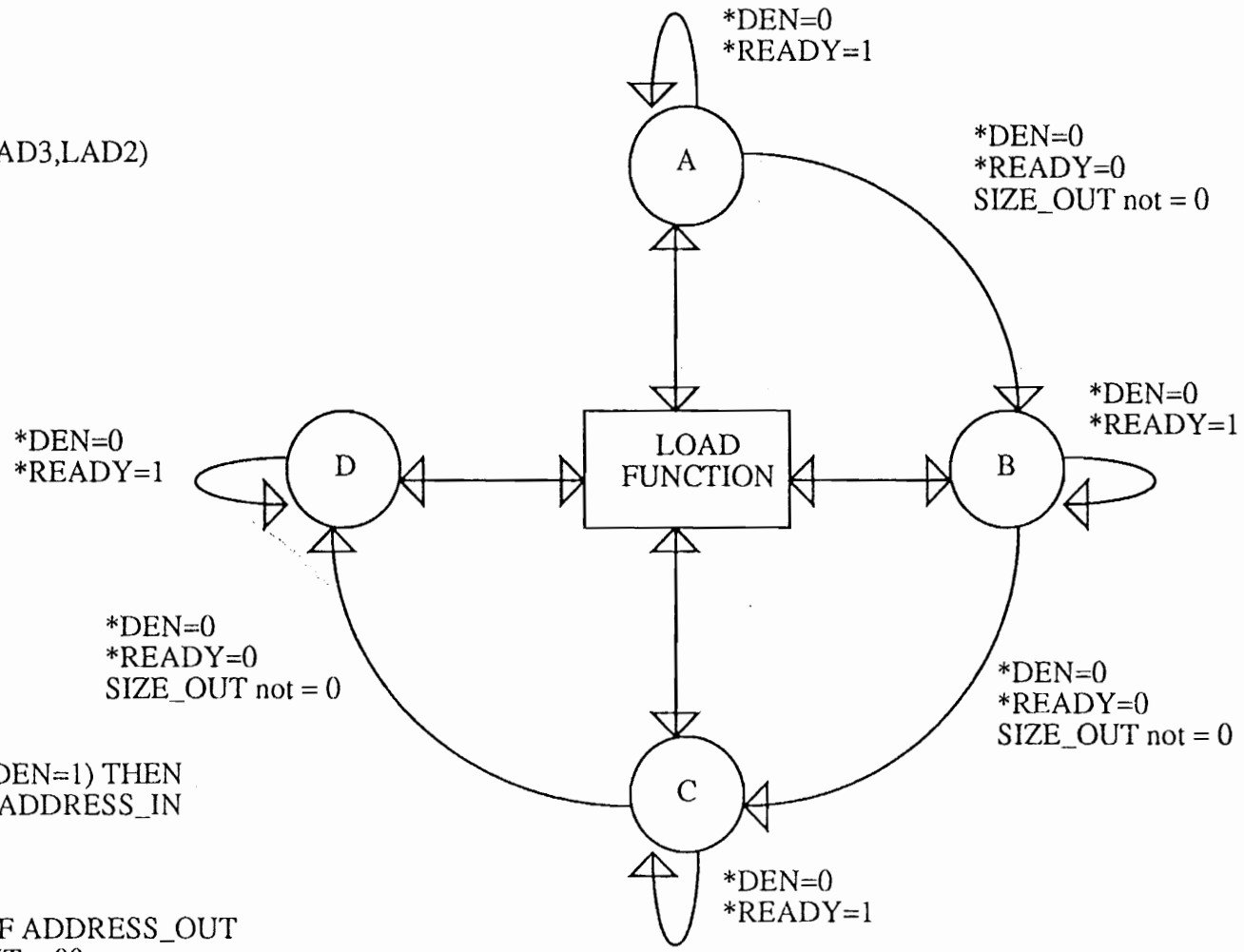


Figure 5.10 - Address Latch and Counter

INPUTS:
CLK2
SIZE_IN = (LAD1,LAD0)
*ADS
*DEN
*READY

OUTPUTS:
SIZE_OUT

LOAD FUNCTION:
IF (*ADS=0 AND *DEN=1) THEN
SIZE_OUT = SIZE _IN
ENDIF

A,B,C,D STATES OF SIZE_OUT
A -> SIZE_OUT = 00
B -> SIZE_OUT = 01
C -> SIZE_OUT = 10
D -> SIZE_OUT = 11

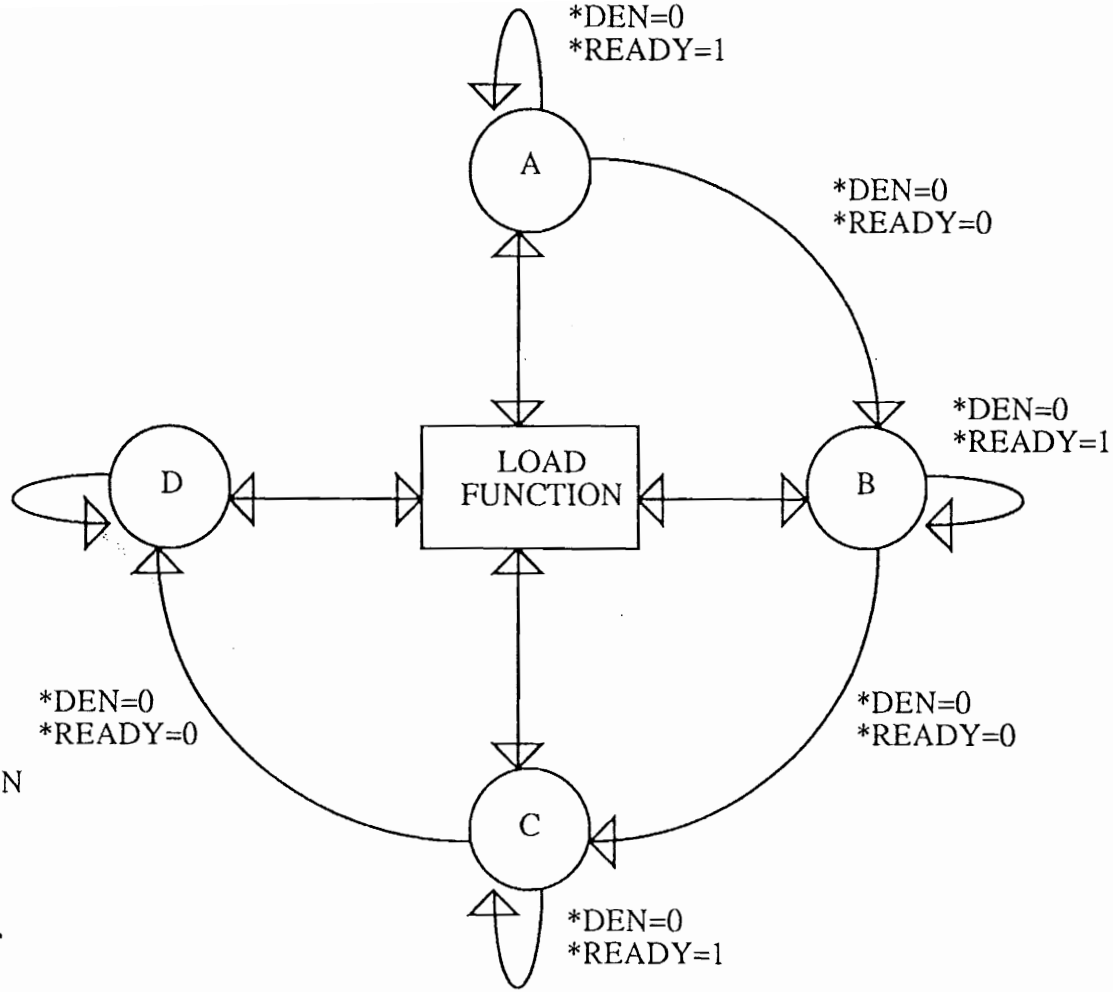
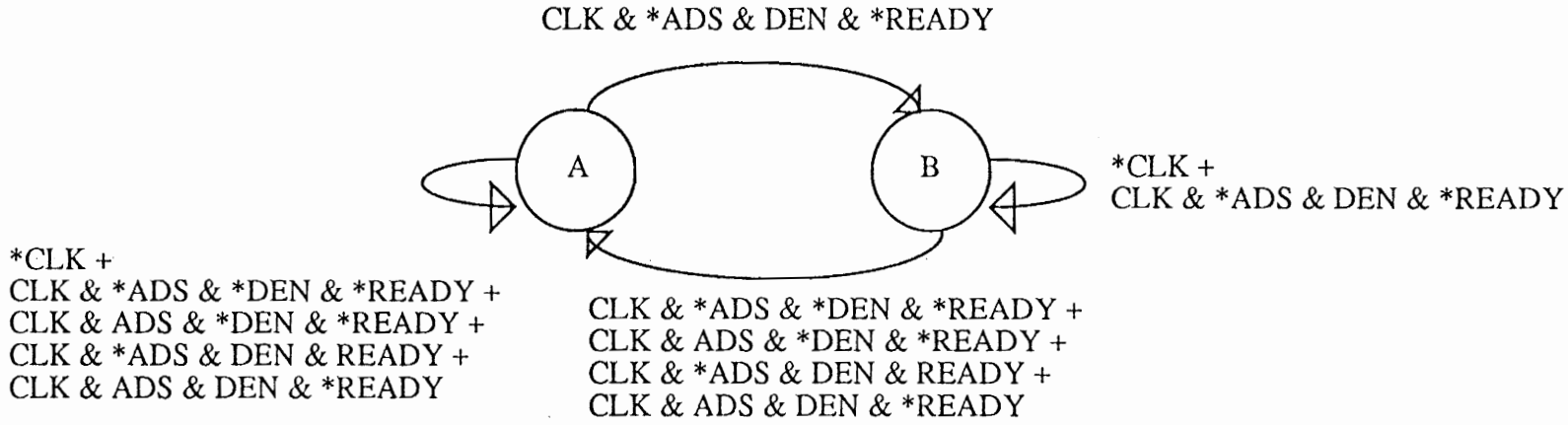


Figure 5.11 - Size Latch and Counter

the peripherals. After discovering this problem, I designed a DDEN signal, or delayed *DEN signal. DDEN is generated by a third PAL. This signal is used to identify the data enable period of the transaction. A state machine was used to generate DDEN. The state machine is clocked using the CLK2 signal to reduce delays. A state diagram for the DDEN signal is shown in Figure 5.12. DDEN is asserted during the first L-bus data cycle, and is unasserted one cycle after the L-bus data cycle coincident with the *READY signal. I have included my state equations and test vectors for the third PAL signal DDEN in Appendix C.

The IAE peripheral requires relatively long data setup and hold times compared to the other peripherals. A special data enable or data select signal needed to be generated to satisfy the timing requirements of the IAE. *IAE_DS is generated in the third PAL similarly to the DDEN signal, and is clocked using the CLK2 signal. Figure 5.13 shows the state diagram for my design of the *IAE_DS signal, and the state equations and test vectors for this signal are given in Appendix C.

Write enable (*WE) and output enable (*OE) signals are required by the peripherals and are also generated by the third PAL. *OE is asserted during DDEN when a read transaction is present, and guarantees sufficient address and chip select setup and hold times for the peripherals. A state machine, clocked by the CLK2 signal, is used to generate the *OE signal. The state diagram for the *OE state machine is shown in Figure 5.14. *WE is asserted during DDEN when a write transaction is present. Due to the timing of the peripherals, *WE must be unasserted before the data is removed to guarantee peripheral data hold time requirements are met. To guarantee the peripheral minimum data hold time requirements, *WE is unasserted one cycle before *READY is asserted. The signal *WE_CUT is generated by the third PAL and is used to shorten *WE by one cycle.



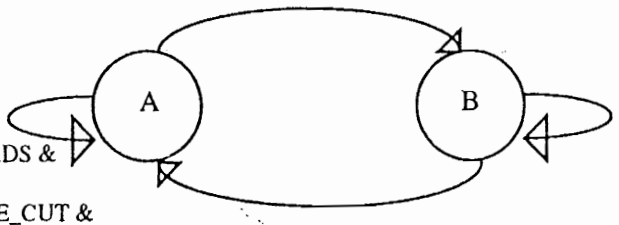
INPUTS:
CLK
*ADS
*DEN
*READY

OUTPUT:
DDEN

A,B STATES OF DDEN
A -> 0
B -> 1

Figure 5.12 - DDEN State Diagram

*IAE_CE +
CLK & IAE_CE & *W__R & *ADS & *DDEN & *READY +
CLK & IAE_CE & *W__R & ADS & *DDEN & *READY +
CLK & IAE_CE & *W__R & *ADS & DDEN & READY +
CLK & IAE_CE & *W__R & ADS & DDEN & *READY +
CLK & IAE_CE & W__R & *WE_+CUT & *ADS & *DDEN & *READY +
CLK & IAE_CE & W__R & *WE_+CUT & ADS & *DDEN & *READY +
CLK & IAE_CE & W__R & *WE_+CUT & *ADS & DDEN & READY +
CLK & IAE_CE & W__R & *WE_+CUT & ADS & DDEN & *READY +
CLK & IAE_CE & W__R & WE_+CUT



*CLK +
CLK & IAE_CE & *W__R & *ADS &
DDEN & *READY +
CLK & IAE_CE & W__R & *WE_CUT &
*ADS & DDEN & *READY

CLK & IAE_CE & *W__R &
*ADS & DDEN & *READY +
CLK & IAE_CE & W__R & *WE_CUT &
*ADS & DDEN & *READY

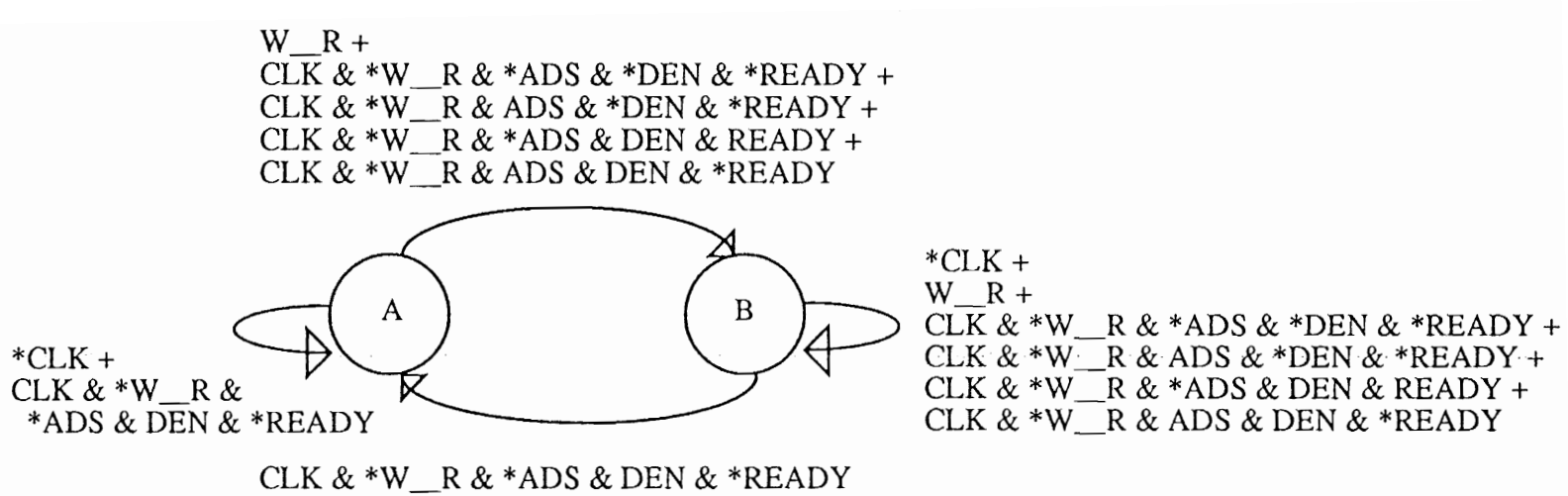
*CLK +
*IAE_CE +
CLK & IAE_CE & *W__R & *ADS & *DDEN & *READY +
CLK & IAE_CE & *W__R & ADS & *DDEN & *READY +
CLK & IAE_CE & *W__R & *ADS & DDEN & READY +
CLK & IAE_CE & *W__R & ADS & DDEN & *READY +
CLK & IAE_CE & W__R & *WE_CUT & *ADS & *DDEN &
*READY +
CLK & IAE_CE & W__R & *WE_CUT & ADS & *DDEN &
*READY +
CLK & IAE_CE & W__R & *WE_CUT & *ADS & DDEN &
READY +
CLK & IAE_CE & W__R & *WE_CUT & ADS & DDEN &
*READY +
CLK & IAE_CE & W__R & WE_CUT

INPUTS:
CLK
W__R
DDEN
*IAE_CE
*ADS
*WE_CUT

OUTPUT:
*IAE_DS

A,B STATES OF *IAE_DS
A -> 0
B -> 1

Figure 5.13 - *IAE_DS State Diagram



INPUTS:
CLK
*ADS
*DEN
*READY
W__R

OUTPUT:
*OE

A,B STATES OF *OE
A -> 0
B -> 1

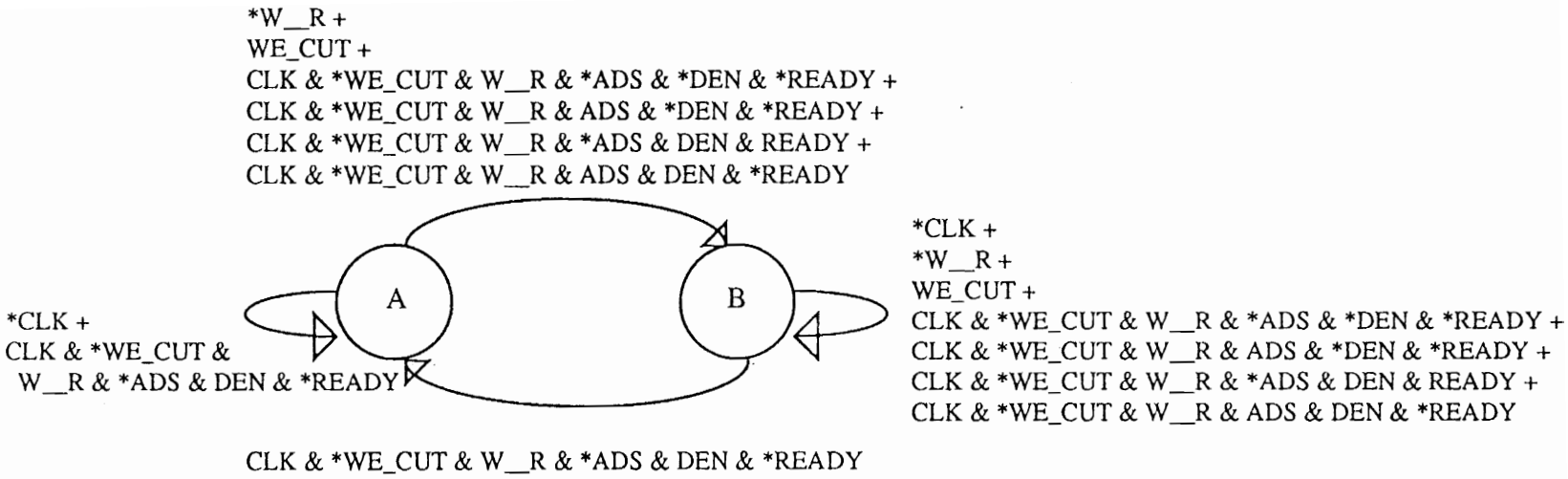
Figure 5.14 - *OE State Diagram

A state machine was used to generate the *WE signal and is clocked by the CLK2 signal. Figure 5.15 shows the state diagram for the *WE signal. I have included my state equations and test vectors for the third PAL signals *OE and *WE in Appendix C.

Slow peripheral devices require that L-bus wait states be generated. Wait states are generated using a third PAL and a counter. The counter is used to count the number of wait states that have occurred. Wait states are generated depending on the count, R_*W, and the peripheral that is selected. Each peripheral is assigned enough wait states to allow proper reading and writing of data. There are two ready signals generated in the third PAL. One ready signal is for read transactions and one is for write transactions. Wait states can be assigned differently for read transaction and write transactions. *READY is generated for the microprocessor when both the read ready (*RREADY) and write ready (*WREADY) signals are active. *WE_CUT is also generated in the third PAL, and uses the counter and the peripheral selected to determine when to be asserted. Write transactions require the use of the *WE_CUT signal, and are one cycle longer than read transactions. The PAL equations and test vectors that I have designed for *RREADY, *WREADY, and *WE_CUT are included in Appendix C.

5.2 PROM

Schematic page 4 of drawing HERCULES-0004 in Appendix D, shows the PROM component of Figure 5.1. The PROM block diagram component contains four 8K by 8 PROMS for flight use, and four 32K by 8 EPROMS for debug and testing. The PROMS contain a PROM checksum, configuration and startup programs, and the HAP application programs. PROMS with fast access times have been chosen so time is not wasted waiting for instructions to be loaded.



INPUTS:
CLK
*ADS
*DEN
*READY
W__R
*WE_CUT

OUTPUT:
*WE

A,B STATES OF *WE
A -> 0
B -> 1

Figure 5.15 - *WE State Diagram

A *BADACC signal is generated when the microprocessor tries to write to PROM. When *BADACC is active the microprocessor enters a fault handling routine that I have written and returns execution of the program to a known location. PROMs that are immune to SEU, have been chosen for the design to prevent loss of the HAP program on orbit [17].

5.3 RAM

Page 5 of the drawing HERCULES-0004 in Appendix D, shows the RAM component of Figure 5.1. The RAM block diagram component consists of four 32K by 8 RAMS [18]. Program data, microprocessor fault stack, microprocessor process stack, interrupt stack and other system data structures are contained in the RAM. High speed RAM was chosen to give the microprocessor fast access to system data structures and HAP application software data. RAM has been designed to allow byte accesses, which are required by the microprocessor.

5.4 UTC CLOCK

The UTC clock component of Figure 5.1 is shown on schematic sheet 6 of drawing HERCULES-0004 and drawings HERCULES-0002 and HERCULES-0010 of Appendix D. Time is a critical piece of information in the HERCULES system. Three independent UTC clock chips and one stable oscillator are used to maintain time [19,20]. UTC clock circuits run continuously, and are battery powered when the prime power is removed from HAP. Time is maintained from hundredths of seconds to years using the UTC clock chips. All three timers are driven by a single 32.768KHz oscillator. The oscillator is a temperature compensated high stability oscillator. UTC time drift between calibrations is calculated in Chapter 7.

Power to the UTC clock and oscillator is regulated with a low power, low dropout linear regulator. Power is taken from the batteries when the HAP is powered off. When the HAP is powered on, the UTC clock and oscillator power comes from the +12 volts power supply. This will help conserve battery life. Alkaline batteries have been chosen to provide power for the UTC clock and oscillator for a minimum of 60 days. Battery life calculations are shown in Chapter 7. Alkaline batteries were chosen because of their established safety and reliability record.

Power Fail Warning (PFW) circuitry has been designed into the UTC clock circuits to prevent disturbance of the UTC clocks during a HAP power up or down sequences. A *PFW signal is asserted when the HAP power is failing, this signal induces the power down mode of the UTC clock chips. When the UTC clock chips enter the power down mode, the address, data, and control lines are ignored. The UTC clock chips become active when the *PFW signal is unasserted.

General purpose inputs were added to the clock block diagram component. The UTC clock component is word addressable only, all UTC clock chips and general purpose inputs are simultaneously read from or written to. General purpose inputs are used to read the following information: count of the IAE interrupts, ESCEB powered on indicator, and battery low indicator. A count of the IAE interrupts is used to verify that the HAP has not missed any of the IAE data. Each time the software enters the IAE data processing routine, a software counter is incremented and compared to the hardware counter. If the two counters are different, the software has failed to process an IAE interrupt and the lost track of IMU flag is set in the HAP data packets being sent to the PGSC. Before the IAE interrupt is enabled, the software counter is initialized to the current value of the hardware counter to synchronize the counters. Use of this count is described in Chapter 6. A

ESCEB powered on indicator is used by HAP to generate the ESCEB on/off flag in the HAP data packets. A low battery voltage indicator is used by HAP to generate the low battery flag in the HAP data packets.

Worst-case timing analysis, given in Chapter 7, showed that the UTC clock component would not remove its data before the end of the L-bus recovery state. If data is not removed from the L-bus before the end of the recovery state, then there will be bus contention and the address of the next transaction will fight the data that the UTC chips have not yet removed. Address values will be corrupted. Bi-directional data transceivers were added to my design to ensure that the data is removed from the L-bus before the end of the L-bus recovery cycle.

5.4.1 LOSS OF TIME ON ORBIT

Time is set on the ground before launch. If time is lost while HERCULES is on the STS, the geolocation process will not be able to continue in real-time. HERCULES software uses the time that is maintained in HAP to propagate the STS state vector. Loss of time knowledge means that the position of the STS cannot be determined. Images can be captured and stored if the time is incorrect, but the geolocation information will be incorrect. Time integrity can be lost by a failure of two of the UTC clock chips, by a SEU changing the UTC counter within two of the UTC clock chips, by a failure of the oscillator, or by a failure of the battery backup system.

If the UTC clocks are still running when they are returned to Earth, then the photographs can be geolocated by additional processing on the ground. The time will be read when the HAP returns. Using the error in the time, the actual time of the photographs will be determined.

It is of utmost importance that the timers maintain correct time from launch

to landing. Each occasion that the microprocessor gathers time, it compares the time of each of the timers. If one of the timers does not match the other two, then the timer in error is set to match the others. If all three times are different, then the time is unknown.

5.4.2 SETTING THE CLOCK

The UTC clock chips need to be set on the ground before launch. Accurate time is not available to STS experiments on the middeck or flight deck. The time will be set on the ground using a PGSC computer and a Global Positioning Satellite (GPS) receiver. The GPS receiver will provide a one pulse per second (PPS) time pulse which is accurate to UTC to within one microsecond [21]. A Pulse Conversion Unit (PCU) is used to convert the PPS pulse to a one pulse per minute (PPM) pulse. Figure 5.16 shows the hardware configuration and the PPS and PPM relationship for the time setting procedure.

The PCU is manually synchronized with the PPS pulse. A PCU reset button is pressed when the GPS receiver reaches 00 seconds. The PCU will then count 59 PPS pulses and output the 60th PPS pulse. An enable switch is used to allow the PPM pulse out of the PCU. The pulse conversion unit is connected to the HAP ESC shutter trigger input. A memory load is performed by the PGSC, which loads the time of an upcoming trigger into HAP. A memory dump is performed by the PGSC to verify that the time has been loaded into memory correctly. After verification of a correct time load into memory, the HAP is sent a command to set the clocks on the next trigger. When the time that was loaded into HAP arrives, the PCU sends a trigger pulse to HAP, and HAP initializes and sets the clocks to that time. Time setting is checked by sending the PCU pulse to the HAP through the

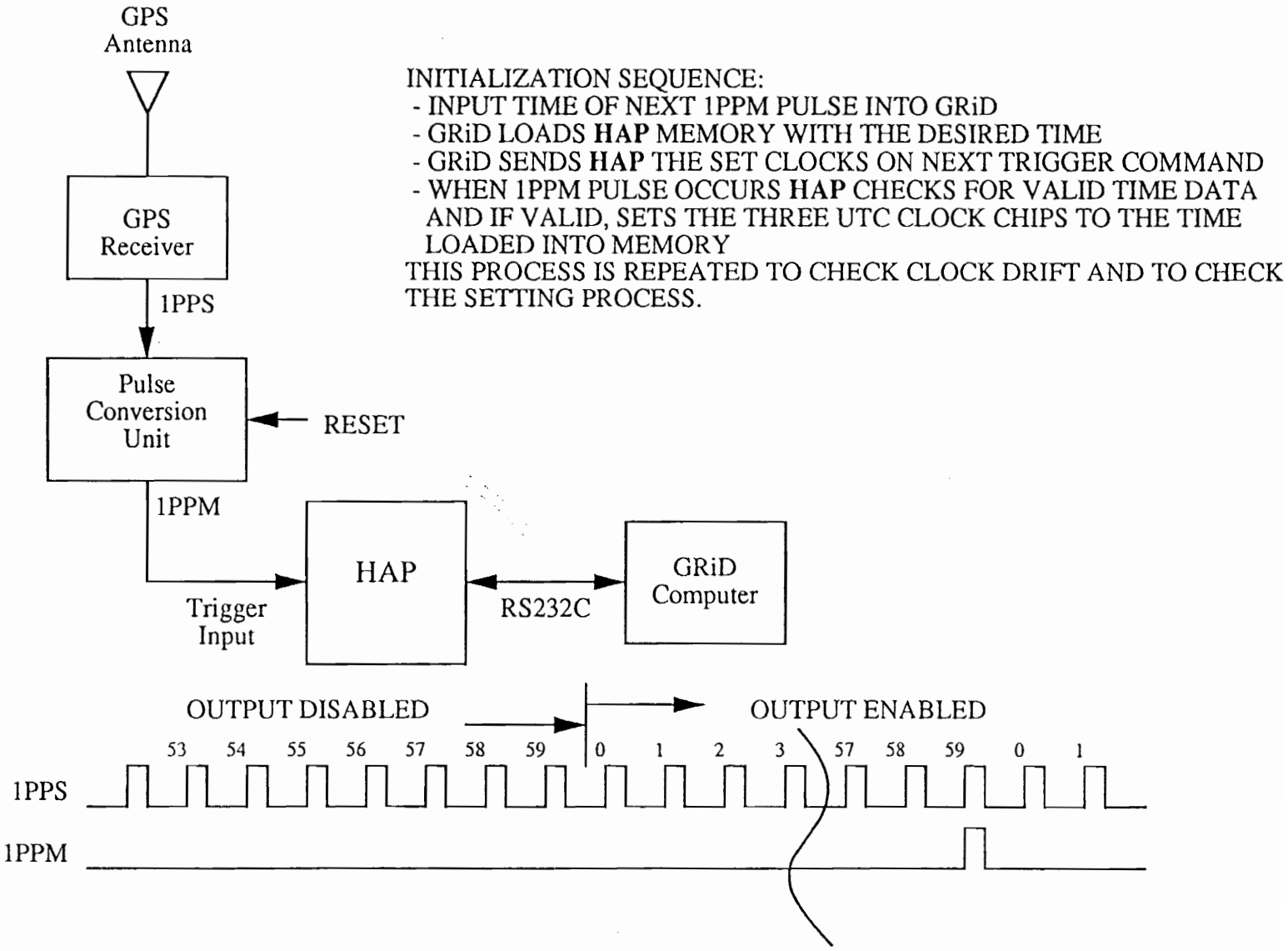


Figure 5.16 - HAP Clock Setting Procedure

HAP ESC shutter trigger input. The PCU pulse is synchronized to the GPS UTC time and the time reported by the HAP will show the current time according to HAP. Time will be read after the STS mission to measure how far the HAP UTC clock has drifted since it was set on the ground.

5.4.3 CLOCK CHIP TIMERS

The clock chips contain internal timers. One timer from each of the three chips is used as a watchdog timer. The watchdog timer must be reset by the microprocessor before the time expires. If the first timer is not reset before it expires, then it is assumed that the microprocessor has become lost. If the first timer expires, the highest priority microprocessor interrupt is generated. The second and third timers are set so the combination of their outputs provides a pulse. If these timers expire, the processor got lost and the first level watchdog timer was unable to complete a recovery. When these timers expire, the pulse causes a HAP hardware reset.

5.5 SERIAL COMMUNICATIONS

The serial communications component of Figure 5.1 is shown on page 7 of the HERCULES-0004 schematic in Appendix D. Serial communications are handled by a single chip [22]. Two Universal Asynchronous Receiver Transmitter (UART) functions are integrated on one chip. These UARTs have internal 16 character read and write First In First Out (FIFO) memories. Because of the FIFOs, the interrupts to the microprocessor are kept to a minimum. UART interrupts are generated when the receive FIFOs are almost full. An interrupt is also generated when the transmit FIFOs have been emptied. On-chip baud rate generator counters allow various baud rates to be programmed and generated with the

same input clock. Both serial interfaces operate at approximately 2400 baud. RS232C drivers and receivers are used in the HAP design [23].

5.6 IAE INTERFACE

The IAE interface component of Figure 5.1 is shown on sheet 8 of the HERCULES-0004 schematic in Appendix D. All IAE data is 16-bits wide, and the IAE interface is only addressable in 16-bit pieces. Signals between the HAP and the IAE are RS422 [24]. RS422 was used to gain speed, noise immunity, and buffering. IAE access time is slow compared to the other peripherals in HAP. The IAE requires 560 nanoseconds to access data or to write data. The IAE also cannot remove its data, during a read transaction, before the end of the L-bus recovery state. A bi-directional data transceiver was added to cut off the IAE data to the L-bus before the end of the L-bus recovery state.

5.7 POWER

The HERCULES-0011 schematic in Appendix D, shows the power component of Figure 5.1. DC input power to the HAP comes from the STS fuel cells. This power is nominally 27 volts, and is a maximum of 32 volts [6]. HAP power is fused on the input to 5 Amps. A fuse holder is located on the front of the HAP enclosure, so fuses can be easily replaced. Power is controlled by a 5 Amp front panel on/off switch. The switches and fuses used throughout HERCULES are the same, to give each component of the HERCULES system the same look and feel.

HAP needs to meet the Electromagnetic Interference and Electromagnetic Compatibility (EMI/EMC) requirements of the STS. Input filters are required on

the power supplies, to meet the requirements of the STS specifications [6]. I have selected a hybrid EMI filter and hybrid power supplies for the HAP design [25]. These filters and power supplies operate over the required voltage range, are power efficient, and small in volume. If the current is not limited, the EMI filter and power supplies draw 25 Amps for several microseconds when power is applied. This inrush current will deteriorate the power on/off switch and will eventually cause a failure of that switch. The inrush current needs to be limited to below the 5 Amp rating of the on/off switch.

Inrush current is limited in my design using a resistor and P-channel MOSFET combination [26]. When the HAP is initially powered on, the MOSFET is off and the inrush current is conducted through the resistor. The resistor is sized to limit the maximum current to 4 amps. When the input filter capacitance has charged the MOSFET is gradually biased on. On-resistance of the MOSFET is 0.3 Ohms, and the current required by the HAP is conducted through the MOSFET. When HAP is powered down, the MOSFET is biased off.

HAP requires regulated +5 volt, +12 volt, and -12 volt power. One hybrid power supply provides the +5 volts, and another hybrid power supply provides +12 volts and -12 volts. Power requirement calculations are given in Chapter 7.

6.0 HAP SOFTWARE IMPLEMENTATION

The software that I have written to control the functions of the HAP are described in this Chapter. HAP hardware must be initialized and boot up self-testing must be performed. Data must be gathered from the IAE, accurately processed, and sent to the PGSC. Camera triggers must be detected and time tagged. Command interpretation and status monitoring must be performed. The software that I have written and tested, manages all of the HAP resources to meet the requirements given in Chapter 3 and Chapter 4.

6.1 HAP SOFTWARE STRUCTURE

An overall philosophy needed to be adopted to allow a structured approach to satisfying all the HAP software requirements. Two software coordination techniques considered are discussed below.

The software was written using an interrupt driven structure. An interrupt driven structure was chosen for several reasons: serial data is asynchronous to the processor execution, IAE data must be read and processed before another IAE data sample is available, IAE data is asynchronous to the processor execution, and ESC shutter trigger pulses are asynchronous to the processor execution. When a signal on one of the four external interrupt pins becomes active the microprocessor automatically schedules a process to service the interrupting event. The interrupts

are handled in order of priority. I have measured the interrupt latency, or time to switch from the current task to the interrupt service routine, to be about 7 micro seconds. This time was short enough for me to consider using an interrupt structure to initiate the gathering and processing of data from the IAE. IAE data must be gathered and processed continuously to maintain ESC attitude knowledge. An interrupt driven structure guarantees that the IAE data will be processed before lower priority activities are handled. ESC shutter trigger pulses need to be time tagged closely. An interrupt driven structure will assure time tagging and ESC attitude data gathering activities occur before lower priority activities are handled.

I also explored a polled software structure. The polled approach uses software to sample status lines which indicate a need for service. In the HAP application, the software would waste valuable computation time looking for service requests. Figure 5.2 shows that the microprocessor needs to use the execution time effectively. Reducing the latency to start executing a service request would require high rate service request line polling, and might require larger tasks to be manually split to allow more service request line polling. Additional software complexity is needed to achieve a low service request to service handling latency. A polled approach will be ineffective for the watchdog timer function. The watchdog timer interrupt will only expire if the software is executing in an undesirable manner. If the software is not executing correctly, the watchdog timer service request might never be seen.

6.2 80960MC EXECUTION ENVIRONMENT

The 80960MC executes instructions and stores and manipulates data within an execution environment [13]. Shown in Figure 6.1, the execution environment

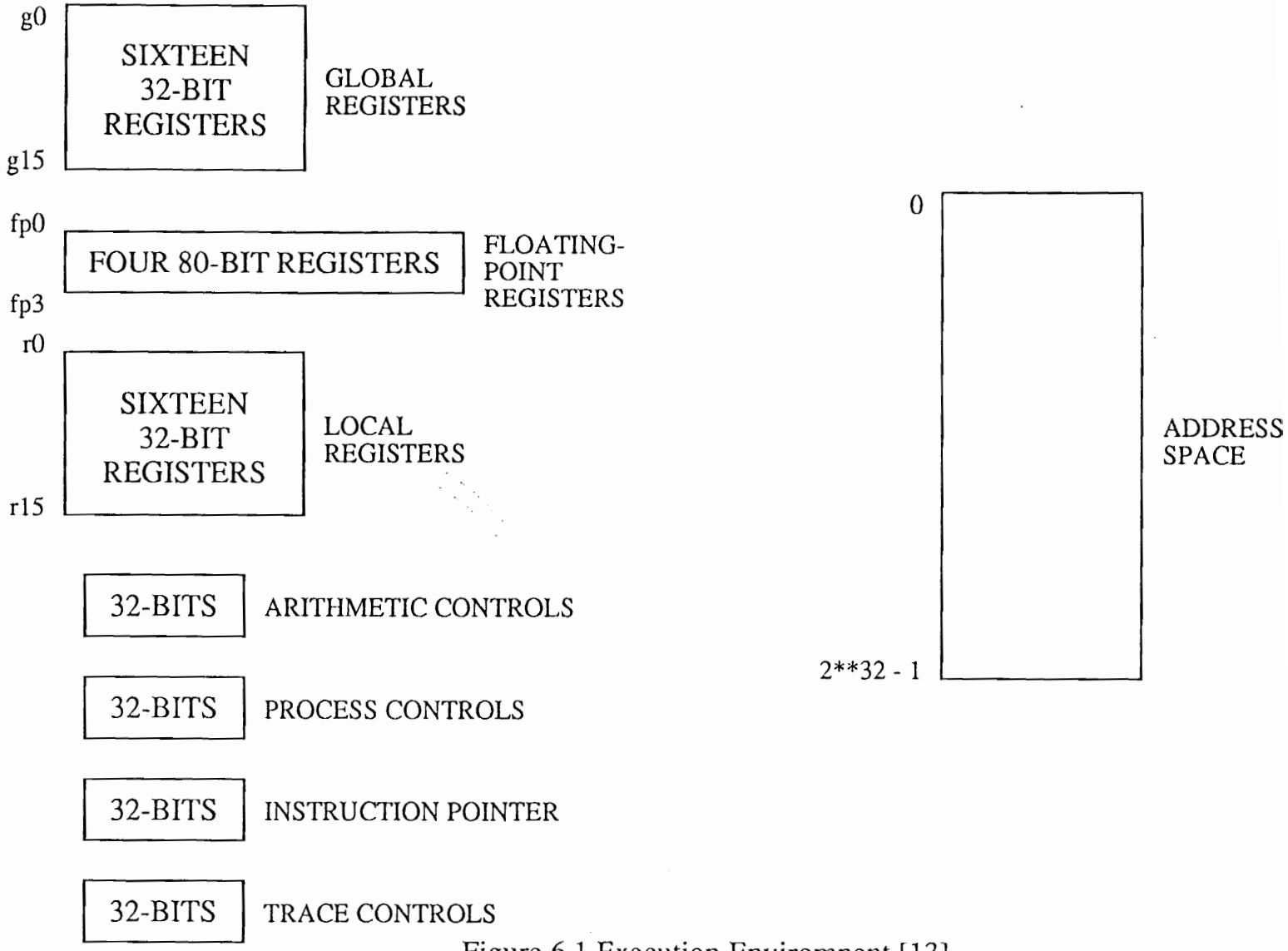


Figure 6.1 Execution Enviromnent [13]

includes a 4 Gbyte address space, 16 global registers, 4 floating point registers, 16 local registers, and a set of processor control registers. When a process is run on the processor, the execution environment is set up for that process and the program can use the execution environment to store and manipulate data.

The address space is a unsegmented, byte addressable 4 Gbyte space usable for data, instruction, and system data structures. For the HAP application the space has been subdivided into memory and peripheral sections. All global and floating point registers can be used for local data storage except g15. Register g15 is used to keep track of the current process data stored on one of the stacks. When a process switch or an interrupt service process switch occurs, the states of the global and floating point registers are preserved. A set of local registers is assigned to each active process. Three of the local registers are used to keep track of the previous process information, and the other local registers are available for general use. The registers available to a procedure are shown in Figure 6.2. The arithmetic control register is used to control the rounding mode of the processor, to set the conditions for an arithmetic fault to be generated, and to provide arithmetic operation status flags. The process control register controls the operating state of the processor and the priority of the executing process. An instruction pointer register keeps track of the current instruction being executed. The trace control register is used for software debugging, and is disabled for the HAP application.

The 80960MC provides a flexible method for automatically handling process calls and returns. Each procedure is automatically allocated a set of local registers and a frame on the stack. When an interrupt or process call occurs, the microprocessor automatically saves the local registers and the stack frame. A new set of local registers and a new stack frame is set up for the called procedure. The 80960MC has an on chip local register cache which can store 4 sets of local registers.

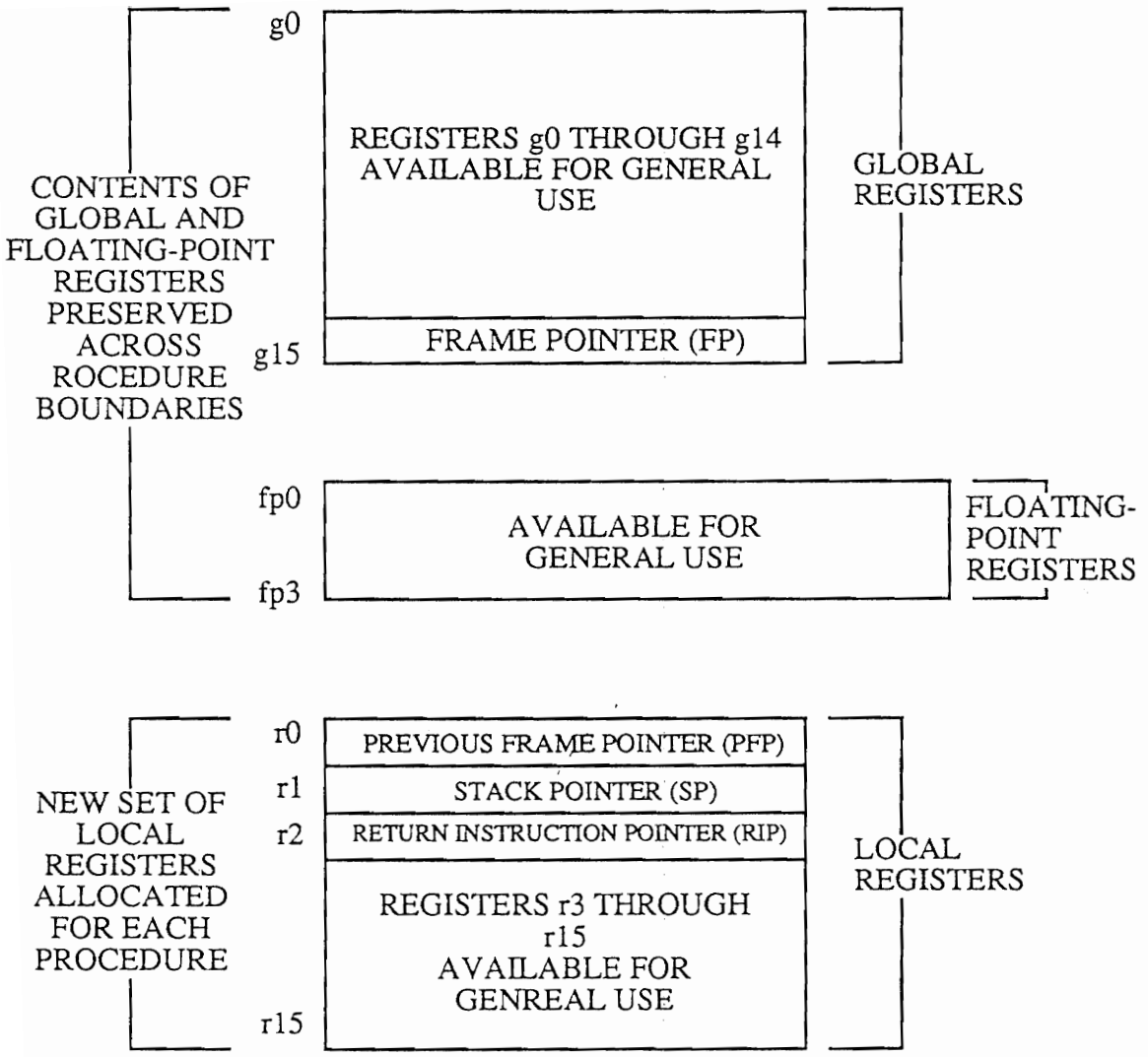


Figure 6.2 - Registers Available to a Single Procedure [13]

When the new process starts execution a set of local registers is assigned to it. If all of the on chip cached local register sets are valid, the least recently used cached local register set will be stored to the stack. If the new process was performed previously, the local registers will be recovered from the stack or the local on chip register cache. An interrupt stack is automatically maintained by the microprocessor and is the only stack used in the HAP application. If an interrupt is being serviced and a lower priority interrupt requests service, an interrupt request flag for that interrupt is placed in the interrupt table. The 80960MC automatically checks the pending interrupts in the interrupt table when a process return is performed. Pending interrupts will be serviced in order of priority. When all interrupt service requests have been completed, execution of normal processes are resumed.

6.3 HAP PROGRAM FLOW

Upon reset the microprocessor begins execution of my software shown in Figure 6.3. The software consists of several major blocks: initialization software, executive software, interrupt service routines, and fault service routines.

HAP software that I have written uses INTEL assembly language [27]. Assembly code was used to allow efficient code implementation. C-language has been used in the IAE interrupt handling routine for the mathematically intensive attitude propagation algorithm [28]. The C-language was used to efficiently code the attitude algorithms and to handle the multiple floating point variables. HAP software is contained within four 8K by 8 PROMs and is executed out of the PROMs.

HAP software is described below. A complete listing of the HAP software that I have written is provided in Appendix E. Appendix F contains the HAP C-language IAE data processing software that I have written.

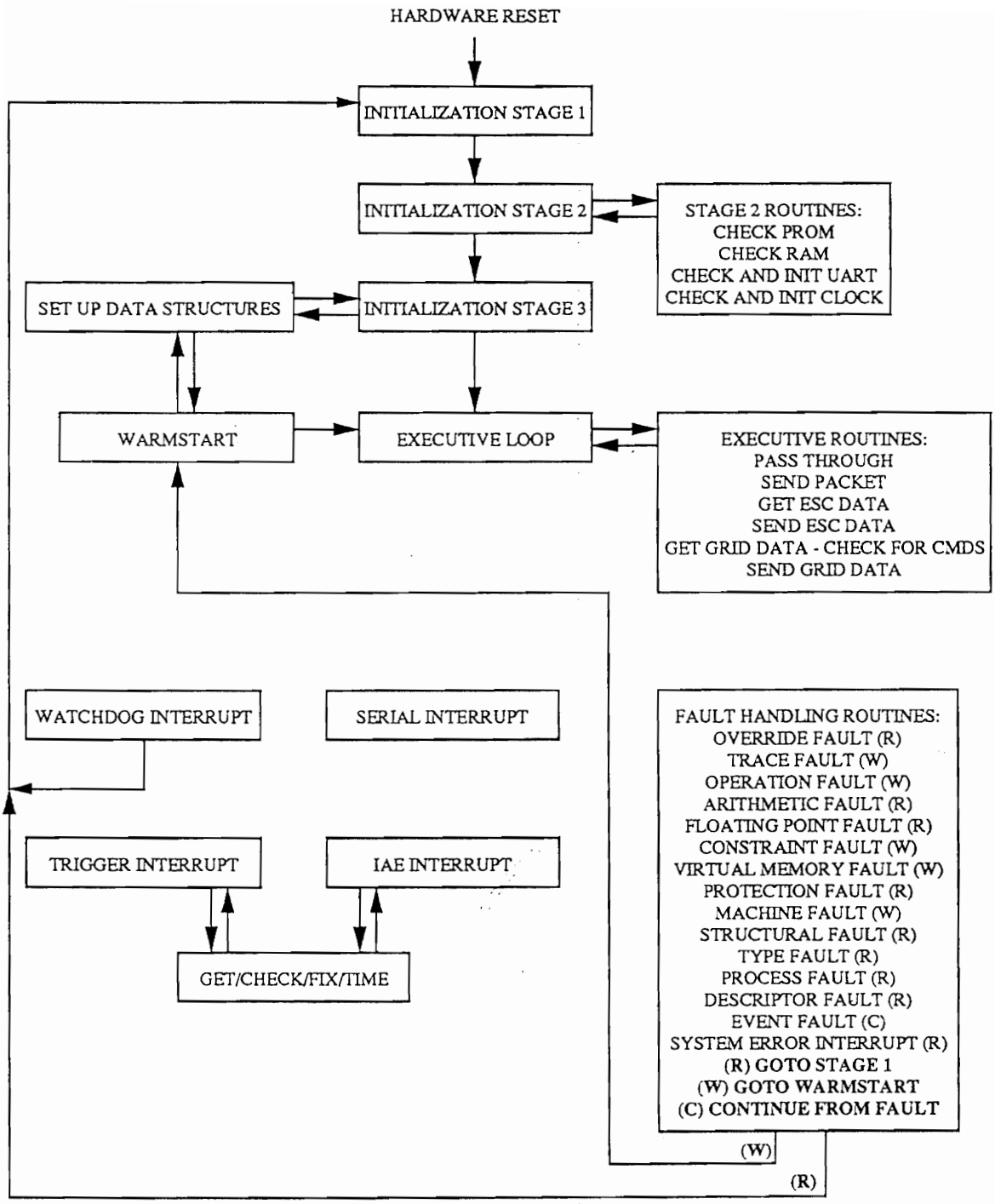


Figure 6.3 - HAP Program Flow

6.3.1 INITIALIZATION SOFTWARE

Whenever the 80960MC is reset, the HAP system will be initialized. Three stages of initialization occur. In the first stage of initialization, on-chip self-tests are performed and pointers and checksum data is read from the PROM. During the second stage of initialization, the HAP peripheral hardware is tested and configured. During the third stage of initialization, the microprocessor executes code to build the system data structures required by the microprocessor so execution of application code can begin.

6.3.1.1 FIRST STAGE INITIALIZATION

Upon reset, the microprocessor begins the first stage of initialization. The microprocessor performs an internal self-test. If this self-test is successful, the microprocessor reads the first eight words from PROM. The first eight words of PROM are called the Initial Memory Image (IMI), see Figure 6.4. A checksum is calculated on the IMI. If the checksum does not equal zero, then the microprocessor declares a failure, and halts execution. If the checksum equals zero then three of the first eight words of PROM provide pointers, as shown in Figure 6.5, to the initial system data structures.

The first pointer is a segment table pointer. A segment table is used to describe regions of memory for use by the memory management unit of the 80960MC. Even though the HAP design does not use memory management, there are required valid entries in the segment table which point to system data structures. Each entry in the segment table is called a segment selector. Valid segment selectors are required for the segment table, process control block (PCB), and other data structures.

The second pointer is a processor control block (PRCB) pointer.

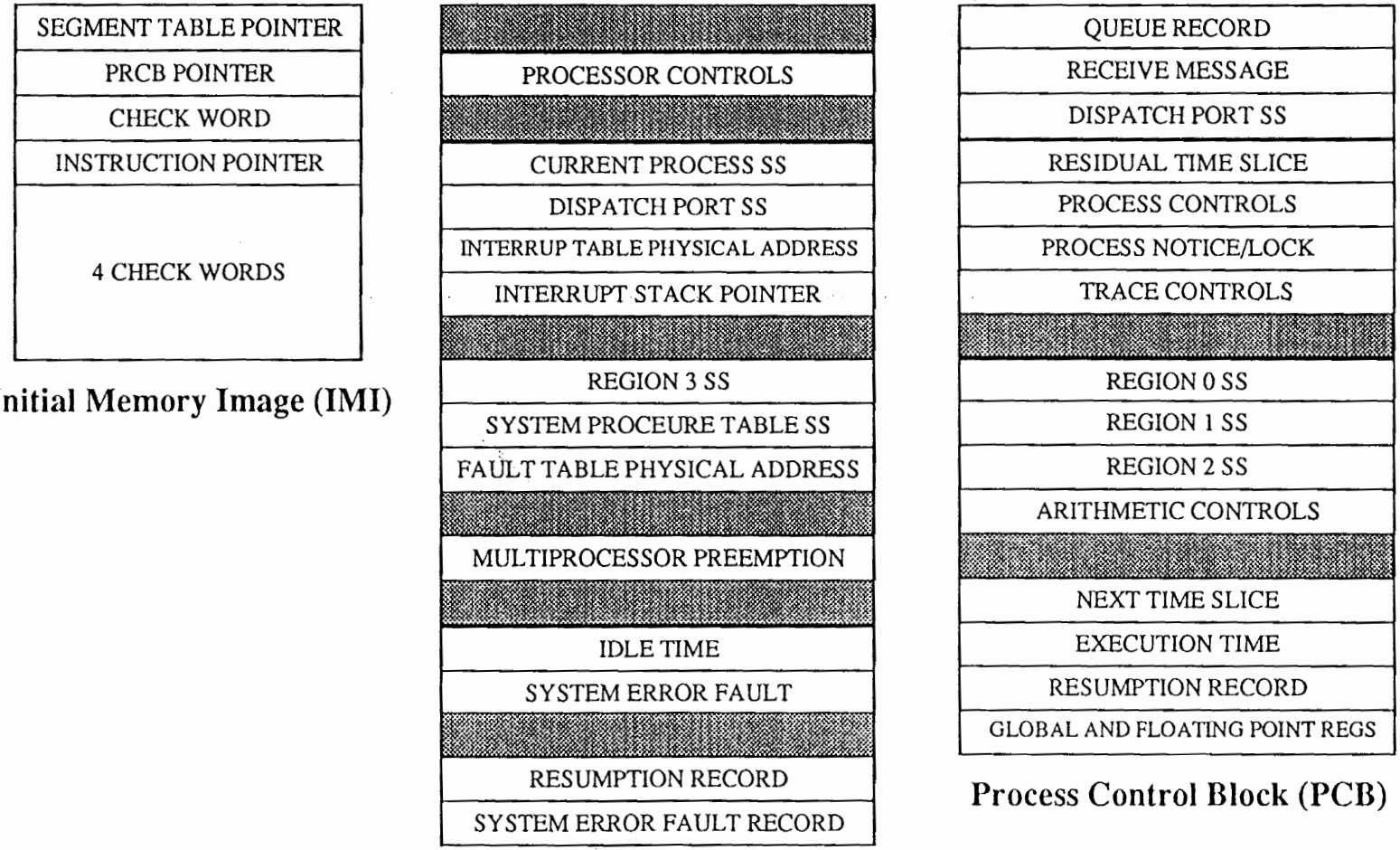


Figure 6.4 - Control Blocks [13]

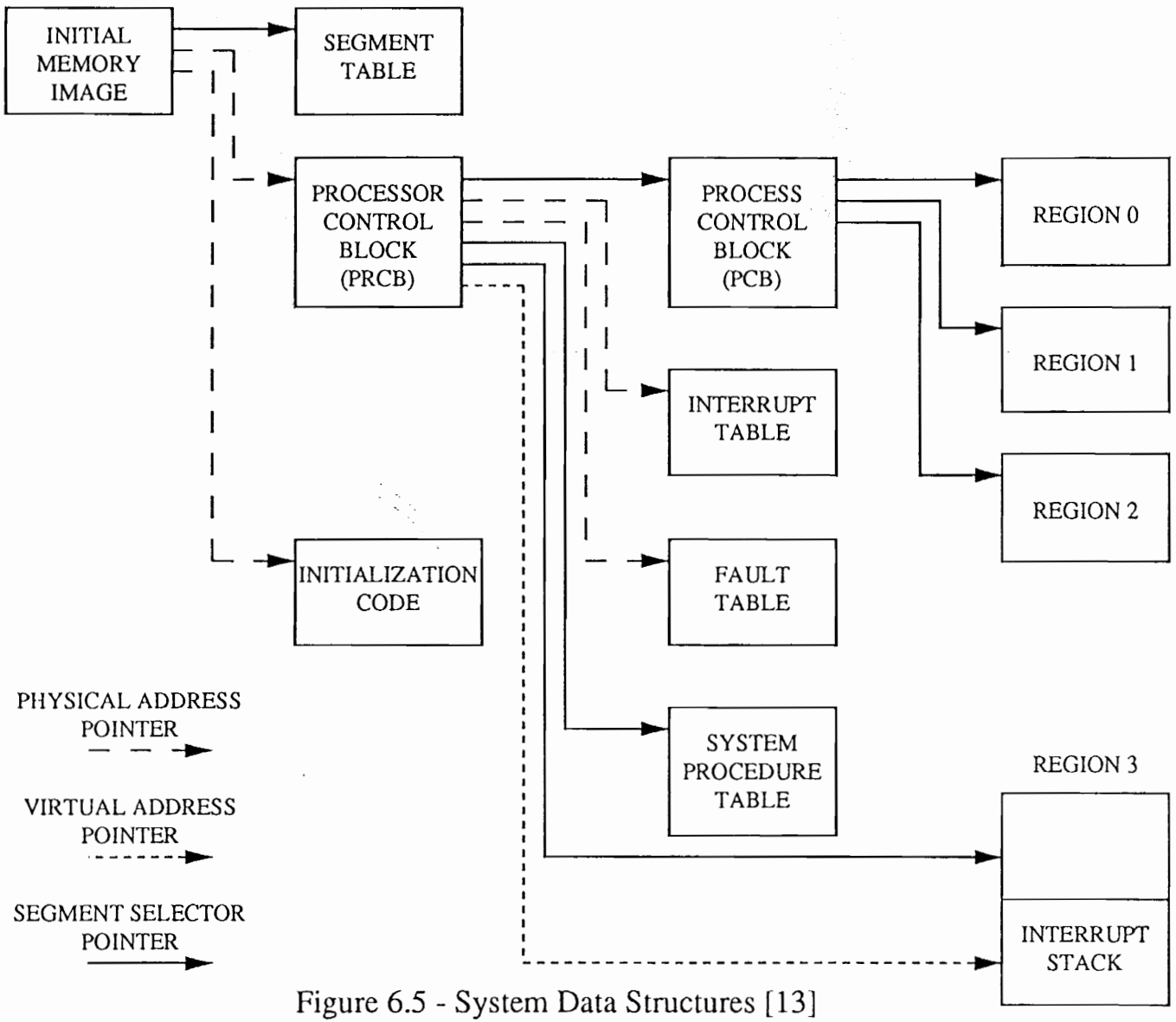


Figure 6.5 - System Data Structures [13]

The processor control block is a data structure that gives the microprocessor information about which modes of operation are enabled, and also contains pointers to other data structures, see Figure 6.4. A processor controls word, contained within the PRCB, identifies whether memory management should be enabled and whether there are multiple microprocessors in the system. HAP uses a single microprocessor and does not use memory management. Address pointers contained in the PRCB include: interrupt table address, interrupt stack address, and fault table address.

The third pointer points to the first instruction to be executed.

6.3.1.2 SECOND STAGE INITIALIZATION

After the first stage of initialization is complete, the microprocessor begins to configure and test the HAP peripherals as described below. The execution of code begins at the position indicated in the IMI.

The HAP interrupt enable register is first configured to disable all interrupts. Interrupts are enabled and disabled by writing a one or zero into the appropriate bit in the interrupt enable register.

PROM testing uses a simple additive checksum. Each word of PROM is added to form a 32 bit sum. This sum must equal zero for the PROM data to be correct. The sum is forced to zero by storing the 2's complement of the PROM data into the PROM. Since a binary number plus its 2's complement is zero, the PROM checksum will be zero. A PROM checksum must be generated and added to the code every time a change is made to the software. If the checksum does not equal zero, a "PROM checksum error flag" is set in the PGSC data packet status field and in the HAP flags. HAP flags are shown in Figure 6.6 and the PGSC data

Bit	Flags function	More_flags
0	PROM checksum error	Long command +@
1	RAM error	Long command +@
2	UART error	N rollover +
3	Watchdog interrupt enable	Fault occurred +
4	IAE interrupt enable	IAE monitor error +
5	Trigger interrupt enable	Mem/*Att frame +@
6	Serial interrupt enable	Mem/*Att frame +@
7	Bad access fault enabled	Interrupts enabled
8	CLOCK 0 error	Set clk next trig +@
9	CLOCK 1 error	Set clk next trig +@
10	CLOCK 2 error	Dump command +@
11	Time 0 bad but corrected+	Dump command +@
12	Time 1 bad but corrected +	Load command +@
13	Time 2 bad but corrected +	Load command +@
14	Time uncorrectable +	Ex. Instruction +@
15	ESCEB trigger detected +	Ex. Instruction +@
16	ESCEB trigger detected +	Invalid intr vect +
17	ESCEB power (0-on,1-off)	Override fault +
18	Reset occured +	Trace fault +
19	Watchdog expired +	Operation fault +
20	Battery low (0-good,1-bad)	Arithmetic fault +
21	Lost track of IMU +	Floating pt fault +
22	Clock setting armed +@	Constraint fault +
23	Memory unlocked +	Virtual mem fault +
24	Grid UART tx empty (b)	Protection fault +
25	Grid UART rx full (b)	Machine fault +
26	ESC UART tx empty (a)	Structural fault +
27	ESC UART rx full (a)	Type fault +
28	Copy packet	Process fault +
29	Copy packet	Descriptor fault +
30	Pass through or *Packet mode	Event fault +
31	Pass through or *Packet mode	Sys error intr +

HAP Interrupt Enable Register

Bit	function
0	PROM checksum error, working light
1	RAM error
2	UART error
3	Watchdog interrupt enable
4	IAE interrupt enable
5	Trigger interrupt enable
6	Serial interrupt enable
7	Bad access fault enable

Notes:

+ -> bit is cleared by the acknowledge flags command
@ -> bit is reset on all commands

Figure 6.6 - HAP Flags

packet is shown in Figure 6.7.

Every bit in the RAM is tested. A walking one pattern is written to every bit of every word in RAM. Each time the pattern is written, a read is performed to verify that the write has been successful. If there is any bit in RAM that does not respond correctly, a "RAM error" flag is set in the PGSC data packet status field and in the HAP flags.

Testing and configuration of both of the UARTs is performed next. Testing is performed using a walking one pattern and one of the UART scratch registers. If an error occurs, an "UART error flag" is set in the PGSC data packet status field and in the HAP flags. The UARTs are then configured to operate at 2400 baud, 8 bits, no parity and 1 stop bit. UART interrupt levels are set on the receiver FIFO to interrupt the processor when 14 bytes have arrived or there is a character in the FIFO for several character times. UART transmit FIFO is set to interrupt when it is empty. All configuration parameters are verified by reading the parameter values set. If there are any problems setting a parameter, the "UART error flag" is set in the PGSC data packet status field and in the HAP flags. A greeting message is sent out of both UARTs, which identifies the HAP software version being run.

The three UTC clock chips are tested and programmed simultaneously. Clock chip hardware has been designed to place each of the clock chips in a different byte of the same address. Clock 0 is in byte 0, clock 1 is in byte1, and clock 2 is in byte 2. The clock chips include the UTC clock, general purpose nonvolatile ram, and timers. A walking one test is used to verify that the clock chips are responding correctly to the microprocessor. If there is an error, the clock chip that caused the error is determined and the appropriate "clock error" flags are set in the PGSC data packet status field and in the HAP flags.

The watchdog timers are set up next. Watchdog timers are used to detect

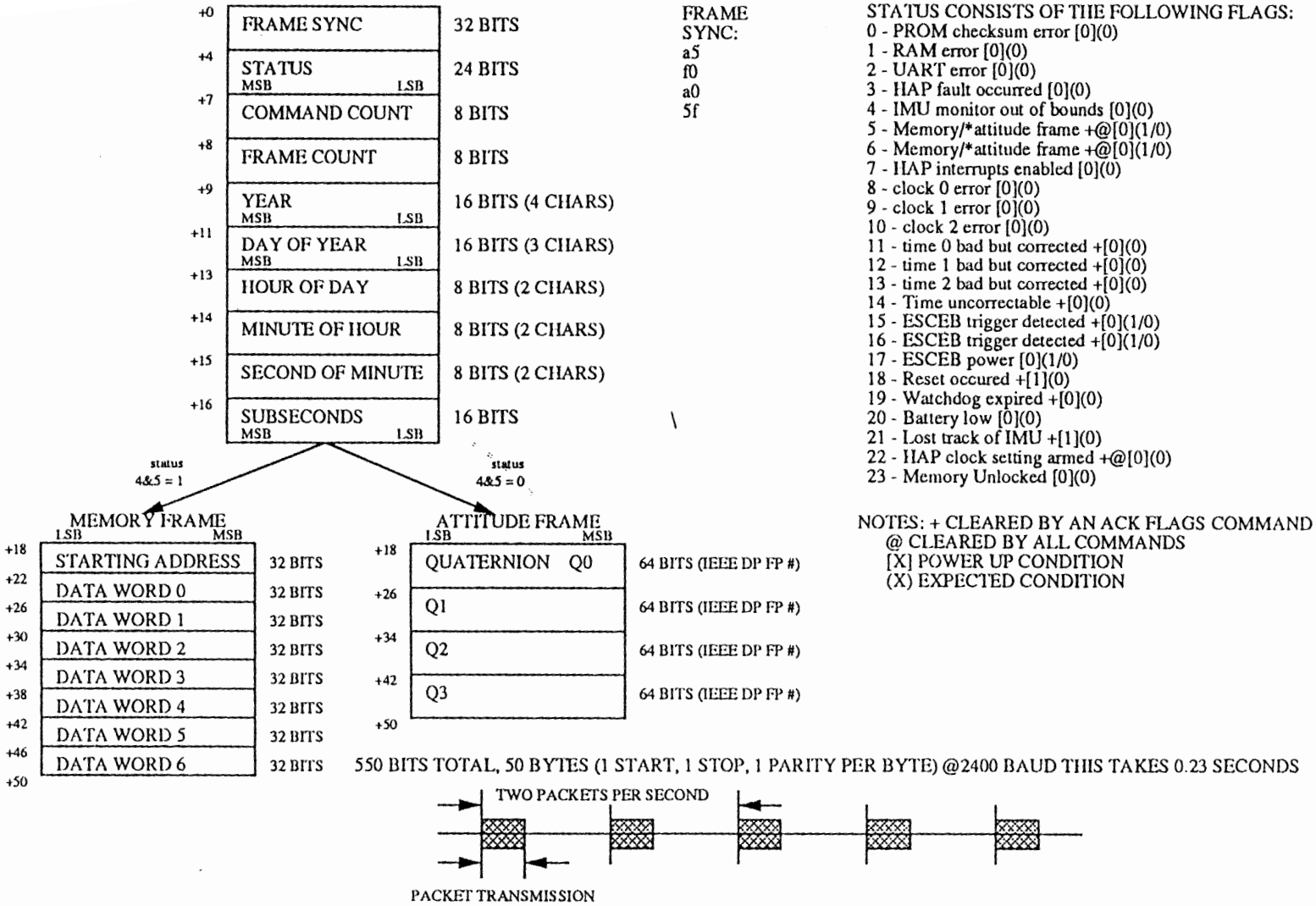


Figure 6.7 - HAP Output Data Packet

abnormal execution times, or errant program flow [38]. Every time IAE data is gathered, all watchdog timers are restarted. There are two levels of watchdog timers. If the first level watchdog timer expires, then the software performs a warmstart. Warmstart activity will be discussed later. If the second level watchdog timer expires then a HAP hardware reset is activated and execution restarts at initialization stage 1. Timer initialization begins with timer hardware configuration. Timer outputs are set to the proper polarity and sent to the proper output pins. The first level of watchdog timer is configured to interrupt the processor if it is not restarted within 5 milliseconds, or 10 IAE interrupts. The second level of watchdog timer is configured to reset HAP if it has not been restarted within 1.979 seconds, which is the maximum timer period. A long time was chosen for the second level of watchdog timer to allow a restart processor fault recovery to complete if one was active. If any errors were detected during the configuration process, the clock chip that caused the error is determined and the appropriate "clock error" flags are set in the PGSC data packet status field and in the HAP flags.

6.3.1.3 THIRD STAGE INITIALIZATION

During the third stage of initialization, the microprocessor executes code to build the system data structures required by the microprocessor so execution of application code, and servicing of interrupts can begin. Figure 6.5 shows the system data structures. The required data structures are the IMI, segment table, PRCB, PCB, interrupt table, fault table, and interrupt stack.

The IMI and segment table are located in PROM and were set up in initialization stage 1. A temporary PRCB and PCB was also set up in PROM for the first stage of initialization.

To ensure proper operation of the interrupt and fault handling process, the

PRCB and PCB have to be relocated to RAM. The processor uses the PRCB to store information about the process being interrupted, if the PRCB is in PROM this information can not be stored and the interrupted process will not be recoverable. The processor uses the PCB to provide information about the process being executed. If the PCB is in PROM the processor will only be able to run a single process and will not be able to recall the information necessary to recover from an interrupt. I have written the third stage of initialization to move the PRCB and the PCB into RAM. When the PRCB is moved into RAM the segment selector for the PCB must be changed to point to the PCB that is in RAM. The fault table pointer is also changed to point to the RAM based fault table. The processor controls word in the PRCB was modified to put the processor into the processor executing state. The PCB is not modified when it is moved into RAM.

The interrupt activities require the use of an interrupt stack. The interrupt stack is used to save the local register set for the interrupted process. I initialize the interrupt stack to enable correct interrupt servicing, and have allowed enough space on the interrupt stack to accommodate 16 nested interrupts. There are a maximum of four nested interrupts in the HAP application, because there are only four interrupt sources.

The interrupt table needs to be located in RAM. Pointers or vectors to the interrupt routines are contained in the interrupt table. The priority of the interrupt is determined by the location of the interrupt vector in the interrupt table. The interrupt priority is the vector number divided by 8. There are 32 priorities (0 - 31) and 256 vectors (0 - 255). The first 8 entries of the interrupt table provide pending interrupt information to the microprocessor and needs to be writable. The first 32-bit entry is used to indicate what priority interrupts are pending. Each bit position

that is set indicates that an interrupt with that priority is pending. The next 7 entries are used to record the vectors that are requesting service. Each bit position that is set indicates an interrupt vector that requires service. When the processor receives an interrupt, it temporarily stops work on its current process. It then reads the interrupt vector from the interrupt register and compares the priority of the vector with the priority of the current process that is being executed. If the priority of the new interrupt is higher than the current process, the processor services the new interrupt immediately after storing the current process local registers to the interrupt stack. If the interrupt priority is equal to or less than the current process, the processor records the new interrupt in the pending interrupt record and continues work on the current process. When a process is returned from, the pending interrupt record is inspected, and the interrupt with the highest priority is serviced next. If no interrupts are pending, the processor executes the originally interrupted process.

In the third stage of initialization, the pending interrupt record and the reserved interrupt vectors are set to zero. All unused interrupt vectors are set to point to a processor warmstart, in case an interrupt vector is somehow calculated incorrectly by the processor. Serial interrupts have been assigned a priority of 5, and the interrupt vector is located in entry 40. IAE interrupts have assigned given a priority of 7, and the interrupt vector is located in entry 56. Trigger interrupts have also been assigned a priority of 7, and the interrupt vector is located in entry 58. The watchdog timer has been assigned the priority of 31, and the interrupt vector is located in entry 254. The interrupt vector for the required system error interrupt fault is located in entry 248.

The executive process has a priority of 3. The serial interrupt was assigned a priority of 5 to allow it to be handled before the executive tasks but after any other

interrupt tasks. The IAE and trigger interrupts were assigned the same priority of 7 to allow the IAE interrupt service routine to complete the processing of the IAE data before a trigger interrupt was serviced. If a trigger interrupt occurs before, after, or during a service of the IAE interrupt, the trigger interrupt service routine will always have access to fully processed attitude data. The watchdog timer interrupt service routine runs at the highest priority of 31, to guarantee that the interrupt will be serviced regardless of other processes that are executing.

The four microprocessor external interrupt pins can be configured as direct interrupt pins. In the HAP application, I use the interrupts as direct interrupt pins. To map the physical pin to an interrupt vector, the processor uses an interrupt control register. I initialize this interrupt control register to point to the serial, IAE, trigger, and watchdog timer interrupt vectors described above. I have used the required special command sequence to program this register.

I move the fault table from PROM into RAM. If the fault table is in ram, the pointers in the fault table will be able to be modified.

Once all the system data structures have been moved into RAM, the processor has to be commanded to start using the RAM based system data structures. The processor is commanded using Interagent Communication (IAC) messages. To initiate an IAC message transfer, the four word IAC message is loaded into memory, then a special move command moves the four word message into the processor for execution. The restart processor IAC message is needed to get the processor to reinitialize all of its system data structures to the new ones specified in the IAC message. After the processor has accepted the new system data structures, it begins execution of the executive.

6.3.2 EXECUTIVE

Overall operation of the system needs to be coordinated. Software tasks in HAP are managed by an executive that I have designed. Processes communicate with the executive, using flags, to schedule services. An executive is a simple operating system. All data variables used by the executive are shown in Figure 6.8.

6.3.2.1 EXECUTIVE INITIALIZATION

The first task of the executive is to initialize all of the HAP pointers, data, and variables. I initialized the following items: serial buffer pointers, command decoding variables, IAE data processing variables and data, watchdog timers, and HAP interrupt enable register. An executive initialization warmstart location is located after the pointer, data and variable initialization. Use of this warmstart location is discussed later.

I have included a hardware counter on the IAE interrupt line. During the executive initialization process, this counter is read and the value is stored in a software counter variable. Every time the IAE interrupt is serviced, the hardware IAE interrupt counter will be read and compared to the software IAE interrupt counter. If the counters do not match, then the processor did not service one of the IAE interrupt requests and a flag can be set indicating lost track of IMU.

Initialization of two IAE configuration registers occurs next. The watchdog timers are started. The HAP interrupt enable register is set to allow the interrupts to come through.

6.3.2.2 EXECUTIVE LOOP

I have used the executive loop to monitor HAP, IAE, and IMU status and to schedule services. Figure 6.9 shows the program flow for the HAP executive loop.

```
flags, 0x01017000  /* 1 word */
more_flags, 0x01017004  /* 1 word */
fault_record, 0x01017008  /* 5 words */
hap_sw_version,0x01017200  /* 4 words */

iae_monitor_points, 0x01017800  /* 21 short, 11 words */
iae_over_flag, 0x0101782c  /* 1 word */
iae_under_flag, 0x01017830  /* 1 word */
iae_over_points, 0x01017840  /* 21 short, 11 words */
iae_under_points, 0x01017860  /* 21 short, 11 words */
iae_discrete_points, 0x01017900  /* 3 short, 2 words */
mux_item_count, 0x01017f00  /* 1 byte, 1 word */
iae_sample_count, 0x01017f80  /* 1 byte, 1 word */

normal_packet, 0x01018000  /* 50 bytes, 13 words */
trigger_packet, 0x01018100  /* 50 bytes, 13 words */
memory_packet, 0x01018200  /* 50 bytes, 13 words */
frame_count, 0x01018700  /* 1 word */
solution_time, 0x01018800  /* 9 words */
trigger_time, 0x01018900  /* 9 words */
desired_time, 0x01018f00  /* 9 words */

grid_tx_buffer, 0x01019000  /* 256 bytes, 64 words */
grid_rx_buffer, 0x01019100  /* 256 bytes, 64 words */
esc_tx_buffer, 0x01019200  /* 256 bytes, 64 words */
esc_rx_buffer, 0x01019300  /* 256 bytes, 64 words */
grid_tx_head, 0x01019400  /* 1 byte, 1 word */
grid_tx_tail, 0x01019404  /* 1 byte, 1 word */
grid_rx_head, 0x01019408  /* 1 byte, 1 word */
grid_rx_tail, 0x0101940c  /* 1 byte, 1 word */
esc_tx_head, 0x01019410  /* 1 byte, 1 word */
esc_tx_tail, 0x01019414  /* 1 byte, 1 word */
esc_rx_head, 0x01019418  /* 1 byte, 1 word */
esc_rx_tail, 0x0101941c  /* 1 byte, 1 word */
command_buffer, 0x01019500  /* 12 bytes, 3 words */
command_count, 0x01019600  /* 1 byte, 1 word */
long_command_character_count, 0x01019604  /* 1 byte, 1 word */

_pin, 0x0101a000  /* 3 words */
_a, 0x0101a100  /* 39 words */
_pfda, 0x0101a200  /* 3 double, 6 words */
_pb, 0x0101a300  /* 3 double, 6 words */
_tho, 0x0101a400  /* 4 double, 8 words */
_current_solution, 0x0101a500  /* 4 double, 8 words */
_solution_count, 0x0101a600  /* 1 word */
iae_interrupt_count, 0x0101a604  /* 1 byte, 1 word */
heartbeat, 0x0101a608  /* 1 byte, 1 word */
```

NOTE: Information is presented in the format: name, address, and size.

Figure 6.8 - HAP Executive Data Variables and Locations

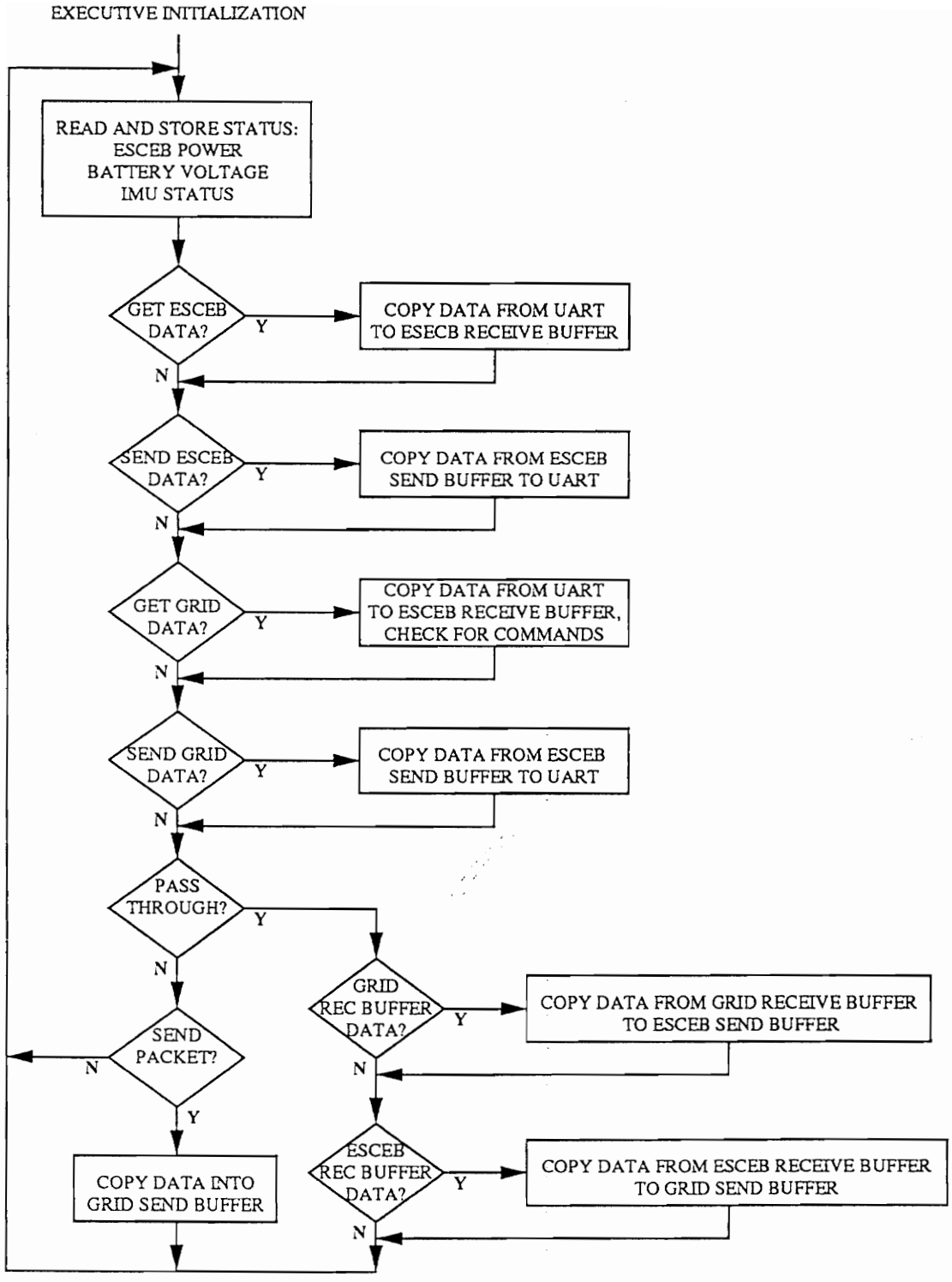


Figure 6.9 - HAP Executive Loop

6.3.2.2.1 STATUS GATHERING

The state of the ESC power is determined by reading the clock chips and looking at the most significant byte. This byte is used as general inputs for HAP. Bit 28 has the state of the ESC, a one means the ESC is off, and a zero means the ESC is on. A flag is set in the HAP flags to indicate the "ESC power" state.

The HAP battery is also determined by reading the clock chip address space and looking at the most significant byte. Bit 30 indicates whether the battery voltage is below the allowable limit or if it is acceptable. A one indicates that the battery is low, and a zero indicates that the battery is good. A flag is set in the HAP flags to indicate the "battery state."

IAE and IMU status is also determined. There are 21 analog monitor points associated with the gyro. Each of these monitor points needs to be examined and compared to known limits. The analog values are converted into digital values in the IAE. The IAE begins the conversion process based on commands from HAP. A conversion takes several microseconds, and the IAE provides a flag to indicate when a conversion is complete. The HAP begins the conversion and then goes to the service check section of the executive loop. Each time through the executive loop, the IAE conversion complete flag is checked. If the conversion is complete the converted value is read from the IAE and stored in the HAP memory area allocated for IAE monitor points. My software then reads the preset minimum and maximum limits for that data point and verifies that it is within limits.

If any of the monitor points are above or below the acceptable limits, several items are stored. An "IMU monitor out of bounds" flag is set in the HAP flags to indicate a problem. A word in the HAP memory is allocated to indicate which IAE item has exceeded its limits. Each bit of that word is used to indicate a different IAE monitor point out of limits. The appropriate bit in this memory location is set

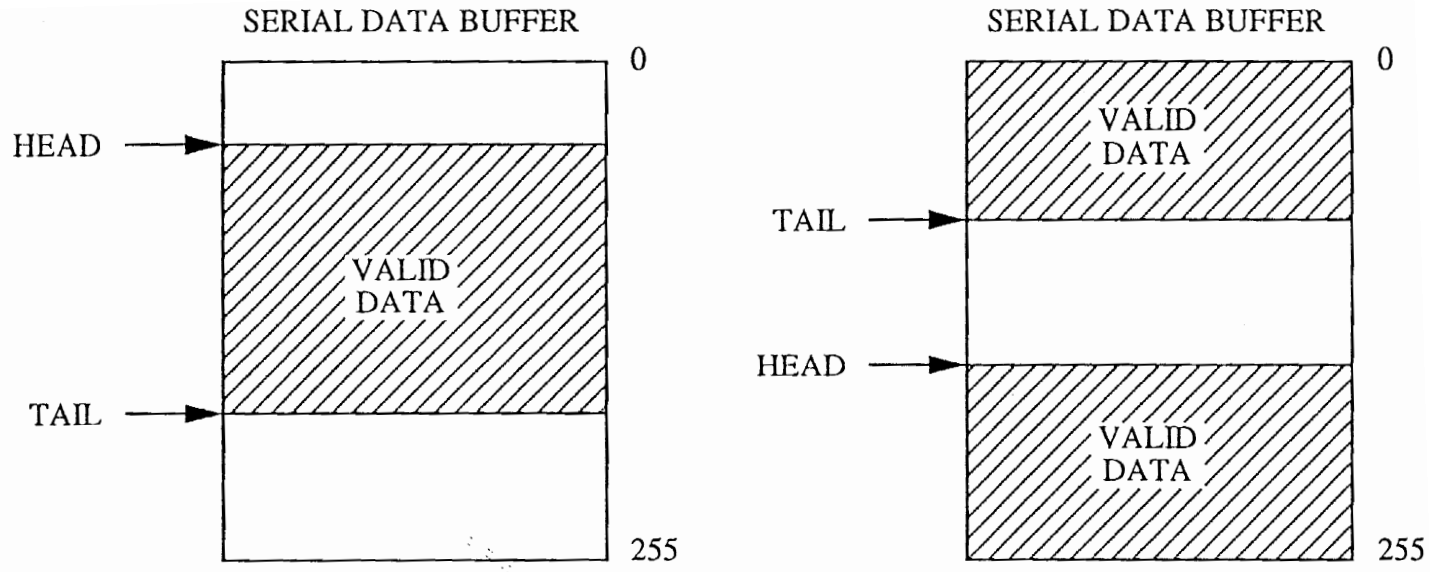
to identify the IAE monitor point in error. The value of the out of limit IAE monitor point reading is also stored in HAP memory for later retrieval. The PGSC periodically requests the IAE monitor point values and the out of limits IAE monitor point data.

The IAE also has three digital status and configuration registers. These IAE registers are read and stored in HAP memory every time a new IAE monitor point is stored.

6.3.2.2.2 CHECKING SERVICE REQUESTS

Services are requested by setting flags in the HAP flags shown in Figure 6.6. Critical flags are redundant, and the service routines verify that the redundant flags are both set before acting on them. The ESC UART flags indicate whether data needs to be read from the ESC UART or whether the ESC UART is available to accept data from the executive. The GRiD UART flags indicate whether data needs to be read from the GRiD UART or whether the GRiD UART is available to accept data from the executive. The "ESCEB trigger detected", "copy packet", "pass through or packet mode", and the "memory or attitude packet" HAP flags are used to determine what is done with serial data.

All serial data is stored in ring buffers, as shown in Figure 6.10. There are four 256 byte buffers, one for the ESC UART data received, one for the ESC UART data to transmit, one for the GRiD UART data received, and one for the GRiD UART data to transmit. Each buffer is operated using the following technique. A head pointer and a tail pointer are used to indicate the oldest and newest entry respectively in the buffer. Initially the head and tail pointers are set equal to each other. When the pointers are equal, the buffer is empty. A byte is written into



ADD DATA TO THE BUFFER USING THE TAIL POINTER AS AN OFFSET FROM THE FIRST WORD IN THE BUFFER, AND INCREMENT TAIL POINTER (MODULO 255)

REMOVE DATA FROM THE BUFFER USING THE HEAD POINTER AS AN OFFSET FROM THE FIRST WORD IN THE BUFFER, AND INCREMENT THE HEAD POINTER (MODULO 255)

IF HEAD = TAIL THE BUFFER IS EMPTY
IF INCREMENTING THE TAIL MAKES THE HEAD = TAIL THAT IS AN OVERFLOW CONDITION

Figure 6.10 - Buffer Format and Pointer Useage

the ring buffer at the location specified by the sum of the beginning of the buffer plus the tail pointer value. After a byte has been stored in the ring buffer, the tail pointer value is incremented. If incrementing the tail pointer caused the tail pointer and head pointer values to equal, an overflow has occurred. I have written the HAP software to detect an overflow condition and to dump the oldest byte in the ring buffer to prevent an overflow. A byte is read from the ring buffer from the location specified by the sum of the beginning of the buffer plus the head pointer value. After a byte has been read from the ring buffer, the head pointer value is incremented.

6.3.2.2.2.1 GET ESC DATA SERVICE

The flag requesting this service is set by the serial interrupt service routine. If this flag is set, it indicates that the ESC UART has received characters. One character takes one byte of memory. My routine reads the ESC UART receive FIFO and then stores the character in the ESC receive ring buffer. The ESC UART line status register is checked to see if the UART receive FIFO is empty. If the FIFO is not empty, the FIFO is read and rechecked until all characters have been read from the ESC UART receive FIFO and stored in the ESC receive ring buffer. When writing to the ring buffer, potential overflows are detected and the head pointer value is incremented to dump the oldest character in the ring buffer and prevent the overflow. When the ESC UART receive FIFO has been emptied, the "ESC UART receive FIFO full" flag of HAP flags is reset, and the routine jumps back to the next check in the executive loop.

6.3.2.2.2.2 SEND ESC DATA SERVICE

The flag requesting this service is set by the serial interrupt service routine.

If this flag is set, it indicates that the ESC UART can send characters. My routine checks the ESC transmit ring buffer for characters to send. If there are no characters to send, the routine jumps back to the next check in the executive loop without resetting the "ESC UART transmit empty" flag of HAP flags. If there is data to send, the routine copies up to 16 characters from the ESC UART transmit ring buffer to the ESC UART transmit FIFO. The GRiD UART transmit FIFO can only hold 16 characters. The routine reads a character from the ring buffer and writes it to the ESC UART transmit FIFO. If there is another character in the ring buffer and 16 characters have not been written into the ESC UART transmit FIFO, another character will be written to the ESC UART transmit FIFO. When no more characters can be written, the "ESC UART transmit empty" flag of HAP flags is reset and the routine jumps back to the next check in the executive loop.

6.3.2.2.2.3 GET GRID DATA SERVICE

The flag requesting this service is set by the serial interrupt service routine. If this flag is set, it indicates that the GRiD UART has received characters. My routine reads the GRiD UART receive FIFO, checks for a valid command, and then stores the character in the GRiD receive ring buffer. After processing any valid commands, the GRiD UART line status register is checked to see if the UART receive FIFO is empty. If the FIFO is not empty, the FIFO is read and rechecked until all characters have been read from the GRiD UART receive FIFO, checked for valid commands, and stored in the GRiD receive ring buffer. When writing to the ring buffer, potential overflows are detected and the head pointer value is incremented to dump the oldest character in the ring buffer and prevent the overflow. When the GRiD UART receive FIFO has been emptied, the "GRiD

UART receive FIFO full" flag of HAP flags is reset, and the routine jumps back to the next check in the executive loop.

There are 10 valid HAP commands, shown in Figures 6.11 and 6.12. Valid commands consist of a four byte prefix and an optional 8 byte data field. Seven of the 10 commands consist of the four byte prefix only. Three of the 10 commands consist of the four byte prefix and the 8 byte data field. A command is detected by comparing the last four characters received from the GRiD UART receive FIFO against the 10 valid command prefixes. Each time a character is read from the FIFO, a check for valid command prefixes is performed. If the last four characters received match one of the command prefixes, the command is processed. A command counter is incremented every time a command is executed. The command counter is sent to the PGSC in the data packet shown in Figure 6.3. When any command is executed, the HAP flags identified in Figure 6.6 are cleared.

The NORMAL command is used to set the HAP operating mode. When this command is received, the HAP flags are modified to cause the executive to send PGSC attitude data packets as shown in Figure 6.3. The four ring buffers are flushed, the UART FIFOs are flushed, and the command counter is incremented when this command is executed.

The PASS THROUGH command is used to allow the PGSC to communicate with the ESCEB. When this command is received, the HAP flags are modified to allow this communication path. Sending of data packets to the PGSC is suspended when the HAP is operating in the pass through mode. The four ring buffers are flushed, the UART FIFOs are flushed, and the command counter is incremented when this command is executed.

The ACKNOWLEDGE FLAGS command is used to clear warning and status flags in the PGSC data packet. Figures 6.6 and 6.7 show which flags are

• NORMAL:
CODE - cntl<NORM> (0e 0f 12 0d)h
Format and send attitude packets 2 times per second
Listen for commands

• PASS THROUGH:
CODE - cntl<XFER> (18 06 05 12)h
Pass data between GRiD and ESCEB
Listen for NORMAL and ACK FLAGS commands only

• ACKNOWLEDGE FLAGS:
CODE - cntl<ANFG> (01 0e 06 07)h
Clears bits in status word, see the packet format diagram

• SET CLOCKS ON NEXT TRIGGER:
CODE - cntl<SNXT> (13 03 18 14)h
Jam time previously stored in the HAP memory (location desired_time) into the clocks and start the clocks on the next trigger.
If the contents of desired_time do not have the proper format the command will be ignored.

• SET CLOCKS NOW:
CODE - cntl<SCKI> (13 03 0b 09)h
Jam time previously stored in the HAP memory (location desired_time) into the clocks and start the clocks immediately.
If the contents of desired_time do not have the proper format the command will be ignored.

Figure 6.11 - HAP Commands

• DUMP MEMORY:
CODE - cntl<DMBK> (04 0D 02 0B)h
FORMAT - (04 0D 02 0B Byte-0 Byte-1 Byte-2 Byte-3 Byte-0 Byte-1 Byte-2 Byte-3)h 12 chars long
Dump the contents of the HAP starting at the address Byte-0,Byte-1,Byte-2,Byte-3.
Byte-0 is the address lsb, and Byte-3 is the address msb.

• LOAD MEMORY:
CODE - cntl<LMWD> (0c 0d 17 04)h
FORMAT - (0c 0d 17 04 Abyte-0 Abyte-1 Abyte-2 Abyte-3 Dbyte-0 Dbyte-1 Dbyte-2 Dbyte-3)h
12 chars long
Load the HAP at address Abyte-0, Abyte-1, Abyte-2, Abyte-3 with data Dbyte-0, Dbyte-1,
Dbyte-2, Dbyte-3. Byte-0 is the lsb and Byte-3 is the msb.
HAP must be unlocked or this command will be ignored.

• UNLOCK HAP:
CODE - cntl<UNLK> (15 0e 0c 0b)h
Sets the HAP memory unlocked bit and enables the Grid to use the LOAD MEMORY or EXECUTE
INSTRUCTION command.

• LOCK HAP:
CODE - cntl<LOCY> (0c 0f 03 19)h
Clears the HAP memory unlocked bit and disables the Grid from using the LOAD MEMORY or
EXECUTE INSTRUCTION command.

• EXECUTE INSTRUCTION:
CODE - cntl<XIAD> (18 09 01 04)h
FORMAT - (18 09 01 04 Byte-0 Byte-1 Byte-2 Byte-3 Byte-0 Byte-1 Byte-2 Byte-3)h 12
characters long. Byte-0 is the address lsb, and Byte-3 is the address msb.
Causes a jump to the specified address, HAP must be unlocked or this command will be ignored.

Figure 6.12 - HAP Commands Continued

cleared by this command. The command counter is incremented when this command is executed.

The SET CLOCKS ON NEXT TRIGGER command is used to set the three UTC clocks using a trigger pulse from the GPS receiver as shown in Figure 5.16. The time setting procedure requires that valid time setting data be loaded into HAP memory before setting the clock. The valid time setting data contains the time that the clocks will be set to. A SET CLOCKS ON NEXT TRIGGER command will not be accepted when the HAP is operating in the PASS THROUGH mode. My routine checks the time setting data to verify that it is valid. Valid time setting data is identified by a hex "ff" in the most significant byte of each time setting data item. If any of the time setting data items are not valid, then the command is rejected and the command counter is not incremented and the clock will not be set. If the time setting data is valid, the "set clocks on next trigger" HAP flags will be set so the trigger interrupt service routine will set the clocks the next time it is executed. Once the time has been set, the time setting data is invalidated by clearing the most significant byte of each time setting data item. The command counter will be incremented if the time setting data is valid.

The SET CLOCKS NOW command is used to set the three UTC clocks. This command will not be accepted when the HAP is operating in the PASS THROUGH mode. My routine checks the time setting data to verify that it is valid. If any of the time setting data items are not valid, then the command is rejected and the command counter is not incremented and the clock will not be set. If the time setting data is valid, my routine will boost the executive process priority to 16, configure the clocks, load the time setting data into the clocks, start the clocks, and restore the executive process priority to 3. Once the time has been set, the time setting data is invalidated by clearing the most significant byte of each

time setting data item. The executive process priority is boosted above the priority of the IAE and trigger interrupt service routines priority to allow the UTC time to be set without the possibility of another process interrupting the setting procedure. If the time setting procedure is interrupted, the time may be set incorrectly. The command counter is incremented after a valid time setting process has completed.

The DUMP MEMORY command causes the HAP to format and send HAP memory information to the PGSC in a memory data packet as shown in Figure 6.7. If the HAP is operating in the PASS THROUGH mode, the command will not be accepted. A DUMP MEMORY command is a 12 byte long command. Flags are set, once the command prefix is recognized, which will collect the next 8 characters of the command. As shown in Figure 6.12, the 8 characters in the data field specify the memory address of the first data item in the PGSC memory data packet. When all 12 characters of the DUMP MEMORY command have been received, my routine gathers all of the data necessary to make a PGSC memory data packet. The memory data area of the packet echoes the address sent in the DUMP MEMORY command and the 7 consecutive data values starting at the commanded address. When the memory data packet is fully gathered, HAP flags are set to allow the memory data packet to be sent to the PGSC, and the command counter is incremented. The same memory data packet will be sent to the PGSC until another command is accepted by the HAP.

The LOAD MEMORY command is used by the PGSC to write data to the HAP memory. Data needs to be written to the HAP memory to set the UTC clocks. If the HAP is operating in the PASS THROUGH mode, or if the UNLOCK HAP command has not been sent, the command will not be accepted. The LOAD MEMORY command is a 12 byte long command. Flags are set, once

the command prefix is recognized, which will collect the next 8 characters of the command. As shown in Figure 6.12, the 8 characters in the data field specify the memory address to modify and the value to place in that memory address. When all 12 characters of the LOAD MEMORY command have been received, my routine loads the commanded data into the commanded address, resets the HAP flags, and increments the command counter.

The UNLOCK HAP command is used to enable the LOAD MEMORY and EXECUTE INSTRUCTION commands. This command sets a bit in the HAP flags and then increments the command counter.

The LOCK HAP command is used to disable the LOAD MEMORY and EXECUTE INSTRUCTION commands. This command clears a bit in the HAP flags and then increments the command counter.

The EXECUTE INSTRUCTION command is used to make the HAP start executing instructions at a new location. If the HAP is operating in the PASS THROUGH mode, or if the UNLOCK HAP command has not been sent, the command will not be accepted. The EXECUTE INSTRUCTION command is a 12 byte long command. HAP flags are set, once the command prefix is recognized, which will collect the next 8 characters of the command. As shown in Figure 6.12, the 8 characters in the data field specify the address that HAP should jump to. When all 12 characters of the command have been received, the command counter is incremented, the HAP flags are reset, and the program execution branches to the commanded address.

6.3.2.2.2.4 SEND GRID DATA SERVICE

The flag requesting this service is set by the serial interrupt service routine. If this flag is set, it indicates that the GRiD UART can send characters. My routine

checks the GRiD transmit ring buffer for characters to send. If there are no characters to send, the routine jumps back to the next check in the executive loop without resetting the "GRiD UART transmit empty" flag of HAP flags. If there is data to send, the routine copies up to 16 characters from the GRiD UART transmit ring buffer to the GRiD UART transmit FIFO. The GRiD UART transmit FIFO can only hold 16 characters. The routine reads a character from the ring buffer and writes it to the GRiD UART transmit FIFO. If there is another character in the ring buffer and 16 characters have not been written into the GRiD UART transmit FIFO, another character will be written to the GRiD UART transmit FIFO. When no more characters can be written, the "GRiD UART transmit empty" flag of HAP flags is reset and the routine jumps back to the next check in the executive loop.

6.3.2.2.3 EXECUTIVE PASS THROUGH MODE

When the HAP is operating in the pass through mode, serial communications are enabled between the ESCEB and the PGSC. The HAP acts as an intermediate transfer stage for the data. All data received from the PGSC is checked for commands, and then sent to the ESCEB. All data received from the ESCEB is sent to the PGSC. This connection is required so the ESCEB can store the geolocation data with the image, and the PGSC can record the ESCEB hard disk number and frame number associated with the last geolocation.

If the GRiD receive ring buffer has data, my routine takes the data out of the GRiD receive ring buffer and puts it into the ESC transmit ring buffer. If the ESC receive ring buffer has data, it will be moved out of the ESC receive ring buffer and put into the GRiD transmit ring buffer. After moving all of the data from one buffer to another, the routine jumps to the top of the executive loop.

Data transfer occurs in the following manner. The serial interrupt service routine alerts the executive that data has been received. The appropriate data service routine is executed to move the received UART data into a receive ring buffer. In servicing the pass through mode, the data is moved from the receive ring buffer to a transmit ring buffer. The serial interrupt service routine alerts the executive that data can be sent. The appropriate data service routine is executed to move the data from the transmit ring buffer into the UART.

6.3.2.2.4 EXECUTIVE PACKET MODE

When the HAP is operating in the packet mode, serial data is being provided to the PGSC at a regular interval. ESC serial data is stored but ignored. The IAE interrupt service routine queues the transfer of data packets to the PGSC. Queuing is accomplished by counting the IAE interrupts in the IAE interrupt service routine, and setting the "copy packet" HAP flag when the count reaches 1000. Since 2000 IAE interrupts occur in one second, a count of 1000 will be reached in 0.5 seconds. Two data packets will be sent to the PGSC per second. When the executive examines the "copy packet" HAP flag and it is not set, the execution jumps to the top of the executive loop. If the "copy packet" HAP flag is set, the executive begins the process for copying a packet into the GRiD transmit ring buffer. The type of packet that is transferred to the ring buffer depends on the state of the HAP flags. Three types of packets can be sent, and are prioritized in the following order: trigger packet, memory packet, and normal packet. All of the packets are the same length and their format is shown in Figure 6.7. Before attempting to copy a packet into the ring buffer, the executive calculates if a ring buffer overflow will occur. If the calculation determines that an overflow will occur, the executive clears the "copy packet" HAP flag and jumps to the top of the

executive loop. If the calculation determines there is no danger of overflow, the appropriate packet data is copied into the GRiD transmit ring buffer. During the transfer of the packet data to the ring buffer, the executive adds the current command counter value and a packet counter value. The packet counter is incremented every time the executive copies a packet to the ring buffer. The "copy packet" HAP flag is cleared, and execution jumps to the top of the executive loop.

Data packets are sent to the PGSC in the following manner. A data packet is first generated in any of three ways. A trigger packet is generated by the trigger interrupt service routine every time it is entered. A normal packet is generated by the IAE interrupt service routine every 1000 times it is entered. A memory packet is generated whenever a valid MEMORY DUMP command is received by the HAP. The IAE interrupt service routine sets a copy packet HAP flag every 1000 IAE interrupts. The executive examines the copy packet HAP flag, and if clear jumps to the top of the executive loop. If the copy packet HAP flag is set, then the executive examines the HAP flags to determine the packet type needed, and copies the appropriate data packet into the PGSC transmit ring buffer. The serial interrupt service routine alerts the executive that data can be sent. The send GRiD data service routine is executed to move the data from the GRiD transmit ring buffer into the GRiD UART transmit FIFO. The copy packet HAP flag is cleared, and execution jumps to the top of the executive loop.

6.3.3 SERIAL INTERRUPT SERVICE ROUTINE

The serial interrupt service routine is called when the UART asserts the microprocessor interrupt pin *INT3. The UARTs were configured to interrupt the microprocessor when it has received 14 characters or has not received a character

for several character transmission times. Interrupts are also generated when the UART transmit FIFO becomes empty. The serial interrupt service routine reads the UART interrupt identification register and sets the HAP flags to indicate that the UART receiver FIFO has characters or to indicate that the UART transmit FIFO has been emptied. After determining the reason for the interrupt, a return is performed.

6.3.4 IAE INTERRUPT SERVICE ROUTINE

The IAE interrupt service routine is called when the IAE asserts the microprocessor interrupt pin INT1. This interrupt is asserted when the IAE has new gyro rate data available.

The first portion of the IAE interrupt service routine is written using the Intel C-language. A listing of the C-language code is given in Appendix F. All gyro data processing is performed in this interrupt service routine, and uses the algorithms described in Chapter 4.

The remainder of the IAE interrupt service routine is written in assembly language. Both levels of watchdog timers are reset as soon as the IAE data has been processed. A hardware IAE interrupt counter is next compared to a software IAE interrupt counter. Both counters should have the same value. If the values are different, a "lost track of IMU" HAP flag is set and the software counter is re-synchronized with the hardware counter. If the values are identical, the software counter is incremented. A solution count is also maintained. This solution counter is used to determine when a packet of data should be generated and sent to the PGSC. The solution counter is incremented and compared with the maximum count value of 1000. A maximum count of 1000 will occur two times per second since the IAE interrupts occur 2000 times per second. If the solution counter has

reached the maximum count, then the routine gathers the information to make a normal data packet for the PGSC and sets the copy packet HAP flag. Every time this routine makes the normal packet, it changes the state of the WORKING light on the front panel of the HAP. This light provides the user with a quick indication that the HAP is processing information and working.

The data contained in the normal packet includes time. I have written a routine to read the time, compare the time in the three clocks, correct any time differences, and report the corrected time. To read time correctly, all components of time need to be read before any of the components change. A clock rollover detection is incorporated into the UTC clock chips. This rollover detection is determined by reading a periodic flag register. Each bit in this register represents a different component of the time word. If a component of the time word changes, a bit in this register is set. Reading this register clears all rollover detection bits.

Time is read as follows: read the periodic flags register to clear the rollover detectors, read all time components, read the periodic flags register and verify that none of the time components have changed. If any of the time components have changed, the process has to be repeated. I have given the algorithm 16 tries to read the time correctly. If time can not be read correctly in 16 tries, the algorithm sets a "more than n rollover" HAP flag and continues with the data it has already read.

Since the clock chips are redundant, a single error in each time component can be corrected. After reading the time value, the time component values read from each of the three clocks are compared. If one of the values does not match the other two values, the clock in error is corrected to match the other clocks and a "time bad but corrected" HAP flag is set for the clock in error. If all three time values are different, clock 0 is assumed to be correct and the other clocks are set to

the same time as clock 0. A "time uncorrectable" HAP flag is set if all three clocks are different.

6.3.5 TRIGGER INTERRUPT SERVICE ROUTINE

The trigger interrupt service routine is called whenever the ESC shutter trigger pulse is sent from the ESC to the HAP microprocessor interrupt pin INT2. A trigger event requires that the time and attitude information be gathered as quickly as possible. The attitude of the camera and the position of the STS in its orbit and the geolocation is determined using this information. IAE and trigger interrupt service routines run at the same priority. I chose to operate this way so the trigger interrupt routine would have no chance of reading partially processed IAE data.

In addition to gathering the time and attitude data, the trigger interrupt routine is used to set the UTC clocks accurately. When the trigger interrupt is connected to the output of the PCU as shown in Figure 5.16 the time can be accurately set. Once the time setting data has been loaded into HAP and the SET CLOCKS ON NEXT TRIGGER command has been sent, the PCU output is enabled. When the PCU pulse is sent to the HAP, a trigger interrupt is received. This routine checks for the "set clocks on next trigger" HAP flag. If set, the routine will begin the clock setting procedure. The time setting data is first checked to make sure that it is valid. If the time setting data is valid, the clocks are configured, the time is set, the clocks are started, and the time setting data is made invalid. If the "set clocks on next trigger" HAP flags are not both set, or if the time setting data is not valid, the clocks will not be modified.

After determining whether to set the clock or not, the routine begins to

gather a trigger data packet. Time is read, and corrected if necessary, using the same routine that the IAE interrupt service routine uses. Status information is gathered. The current attitude solution is also gathered. After the trigger packet has been made, a return is performed.

6.3.6 WATCHDOG INTERRUPT SERVICE ROUTINE

The watchdog timers provide a safety mechanism. If the code becomes lost or takes too long to complete, the watchdog timer will expire and this watchdog interrupt service routine will be called. This routine disables all interrupts to the microprocessor while it is being executed, and sets the "watchdog expired" HAP flag. The watchdog timers are reinitialized to guard against corrupted watchdog timer data. The execution then branches to the warmstart routine.

The warmstart routine is used by the fault handling routines as well. This routine reinitializes all system data structures using the same routine as in initialization stage 3. After re-initialization of the system data structures, the execution sent to the executive initialization warmstart location. This location is placed just after the IAE data processing variables are initialized. Jumping to this location allows the possibility that the attitude knowledge of the ESC will not be lost. The executive initialization continues as described earlier, until the system is running normally again.

6.3.7 FAULT HANDLING ROUTINES

Recovery from faults depends on the cause of the fault. I was required to write all fault handlers for the HAP application. I formulated three responses to a fault: restart the processor, warmstart the processor, return to the faulting instruc-

tion. In all cases the state of the microprocessor is stored automatically in a fault record. The fault record contains information regarding the type of fault, the faulting instruction location, and the state of the instruction execution. Regardless of the fault type, a fault will cause a flag to be set in the data packet that is sent to the PGSC and in the HAP flags. This set flag will indicate to the PGSC that a fault has occurred, and the fault record will be dumped to the PGSC and stored for later analysis.

A restart processor is performed on 8 of the 14 faults. The restart processor response is only used if no application recovery is possible. The software execution jumps to the same location that is executed after a hardware reset. All system data will be reinitialized and all peripherals will be configured and tested. Fault record data is preserved. If the IMU was aligned, the alignment will be lost.

A warmstart processor is performed on 5 of the 14 faults. The warmstart processor response is a graceful recovery from an unexpected recoverable fault condition. All system data structures are reinitialized and execution returns to the executive initialization warmstart location. Fault record data is preserved. IMU alignment is not lost on a warmstart response.

An operation fault is generated whenever the microprocessor tries to execute an illegal opcode or an instruction with illegal syntax. This fault is one of the warmstart faults. This fault will protect against the microprocessor executing data. A machine fault is generated whenever the microprocessor tries to write a word to PROM. This fault is also one of the warmstart faults. This fault will protect against errant program flow.

A return to the faulting instruction is performed for one of the faults. The microprocessor contains software breakpoint and trace hardware on the chip. If a breakpoint or trace event is detected, the microprocessor generates an event fault.

The debug hardware is unused in the HAP application software so the event fault is ignored and execution returns to the faulting instruction.

7.0 RESULTS AND ANALYSIS

My design and implementation of the HAP have meet all design requirements set in Chapter 4.1, and by the HERCULES system. The results of the HAP implementation are presented below.

7.1 HAP ENCLOSURE

The HAP enclosure is 5.42" high by 11.5" wide and 16.32" deep. This volume was driven mainly by the IAE. The IAE is 4.2" high by 8.0" wide and 8.1" deep. Another factor in the overall size of the HAP were the batteries needed for the UTC clock. Six D-cell batteries were used. The battery pack is 3.94" high by 4.5" wide by 5.06" deep. Both the battery pack and the IAE are mounted to one side wall of the HAP enclosure. The printed circuit boards (PCB) for the HAP computer, the UTC oscillator, and the power supplies are mounted on the other side wall of the HAP enclosure. The front panel of the HAP enclosure has the power switch, status lights, and reset buttons mounted to it. All interface connectors for the HAP are located on the rear panel of the HAP. Top and bottom panels enclose the internal components. Figure 7.1 shows the HAP enclosure, which I worked closely with Mr. Charles Jones to develop.

NOTES: UNLESS OTHERWISE SPECIFIED

1. INTERPRET DRAWING PER MIL-STD-100.
2. INSTALL NUTS, BOLTS, SCREWS, AND WASHERS PER SSD-PS-018.
3. ALL ASSEMBLY PARTS SECURED BY CUSTOMER SHALL BE REMOVABLE FOR EASY ACCESS.

REVISIONS				
ZONE	LTR	DESCRIPTION	DATE	APPROVED
	A	REVISED	02/27/91	
	B	REVISED BATTERY	03/20/91	
	C	REVISED BATTERY POSITION	03/22/91	
	D	REVISED	04/02/91	
	E	REVISED	04/23/91	
	F	ADDED VELCRO	04/29/91	
	G	REVISED	05/14/91	
	H	REVISED	05/22/91	
	I	REVISED BATTERY COVER	05/31/91	
	J	ADDED ENVELOPE DIMENSIONS	07/08/91	
	K	REVISED SHEET NUMBER	11/27/91	

-25 (OSCILLATOR) 10 | 2 E

MS16995-10
AN960C4
16 PLACES

34095910-001

MS24693-C24, 28 PLACES

-23 (POWER SUPPLY) 10 | 6 C

-21 (COMPUTER) 10 | 4 C

-19 9 | 6 C

813, 6 PLACES

FRONT

REAR

11.50

282-002

-20 9 | 3 C

MS16995-10
AN960C4
6 PLACES

211638-4

15.74

16.32

NAS1189E06P5LK, 8 PLACES

7101J51Z0E

-11 5 | 4 C

MPA-103C
C-80
G-12-KB

-15 8 | 6 C

-17 8 | 3 C

NB0E14-12SNT

LJTP02RE11-35S

5.42

POWER

FUSE
5 AMPS

WORKING

RESET

SPARE FUSES

ON

OFF

276-068A

282-002

276-069A

-14 7 | 4 C

-13 6 | 4 C

MS24693-C274
NAS1291C3M
AN960C10L
6 PLACES

NAS1189E06P4LK
28 PLACES

MS24693-C29
7 PLACES

NAS1189E06P5LK, 8 PLACES

-11 5 | 4 C

LJTP02RE19-35S

MS16995-10
NAS1291C04M
AN960C4L
4 PLACES

MS16995-10
NAS1291C04M
AN960C4L
4 REQD EA
2 PLACES

LJTP02RE11-35P

MS16995-10
NAS1291C04M
AN960C4L
4 REQD EA
2 PLACES

NB0E14-12PNT2

-03

PARTS LIST

NAVAL RESEARCH LABORATORY
NAVAL CENTER FOR SPACE TECHNOLOGY
WASHINGTON, D.C. 20375-5000

HERCULES

CHASSIS ASSEMBLY

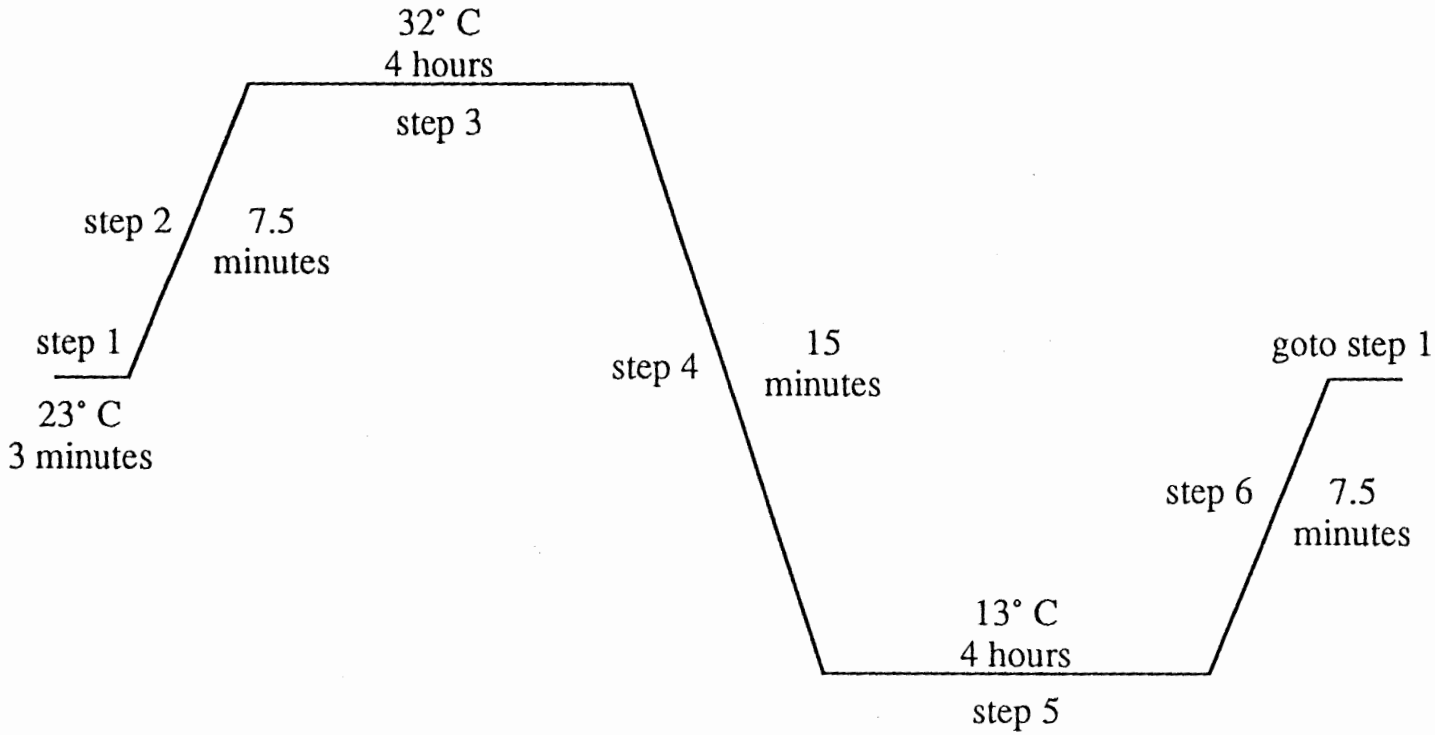
E81995 HERCULES-0001

Figure 7.1 - HAP Enclosure

7.2 HAP THERMAL AND VIBRATION TESTING

To verify that the HAP will survive the STS launch and landing, a vibration test was performed on the HAP. I functionally tested the HAP before the vibration test. The HAP was attached to the vibration test fixture, and the unit was vibrated in each of the three axes at 6.5 Grms for 60 seconds. This vibration level is the amount of vibration that a hard mounted item will see during launch and landing. The HAP is packed in foam in a locker for launch and landing, so the vibration level actually seen during flight will be less than the level to which I have tested the HAP. After the vibration testing was completed, I again functionally tested the HAP. All functions of the HAP were working properly after the vibration test. I removed the top cover of the HAP enclosure and inspected the components for damage. There were no damaged components.

I tested the HAP over the STS operating temperature range. The operating temperature range on the STS is 18 degrees Celsius to 27 degrees Celsius. I have tested the HAP over the temperature range of 13 degrees Celsius to 32 degrees Celsius. I have expanded the testing temperature range to search for temperature sensitivities which would adversely affect the HAP operation on the STS. A thermal chamber controlled the operating environment of the HAP and IMU during the test. ESC shutter trigger pulses were provided to the HAP approximately eight minutes apart. Data was recorded for each of the trigger pulses. Temperature data was recorded on several HAP locations. Temperature was controlled by the chamber, and cycled the temperature over through the profile shown in Figure 7.2. Eight temperature cycles were performed. When the test was completed, I checked the data gathered during the test. This data showed that the HAP operated properly over the required temperature range. All external surfaces of the HAP are cool enough to prevent a burn hazard, even after the HAP has been powered on for



1 temperature cycle takes 8 hours and 33 minutes

Figure 7.2 - Temperature Cycle Profile

several days.

7.3 HAP POWER REQUIREMENTS

I have designed the HAP to operate on 19-40 Volt D.C. power, a wider range than required in Chapter 4.1. The power is converted to the necessary voltages, as described in Chapter 5.7, by the power supply board. The power requirements were analyzed, and are shown in Figure 7.3. Power draw from the HAP was measured to be 27 Watts, which is below the specified maximum of 30 Watts. I had observe low power design rules perpetually to keep the power drawn by HAP below the maximum allowable power requirement.

7.4 HAP WORST-CASE ANALYSIS

A worst-case timing analysis has been performed for all of the HAP digital circuits. The analysis uncovered several initial design flaws that I have corrected in the final design. Analysis presented below reflects the analysis of the final design. Worst-case analysis was performed using the minimum and maximum timing values given in the component data books. These minimum and maximum values are guaranteed over the operating environment of the component. Where no minimums were given in the component data books, 0 nanoseconds was assumed. All setup, hold time, and pulse width requirements were analyzed. The final design does not violate any of the timing requirements of the HAP components.

I generated timing diagrams of the basic microprocessor support hardware. These diagrams are presented in Figures 7.4 through 7.7. All worst-case analyses have the format described below. L-bus states are designated at the top of the timing diagram. HAP signal names are listed on the left side of the diagram. The legend shows there are three patterns used to designate signal conditions.

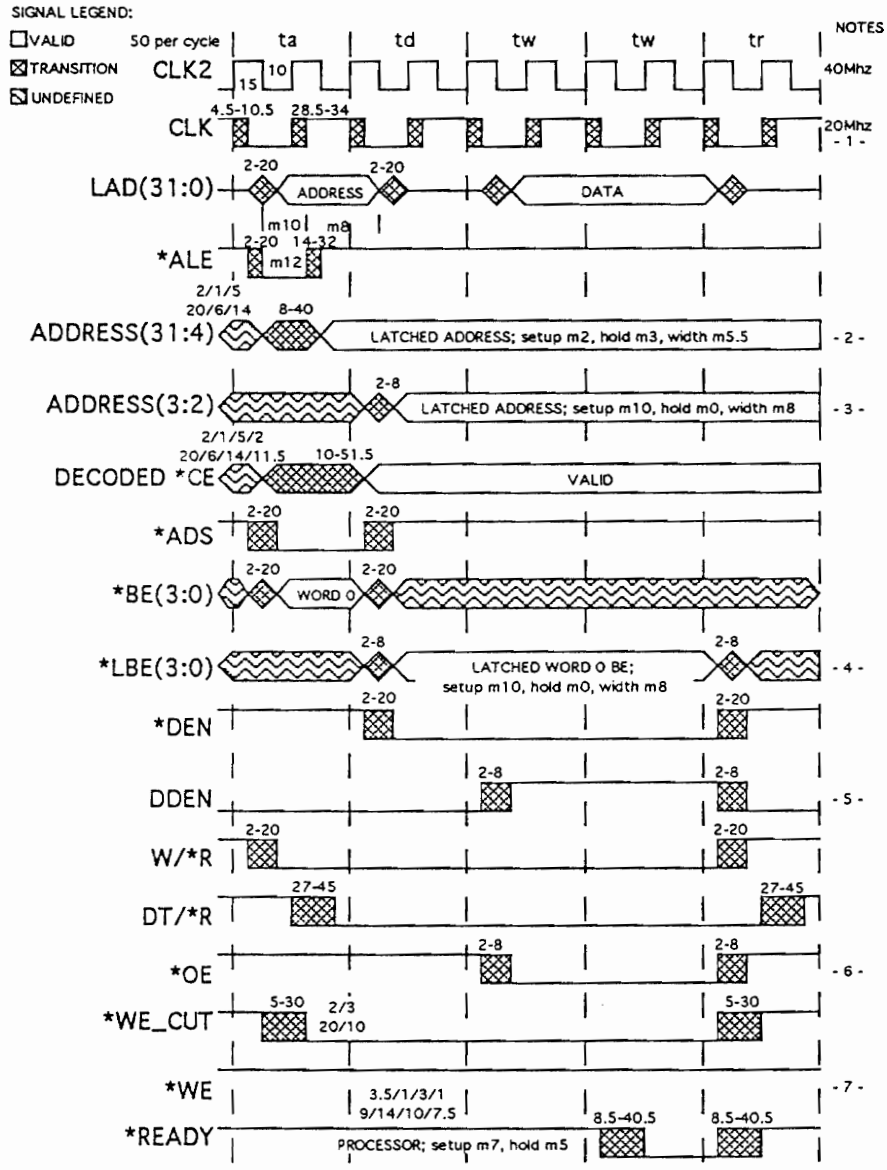
PART #	QTY	CURRENT	TOTAL I
LM139AJ	1	0.006	0.006
LP2951ACJ	1	0.001	0.001
CO-402B-2B-40MHZ	1	0.020	0.020
CO-252-A58-32.768KHZ	1	0.005	0.005
MQ80960MC-20	1	0.500	0.500
MT5C2568C-25	4	0.030	0.120
R29793S	4	0.050	0.200
DP8571AN	3	0.012	0.036
NS16C552V	1	0.030	0.030
TIBPAL20R4-10MJT	3	0.220	0.660
MC1488J	1	0.010	0.010
DS14C89AJ	1	0.001	0.001
SN55ALS192J	6	0.045	0.270
SN55ALS193J	5	0.035	0.175
SN54AS00J	1	0.017	0.017
SN54AS04J	1	0.026	0.026
SM54AS08J	1	0.024	0.024
SN54AS32J	4	0.027	0.108
SN54AS74J	2	0.016	0.032
SN54AS138J	1	0.020	0.020
SN54AS163J	2	0.053	0.106
SN54AS245J	6	0.123	0.738
SN54AS373J	4	0.090	0.360
SN54AS825J	2	0.090	0.180

TOTAL DESIGN CURRENTS:	at +5 volts	3.645 Amps
	at +12 volts	0.1 mA
	at -12 volts	0.1 mA

TOTAL DESIGN REGULATED POWER: 20.6 Watts

TOTAL DESIGN RAW +28V INPUT POWER: 27.5 Watts

Figure 7.3 - HAP Power Requirements



ALL TIMES IN NANO SECONDS
X/X/X/X SHOWS TIMING THROUGH PATH
mX MEANS MINIMUM VALUE

- 1 -
TI S4AS74
clk high m4, design m15
clk low m5.5, design m10
data setup before clk m4.5, design m14.5
data hold after clk m0, design m3.5

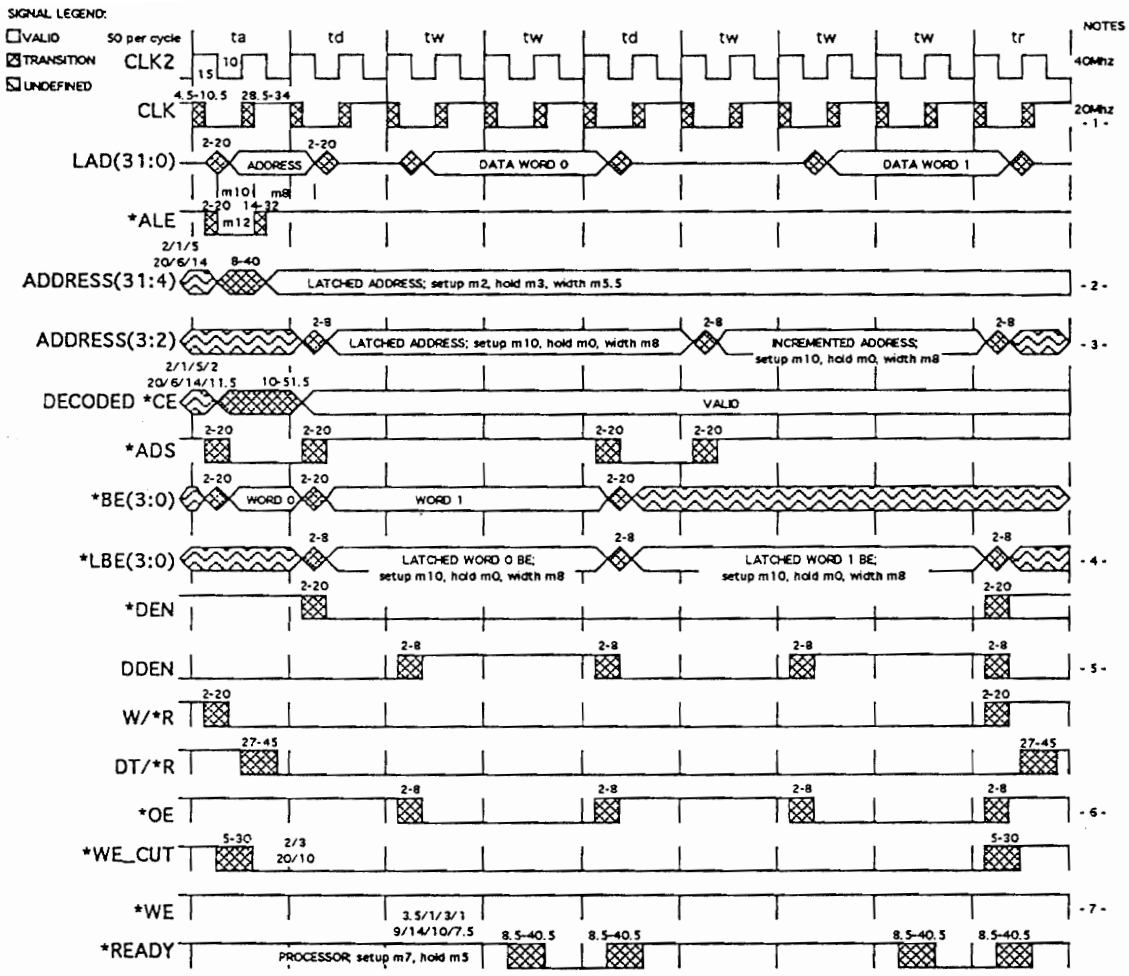
- 2 -
TI S4AS373
c width m5.5, design m10.5
data setup before c m2, design m11
data hold after c m3, design m5.5

- 3 -
TIBPAL20R4-10M
clk high m8, design m15
clk low m8, design m10
data setup before clk m10, design m14.5
data hold after clk m0, design m2

- 4 -
TIBPAL20R4-10M
clk high m8, design m15
clk low m8, design m10
data setup before clk m10, design m14.5
data hold after clk m0, design m2

- 5 - 6 - 7 -
TIBPAL20R4-10M
clk high m8, design m15
clk low m8, design m10
data setup before clk m10, design m14.5
data hold after clk m0, design m2

Figure 7.4 - Single Read Worst Case Timing Analysis



ALL TIMES IN NANO SECONDS
X/X/X/X SHOWS TIMING THROUGH PATH
mX MEANS MINIMUM VALUE

- 1 -
TI 54AS74
clk high m4, design m15
clk low m5.5, design m10
data setup before clk m4.5, design m14.5
data hold after clk m0, design m3.5

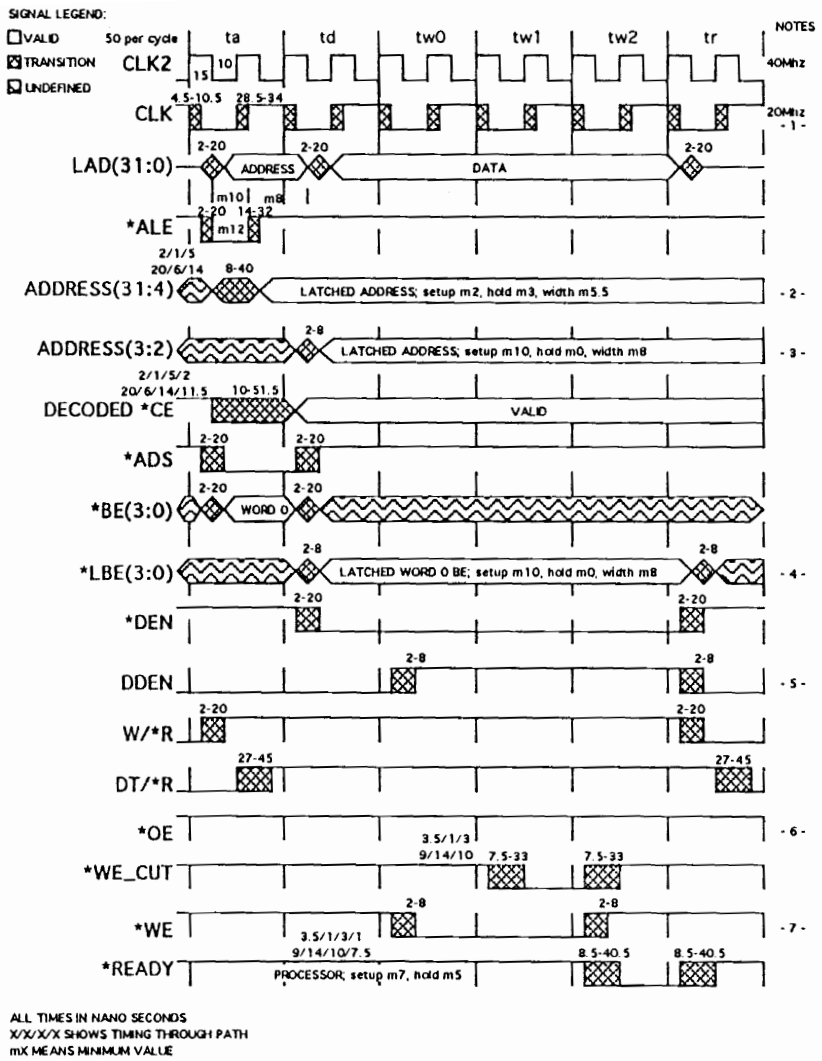
- 2 -
TI 54AS373
c width m5.5, design m10.5
data setup before c m2, design m11
data hold after c m3, design m5.5

- 3 -
TIBPAL20R4-10M
clk high m8, design m15
clk low m8, design m10
data setup before clk m10, design m14.5
data hold after clk m0, design m2

- 4 -
TIBPAL20R4-10M
clk high m8, design m15
clk low m8, design m10
data setup before clk m10, design m14.5
data hold after clk m0, design m2

- 5 - 6 - 7 -
TIBPAL20R4-10M
clk high m8, design m15
clk low m8, design m10
data setup before clk m10, design m14.5
data hold after clk m0, design m2

Figure 7.5 - Burst Read Worst Case Timing Analysis



- 1 -
TI 54AS74
clk high m4, design m15
clk low m5.5, design m10
data setup before clk m4.5, design m14.5
data hold after clk m0, design m3.5

- 2 -
TI 54AS373
c width m5.5, design m10.5
data setup before c m2, design m11
data hold after c m3, design m5.5

- 3 -
TIBPAL20R4-10M
clk high m8, design m15
clk low m8, design m10
data setup before clk m10, design m14.5
data hold after clk m0, design m2

- 4 -
TIBPAL20R4-10M
clk high m8, design m15
clk low m8, design m10
data setup before clk m10, design m14.5
data hold after clk m0, design m2

- 5 - 6 - 7 -
TIBPAL20R4-10M
clk high m8, design m15
clk low m8, design m10
data setup before clk m10, design m14.5
data hold after clk m0, design m2

Figure 7.6 - Single Write Worst Case Timing Analysis

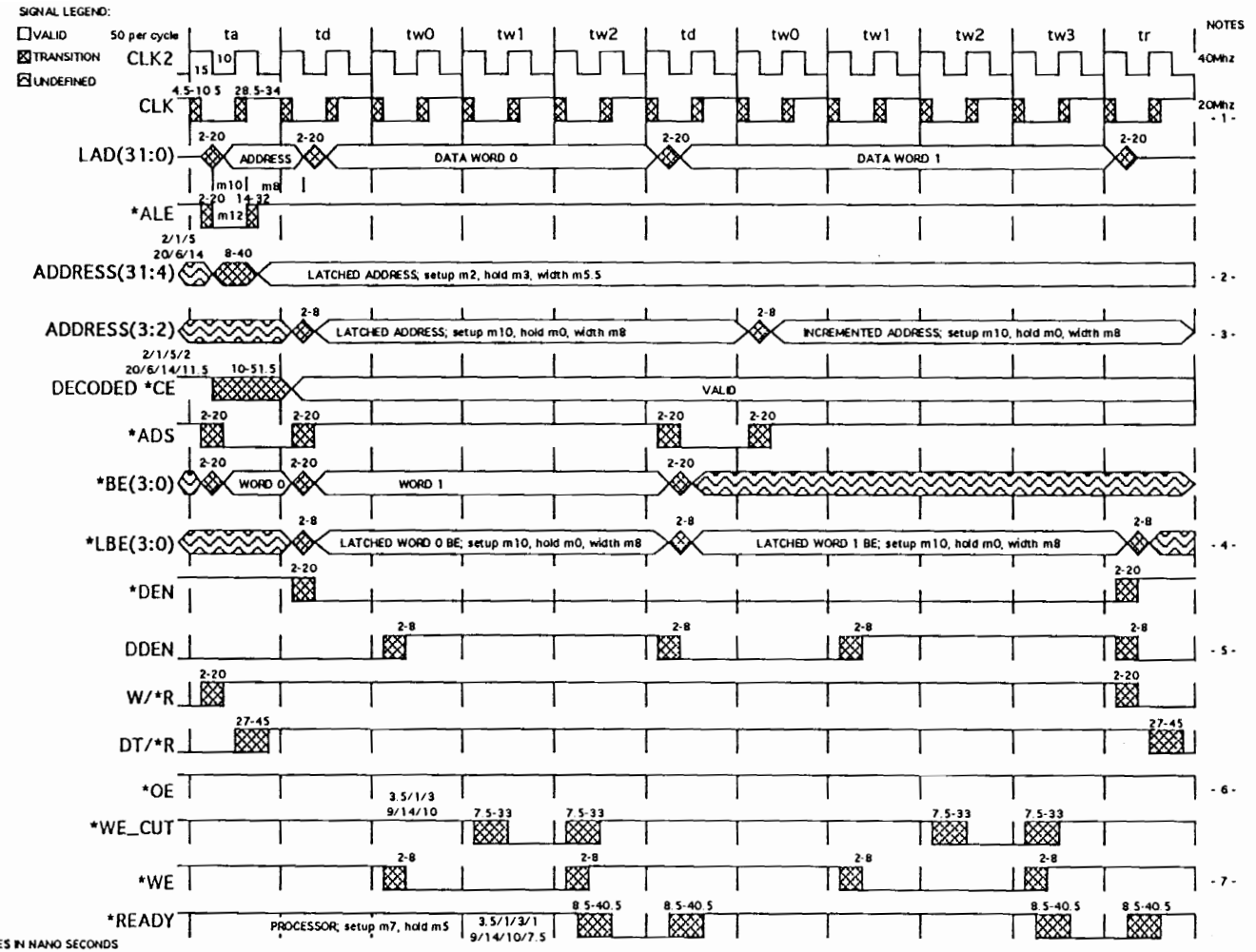


Figure 7.7 - Burst Write Worst Case Timing Analysis

Plain white background designates that the signal or group of signals is valid, and not in transition. A checkerboard pattern is used to designate that the signal or group of signals is in transition. The exact time of the signal transition can be any time in the checkerboard area. A wave pattern is used to designate unspecified or undefined signal conditions. High impedance signal conditions are shown with a signal level in the middle. Timing values, in nanoseconds, are given for each signal transition. The path used to obtain the timing values is given and the minimum and maximum transition times are given. Notes about the timing calculations are given on the right side and on the bottom of the diagram.

Several circuits were added to my initial design to meet the worst-case analysis requirement. The worst-case analysis of the initial design showed that the address would not be valid as late as 8 nanoseconds into the L-bus data cycle. Peripheral chip output enables and write enables were generated using *DEN. These enables could be asserted as early as 8 nanoseconds. The timing requirements most of the peripherals, including the clock chips and UART chip and IAE interface, required that the address be valid several nanoseconds before the enables were asserted. Using *DEN this setup time could not be met. To prevent violation of the data setup requirements of the peripherals, a delayed *DEN signal (DDEN) was generated. I implemented the DDEN signal as described in Chapter 5.1.3.

Another problem was uncovered by the worst-case analysis of the initial design. This problem was found in the byte enable latch. The initial design used a 54AS825 8 bit latch to latch the byte enables. Worst-case analysis of the initial design showed that the data setup times of the latch would not be met in all conditions. I fixed this timing problem in the final design using a PAL to latch the byte enables. Worst-case analysis of the final design using the PAL showed that all timing requirements were met.

A problem was also found with the burst transactions of the original design. The *DEN signal is active for the entire transaction. When the address changes, *DEN is active. This again would violate the address setup time of some of the peripherals. The DDEN signal was designed to become unasserted before the address change, and become asserted after the address setup time had been met. A worst-case analysis of the final design showed that these problems had been fixed.

Figure 7.4 shows the single read worst-case analysis of the signals generated by my microprocessor support logic. This logic includes the address latch, address decoder, byte enable latches, output enable (*OE), write enable (*WE), and the ready signal. DDEN is shown in this Figure 7.4, note that the address is valid before the DDEN and *OE are asserted. The notes on the bottom of Figure 7.4 show the timing margins for each of the components in the microprocessor support logic.

Figure 7.5 shows the burst read worst-case analysis of the signals generated by my microprocessor support logic. DDEN is also shown in this Figure 7.5, note that the DDEN and *OE signals are enabled when the address is valid and disabled when the address is invalid. Note the timing for the address incrementing function of the burst logic PAL. After the first read has been completed, the support logic asserts the ready signal. After the first ready, the address is incremented and the second read is completed. The byte enable latch PAL is also shown latching the byte enables at the correct times.

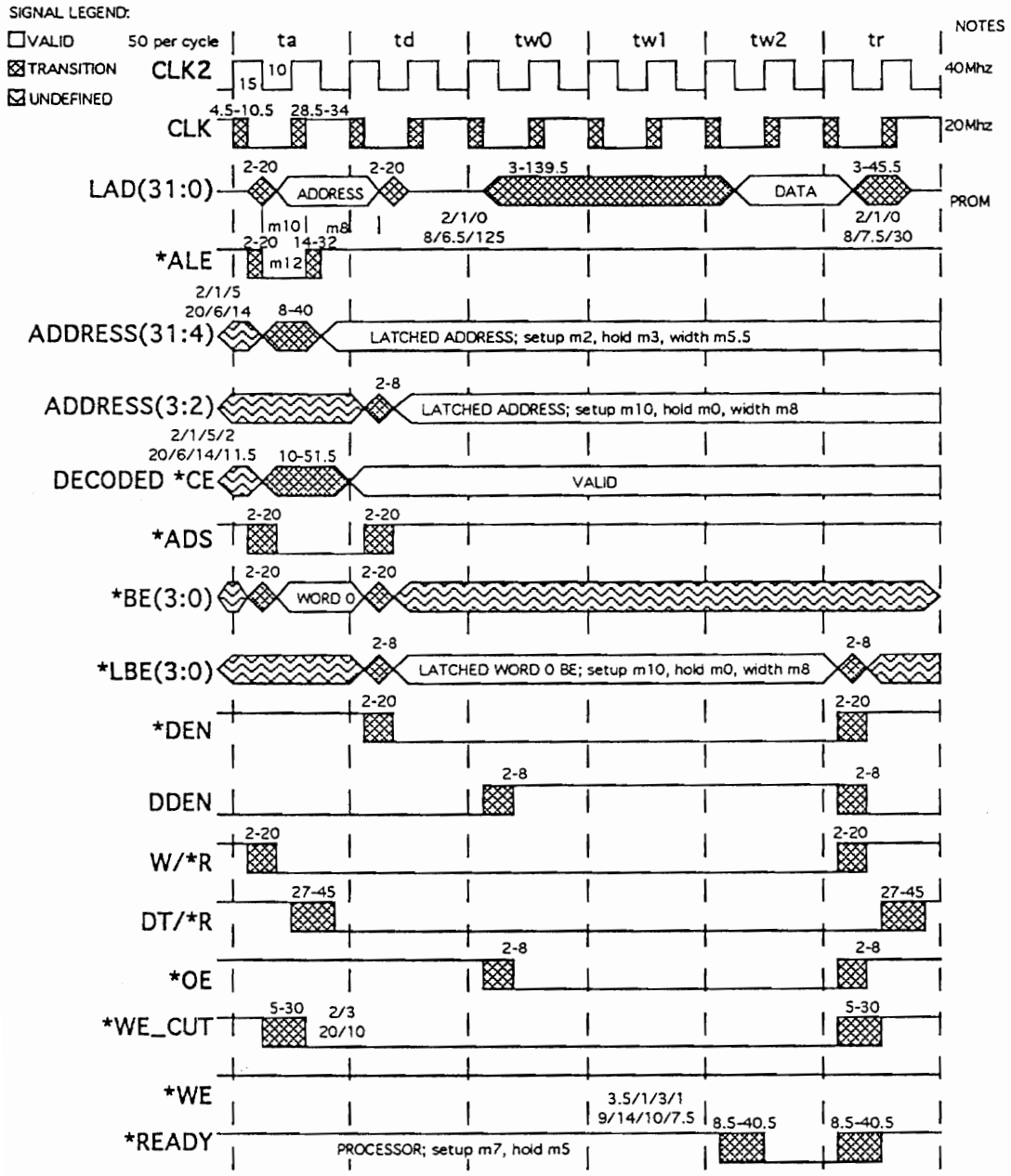
Worst-case timing for a single write transaction is shown in Figure 7.6. A problem was found with the write transactions in the initial design. All of the peripherals require a data hold time of at least 0 nanoseconds on a write transaction before the write enable (*WE) is removed. Using the *DEN or DDEN signal

to unassert the write enable can not guarantee that the data will still be valid when the write enable is unasserted. Data could be changing when the signals are still telling the peripheral to perform a write. This problem was corrected in the final design by shortening the write enable signal by one clock cycle at the end. Note the write enable, shown in Figure 7.6, is unasserted at least one clock cycle before the data is removed in the final design.

Figure 7.7 shows the burst write worst-case timing analysis. This timing diagram also shows the use of the DDEN and shortened write enable signals to guarantee that timing conditions are met during a burst write transaction.

A worst-case timing analysis has been performed on the PROM section for a single read, and a burst read transaction. Two wait states were added during DDEN to guarantee that all timing requirements were met for a read from PROM. The timing analysis, shown in Figures 7.8 and 7.9, showed there is additional timing margin over the minimum timing values. A write to PROM does not affect the contents of the PROM but causes a fault with a warmstart recovery.

A worst-case timing analysis has been performed on the RAM section for a single read, a burst read, a single write, and a burst write transaction. The HAP interrupt enable register timing analysis is included for a single write and a burst write transaction. No wait states were required during DDEN for a single or burst read transaction to meet minimum timing requirements of the RAM section. Single and burst read transaction worst-case timing analyses for the RAM section are shown in Figure 7.10 and 7.11. The HAP interrupt enable register is a write only register. One wait state was required during DDEN for a single or burst write transactions to meet minimum timing requirements of the RAM or HAP interrupt enable section. One more wait state was required for the write then the read because the write enable pulse is one cycle shorter than the read enable pulse.



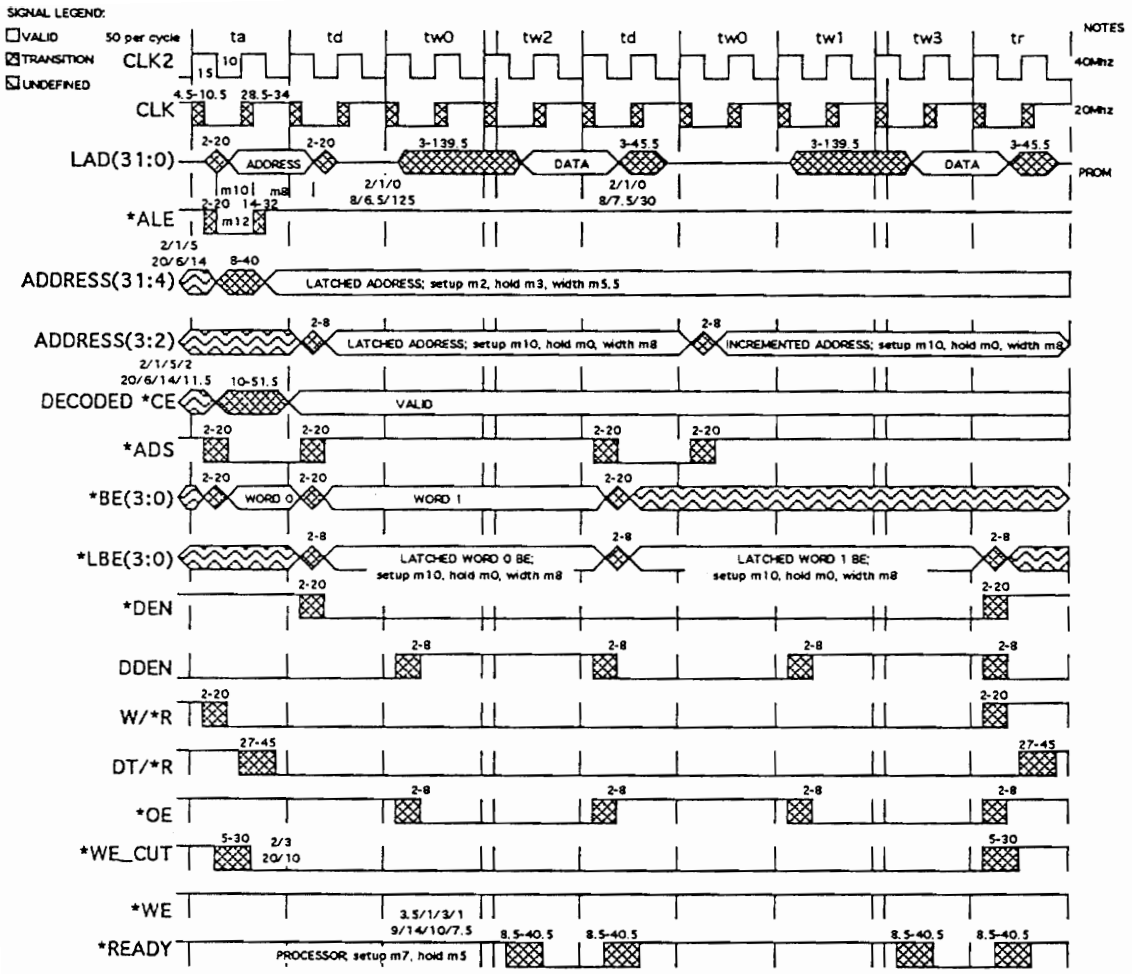
ALL TIMES IN NANO SECONDS
X/X/X/X SHOWS TIMING THROUGH PATH
mX MEANS MINIMUM VALUE

Prom:
Raytheon R29793
enable access m125, design allows m135.5
enable recovery m30

2 wait states during dden

Byte enables not used

Figure 7.8 - PROM Single Read Worst Case Timing Analysis



ALL TIMES IN NANO SECONDS
X/X/X/X SHOWS TIMING THROUGH PATH
mX MEANS MINIMUM VALUE

Prom:
Raytheon R29793
enable access m125, design allows m135.5
enable recovery m30

2 wait states during dden

Byte enables not used

Figure 7.9 - PROM Burst Read Worst Case Timing Analysis

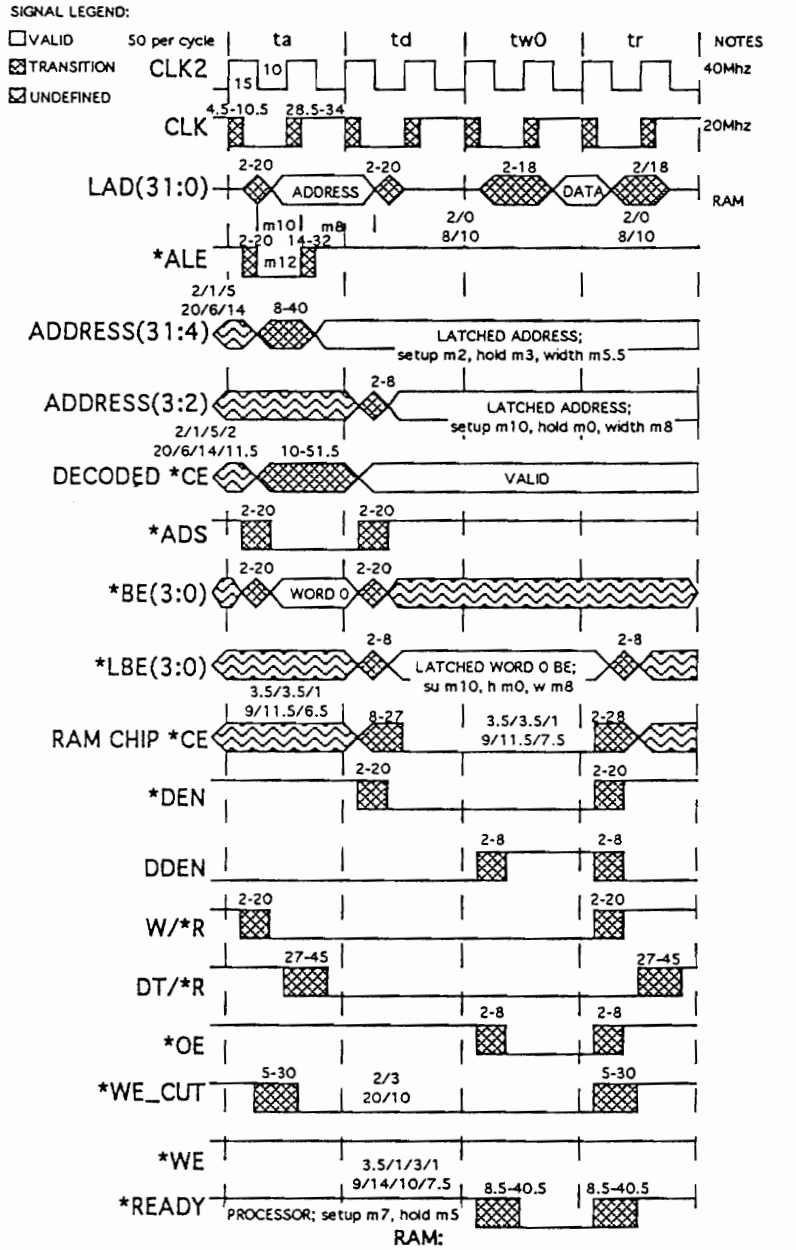
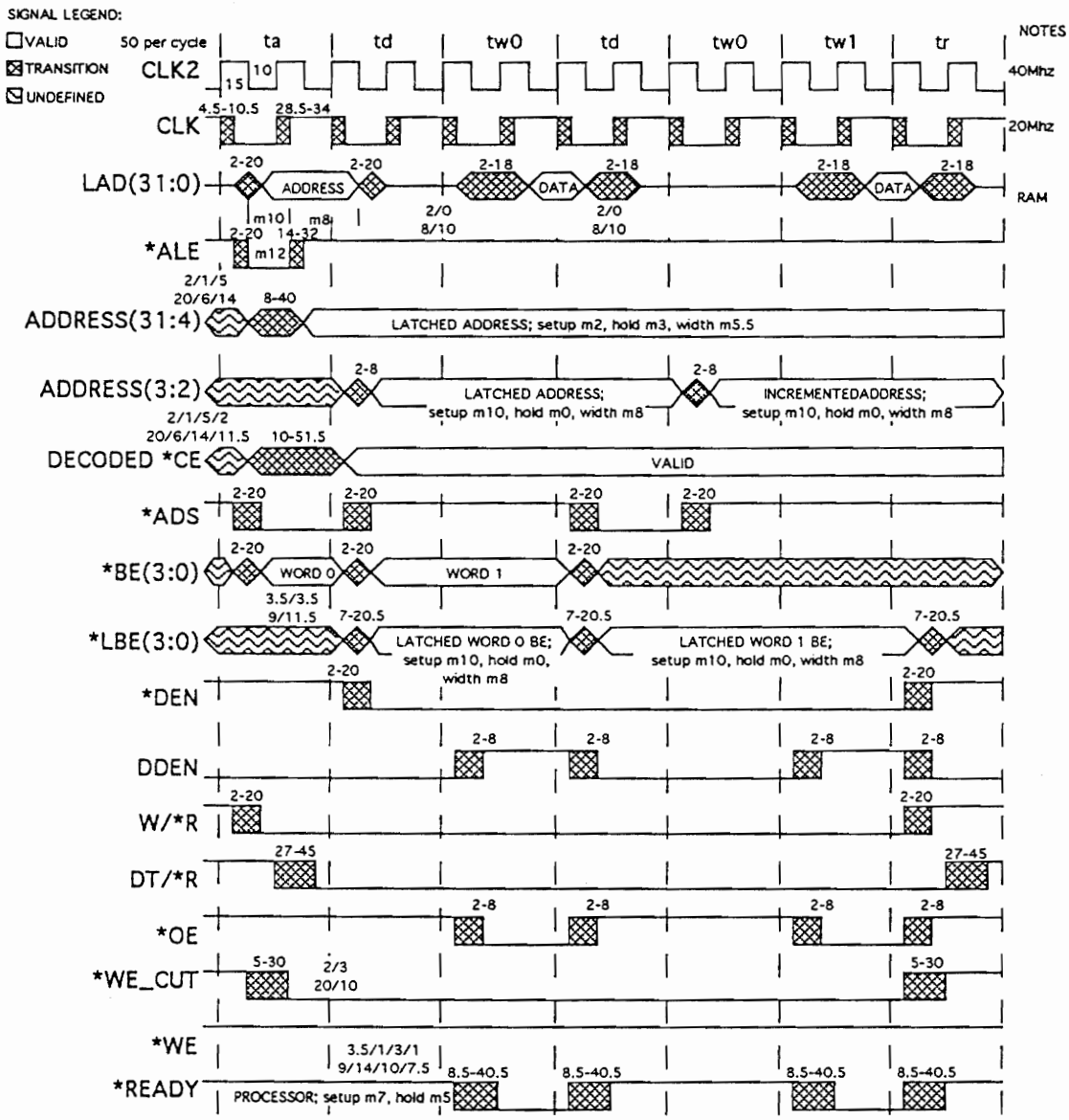


Figure 7.10 - RAM Single Read Worst Case Timing Analysis



ALL TIMES IN NANO SECONDS
X/X/X/X SHOWS TIMING THROUGH PATH
mX MEANS MINIMUM VALUE

RAM:
Micron MT5C2568-25
*ce width m25, design m158.5
*ce access m25, not used
*oe access m10, design allows m42
*ce recovery m15, not used
*oe recovery m10
address setup before *oe m0, design m44
address hold after *oe m0, design m44

0 wait states during dden

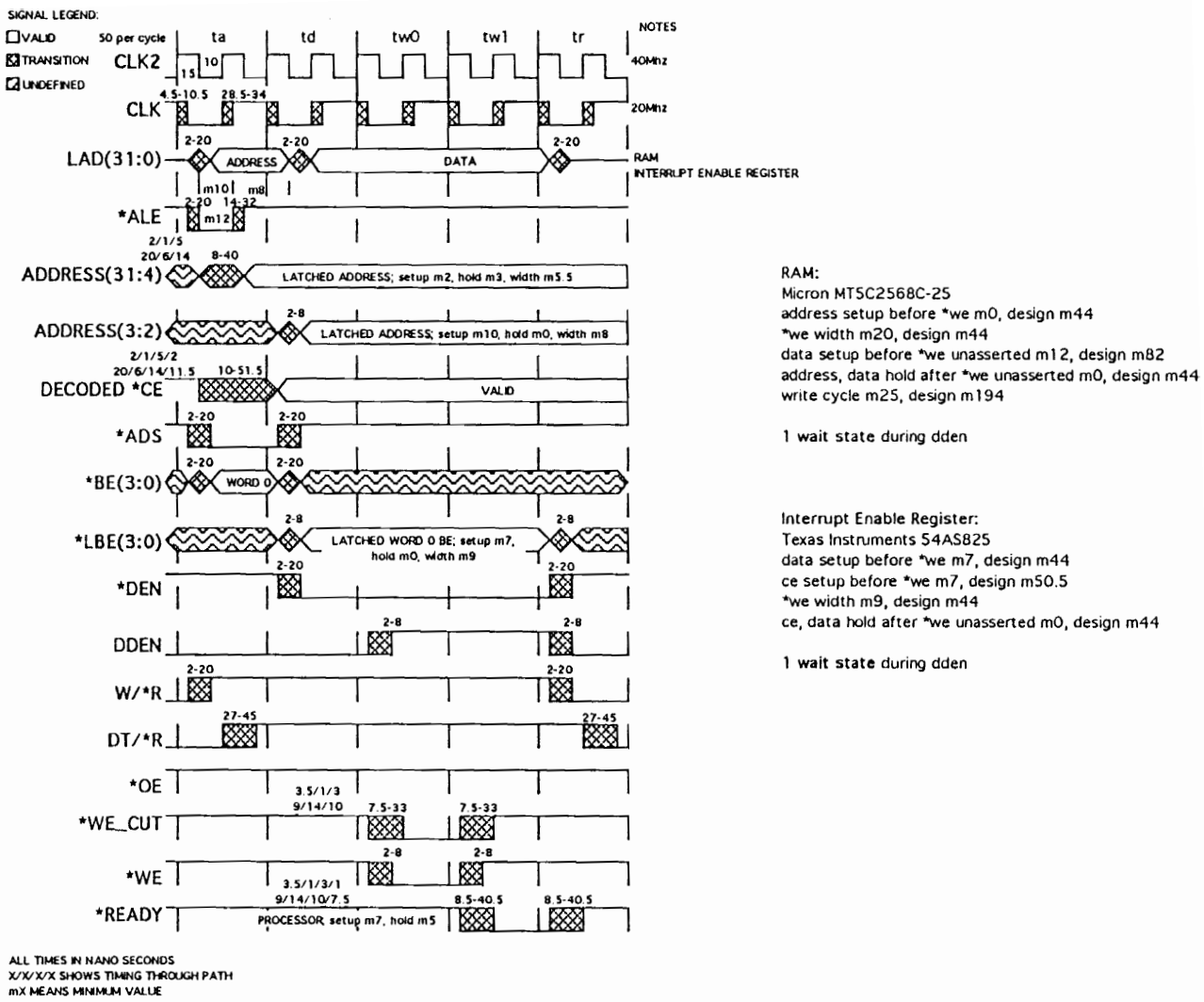
Figure 7.11 - RAM Burst Read Worst Case Timing Analysis

Figures 7.12 and 7.13 show the worst-case timing analyses for the single and burst write transactions to the RAM and HAP interrupt enable sections.

I have performed a worst-case analysis of the IAE section. A single read transaction is shown in Figure 7.14. Thirteen wait states are required during DDEN for the IAE read transaction. An enable signal (*IAE_DS) is required by the IAE which has special timing requirements. A PAL described in Chapter 5.1.3 was used to generate this signal. Special timing requirements were also put on the IAE address signals. The IAE interface design uses latches to guarantee this timing requirement. The *IAE_DS signal and the address signals timing relations are shown in Figure 7.14. Fifteen wait states are required during DDEN for the IAE write transaction shown in Figure 7.15. The IAE interface section meets the worst-case timing requirements.

A worst-case timing analysis has been performed on the Clock section for a single read, a burst read, a single write, and a burst write transaction. The single and burst read transactions, shown in Figures 7.16 and 7.17, require 2 wait states during DDEN to guarantee operation over worst-case conditions. The single and burst write transactions, shown in Figures 7.18 and 7.19, require 3 wait states during DDEN to guarantee operation over worst-case conditions. The Clock section meets the worst-case timing requirements.

A worst-case timing analysis has been performed on the Serial section for a single read, a burst read, a single write, and a burst write transaction. The single and burst read transactions, shown in Figures 7.20 and 7.21, require 1 wait state during DDEN to guarantee operation over worst-case conditions. The single and burst write transactions, shown in Figures 7.22 and 7.23, require 3 wait states during DDEN to guarantee operation over worst-case conditions. The Serial section meets the worst-case timing requirements.



RAM:
Micron MT5C2568C-25
address setup before *we m0, design m44
*we width m20, design m44
data setup before *we unasserted m12, design m82
address, data hold after *we unasserted m0, design m44
write cycle m25, design m194

1 wait state during dden

Interrupt Enable Register:
Texas Instruments 54AS825
data setup before *we m7, design m44
ce setup before *we m7, design m50.5
*we width m9, design m44
ce, data hold after *we unasserted m0, design m44

1 wait state during dden

Figure 7.12 - RAM and HAP Interrupt Enable Register Single Write Worst Case Timing Analysis

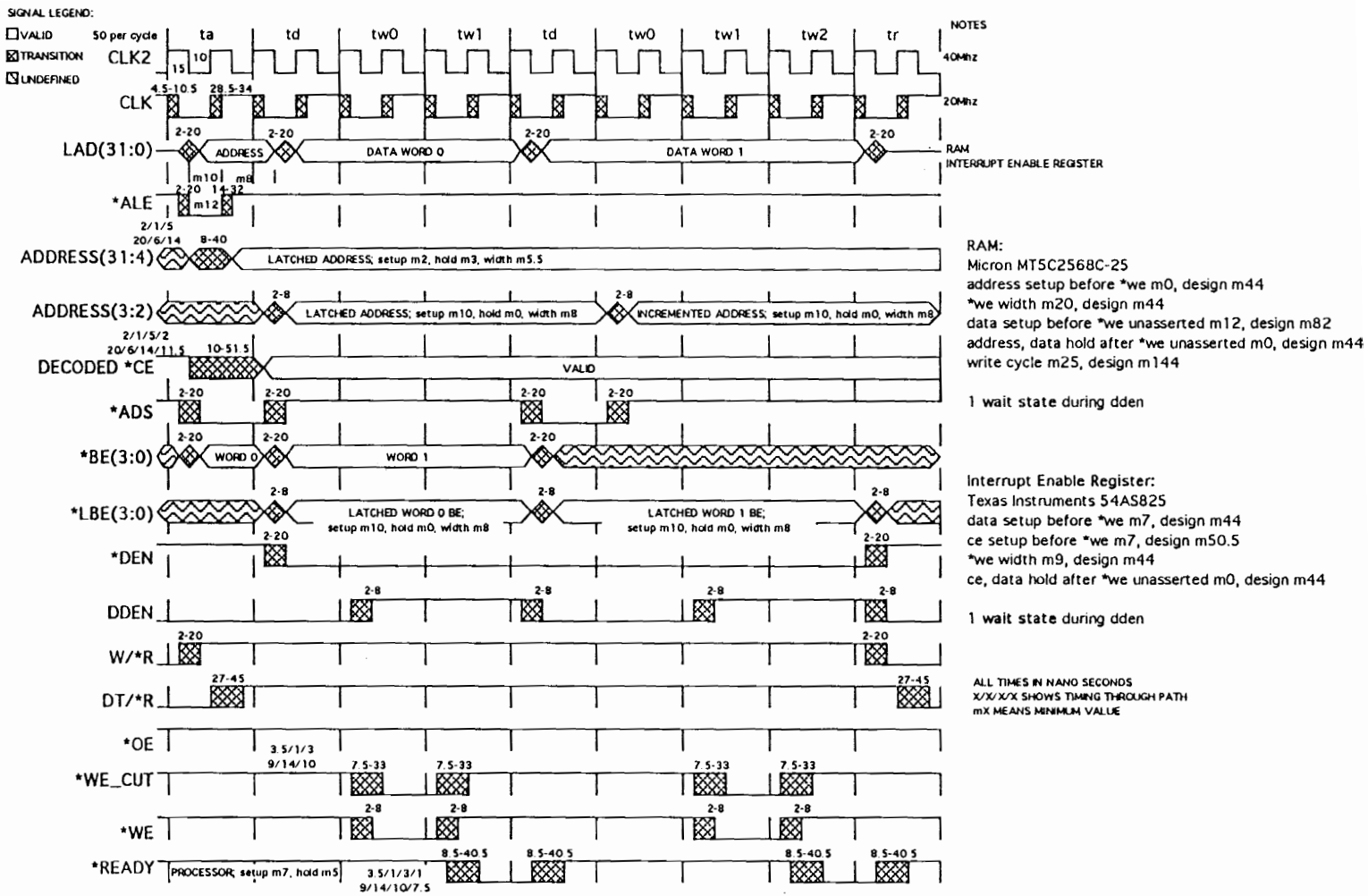
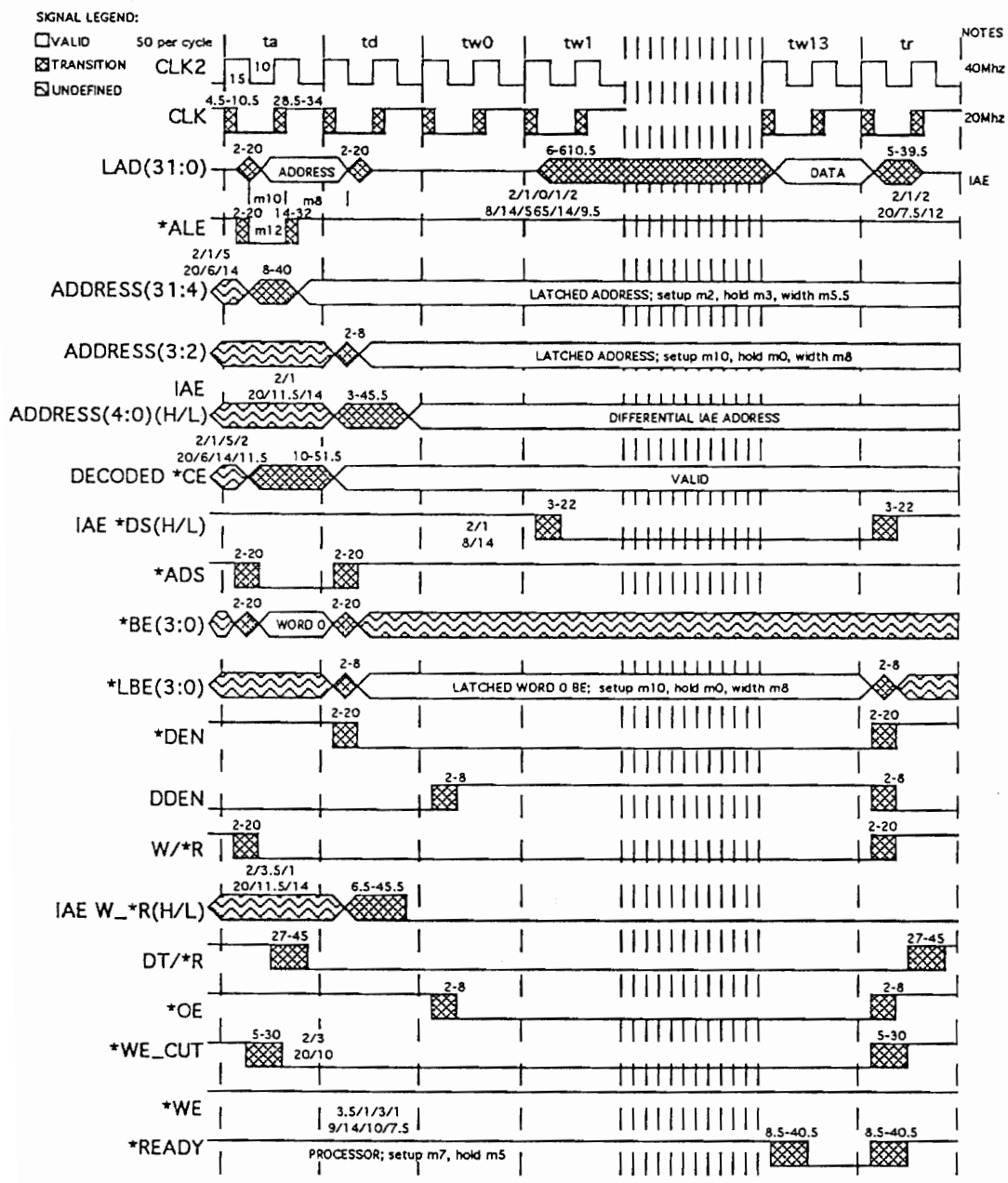


Figure 7.13 - RAM and HAP Interrupt Enable Register Burst Write Worst Case Timing Analysis



ALL TIMES IN NANO SECONDS
X/X/X/X SHOWS TIMING THROUGH PATH
mX MEANS MINIMUM VALUE

IAE:
Honeywell inertial attitude electronics
address & w_*r setup before *ds m40, design m57.5
*ds access time m565, design allows m604.5
address & w_*r hold after *ds m40, design m181

13 wait states during dden

Byte enables not used

Figure 7.14 - IAE Single Read Worst Case Timing Analysis

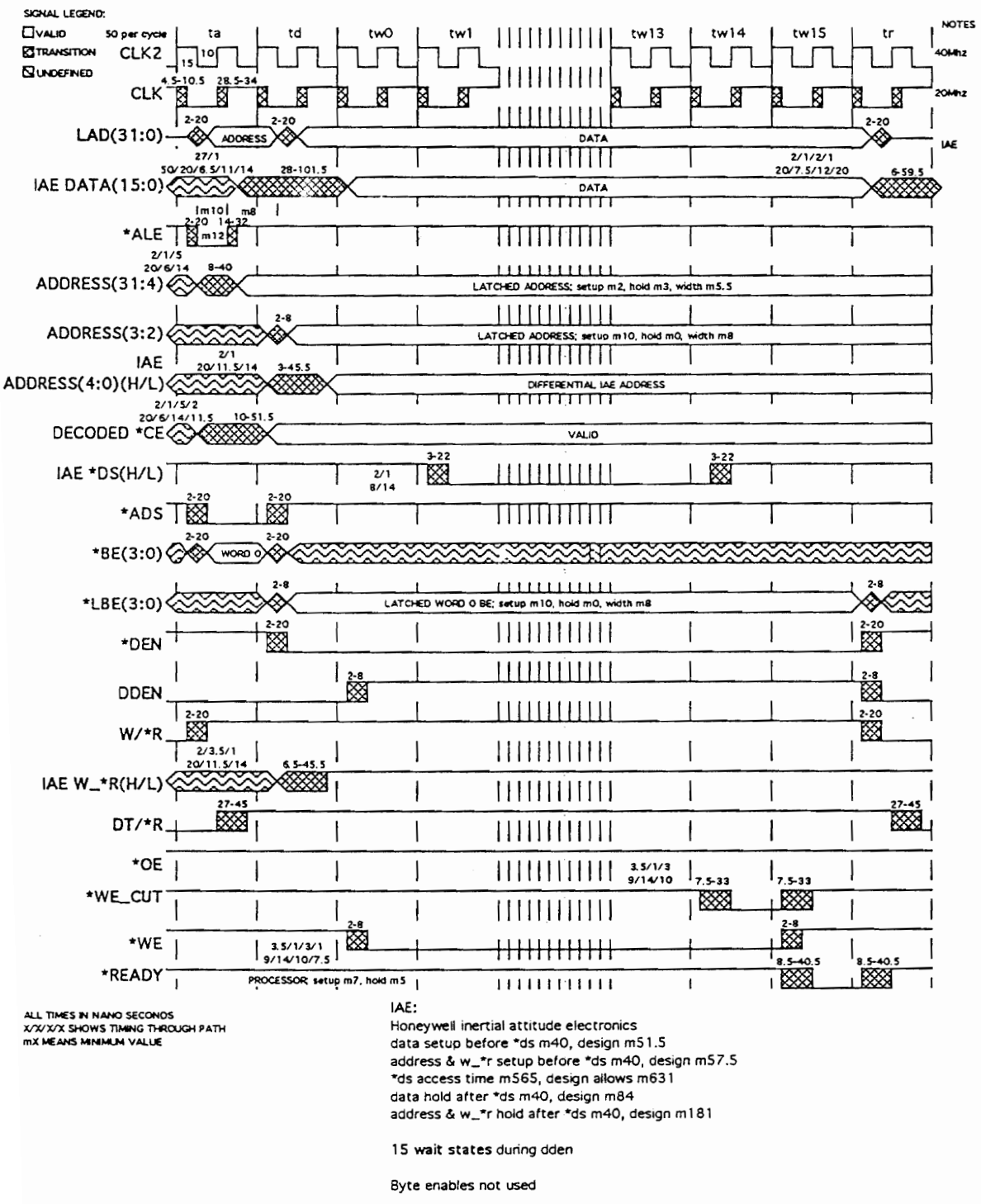
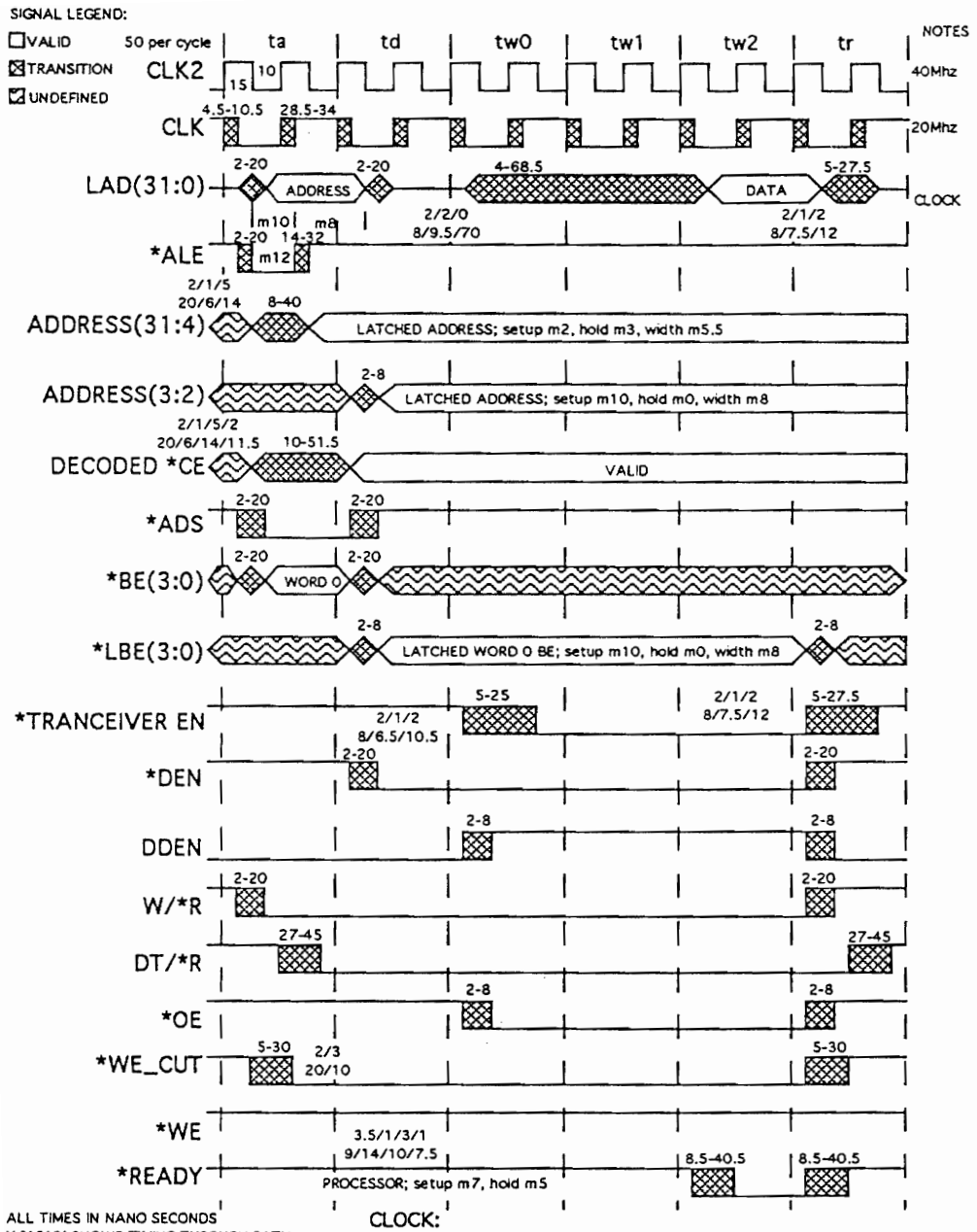


Figure 7.15 - IAE Single Write Worst Case Timing Analysis



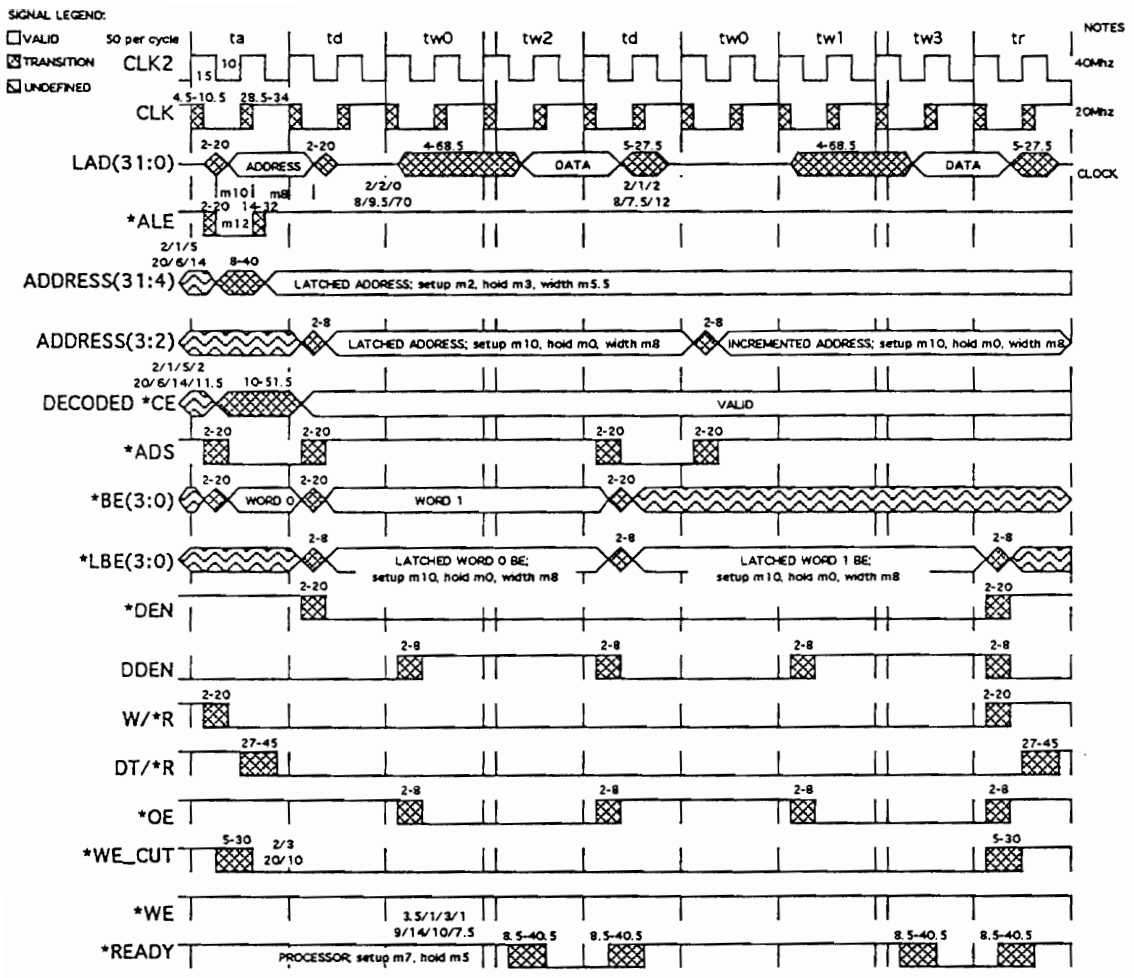
ALL TIMES IN NANO SECONDS
X/X/X/X SHOWS TIMING THROUGH PATH
mX MEANS MINIMUM VALUE

CLOCK:
National DP8571A
address setup before *oe m20, design m44
*oe access m70, design m151.5
*ce access m80, not used
*oe width m80, design m144
address hold after *oe m3, design m42
*oe recovery m60
*ce hold after *oe m0, design m52
read cycle m50, design m144

2 wait states during dden

Byte enables not used

Figure 7.16 - Clock Single Read Worst Case Timing Analysis



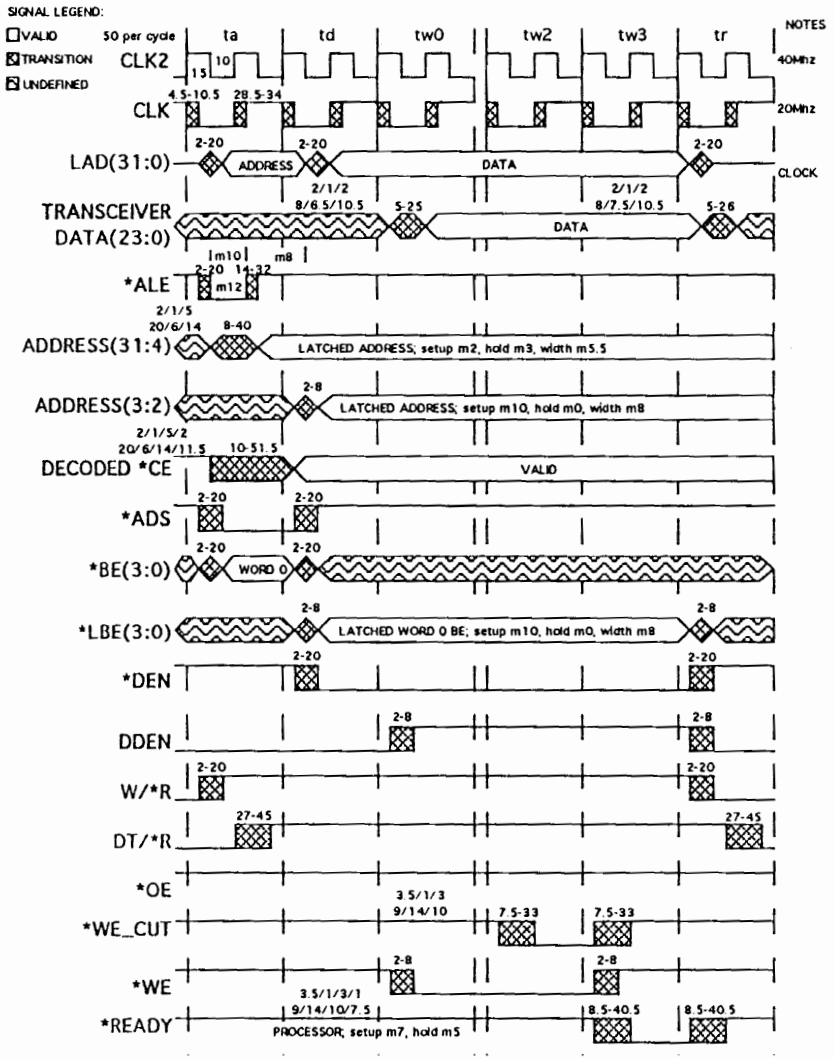
ALL TIMES IN NANO SECONDS
X/X/X/X SHOWS TIMING THROUGH PATH
mX MEANS MINIMUM VALUE

CLOCK:
National DP8571A
address setup before *oe m20, design m44
*oe access m70, design m151.5
*ce access m80, not used
*oe width m80, design m144
address hold after *oe m3, design m44
*oe recovery m60
*ce hold after *oe m0, design m52
read cycle m50, design m94

2 wait states during dden

Byte enables not used

Figure 7.17 - Clock Burst Read Worst Case Timing Analysis



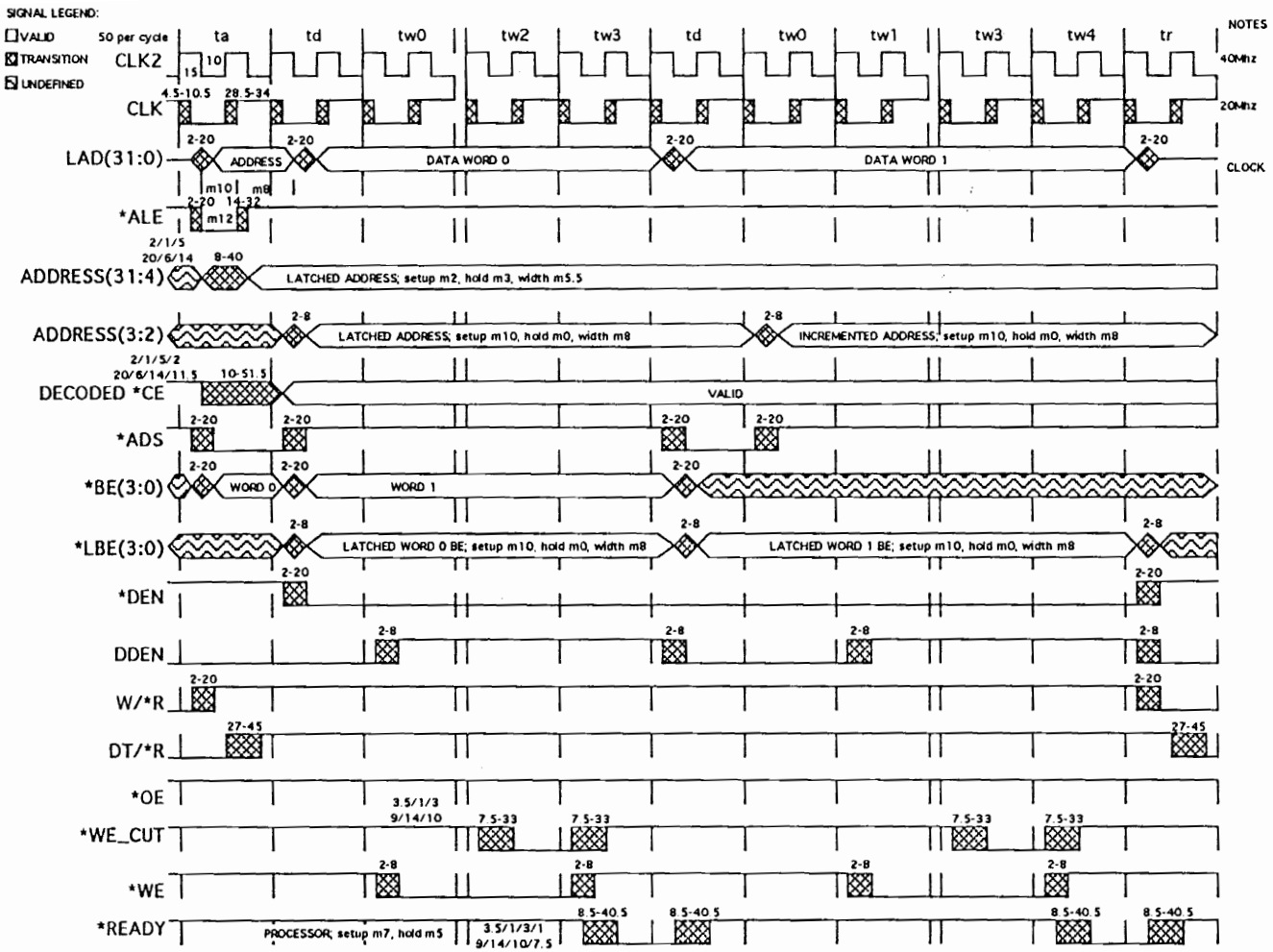
Clock:
National DP8571A
address setup before *we m20, design m44
*we width m80, design m144
*ce width m90, design m308.5
data setup before *we unasserted m50, design m127
address, data hold after *we unasserted m3, design m47
*ce hold after *we m0, design m102
write cycle m50, design m194

3 wait states during dden

Byte enables not used

ALL TIMES IN NANO SECONDS
X/X/X/X SHOWS TIMING THROUGH PATH
mX MEANS MINIMUM VALUE

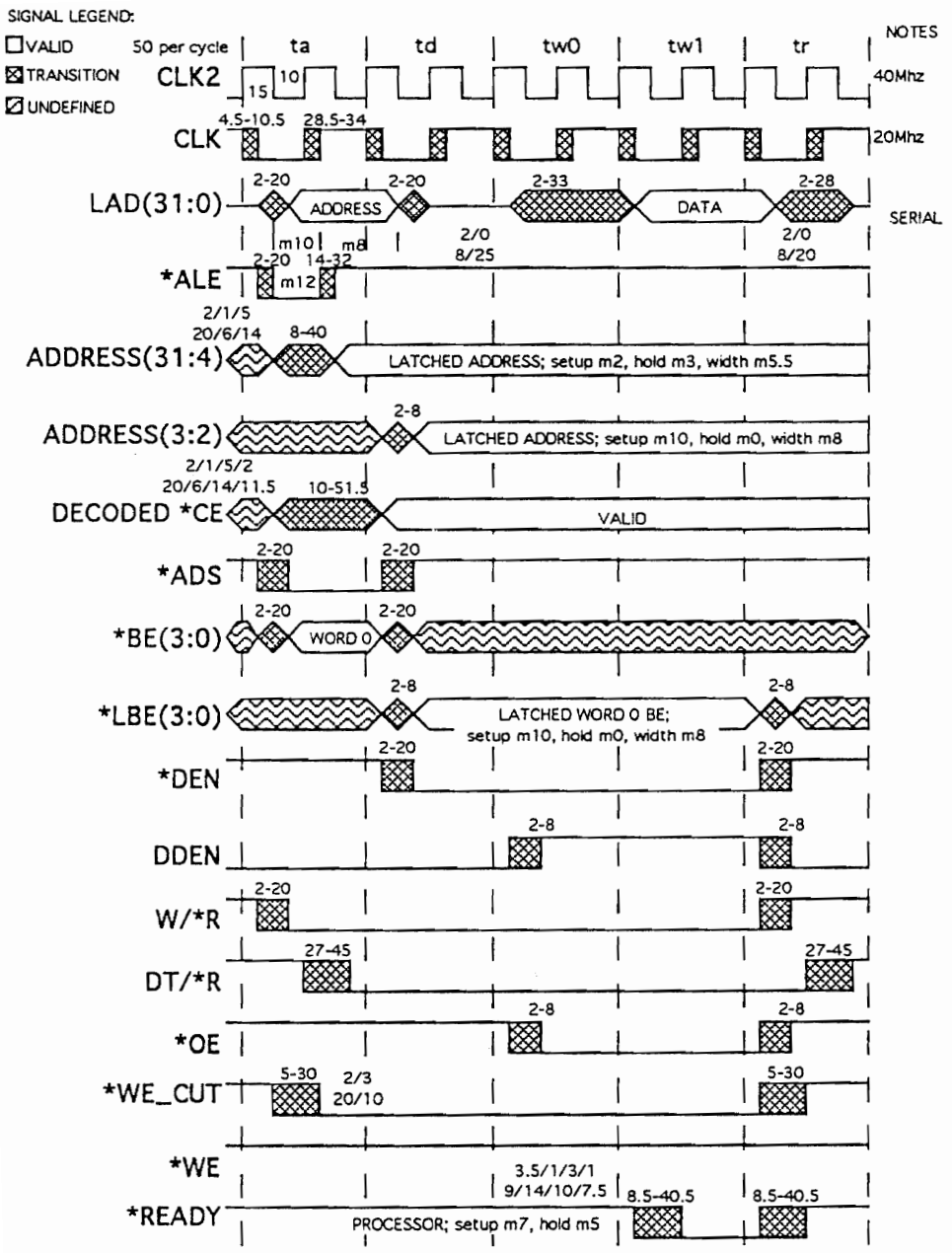
Figure 7.18 - Clock Single Write Worst Case Timing analysis



ALL TIMES IN NANO SECONDS
X/X/X/X SHOWS TIMING THROUGH PATH
mX MEANS MINIMUM VALUE

Clock:
National DP8571A
address setup before *we m20, design m44
*we width m80, design m144
*ce width m90, design m608.5
data setup before *we unasserted m50, design m127
address, data hold after *we unasserted m3, design m47
*ce hold after *we m0, design m102
write cycle m50, design m144
3 wait states during dden
Byte enables not used

Figure 7.19 - Clock Burst Write Worst Case Timing Analysis



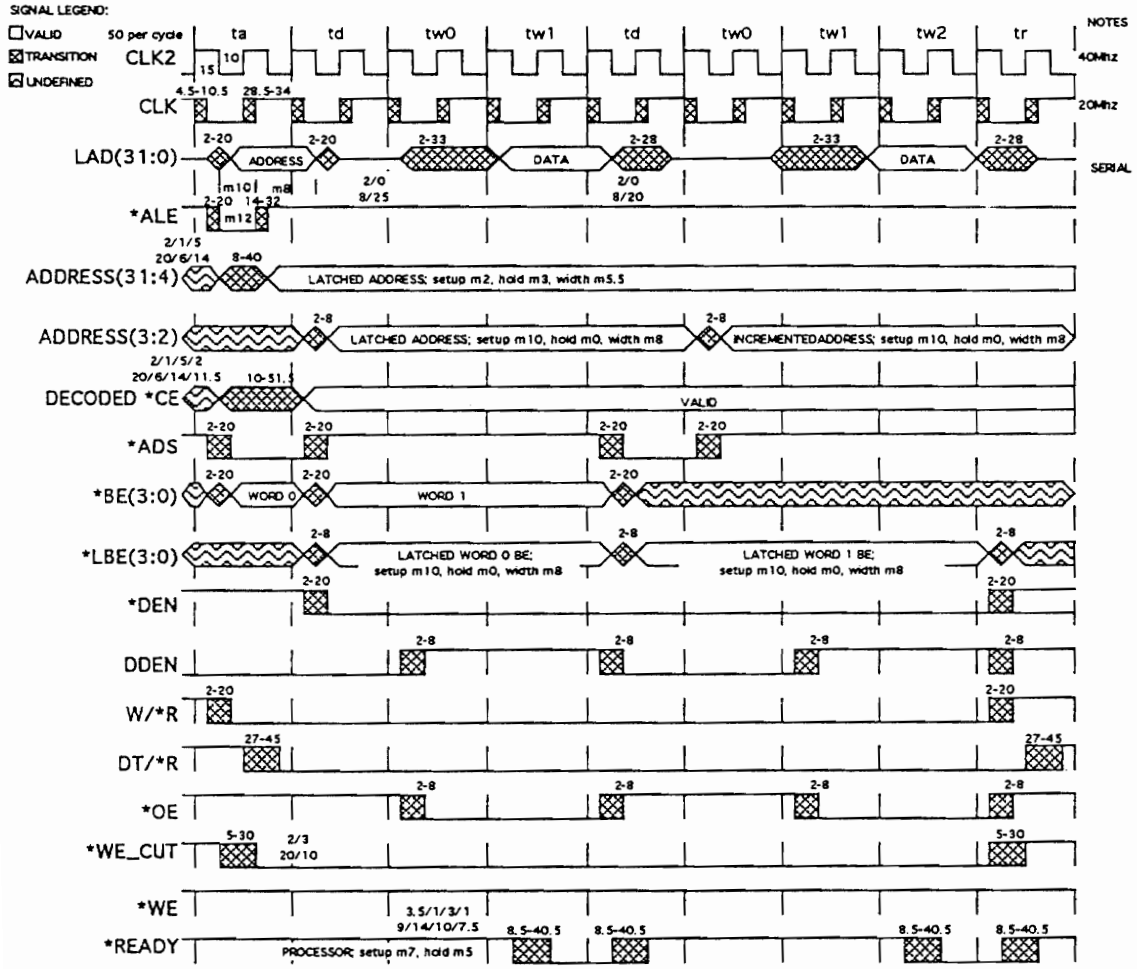
ALL TIMES IN NANO SECONDS
X/X/X/X SHOWS TIMING THROUGH PATH
mX MEANS MINIMUM VALUE

SERIAL:
National NS16C552V
address and *ce setup before *oe m15, design m44
read access m25, design m92
read width m40, design m94
read recovery m20
address and *ce hold after *oe m0, design m44

1 wait state during dden

Byte enables not used

Figure 7.20 - Serial Single Read Worst Case Timing Analysis



ALL TIMES IN NANO SECONDS
X/X/X/X SHOWS TIMING THROUGH PATH
mX MEANS MINIMUM VALUE

SERIAL:
National NS16C552V
address and *ce setup before *oe m15, design m44
read access m25, design m92
read width m40, design m94
read recovery m20
address and *ce hold after *oe m0, design m44

1 wait state during dden

Byte enables not used

Figure 7.21 - Serial Burst Read Worst Case Timing Analysis

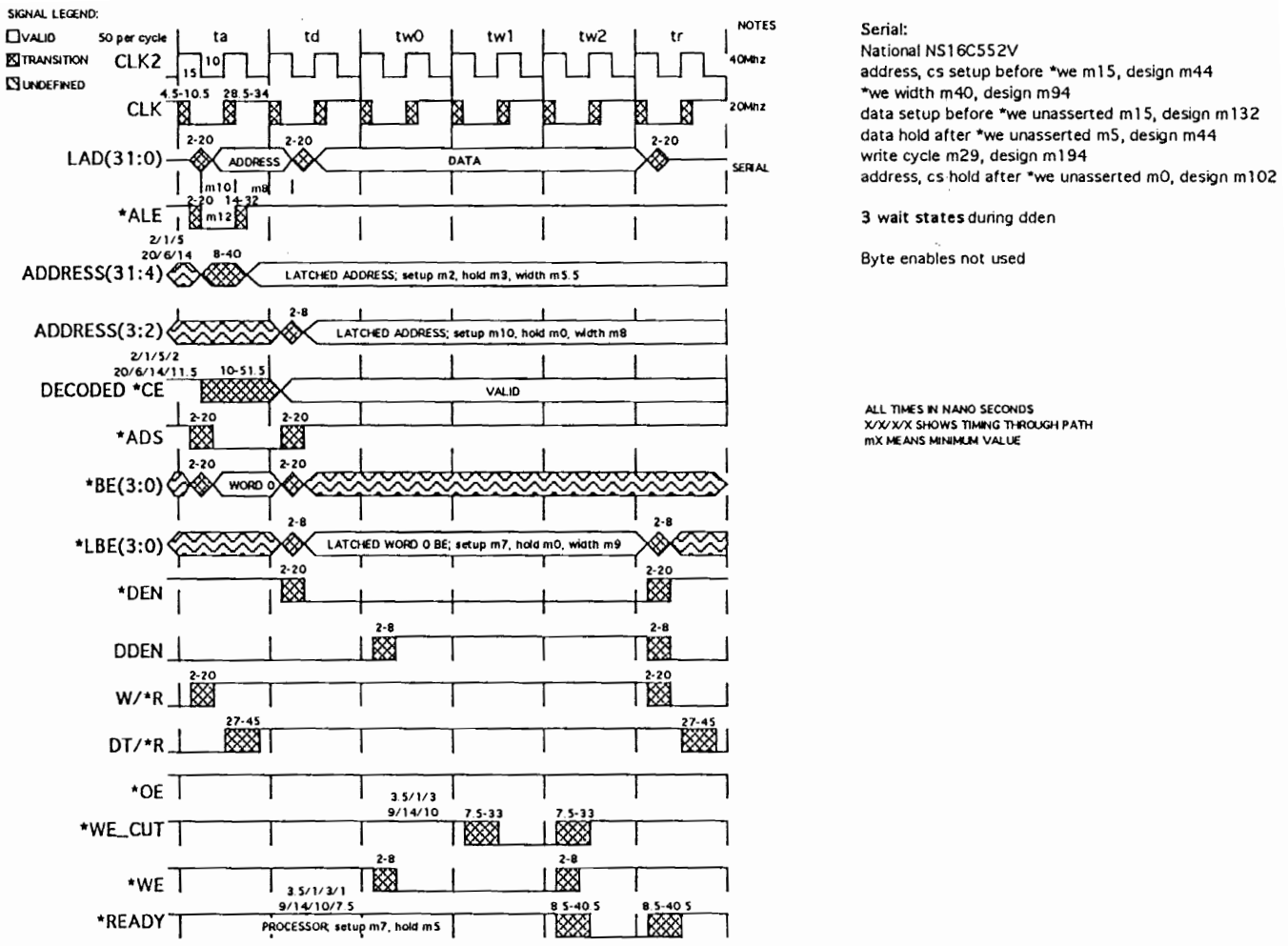


Figure 7.22 - Serial Single Write Worst Case Timing Analysis

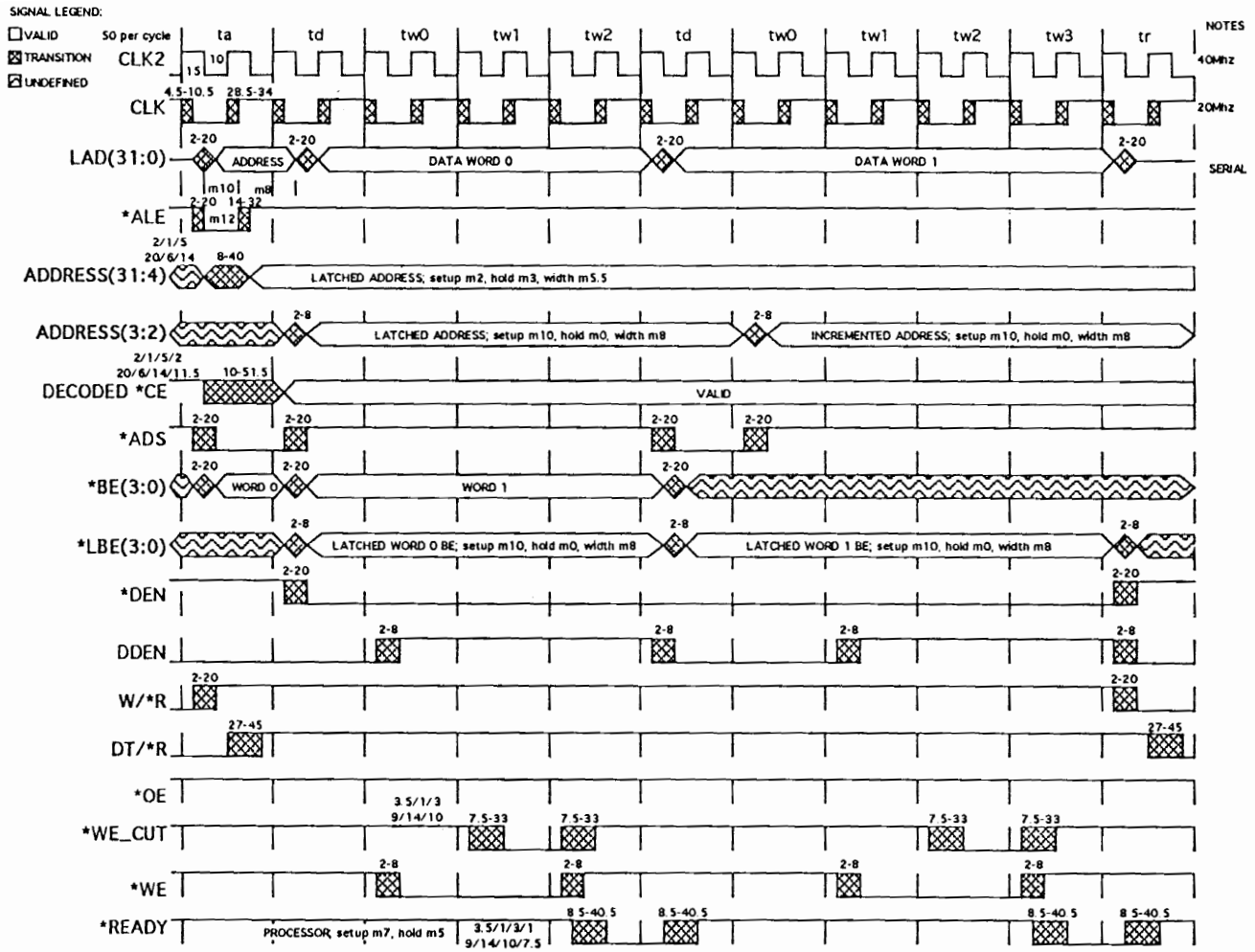


Figure 7.23 - Serial Burst Write Worst Case Timing Analysis

All analog components in the power supply section were analyzed for worst-case conditions. The analog components of the UTC oscillator and the microprocessor section were also analyzed for worst-case conditions. No operating parameters for the components are over the maximum specified limits.

7.5 IAE DATA PROCESSING

The HAP interfaces with the IAE through the IAE interface section. The IAE data processing algorithm is described in Chapters 4.3 and 6.3.4. Data is processed whenever the IAE has data available, and a detection mechanism is used to determine if any processing has been missed. The IAE algorithms have been tested extensively. The testing is grouped into two types of tests, Earth rate testing, and star sighting testing.

Earth rate tests were performed with no special test equipment. The IMU was simply placed on a stabile surface and the HAP attitude data was recorded periodically. After an hour or more of data had been collected, the attitude data was analyzed. The HAP attitude and time data was processed to determine the rotation rate of the IMU, this rate was compared to the know rotation rate of the Earth. This test was performed with the IMU in several orientations. The HAP and IMU were able to track the Earth rate to within 0.005 degrees per hour in some cases. Worst-case Earth rate data was below the specification limit of 0.1 degrees per hour. Figure 7.24 shows some of the Earth rate data taken.

Star sighting testing was performed using an equatorial mount tripod with right ascension and declination clock drives. The IMU and a telescope were mounted to the tripod and the tripod was aligned with Polaris, the North Star. With the clock drives enabled, the telescope and IMU remain inertially fixed with respect to the Earth's frame of reference. Known stars were sighted in the

trigger	time	q0	q1	q2	q3	normality
10	16:44:59.8	.9994436	-.0006057	-.0248512	-.0222391	-.2264D-08
11	16:45:59.8	.9993681	-.0006559	-.0266442	-.0235155	.1122D-08
12	16:46:59.8	.9992894	-.0007068	-.0283962	-.0247746	.3032D-08
13	16:47:59.8	.9992055	-.0007528	-.0301595	-.0260438	-.3673D-08
14	16:48:59.8	.9991156	-.0008362	-.0319289	-.0273481	-.2385D-08
15	16:49:59.8	.9990221	-.0008769	-.0336742	-.0286367	-.3431D-08
16	16:50:59.8	.9989226	-.0009548	-.0354526	-.0299293	.2255D-08
17	16:51:59.8	.9988188	-.0010376	-.0372248	-.0312127	-.3832D-08
18	16:52:59.8	.9987096	-.0011052	-.0390080	-.0324995	-.5003D-08
19	16:53:59.8	.9985959	-.0011625	-.0407897	-.0337795	-.2453D-08
20	16:54:59.8	.9984791	-.0012023	-.0425259	-.0350662	.1472D-08
21	16:55:59.8	.9983578	-.0012542	-.0442767	-.0363288	-.5860D-09
22	16:56:59.8	.9982303	-.0013213	-.0460551	-.0375969	-.6411D-09
23	16:57:59.8	.9980966	-.0013774	-.0478157	-.0389232	-.1981D-08
24	16:58:59.8	.9979585	-.0014197	-.0496008	-.0402064	-.4005D-08
25	16:59:59.8	.9978167	-.0014787	-.0513500	-.0415072	.7929D-09
26	17:00:59.8	.9976701	-.0015170	-.0530976	-.0428102	-.7986D-09
27	17:01:59.8	.9975202	-.0015715	-.0548500	-.0440730	.3882D-08
28	17:02:59.8	.9973632	-.0016378	-.0566316	-.0453518	.3117D-09
29	17:03:59.8	.9972026	-.0016895	-.0583856	-.0466391	.2378D-08
30	17:04:59.8	.9970379	-.0017393	-.0601351	-.0479175	-.6814D-09

#	time (sec)	angle (deg)	rate (rad/sec)	drift (deg/hr)
10	540.000	2.258316908	.72990862D-04	.01437918
11	600.000	2.510712063	.73033653D-04	.02320550
12	660.000	2.758161094	.72937867D-04	.00344818
13	720.000	3.007387075	.72901120D-04	-.00413150
14	780.000	3.259637739	.72937706D-04	.00341500
15	840.000	3.508501032	.72898684D-04	-.00463388
16	900.000	3.760827169	.72932019D-04	.00224180
17	960.000	4.012021911	.72940617D-04	.00401529
18	1020.000	4.264436324	.72969073D-04	.00988486
19	1080.000	4.516238747	.72984478D-04	.01306228
20	1140.000	4.764247779	.72940184D-04	.00392611
21	1200.000	5.012074273	.72897665D-04	-.00484406
22	1260.000	5.262897872	.72900711D-04	-.00421581
23	1320.000	5.515998261	.72933584D-04	.00256475
24	1380.000	5.768455412	.72955464D-04	.00707768
25	1440.000	6.018848981	.72950508D-04	.00605558
26	1500.000	6.269230393	.72945808D-04	.00508607
27	1560.000	6.517408522	.72916819D-04	-.00089336
28	1620.000	6.769470831	.72931824D-04	.00220164
29	1680.000	7.019530564	.72924953D-04	.00078433
30	1740.000	7.268614369	.72908766D-04	-.00255439

Figure 7.24 - Earth Rate Test Data

telescope and ESC shutter triggers were sent to the HAP when the star was exactly in the center of the telescope reticle. The HAP provided attitude and time data was stored and analyzed. Many sightings during the same session were taken of known stars. The HERCULES software was exercised to perform the IMU alignment process. When this alignment was used to sight other known stars, an angular error could be determined from the HAP and IMU measured position to the known position of the star. The angular accuracy with which the telescope could be aligned to a star is 0.01 degrees. After rotating the IMU more than 10 times, the measured stellar position measurement errors were below 0.05 degrees. The IMU was also aligned while being hand held. When a proper alignment was performed, the stellar position measurement errors were also below 0.05 degrees. Figure 7.25 shows some of the stellar position measurement data taken in this configuration.

7.6 UTC CLOCK DRIFT

Time is required in the HERCULES system to perform a geolocation of an image. Propagation of the STS state vector is based on the time provided by the HAP UTC clock. If there is an error in the UTC clock, there will be a corresponding error in the STS position determination. An STS position error induces an error in the geolocation of the image.

To obtain accurate time, I have chosen a high stability temperature compensated crystal oscillator. The oscillator drives three independent UTC clock chips. Three UTC clocks chips were used to safeguard against losing time due to SEUs or other unforeseen phenomena. These chips count the oscillator pulses, and maintain the UTC time. Leap year compensation is designed into the UTC clock chip. Time is maintained in the UTC clock chips from hundredths of a second to years.

The time needs to be set accurately before launch. Time setting using the

sighting	name	time	normality
1	sirius	03:41:27.5	-.186806814D-08
2	sirius	03:42:20.7	+.450924187D-09
3	capella	03:43:23.7	-.541201195D-08
4	capella	03:45:28.0	+.941183131D-09
5	pollux	03:48: 1.4	+.216704366D-08
6	castor	03:50:54.7	-.145936829D-08
7	menkalina	03:57: 6.7	+.720497662D-09
10	capella	04:02: 2.9	-.182708759D-08
11	regulus	04:05:36.0	+.233049069D-08
13	alphard	04:13:35.6	+.311860493D-08
15	kochab	04:17:27.2	-.497699659D-10
16	polaris	04:20:29.4	+.851902549D-09
17	el naih	04:24:17.1	+.419294777D-08
18	merak	04:26:50.8	-.378494380D-08
20	dubhe	04:29:28.2	-.177290449D-08
21	denebola	04:32:59.8	+.277922529D-08
22	zozca	04:37: 9.6	+.362682995D-09
23	aliotha	04:53:11.9	-.164465852D-08
24	mizar	04:55:10.0	-.275579892D-09
25	alkard	04:56:59.5	-.176467219D-08

Star sightings 1, 2, and 15 used for alignment, alignment angle error 0.000

Angles in table are in degrees unless otherwise noted.

double stars		2nd star	star	Rt Ascention read	Rt Ascention truth	Declination read	Declination truth	angular error in arc minutes
1	2	15	3	78.93	79.03	45.94	45.99	5.167387
1	2	15	4	78.92	79.03	45.96	45.99	4.957593
1	2	15	5	116.16	116.21	28.05	28.04	2.868681
1	2	15	6	113.48	113.53	31.90	31.90	2.447835
1	2	15	7	89.65	89.74	44.93	44.95	3.877187
1	2	15	10	79.01	79.03	45.95	45.99	2.638418
1	2	15	11	151.92	151.99	12.05	12.00	5.362654
1	2	15	13	141.69	141.80	-8.59	-8.62	6.991413
1	2	15	16	37.47	35.58	89.18	89.23	3.288273
1	2	15	17	81.45	81.45	28.65	28.60	3.400289
1	2	15	18	165.27	165.35	56.35	56.42	5.134108
1	2	15	20	165.70	165.82	61.79	61.79	3.438972
1	2	15	21	177.03	177.17	14.65	14.61	8.393071
1	2	15	22	168.26	168.43	20.56	20.56	9.696571
1	2	15	23	-166.64	-166.56	55.95	55.99	3.553264
1	2	15	24	-159.26	-159.08	54.95	54.96	6.068354
1	2	15	25	-153.36	-153.18	49.33	49.34	7.237061

Figure 7.25 - Star Position Measurement Data

SET CLOCKS ON NEXT TRIGGER provides an accurate method for setting the UTC clocks. Using the setup shown in Figure 5.16, I have consistently set the clocks to better than 0.01 seconds. The clock is set between two IAE interrupts, and is set to within 0.0005 seconds. Using the setup shown in Figure 5.16, I have been able to verify this time setting accuracy is better than 0.01 seconds.

An error in time is accumulated when the oscillator frequency is not exactly 32.768KHz. Deviations in the oscillator frequency from the desired frequency are grouped into three different terms. One source of deviation is caused by the inaccuracy of exactly fixing the oscillator frequency. This deviation is called the setability error, and is independent of time. Another source of deviation is oscillator drift due to temperature. One more source of deviation is oscillator crystal aging.

The oscillator deviations are as follows:

setability = 1×10^{-7} seconds per second

drift = 5×10^{-8} seconds per second

aging = 3×10^{-9} seconds per day

The UTC clocks will be set 2 days before launch of the STS. HERCULES will be used on-orbit for less than 14 days. The STS will then return to Earth, and the UTC clocks will be read on the ground within 2 days after landing.

To find the maximum time error at the end of the 18 day period, each of the error sources is added. The analysis is shown below.

seconds in 18 days = $18 \times 24 \times 60 \times 60$ = 1,555,200 seconds

error due to setability term:

1555200×10^{-7} = 0.15552 seconds

error due to drift term:

$1555200 \times 5 \times 10^{-8}$ = 0.07776 seconds

error due to aging term:

$1555200 \times 3 \times 10^{-9} = 0.00466$ seconds

maximum time error in an 18 day mission = 0.23794 seconds

The maximum time error specified in Chapter 4.1 is 0.25 seconds, and my design meets this requirement. Using the time read at landing, the actual drift and aging of the oscillator can be determined. Assuming that this was somewhat linear, the stored UTC times of the images will be adjusted and the geolocation will be recomputed.

7.7 UTC CLOCK BATTERY SIZING

Alkaline batteries are used to keep the UTC clock circuits powered. These batteries have to last, at a minimum from the time they are installed, 2 days before launch, to the time when the UTC clock are read 2 days after landing. The batteries power the three UTC clock chips, a CMOS NOR gate chip, the high stability temperature compensated oscillator, and a low power linear regulator.

The current requirements, at +5 volts, for each of the components is given below.

3 -	DP8571A, UTC clock chips	=	240 uA total
1 -	CD4001B, NOR gate chip	=	50 uA
1 -	CO252-A58, oscillator	=	4.0 mA
1 -	LP2951ACN, regulator	=	0.12 mA
	total current	=	4.41 mA
	battery capacity required	=	18 x 24 x 0.00441
		=	1.90 amp-hour

A 7 amp-hour battery pack configuration was chosen for the HAP. There are six D-cell batteries in the battery pack. The nominal battery pack voltage is 9 volts. This battery pack capacity will provide enough margin to allow launch slips and recovery delays without worrying about loosing UTC knowledge.

7.8 HAP SOFTWARE TESTING

The HAP software listing is in Appendix E and Appendix F. I have tested every routine in the HAP software. The software is fully functional, and no known problems exist.

The HAP sends data packets to the PGSC twice every second, this rate was determined by the processing rate of the PGSC. The initial versions of the HAP software sent the PGSC data packets four times per second, but the PGSC HERCULES software was unable to process the HAP information at that rate. All of the HAP commands discussed in Chapter 6.3.2.2.2.3 have been tested, and are functioning properly. Memory loads and memory dumps have been tested, and are used every time the UTC clocks are set. Memory dumps are also performed to gather the IMU monitor point data. The IMU monitor data points are requested by the PGSC every time the PGSC HERCULES software is started or when the PGSC HERCULES software calculates an IMU alignment.

I have tested the ability of the HAP to report the time of an ESC shutter trigger accurately. Using the clock setting configuration shown in Figure 5.16, I set the UTC clock. After setting the clock, I allowed the clock setting hardware to continue to send a trigger once every minute. Each time a trigger was received, I stored the time tag assigned to that trigger from the HAP. The HAP never reported a time tag which was off by 0.01 seconds or more. This time tagging capability

meets the requirements given in Chapter 4.1.

The HAP uses a watchdog timer to protect against SEU disruption of the program flow. The only truly critical piece of HAP information is the UTC. HAP hardware is triply redundant for maintaining UTC. HAP software reads and corrects the UTC clock time twice per second while the HAP is operating. SEUs will have to disrupt the same time component in two of the three clock chips within 0.5 seconds for the time to be lost while HAP is operating.

Self-testing of all peripherals is performed whenever the HAP is reset or whenever a fault generates a restart event. The self-testing has been verified in several ways. PROMs were checked by forcing the data in the PROMs to generate a non-zero checksum. The RAMS were tested by removing a RAM from the socket and verifying that the missing RAM chip was detected. The UTC clock chips were tested by removing a clock chip from the socket and verifying that the missing clock chip was detected. The UART was tested by disconnecting a data pin from the chip and verifying the missing data pin was detected.

The IAE is tested continuously by the executive. IAE monitor points and read and comparing to the predetermined limits for each of the monitor points.

8.0 CONCLUSION

The HAP hardware design has been completed and tested. A worst-case analysis of the design shows there are no design flaws. Two HAP units have been built and tested. Both of the units are functioning properly. The HAP hardware design has meet or exceed all design requirements.

HAP software has been completed. Like any software endeavor, the road to the final software version was long and studded with potholes. I have spent many hours debugging and writing the HAP operational software. The software has been fully written and tested. All software design requirements were meet or exceeded. I have developed a variation of the operational software to simulate IMU data for astronaut training purposes. The two HAP units built have the operational software installed, and are being actively used with the rest of the HERCULES system for STS integration testing and astronaut training at NASA JSC.

The entire HERCULES system has been tested. Testing the system required persistence. Numerous IMU hardware failures slowed the progress of system and HAP testing. After a final, working IMU was delivered, the HERCULES and HAP testing progressed rapidly. Testing the system proved to be a personal and technical challenge. The IMU is able read the Earth's rotation rate. Initial testing of the system was done using a two axis rotation fixture. Final testing required a highly precise angular target field. The only accurate targets available

were the stars. Testing was performed, mostly in the cold winter months, outdoors after dusk. It was a constant task to keep both body and equipment warm while accurately sighting the stars. Ultimately, hand held tests of the ESC and IMU stack were performed. This testing has fully checked the HERCULES and HAP software as they will be used on the STS during flight. An error estimate for the HERCULES system is shown in Figure 8.1. This estimate shows that the system geolocation goals of 1 to 2 nautical miles will be meet.

The need to observe and study terrestrial phenomena is still great. The HERCULES system will be used extensively in the future. The first flight is scheduled in October of 1992, and a second flight is being pursued in May of 1993. Enhancements and improvements to the system will be made in the future.

As shown in Figure 8.1, the single largest error source in the HERCULES system is due to inaccurate state vectors. Plans are being made to incorporate GPS position and time determination into the HERCULES system on orbit. The HAP will house the GPS receiver and provide the HERCULES system interface to GPS. With the GPS receiver, position and time errors will be measured in feet and microseconds, instead of miles and seconds.

Gyro technology will also be monitored. The current IMU is a cylindrical canister slightly larger than a large coffee can. As small accurate gyros become available, they will be incorporated into the HERCULES system to allow an easier to handle compact ESC/IMU stack.

The ESC will also be improved. Plans are under way to increase the CCD size to 2K x 2K which will increase resolution. CCDs are being investigated to allow the incorporation of color and multispectral-imagery.

	Geolocation Error (nmi)	
Error Sources	Worst Case	Typical Case
Gyro/Attititude	0.80	0.18
Initial State Vector Inaccuracy	5.71	0.93
State Vector Modelling	0.25	0.03
Geolocation Mismodelling	0.60	0.16
Time	1.01	0.56
Lens change and zoom	0.81	0.04
RSS Total at Nadir	5.95	1.11
20 degrees off-nadir, 1 nmi target alt	5.96 RSS	1.17 RSS
45 degrees off-nadir, 5 nmi target alt	7.77 RSS	5.12 RSS

Figure 8.1 - HERCULES Error Budget

The HAP hardware and HAP software have been fully built and tested. A flight HAP unit was delivered for STS integration along with the rest of the HERCULES system on April 1, 1992. I introduced the HERCULES system to the STS-53 astronauts, and am pleased with their excitement about the system. The astronauts are now training with the HERCULES system for their flight in October 1992. I am anxiously awaiting the first flight success of the HERCULES system and the HAP.

References

[1] J. Thibodeau, *Latitude Longitude Locator (Middeck), Payload Integration Plan*, NSTS 21097, NASA JSC, Houston, Texas, May 1988

[2] J. Thibodeau, *Cargo Systems Manual: Latitude Longitude Locator*, JSC 23727, NASA JSC, Houston, Texas, June 1989

[3] D. Holland, *NASA/NAVY Electronic Still Camera Project (ESC)*, technical meeting notes, NASA JSC, Houston, Texas, March 1990.

[4] NASA JSC, *Shuttle/Payload Interface Definition Document for the Payload and General Support Computer (PGSC)*, NSTS 21000-IDD-PSC, Houston, Texas, 1990.

[5] M. T. Soyka, P. J. Melvin, *HERCULES: Gyro-Based, Real-Time Geolocation for an Astronaut Camera*, (AAS-130), Presented at the 1st Annual AAS/AIAA Spaceflight Mechanics Meeting, Houston, Texas, Feb. 11-13, 1991.

[6] NASA JSC, *Shuttle/Payload Interface Definition Document for Middeck Accommodations*, NSTS 21000-IDD-MDK, Houston, Texas, 1988.

[7] Integrated Device Technology, *IDT RISC R3000 Family Product Information*, Santa Clara, California, 1990.

[8] LSI Logic, *LR3000 High Performance RISC Microprocessor and LR3010 Floating-Point Accelerator*, Milpitas, California, 1988.

[9] G. Kane, *MIPS RISC Architecture*, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[10] Motorola, *MC68040: 32-Bit Third-Generation Microprocessor*, Phoenix, Arizona, 1989.

[11] Motorola, *Programmer's Reference Manual*, M68000PM/AD, Phoenix, Arizona, 1989.

[12] Intel, *80960MC Hardware Designer's Reference Manual*, Santa Clara, California, 1989.

[13] Intel, *80960MC Programmer's Reference Manual*, Santa Clara, California, 1988.

[14] Intel, *16-/32-Bit Embedded Processors*, Santa Clara, California, 1991.

[15] IEEE standard, *IEEE 754-1985 standard for Binary Floating-Point Arithmetic*, New York, New York, August 1985.

[16] Harris Corporation, *Rad-Hard/Hi-Rel CICD Data Book, Radiation Effects on CMOS*, Melbourne, Florida, 1987.

[17] Raytheon, *R29793 Power Switched Bipolar PROM*, Mountain View California, 1989.

[18] Micron, *MOS Data Book*, Boise, Idaho, 1990.

[19] National Semiconductor, *Advanced Peripherals, Real Time Clock, Handbook*, Santa Clara, California, 1989.

[20] Vectron, *Crystal Oscillators*, Norwalk, Connecticut, 1989.

[21] Trimble Navigation, *Trimpack, GPS Receiver Users Manual*, Sunnyvale, California, 1991.

[22] National Semiconductor, *Advanced Peripherals, Data Communication/LAN/UARTs*, Santa Clara, California, 1989.

[23] EIA Standard, *EIA-232-C, Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange*, Electronic Industries Association, Washington D.C., 1987.

[24] National Semiconductor, *Interface Data Book*, Santa Clara, California, 1989.

[25] Interpoint, *DC-DC Converters*, Redmond, Washington, 1990.

[26] International Rectifier, *Hexfets*, El Segundo, California, 1982.

[27] Intel, *ASM-960*, Version 3.3, Santa Clara, California, 1991.

[28] Intel, *iC-960*, Version 3.0, Santa Clara, California, 1991.

[29] H.J. Buschelberger, Dr. E. Handrich, Dr. H. Malthan, G. Schmidt, *Laser Gyros in System Application with Rate-Bias Technique*, Presented at the Symposium for Gyro Technology, Stuttgart, Germany, 1987.

[30] R. G. Reeves, A. Anson, D. Landen, *Manual of Remote Sensing*, American Society of Photogrammetry, Falls Church, Virginia, 1975.

[31] E. C. Barrett, L. F. Curtis, *Environmental Remote Sensing 2: Practices and Problems*, Crane Russak & Company Inc., New York, New York, 1977.

[32] C.V. Heer, *History of the Laser Gyro*, Presented at Physics of Optical Ring Gyros, Snowbird, Utah, 1984.

[33] J. Wertz, *Spacecraft Attitude Determination and Control*, Kluwer Academic Publishers, Dordrecht, Netherlands, 1978.

[34] Texas Instruments, *AS/ALS Logic Data Book*, Richardson, Texas, 1986.

[35] A. A. Morgan, *Algorithms for ARU*, Honeywell Interoffice Correspondence, Tampa, Florida, January 1991.

[36] M. T. Soyka, P. J. Melvin, R. F. Higgins, *Algorithm Testing for the Gyro-Based, Astronaut-Held Latitude-Longitude-Locator*, (AAS 91-533), Presented at the Astrodynamics Specialist Conference, Durango, Colorado, August 19-22, 1991.

[37] L. B. Jackson, *Digital Filters and Signal Processing*, Kluwer Academic Publishers, Norwell, MA, 1986.

[38] D. P. Siewiorek, R. S. Swarz, *The Theory and Practice of Reliable System Design*, Digital Press, Bedford, MA, 1982.

[39] J. G. Tront, J. R. Armstrong, J. V. Oak, *Software Techniques for Detecting Single Event Upsets in Satellite Computers*, IEEE Transactions on Nuclear Science, Vol. NS-32, No. 6, December 1985.

[40] M. Z. Khan, J. G. Tront, *Detection of Upset Induced Execution Errors in Microprocessors*, Presented at the Phoenix Conference on Computers and Communications, Phoenix, Arizona, March 1989.

List of Acronyms

ADS	Microprocessor Address Cycle Strobe Signal
ALE	Microprocessor Address Latch Enable Signal
BADACC	Microprocessor Bad Access Signal
BE	Microprocessor Byte Enable Signal
CACHE	Microprocessor Cache signal
CCD	Charge Coupled Device
CCIU	Camera Computer Interface Unit
CCTV	Closed Captioned Television System
CLK	Microprocessor Clock Signal
CLK2	Microprocessor 2 Times Clock Signal
CRC	Coning Recovery Compensator
DDEN	Microprocessor Delayed Data Enable Signal
DEN	Microprocessor Data Enable Signal
DMA	Direct Memory Access
DT/*R	Microprocessor Data Transmit or Receive Signal
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
EPROM	Erasable Programmable Read Only Memory
ESC	Electronic Still Camera
ESCEB	Electronic Still Camera Electronics Box
FIFO	First In First Out memory
FIR	Finite Impulse Response
GPS	Global Positioning Satellite
GRiD	Graphic Retrieval and Information Display computer
HAP	HERCULES Attitude Processor
HERCULES	Hand held, Earth oriented, Real time, Cooperative, User friendly, Location, Targeting, and Environmental System
HeNe	Helium-Neon
IAC	Microprocessor Interagent Communication
IAE	Inertial Attitude Electronics
IAE_DS	Microprocessor Signal
IER	Interrupt Enable Register
IIR	Infinite Impulse Response
IMI	Initial Memory Image
IMU	Inertial Measurement Unit
I/O	Input-Output
JSC	Johnson Space Center

LAD	Microprocessor Latched Address and Data
LBE	Microprocessor Latched Byte Enable Signal
LED	Light Emitting Diode
LOCK	Microprocessor Lock Signal
L-bus	Microprocessor communication bus
L-cubed	Latitude and Longitude Locator
MIPS	Million Instructions Per Second
NASA	National Aeronautics and Space Administration
NCST	Naval Center for Space Technology
NRL	Naval Research Laboratory
OE	Microprocessor Output Enable Signal
PAL	Programmable Array logic
PCU	Pulse Conversion Unit
PCB	Microprocessor Process Control Block
PCB	Printed Circuit Board
PFW	Microprocessor Power Fail Warning Signal
PGSC	Payload and General Support Computer
PPM	Pulse Per Minute
PPS	Pulse Per Second
PRCB	Microprocessor Processor Control Block
PROM	Programmable Read Only Memory
RAM	Random Access Memory
READY	Microprocessor Ready Signal
RISC	Reduced Instruction Set Computer
RLG	Ring Laser Gyro
RREADY	Microprocessor Read Ready Signal
SEU	Single Event Upset
STS	Space Transportation System
Ta	Microprocessor L-bus Address State
Td	Microprocessor L-bus Data State
Ti	Microprocessor L-bus Idle State
Tr	Microprocessor L-bus Recovery State
Tw	Microprocessor L-bus Wait State
UART	Universal Asynchronous Receiver Transmitter
UTC	Universal Time Coordinated
WE	Microprocessor Write Enable Signal
W/*R	Microprocessor Write or Read Signal
WREADY	Microprocessor Write Ready Signal
&	Designates Logical "and" Operation
+	Designates Logical "or" Operation
! or *	Designates Logical "not" Operation

Appendix A - Byte Enable Latch, PAL Equations and Test Vector Listing

```
MODULE HAPBYTE

TITLE 'Byte enable latch Pal
       for the 80960MC-20,
       used on the Hercules program, HAP box
       Robert Higgins July 31, 1991'


"Device type

    Ubyte1 DEVICE 'p20r4';


"Pin definitions

"Inputs
    clk2        PIN    1;
    _wready     PIN    2;
    _rready     PIN    3;
    _ads        PIN    4;
    clk         PIN    5;
    _be3        PIN    6;
    _be2        PIN    7;
    _be1        PIN    8;
    _be0        PIN    9;
    _oe         PIN    13;

"Outputs
    _lbe3       PIN    20;
    _lbe2       PIN    19;
    _lbe1       PIN    18;
    _lbe0       PIN    17;

"Unused
    unused1     PIN    10;
    unused2     PIN    11;
    unused3     PIN    14;
    unused4     PIN    23;
    unused5     PIN    15;
    unused6     PIN    16;
    unused7     PIN    21;
    unused8     PIN    22;


"Define constants
    _be      = [_be3, _be2, _be1, _be0];
    _lbe     = [_lbe3,_lbe2,_lbe1,_lbe0];
    zero     = 0 ;
    one      = 1 ;


"latched byte enable state diagram

"byte enable 0
State_diagram [_lbe0]
```

```
    State zero:    CASE                                !clk  : zero;
                                    clk &  _ads & _rready : zero;
                                    clk &  _ads & _wready : zero;
                          clk &  _ads & _rready & _wready : zero;
                                       clk & !_ads &  _be0 : one;
                                       clk & !_ads & !_be0 : zero;
                       clk & !_rready & !_wready &  _be0 : one;
                       clk & !_rready & !_wready & !_be0 : zero;
                   ENDCASE;

    State one:     CASE                                !clk  : one;
                                    clk &  _ads & _rready : one;
                                    clk &  _ads & _wready : one;
                          clk &  _ads & _rready & _wready : one;
                                       clk & !_ads &  _be0 : one;
                                       clk & !_ads & !_be0 : zero;
                       clk & !_rready & !_wready &  _be0 : one;
                       clk & !_rready & !_wready & !_be0 : zero;
                   ENDCASE;

"byte enable 1
State_diagram [_lbe1]

    State zero:    CASE                                !clk  : zero;
                                    clk &  _ads & _rready : zero;
                                    clk &  _ads & _wready : zero;
                          clk &  _ads & _rready & _wready : zero;
                                       clk & !_ads &  _be1 : one;
                                       clk & !_ads & !_be1 : zero;
                       clk & !_rready & !_wready &  _be1 : one;
                       clk & !_rready & !_wready & !_be1 : zero;
                   ENDCASE;

    State one:     CASE                                !clk  : one;
                                    clk &  _ads & _rready : one;
                                    clk &  _ads & _wready : one;
                          clk &  _ads & _rready & _wready : one;
                                       clk & !_ads &  _be1 : one;
                                       clk & !_ads & !_be1 : zero;
                       clk & !_rready & !_wready &  _be1 : one;
                       clk & !_rready & !_wready & !_be1 : zero;
                   ENDCASE;

"byte enable 2
State_diagram [_lbe2]

    State zero:    CASE                                !clk  : zero;
                                    clk &  _ads & _rready : zero;
                                    clk &  _ads & _wready : zero;
                          clk &  _ads & _rready & _wready : zero;
                                       clk & !_ads &  _be2 : one;
                                       clk & !_ads & !_be2 : zero;
                       clk & !_rready & !_wready &  _be2 : one;
                       clk & !_rready & !_wready & !_be2 : zero;
                   ENDCASE;

    State one:     CASE                                !clk  : one;
                                    clk &  _ads & _rready : one;
                                    clk &  _ads & _wready : one;
                          clk &  _ads & _rready & _wready : one;
                                       clk & !_ads &  _be2 : one;
                                       clk & !_ads & !_be2 : zero;
                       clk & !_rready & !_wready &  _be2 : one;
                       clk & !_rready & !_wready & !_be2 : zero;
                   ENDCASE;

"byte enable 3
State_diagram [_lbe3]

    State zero:    CASE                                !clk  : zero;
                                    clk &  _ads & _rready : zero;
                                    clk &  _ads & _wready : zero;
                          clk &  _ads & _rready & _wready : zero;
                                       clk & !_ads &  _be3 : one;
                                       clk & !_ads & !_be3 : zero;
                       clk & !_rready & !_wready &  _be3 : one;
                       clk & !_rready & !_wready & !_be3 : zero;
                   ENDCASE;
```

```
    State one:     CASE                                  !clk  : one;
                                          clk &  _ads & _rready : one;
                                          clk &  _ads & _wready : one;
                                clk &  _ads & _rready & _wready : one;
                                          clk & !_ads &  _be3  : one;
                                          clk & !_ads & !_be3  : zero;
                               clk & !_rready & !_wready &  _be3 : one;
                               clk & !_rready & !_wready & !_be3 : zero;
                    ENDCASE;


"byte enable 0 latch test vectors
Test_vectors ([clk2,clk,_ads,_rready,_wready,_be0] -> _lbe0)

"V0001
"Ta write transaction
                    [ .c., 1 ,  0 ,   0   ,   1   ,  0 ] ->  0;
                    [ .c., 0 ,  0 ,   0   ,   1   ,  0 ] ->  0;
                    [ .c., 1 ,  0 ,   0   ,   1   ,  0 ] ->  0;
"Td
                    [ .c., 0 ,  1 ,   0   ,   1   ,  0 ] ->  0;
                    [ .c., 1 ,  1 ,   0   ,   1   ,  0 ] ->  0;
"ready
                    [ .c., 0 ,  1 ,   0   ,   1   ,  0 ] ->  0;
                    [ .c., 1 ,  1 ,   0   ,   0   ,  0 ] ->  0;

"V0008
"Ta write transaction
                    [ .c., 0 ,  0 ,   0   ,   1   ,  0 ] ->  0;
                    [ .c., 1 ,  0 ,   0   ,   1   ,  0 ] ->  0;
"Td
                    [ .c., 0 ,  1 ,   0   ,   1   ,  0 ] ->  0;
                    [ .c., 1 ,  1 ,   0   ,   1   ,  0 ] ->  0;
"ready
                    [ .c., 0 ,  1 ,   0   ,   1   ,  1 ] ->  0;
                    [ .c., 1 ,  1 ,   0   ,   0   ,  1 ] ->  1;

"V0014
"Ta write transaction
                    [ .c., 0 ,  0 ,   0   ,   1   ,  1 ] ->  1;
                    [ .c., 1 ,  0 ,   0   ,   1   ,  1 ] ->  1;
"Td
                    [ .c., 0 ,  1 ,   0   ,   1   ,  1 ] ->  1;
                    [ .c., 1 ,  1 ,   0   ,   1   ,  1 ] ->  1;
"ready
                    [ .c., 0 ,  1 ,   0   ,   1   ,  1 ] ->  1;
                    [ .c., 1 ,  1 ,   0   ,   0   ,  1 ] ->  1;

"V0020
"Ta write transaction
                    [ .c., 0 ,  0 ,   0   ,   1   ,  1 ] ->  1;
                    [ .c., 1 ,  0 ,   0   ,   1   ,  1 ] ->  1;
"Td
                    [ .c., 0 ,  1 ,   0   ,   1   ,  1 ] ->  1;
                    [ .c., 1 ,  1 ,   0   ,   1   ,  1 ] ->  1;
"ready
                    [ .c., 0 ,  1 ,   0   ,   1   ,  0 ] ->  1;
                    [ .c., 1 ,  1 ,   0   ,   0   ,  0 ] ->  0;


"V0026
"Ta read transaction
                    [ .c., 0 ,  0 ,   1   ,   0   ,  0 ] ->  0;
                    [ .c., 1 ,  0 ,   1   ,   0   ,  0 ] ->  0;
"Td
                    [ .c., 0 ,  1 ,   1   ,   0   ,  0 ] ->  0;
                    [ .c., 1 ,  1 ,   1   ,   0   ,  0 ] ->  0;
"ready
                    [ .c., 0 ,  1 ,   1   ,   0   ,  0 ] ->  0;
                    [ .c., 1 ,  1 ,   0   ,   0   ,  0 ] ->  0;

"V0032
"Ta read transaction
                    [ .c., 0 ,  0 ,   1   ,   0   ,  0 ] ->  0;
                    [ .c., 1 ,  0 ,   1   ,   0   ,  0 ] ->  0;
"Td
                    [ .c., 0 ,  1 ,   1   ,   0   ,  0 ] ->  0;
                    [ .c., 1 ,  1 ,   1   ,   0   ,  0 ] ->  0;
```

```
"ready
                [ .c., 0 ,  1 ,   1   ,   0   ,  1 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   0   ,  1 ] ->  1;

"V0038
"Ta read transaction
                [ .c., 0 ,  0 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  0 ,   1   ,   0   ,  1 ] ->  1;
"Td
                [ .c., 0 ,  1 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   1   ,   0   ,  1 ] ->  1;
"ready
                [ .c., 0 ,  1 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   0   ,  1 ] ->  1;

"V0044
"Ta read transaction
                [ .c., 0 ,  0 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  0 ,   1   ,   0   ,  1 ] ->  1;
"Td
                [ .c., 0 ,  1 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   1   ,   0   ,  1 ] ->  1;
"ready
                [ .c., 0 ,  1 ,   1   ,   0   ,  0 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   0   ,  0 ] ->  0;


"byte enable 1 latch test vectors
Test_vectors ([clk2,clk,_ads,_rready,_wready,_be1] -> _lbe1)

"V0050
"Ta write transaction
                [ .c., 1 ,  0 ,   0   ,   1   ,  0 ] ->  0;
                [ .c., 0 ,  0 ,   0   ,   1   ,  0 ] ->  0;
                [ .c., 1 ,  0 ,   0   ,   1   ,  0 ] ->  0;
"Td
                [ .c., 0 ,  1 ,   0   ,   1   ,  0 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   1   ,  0 ] ->  0;
"ready
                [ .c., 0 ,  1 ,   0   ,   1   ,  0 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   0   ,  0 ] ->  0;

"V0057
"Ta write transaction
                [ .c., 0 ,  0 ,   0   ,   1   ,  0 ] ->  0;
                [ .c., 1 ,  0 ,   0   ,   1   ,  0 ] ->  0;
"Td
                [ .c., 0 ,  1 ,   0   ,   1   ,  0 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   1   ,  0 ] ->  0;
"ready
                [ .c., 0 ,  1 ,   0   ,   1   ,  1 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   0   ,  1 ] ->  1;

"V0063
"Ta write transaction
                [ .c., 0 ,  0 ,   0   ,   1   ,  1 ] ->  1;
                [ .c., 1 ,  0 ,   0   ,   1   ,  1 ] ->  1;
"Td
                [ .c., 0 ,  1 ,   0   ,   1   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   1   ,  1 ] ->  1;
"ready
                [ .c., 0 ,  1 ,   0   ,   1   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   0   ,  1 ] ->  1;

"V0069
"Ta write transaction
                [ .c., 0 ,  0 ,   0   ,   1   ,  1 ] ->  1;
                [ .c., 1 ,  0 ,   0   ,   1   ,  1 ] ->  1;
"Td
                [ .c., 0 ,  1 ,   0   ,   1   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   1   ,  1 ] ->  1;
"ready
                [ .c., 0 ,  1 ,   0   ,   1   ,  0 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   0   ,  0 ] ->  0;


"V0075
"Ta read transaction
                [ .c., 0 ,  0 ,   1   ,   0   ,  0 ] ->  0;
                [ .c., 1 ,  0 ,   1   ,   0   ,  0 ] ->  0;
```

```
"Td
                [ .c., 0 ,  1 ,   1   ,   0   ,  0 ] ->  0;
                [ .c., 1 ,  1 ,   1   ,   0   ,  0 ] ->  0;
"ready
                [ .c., 0 ,  1 ,   1   ,   0   ,  0 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   0   ,  0 ] ->  0;

"V0081
"Ta read transaction
                [ .c., 0 ,  0 ,   1   ,   0   ,  0 ] ->  0;
                [ .c., 1 ,  0 ,   1   ,   0   ,  0 ] ->  0;
"Td
                [ .c., 0 ,  1 ,   1   ,   0   ,  0 ] ->  0;
                [ .c., 1 ,  1 ,   1   ,   0   ,  0 ] ->  0;
"ready
                [ .c., 0 ,  1 ,   1   ,   0   ,  1 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   0   ,  1 ] ->  1;

"V0087
"Ta read transaction
                [ .c., 0 ,  0 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  0 ,   1   ,   0   ,  1 ] ->  1;
"Td
                [ .c., 0 ,  1 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   1   ,   0   ,  1 ] ->  1;
"ready
                [ .c., 0 ,  1 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   0   ,  1 ] ->  1;

"V0093
"Ta read transaction
                [ .c., 0 ,  0 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  0 ,   1   ,   0   ,  1 ] ->  1;
"Td
                [ .c., 0 ,  1 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   1   ,   0   ,  1 ] ->  1;
"ready
                [ .c., 0 ,  1 ,   1   ,   0   ,  0 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   0   ,  0 ] ->  0;


"byte enable 2 latch test vectors
Test_vectors ([clk2,clk,_ads,_rready,_wready,_be2] -> _lbe2)

"V0099
"Ta write transaction
                [ .c., 1 ,  0 ,   0   ,   1   ,  0 ] ->  0;
                [ .c., 0 ,  0 ,   0   ,   1   ,  0 ] ->  0;
                [ .c., 1 ,  0 ,   0   ,   1   ,  0 ] ->  0;
"Td
                [ .c., 0 ,  1 ,   0   ,   1   ,  0 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   1   ,  0 ] ->  0;
"ready
                [ .c., 0 ,  1 ,   0   ,   1   ,  0 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   0   ,  0 ] ->  0;

"V0106
"Ta write transaction
                [ .c., 0 ,  0 ,   0   ,   1   ,  0 ] ->  0;
                [ .c., 1 ,  0 ,   0   ,   1   ,  0 ] ->  0;
"Td
                [ .c., 0 ,  1 ,   0   ,   1   ,  0 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   1   ,  0 ] ->  0;
"ready
                [ .c., 0 ,  1 ,   0   ,   1   ,  1 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   0   ,  1 ] ->  1;

"V0112
"Ta write transaction
                [ .c., 0 ,  0 ,   0   ,   1   ,  1 ] ->  1;
                [ .c., 1 ,  0 ,   0   ,   1   ,  1 ] ->  1;
"Td
                [ .c., 0 ,  1 ,   0   ,   1   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   1   ,  1 ] ->  1;
"ready
                [ .c., 0 ,  1 ,   0   ,   1   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   0   ,  1 ] ->  1;
```

```
"V0118
"Ta write transaction
                [ .c., 0 ,  0 ,   0   ,   1   ,  1 ] ->  1;
                [ .c., 1 ,  0 ,   0   ,   1   ,  1 ] ->  1;
"Td
                [ .c., 0 ,  1 ,   0   ,   1   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   1   ,  1 ] ->  1;
"ready
                [ .c., 0 ,  1 ,   0   ,   1   ,  0 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   0   ,  0 ] ->  0;


"V0124
"Ta read transaction
                [ .c., 0 ,  0 ,   1   ,   0   ,  0 ] ->  0;
                [ .c., 1 ,  0 ,   1   ,   0   ,  0 ] ->  0;
"Td
                [ .c., 0 ,  1 ,   1   ,   0   ,  0 ] ->  0;
                [ .c., 1 ,  1 ,   1   ,   0   ,  0 ] ->  0;
"ready
                [ .c., 0 ,  1 ,   1   ,   0   ,  0 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   0   ,  0 ] ->  0;

"V0130
"Ta read transaction
                [ .c., 0 ,  0 ,   1   ,   0   ,  0 ] ->  0;
                [ .c., 1 ,  0 ,   1   ,   0   ,  0 ] ->  0;
"Td
                [ .c., 0 ,  1 ,   1   ,   0   ,  0 ] ->  0;
                [ .c., 1 ,  1 ,   1   ,   0   ,  0 ] ->  0;
"ready
                [ .c., 0 ,  1 ,   1   ,   0   ,  1 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   0   ,  1 ] ->  1;

"V0136
"Ta read transaction
                [ .c., 0 ,  0 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  0 ,   1   ,   0   ,  1 ] ->  1;
"Td
                [ .c., 0 ,  1 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   1   ,   0   ,  1 ] ->  1;
"ready
                [ .c., 0 ,  1 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   0   ,  1 ] ->  1;

"V0142
"Ta read transaction
                [ .c., 0 ,  0 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  0 ,   1   ,   0   ,  1 ] ->  1;
"Td
                [ .c., 0 ,  1 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   1   ,   0   ,  1 ] ->  1;
"ready
                [ .c., 0 ,  1 ,   1   ,   0   ,  0 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   0   ,  0 ] ->  0;


"byte enable 3 latch test vectors
Test_vectors ([clk2,clk,_ads,_rready,_wready,_be3] -> _lbe3)

"V0148
"Ta write transaction
                [ .c., 1 ,  0 ,   0   ,   1   ,  0 ] ->  0;
                [ .c., 0 ,  0 ,   0   ,   1   ,  0 ] ->  0;
                [ .c., 1 ,  0 ,   0   ,   1   ,  0 ] ->  0;
"Td
                [ .c., 0 ,  1 ,   0   ,   1   ,  0 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   1   ,  0 ] ->  0;
"ready
                [ .c., 0 ,  1 ,   0   ,   1   ,  0 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   0   ,  0 ] ->  0;

"V0155
"Ta write transaction
                [ .c., 0 ,  0 ,   0   ,   1   ,  0 ] ->  0;
                [ .c., 1 ,  0 ,   0   ,   1   ,  0 ] ->  0;
"Td
                [ .c., 0 ,  1 ,   0   ,   1   ,  0 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   1   ,  0 ] ->  0;
```

```
"ready
                [ .c., 0 ,  1 ,   0   ,   1   ,  1 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   0   ,  1 ] ->  1;

"V0161
"Ta write transaction
                [ .c., 0 ,  0 ,   0   ,   1   ,  1 ] ->  1;
                [ .c., 1 ,  0 ,   0   ,   1   ,  1 ] ->  1;
"Td
                [ .c., 0 ,  1 ,   0   ,   1   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   1   ,  1 ] ->  1;
"ready
                [ .c., 0 ,  1 ,   0   ,   1   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   0   ,  1 ] ->  1;

"V0167
"Ta write transaction
                [ .c., 0 ,  0 ,   0   ,   1   ,  1 ] ->  1;
                [ .c., 1 ,  0 ,   0   ,   1   ,  1 ] ->  1;
"Td
                [ .c., 0 ,  1 ,   0   ,   1   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   1   ,  1 ] ->  1;
"ready
                [ .c., 0 ,  1 ,   0   ,   1   ,  0 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   0   ,  0 ] ->  0;


"V0173
"Ta read transaction
                [ .c., 0 ,  0 ,   1   ,   0   ,  0 ] ->  0;
                [ .c., 1 ,  0 ,   1   ,   0   ,  0 ] ->  0;
"Td
                [ .c., 0 ,  1 ,   1   ,   0   ,  0 ] ->  0;
                [ .c., 1 ,  1 ,   1   ,   0   ,  0 ] ->  0;
"ready
                [ .c., 0 ,  1 ,   1   ,   0   ,  0 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   0   ,  0 ] ->  0;

"V0179
"Ta read transaction
                [ .c., 0 ,  0 ,   1   ,   0   ,  0 ] ->  0;
                [ .c., 1 ,  0 ,   1   ,   0   ,  0 ] ->  0;
"Td
                [ .c., 0 ,  1 ,   1   ,   0   ,  0 ] ->  0;
                [ .c., 1 ,  1 ,   1   ,   0   ,  0 ] ->  0;
"ready
                [ .c., 0 ,  1 ,   1   ,   0   ,  1 ] ->  0;
                [ .c., 1 ,  1 ,   0   ,   0   ,  1 ] ->  1;

"V0185
"Ta read transaction
                [ .c., 0 ,  0 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  0 ,   1   ,   0   ,  1 ] ->  1;
"Td
                [ .c., 0 ,  1 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   1   ,   0   ,  1 ] ->  1;
"ready
                [ .c., 0 ,  1 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   0   ,  1 ] ->  1;

"V0191
"Ta read transaction
                [ .c., 0 ,  0 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  0 ,   1   ,   0   ,  1 ] ->  1;
"Td
                [ .c., 0 ,  1 ,   1   ,   0   ,  1 ] ->  1;
                [ .c., 1 ,  1 ,   1   ,   0   ,  1 ] ->  1;
"ready
                [ .c., 0 ,  1 ,   1   ,   0   ,  0 ] ->  1;
                [ .c., 1 ,  1 ,   0   ,   0   ,  0 ] ->  0;

End;
```

Appendix B - Burst Control Logic, PAL Equation and Test Vector Listing

```
MODULE HAPBURST

TITLE 'Burst logic Pal for the 80960MC-20,
       used on the Hercules program, HAP box
       Robert Higgins july 15, 1991'


"Device type

    Ubur8 DEVICE 'p20r4';


"Pin definitions

"Inputs
    clk2      PIN    1;
    lad3      PIN    2;
    lad2      PIN    3;
    lad1      PIN    4;
    lad0      PIN    5;
    _ads      PIN    6;
    _den      PIN    7;
    _ready    PIN    8;
    clk       PIN    9;
    _oe       PIN    13;

"Outputs
    ad2       PIN    20;
    ad3       PIN    19;
    size0     PIN    18;
    size1     PIN    17;

"Unused
    unused1   PIN    10;
    unused2   PIN    11;
    unused3   PIN    14;
    unused4   PIN    15;
    unused5   PIN    16;
    unused6   PIN    21;
    unused7   PIN    22;
    unused8   PIN    23;


"Burst length

    size_in     = [lad1,lad0];
    size_out    = [size1,size0];


"First address

    addr_in     = [lad3,lad2];
    addr_out    = [ad3,ad2];
```

```
"Constants
    zero        = [ 0 , 0 ];
    one         = [ 0 , 1 ];
    two         = [ 1 , 0 ];
    three       = [ 1 , 1 ];


"States

    a = ^b00;
    b = ^b01;
    c = ^b10;
    d = ^b11;


"Size latch and counter
State_diagram [size1,size0]

    State a:    CASE clk & !_ads & _den & (size_in == zero)  : a;
                     clk & !_ads & _den & (size_in == one)   : b;
                     clk & !_ads & _den & (size_in == two)   : c;
                     clk & !_ads & _den & (size_in == three) : d;
                                          clk & !_ads & !_den  : a;
                                          clk &  _ads & !_den  : a;
                                          clk &  _den & _ads   : a;
                                                        !clk   : a;
                ENDCASE;

    State b:    CASE clk & !_ads & _den & (size_in == zero)  : a;
                     clk & !_ads & _den & (size_in == one)   : b;
                     clk & !_ads & _den & (size_in == two)   : c;
                     clk & !_ads & _den & (size_in == three) : d;
                                     clk & !_ads & !_den   : a;
                                     clk &  _ads & !_den   : b;
                                     clk &  _ads &  _den   : b;
                                                    !clk   : b;
                ENDCASE;

    State c:    CASE clk & !_ads & _den & (size_in == zero)  : a;
                     clk & !_ads & _den & (size_in == one)   : b;
                     clk & !_ads & _den & (size_in == two)   : c;
                     clk & !_ads & _den & (size_in == three) : d;
                                     clk & !_ads & !_den   : b;
                                     clk &  _ads & !_den   : c;
                                     clk &  _ads &  _den   : c;
                                                    !clk   : c;
                ENDCASE;

    State d:    CASE clk & !_ads & _den & (size_in == zero)  : a;
                     clk & !_ads & _den & (size_in == one)   : b;
                     clk & !_ads & _den & (size_in == two)   : c;
                     clk & !_ads & _den & (size_in == three) : d;
                                     clk & !_ads & !_den   : c;
                                     clk &  _ads & !_den   : d;
                                     clk &  _ads &  _den   : d;
                                                    !clk   : d;
                ENDCASE;


"Address latch and counter
State_diagram [ad3,ad2]

    State a:    CASE clk & !_ads &  _den & (addr_in == zero)    : a;
                     clk & !_ads &  _den & (addr_in == one)     : b;
                     clk & !_ads &  _den & (addr_in == two)     : c;
                     clk & !_ads &  _den & (addr_in == three)   : d;
                     clk & !_ads & !_den & !(size_out == zero)  : b;
                                     clk & _ads & !_den   : a;
                                     clk & _ads &  _den   : a;
                                                  !clk    : a;
                ENDCASE;

    State b:    CASE clk & !_ads &  _den & (addr_in == zero)    : a;
                     clk & !_ads &  _den & (addr_in == one)     : b;
                     clk & !_ads &  _den & (addr_in == two)     : c;
                     clk & !_ads &  _den & (addr_in == three)   : d;
                     clk & !_ads & !_den & !(size_out == zero)  : c;
                                     clk & _ads & !_den   : b;
                                     clk & _ads &  _den   : b;
                                                  !clk    : b;
                ENDCASE;
```

```
    State c:     CASE clk & !_ads &  _den & (addr_in == zero)    : a;
                      clk & !_ads &  _den & (addr_in == one)     : b;
                      clk & !_ads &  _den & (addr_in == two)     : c;
                      clk & !_ads &  _den & (addr_in == three)   : d;
                      clk & !_ads & !_den & !(size_out == zero)  : d;
                                            clk & _ads & !_den   : c;
                                            clk & _ads &  _den   : c;
                                                         !clk    : c;
                 ENDCASE;

    State d:     CASE clk & !_ads & _den & (addr_in == zero)     : a;
                      clk & !_ads & _den & (addr_in == one)      : b;
                      clk & !_ads & _den & (addr_in == two)      : c;
                      clk & !_ads & _den & (addr_in == three)    : d;
                                           clk & !_ads & !_den   : d;
                                           clk &  _ads & !_den   : d;
                                           clk &  _ads &  _den   : d;
                                                         !clk    : d;
                 ENDCASE;


Test_vectors
([ clk2, clk , _oe, size_in, addr_in, _ads, _den] -> [ size_out, addr_out])


"4 word transaction

"load verify
 [ .c. ,  1  ,  0 , three  ,  zero  ,  0  ,  1  ] -> [  three  ,  zero   ];
 [ .c. ,  0  ,  0 , three  ,  zero  ,  0  ,  0  ] -> [  three  ,  zero   ];
 [ .c. ,  0  ,  0 , three  ,  zero  ,  0  ,  1  ] -> [  three  ,  zero   ];
 [ .c. ,  0  ,  0 , three  ,  zero  ,  1  ,  0  ] -> [  three  ,  zero   ];
 [ .c. ,  0  ,  0 , three  ,  zero  ,  1  ,  1  ] -> [  three  ,  zero   ];
"hold
 [ .c. ,  1  ,  0 , three  ,  zero  ,  1  ,  0  ] -> [  three  ,  zero   ];
 [ .c. ,  1  ,  0 , three  ,  zero  ,  1  ,  1  ] -> [  three  ,  zero   ];

"v0008
"increment verify
 [ .c. ,  1  ,  0 , three  ,  zero  ,  0  ,  0  ] -> [  two    ,  one    ];
 [ .c. ,  0  ,  0 , three  ,  zero  ,  0  ,  0  ] -> [  two    ,  one    ];
 [ .c. ,  0  ,  0 , three  ,  zero  ,  0  ,  1  ] -> [  two    ,  one    ];
 [ .c. ,  0  ,  0 , three  ,  zero  ,  1  ,  0  ] -> [  two    ,  one    ];
 [ .c. ,  0  ,  0 , three  ,  zero  ,  1  ,  1  ] -> [  two    ,  one    ];
"hold
 [ .c. ,  1  ,  0 , three  ,  zero  ,  1  ,  0  ] -> [  two    ,  one    ];
 [ .c. ,  1  ,  0 , three  ,  zero  ,  1  ,  1  ] -> [  two    ,  one    ];

"v0015
"increment verify
 [ .c. ,  1  ,  0 , three  ,  zero  ,  0  ,  0  ] -> [  one    ,  two    ];
 [ .c. ,  0  ,  0 , three  ,  zero  ,  0  ,  0  ] -> [  one    ,  two    ];
 [ .c. ,  0  ,  0 , three  ,  zero  ,  0  ,  1  ] -> [  one    ,  two    ];
 [ .c. ,  0  ,  0 , three  ,  zero  ,  1  ,  0  ] -> [  one    ,  two    ];
 [ .c. ,  0  ,  0 , three  ,  zero  ,  1  ,  1  ] -> [  one    ,  two    ];
"hold
 [ .c. ,  1  ,  0 , three  ,  zero  ,  1  ,  0  ] -> [  one    ,  two    ];
 [ .c. ,  1  ,  0 , three  ,  zero  ,  1  ,  1  ] -> [  one    ,  two    ];

"v0022
"increment verify
 [ .c. ,  1  ,  0 , three  ,  zero  ,  0  ,  0  ] -> [  zero   ,  three  ];
 [ .c. ,  0  ,  0 , three  ,  zero  ,  0  ,  0  ] -> [  zero   ,  three  ];
 [ .c. ,  0  ,  0 , three  ,  zero  ,  0  ,  1  ] -> [  zero   ,  three  ];
 [ .c. ,  0  ,  0 , three  ,  zero  ,  1  ,  0  ] -> [  zero   ,  three  ];
 [ .c. ,  0  ,  0 , three  ,  zero  ,  1  ,  1  ] -> [  zero   ,  three  ];
"hold
 [ .c. ,  1  ,  0 , three  ,  zero  ,  1  ,  0  ] -> [  zero   ,  three  ];
 [ .c. ,  1  ,  0 , three  ,  zero  ,  1  ,  1  ] -> [  zero   ,  three  ];

"v0029
"3 word transaction with starting address zero
 [ .c. ,  1  ,  0 , two    ,  zero  ,  0  ,  1  ] -> [  two    ,  zero   ];
 [ .c. ,  0  ,  0 , two    ,  zero  ,  0  ,  0  ] -> [  two    ,  zero   ];
 [ .c. ,  0  ,  0 , two    ,  zero  ,  0  ,  1  ] -> [  two    ,  zero   ];
 [ .c. ,  0  ,  0 , two    ,  zero  ,  1  ,  0  ] -> [  two    ,  zero   ];
 [ .c. ,  0  ,  0 , two    ,  zero  ,  1  ,  1  ] -> [  two    ,  zero   ];
```

```
"hold
 [ .c. , 1 , 0 , two   , zero , 1 , 0 ] -> [ two   , zero ];
 [ .c. , 1 , 0 , two   , zero , 1 , 1 ] -> [ two   , zero ];

"v0036
"increment verify
 [ .c. , 1 , 0 , two   , zero , 0 , 0 ] -> [ one   , one  ];
 [ .c. , 0 , 0 , two   , zero , 0 , 0 ] -> [ one   , one  ];
 [ .c. , 0 , 0 , two   , zero , 0 , 1 ] -> [ one   , one  ];
 [ .c. , 0 , 0 , two   , zero , 1 , 0 ] -> [ one   , one  ];
 [ .c. , 0 , 0 , two   , zero , 1 , 1 ] -> [ one   , one  ];
"hold
 [ .c. , 1 , 0 , two   , zero , 1 , 0 ] -> [ one   , one  ];
 [ .c. , 1 , 0 , two   , zero , 1 , 1 ] -> [ one   , one  ];

"v0043
"increment verify
 [ .c. , 1 , 0 , two   , zero , 0 , 0 ] -> [ zero  , two  ];
 [ .c. , 0 , 0 , two   , zero , 0 , 0 ] -> [ zero  , two  ];
 [ .c. , 0 , 0 , two   , zero , 0 , 1 ] -> [ zero  , two  ];
 [ .c. , 0 , 0 , two   , zero , 1 , 0 ] -> [ zero  , two  ];
 [ .c. , 0 , 0 , two   , zero , 1 , 1 ] -> [ zero  , two  ];
"hold
 [ .c. , 1 , 0 , two   , zero , 1 , 0 ] -> [ zero  , two  ];
 [ .c. , 1 , 0 , two   , zero , 1 , 1 ] -> [ zero  , two  ];


"v0050
"3 word transaction with starting address one
 [ .c. , 1 , 0 , two   , one  , 0 , 1 ] -> [ two   , one  ];
 [ .c. , 0 , 0 , two   , one  , 0 , 0 ] -> [ two   , one  ];
 [ .c. , 0 , 0 , two   , one  , 0 , 1 ] -> [ two   , one  ];
 [ .c. , 0 , 0 , two   , one  , 1 , 0 ] -> [ two   , one  ];
 [ .c. , 0 , 0 , two   , one  , 1 , 1 ] -> [ two   , one  ];
"hold
 [ .c. , 1 , 0 , two   , one  , 1 , 0 ] -> [ two   , one  ];
 [ .c. , 1 , 0 , two   , one  , 1 , 1 ] -> [ two   , one  ];

"v0057
"increment verify
 [ .c. , 1 , 0 , two   , one  , 0 , 0 ] -> [ one   , two  ];
 [ .c. , 0 , 0 , two   , one  , 0 , 0 ] -> [ one   , two  ];
 [ .c. , 0 , 0 , two   , one  , 0 , 1 ] -> [ one   , two  ];
 [ .c. , 0 , 0 , two   , one  , 1 , 0 ] -> [ one   , two  ];
 [ .c. , 0 , 0 , two   , one  , 1 , 1 ] -> [ one   , two  ];
"hold
 [ .c. , 1 , 0 , two   , one  , 1 , 0 ] -> [ one   , two  ];
 [ .c. , 1 , 0 , two   , one  , 1 , 1 ] -> [ one   , two  ];

"v0064
"increment verify
 [ .c. , 1 , 0 , two   , one  , 0 , 0 ] -> [ zero  , three];
 [ .c. , 0 , 0 , two   , one  , 0 , 0 ] -> [ zero  , three];
 [ .c. , 0 , 0 , two   , one  , 0 , 1 ] -> [ zero  , three];
 [ .c. , 0 , 0 , two   , one  , 1 , 0 ] -> [ zero  , three];
 [ .c. , 0 , 0 , two   , one  , 1 , 1 ] -> [ zero  , three];
"hold
 [ .c. , 1 , 0 , two   , one  , 1 , 0 ] -> [ zero  , three];
 [ .c. , 1 , 0 , two   , one  , 1 , 1 ] -> [ zero  , three];


"v0071
"2 word transaction with starting address zero
 [ .c. , 1 , 0 , one   , zero , 0 , 1 ] -> [ one   , zero ];
 [ .c. , 0 , 0 , one   , zero , 0 , 0 ] -> [ one   , zero ];
 [ .c. , 0 , 0 , one   , zero , 0 , 1 ] -> [ one   , zero ];
 [ .c. , 0 , 0 , one   , zero , 1 , 0 ] -> [ one   , zero ];
 [ .c. , 0 , 0 , one   , zero , 1 , 1 ] -> [ one   , zero ];
"hold
 [ .c. , 1 , 0 , one   , zero , 1 , 0 ] -> [ one   , zero ];
 [ .c. , 1 , 0 , one   , zero , 1 , 1 ] -> [ one   , zero ];

"v0078
"increment verify
 [ .c. , 1 , 0 , one   , zero , 0 , 0 ] -> [ zero  , one  ];
 [ .c. , 0 , 0 , one   , zero , 0 , 0 ] -> [ zero  , one  ];
 [ .c. , 0 , 0 , one   , zero , 0 , 1 ] -> [ zero  , one  ];
 [ .c. , 0 , 0 , one   , zero , 1 , 0 ] -> [ zero  , one  ];
 [ .c. , 0 , 0 , one   , zero , 1 , 1 ] -> [ zero  , one  ];
```

```
"hold
 [ .c. , 1 , 0 , one  , zero , 1 , 0 ] -> [ zero , one  ];
 [ .c. , 1 , 0 , one  , zero , 1 , 1 ] -> [ zero , one  ];

"v0085
"2 word transaction with starting address one
 [ .c. , 1 , 0 , one  , one  , 0 , 1 ] -> [ one  , one  ];
 [ .c. , 0 , 0 , one  , one  , 0 , 0 ] -> [ one  , one  ];
 [ .c. , 0 , 0 , one  , one  , 0 , 1 ] -> [ one  , one  ];
 [ .c. , 0 , 0 , one  , one  , 1 , 0 ] -> [ one  , one  ];
 [ .c. , 0 , 0 , one  , one  , 1 , 1 ] -> [ one  , one  ];
"hold
 [ .c. , 1 , 0 , one  , one  , 1 , 0 ] -> [ one  , one  ];
 [ .c. , 1 , 0 , one  , one  , 1 , 1 ] -> [ one  , one  ];

"v0092
"increment verify
 [ .c. , 1 , 0 , one  , one  , 0 , 0 ] -> [ zero , two  ];
 [ .c. , 0 , 0 , one  , one  , 0 , 0 ] -> [ zero , two  ];
 [ .c. , 0 , 0 , one  , one  , 0 , 1 ] -> [ zero , two  ];
 [ .c. , 0 , 0 , one  , one  , 1 , 0 ] -> [ zero , two  ];
 [ .c. , 0 , 0 , one  , one  , 1 , 1 ] -> [ zero , two  ];
"hold
 [ .c. , 1 , 0 , one  , one  , 1 , 0 ] -> [ zero , two  ];
 [ .c. , 1 , 0 , one  , one  , 1 , 1 ] -> [ zero , two  ];

"v0099
"2 word transaction with starting address two
 [ .c. , 1 , 0 , one  , two  , 0 , 1 ] -> [ one  , two  ];
 [ .c. , 0 , 0 , one  , two  , 0 , 0 ] -> [ one  , two  ];
 [ .c. , 0 , 0 , one  , two  , 0 , 1 ] -> [ one  , two  ];
 [ .c. , 0 , 0 , one  , two  , 1 , 0 ] -> [ one  , two  ];
 [ .c. , 0 , 0 , one  , two  , 1 , 1 ] -> [ one  , two  ];
"hold
 [ .c. , 1 , 0 , one  , two  , 1 , 0 ] -> [ one  , two  ];
 [ .c. , 1 , 0 , one  , two  , 1 , 1 ] -> [ one  , two  ];

"v0106
"increment verify
 [ .c. , 1 , 0 , one  , two  , 0 , 0 ] -> [ zero , three ];
 [ .c. , 0 , 0 , one  , two  , 0 , 0 ] -> [ zero , three ];
 [ .c. , 0 , 0 , one  , two  , 0 , 1 ] -> [ zero , three ];
 [ .c. , 0 , 0 , one  , two  , 1 , 0 ] -> [ zero , three ];
 [ .c. , 0 , 0 , one  , two  , 1 , 1 ] -> [ zero , three ];
"hold
 [ .c. , 1 , 0 , one  , two  , 1 , 0 ] -> [ zero , three ];
 [ .c. , 1 , 0 , one  , two  , 1 , 1 ] -> [ zero , three ];

"v0113
"1 word transaction with starting address zero
 [ .c. , 1 , 0 , zero , zero , 0 , 1 ] -> [ zero , zero ];
 [ .c. , 0 , 0 , zero , zero , 0 , 0 ] -> [ zero , zero ];
 [ .c. , 0 , 0 , zero , zero , 0 , 1 ] -> [ zero , zero ];
 [ .c. , 0 , 0 , zero , zero , 1 , 0 ] -> [ zero , zero ];
 [ .c. , 0 , 0 , zero , zero , 1 , 1 ] -> [ zero , zero ];
"hold
 [ .c. , 1 , 0 , zero , zero , 1 , 0 ] -> [ zero , zero ];
 [ .c. , 1 , 0 , zero , zero , 1 , 1 ] -> [ zero , zero ];

"v0120
"1 word transaction with starting address one
 [ .c. , 1 , 0 , zero , one  , 0 , 1 ] -> [ zero , one  ];
 [ .c. , 0 , 0 , zero , one  , 0 , 0 ] -> [ zero , one  ];
 [ .c. , 0 , 0 , zero , one  , 0 , 1 ] -> [ zero , one  ];
 [ .c. , 0 , 0 , zero , one  , 1 , 0 ] -> [ zero , one  ];
 [ .c. , 0 , 0 , zero , one  , 1 , 1 ] -> [ zero , one  ];
"hold
 [ .c. , 1 , 0 , zero , one  , 1 , 0 ] -> [ zero , one  ];
 [ .c. , 1 , 0 , zero , one  , 1 , 1 ] -> [ zero , one  ];
```

```
"v0127
"1 word transaction with starting address two
  [ .c. ,  1  ,  0 , zero   ,  two   ,  0  ,  1  ] -> [  zero   ,  two    ];
  [ .c. ,  0  ,  0 , zero   ,  two   ,  0  ,  0  ] -> [  zero   ,  two    ];
  [ .c. ,  0  ,  0 , zero   ,  two   ,  0  ,  1  ] -> [  zero   ,  two    ];
  [ .c. ,  0  ,  0 , zero   ,  two   ,  1  ,  0  ] -> [  zero   ,  two    ];
  [ .c. ,  0  ,  0 , zero   ,  two   ,  1  ,  1  ] -> [  zero   ,  two    ];
"hold
  [ .c. ,  1  ,  0 , zero   ,  two   ,  1  ,  0  ] -> [  zero   ,  two    ];
  [ .c. ,  1  ,  0 , zero   ,  two   ,  1  ,  1  ] -> [  zero   ,  two    ];


"v0134
"1 word transaction with starting address three
  [ .c. ,  1  ,  0 , zero   ,  three ,  0  ,  1  ] -> [  zero   ,  three  ];
  [ .c. ,  0  ,  0 , zero   ,  three ,  0  ,  0  ] -> [  zero   ,  three  ];
  [ .c. ,  0  ,  0 , zero   ,  three ,  0  ,  1  ] -> [  zero   ,  three  ];
  [ .c. ,  0  ,  0 , zero   ,  three ,  1  ,  0  ] -> [  zero   ,  three  ];
  [ .c. ,  0  ,  0 , zero   ,  three ,  1  ,  1  ] -> [  zero   ,  three  ];
"hold
  [ .c. ,  1  ,  0 , zero   ,  three ,  1  ,  0  ] -> [  zero   ,  three  ];
  [ .c. ,  1  ,  0 , zero   ,  three ,  1  ,  1  ] -> [  zero   ,  three  ];


End;
```

Appendix C - Wait State Generator and Peripheral Control Logic, PAL Equation and Test Vector Listing

```
MODULE HAPWAIT

TITLE 'Wait state generator and peripheral signal generator Pal
       for the 80960MC-20,
       used on the Hercules program, HAP box
       Robert Higgins Mar 17, 1991'


"Device type

    Uwa11 DEVICE 'p20r4';


"Pin definitions

"Inputs
    clk2        PIN    1;
    _prom_ce    PIN    2;
    _iae_ce     PIN    3;
    _clock_ce   PIN    4;
    _serial_ce  PIN    5;
    cnt0        PIN    6;
    cnt1        PIN    7;
    cnt2        PIN    8;
    cnt3        PIN    9;
    _ads        PIN    10;
    _den        PIN    11;
    _ffoe       PIN    13;
    _ready      PIN    14;
    W__R        PIN    23;
    clk         PIN    21;

"Outputs
    _rready     PIN    15;
    _wready     PIN    16;
    _we         PIN    17;
    dden        PIN    18;
    _oe         PIN    19;
    _iae_ds     PIN    20;
    _we_out     PIN    22;


"Define constants
    count       = [cnt3,cnt2,cnt1,cnt0];
    wait_type   = [_iae_ce,_clock_ce,_serial_ce,_prom_ce];
    ram         = [ 1 , 1 , 1 , 1 ];
    iae         = [ 0 , 1 , 1 , 1 ];
    clock       = [ 1 , 0 , 1 , 1 ];
    serial      = [ 1 , 1 , 0 , 1 ];
    prom        = [ 1 , 1 , 1 , 0 ];
    one         = [ 1 ];
    zero        = [ 0 ];
```

```
Equations

"write enable cutoff output
     !_we_cut = !w__r #

"give ram/interrupt enable register 1 wait state
     cnt0 & _iae_ce & _clock_ce & _serial_ce & _prom_ce #

"give prom 3 wait states
     cnt1 & cnt0 & !_prom_ce #

"give iae 12 wait states
     cnt3 & cnt2 & !_iae_ce #

"give clock 4 wait states
     cnt2 & !_clock_ce #

"give serial 2 wait states
     cnt1 & !_serial_ce;


"ready output for a read

     !_rready = w__r #

"give ram/interrupt enable register 1 wait state
     cnt0 & _iae_ce & _clock_ce & _serial_ce & _prom_ce #

"give prom 3 wait states
     cnt1 & cnt0 & !_prom_ce #

"give iae 12 wait states
     cnt3 & cnt2 & !_iae_ce #

"give clock 4 wait states
     cnt2 & !_clock_ce #

"give serial 2 wait states
     cnt1 & !_serial_ce;


"ready output for a write, add one wait state to desired number for we_cut

     !_wready = !w__r #

"give ram/interrupt enable register 2 wait states
     cnt1 & _iae_ce & _clock_ce & _serial_ce & _prom_ce #

"give prom 4 wait states
     cnt2 & !_prom_ce #

"give iae 13 wait states
     cnt3 & cnt2 & cnt0 & !_iae_ce #

"give clock 5 wait states
     cnt2 & cnt0 & !_clock_ce #

"give serial 3 wait states
     cnt1 & cnt0 & !_serial_ce;


"Define enables
    enable  clk     = 0;
    enable _we_cut = 1;
    enable _rready = 1;
    enable _wready = 1;


"dden output
State_diagram [dden]

    State zero:    CASE                                  !clk  : zero;
                                clk &  _ads &  _den &  _ready : zero;
                                clk & !_ads &  _den &  _ready : zero;
                                clk &  _ads & !_den &  _ready : one;
                                clk &  _ads & !_den & !_ready : zero;
                                clk & !_ads & !_den &  _ready : zero;
```

```
"unused cases
                        clk & !_ads & !_den & !_ready : zero;
                        clk & !_ads &  _den & !_ready : zero;
                        clk &  _ads &  _den & !_ready : zero;
                ENDCASE;

    State one:  CASE                                    !clk  : one;
                        clk &  _ads &  _den &  _ready : zero;
                        clk & !_ads &  _den &  _ready : zero;
                        clk &  _ads & !_den &  _ready : one;
                        clk &  _ads & !_den & !_ready : zero;
                        clk & !_ads & !_den &  _ready : zero;
"unused cases
                        clk & !_ads & !_den & !_ready : zero;
                        clk & !_ads &  _den & !_ready : zero;
                        clk &  _ads &  _den & !_ready : zero;
                ENDCASE;


"_oe output
State_diagram [_oe]

    State zero: CASE                                    !clk  : zero;
                                                        w__r  : one;
                       clk & !w__r &  _ads &  _den &  _ready : one;
                       clk & !w__r & !_ads &  _den &  _ready : one;
                       clk & !w__r &  _ads & !_den &  _ready : zero;
                       clk & !w__r &  _ads & !_den & !_ready : one;
                       clk & !w__r & !_ads & !_den &  _ready : one;
"unused cases
                       clk & !w__r & !_ads & !_den & !_ready : one;
                       clk & !w__r & !_ads &  _den & !_ready : one;
                       clk & !w__r &  _ads &  _den & !_ready : one;
                ENDCASE;

    State one:  CASE                                    !clk  : one;
                                                        w__r  : one;
                       clk & !w__r &  _ads &  _den &  _ready : one;
                       clk & !w__r & !_ads &  _den &  _ready : one;
                       clk & !w__r &  _ads & !_den &  _ready : zero;
                       clk & !w__r &  _ads & !_den & !_ready : one;
                       clk & !w__r & !_ads & !_den &  _ready : one;
"unused cases
                       clk & !w__r & !_ads & !_den & !_ready : one;
                       clk & !w__r & !_ads &  _den & !_ready : one;
                       clk & !w__r &  _ads &  _den & !_ready : one;
                ENDCASE;


"_we output
State_diagram [_we]

   State zero:  CASE                                        !clk  : zero;
                                                          !w__r  : one;
                                                        !_we_cut  : one;
                  clk & _we_cut &  w__r &  _ads &  _den &  _ready : one;
                  clk & _we_cut &  w__r & !_ads &  _den &  _ready : one;
                  clk & _we_cut &  w__r &  _ads & !_den &  _ready : zero;
                  clk & _we_cut &  w__r &  _ads & !_den & !_ready : one;
                  clk & _we_cut &  w__r & !_ads & !_den &  _ready : one;
"unused cases
                  clk & _we_cut &  w__r & !_ads & !_den & !_ready : one;
                  clk & _we_cut &  w__r & !_ads &  _den & !_ready : one;
                  clk & _we_cut &  w__r &  _ads &  _den & !_ready : one;
                ENDCASE;

   State one:   CASE                                        !clk  : one;
                                                          !w__r  : one;
                                                        !_we_cut  : one;
                  clk & _we_cut &  w__r &  _ads &  _den &  _ready : one;
                  clk & _we_cut &  w__r & !_ads &  _den &  _ready : one;
                  clk & _we_cut &  w__r &  _ads & !_den &  _ready : zero;
                  clk & _we_cut &  w__r &  _ads & !_den & !_ready : one;
                  clk & _we_cut &  w__r & !_ads & !_den &  _ready : one;
"unused cases
                  clk & _we_cut &  w__r & !_ads & !_den & !_ready : one;
                  clk & _we_cut &  w__r & !_ads &  _den & !_ready : one;
                  clk & _we_cut &  w__r &  _ads &  _den & !_ready : one;
                ENDCASE;
```

```
"_iae_ds output
State_diagram [_iae_ds]

"more needs to be added for early cut off

    State zero:    CASE                                                !clk  : zero;
                                                                     _iae_ce  : one;
                    clk & !_iae_ce & !w__r &  _ads & !dden &  _ready : one;
                    clk & !_iae_ce & !w__r & !_ads & !dden &  _ready : one;
                    clk & !_iae_ce & !w__r &  _ads &  dden &  _ready : zero;
                    clk & !_iae_ce & !w__r &  _ads &  dden & !_ready : one;
                    clk & !_iae_ce & !w__r & !_ads &  dden &  _ready : one;
          clk & !_iae_ce & w__r &  _we_cut &  _ads & !dden &  _ready : one;
          clk & !_iae_ce & w__r &  _we_cut & !_ads & !dden &  _ready : one;
          clk & !_iae_ce & w__r &  _we_cut &  _ads &  dden &  _ready : zero;
          clk & !_iae_ce & w__r &  _we_cut &  _ads &  dden & !_ready : one;
          clk & !_iae_ce & w__r &  _we_cut & !_ads &  dden &  _ready : one;
          clk & !_iae_ce & w__r & !_we_cut                           : one;
"unused cases
                          clk & !_iae_ce & !_ads &  dden & !_ready : one;
                          clk & !_iae_ce & !_ads & !dden & !_ready : one;
                          clk & !_iae_ce &  _ads & !dden & !_ready : one;
                    ENDCASE;

    State one :    CASE                                                !clk  : one;
                                                                     _iae_ce  : one;
                    clk & !_iae_ce & !w__r &  _ads & !dden &  _ready : one;
                    clk & !_iae_ce & !w__r & !_ads & !dden &  _ready : one;
                    clk & !_iae_ce & !w__r &  _ads &  dden &  _ready : zero;
                    clk & !_iae_ce & !w__r &  _ads &  dden & !_ready : one;
                    clk & !_iae_ce & !w__r & !_ads &  dden &  _ready : one;
          clk & !_iae_ce & w__r &  _we_cut &  _ads & !dden &  _ready : one;
          clk & !_iae_ce & w__r &  _we_cut & !_ads & !dden &  _ready : one;
          clk & !_iae_ce & w__r &  _we_cut &  _ads &  dden &  _ready : zero;
          clk & !_iae_ce & w__r &  _we_cut &  _ads &  dden & !_ready : one;
          clk & !_iae_ce & w__r &  _we_cut & !_ads &  dden &  _ready : one;
          clk & !_iae_ce & w__r & !_we_cut                           : one;
"unused cases
                          clk & !_iae_ce & !_ads &  dden & !_ready : one;
                          clk & !_iae_ce & !_ads & !dden & !_ready : one;
                          clk & !_iae_ce &  _ads & !dden & !_ready : one;
                    ENDCASE;



"Wait state generator
Test_vectors ([w__r,count,wait_type] -> [_rready,_wready,_we_cut])

"V0001
 "read from RAM or IER, one wait state
              [ 0 ,  0  ,  ram    ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  1  ,  ram    ] ->  [ 0 , 0 , 0 ] ;
"V0003
 "read from PROM, two wait states
              [ 0 ,  0  ,  prom   ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  1  ,  prom   ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  2  ,  prom   ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  3  ,  prom   ] ->  [ 0 , 0 , 0 ] ;
"V0007
 "read from serial, two wait states
              [ 0 ,  0  ,  serial ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  1  ,  serial ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  2  ,  serial ] ->  [ 0 , 0 , 0 ] ;
```

```
"V0010
 "read from iae, twelve wait states
              [ 0 ,  0  ,  iae    ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  1  ,  iae    ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  2  ,  iae    ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  3  ,  iae    ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  4  ,  iae    ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  5  ,  iae    ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  6  ,  iae    ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  7  ,  iae    ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  8  ,  iae    ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  9  ,  iae    ] ->  [ 1 , 0 , 0 ] ;
              [ 0 , 10  ,  iae    ] ->  [ 1 , 0 , 0 ] ;
              [ 0 , 11  ,  iae    ] ->  [ 1 , 0 , 0 ] ;
              [ 0 , 12  ,  iae    ] ->  [ 0 , 0 , 0 ] ;
"V0023
 "read from clock, five wait states
              [ 0 ,  0  ,  clock  ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  1  ,  clock  ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  2  ,  clock  ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  3  ,  clock  ] ->  [ 1 , 0 , 0 ] ;
              [ 0 ,  4  ,  clock  ] ->  [ 0 , 0 , 0 ] ;


"V0028
 "write to RAM or IER, three wait states
              [ 1 ,  0  ,  ram    ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  1  ,  ram    ] ->  [ 0 , 1 , 0 ] ;
              [ 1 ,  2  ,  ram    ] ->  [ 0 , 0 , 1 ] ;
"V0031
 "write to PROM, illegal
              [ 1 ,  0  ,  prom   ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  1  ,  prom   ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  2  ,  prom   ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  3  ,  prom   ] ->  [ 0 , 1 , 0 ] ;
              [ 1 ,  4  ,  prom   ] ->  [ 0 , 0 , 1 ] ;
"V0036
 "write to serial, four wait states
              [ 1 ,  0  ,  serial ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  1  ,  serial ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  2  ,  serial ] ->  [ 0 , 1 , 0 ] ;
              [ 1 ,  3  ,  serial ] ->  [ 0 , 0 , 0 ] ;
"V0038
 "write to iae, thirteen wait states
              [ 1 ,  0  ,  iae    ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  1  ,  iae    ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  2  ,  iae    ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  3  ,  iae    ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  4  ,  iae    ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  5  ,  iae    ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  6  ,  iae    ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  7  ,  iae    ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  8  ,  iae    ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  9  ,  iae    ] ->  [ 0 , 1 , 1 ] ;
              [ 1 , 10  ,  iae    ] ->  [ 0 , 1 , 1 ] ;
              [ 1 , 11  ,  iae    ] ->  [ 0 , 1 , 1 ] ;
              [ 1 , 12  ,  iae    ] ->  [ 0 , 1 , 0 ] ;
              [ 1 , 13  ,  iae    ] ->  [ 0 , 0 , 0 ] ;
"V0052
 "write to clock, six wait states
              [ 1 ,  0  ,  clock  ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  1  ,  clock  ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  2  ,  clock  ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  3  ,  clock  ] ->  [ 0 , 1 , 1 ] ;
              [ 1 ,  4  ,  clock  ] ->  [ 0 , 1 , 0 ] ;
              [ 1 ,  5  ,  clock  ] ->  [ 0 , 0 , 0 ] ;

"V0058
"dden output
Test_vectors ([clk2,clk,_ads,_den,_ready] -> dden)

"burst transaction
"Ti
              [ .c., 1 , 1 ,  1  ,   1  ] ->   0  ;
              [ .c., 0 , 1 ,  1  ,   1  ] ->   0  ;
              [ .c., 1 , 1 ,  1  ,   1  ] ->   0  ;
"Ta
              [ .c., 0 , 0 ,  1  ,   1  ] ->   0  ;
              [ .c., 1 , 0 ,  1  ,   1  ] ->   0  ;
```

```
"Td
            [ .c., 0 , 1 ,  0  ,   1  ] ->   0  ;
            [ .c., 1 , 1 ,  0  ,   1  ] ->   1  ;
"Tw0
            [ .c., 0 , 1 ,  0  ,   1  ] ->   1  ;
            [ .c., 1 , 1 ,  0  ,   1  ] ->   1  ;
"Tw1
            [ .c., 0 , 1 ,  0  ,   1  ] ->   1  ;
            [ .c., 1 , 1 ,  0  ,   1  ] ->   1  ;
"ready
            [ .c., 0 , 1 ,  0  ,   0  ] ->   1  ;
            [ .c., 1 , 1 ,  0  ,   0  ] ->   0  ;
"V0071
"burst Td
            [ .c., 0 , 0 ,  0  ,   0  ] ->   0  ;
            [ .c., 0 , 0 ,  0  ,   1  ] ->   0  ;
            [ .c., 1 , 0 ,  0  ,   1  ] ->   0  ;
"Tw1
            [ .c., 0 , 1 ,  0  ,   1  ] ->   0  ;
            [ .c., 1 , 1 ,  0  ,   1  ] ->   1  ;
"Tw2
            [ .c., 0 , 1 ,  0  ,   1  ] ->   1  ;
            [ .c., 1 , 1 ,  0  ,   1  ] ->   1  ;
"Tw3
            [ .c., 0 , 1 ,  0  ,   1  ] ->   1  ;
            [ .c., 1 , 1 ,  0  ,   1  ] ->   1  ;
"ready
            [ .c., 0 , 1 ,  0  ,   0  ] ->   1  ;
            [ .c., 1 , 1 ,  0  ,   0  ] ->   0  ;


"_iae_ds output
Test vectors ([clk2,clk,_ads,_den,_ready,w__r,count,wait_type] -> _iae_ds)
"V0082
"Read IAE transaction
"Ti
            [ .c., 1 , 1 ,  1  ,   1  , 1  ,  0   ,  ram  ] ->   1   ;
            [ .c., 0 , 1 ,  1  ,   1  , 1  ,  0   ,  ram  ] ->   1   ;
            [ .c., 1 , 1 ,  1  ,   1  , 1  ,  0   ,  ram  ] ->   1   ;
"Ta
            [ .c., 0 , 0 ,  1  ,   1  , 0  ,  0   ,  ram  ] ->   1   ;
            [ .c., 1 , 0 ,  1  ,   1  , 0  ,  0   ,  ram  ] ->   1   ;
            [ .c., 0 , 0 ,  1  ,   1  , 0  ,  0   ,  iae  ] ->   1   ;
            [ .c., 1 , 0 ,  1  ,   1  , 0  ,  0   ,  iae  ] ->   1   ;
"V0089
"Td
            [ .c., 0 , 1 ,  0  ,   1  , 0  ,  0   ,  iae  ] ->   1   ;
            [ .c., 1 , 1 ,  0  ,   1  , 0  ,  0   ,  iae  ] ->   1   ;
"Tw0
            [ .c., 0 , 1 ,  0  ,   1  , 0  ,  0   ,  iae  ] ->   1   ;
            [ .c., 1 , 1 ,  0  ,   1  , 0  ,  0   ,  iae  ] ->   0   ;
"Tw1
            [ .c., 0 , 1 ,  0  ,   1  , 0  ,  1   ,  iae  ] ->   0   ;
            [ .c., 1 , 1 ,  0  ,   1  , 0  ,  1   ,  iae  ] ->   0   ;
"Tw12
            [ .c., 0 , 1 ,  0  ,   1  , 0  ,  12  ,  iae  ] ->   0   ;
            [ .c., 1 , 1 ,  0  ,   1  , 0  ,  12  ,  iae  ] ->   0   ;
"V0097
"ready
            [ .c., 0 , 1 ,  0  ,   0  , 0  ,  13  ,  iae  ] ->   0   ;
            [ .c., 1 , 1 ,  0  ,   0  , 0  ,  13  ,  iae  ] ->   1   ;

"V0099
"Write IAE transaction
"Ti
            [ .c., 1 , 1 ,  1  ,   1  , 1  ,  0   ,  ram  ] ->   1   ;
            [ .c., 0 , 1 ,  1  ,   1  , 1  ,  0   ,  ram  ] ->   1   ;
            [ .c., 1 , 1 ,  1  ,   1  , 1  ,  0   ,  ram  ] ->   1   ;
"Ta
            [ .c., 0 , 0 ,  1  ,   1  , 1  ,  0   ,  ram  ] ->   1   ;
            [ .c., 1 , 0 ,  1  ,   1  , 1  ,  0   ,  ram  ] ->   1   ;
            [ .c., 0 , 0 ,  1  ,   1  , 1  ,  0   ,  iae  ] ->   1   ;
            [ .c., 1 , 0 ,  1  ,   1  , 1  ,  0   ,  iae  ] ->   1   ;
"V0106
"Td
            [ .c., 0 , 1 ,  0  ,   1  , 1  ,  0   ,  iae  ] ->   1   ;
            [ .c., 1 , 1 ,  0  ,   1  , 1  ,  0   ,  iae  ] ->   1   ;
"Tw0
            [ .c., 0 , 1 ,  0  ,   1  , 1  ,  0   ,  iae  ] ->   1   ;
            [ .c., 1 , 1 ,  0  ,   1  , 1  ,  0   ,  iae  ] ->   0   ;
```

```
"Tw1
              [ .c., 0 , 1 ,  0  ,   1  , 1  ,  1    ,  iae  ] ->   0   ;
              [ .c., 1 , 1 ,  0  ,   1  , 1  ,  1    ,  iae  ] ->   0   ;
"Tw14
              [ .c., 0 , 1 ,  0  ,   1  , 1  ,  14   ,  iae  ] ->   0   ;
              [ .c., 1 , 1 ,  0  ,   1  , 1  ,  14   ,  iae  ] ->   1   ;
"V0114
"ready
              [ .c., 0 , 1 ,  0  ,   0  , 1  ,  15   ,  iae  ] ->   1   ;
              [ .c., 1 , 1 ,  0  ,   0  , 1  ,  15   ,  iae  ] ->   1   ;


"V0116
"_oe output
Test_vectors ([clk2,clk,w__r,_ads,_den,_ready] -> _oe)
"Ti write transaction
              [ .c., 1 , 1  ,  1 , 1  ,  1   ] -> 1  ;
              [ .c., 0 , 1  ,  1 , 1  ,  1   ] -> 1  ;
              [ .c., 1 , 1  ,  1 , 1  ,  1   ] -> 1  ;
"Ta
              [ .c., 0 , 1  ,  0 , 1  ,  1   ] -> 1  ;
              [ .c., 1 , 1  ,  0 , 1  ,  1   ] -> 1  ;
"Td
              [ .c., 0 , 1  ,  1 , 0  ,  1   ] -> 1  ;
              [ .c., 1 , 1  ,  1 , 0  ,  1   ] -> 1  ;
"V0123
"ready
              [ .c., 0 , 1  ,  1 , 0  ,  0   ] -> 1  ;
              [ .c., 1 , 1  ,  1 , 0  ,  0   ] -> 1  ;
"burst
              [ .c., 0 , 1  ,  0 , 0  ,  1   ] -> 1  ;
              [ .c., 1 , 1  ,  0 , 0  ,  1   ] -> 1  ;
"V0127
"Ti read transaction
              [ .c., 0 , 0  ,  1 , 1  ,  1   ] -> 1  ;
              [ .c., 1 , 0  ,  1 , 1  ,  1   ] -> 1  ;
"Ta
              [ .c., 0 , 0  ,  0 , 1  ,  1   ] -> 1  ;
              [ .c., 1 , 0  ,  0 , 1  ,  1   ] -> 1  ;
"Td
              [ .c., 0 , 0  ,  1 , 0  ,  1   ] -> 1  ;
              [ .c., 1 , 0  ,  1 , 0  ,  1   ] -> 0  ;
"Tw0
              [ .c., 0 , 0  ,  1 , 0  ,  1   ] -> 0  ;
              [ .c., 1 , 0  ,  1 , 0  ,  1   ] -> 0  ;
"V0135
"ready
              [ .c., 0 , 0  ,  1 , 0  ,  0   ] -> 0  ;
              [ .c., 1 , 0  ,  1 , 0  ,  0   ] -> 1  ;
"burst
              [ .c., 0 , 0  ,  0 , 0  ,  1   ] -> 1  ;
              [ .c., 1 , 0  ,  0 , 0  ,  1   ] -> 1  ;


"V0139
Test_vectors
        ([clk2,clk,w__r,_ads,_den,_ready,count,wait_type] ->
                                                    [_we_cut,_we])
"Ti read transaction
          [ .c., 1 , 0  ,  1 ,  1 ,   1  ,  0  ,  ram    ] ->
                                                    [ 0     , 1 ] ;
          [ .c., 0 , 0  ,  1 ,  1 ,   1  ,  0  ,  ram    ] ->
                                                    [ 0     , 1 ] ;
          [ .c., 1 , 0  ,  1 ,  1 ,   1  ,  0  ,  ram    ] ->
                                                    [ 0     , 1 ] ;
"Ta
          [ .c., 0 , 0  ,  0 ,  1 ,   1  ,  0  ,  ram    ] ->
                                                    [ 0     , 1 ] ;
          [ .c., 1 , 0  ,  0 ,  1 ,   1  ,  0  ,  ram    ] ->
                                                    [ 0     , 1 ] ;
"V0144
"Td
          [ .c., 0 , 0  ,  1 ,  0 ,   1  ,  0  ,  ram    ] ->
                                                    [ 0     , 1 ] ;
          [ .c., 1 , 0  ,  1 ,  0 ,   1  ,  0  ,  ram    ] ->
                                                    [ 0     , 1 ] ;
```

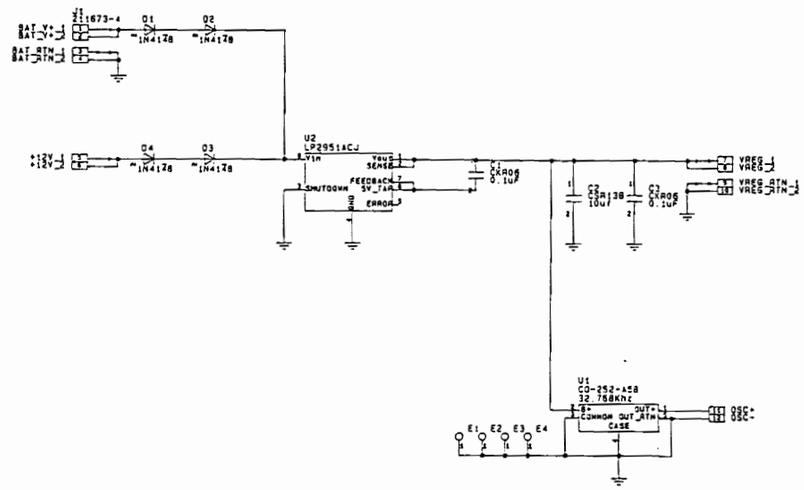
```
"Tw0
        [ .c., 0 , 0  , 1 , 0 ,  1  , 1  , ram   ] ->
                                                    [ 0    , 1 ] ;
        [ .c., 1 , 0  , 1 , 0 ,  1  , 1  , ram   ] ->
                                                    [ 0    , 1 ] ;
"Tw1
        [ .c., 0 , 0  , 1 , 0 ,  1  , 2  , ram   ] ->
                                                    [ 0    , 1 ] ;
        [ .c., 1 , 0  , 1 , 0 ,  1  , 2  , ram   ] ->
                                                    [ 0    , 1 ] ;
"V0150
"ready
        [ .c., 0 , 0  , 1 , 0 ,  0  , 0  , ram   ] ->
                                                    [ 0    , 1 ] ;
        [ .c., 1 , 0  , 1 , 0 ,  0  , 0  , ram   ] ->
                                                    [ 0    , 1 ] ;
"burst
        [ .c., 0 , 0  , 0 , 0 ,  1  , 0  , ram   ] ->
                                                    [ 0    , 1 ] ;
        [ .c., 1 , 0  , 0 , 0 ,  1  , 0  , ram   ] ->
                                                    [ 0    , 1 ] ;

"V0154
"Ti write transaction
        [ .c., 0 , 1  , 1 , 1 ,  1  , 0  , ram   ] ->
                                                    [ 1    , 1 ] ;
        [ .c., 1 , 1  , 1 , 1 ,  1  , 0  , ram   ] ->
                                                    [ 1    , 1 ] ;
"Ta
        [ .c., 0 , 1  , 0 , 1 ,  1  , 0  , ram   ] ->
                                                    [ 1    , 1 ] ;
        [ .c., 1 , 1  , 0 , 1 ,  1  , 0  , ram   ] ->
                                                    [ 1    , 1 ] ;
"Td
        [ .c., 0 , 1  , 1 , 0 ,  1  , 0  , ram   ] ->
                                                    [ 1    , 1 ] ;
        [ .c., 1 , 1  , 1 , 0 ,  1  , 0  , ram   ] ->
                                                    [ 1    , 0 ] ;
"V0160
"Tw0
        [ .c., 0 , 1  , 1 , 0 ,  1  , 0  , ram   ] ->
                                                    [ 1    , 0 ] ;
        [ .c., 1 , 1  , 1 , 0 ,  1  , 0  , ram   ] ->
                                                    [ 1    , 0 ] ;
"Tw1
        [ .c., 0 , 1  , 1 , 0 ,  1  , 1  , ram   ] ->
                                                    [ 0    , 0 ] ;
        [ .c., 1 , 1  , 1 , 0 ,  1  , 1  , ram   ] ->
                                                    [ 0    , 1 ] ;
"ready
        [ .c., 0 , 1  , 1 , 0 ,  0  , 2  , ram   ] ->
                                                    [ 1    , 1 ] ;
        [ .c., 1 , 1  , 1 , 0 ,  0  , 2  , ram   ] ->
                                                    [ 1    , 1 ] ;
"V0166
"burst
        [ .c., 0 , 1  , 0 , 0 ,  1  , 0  , ram   ] ->
                                                    [ 1    , 1 ] ;
        [ .c., 1 , 1  , 0 , 0 ,  1  , 0  , ram   ] ->
                                                    [ 1    , 1 ] ;

End;
```

Appendix D - HAP Drawings

The following drawings are contained in this appendix:

HERCULES-0002	HAP UTC Oscillator Board, 1 sheet
HERCULES-0004	HAP Processor/Memory/Peripherals, 8 sheets
HERCULES-0010	HAP UTC Battery Pack, 1 sheet
HERCULES-0011	HAP Power Supply, 1 sheet

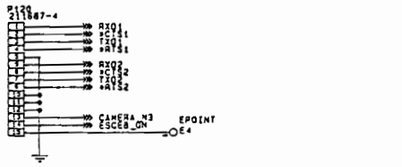
REVISIONS				
ZONE	LTR	DESCRIPTION	DATE	APPROVED
	A	AS BUILT	5/6/91	

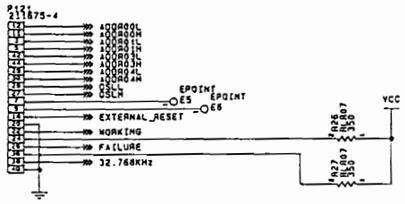


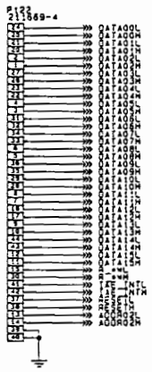
CONTRACT NO.		NAVAL RESEARCH LABORATORY NAVAL CENTER FOR SPACE TECHNOLOGY WASHINGTON D.C. 20375
DRAWN R. HIGGINS	DATE 5/6/91	TITLE HERCULES ATTITUDE PROCESSOR OSCILLATOR BOARD
CHECK R. COOKSEY		
DESIGN R. HIGGINS		
DESIGN ACTIVITY		SIZE E FSCM NO 81995 DWG NO. HERCULES-0002
CUSTOMER		SCALE NONE RELEASE DATE SHEET 1 OF 1

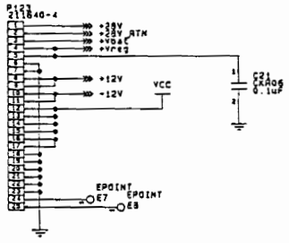
HERCULES-0002 A 1

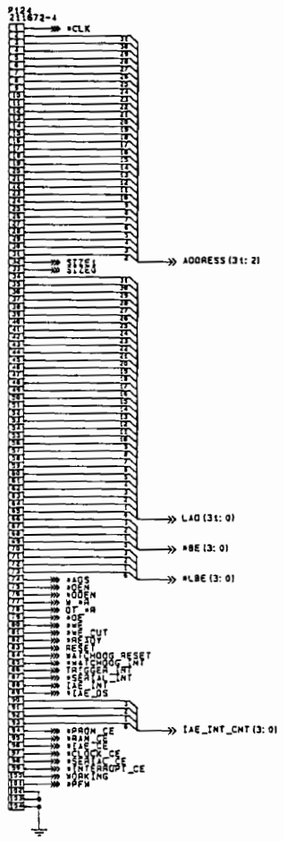
REVISIONS				
ZONE	LTR	DESCRIPTION	DATE	APPROVED
	A	SENT FOR PCB LAYOUT	8/12/91	
C0234	B	CHANGE CAPS TO 0.1uF	12/10/91	



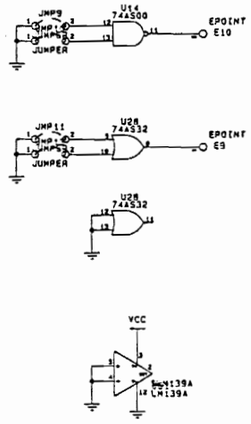




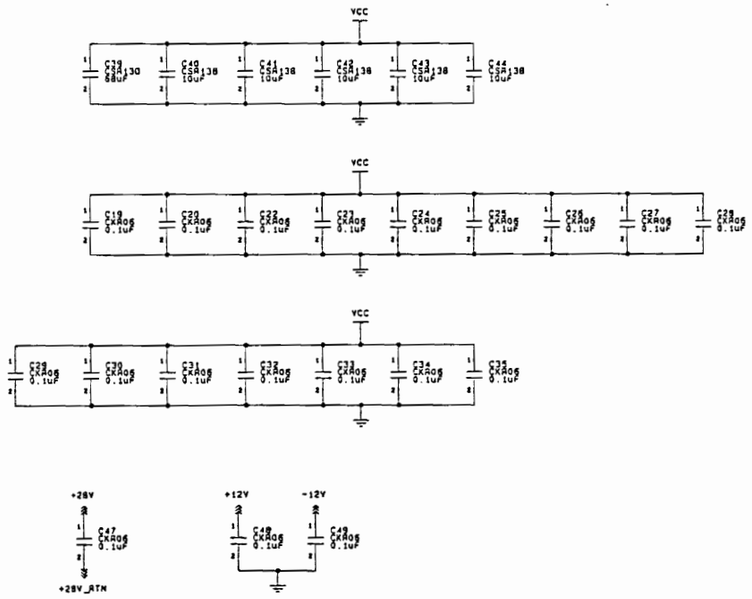




SPARE COMPONENTS

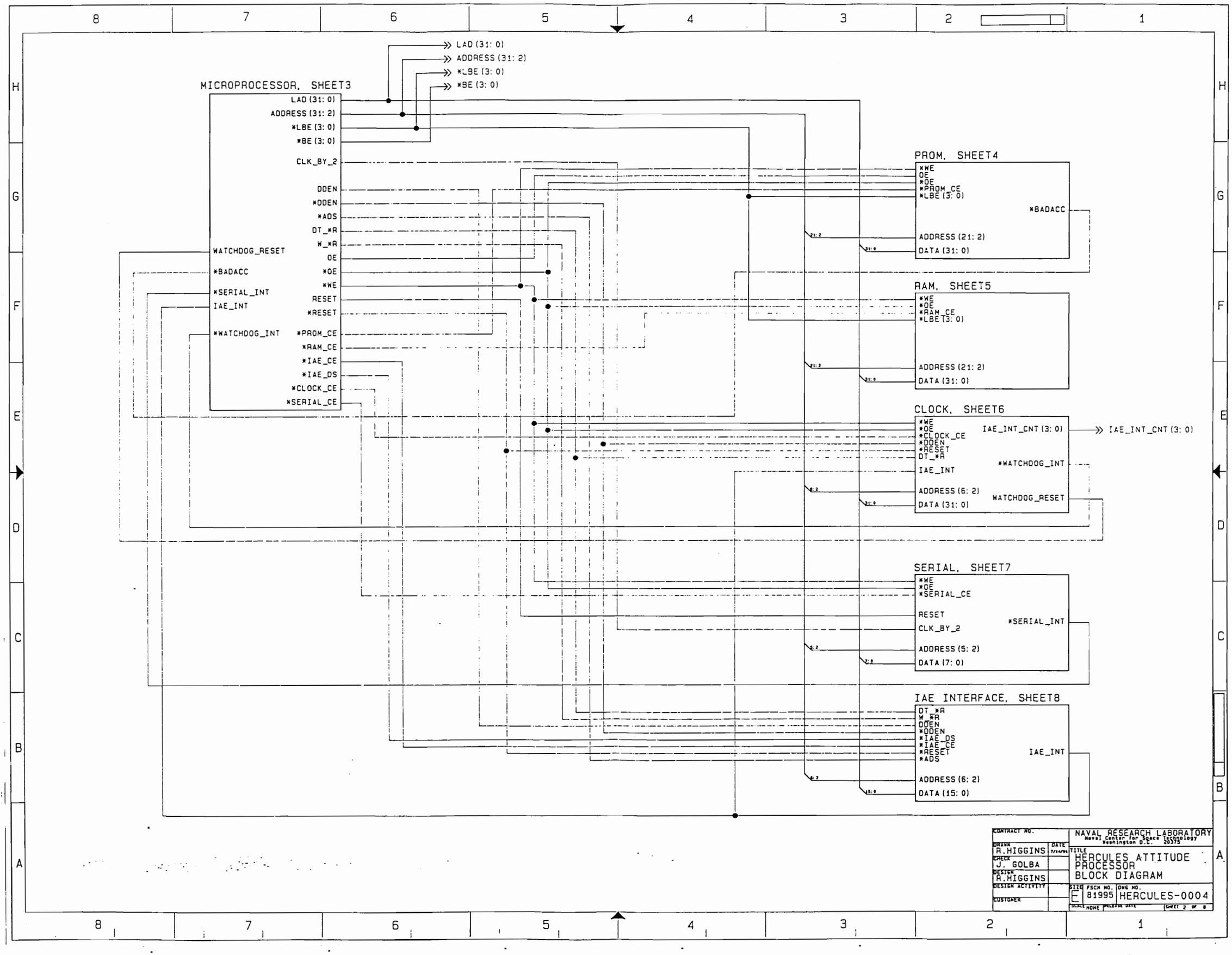


BYPASS CAPACITORS



CONTRACT NO.		NAVAL RESEARCH LABORATORY NAVAL CENTER FOR SPACE TECHNOLOGY WASHINGTON D.C. 20375	
DRAWN R. HIGGINS	DATE 12/10/91	TITLE HERCULES ATTITUDE PROCESSOR I/O AND MISC.	
CHECK J. BOLBA			
DESIGN R. HIGGINS			
DESIGN ACTIVITY		SIZE E	FSCM NO. 81995
CUSTOMER		SCALE NONE	RELEASE DATE
			SHEET 1 OF 9

HERCULES-0004

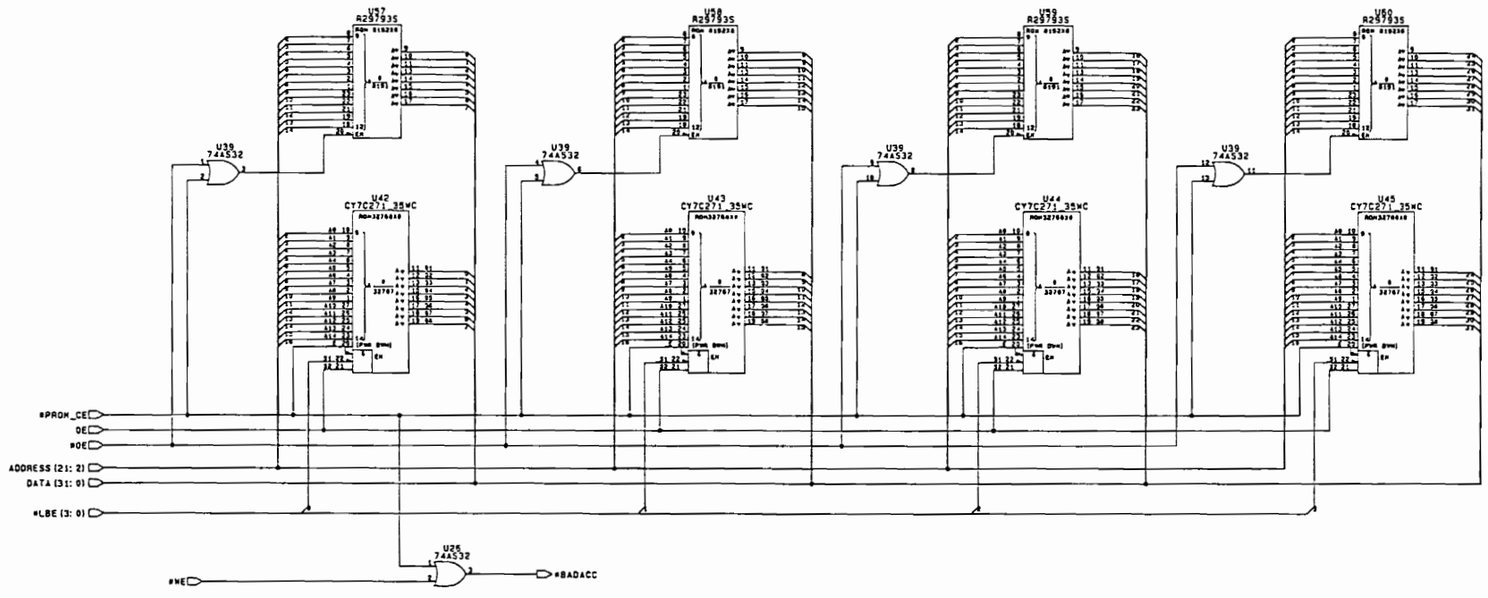

LAD (31: 0)
ADDRESS (31: 2)
*LBE (3: 0)
*BE (3: 0)
MICROPROCESSOR, SHEET3
LAD (31: 0)
ADDRESS (31: 2)
*LBE (3: 0)
*BE (3: 0)
CLK_BY_2
DDEN
*DDEN
*ADS
DT_*R
W_*R
OE
*OE
*WE
RESET
*RESET
*PROM_CE
*RAM_CE
*IAE_CE
*IAE_DS
*CLOCK_CE
*SERIAL_CE
WATCHDOG_RESET
*BADACC
*SERIAL_INT
IAE_INT
*WATCHDOG_INT
PROM, SHEET4
*WE
OE
*OE
*PROM_CE
*LBE (3: 0)
*BADACC
ADDRESS (21: 2)
DATA (31: 0)
RAM, SHEET5
*WE
*OE
*RAM_CE
*LBE (3: 0)
ADDRESS (21: 2)
DATA (31: 0)
CLOCK, SHEET6
*WE
*OE
*CLOCK_CE
*DDEN
*RESET
DT_*R
IAE_INT
ADDRESS (6: 2)
DATA (31: 0)
IAE_INT_CNT (3: 0)
*WATCHDOG_INT
WATCHDOG_RESET
IAE_INT_CNT (3: 0)
SERIAL, SHEET7
*WE
*OE
*SERIAL_CE
RESET
CLK_BY_2
ADDRESS (5: 2)
DATA (7: 0)
*SERIAL_INT
IAE INTERFACE, SHEET8
DT_*R
W_*R
DDEN
*DDEN
*IAE_DS
*IAE_CE
*RESET
*ADS
ADDRESS (6: 2)
DATA (15: 0)
IAE_INT
CONTRACT NO.
NAVAL RESEARCH LABORATORY
Naval Center for Space Technology
Washington D.C. 20375
DRAWN R.HIGGINS
CHECK J. GOLBA
DESIGN R.HIGGINS
DESIGN ACTIVITY
CUSTOMER
TITLE
HERCULES ATTITUDE PROCESSOR BLOCK DIAGRAM
SIZE E
FSCM NO. 81995
DWG NO. HERCULES-0004
SCALE NONE
SHEET 2 OF 8

REVISIONS				
ZONE	LTR	DESCRIPTION	DATE	APPROVED
	A	SENT TO PCB LAYOUT	7/15/91	
G-7	B	R23 IS 91K	12/10/91	
E-7	B	R17 AND R20 ARE UNUSED	12/10/91	

CONTRACT NO.		NAVAL RESEARCH LABORATORY NAVAL CENTER FOR SPACE TECHNOLOGY WASHINGTON D.C. 20375	
DRAWN R. HIGGINS	DATE 12/10/91	TITLE: HERCULES ATTITUDE PROCESSOR MICROPROCESSOR	
CHECK J. Golbe			
DESIGN R. HIGGINS			
DESIGN ACTIVITY		SIZE E	FSCM NO. 81995
CUSTOMER		SCALE NONE	RELEASE DATE
			DWG. NO. HERCULES-0004
			SHEET 3 OF 8

REVISIONS				
ZONE	LTR	DESCRIPTION	DATE	APPROVED
	A	USE CY7C271 PARTS	7/15/91	

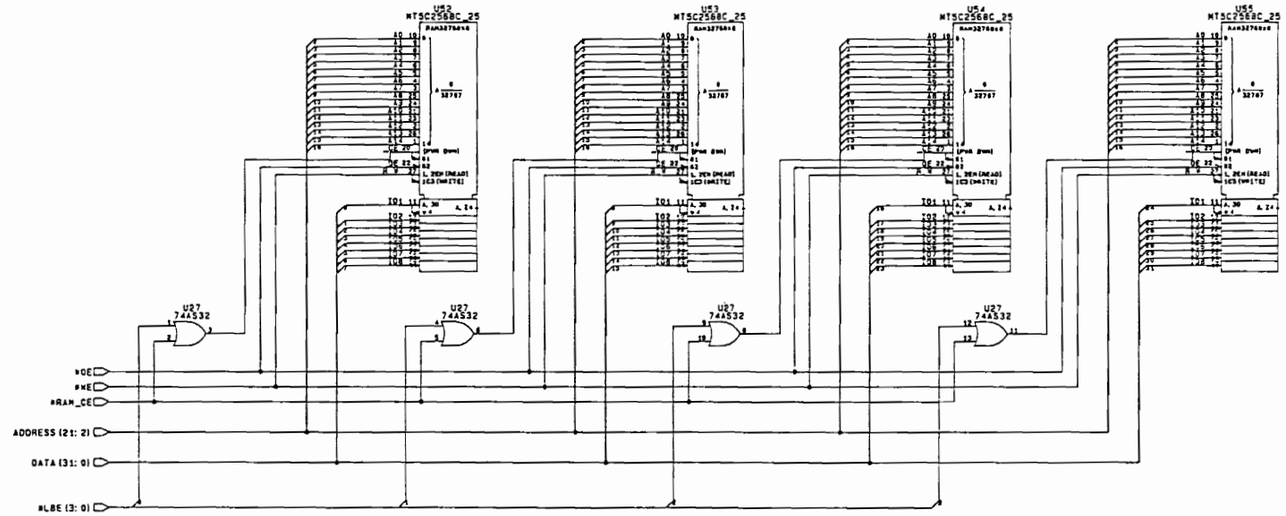
Either use R29793S or CY7C271-35WC



CONTRACT NO.		NAVAL RESEARCH LABORATORY NAVAL CENTER FOR SPACE TECHNOLOGY WASHINGTON D.C. 20375	
DRAWN R. HIGGINS	DATE 7/15/91	TITLE HERCULES ATTITUDE PROCESSOR PROM	
CHECK J. GOLBA			
DESIGN R. HIGGINS			
DESIGN ACTIVITY		SIZE E	FSCM NO. 81995
CUSTOMER		SCALE NONE	RELEASE DATE
			DWG. NO. HERCULES-0004
			SHEET 4 OF 8

DATA | HERCULES-0004

REVISIONS				
ZONE	LTR	DESCRIPTION	DATE	APPROVED
	A	AS SENT FOR PCB LAYOUT	7/14/91	

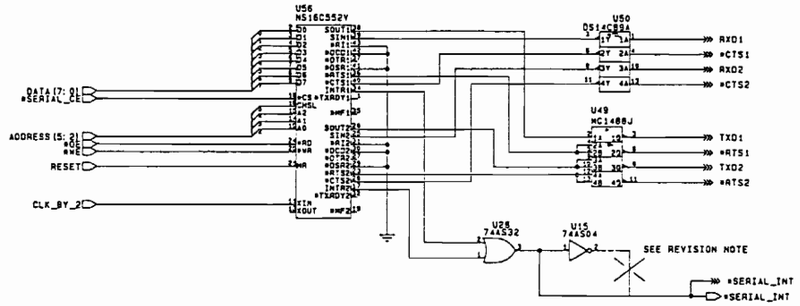


CONTRACT NO.		NAVAL RESEARCH LABORATORY NAVAL CENTER FOR SPACE TECHNOLOGY WASHINGTON D.C. 20375
DRAWN R. HIGGINS	DATE 7/14/91	TITLE HERCULES ATTITUDE PROCESSOR RAM
CHECK J. GOLBA		
DESIGN R. HIGGINS		
DESIGN ACTIVITY		SIZE E | FSCM NO. 81995 | DWG. NO. HERCULES-0004
CUSTOMER		SCALE NONE | RELEASE DATE | SHEET 5 OF 8

REVISIONS				
ZONE	LTR	DESCRIPTION	DATE	APPROVED
	A	AS SENT FOR PCB LAYOUT	7/30/91	
B-8	B	CHANGE R11 TO 91K	12/10/91	
B-7	B	R12 IS UNUSED	12/10/91	
A-7	B	CHANGE R13 AND R14 TO 110K	12/10/91	
G-5	C	ADD VCC TO R29 PIN 2	03/18/92	
G-5	C	CHANGE VALUE R29 FROM 10K TO 750	03/18/92	
G-5	C	ADD JUMPER FROM R6-3 TO R6-4	03/18/92	
F-5	C	ADD GROUND TO R6 PIN 8	03/18/92	
A-7	C	CHANGE VALUE R13 FROM 110K TO 160K	03/18/92	

CONTRACT NO.		NAVAL RESEARCH LABORATORY NAVAL CENTER FOR SPACE TECHNOLOGY WASHINGTON D.C. 20375	
DRAWN R. HIGGINS	DATE 12/10/90	TITLE HERCULES ATTITUDE PROCESSOR REAL TIME CLOCK	
CHECK J. GOLBA			
DESIGN R. HIGGINS			
DESIGN ACTIVITY		SIZE E	FSCM NO. 81995
CUSTOMER		SCALE NONE	RELEASE DATE
			SHEET 6 OF 8

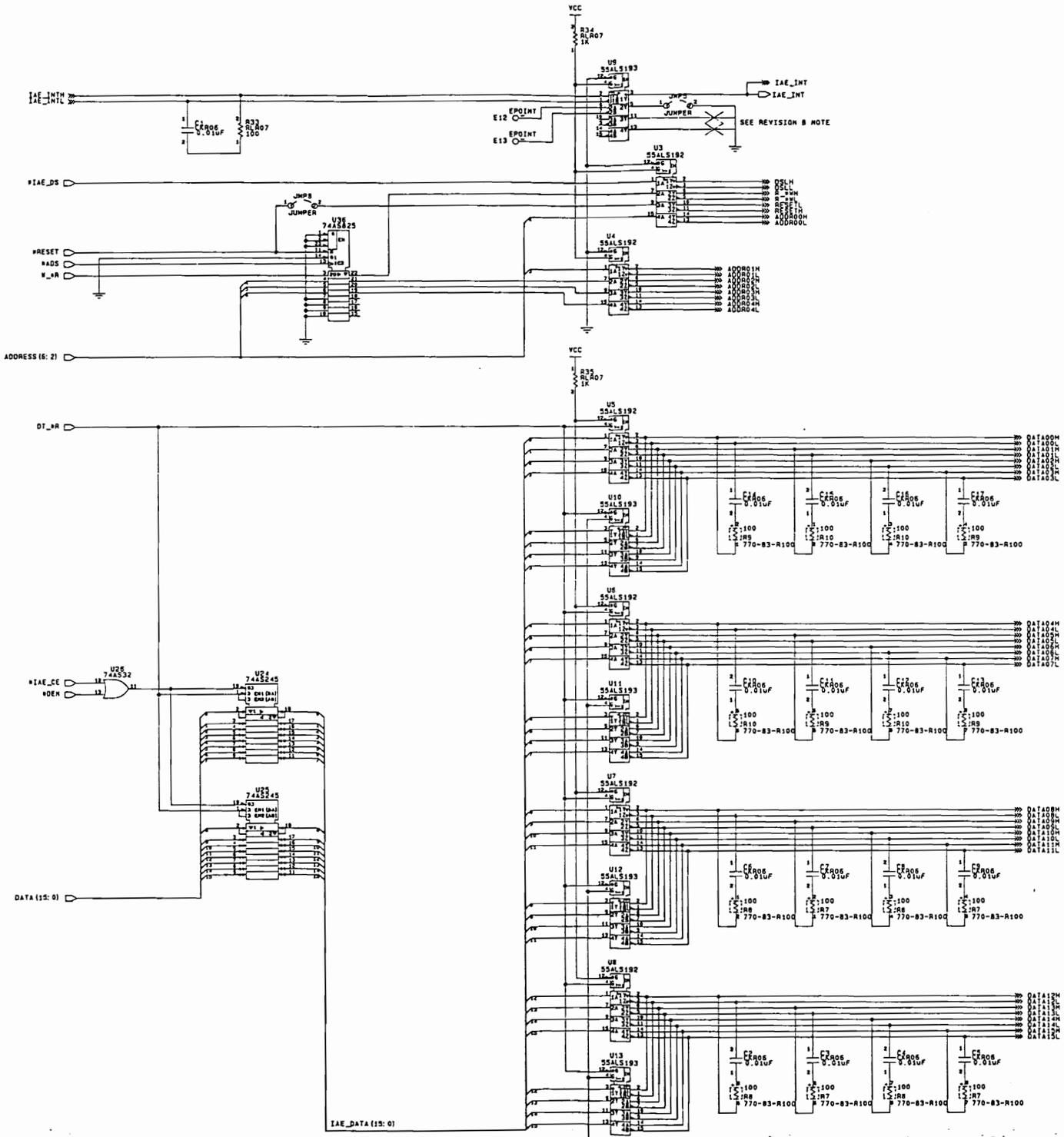
REVISIONS				
ZONE	LTR	DESCRIPTION	DATE	APPROVED
	A	AS SENT FOR PCB FABRICATION	7/30/91	
D-4	B	DESIGN CORRECTION: U15 PIN 2 IS	12/10/91	
D-4	B	REMOVED. PCB PADS FOR U15 PINS	12/10/91	
D-4	B	1 AND 2 ARE JUMPERED	12/10/91	
E-4	B	CHANGE U49 TO MC1488J	12/10/91	



CONTRACT NO.		NAVAL RESEARCH LABORATORY NAVAL CENTER FOR SPACE TECHNOLOGY WASHINGTON D.C. 20375	
DRAWN R. HIGGINS	DATE 12/10/91	TITLE HERCULES ATTITUDE PROCESSOR SERIAL	
CHECK J. GOLBA			
DESIGN R. HIGGINS			
DESIGN ACTIVITY		SIZE E	FSCM NO. 81995
CUSTOMER		SCALE NONE	RELEASE DATE
			DWG. NO. HERCULES-0004
			SHEET 7 OF 8

HERCULES-0004 REV B

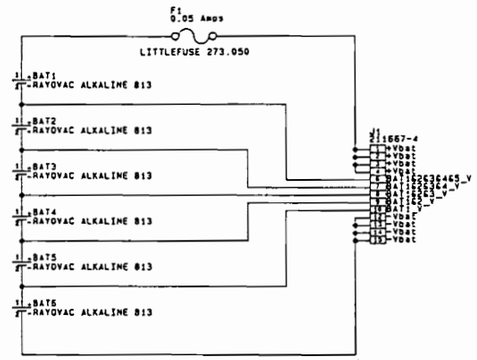
REVISIONS				
ZONE	LTR	DESCRIPTION	DATE	APPROVED
	A	AS SENT FOR PCB FABRICATION	7/31/91	
G-4	B	DESIGN ERROR: CUT PINS 11 AND 13	12/10/91	
G-4	B	OF U9. NEVER INSTALL JMP5	12/10/91	
G-7	C	CHANGE U36-14 ODEN TO GROUND	03/30/92	
D-7	C	CHANGE U26-13 *ODEN TO *DEN	03/30/92	



CONTRACT NO.		NAVAL RESEARCH LABORATORY NAVAL CENTER FOR SPACE TECHNOLOGY WASHINGTON D.C. 20375	
DRAWN R. HIGGINS	DATE 12/10/91	TITLE HERCULES ATTITUDE PROCESSOR IAE INTERFACE	
CHECK J. GOLBA			
DESIGN R. HIGGINS			
DESIGN ACTIVITY		SIZE E	FSCM NO. 81995
CUSTOMER		SCALE NONE	RELEASE DATE
			SHEET 8 OF 8

HERCULES-0004

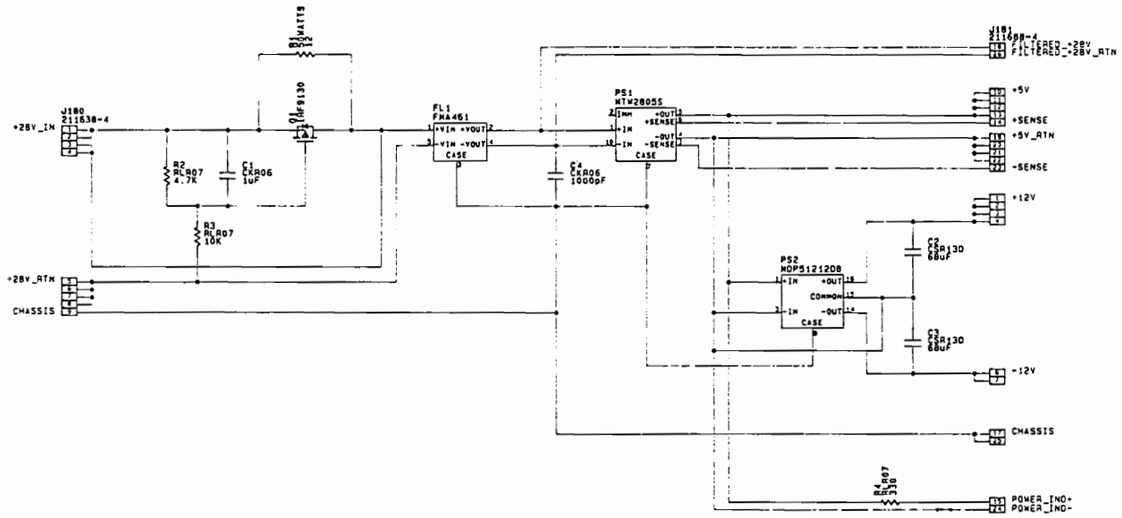
REVISIONS				
ZONE	LTR	DESCRIPTION	DATE	APPROVED
	A	AS BUILT	08/03/91	
E-6	B	CHANGE FUSE VALUE FROM 0.3 TO 0.05	03/30/92	



CONTRACT NO.		NAVAL RESEARCH LABORATORY NAVAL CENTER FOR SPACE TECHNOLOGY WASHINGTON D.C. 20375
DRAWN R. HIGGINS	DATE 08/8/91	TITLE HERCULES ATTITUDE PROCESSOR BATTERY PACK
CHECK C. EARNER		
DESIGN R. HIGGINS		
DESIGN ACTIVITY		SIZE E / FSCM NO. 81995 / DWG. NO. HERCULES-0010
CUSTOMER		SCALE NONE / RELEASE DATE / SHEET 1 OF 1

HERCULES-0010 A

REVISIONS				
ZONE	LTR	DESCRIPTION	DATE	APPROVED
	A	AS BUILT	12/3/91	
E-4	B	REMOVE INPUT TO OUTPUT GND CONNECT	08/10/92	



CONTRACT NO.		NAVAL RESEARCH LABORATORY NAVAL CENTER FOR SPACE TECHNOLOGY WASHINGTON D.C. 20375	
DRAWN R. HIGGINS	DATE 3/10/92	TITLE HERCULES ATTITUDE PROCESSOR POWER SUPPLY	
CHECK J. SOLBA			
DESIGN R. HIGGINS			
DESIGN ACTIVITY		SIZE FSCM NO. E 81995	DWG. NO. HERCULES-0011
CUSTOMER		SCALE NONE RELEASE DATE	SHEET 1 OF 1

Appendix E - HAP Complete Assembly Code Listing

```
0000                    #line 1 "-"
0000                    #line 1 "hap.src"
0000                    /*************************************************************************/
0000                    /*                                                                       */
0000                    /*      INIT.SRC : Initialization routine for the HAP processor using    */
0000                    /*                 the 80960MC.                                          */
0000                    /*      R. Higgins                                                       */
0000                    /*      DATE     : 30-july-1991                                          */
0000                    /*                 11/1/91 add data section alignment                    */
0000                    /*                 12/9/91 change synmov to synmovq                      */
0000                    /*                 1/2/92 change restart stuff                           */
0000                    /*                 1/7/92 move data structure and restart stuff to       */
0000                    /*                        movdata.src, reorder tests                     */
0000                    /*                 1/9/92 redo the segment table and pcb                 */
0000                    /*                 1/23/92 try new seg table, prcb, pcb, intr_stack      */
0000                    /*                 2/4/92   fault record retention                       */
0000                    /*************************************************************************/
0000
0000                    #line 1 "hap.src:sys.inc"
0000
0000                    /*************************************************************************/
0000                    /*       PROM globals defined...                                         */
0000                    /*************************************************************************/
0000
0000                      .set prom$st, 0x0
0000                      .set prom$end, 0x1FFFF
0000
0000                    /*************************************************************************/
0000                    /*       RAM globals defined...                                          */
0000                    /*************************************************************************/
0000
0000                      .set ram$st, 0x01000000
0000                      .set ram$end, 0x0101FFFF
0000
0000                    /*************************************************************************/
0000                    /*       IAE globals defined...                                          */
0000                    /*************************************************************************/
0000
0000                      .set _iae_a1, 0x02000040  /* roll gyro accumulator */
0000                      .set _iae_a2, 0x02000044  /* pitch gyro accumulator */
0000                      .set _iae_a3, 0x02000048  /* yaw gyro accumulator */
0000                      .set iae_odp, 0x0200001c  /* output discrete port */
0000                      .set iae_cr, 0x02000000  /* command register */
0000                      .set iae_ifp, 0x02000014  /* internal flag port */
0000                      .set iae_amar, 0x02000038  /* analog mux address reg */
0000                      .set iae_ad_result, 0x0200003c  /* a/d result reg */
0000
3d40                      .data
3d40                      .align 2
3d40                    valid_iae_mux_addr:
3d40  0000                .short          0x0000  /* gx temp */
3d42  0001                .short          0x0001  /* gy temp */
3d44  0002                .short          0x0002  /* gz temp */
3d46  0003                .short          0x0003  /* btemp */
3d48  0004                .short          0x0004  /* strtm */
3d4a  0005                .short          0x0005  /* 180vm */
3d4c  0008                .short          0x0008  /* gx plcm */
3d4e  0010                .short          0x0010  /* gy plcm */
```

```
3d50  0018                .short          0x0018  /* gz plcm */
3d52  0020                .short          0x0020  /* gz dithm */
3d54  0028                .short          0x0028  /* -5v */
3d56  0030                .short          0x0030  /* +5v */
3d58  0038                .short          0x0038  /* n750vm */
3d5a  0040                .short          0x0040  /* gx lim */
3d5c  0048                .short          0x0048  /* gy lim */
3d5e  0050                .short          0x0050  /* gz lim */
3d60  0058                .short          0x0058  /* gx ccm */
3d62  0060                .short          0x0060  /* gy ccm */
3d64  0068                .short          0x0068  /* gz ccm */
3d66  0070                .short          0x0070  /* gx dithm */
3d68  0078                .short          0x0078  /* gy dithm */
3d68
3d6a                        .align 2
3d6a                            /* values provided from Honeywell on 3/17/92 */
3d6c                      min_iae_mux_item_value:
3d6c  f040                .short          0xf040  /* min gx temp */
3d6e  f040                .short          0xf040  /* min gy temp */
3d70  f040                .short          0xf040  /* min gz temp */
3d72  f040                .short          0xf040  /* min btemp */
3d74  8000                .short          0x8000  /* min strtm */
3d76  0700                .short          0x0700  /* min 180vm */
3d78  ecc0                .short          0xecc0  /* min gx plcm */
3d7a  ecc0                .short          0xecc0  /* min gy plcm */
3d7c  ecc0                .short          0xecc0  /* min gz plcm */
3d7e  0900                .short          0x0900  /* min gz dithm */
3d80  b760                .short          0xb760  /* min -5v */
3d82  3cd0                .short          0x3cd0  /* min +5v */
3d84  cc00                .short          0xcc00  /* min n750vm */
3d86  2870                .short          0x2870  /* min gx lim */
3d88  2870                .short          0x2870  /* min gy lim */
3d8a  2870                .short          0x2870  /* min gz lim */
3d8c  1ae0                .short          0x1ae0  /* min gx ccm */
3d8e  1ae0                .short          0x1ae0  /* min gy ccm */
3d90  1ae0                .short          0x1ae0  /* min gz ccm */
3d92  0900                .short          0x0900  /* min gx dithm */
3d94  0900                .short          0x0900  /* min gy dithm */
3d94
3d96                        .align 2
3d96                            /* values provided from Honeywell on 3/17/92 */
3d98                      max_iae_mux_item_value:
3d98  4f40                .short          0x4f40  /* max gx temp */
3d9a  4f40                .short          0x4f40  /* max gy temp */
3d9c  4f40                .short          0x4f40  /* max gz temp */
3d9e  4f40                .short          0x4f40  /* max btemp */
3da0  8000                .short          0x8000  /* max strtm */
3da2  0d00                .short          0x0d00  /* max 180vm */
3da4  1330                .short          0x1330  /* max gx plcm */
3da6  1330                .short          0x1330  /* max gy plcm */
3da8  1330                .short          0x1330  /* max gz plcm */
3daa  1400                .short          0x1400  /* max gz dithm */
3dac  c430                .short          0xc430  /* max -5v */
3dae  4600                .short          0x4600  /* max +5v */
3db0  f3d0                .short          0xf3d0  /* max n750vm */
3db2  7ff0                .short          0x7ff0  /* max gx lim */
3db4  7ff0                .short          0x7ff0  /* max gy lim */
3db6  7ff0                .short          0x7ff0  /* max gz lim */
3db8  2670                .short          0x2670  /* max gx ccm */
3dba  2670                .short          0x2670  /* max gy ccm */
3dbc  2670                .short          0x2670  /* max gz ccm */
3dbe  1400                .short          0x1400  /* max gx dithm */
3dc0  1400                .short          0x1400  /* max gy dithm */
3dc0
3dc0
3dc0                      /*****************************************************************/
3dc0                      /*       Clock globals defined...                                */
3dc0                      /*****************************************************************/
3dc0
3dc2                        .set clock_msr, 0x03000000 /* main status register, page & reg sel */
3dc2                        .set clock_ram_1E, 0x03000078 /* ram in either page */
3dc2                        .set clock_timer0_msb, 0x03000040 /* page select 0 */
3dc2                        .set clock_timer0_lsb, 0x0300003c /* page select 0 */
3dc2                        .set clock_timer1_msb, 0x03000048 /* page select 0 */
3dc2                        .set clock_timer1_lsb, 0x03000044 /* page select 0 */
3dc2                        .set clock_irr, 0x03000010 /* register select 0 */
3dc2                        .set clock_omr, 0x03000008 /* register select 1 */
3dc2                        .set clock_icr0, 0x0300000c /* register select 1 */
3dc2                        .set clock_icr1, 0x03000010 /* register select 1 */
3dc2                        .set clock_timer0_cntl, 0x03000004 /* register select 0 */
3dc2                        .set clock_timer1_cntl, 0x03000008 /* pg 0 reg sel 0 */
```

```
3dc2                    .set clock_years, 0x0300002c
3dc2                    .set clock_date_msb, 0x03000034
3dc2                    .set clock_date_lsb, 0x03000030
3dc2                    .set clock_hours, 0x03000020
3dc2                    .set clock_minutes, 0x0300001c
3dc2                    .set clock_seconds, 0x03000018
3dc2                    .set clock_hundredths, 0x03000014
3dc2                    .set clock_pfr, 0x0300000c
3dc2                    .set clock_rtm, 0x03000004
3dc2
3dc2                    .set init_timer_0_msb, 0x00fdff00  /* watchdog timer init values */
3dc2                    .set init_timer_0_lsb, 0x0070ffa4  /* clk 0  5ms, 164 = (00a4)h
3dc2                                                          clk 1 1.999s, 65535 = (ffff)h
3dc2                                                          clk 2 1.979s, 64880 = (fd70)h */
3dc2
3dc2                  /*****************************************************************/
3dc2                  /*       Uart registers defined...                               */
3dc2                  /*****************************************************************/
3dc2
3dc2                    .set uarta_rbr, 0x04000000 /* a rec buf reg, read only dlab=0 */
3dc2                    .set uarta_thr, 0x04000000 /* a xmitr hold reg, write only, dlab=0 */
3dc2                    .set uarta_ier, 0x04000004 /* a interrupt enable register */
3dc2                    .set uarta_iir, 0x04000008 /* a interrupt ident. reg, read only */
3dc2                    .set uarta_fcr, 0x04000008 /* a fifo control register, write only */
3dc2                    .set uarta_lcr, 0x0400000C /* a line control register */
3dc2                    .set uarta_mcr, 0x04000010 /* a modem control register */
3dc2                    .set uarta_lsr, 0x04000014 /* a line status register */
3dc2                    .set uarta_msr, 0x04000018 /* a modem status register */
3dc2                    .set uarta_scr, 0x0400001C /* a scratch register */
3dc2                    .set uarta_dll, 0x04000000 /* a divisor latch, ls, dlab=1 */
3dc2                    .set uarta_dlm, 0x04000004 /* a divisor latch, ms, dlab=1 */
3dc2                    .set uarta_afr, 0x04000008 /* a alternate function register */
3dc2                    .set uartb_rbr, 0x04000020 /* b receive buf reg, read only dlab=0 */
3dc2                    .set uartb_thr, 0x04000020 /* b xmitr hold reg, write only, dlab=0 */
3dc2                    .set uartb_ier, 0x04000024 /* b interrupt enable register */
3dc2                    .set uartb_iir, 0x04000028 /* b interrupt ident. reg, read only */
3dc2                    .set uartb_fcr, 0x04000028 /* b fifo control register, write only */
3dc2                    .set uartb_lcr, 0x0400002C /* b line control register */
3dc2                    .set uartb_mcr, 0x04000030 /* b modem control register */
3dc2                    .set uartb_lsr, 0x04000034 /* b line status register */
3dc2                    .set uartb_msr, 0x04000038 /* b modem status register */
3dc2                    .set uartb_scr, 0x0400003C /* b scratch register */
3dc2                    .set uartb_dll, 0x04000020 /* b divisor latch, ls, dlab=1 */
3dc2                    .set uartb_dlm, 0x04000024 /* b divisor latch, ms, dlab=1 */
3dc2                    .set uartb_afr, 0x04000028 /* b alternate function register */
3dc2
3dc2                    .set divisor, 260
3dc2                    .set lcr_dlab1, 0x83  /* dlab=1, no parity, 1 stop, 8 data */
3dc2                    .set lcr_dlab0, 0x03  /* dlab=0, no parity, 1 stop, 8 data */
3dc2                    .set afr, 0x02  /* *mf=*baudout */
3dc2                    .set fcr, 0xCF  /* 11001111, rcvr trig lvl=14 bytes,
3dc2                               mode 1 txrdy/rxrdy, clear rcvr/xmtr fifos, enable fifos */
3dc2                    .set ier, 0x03  /* enable rcvr/xmtr interrupt, disable others */
3dc2                    .set mcr, 0
3dc2
3dc2                  /*****************************************************************/
3dc2                  /*       IER globals defined...                                  */
3dc2                  /*****************************************************************/
3dc2
3dc2                    .set hap_ier, 0x05000000
3dc2
3dc2                  /*****************************************************************/
3dc2                  /*       Initialization variables defined...                     */
3dc2                  /*****************************************************************/
3dc2
3dc2                   .set intr_stack,     0x0101f000  /* 1f000 to 1f4ff */
3dc2                   .set sys_proc_stack, 0x0101f500  /* 1f500 to 1f8ff */
3dc2                   .set operation_prcb, 0x0101f900  /* 1f900 to 1f9ab */
3dc2                   .set operation_pcb,  0x0101fa00  /* 1fa00 to 1faeb */
3dc2                   .set intr_table,     0x0101fb00  /* 1fb00 to 1feff */
3dc2                   .set fault_table,    0x0101ff00  /* 1ff00 to 1fffb */
3dc2                   .set region_3,        0x0101e000  /* unused */
3dc2                   .set sys_proc_table, 0x0101d000  /* unused */
3dc2
3dc2                  /*****************************************************************/
3dc2                  /*       IAC definitions                                         */
3dc2                  /*****************************************************************/
3dc2
3dc2                   .set proc_icr_addr, 0xff000004  /* page 10-7 */
3dc2                   .set proc_iac, 0xff000010  /* iac address port */
3dc2
```

```
3dc2
3dc2                    /*************************************************************/
3dc2                    /* data definitions                                          */
3dc2                    /*************************************************************/
3dc2
3dc2                     .set frame_sync,0x5fa0f0a5  /* byte reversed, frame sync pattern */
3dc2                     .set output_rate,1000  /* number of iae intr before packet out */
3dc2                     .set  packet_length, 50  /* bytes in packet */
3dc2
3dc2                     .set normal_code, 0x0e0f120d  /* cntl (norm) */
3dc2                     .set pass_through_code, 0x18060512  /* cntl (xfer) */
3dc2                     .set ack_flags_code, 0x010e0607
3dc2                     .set set_clk_nxt_trig_code, 0x130e1814
3dc2                     .set set_clk_now_code, 0x13030b09
3dc2                     .set dump_memory_code, 0x040d020b
3dc2                     .set load_memory_code, 0x0c0d1704
3dc2                     .set unlock_memory_code, 0x150e0c0b
3dc2                     .set lock_memory_code, 0x0c0f0319
3dc2                     .set execute_instruction_code, 0x18090104
3dc2
3dc2
3dc2                    /*************************************************************/
3dc2                    /* data locators in memory                                   */
3dc2                    /*************************************************************/
3dc2
3dc2                     .set flags, 0x01017000  /* 1 word */
3dc2                     .set more_flags, 0x01017004  /* 1 word */
3dc2                     .set fault_record, 0x01017008  /* 5 words */
3dc2                     .set hap_sw_version,0x01017200  /* 4 words */
3dc2
3dc2                     .set iae_monitor_points, 0x01017800  /* 21 short, 11 words */
3dc2                     .set iae_over_flag, 0x0101782c  /* 1 word */
3dc2                     .set iae_under_flag, 0x01017830  /* 1 word */
3dc2                     .set iae_over_points, 0x01017840  /* 21 short, 11 words */
3dc2                     .set iae_under_points, 0x01017860  /* 21 short, 11 words */
3dc2                     .set iae_discrete_points, 0x01017900  /* 3 short, 2 words */
3dc2                     .set mux_item_count, 0x01017f00  /* 1 byte, 1 word */
3dc2                     .set iae_sample_count, 0x01017f80  /* 1 byte, 1 word */
3dc2
3dc2                     .set normal_packet, 0x01018000  /* 50 bytes, 13 words */
3dc2                     .set trigger_packet, 0x01018100  /* 50 bytes, 13 words */
3dc2                     .set memory_packet, 0x01018200  /* 50 bytes, 13 words */
3dc2                     .set frame_count, 0x01018700  /* 1 word */
3dc2
3dc2                     .set solution_time, 0x01018800  /* 9 words */
3dc2                     .set trigger_time, 0x01018900  /* 9 words */
3dc2                     .set desired_time, 0x01018f00  /* 9 words */
3dc2
3dc2                     .set grid_tx_buffer, 0x01019000  /* 256 bytes, 64 words */
3dc2                     .set grid_rx_buffer, 0x01019100  /* 256 bytes, 64 words */
3dc2                     .set esc_tx_buffer, 0x01019200  /* 256 bytes, 64 words */
3dc2                     .set esc_rx_buffer, 0x01019300  /* 256 bytes, 64 words */
3dc2                     .set grid_tx_head, 0x01019400  /* 1 byte, 1 word */
3dc2                     .set grid_tx_tail, 0x01019404  /* 1 byte, 1 word */
3dc2                     .set grid_rx_head, 0x01019408  /* 1 byte, 1 word */
3dc2                     .set grid_rx_tail, 0x0101940c  /* 1 byte, 1 word */
3dc2                     .set esc_tx_head, 0x01019410  /* 1 byte, 1 word */
3dc2                     .set esc_tx_tail, 0x01019414  /* 1 byte, 1 word */
3dc2                     .set esc_rx_head, 0x01019418  /* 1 byte, 1 word */
3dc2                     .set esc_rx_tail, 0x0101941c  /* 1 byte, 1 word */
3dc2                     .set command_buffer, 0x01019500  /* 12 bytes, 3 words */
3dc2                     .set command_count, 0x01019600  /* 1 byte, 1 word */
3dc2                     .set long_command_character_count, 0x01019604  /* 1 byte, 1 word */
3dc2
3dc2                     .set _pin, 0x0101a000  /* 3 words */
3dc2                     .set _a, 0x0101a100  /* 39 words */
3dc2                     .set _pfda, 0x0101a200  /* 3 double, 6 words */
3dc2                     .set _pb, 0x0101a300  /* 3 double, 6 words */
3dc2                     .set _tho, 0x0101a400  /* 4 double, 8 words */
3dc2                     .set _current_solution, 0x0101a500  /* 4 double, 8 words */
3dc2                     .set _solution_count, 0x0101a600  /* 1 word */
3dc2                     .set iae_interrupt_count, 0x0101a604  /* 1 byte, 1 word */
3dc2                     .set heartbeat, 0x0101a608  /* 1 byte, 1 word */
3dc2
3dc2
3dc2                          .globl startup_prcb
3dc2                          .globl startup_ip
3dc2                          .globl chkwd_1
3dc2
```

```
0000                            .text
0000                    seg_table:
0000  00000000                .word seg_table
0004  00001000                .word startup_prcb
0008  00000000                .word 0
000c  000012f0                .word startup_ip  /* label aligned on 16byte boundary */
0010  00000000                .word chkwd_1
0014  00000000                .word 0
0018  00000000                .word 0
001c  fffffff8                .word -8  /* lower 3 bits 0 for invalid */
001c
001c
001c                    /************************************************************************/
001c                    /*   Initailization segment table, (in PROM)                            */
001c                    /************************************************************************/
001c
001c                    /* entry 2 */
0020  00000000 00000000       .word 0,0,0,0
0028  00000000 00000000
0020
0020                    /* entry 3 */
0030  00000000 00000000       .word 0,0,0,0
0038  00000000 00000000
0030
0030                    /* entry 4 */
0040  00000000 00000000       .word 0,0,0,0
0048  00000000 00000000
0040
0040                    /* entry 5 */
0050  00000000 00000000       .word 0,0,0,0
0058  00000000 00000000
0050
0050                    /* entry 6 */
0060  00000000 00000000       .word 0,0,0,0
0068  00000000 00000000
0060
0060                    /* entry 7 */
0070  00000000 00000000       .word 0,0,0,0
0078  00000000 00000000
0070
0070                    /* entry 8, segment table SS */
0080                      .set seg_table_ss,(( . - seg_table ) << 2 | 0x3f )
0080  00000000              .word 0
0084  00000000              .word 0
0088  00000000              .word seg_table
008c  00fc00fb              .word 0x00fc00fb
008c
008c                    /* entry 9, proc table SS */
0090                      .set proc_table_ss, (( . - seg_table ) << 2 | 0x3f )
0090  00000000              .word 0
0094  00000000              .word 0
0098  0101d000              .word sys_proc_table
009c  304000fb              .word 0x304000fb
009c
009c                    /* entry 10, region 3 SS */
00a0                      .set region3_ss, (( . - seg_table ) << 2 | 0x3f )
00a0  00000000              .word 0
00a4  00000000              .word 0
00a8  00000000              .word 0
00ac  00fc00fb              .word 0x00fc00fb
00ac
00ac                    /* entry 11, dispatch port SS */
00b0                      .set dispatch_port_ss, (( . - seg_table ) << 2 | 0x3f )
00b0  00000000              .word 0
00b4  00000000              .word 0
00b8  00003f40              .word dispatch_port
00bc  504000fb              .word 0x504000fb
00bc
00bc                    /* entry 12, startup pcb ss */
00c0                      .set startup_pcb_ss, (( . - seg_table ) << 2 | 0x3f )
00c0  00000000              .word 0
00c4  00000000              .word 0
00c8  00001200              .word startup_pcb
00cc  204000fb              .word 0x204000fb
00cc
00cc                    /* entry 13, operation pcb ss */
00d0                      .set operation_pcb_ss, (( . - seg_table ) << 2 | 0x3f )
00d0  00000000              .word 0
00d4  00000000              .word 0
00d8  0101fa00              .word operation_pcb
00dc  204000fb              .word 0x204000fb
```

```
00dc
00dc                    /* entry 14, region 0 ss */
00e0                      .set region0_ss, (( . - seg_table ) << 2 | 0x3f )
00e0  00000000                  .word 0
00e4  00000000                  .word 0
00e8  00000000                  .word 0
00ec  00fc00fb                  .word 0x00fc00fb
00ec
00ec                    /* entry 15, region 1 ss */
00f0                      .set region1_ss, (( . - seg_table ) << 2 | 0x3f )
00f0  00000000                  .word 0
00f4  00000000                  .word 0
00f8  00000000                  .word 0
00fc  00fc00fb                  .word 0x00fc00fb
00fc
00fc                    /* entry 16, region 2 ss */
0100                      .set region2_ss, (( . - seg_table ) << 2 | 0x3f )
0100  00000000                  .word 0
0104  00000000                  .word 0
0108  00000000                  .word 0
010c  00fc00fb                  .word 0x00fc00fb
010c
0110                      .space ( 255 - 16 ) * 16  /* segment table entries 17 - 255 */
0110
0110
0110                    /*****************************************************************/
0110                    /*  Initialize PRCB, (in PROM)                                   */
0110                    /*****************************************************************/
0110
1000                      .align 4
1000
1000                    startup_prcb:
1000  00000000              .word   0x0                     /*   0 - res'v */
1004  00000008              .word   0x00000008              /*   4 - Processor Controls Word */
1008  00000000              .word   0x0                     /*   8 - res'v */
100c  0000033f              .word   startup_pcb_ss          /*  12 - Current process SS */
1010  000002ff              .word   dispatch_port_ss        /*  16 - Dispatch port SS */
1014  0101fb00              .word   intr_table              /*  20 - Inter. Table Phys. Addr. */
1018  0101f000              .word   intr_stack              /*  24 - Interrupt Stack Pointer */
101c  00000000              .word   0x0                     /*  28 - res'v */
1020  000002bf              .word   region3_ss              /*  32 - Region 3 SS */
1024  0000027f              .word   proc_table_ss           /*  36 - System Process table SS */
1028  000010b0              .word   prom_fault_table        /*  40 - Fault Table Phys. Addr. */
102c  00000000              .word   0x0                     /*  44 - reserved */
1030                        .space  12                      /*  48 - multiprocessor preemption */
103c  00000000              .word   0x0                     /*  60 - reserved */
1040                        .space  8                       /*  64 - idle time */
1048  00000000              .word   0x0                     /*  72 - system error fault */
104c  00000000              .word   0x0                     /*  76 - reserved */
1050                        .space  48                      /*  80 - resumption record */
1080                        .space  48                      /* 128 - system error fault record */
1080
1080
1080                    /*****************************************************************/
1080                    /* dispatch port                                                 */
1080                    /*****************************************************************/
3dc2                      .data
3dc2                      .align 9
3dc2
3f40                    dispatch_port:
3f40  00010000              .word     0x00010000  /* priority port */
3f44  00000000              .word      0x0
3f48                       .space    32 * 8
3f48
10b0                      .text
10b0
10b0
```

```
10b0                    /*************************************************************************/
10b0                    /* add the fault table into prom */
10b0                    /*************************************************************************/
10b0
10b0                    #line 1 "hap.src:ftable.inc"
10b0                    /* ***********************************************************************/
10b0                    /*    User Fault Table                                                   */
10b0                    /*                                                                       */
10b0                    /*    R. Higgins                                                         */
10b0                    /*    10/15/91                                                           */
10b0                    /*                                                                       */
10b0                    /* all entries are local procedure entries                               */
10b0                    /* first word is on a word boundary and is the procedure address         */
10b0                    /* second word is set to 0                                               */
10b0                    /* ***********************************************************************/
10b0
10b0                            .globl   prom_fault_table
10b0
10b0                    prom_fault_table:
10b0
10b0                           .globl override_fault
10b0  00003664                 .word    override_fault
10b4  00000000                 .word    0
10b4
10b8                           .globl trace_fault
10b8  000036ac                 .word    trace_fault
10bc  00000000                 .word    0
10bc
10c0                           .globl operation_fault
10c0  00003718                 .word    operation_fault
10c4  00000000                 .word    0
10c4
10c8                           .globl arithmetic_fault
10c8  00003784                 .word    arithmetic_fault
10cc  00000000                 .word    0
10cc
10d0                           .globl floating_fault
10d0  000037cc                 .word     floating_fault
10d4  00000000                 .word    0
10d4
10d8                           .globl constraint_fault
10d8  00003814                 .word    constraint_fault
10dc  00000000                 .word    0
10dc
10e0                           .globl virtual_memory_fault
10e0  00003880                 .word    virtual_memory_fault
10e4  00000000                 .word    0
10e4
10e8                           .globl protection_fault
10e8  000038ec                 .word    protection_fault
10ec  00000000                 .word    0
10ec
10f0                           .globl machine_fault
10f0  00003934                 .word    machine_fault
10f4  00000000                 .word    0
10f4
10f8                           .globl structural_fault
10f8  000039a0                 .word    structural_fault
10fc  00000000                 .word    0
10fc
1100                           .globl type_fault
1100  000039e8                 .word    type_fault
1104  00000000                 .word    0
1104
1108  00000000                 .word    0          # Type 11 Reserved Fault Handler
110c  00000000                 .word    0
110c
1110                           .globl process_fault
1110  00003a30                 .word    process_fault
1114  00000000                 .word    0
1114
1118                           .globl descriptor_fault
1118  00003a78                 .word    descriptor_fault
111c  00000000                 .word    0
111c
1120                           .globl event_fault
1120  00003ac0                 .word    event_fault
1124  00000000                 .word    0
1124
1128  00000000                 .word    0          # Type 15 Reserved Fault Handler
112c  00000000                 .word    0
```

```
112c
1130  00000000                .word    0          # Type 16 Reserved Fault Handler
1134  00000000                .word    0
1134
1138  00000000                .word    0          # Type 17 Reserved Fault Handler
113c  00000000                .word    0
113c
1140  00000000                .word    0          # Type 18 Reserved Fault Handler
1144  00000000                .word    0
1144
1148  00000000                .word    0          # Type 19 Reserved Fault Handler
114c  00000000                .word    0
114c
1150  00000000                .word    0          # Type 20 Reserved Fault Handler
1154  00000000                .word    0
1154
1158  00000000                .word    0          # Type 21 Reserved Fault Handler
115c  00000000                .word    0
115c
1160  00000000                .word    0          # Type 22 Reserved Fault Handler
1164  00000000                .word    0
1164
1168  00000000                .word    0          # Type 23 Reserved Fault Handler
116c  00000000                .word    0
116c
1170  00000000                .word    0          # Type 24 Reserved Fault Handler
1174  00000000                .word    0
1174
1178  00000000                .word    0          # Type 25 Reserved Fault Handler
117c  00000000                .word    0
117c
1180  00000000                .word    0          # Type 26 Reserved Fault Handler
1184  00000000                .word    0
1184
1188  00000000                .word    0          # Type 27 Reserved Fault Handler
118c  00000000                .word    0
118c
1190  00000000                .word    0          # Type 28 Reserved Fault Handler
1194  00000000                .word    0
1194
1198  00000000                .word    0          # Type 29 Reserved Fault Handler
119c  00000000                .word    0
119c
11a0  00000000                .word    0          # Type 30 Reserved Fault Handler
11a4  00000000                .word    0
11a4
11a8  00000000                .word    0          # Type 31 Reserved Fault Handler
11ac  00000000                .word     0
11ac
11ac
11ac                    /*************************************************************************/
11ac                    /*          Initialize PCB, (in PROM)                                    */
11ac                    /*************************************************************************/
11ac
11b0                      .globl startup_pcb
11b0
11b0                      .align 8
11b0
1200                    startup_pcb:
1200  00000000                .word   0x0                 /* link ss */
1204  00000000                .word   0x0                 /* current port/semaphone ss */
1208  00000000                .word   0x0                 /* receive message */
120c  000002ff                .word   dispatch_port_ss    /* dispatch port ss */
1210  00000000                .word   0x0                 /* residual time slice */
1214  00032002                .word   0x00032002          /* process controls word */
1218  00000000                .word   0x0                 /* process notice */
121c  00000000                .word   0x0                 /* trace contrls */
1220                          .space  16                  /* reserved */
1230  000003bf                .word   region0_ss          /* region 0 ss */
1234  000003ff                .word   region1_ss          /* region 1 ss */
1238  0000043f                .word   region2_ss          /* region 2 ss */
123c                          .word   0x3f001000          /* arithmetic controls
123c                                                        round to nearest,
123c  3f001000                                              mask all arithmetic faults */
1240  00000000                .word   0x0                 /* reserved */
1244  00000000                .word   0x0                 /* next time slice */
1248                          .space  8                   /* execution time */
1250                          .space  48                  /* resumption record */
1280                          .space  4 * 15              /* global registers g0 - g14 */
12bc  0101f000                .word   intr_stack          /* frame pointer g15 */
12c0                          .space  12 * 4              /* floating point registers */
```

```
12c0
12c0
12c0                    /************************************************************************/
12c0                    /*       START EXECUTION HERE                                           */
12c0                    /*         Clear error flag global registers                            */
12c0                    /*         HAP interrupt enable register global variable                */
12c0                    /************************************************************************/
12c0
12f0                      .globl hap_ier
12f0                      .globl flags
12f0                      .globl more_flags
12f0
12f0                      .align 4  /* align label on 16 byte boundary */
12f0
12f0  5cf01e00          startup_ip:  mov      0,g14
12f0
12f4  5c801e00                       ldconst  0x0,g0  /* clear error flag register */
12f8  5c881e00                       ldconst  0x0,g1  /* clear more_flags register */
12fc  5c901e00                       ldconst  0x0,g2  /* clear the fault record */
1300  5c981e00                       ldconst  0x0,g3
1304  5ca01e00                       ldconst  0x0,g4
1308  5ca81e00                       ldconst  0x0,g5
130c  5cb01e00                       ldconst  0x0,g6
130c
1310  5c981e00          restart:     ldconst  0x0,g3
1314  82983000 05000000             stob     g3,hap_ier  /* clear the IER register */
1314
1314                    /************************************************************************/
1314                     /* compute and verify prom checksums, error in bit 0 of g0 and hap_ier */
1314                    /************************************************************************/
1314
1314                    #line 1 "hap.src:chkprom.src"
1314                    /************************************************************************/
1314                    /*   Compute and verify PROM checksums...                               */
1314                    /*     This is the PROM checksum portion of the INIT code.  Code will   */
1314                    /*     verify that the HAP PROM is sound by verifying checksum of the   */
1314                    /*     entire PROM.  This routine will add all the data in the PROM     */
1314                    /*     together and verify that the PROM checksum is zero, a word is    */
1314                    /*     defined in the data section to force the checksum to zero.       */
1314                    /*     R. Higgins                                                       */
1314                    /*     7/25/91                                                          */
1314                    /*     11/10/91  added checksum to zero modification                    */
1314                    /*                                                                      */
1314                    /************************************************************************/
1314                    /*   Registers used for the PROM checksum stuff:                        */
1314                    /*     r4   Summing register (init to zero)                             */
1314                    /*     r5   PROM offset    (init to zero)                               */
1314                    /*     r6   input register                                              */
1314                    /*     r7   end of prom                                                 */
1314                    /*     g0   bit 0 Prom checksum error flag                              */
1314                    /*   Global variables used in the PROM test stuff:                      */
1314                    /*     prom$st   Starting address of PROM                               */
1314                    /*     prom$end  ending address of PROM                                 */
1314                    /*                                                                      */
1314                    /************************************************************************/
1314
131c                      .globl prom$st
131c                      .globl prom$end
131c                      .globl flags
131c                      .globl prom_tst
131c                      .globl program_checksum
4048                      .data
4048                         .word  program_checksum  /* this word makes the checksum in PROM
4048 00000000                                         turn out to be zero */
131c                    .text
131c
131c  8c383000 00007fff  prom_tst:     ldconst   (( prom$end - prom$st ) / 4 ),r7
1324  5c201e00                         ldconst   0x0,r4
1328  5c281e00                         ldconst   0x0,r5
1328
132c  90303905 00000000  prom_addr:    ld        prom$st [r5*4],r6 /* load prom word */
1334  59218004                         addo      r4,r6,r4  /* sum values */
1338  59294801                         addo      0x01,r5,r5  /* inc address -1 */
133c  3629dff0                         cmpoble   r5,r7,prom_addr
1340  32012010                         cmpobe    0x00,r4,ram_tst  /* no er, go to ram tst */
1340
1344  58840980          chksum_error: setbit    0,g0,g0
1348  82803000 05000000               stob      g0,hap_ier  /* turn off heartbeat */
1348
1348
1348
```

```
1348                    /*************************************************************************/
1348                    /* check ram, error in bit 1 of g0 and hap_ier, leave zeroes             */
1348                    /*************************************************************************/
1348
1348                    #line 1 "hap.src:chkram.src"
1348                    /*************************************************************************/
1348                    /*   Initialize RAM...                                                   */
1348                    /*     This is the RAM test portion of the INIT code.  This code will    */
1348                    /*     verify that the HAP ram is sound by walking a logic ONE through   */
1348                    /*     every bit of RAM. After the RAM is checked, this routine will     */
1348                    /*     check the burst mode logic by doing long word, triple word and    */
1348                    /*     quad word writes and verifies to RAM memory.                      */
1348                    /*     R. Higgins                                                        */
1348                    /*     7/10/91                                                           */
1348                    /*     10/20/91 eliminated ram$size definition                           */
1348                    /*************************************************************************/
1348                    /*   Registers used for the RAM test stuff:                              */
1348                    /*     r4  Pattern register  (init to zero)                              */
1348                    /*     r5  Ram offset     (init to zero)                                 */
1348                    /*     r6  input register                                               */
1348                    /*     r7  end of ram                                                   */
1348                    /*     g0 bit 1 Ram test failed flag                                    */
1348                    /*                                                                      */
1348                    /*   Global variables used in the RAM test stuff:                        */
1348                    /*     ram$st  Starting address of RAM                                  */
1348                    /*     ram$end  End of RAM space - 4 bytes                              */
1348                    /*                                                                      */
1348                    /*   Note: Ram space is equal to ram$st + ram$end                       */
1348                    /*                                                                      */
1348                    /*************************************************************************/
1348
1350                      .globl  ram$st
1350                      .globl  ram$end
1350                      .globl  flags
1350                      .globl  ram_tst
1350
1350  5c281e00          ram_tst:      ldconst  0x0,r5
1354  8c383000 00007fff               ldconst  (( ram$end - ram$st ) / 4 ),r7
135c  5c201e00                        ldconst  0x0,r4
135c
1360  92203905 01000000  ram_addr:     st       r4,ram$st [r5]  /* store pattern */
1368  90303905 01000000               ld       ram$st [r5],r6  /* read pattern */
1370  3521803c                        cmpobne  r4,r6,ram_error
1374  5c201e01                        ldconst  0x01,r4
1378  92203905 01000000               st       r4,ram$st [r5]  /* store pattern */
1380  90303905 01000000               ld       ram$st [r5],r6  /* read pattern */
1388  35218024                        cmpobne  r4,r6,ram_error
1388
138c  59210e01          ram_bit:      shlo     1,r4,r4  /* change pattern */
1390  92203905 01000000               st       r4,ram$st [r5]  /* store pattern */
1398  90303905 01000000               ld       ram$st [r5],r6  /* read pattern */
13a0  3521800c                        cmpobne  r4,r6,ram_error /* check ram contents */
13a4  37f92020                        bbs      31,r4,inc_ram_addr
13a8  ffffe4 08                       b        ram_bit
13a8
13ac  5c201e00          ram_error:    ldconst  0x0,r4
13b0  92203905 01000000               st       r4,ram$st [r5]
13b8  58840981                        setbit   1,g0,g0  /* set error flag */
13bc  82803000 05000000               stob     g0,hap_ier  /* set hw error flag */
13bc
13c4  5c201e00          inc_ram_addr: ldconst  0x0,r4  /* clear the ram location */
13c8  92203905 01000000               st       r4,ram$st [r5]
13d0  59294801                        addo     0x01,r5,r5  /* increment address -1 */
13d4  3629df8c                        cmpoble  r5,r7,ram_addr
13d4
13d4
13d4
13d4                    /*************************************************************************/
13d4                    /* move the information in global registers to ram                       */
13d4                    /*************************************************************************/
13d4
13d4                                  /* save the flags */
13d8  92803000 01017000               st       g0,flags
13e0  92883000 01017004               st       g1,more_flags
13e0
13e0                                  /* save the fault info */
13e8  92903000 01017008               st      g2,fault_record
13f0  92983000 0101700c               st      g3,fault_record + 1*4
13f8  92a03000 01017010               st      g4,fault_record + 2*4
1400  92a83000 01017014               st      g5,fault_record + 3*4
1408  92b03000 01017018               st      g6,fault_record + 4*4
```

```
1408
1408
1408                    /*************************************************************************/
1408                    /* check and configure the uart, error in bit 2 of flags and hap_ier  */
1408                    /*************************************************************************/
1408
1408                    #line 1 "hap.src:chkuart.src"
1408                    /*************************************************************************/
1408                    /*   Initialize UART...                                                  */
1408                    /*     This is the UART test portion of the INIT code.  This code will */
1408                    /*     verify that the HAP UART is sound by walking a ONE through      */
1408                    /*     the scratch reg. After the UART is checked, this routine will   */
1408                    /*     configure the uart for the following:                           */
1408                    /*       divisor# = freq/(baudrate*16)                                 */
1408                    /*         @10MHz, divisor=521, baud=1200                              */
1408                    /*         @10MHz, divisor=260, baud=2400  <- used by HAP              */
1408                    /*       dlab=1, odd parity, 1 stop, 8 data bits                       */
1408                    /*       *mf=*baudout                                                  */
1408                    /*       rcvr trig lvl=14 bytes                                        */
1408                    /*       1 txrdy/rxrdy, clear rcvr/xmtr fifos, enable fifos            */
1408                    /*       enable rcvr/xmtr interrupt, disable others                    */
1408                    /*************************************************************************/
1408                    /*   Registers used for the UART stuff:                                  */
1408                    /*     r3, r4, r5, r6, r7, r9, r10, r11, r12, r13, r15                   */
1408                    /*     g0 bit 3 uart error flag                                        */
1408                    /*                                                                     */
1408                    /*************************************************************************/
1408
1410                      .globl uarta_rbr /* a receive buffer register, read only dlab=0 */
1410                      .globl uarta_thr /* a xmitr hold register, write only, dlab=0 */
1410                      .globl uarta_ier /* a interrupt enable register */
1410                      .globl uarta_iir /* a interrupt ident. register, read only */
1410                      .globl uarta_fcr /* a fifo control register, write only */
1410                      .globl uarta_lcr /* a line control register */
1410                      .globl uarta_mcr /* a modem control register */
1410                      .globl uarta_lsr /* a line status register */
1410                      .globl uarta_msr /* a modem status register */
1410                      .globl uarta_scr /* a scratch register */
1410                      .globl uarta_dll /* a divisor latch, ls, dlab=1 */
1410                      .globl uarta_dlm /* a divisor latch, ms, dlab=1 */
1410                      .globl uarta_afr /* a alternate function register */
1410                      .globl uartb_rbr /* b receive buffer register, read only dlab=0 */
1410                      .globl uartb_thr /* b xmitr hold register, write only, dlab=0 */
1410                      .globl uartb_ier /* b interrupt enable register */
1410                      .globl uartb_iir /* b interrupt ident. register, read only */
1410                      .globl uartb_fcr /* b fifo control register, write only */
1410                      .globl uartb_lcr /* b line control register */
1410                      .globl uartb_mcr /* b modem control register */
1410                      .globl uartb_lsr /* b line status register */
1410                      .globl uartb_msr /* b modem status register */
1410                      .globl uartb_scr /* b scratch register */
1410                      .globl uartb_dll /* b divisor latch, ls, dlab=1 */
1410                      .globl uartb_dlm /* b divisor latch, ms, dlab=1 */
1410                      .globl uartb_afr /* b alternate function register */
1410                      .globl flags
1410
1410                    /*************************************************************************/
1410                    /*   Test uart scratch register                                          */
1410                    /*************************************************************************/
1410                      .globl uart_tst
1410
1410  5c281e00          uart_tst:       ldconst  0x0,r5
1414  5c801e00                          ldconst  0x0, g0
1418  82803000 05000000                 stob     g0,hap_ier
1418
1420  5c201e00          uart_addr:      ldconst  0x0,r4
1424  82203000 0400001c                 stob     r4,uarta_scr  /* store pattern in a */
142c  80303000 0400001c                 ldob     uarta_scr,r6  /* read pattern in a */
1434  35218078                         cmpobne  r4,r6,uart_tst_error
1438  82203000 0400003c                 stob     r4,uartb_scr  /* store pattern in b */
1440  80303000 0400003c                 ldob     uartb_scr,r6  /* read pattern in b */
1448  35218064                         cmpobne  r4,r6,uart_tst_error
1448
144c  5c201e01                          ldconst  0x01,r4
1450  82203000 0400001c                 stob     r4,uarta_scr  /* store pattern in a */
1458  80303000 0400001c                 ldob     uarta_scr,r6  /* read pattern in a */
1460  3521804c                         cmpobne  r4,r6,uart_tst_error
1464  82203000 0400003c                 stob     r4,uartb_scr  /* store pattern in b */
146c  80303000 0400003c                 ldob     uartb_scr,r6  /* read pattern in b */
1474  35218038                         cmpobne  r4,r6,uart_tst_error
1474
```

```
1478  59210e01             uart_bit:       shlo    1,r4,r4  /* change pattern */
147c  82203000 0400001c                      stob    r4,uarta_scr  /* store pattern in a */
1484  80303000 0400001c                      ldob    uarta_scr,r6  /* read pattern in a */
148c  35218020                              cmpobne r4,r6,uart_tst_error
1490  82203000 0400003c                      stob    r4,uartb_scr  /* store pattern in b */
1498  80303000 0400003c                      ldob    uartb_scr,r6  /* read pattern in b */
14a0  3521800c                              cmpobne r4,r6,uart_tst_error
14a4  37392028                              bbs     7,r4,uart_init
14a8  ffffd0 08                             b       uart_bit
14a8
14ac  58840983             uart_tst_error: setbit  3,g0,g0  /* set error flag */
14b0  82803000 05000000                      stob    g0,hap_ier  /* set hw error flag */
14b8  90203000 01017000                      ld      flags,r4  /* store flag */
14c0  58210983                               setbit  3,r4,r4
14c4  92203000 01017000                      st      r4,flags
14c4
14c4                        /*************************************************************************/
14c4                        /*   Initialize uart and check setup                                     */
14c4                        /*************************************************************************/
14c4
14cc                        uart_init:
14cc
14cc                        /* line control register, dlab = 1  */
14cc  8c200083                   ldconst  lcr_dlab1,r4
14d0  82203000 0400000c          stob     r4,uarta_lcr
14d8  82203000 0400002c          stob     r4,uartb_lcr
14d8
14d8
14d8                        /* alternate function register */
14e0  5c201e02                   ldconst  afr,r4
14e4  82203000 04000008          stob     r4,uarta_afr
14ec  82203000 04000028          stob     r4,uartb_afr
14ec
14ec                               /* check the alternate function register */
14f4  80203000 04000008          ldob     uarta_afr, r4
14fc  35112250                   cmpobne  afr,r4,uart_init_error
1500  80203000 04000028          ldob     uartb_afr, r4
1508  35112244                   cmpobne  afr,r4,uart_init_error
1508
1508
1508                        /* set divisor latches, store divisor in both */
150c  8c200104                   ldconst  divisor,r4
1510  82203000 04000000          stob     r4,uarta_dll
1518  82203000 04000020          stob     r4,uartb_dll
1520  59290c08                   shro     8,r4,r5
1524  82283000 04000004          stob     r5,uarta_dlm
152c  82283000 04000024          stob     r5,uartb_dlm
152c
152c                               /* Check the divisor latch registers */
1534  80203000 04000004          ldob     uarta_dlm,r4  /* load msb */
153c  80283000 04000024          ldob     uartb_dlm,r5
1544  59210e08                   shlo     8,r4,r4
1548  59294e08                   shlo     8,r5,r5
154c  80303000 04000000          ldob     uarta_dll,r6  /* load lsb */
1554  80383000 04000020          ldob     uartb_dll,r7
155c  58210386                   or       r6,r4,r4  /* combile msb and lsb */
1560  58294387                   or       r7,r5,r5
1564  8c300104                   ldconst  divisor,r6
1568  353101e4                   cmpobne  r6,r4,uart_init_error
156c  353141e0                   cmpobne  r6,r5,uart_init_error
156c
156c
156c                        /* line control register, dlab = 0  */
1570  5c201e03                   ldconst  lcr_dlab0,r4
1574  82203000 0400000c          stob     r4,uarta_lcr
157c  82203000 0400002c          stob     r4,uartb_lcr
157c
157c                               /* check the line control register */
1584  80203000 0400000c          ldob     uarta_lcr,r4
158c  351921c0                   cmpobne  lcr_dlab0,r4,uart_init_error
1590  80203000 0400002c          ldob     uartb_lcr,r4
1598  351921b4                   cmpobne  lcr_dlab0,r4,uart_init_error
1598
1598
1598                        /* modem control register, store in both */
159c  5c201e00                   ldconst  mcr,r4
15a0  82203000 04000010          stob     r4,uarta_mcr  /* no loop back */
15a8  82203000 04000030          stob     r4,uartb_mcr
15a8
```

```
15a8                                    /* check the modem control register */
15b0  80203000 04000010         ldob    uarta_mcr, r4
15b8  35012194                  cmpobne  mcr,r4,uart_init_error
15bc  80203000 04000030         ldob    uartb_mcr,r4
15c4  35012188                  cmpobne  mcr,r4,uart_init_error
15c4
15c4
15c4                          /* fifo control register, store in both */
15c8  8c2000cf                  ldconst  fcr, r4
15cc  82203000 04000008         stob    r4, uarta_fcr
15d4  82203000 04000028         stob    r4, uartb_fcr
15d4
15d4
15d4                          /* interrupt enable register, store in both */
15dc  5c201e03                  ldconst  ier,r4
15e0  82203000 04000004         stob    r4,uarta_ier
15e8  82203000 04000024         stob    r4,uartb_ier
15e8
15e8                                    /* check the interrupt enable register */
15f0  80203000 04000004         ldob    uarta_ier,r4
15f8  35192154                  cmpobne  ier,r4,uart_init_error
15fc  80203000 04000024         ldob    uartb_ier,r4
1604  35192148                  cmpobne  ier,r4,uart_init_error
1604
1604
1608                          send_uart_string:     /* hello world */
1608  b0203000 0000404c         ldq     uart_string, r4  /* load uart string in r4,r5,r6,r7 */
1610  b2203000 01017200         stq     r4,hap_sw_version
1610
1618  82203000 04000000         stob    r4,uarta_thr   /* first character */
1620  82203000 04000020         stob    r4,uartb_thr
1628  59210c08                  shro    8,r4,r4
162c  82203000 04000000         stob    r4,uarta_thr   /* second character */
1634  82203000 04000020         stob    r4,uartb_thr
163c  59210c08                  shro    8,r4,r4
1640  82203000 04000000         stob    r4,uarta_thr   /* third character */
1648  82203000 04000020         stob    r4,uartb_thr
1650  59210c08                  shro    8,r4,r4
1654  82203000 04000000         stob    r4,uarta_thr   /* fourth character */
165c  82203000 04000020         stob    r4,uartb_thr
165c
1664  82283000 04000000         stob    r5,uarta_thr   /* fifth character */
166c  82283000 04000020         stob    r5,uartb_thr
1674  59294c08                  shro    8,r5,r5
1678  82283000 04000000         stob    r5,uarta_thr   /* sixth character */
1680  82283000 04000020         stob    r5,uartb_thr
1688  59294c08                  shro    8,r5,r5
168c  82283000 04000000         stob    r5,uarta_thr   /* seventh character */
1694  82283000 04000020         stob    r5,uartb_thr
169c  59294c08                  shro    8,r5,r5
16a0  82283000 04000000         stob    r5,uarta_thr   /* eighth character */
16a8  82283000 04000020         stob    r5,uartb_thr
16a8
16b0  82303000 04000000         stob    r6,uarta_thr   /* 9th character */
16b8  82303000 04000020         stob    r6,uartb_thr
16c0  59318c08                  shro    8,r6,r6
16c4  82303000 04000000         stob    r6,uarta_thr   /* 10th character */
16cc  82303000 04000020         stob    r6,uartb_thr
16d4  59318c08                  shro    8,r6,r6
16d8  82303000 04000000         stob    r6,uarta_thr   /* 11th character */
16e0  82303000 04000020         stob    r6,uartb_thr
16e8  59318c08                  shro    8,r6,r6
16ec  82303000 04000000         stob    r6,uarta_thr   /* 12th character */
16f4  82303000 04000020         stob    r6,uartb_thr
16f4
16fc  82383000 04000000         stob    r7,uarta_thr   /* 13th character */
1704  82383000 04000020         stob    r7,uartb_thr
170c  5939cc08                  shro    8,r7,r7
1710  82383000 04000000         stob    r7,uarta_thr   /* 14th character */
1718  82383000 04000020         stob    r7,uartb_thr
1720  5939cc08                  shro    8,r7,r7
1724  82383000 04000000         stob    r7,uarta_thr   /* 15th character */
172c  82383000 04000020         stob    r7,uartb_thr
1734  5939cc08                  shro    8,r7,r7
1738  82383000 04000000         stob    r7,uarta_thr   /* 16th character */
1740  82383000 04000020         stob    r7,uartb_thr
1740
1748  000028 08                 b       clock_tst
1748
```

```
174c                        uart_init_error:
174c  58840983                   setbit   3,g0,g0  /* set error flag */
1750  82803000 05000000           stob     g0,hap_ier  /* set hw error flag */
1758  90383000 01017000           ld       flags,r7  /* store flag */
1760  5839c983                    setbit   3,r7,r7
1764  92383000 01017000           st       r7,flags
1764
176c  000004 08                   b        clock_tst
176c
404c                          .data
404c
404c                          .align 2
404c
404c                        uart_string:
404c  52 20 46 20 48 69 67 67    .asciz "R F Higgins v2.5"
4054  69 6e 73 20 76 32 3e 35
405c  00
404c
1770                          .text
1770
1770
1770                        /*************************************************************************/
1770                         /* check and configure the clock chips, only configure watchdog timers.
1770                              errors shown in bits 8,9,10 of flags */
1770                        /*************************************************************************/
1770
1770                        #line 1 "hap.src:chkclock.src"
1770                        /*************************************************************************/
1770                        /*   Test GMT clock chips...                                             */
1770                        /*     This is the clock test portion of the INIT code.  This code will  */
1770                        /*     verify that the microprocessor can communicate with the GMT clk   */
1770                        /*     chips.  The general purpose clock ram loc 1E will be tested by    */
1770                        /*     walking a logic ONE through.  The GMT clock time is only          */
1770                        /*     initialized by command.                                           */
1770                        /*                                                                       */
1770                        /*     1/14/92 change watchdog timer control                             */
1770                        /*                                                                       */
1770                        /*************************************************************************/
1770                        /*   Registers used for the clock stuff:                                 */
1770                        /*     r4 pattern register                                               */
1770                        /*     r5 masked result register                                         */
1770                        /*     r6 result register                                                */
1770                        /*     r7 masked pattern register                                        */
1770                        /*     r8 comparison mask                                                */
1770                        /*     g0 bit 8 clock chip 0 setup error                                 */
1770                        /*     g0 bit 9 clock chip 1 setup error                                 */
1770                        /*     g0 bit 10 clock chip 2 setup error                                */
1770                        /*                                                                       */
1770                        /*************************************************************************/
1770
1770                          .globl clock_msr
1770                          .globl clock_ram_1E
1770                          .globl flags
1770                          .globl clock_tst
1770
1770  8c203000 003f3f3f   clock_tst:    ldconst  0x003f3f3f,r4
1778  8c403000 00ffffff                 ldconst  0x00ffffff,r8
1780  92203000 03000000                 st       r4,clock_msr /* page 0 register set 0 */
1780
1788  5c201e00                          ldconst  0x0,r4
178c  92203000 03000078                 st       r4,clock_ram_1E  /* store pattern */
1794  90303000 03000078                 ld       clock_ram_1E,r6  /* read pattern */
179c  58318088                          and      r8,r6,r6
17a0  35218048                          cmpobne  r4,r6,clock_error0
17a4  8c203000 00010101                 ldconst  0x00010101,r4
17ac  92203000 03000078                 st       r4,clock_ram_1E  /* store pattern */
17b4  90303000 03000078                 ld       clock_ram_1E,r6  /* read pattern */
17bc  58318088                          and      r8,r6,r6
17c0  35218028                          cmpobne  r4,r6,clock_error0
17c0
17c0
17c4  59210e01              clock_bit:  shlo     1,r4,r4  /* change pattern */
17c8  92203000 03000078                 st       r4,clock_ram_1E  /* store pattern */
17d0  90303000 03000078                 ld       clock_ram_1E,r6  /* read pattern */
17d8  58318088                          and      r8,r6,r6
17dc  3521800c                          cmpobne  r4,r6,clock_error0
17e0  37392088                          bbs      7,r4,clock_init
17e4  ffffe0 08                         b        clock_bit
17e4
```

```
17e8  8c4000ff            clock_error0: ldconst  0x000000ff,r8
17ec  58298088                          and      r8,r6,r5
17f0  58390088                          and      r8,r4,r7
17f4  3229c01c                          cmpobe   r5,r7,clock_error1
17f8  58840988                          setbit   8,g0,g0
17fc  90403000 01017000                 ld       flags,r8  /* set flags */
1804  58420988                          setbit   8,r8,r8
1808  92403000 01017000                 st       r8,flags
1810  8c403000 0000ff00   clock_error1:   ldconst  0x0000ff00,r8
1818  58298088                          and      r8,r6,r5
181c  58390088                          and      r8,r4,r7
1820  3229c01c                          cmpobe   r5,r7,clock_error2
1824  58840989                          setbit   9,g0,g0
1828  90403000 01017000                 ld       flags,r8  /* set flags */
1830  58420989                          setbit   9,r8,r8
1834  92403000 01017000                 st       r8,flags
183c  8c403000 00ff0000   clock_error2:   ldconst  0x00ff0000,r8
1844  58298088                          and      r8,r6,r5
1848  58390088                          and      r8,r4,r7
184c  3229c01c                          cmpobe   r5,r7,clock_init
1850  5884098a                          setbit   10,g0,g0
1854  90403000 01017000                 ld       flags,r8  /* set flags */
185c  5842098a                          setbit   10,r8,r8
1860  92403000 01017000                 st       r8,flags
1860
1860                              /*************************************************************************/
1860                              /*   Clock watchdog timer initialization...                              */
1860                              /*     set timer 0 msb,lsb value, set interrupt routing register, set    */
1860                              /*     output mode register, interrupt control register 0 and 1, and     */
1860                              /*     timer 0 control register.                                         */
1860                              /*************************************************************************/
1860
1868                                   .globl clock_timer0_msb
1868                                   .globl clock_timer0_lsb
1868                                   .globl clock_irr
1868                                   .globl clock_omr
1868                                   .globl clock_icr0
1868                                   .globl clock_icr1
1868                                   .globl clock_timer0_cntl
1868
1868  8c203000 0070ffa4   clock_init:  ldconst  init_timer_0_lsb,r4  /* timer 0 lsb count */
1870  8c403000 00ffffff                ldconst  0x00ffffff,r8  /* all bits read/write */
1878  92203000 0300003c                st       r4,clock_timer0_lsb
1880  90303000 0300003c                ld       clock_timer0_lsb,r6
1888  58318088                         and      r8,r6,r6
188c  352180c0                         cmpobne  r4,r6,clock_init_err0
188c
1890  8c203000 00fdff00                ldconst  init_timer_0_msb,r4  /* timer 0 msb count */
1898  92203000 03000040                st       r4,clock_timer0_msb
18a0  90303000 03000040                ld       clock_timer0_msb,r6
18a8  58318088                         and      r8,r6,r6
18ac  352180a0                         cmpobne  r4,r6,clock_init_err0
18ac
18b0  8c203000 00171708                ldconst  0x00171708,r4  /* interrupt routing reg */
18b8  8c403000 00bfbfbf                ldconst  0x00bfbfbf,r8  /* bit 6 rd only, disable */
18c0  92203000 03000010                st       r4,clock_irr  /* clk1&2 timer 0 to intr */
18c8  90303000 03000010                ld       clock_irr,r6  /* clk0 tim0 > mfo, fast pf */
18d0  58318088                         and      r8,r6,r6
18d4  35218078                         cmpobne  r4,r6,clock_init_err0
18d4
18d8  8c203000 007f7f7f                ldconst  0x007f7f7f,r4 /* select reg set 1 */
18e0  92203000 03000000                st       r4,clock_msr
18e0
18e8  8c203000 00bcb878                ldconst  0x00bcb878,r4  /* output mode register */
18f0  8c403000 00ffffff                ldconst  0x00ffffff,r8  /* all bits read/write */
18f8  92203000 03000008                st       r4,clock_omr  /* clk0 tim0 > mfo, pp, hi */
1900  90303000 03000008                ld       clock_omr,r6  /* clk1 intr, pp, lo */
1908  58318088                         and      r8,r6,r6      /* clk2 intr, pp, hi */
190c  35218040                         cmpobne  r4,r6,clock_init_err0
190c
1910  8c203000 00404040                ldconst  0x00404040,r4  /* interrupt control reg0 */
1918  92203000 0300000c                st       r4,clock_icr0  /* timer 0 enable */
1920  90303000 0300000c                ld       clock_icr0,r6
1928  58318088                         and      r8,r6,r6
192c  35218020                         cmpobne  r4,r6,clock_init_err0
192c
1930  8c200000                         lda      0x00000000,r4  /* interrupt control reg1 */
1934  92203000 03000010                st       r4,clock_icr1  /* pfw disable */
193c  90303000 03000010                ld       clock_icr1,r6
1944  58318088                         and      r8,r6,r6
1948  32218084                         cmpobe   r4,r6,clock_init_end
```

```
1948
194c  8c4000ff             clock_init_err0: ldconst  0x000000ff,r8
1950  58298088                             and      r8,r6,r5
1954  58390088                             and      r8,r4,r7
1958  3229c01c                             cmpobe   r5,r7,clock_init_err1
195c  58840988                             setbit   8,g0,g0
1960  90403000 01017000                    ld       flags,r8  /* set flags */
1968  58420988                             setbit   8,r8,r8
196c  92403000 01017000                    st       r8,flags
1974  8c403000 0000ff00   clock_init_err1: ldconst  0x0000ff00,r8
197c  58298088                             and      r8,r6,r5
1980  58390088                             and      r8,r4,r7
1984  3229c01c                             cmpobe   r5,r7,clock_init_err2
1988  58840989                             setbit   9,g0,g0
198c  90403000 01017000                    ld       flags,r8  /* set flags */
1994  58420989                             setbit   9,r8,r8
1998  92403000 01017000                    st       r8,flags
19a0  8c403000 00ff0000   clock_init_err2: ldconst  0x00ff0000,r8
19a8  58298088                             and      r8,r6,r5
19ac  58390088                             and      r8,r4,r7
19b0  3229c01c                             cmpobe   r5,r7,clock_init_end
19b4  5884098a                             setbit   10,g0,g0
19b8  90403000 01017000                    ld       flags,r8  /* set flags */
19c0  5842098a                             setbit   10,r8,r8
19c4  92403000 01017000                    st       r8,flags
19c4
19cc  8c203000 003f3f3f   clock_init_end:  ldconst  0x003f3f3f,r4 /* page 0 reg set 0 */
19d4  92203000 03000000                    st       r4,clock_msr
19d4
19d4
19d4                                 /*************************************************************************/
19d4                                  /* make the system data structures in ram, restart processor using new
19d4                                     data structures, bind to the processor */
19d4                                 /*************************************************************************/
19d4
19dc  85783000 00003b58                      balx     make_data_structures,r15
19dc
19dc
19dc                                 /*************************************************************************/
19dc                                 /* call the executive                                                    */
19dc                                 /*************************************************************************/
19dc
19dc                                 #line 1 "hap.src:exec.src"
19dc                                 /*************************************************************************/
19dc                                 /*   EXEC.SRC                                                            */
19dc                                 /*     Executive routine.  This routine manages the serial activity,     */
19dc                                 /*     does self tests, interprets commands, and sets the mode of        */
19dc                                 /*     operation.                                                        */
19dc                                 /*                                                                       */
19dc                                 /*     9/10/91  r.h.                                                     */
19dc                                 /*     9/12/91 changed uart ports used                                   */
19dc                                 /*     9/15/91 added pass through mode and frame counter                 */
19dc                                 /*     9/20/91 add command detection                                     */
19dc                                 /*     11/10/91 added time correction command                            */
19dc                                 /*     12/31/91 initialize reset and lost track of imu flags             */
19dc                                 /*     1/13/92 changed normal, passthrough and ack flags cmd             */
19dc                                 /*     1/21/92 initialize iae registers                                  */
19dc                                 /*     2/4/92  add memory frame type and new commands                    */
19dc                                 /*     2/19/92 add new commands, change warmstart                        */
19dc                                 /*     3/9/92 force attitude frames as default, clk cmds  leap yr        */
19dc                                 /*     3/11/92 cmds force attitude frames, fixed gyro monitor, mem dump */
19dc                                 /*     3/30/92 take multiple samples of iae monitor points, report last */
19dc                                 /*     3/31/92 change dump command to report 0 time, add HIMU over under*/
19dc                                 /*               flags, fix set clocks now command                       */
19dc                                 /*************************************************************************/
19dc
19e4                                   .globl exec_init
19e4
19e4                                 exec_init:    /* initialize ring buffer head and tail */
19e4  5c201e00                                     ldconst  0x0,r4
19e8  5c281e00                                     ldconst  0x0,r5
19ec  5c301e00                                     ldconst  0x0,r6
19f0  5c381e00                                     ldconst  0x0,r7
19f4  82203000 01019400                            stob     r4,grid_tx_head
19fc  82203000 01019404                            stob     r4,grid_tx_tail
1a04  82203000 01019408                            stob     r4,grid_rx_head
1a0c  82203000 0101940c                            stob     r4,grid_rx_tail
1a14  82203000 01019410                            stob     r4,esc_tx_head
1a1c  82203000 01019414                            stob     r4,esc_tx_tail
1a24  82203000 01019418                            stob     r4,esc_rx_head
```

```
1a2c  82203000 0101941c                stob     r4,esc_rx_tail
1a2c
1a34  82203000 01018700                stob     r4,frame_count
1a34
1a3c  82203000 01019600                stob     r4,command_count
1a3c
1a44  82203000 01019604                stob     r4,long_command_character_count
1a44
1a4c  b2203000 01019500                stq      r4,command_buffer
1a4c
1a4c                                   /* initialize the quaternion */
1a54  6d801c96                         movrl    0d1.0,g0
1a58  6d901c90                         movrl    0d0.0,g2
1a5c  6da01c90                         movrl    0d0.0,g4
1a60  6db01c90                         movrl    0d0.0,g6
1a60
1a64  b2803000 0101a500                stq      g0,_current_solution
1a6c  b2a03000 0101a510                stq      g4,_current_solution + 16
1a6c
1a6c                                   /* initialize the solution counter */
1a74  8a203000 0101a600                stos     r4,_solution_count
1a74
1a74                                   /* initialize iae filter storage */
1a7c                             .globl _a
1a7c  b2203000 0101a100                stq      r4,_a
1a84  b2203000 0101a110                stq      r4,_a + 16
1a8c  b2203000 0101a120                stq      r4,_a + 32
1a8c
1a8c                                   /* initialize iae processing storage */
1a94                             .globl _pfda
1a94  b2203000 0101a200                stq      r4,_pfda
1a9c  b2203000 0101a210                stq      r4,_pfda + 16
1a9c
1aa4                             .globl _pin
1aa4  b2203000 0101a000                stq      r4,_pin
1aa4
1aac                             .globl _pb
1aac  b2203000 0101a300                stq      r4,_pb
1ab4  b2203000 0101a310                stq      r4,_pb + 16
1ab4
1abc                             .globl _tho
1abc  b2203000 0101a400                stq      r4,_tho
1ac4  b2203000 0101a410                stq      r4,_tho + 16
1ac4
1ac4                                   /* read iae interrupt counter and store */
1acc  90203000 03000000                ld       clock_msr,r4
1ad4  59210e04                         shlo     4,r4,r4
1ad8  59210c1c                         shro     28,r4,r4
1adc  82203000 0101a604                stob     r4,iae_interrupt_count
1adc
1ae4                           exec_warmstart:
1ae4                                   /* initialize iae registers */
1ae4  5c201e0e                         ldconst  0x0000000e,r4
1ae8  8a203000 0200001c                stos     r4,iae_odp
1af0  5c201e00                         ldconst  0x00000000,r4
1af4  8a203000 02000000                stos     r4,iae_cr
1af4
1af4                                   /* initialize iae variables */
1afc  5c201e00                         ldconst  0x00,r4
1b00  82203000 01017f00                stob     r4,mux_item_count
1b08  82203000 01017f80                stob     r4,iae_sample_count
1b08
1b08                                   /* set the watchdog timer */
1b10  8c203000 003f3f3f                ldconst  0x003f3f3f,r4  /* select reg pg 0 reg set 0 */
1b18  92203000 03000000                st       r4,clock_msr
1b18
1b20  8c203000 008f8f8f                ldconst  0x008f8f8f,r4  /* timer 0 control reg */
1b28  92203000 03000004                st       r4,clock_timer0_cntl  /* mode 3, osc, start cnt */
1b28
1b28                                      /* enable interrupts */
1b30  90203000 01017000                ld       flags,r4
1b38  58210986                         setbit   6,r4,r4  /* out 4 now, set bad access intr en */
1b3c  58210986                         setbit   6,r4,r4  /* set serial interrupt enable */
1b40  58210985                         setbit   5,r4,r4  /* set trigger interrupt enable */
1b44  58210984                         setbit   4,r4,r4  /* set iae interrupt enable */
1b48  58210983                         setbit   3,r4,r4  /* set watchdog interrupt enable */
1b48
1b4c  58210992                         setbit   18,r4,r4  /* set reset occurred flag */
1b50  58210995                         setbit   21,r4,r4  /* set lost track of IMU flag */
1b50
1b54  92203000 01017000                st       r4,flags  /* store flags */
```

```
1b54
1b54                                    /* set more_flags */
1b5c  90283000 01017004                 ld      more_flags,r5
1b64  58294987                          setbit  7,r5,r5  /* set interrupts enabled bit */
1b68  58294e05                          clrbit  5,r5,r5  /* attitude frames */
1b6c  58294e06                          clrbit  6,r5,r5  /* attitude frames */
1b70  92283000 01017004                 st      r5,more_flags
1b70
1b78  82203000 05000000                 stob    r4,hap_ier  /* set hap int hardware */
1b78
1b78
1b78                                /*************************************************************************/
1b78                                /* executive major loop                                                  */
1b78                                /*************************************************************************/
1b80                                  .globl exec_top
1b80
1b80                                exec_top:   /* check for general status */
1b80  90203000 03000000                 ld      clock_msr,r4
1b80
1b80                                    /* set esceb power flag */
1b80                                    /* 0 - power on, 1 - power off */
1b88  8c403000 01017000                 ldconst  flags,r8
1b90  59485e11                          ldconst  0x00020000,r9
1b94  5c501604                          mov      r4,r10
1b98  59528c0b                          shro     11,r10,r10
1b9c  61524008                          atmod    r8,r9,r10  /* modify bit 17 of flags */
1b9c
1b9c                                    /* set battery voltage flag */
1b9c                                    /* 0 - bat good, 1 - bat low */
1ba0  59485e14                          ldconst  0x00100000,r9
1ba4  5c501604                          mov      r4,r10
1ba8  59528c0a                          shro     10,r10,r10
1bac  61524008                          atmod    r8,r9,r10  /* modify bit 20 of flags */
1bac
1bac                                    /* check IAE/IMU status */
1bb0  88203000 02000014                 ldos     iae_ifp,r4  /* read iae internal flag port */
1bb8  37792104                          bbs      15,r4,exec_service_check  /* see if a/d ready */
1bb8
1bbc  c8503000 0200003c                 ldis     iae_ad_result,r10  /* read value */
1bc4  80203000 01017f00                 ldob     mux_item_count,r4
1bcc  ca503884 01017800                 stis     r10,iae_monitor_points [ r4 * 2 ]  /* store val */
1bcc
1bd4  80283000 01017f80                 ldob     iae_sample_count,r5  /* take multiple samples */
1bdc  3121608c                          cmpobg   4,r5,start_next_conversion
1bdc
1bdc                                     /* make sure value is over min */
1be0  c8483884 00003d6c                 ldis     min_iae_mux_item_value [ r4 * 2 ],r9  /* min val */
1be8  3b524034                          cmpibge  r10,r9,chk_iae_max
1be8
1be8                                     /* set iae monitor error flag */
1bec  5c481e10                          ldconst  0x00000010,r9  /* mask */
1bf0  5c581e10                          ldconst  0x00000010,r11  /* value */
1bf4  8c603000 01017004                 ldconst  more_flags,r12
1bfc  615a400c                          atmod    r12,r9,r11  /* set bit 4 of more_flags */
1bfc
1bfc                                     /* store actual exceedence value */
1c00  ca503884 01017860                 stis     r10,iae_under_points [ r4 * 2 ]
1c00
1c00                                     /* set bit for point under min error */
1c08  90583000 01017830                 ld       iae_under_flag,r11
1c10  585ac184                          setbit   r4,r11,r11  /* set bit for point in error */
1c14  92583000 01017830                 st       r11,iae_under_flag
1c14
1c14                                     /* make sure value is under max */
1c1c  c8483884 00003d98   chk_iae_max:  ldis     max_iae_mux_item_value [ r4 * 2 ],r9  /* max val */
1c24  3e524034                          cmpible  r10,r9,inc_mux_item_count
1c24
1c24                                     /* set iae monitor error flag */
1c28  5c481e10                          ldconst  0x00000010,r9  /* mask */
1c2c  5c581e10                          ldconst  0x00000010,r11  /* value */
1c30  8c603000 01017004                 ldconst  more_flags,r12
1c38  615a400c                          atmod    r12,r9,r11  /* set bit 4 of more_flags */
1c38
1c38                                     /* store actual exceedence value */
1c3c  ca503884 01017840                 stis     r10,iae_over_points [ r4 * 2 ]
1c3c
1c3c                                     /* set bit for point over max error */
1c44  90583000 0101782c                 ld       iae_over_flag,r11
1c4c  585ac184                          setbit   r4,r11,r11  /* set bit for point in error */
1c50  92583000 0101782c                 st       r11,iae_over_flag
1c50
```

```
1c58                                inc_mux_item_count:
1c58  5c281e00                              ldconst  0x00,r5
1c5c  59210801                              addo     0x01,r4,r4  /* inc mux_item_count */
1c60  31a92008                              cmpobg   21,r4,start_next_conversion  /* chk for last item */
1c64  5c201e00                              ldconst  0x00,r4
1c64
1c68                                start_next_conversion:
1c68  82203000 01017f00                     stob     r4,mux_item_count
1c70  88483884 00003d40                     ldos     valid_iae_mux_addr [ r4 * 2 ],r9  /* ld item addr */
1c78  8a483000 02000038                     stos     r9,iae_amar  /* start next item conversion */
1c78
1c80  59294801                              addo     1,r5,r5  /* inc sample count */
1c84  82283000 01017f80                     stob     r5,iae_sample_count
1c84
1c84                                        /* read discrete points */
1c8c  88203000 02000014                     ldos     iae_ifp,r4  /* internal flag port */
1c94  8a203000 01017900                     stos     r4,iae_discrete_points
1c9c  88203000 0200001c                     ldos     iae_odp,r4  /* output discrete port */
1ca4  8a203000 01017902                     stos     r4,iae_discrete_points + 2
1cac  88203000 02000000                     ldos     iae_cr,r4  /* command register */
1cb4  8a203000 01017904                     stos     r4,iae_discrete_points + 4
1cb4
1cb4
1cbc                                exec_service_check:
1cbc  90203000 01017000                     ld       flags,r4
1cbc
1cc4  30d9200c                      chk_27:         bbc      27,r4,chk_26
1cc8  85783000 00001ed4                     balx     get_esc_data,r15
1cc8
1cd0  30d1200c                      chk_26:         bbc      26,r4,chk_25
1cd4  85783000 00001f44                     balx     send_esc_data,r15
1cd4
1cdc  30c9200c                      chk_25:         bbc      25,r4,chk_24
1ce0  85783000 00001fa8                     balx     get_grid_data,r15
1ce0
1ce8  30c1200c                      chk_24:         bbc      24,r4,what_mode
1cec  85783000 0000267c                     balx     send_grid_data,r15
1cec
1cf4  37f12014                      what_mode:      bbs      30,r4,pass_through  /* else packet mode */
1cf8  37f92010                              bbs      31,r4,pass_through
1cf8
1cfc  37e12100                              bbs      28,r4,load_packet
1d00  37e920fc                              bbs      29,r4,load_packet
1d00
1d04  fffe7c 08                             b        exec_top
1d04
1d04
1d04                                /*************************************************************************/
1d04                                /* pass through mode */
1d04                                /*************************************************************************/
1d04
1d08  30f13e78                      pass_through: bbc      30,r4,exec_top  /* only one bit set */
1d08
1d0c  8c4800ff                              ldconst  0x000000ff,r9  /* loop around mask */
1d0c
1d0c                                        /* check for grid data to pass */
1d10  80283000 01019408                     ldob     grid_rx_head,r5
1d18  80303000 0101940c                     ldob     grid_rx_tail,r6
1d18
1d20  32298064                              cmpobe   r5,r6,pass_esc_data
1d24  8c383000 01019100                     ldconst  grid_rx_buffer,r7
1d2c  80503000 01019410                     ldob     esc_tx_head,r10
1d34  80583000 01019414                     ldob     esc_tx_tail,r11
1d3c  8c603000 01019200                     ldconst  esc_tx_buffer,r12
1d3c
1d3c                                        /* transfer the grid rx buf to the esc tx buf */
1d44                                pass_grid_to_esc:
1d44  8041dc05                              ldob     (r7) [r5 * 1],r8  /* load from head */
1d48  59294801                              addo     0x01,r5,r5  /* inc head */
1d4c  58294089                              and      r9,r5,r5  /* keep head looping */
1d50  82283000 01019408                     stob     r5,grid_rx_head
1d50
1d58  82431c0b                              stob     r8, (r12) [r11 * 1]  /* store in tail */
1d5c  595ac801                              addo     0x01,r11,r11  /* inc tail */
1d60  585ac089                              and      r9,r11,r11  /* keep tail looping */
1d64  82583000 01019414                     stob     r11,esc_tx_tail  /* store new tail */
1d64
1d6c  3552c014                              cmpobne  r10,r11,check_pass_grid  /* check for overflow */
1d70  59528801                              addo     0x01,r10,r10  /* inc head */
1d74  58528089                              and      r9,r10,r10  /* keep head looping */
1d78  82503000 01019410                     stob     r10,esc_tx_head  /* store head */
```

```
1d78
1d80                        check_pass_grid:
1d80  35299fc4                        cmpobne  r5,r6,pass_grid_to_esc
1d80
1d80                                  /* pass the esc rx buf to the grid tx buf */
1d84  80283000 01019418     pass_esc_data:   ldob      esc_rx_head,r5
1d8c  80303000 0101941c               ldob     esc_rx_tail,r6
1d8c
1d94  32299dec                        cmpobe   r5,r6,exec_top  /* nothing to pass */
1d98  8c383000 01019300               ldconst  esc_rx_buffer,r7
1da0  80503000 01019400               ldob     grid_tx_head,r10
1da8  80583000 01019404               ldob     grid_tx_tail,r11
1db0  8c603000 01019000               ldconst  grid_tx_buffer,r12
1db0
1db8                        pass_esc_to_grid:
1db8  8041dc05                        ldob     (r7) [r5 * 1],r8  /* load from head */
1dbc  59294801                        addo     0x01,r5,r5  /* inc head */
1dc0  58294089                        and      r9,r5,r5
1dc4  82283000 01019418               stob     r5,esc_rx_head
1dc4
1dcc  82431c0b                        stob     r8, (r12) [r11 * 1]  /* store in tail */
1dd0  595ac801                        addo     0x01,r11,r11  /* inc tail */
1dd4  585ac089                        and      r9,r11,r11
1dd8  82583000 01019404               stob     r11,grid_tx_tail  /* store new tail */
1dd8
1de0  3552c014                        cmpobne  r10,r11,check_pass_esc  /* check for overflow */
1de4  59528801                        addo     0x01,r10,r10  /* inc head */
1de8  58528089                        and      r9,r10,r10
1dec  82503000 01019400               stob     r10,grid_tx_head  /* store head */
1dec
1df4                        check_pass_esc:
1df4  35299fc4                        cmpobne  r5,r6,pass_esc_to_grid
1df4
1df4
1df8  fffd88 08                       b        exec_top
1df8
1df8
1df8                        /*************************************************************************/
1df8                        /* load packet into transmit buffer routine */
1df8                        /*************************************************************************/
1df8
1dfc  594cd81f              load_packet:     ldconst  packet_length,r9
1e00  8c5800ff                        ldconst  0x000000ff,r11
1e04  8c683000 01019000               ldconst  grid_tx_buffer,r13
1e0c  90303000 01017004               ld       more_flags,r6
1e0c
1e0c                                  /* determine packet type needed */
1e14  8c283000 01018100               ldconst  trigger_packet,r5
1e1c  37792020                        bbs      15,r4,transfer  /* branch on trigger packet */
1e20  3781201c                        bbs      16,r4,transfer  /* branch on trigger packet */
1e24  8c283000 01018200               ldconst  memory_packet,r5
1e2c  3729a010                        bbs      5,r6,transfer  /* branch on memory frame */
1e30  3731a00c                        bbs      6,r6,transfer  /* branch on memory frame */
1e34  8c283000 01018000               ldconst  normal_packet,r5
1e34
1e34
1e3c  59624005              transfer:     addo     r5,r9,r12  /* calc last packet address */
1e3c
1e40  80303000 01019400               ldob     grid_tx_head,r6
1e48  80383000 01019404               ldob     grid_tx_tail,r7
1e48
1e50  3231c028                        cmpobe   r6,r7,fill_er_up
1e50
1e50                                  /* check for possible overflow */
1e54  5941c186                        subi     r6,r7,r8
1e58  3c022014                        cmpibl   0x0,r8,pos_overflow_chk
1e58
1e58                                  /* see if sign remains negative */
1e5c  59524007                        addo     r7,r9,r10
1e60  59528186                        subi     r6,r10,r10
1e64  3c02a058                        cmpibl   0x0,r10,overflow
1e68  000010 08                       b        fill_er_up
1e68
1e6c                        pos_overflow_chk:
1e6c  59524007                        addo     r7,r9,r10
1e70  59528186                        subi     r6,r10,r10
1e74  3902a048                        cmpibg   0x0,r10,overflow
1e74
1e74
```

```
1e78  80403000 01019600   fill_er_up:   ldob    command_count,r8  /* load the command counter */
1e80  82416007                          stob    r8, 7 (r5)  /* store command count in frame */
1e80
1e84  80403000 01018700                 ldob    frame_count,r8  /* load the frame counter */
1e8c  82416008                          stob    r8, 8 (r5)  /* store frame count in frame */
1e90  59420801                          addo    0x01,r8,r8  /* inc and store frame count */
1e94  82403000 01018700                 stob    r8,frame_count
1e94
1e94
1e9c                      more_fill_er_up:
1e9c  80315000                          ldob    (r5),r6  /* read word from packet */
1ea0  92335c07                          st      r6, (r13) [r7 * 1]  /* store in tail */
1ea0
1ea4  5939c801                          addo    0x01,r7,r7  /* increment tail */
1ea8  583ac087                          and     r7,r11,r7  /* make -1 stay in ring */
1eac  82383000 01019404                 stob    r7,grid_tx_tail  /* store new tail value */
1eac
1eb4  59294801                          addo    0x01,r5,r5  /* increment packet pointer */
1eb8  342b1fe4                          cmpobl  r5,r12,more_fill_er_up  /* done with transfer */
1eb8
1eb8
1ebc                      overflow:       /* reset copy packet flag */
1ebc  8c283000 01017000                 ldconst flags,r5
1ec4  5930de1c                          ldconst 0x30000000,r6
1ec8  5c381e00                          ldconst 0x00000000,r7
1ecc  61398005                          atmod   r5,r6,r7  /* reset bits 28, 29 of flags */
1ecc
1ed0  fffcb0 08                         b       exec_top
1ed0
1ed0
1ed0                      /*************************************************************************/
1ed0                      /*  get esc data from the uart */
1ed0                      /*************************************************************************/
1ed0
1ed4  80303000 01019418   get_esc_data:    ldob    esc_rx_head,r6
1edc  80383000 0101941c                 ldob    esc_rx_tail,r7
1ee4  8c683000 01019300                 ldconst esc_rx_buffer,r13
1eec  8c4000ff                          ldconst 0x000000ff,r8
1eec
1ef0                      get_esc_data_top:
1ef0  80283000 04000000                 ldob    uarta_rbr,r5  /* get character */
1ef0
1ef8  822b5c07                          stob    r5,(r13) [r7 * 1]  /* store in buffer tail */
1ef8
1efc  5939c801                          addo    0x01,r7,r7  /* inc and store tail */
1f00  583a0087                          and     r7,r8,r7  /* make it wrap around */
1f04  82383000 0101941c                 stob    r7,esc_rx_tail
1f04
1f0c  3531c014                          cmpobne r6,r7,check_uarta_for_more
1f0c
1f10  59318801                          addo    0x01,r6,r6  /* dump the oldest info when overflow */
1f14  58320086                          and     r6,r8,r6  /* make it wrap around */
1f18  82303000 01019418                 stob    r6,esc_rx_head  /* inc and store head */
1f18
1f20                      check_uarta_for_more:
1f20  80283000 04000014                 ldob    uarta_lsr,r5
1f28  37017fc8                          bbs     0,r5,get_esc_data_top
1f28
1f28                         /* reset esc uart rx full flag */
1f2c  8c283000 01017000                 ldconst flags,r5
1f34  59305e1b                          ldconst 0x08000000,r6
1f38  5c381e00                          ldconst 0x0,r7
1f3c  61398005                          atmod   r5,r6,r7  /* reset bit 27 of flags */
1f3c
1f40  8403d000                          bx      (r15)
1f40
1f40
1f40                      /*************************************************************************/
1f40                      /*  send esc data to the uart */
1f40                      /*************************************************************************/
1f40
1f44  5c281e00            send_esc_data:   ldconst    0x0,r5
1f48  80303000 01019410                 ldob    esc_tx_head,r6
1f50  80383000 01019414                 ldob    esc_tx_tail,r7
1f58  3231c04c                          cmpobe  r6,r7,end_esc_send_flag  /* 0 to send, leave flag */
1f5c  8c683000 01019200                 ldconst esc_tx_buffer,r13
1f64  8c4000ff                          ldconst 0x000000ff,r8
1f64
```

```
1f68                            send_esc_data_top:
1f68  3231c028                          cmpobe  r6,r7,reset_esc_send_flag  /* no more data to send */
1f68
1f6c  804b5c06                          ldob    (r13) [r6 * 1],r9  /* read buffer */
1f70  82483000 04000000                 stob    r9,uarta_thr  /* store in uart */
1f70
1f78  59318801                          addo    0x01,r6,r6  /* increment buffer head */
1f7c  58320086                          and     r6,r8,r6  /* make it loop */
1f80  82303000 01019410                 stob    r6,esc_tx_head  /* store buffer head */
1f80
1f88  59294801                          addo    0x01,r5,r5
1f8c  31817fdc                          cmpobg  16,r5,send_esc_data_top
1f8c
1f8c
1f90                            reset_esc_send_flag:
1f90  8c283000 01017000                 ldconst flags,r5
1f98  59305e1a                          ldconst 0x04000000,r6
1f9c  5c381e00                          ldconst 0x0,r7
1fa0  61398005                          atmod   r5,r6,r7  /* reset bit 26 of flags */
1fa0
1fa4                            end_esc_send_flag:
1fa4  8403d000                          bx      (r15)
1fa4
1fa4
1fa4
1fa4                            /*************************************************************************/
1fa4                            /*  get grid data from the uart and check for commands */
1fa4                            /*************************************************************************/
1fa4
1fa8  80303000 01019408 get_grid_data:   ldob       grid_rx_head,r6
1fb0  80383000 0101940c                 ldob    grid_rx_tail,r7
1fb8  8c683000 01019100                 ldconst grid_rx_buffer,r13
1fc0  8c4000ff                          ldconst 0x000000ff,r8
1fc0
1fc4                            get_grid_data_top:
1fc4  80283000 04000020                 ldob    uartb_rbr,r5  /* get character */
1fc4
1fcc  822b5c07                          stob    r5,(r13) [r7 * 1]  /* store in buffer tail */
1fcc
1fd0  5939c801                          addo    0x01,r7,r7  /* inc and store tail */
1fd4  583a0087                          and     r7,r8,r7  /* make it wrap around */
1fd8  82383000 0101940c                 stob    r7,grid_rx_tail
1fd8
1fd8                                       /* check incomming data for commands */
1fd8
1fe0  90483000 01019500                 ld      command_buffer,r9  /* recall command buf */
1fe8  594a4e08                          shlo    8,r9,r9  /* make room for new char */
1fec  584a4385                          or      r5,r9,r9  /* add new char */
1ff0  92483000 01019500                 st      r9,command_buffer
1ff0
1ff0                                       /* check command buffer for valid commands */
1ff0
1ff8  8c503000 0e0f120d                 ldconst normal_code,r10  /* check for normal mode */
2000  324a80d0                          cmpobe  r9,r10,set_normal_mode
2000
2004  8c503000 18060512                 ldconst pass_through_code,r10  /* check for pass mode */
200c  324a8144                          cmpobe  r9,r10,set_pass_mode
200c
2010  8c503000 010e0607                 ldconst ack_flags_code,r10  /* check for ack flags cmd */
2018  324a81bc                          cmpobe  r9,r10,ack_flags
2018
201c  8c503000 130e1814                 ldconst set_clk_nxt_trig_code,r10  /* check for cmd */
2024  324a81f8                          cmpobe  r9,r10,set_clk_nxt_trig
2024
2028  8c503000 13030b09                 ldconst set_clk_now_code,r10  /* check for cmd */
2030  324a8270                          cmpobe  r9,r10,set_clocks_now
2030
2034  8c503000 040d020b                 ldconst dump_memory_code,r10  /* check for cmd */
203c  324a8308                          cmpobe  r9,r10,dump_memory
203c
2040  8c503000 0c0d1704                 ldconst load_memory_code,r10  /* check for cmd */
2048  324a833c                          cmpobe  r9,r10,load_memory
2048
204c  8c503000 150e0c0b                 ldconst unlock_memory_code,r10  /* check for cmd */
2054  324a8580                          cmpobe  r9,r10,unlock_memory
2054
2058  8c503000 0c0f0319                 ldconst lock_memory_code,r10  /* check for cmd */
2060  324a85c8                          cmpobe  r9,r10,lock_memory
2060
2064  8c503000 18090104                 ldconst execute_instruction_code,r10  /* check for cmd */
206c  324a835c                          cmpobe  r9,r10,execute_instruction
```

```
206c
206c
2070  90583000 01017004              ld      more_flags,r11  /* check for long command */
2078  3702e394                       bbs     0,r11,add_char_to_cmd_buffer
207c  370ae390                       bbs     1,r11,add_char_to_cmd_buffer
207c
2080                          check_grid_rx_overflow:
2080  80303000 01019408              ldob    grid_rx_head,r6
2088  80383000 0101940c              ldob    grid_rx_tail,r7
2090  3531c014                       cmpobne r6,r7,check_uartb_for_more
2090
2094  59318801                       addo    0x01,r6,r6  /* dump the oldest info when overflow */
2098  58320086                       and     r6,r8,r6  /* make it wrap around */
209c  82303000 01019408              stob    r6,grid_rx_head  /* inc and store head */
209c
20a4                          check_uartb_for_more:
20a4  80283000 04000034              ldob    uartb_lsr,r5
20ac  37017f18                       bbs     0,r5,get_grid_data_top
20ac
20ac                                 /* reset grid uart rx full flag */
20b0  8c283000 01017000              ldconst flags,r5
20b8  59305e19                       ldconst 0x02000000,r6
20bc  5c381e00                       ldconst 0x0,r7
20c0  61398005                       atmod   r5,r6,r7  /* reset bit 25 of flags */
20c0
20c4  90203000 01017000              ld      flags,r4  /* restore flags to r4 */
20c4
20cc  8403d000                       bx      (r15)
20cc
20cc
20cc                          /*****************************************************************************/
20cc                          /*  normal mode cmd */
20cc                          /*****************************************************************************/
20cc
20d0                          set_normal_mode:
20d0  8c503000 ff400000              ldconst 0xff400000,r10  /* mask */
20d8  59595e18                       ldconst 0x05000000,r11  /* value */
20dc  8c603000 01017000              ldconst flags,r12
20e4  615a800c                       atmod   r12,r10,r11  /* clear 31,30,29,28,22 of flags */
20e4                                                      /* re-initialize uart flags */
20e4
20e8  8c503000 0000ff63              ldconst 0x0000ff63,r10  /* mask */
20f0  5c581e00                       ldconst 0x00000000,r11  /* value */
20f4  8c603000 01017004              ldconst more_flags,r12
20fc  615a800c                       atmod   r12,r10,r11
20fc                                    /* clr bits 0,1,5,6,8,9,10,11,12,13,14,15 of more_flags */
20fc
20fc                                 /* reset receive buffer flags */
2100  5c501e00                       ldconst 0x00,r10
2104  82503000 01019408              stob    r10,grid_rx_head
210c  82503000 0101940c              stob    r10,grid_rx_tail
2114  82503000 01019418              stob    r10,esc_rx_head
211c  82503000 0101941c              stob    r10,esc_rx_tail
211c
211c                                 /* flush uart fifos */
2124  8c5000cf                       ldconst fcr, r10
2128  82503000 04000008              stob    r10, uarta_fcr
2130  82503000 04000028              stob    r10, uartb_fcr
2130
2130                                 /* increment command counter */
2138  80503000 01019600              ldob    command_count,r10
2140  59528801                       addo    0x01,r10,r10
2144  82503000 01019600              stob    r10,command_count
2144
214c  ffff34 08                      b       check_grid_rx_overflow
214c
214c
214c                          /*****************************************************************************/
214c                          /*  pass through mode cmd */
214c                          /*****************************************************************************/
214c
2150                          set_pass_mode:
2150  8c503000 ff400000              ldconst 0xff400000,r10  /* mask */
2158  8c583000 c5000000              ldconst 0xc5000000,r11  /* value */
2160  8c603000 01017000              ldconst flags,r12
2168  615a800c                       atmod   r12,r10,r11  /* set bits 30 & 31 of flags,
2168                                                         clear bits 22,28,29 of flags */
2168                                                      /* re-initialize uart flags */
2168
216c  8c503000 0000ff63              ldconst 0x0000ff63,r10  /* mask */
2174  5c581e00                       ldconst 0x00000000,r11  /* value */
```

```
2178  8c603000 01017004                ldconst more_flags,r12
2180  615a800c                          atmod   r12,r10,r11
2180                                      /* clr bits 0,1,5,6,8,9,10,11,12,13,14,15 of more_flags */
2180
2180                                    /* reset receive buffer flags */
2184  5c501e00                          ldconst 0x00,r10
2188  82503000 01019408                 stob    r10,grid_rx_head
2190  82503000 0101940c                 stob    r10,grid_rx_tail
2198  82503000 01019418                 stob    r10,esc_rx_head
21a0  82503000 0101941c                 stob    r10,esc_rx_tail
21a0
21a0                                    /* flush uart fifos */
21a8  8c5000cf                          ldconst fcr, r10
21ac  82503000 04000008                 stob    r10, uarta_fcr
21b4  82503000 04000028                 stob    r10, uartb_fcr
21b4
21b4                                    /* increment command counter */
21bc  80503000 01019600                 ldob    command_count,r10
21c4  59528801                          addo    0x01,r10,r10
21c8  82503000 01019600                 stob    r10,command_count
21c8
21d0  fffeb0 08                         b       check_grid_rx_overflow
21d0
21d0
21d0                                  /*************************************************************************/
21d0                                  /*  acknowledge flags cmd */
21d0                                  /*************************************************************************/
21d0
21d4                                  ack_flags:      /* clear flags in any mode */
21d4  8c503000 00edf800                 ldconst 0x00edf800,r10  /* mask */
21dc  5c581e00                          ldconst 0x00000000,r11  /* value */
21e0  8c603000 01017000                 ldconst flags,r12
21e8  615a800c                          atmod   r12,r10,r11
21e8                                      /* clear 11,12,13,14,15,16,18,19,21,22,23 */
21e8
21ec  8c503000 ffffff1f                 ldconst 0xffffff1f,r10  /* mask */
21f4  5c581e00                          ldconst 0x00000000,r11  /* value */
21f8  8c603000 01017004                 ldconst more_flags,r12
2200  615a800c                          atmod   r12,r10,r11
2200                                      /* clear 0,1,2,3,4,8,9,10,11,12,13,14,15,16,17,18,19,20,
2200                                            21,22,23,24,25,26,27,28,29,30,31 of more_flags */
2200
2200                                    /* increment command counter */
2204  80503000 01019600                 ldob    command_count,r10
220c  59528801                          addo    0x01,r10,r10
2210  82503000 01019600                 stob    r10,command_count
2210
2218  fffe68 08                         b       check_grid_rx_overflow
2218
2218
2218                                  /*************************************************************************/
2218                                  /* set clock on next trigger command */
2218                                  /*************************************************************************/
2218
221c                                  set_clk_nxt_trig:  /* only perform command in packet mode */
221c  90503000 01017000                 ld      flags,r10
2224  37f2be5c                          bbs     30,r10,check_grid_rx_overflow
2228  37fabe58                          bbs     31,r10,check_grid_rx_overflow
2228
2228                                    /* check for bad setting data */
222c  5c501e09                          ldconst 10 - 1,r10  /* 10 words to check */
2230  8c583000 ff000000                 ldconst 0xff000000,r11
2238                                  set_clk_nxt_trig_loop:
2238  9060390a 01018f00                 ld      desired_time [ r10 * 4 ],r12  /* load time value */
2240  585b008b                          and     r11,r12,r11  /* check msb byte */
2244  59528901                          subo    0x01,r10,r10
2248  3402bff0                          cmpobl  0,r10,set_clk_nxt_trig_loop  /* accumulate msbs */
2248
224c  8c603000 ff000000                 ldconst 0xff000000,r12  /* check for all msbs good */
2254  585b030b                          xor     r11,r12,r11
2258  3502fe28                          cmpobne 0,r11,check_grid_rx_overflow  /* some msb is bad */
2258
2258                                    /* set up flags */
225c  59505e16                          ldconst 0x00400000,r10  /* mask */
2260  59585e16                          ldconst 0x00400000,r11  /* value */
2264  8c603000 01017000                 ldconst flags,r12
226c  615a800c                          atmod   r12,r10,r11  /* set bit 22 of flags */
226c
2270  8c503000 0000ff03                 ldconst 0x0000ff03,r10  /* mask */
2278  5958de08                          ldconst 0x00000300,r11  /* value */
227c  8c603000 01017004                 ldconst more_flags,r12
```

```
2284  615a800c                          atmod   r12,r10,r11  /* set bits 8,9 of more_flags */
2284                                      /* clear bits 0,1,10,11,12,13,14,15 of more_flags */
2284
2284                                    /* increment command counter */
2288  80503000 01019600                 ldob    command_count,r10
2290  59528801                          addo    0x01,r10,r10
2294  82503000 01019600                 stob    r10,command_count
2294
229c  fffde4 08                         b       check_grid_rx_overflow
229c
229c
229c                              /*********************************************************************/
229c                              /* set clocks now command */
229c                              /*********************************************************************/
229c
22a0                              set_clocks_now:  /* only perform in packet mode */
22a0  90503000 01017000                 ld      flags,r10
22a8  37f2bdd8                          bbs     30,r10,check_grid_rx_overflow
22ac  37fabdd4                          bbs     31,r10,check_grid_rx_overflow
22ac
22ac                                    /* check for bad setting data */
22b0  5c501e09                          ldconst 10 - 1,r10  /* 10 words to check */
22b4  8c583000 ff000000                 ldconst 0xff000000,r11
22bc                              set_clocks_now_loop:
22bc  9060390a 01018f00                 ld      desired_time [ r10 * 4 ],r12  /* load time value */
22c4  585b008b                          and     r11,r12,r11  /* check msb byte */
22c8  59528901                          subo    0x01,r10,r10
22cc  3402bff0                          cmpobl  0,r10,set_clocks_now_loop  /* accumulate msbs */
22cc
22d0  8c603000 ff000000                 ldconst 0xff000000,r12  /* check for all msbs good */
22d8  585b030b                          xor     r11,r12,r11
22dc  3502fda4                          cmpobne 0,r11,check_grid_rx_overflow  /* some msb is bad */
22dc
22dc                                    /* boost priority for executive time setting proceedure */
22e0  5957de10                          ldconst 0x001f0000,r10  /* priority mask */
22e4  595c5e10                          ldconst 0x00110000,r11  /* priority of 16 */
22e8  655a828a                          modpc   r10,r10,r11  /* change process priority to 16 */
22e8
22e8                                    /* set up flags */
22ec  59505e16                          ldconst 0x00400000,r10  /* mask */
22f0  5c581e00                          ldconst 0x00000000,r11  /* value */
22f4  8c603000 01017000                 ldconst flags,r12
22fc  615a800c                          atmod   r12,r10,r11  /* clear bit 22 of flags */
22fc
2300  8c503000 0000ff03                 ldconst 0x0000ff03,r10  /* mask */
2308  5c581e00                          ldconst 0x00000000,r11  /* value */
230c  8c603000 01017004                 ldconst more_flags,r12
2314  615a800c                          atmod   r12,r10,r11
2314                                      /* clear bits 0,1,8,9,10,11,12,13,14,15 of more_flags */
2314
2318  85703000 000034d4                 balx    set_clocks,r14
2318
2318                                    /* restore priority for executive */
2320  5957de10                          ldconst 0x001f0000,r10  /* priority mask */
2324  5958de10                          ldconst 0x00030000,r11  /* priority of 3 */
2328  655a828a                          modpc   r10,r10,r11  /* change process priority to 3 */
2328
2328                                    /* increment command counter */
232c  80503000 01019600                 ldob    command_count,r10
2334  59528801                          addo    0x01,r10,r10
2338  82503000 01019600                 stob    r10,command_count
2338
2340  fffd40 08                         b       check_grid_rx_overflow
2340
2340
2340                              /*********************************************************************/
2340                              /* dump memory command */
2340                              /*********************************************************************/
2340
2344                              dump_memory:    /* only perform in packet mode */
2344  90503000 01017000                 ld      flags,r10
234c  37f2bd34                          bbs     30,r10,check_grid_rx_overflow
2350  37fabd30                          bbs     31,r10,check_grid_rx_overflow
2350
2354  59505e16                          ldconst 0x00400000,r10  /* mask */
2358  5c581e00                          ldconst 0x00000000,r11  /* value */
235c  8c603000 01017000                 ldconst flags,r12
2364  615a800c                          atmod   r12,r10,r11  /* clear bit 22 of flags */
2364
2368  8c503000 0000ff03                 ldconst 0x0000ff03,r10  /* mask */
2370  8c580c03                          ldconst 0x00000c03,r11  /* value */
```

```
2374  8c603000 01017004                ldconst more_flags,r12
237c  615a800c                          atmod   r12,r10,r11  /* set bits 0,1,10,11 of more_flags */
237c                                      /* clr bits 8,9,12,13,14,15 of more_flags */
237c
2380  fffd00 08                         b       check_grid_rx_overflow
2380
2380
2380                          /*************************************************************************/
2380                          /* load memory command */
2380                          /*************************************************************************/
2380
2384                          load_memory:    /* only perform in packet mode */
2384  90503000 01017000                ld      flags,r10
238c  37f2bcf4                          bbs     30,r10,check_grid_rx_overflow
2390  37fabcf0                          bbs     31,r10,check_grid_rx_overflow
2390
2394  59505e16                          ldconst 0x00400000,r10  /* mask */
2398  5c581e00                          ldconst 0x00000000,r11  /* value */
239c  8c603000 01017000                ldconst flags,r12
23a4  615a800c                          atmod   r12,r10,r11  /* clear bit 22 of flags */
23a4
23a8  8c503000 0000ff03                ldconst 0x0000ff03,r10  /* mask */
23b0  8c583000 00003003                ldconst 0x00003003,r11  /* value */
23b8  8c603000 01017004                ldconst more_flags,r12
23c0  615a800c                          atmod   r12,r10,r11  /* set bits 0,1,12,13 of more_flags */
23c0                                      /* clr bits 8,9,10,11,14,15 of more_flags */
23c0
23c4  fffcbc 08                         b       check_grid_rx_overflow
23c4
23c4
23c4                          /*************************************************************************/
23c4                          /* execute instruction command */
23c4                          /*************************************************************************/
23c4
23c8                          execute_instruction:
23c8                                    /* only perform in packet mode */
23c8  90503000 01017000                ld      flags,r10
23d0  37f2bcb0                          bbs     30,r10,check_grid_rx_overflow
23d4  37fabcac                          bbs     31,r10,check_grid_rx_overflow
23d4
23d8  59505e16                          ldconst 0x00400000,r10  /* mask */
23dc  5c581e00                          ldconst 0x00000000,r11  /* value */
23e0  8c603000 01017000                ldconst flags,r12
23e8  615a800c                          atmod   r12,r10,r11  /* clear bit 22 of flags */
23e8
23ec  8c503000 0000ff03                ldconst 0x0000ff03,r10  /* mask */
23f4  8c583000 0000c003                ldconst 0x0000c003,r11  /* value */
23fc  8c603000 01017004                ldconst more_flags,r12
2404  615a800c                          atmod   r12,r10,r11  /* set bits 0,1,14,15 of more_flags */
2404                                      /* clr bits 8,9,10,11,12,13 of more_flags */
2404
2408  fffc78 08                         b       check_grid_rx_overflow
2408
2408
2408                          /*************************************************************************/
2408                          /*  add char to command buffer */
2408                          /*************************************************************************/
2408
240c                          add_char_to_cmd_buffer:
240c                                    /* store new character in command buffer */
240c  80483000 01019604                ldob    long_command_character_count,r9
2414  82283809 01019504                stob    r5,command_buffer + 4 [r9 * 1]
241c  594a4801                          addo    0x01,r9,r9
2420  82483000 01019604                stob    r9,long_command_character_count
2420
2428  31427c58                          cmpobg  8,r9,check_grid_rx_overflow  /* last character ? */
2428                                      /* long commands are 8 characters long */
2428
2428                                    /* have the whole command, reset character count */
242c  5c481e00                          ldconst 0x0,r9
2430  82483000 01019604                stob    r9,long_command_character_count
2430
2438  90283000 01017004                ld      more_flags,r5
2440  375160a4                          bbs     10,r5,dump_memory_locations
2444  375960a0                          bbs     11,r5,dump_memory_locations
2448  37716034                          bbs     14,r5,execute_instruction_now
244c  37796030                          bbs     15,r5,execute_instruction_now
2450  30616154                          bbc     12,r5,reset_long_command_flags
2454  30696150                          bbc     13,r5,reset_long_command_flags
2454
2454
```

```
2458                            load_memory_locations:   /* check for memory unlock */
2458  90283000 01017000                 ld      flags,r5
2460  30b96144                          bbc     23,r5,reset_long_command_flags  /* chk for unlock */
2460
2464  90283000 01019504                 ld      command_buffer + 4,r5  /* location to modify */
246c  90303000 01019508                 ld      command_buffer + 8,r6  /* value to put there */
2474  92315000                          st      r6,(r5)
2474
2478  000118 08                         b       reset_long_command_stuff
2478
2478
247c                            execute_instruction_now:  /* check for valid address, and hap unlock */
247c  90283000 01017000                 ld      flags,r5
2484  30b96120                          bbc     23,r5,reset_long_command_flags  /* chk for unlock */
2484
2488  90283000 01019504                 ld      command_buffer + 4,r5  /* jump address */
2490  90303000 01019508                 ld      command_buffer + 8,r6  /* redundant jump address */
2498  58398305                          xor     r5,r6,r7  /* check for differences */
249c  3501e108                          cmpobne 0,r7,reset_long_command_flags
249c
249c                                    /* good command */
249c                                    /* increment command counter */
24a0  80503000 01019600                 ldob    command_count,r10
24a8  59528801                          addo    0x01,r10,r10
24ac  82503000 01019600                 stob    r10,command_count
24ac
24b4  59505e16                          ldconst 0x00400000,r10  /* mask */
24b8  5c581e00                          ldconst 0x00000000,r11  /* value */
24bc  8c603000 01017000                 ldconst flags,r12
24c4  615a800c                          atmod   r12,r10,r11  /* clear bit 22 of flags */
24c4
24c8  8c503000 0000ff03                 ldconst 0x0000ff03,r10  /* mask */
24d0  5c581e00                          ldconst 0x00000000,r11  /* value */
24d4  8c603000 01017004                 ldconst more_flags,r12
24dc  615a800c                          atmod   r12,r10,r11
24dc                                      /* clear bits 0,1,8,9,10,11,12,13,14,15 of more_flags */
24dc
24e0  84015000                          bx      (r5)  /* jump to new instruction address */
24e0
24e0
24e4                            dump_memory_locations:  /* make memory packet */
24e4  8c283000 5fa0f0a5                 ldconst frame_sync,r5
24ec  92283000 01018200                 st      r5,memory_packet /* store frame sync */
24ec
24ec                                    /* set memory frame flags */
24f4  5950de05                          ldconst 0x00000060,r10  /* mask */
24f8  5958de05                          ldconst 0x00000060,r11  /* value */
24fc  8c603000 01017004                 ldconst more_flags,r12
2504  615a800c                          atmod   r12,r10,r11  /* set bits 5,6 of more_flags */
2504
2504                                    /* store flag in packet, let exec add frame count */
2508  90283000 01017000                 ld      flags,r5
2510  8c303000 ffffff07                 ldconst 0xffffff07,r6
2518  58298085                          and     r5,r6,r5  /* take out intr enable bits */
251c  58301506                          not     r6,r6
2520  90383000 01017004                 ld      more_flags,r7
2528  5839c086                          and     r6,r7,r7  /* leave packet bits */
252c  5829c385                          or      r5,r7,r5  /* combine to get packet status */
252c
2530  82283000 01018206                 stob    r5,memory_packet + 6  /* store flags */
2538  59294c08                          shro    8,r5,r5
253c  82283000 01018205                 stob    r5,memory_packet + 5
2544  59294c08                          shro    8,r5,r5
2548  82283000 01018204                 stob    r5,memory_packet + 4
2548
2548
2548                                    /* force the dump time to be 0 */
2548
2550  5c401e00                          ldconst 0x0,r8
2554  5c481e00                          ldconst 0x0,r9
2558                            move_dump_time:
2558  82483808 01018209                 stob    r9,memory_packet + 9 [r8 * 1]
2560  59420801                          addo    0x01,r8,r8
2564  314a3ff4                          cmpobg  9,r8,move_dump_time
2564
2564
2564                                    /* store dump address into frame */
2568  90283000 01019504                 ld      command_buffer + 4,r5
2570  92283000 01018212                 st      r5,memory_packet + 18
2570
2570                                    /* store dump data into frame */
```

```
2578  5c381e00                  ldconst 0,r7
257c  90315d07            get_dump_data:  ld    (r5) [ r7 * 4 ],r6
2580  92303907 01018216       st      r6,memory_packet + 22 [ r7 * 4 ]
2588  5939c801                addo    0x01,r7,r7
258c  3139fff0                cmpobg  7,r7,get_dump_data
258c
258c
2590                      reset_long_command_stuff:
2590                            /* increment command counter */
2590  80503000 01019600         ldob    command_count,r10
2598  59528801                  addo    0x01,r10,r10
259c  82503000 01019600         stob    r10,command_count
259c
25a4                      reset_long_command_flags:
25a4  59505e16                  ldconst 0x00400000,r10  /* mask */
25a8  5c581e00                  ldconst 0x00000000,r11  /* value */
25ac  8c603000 01017000         ldconst flags,r12
25b4  615a800c                  atmod   r12,r10,r11  /* clear bit 22 of flags */
25b4
25b8  8c503000 0000ff03         ldconst 0x0000ff03,r10  /* mask */
25c0  5c581e00                  ldconst 0x00000000,r11  /* value */
25c4  8c603000 01017004         ldconst more_flags,r12
25cc  615a800c                  atmod   r12,r10,r11
25cc                              /* clr bits 0,1,8,9,10,11,12,13,14,15 of more_flags */
25cc
25d0  fffab0 08                 b       check_grid_rx_overflow
25d0
25d0
25d0                      /*************************************************************************/
25d0                      /* unlock memory command */
25d0                      /*************************************************************************/
25d0
25d4                      unlock_memory:  /* only perform in packet mode */
25d4  90503000 01017000         ld      flags,r10
25dc  37f2baa4                  bbs     30,r10,check_grid_rx_overflow
25e0  37fabaa0                  bbs     31,r10,check_grid_rx_overflow
25e0
25e4  5950de16                  ldconst 0x00c00000,r10  /* mask */
25e8  59585e17                  ldconst 0x00800000,r11  /* value */
25ec  8c603000 01017000         ldconst flags,r12
25f4  615a800c                  atmod   r12,r10,r11  /* clear bit 22 of flags */
25f4                                                 /* set bit 23 of flags */
25f4
25f8  8c503000 0000ff03         ldconst 0x0000ff03,r10  /* mask */
2600  5c581e00                  ldconst 0x00000000,r11  /* value */
2604  8c603000 01017004         ldconst more_flags,r12
260c  615a800c                  atmod   r12,r10,r11
260c                              /* clr bits 0,1,8,9,10,11,12,13,14,15 of more_flags */
260c
260c                            /* increment command counter */
2610  80503000 01019600         ldob    command_count,r10
2618  59528801                  addo    0x01,r10,r10
261c  82503000 01019600         stob    r10,command_count
261c
2624  fffa5c 08                 b       check_grid_rx_overflow
2624
2624
2624                      /*************************************************************************/
2624                      /* lock memory command */
2624                      /*************************************************************************/
2624
2628                      lock_memory:  /* only perform in packet mode */
2628  90503000 01017000         ld      flags,r10
2630  37f2ba50                  bbs     30,r10,check_grid_rx_overflow
2634  37faba4c                  bbs     31,r10,check_grid_rx_overflow
2634
2638  5950de16                  ldconst 0x00c00000,r10  /* mask */
263c  5c581e00                  ldconst 0x00000000,r11  /* value */
2640  8c603000 01017000         ldconst flags,r12
2648  615a800c                  atmod   r12,r10,r11  /* clear bit 22,23 of flags */
2648
264c  8c503000 0000ff03         ldconst 0x0000ff03,r10  /* mask */
2654  5c581e00                  ldconst 0x00000000,r11  /* value */
2658  8c603000 01017004         ldconst more_flags,r12
2660  615a800c                  atmod   r12,r10,r11
2660                              /* clr bits 0,1,8,9,10,11,12,13,14,15 of more_flags */
2660
2660                            /* increment command counter */
2664  80503000 01019600         ldob    command_count,r10
266c  59528801                  addo    0x01,r10,r10
2670  82503000 01019600         stob    r10,command_count
```

```
2670
2678  fffa08 08                       b       check_grid_rx_overflow
2678
2678
2678                          /*************************************************************************/
2678                          /*  send grid data to the uart */
2678                          /*************************************************************************/
2678
267c  5c281e00                send_grid_data:  ldconst   0x0,r5
2680  80303000 01019400                ldob    grid_tx_head,r6
2688  80383000 01019404                ldob    grid_tx_tail,r7
2690  3231c04c                         cmpobe  r6,r7,end_send_grid_data  /* 0 to send, leave flg */
2690
2694  8c683000 01019000                ldconst grid_tx_buffer,r13
269c  8c4000ff                         ldconst 0x000000ff,r8
269c
26a0                          send_grid_data_top:
26a0  3231c028                         cmpobe  r6,r7,reset_grid_send_flag  /* nothing to send */
26a0
26a4  804b5c06                         ldob    (r13) [r6 * 1],r9  /* read buffer */
26a8  82483000 04000020                stob    r9,uartb_thr  /* store in uart */
26a8
26b0  59318801                         addo    0x01,r6,r6  /* increment buffer head */
26b4  58320086                         and     r6,r8,r6  /* make it loop */
26b8  82303000 01019400                stob    r6,grid_tx_head  /* store buffer head */
26b8
26c0  59294801                         addo    0x01,r5,r5
26c4  31817fdc                         cmpobg  16,r5,send_grid_data_top
26c4
26c4
26c8                          reset_grid_send_flag:
26c8  8c283000 01017000                ldconst flags,r5
26d0  59305e18                         ldconst 0x01000000,r6
26d4  5c381e00                         ldconst 0x0,r7
26d8  61398005                         atmod   r5,r6,r7  /* reset bit 24 of flags */
26d8
26dc                          end_send_grid_data:
26dc  8403d000                         bx      (r15)
26dc
26dc
26dc                          #line 1 "hap.src:watchdog.src"
26dc                          /*************************************************************************/
26dc                          /*    WATCHDOG.SRC                                                       */
26dc                          /*    Watchdog interrupt hanling routine                                 */
26dc                          /*                                                                       */
26dc                          /*    9/10/91  r.h.                                                      */
26dc                          /*    1/6/92  store flags and more flags is g0,g1                        */
26dc                          /*    1/13/92 make this a warmstart                                      */
26dc                          /*    1/16/92 add clock timer initialization                             */
26dc                          /*                                                                       */
26dc                          /*************************************************************************/
26dc
26e0                                .globl flags
26e0                                .globl prom_tst
26e0                                .globl hap_ier
26e0                                .globl watchdog_intr
26e0
26e0  5c801e00                watchdog_intr:  ldconst   0x0,g0
26e4  92803000 05000000                st              g0,hap_ier  /* disable all interrupts */
26e4
26e4                                    /* set watchdog expired flag */
26ec  8c403000 01017000                 ldconst         flags,r8
26f4  59485e13                          ldconst         0x00080000,r9
26f8  59505e13                          ldconst         0x00080000,r10
26fc  61524008                          atmod           r8,r9,r10
26fc
26fc                                    /* initialize clock timers */
2700  8c203000 0070ffa4                 ldconst         init_timer_0_lsb,r4  /* timer 0 lsb count */
2708  8c403000 00ffffff                 ldconst         0x00ffffff,r8  /* all bits read/write */
2710  92203000 0300003c                 st              r4,clock_timer0_lsb
2718  90303000 0300003c                 ld              clock_timer0_lsb,r6
2720  58318088                          and             r8,r6,r6
2724  352180c0                          cmpobne         r4,r6,watchdog_clk_init_err0
2724
2728  8c203000 00fdff00                 ldconst         init_timer_0_msb,r4  /* timer 0 msb count */
2730  92203000 03000040                 st              r4,clock_timer0_msb
2738  90303000 03000040                 ld              clock_timer0_msb,r6
2740  58318088                          and             r8,r6,r6
2744  352180a0                          cmpobne         r4,r6,watchdog_clk_init_err0
2744
2748  8c203000 00171708                 ldconst         0x00171708,r4  /* interrupt routing reg */
```

```
2750  8c403000 00bfbfbf                ldconst         0x00bfbfbf,r8  /* bit 6 rd only, disable */
2758  92203000 03000010                st              r4,clock_irr  /* timer 0 to intr, fast pf */
2760  90303000 03000010                ld              clock_irr,r6
2768  58318088                         and             r8,r6,r6
276c  35218078                         cmpobne         r4,r6,watchdog_clk_init_err0
276c
2770  8c203000 007f7f7f                ldconst         0x007f7f7f,r4 /* select reg set 1 */
2778  92203000 03000000                st              r4,clock_msr
2778
2780  8c203000 00bcb878                ldconst         0x00bcb878,r4  /* output mode register */
2788  8c403000 00ffffff                ldconst         0x00ffffff,r8  /* all bits read/write */
2790  92203000 03000008                st              r4,clock_omr  /* mfo,*int1,int2 */
2798  90303000 03000008                ld              clock_omr,r6
27a0  58318088                         and             r8,r6,r6
27a4  35218040                         cmpobne         r4,r6,watchdog_clk_init_err0
27a4
27a8  8c203000 00404040                ldconst         0x00404040,r4  /* interrupt control reg0 */
27b0  92203000 0300000c                st              r4,clock_icr0  /* timer 0 enable */
27b8  90303000 0300000c                ld              clock_icr0,r6
27c0  58318088                         and             r8,r6,r6
27c4  35218020                         cmpobne         r4,r6,watchdog_clk_init_err0
27c4
27c8  8c200000                         lda             0x00000000,r4  /* interrupt control reg1 */
27cc  92203000 03000010                st              r4,clock_icr1  /* pfw enable */
27d4  90303000 03000010                ld              clock_icr1,r6
27dc  58318088                         and             r8,r6,r6
27e0  32218084                         cmpobe          r4,r6,watchdog_clk_init_end
27e0
27e4                           watchdog_clk_init_err0:
27e4  8c4000ff                         ldconst         0x000000ff,r8
27e8  58298088                         and             r8,r6,r5
27ec  58390088                         and             r8,r4,r7
27f0  3229c01c                         cmpobe          r5,r7,watchdog_clk_init_err1
27f4  58840988                         setbit          8,g0,g0
27f8  90403000 01017000                ld              flags,r8  /* set flags */
2800  58420988                         setbit          8,r8,r8
2804  92403000 01017000                st              r8,flags
280c                           watchdog_clk_init_err1:
280c  8c403000 0000ff00                ldconst         0x0000ff00,r8
2814  58298088                         and             r8,r6,r5
2818  58390088                         and             r8,r4,r7
281c  3229c01c                         cmpobe          r5,r7,watchdog_clk_init_err2
2820  58840989                         setbit          9,g0,g0
2824  90403000 01017000                ld              flags,r8  /* set flags */
282c  58420989                         setbit          9,r8,r8
2830  92403000 01017000                st              r8,flags
2838                           watchdog_clk_init_err2:
2838  8c403000 00ff0000                ldconst         0x00ff0000,r8
2840  58298088                         and             r8,r6,r5
2844  58390088                         and             r8,r4,r7
2848  3229c01c                         cmpobe          r5,r7,watchdog_clk_init_end
284c  5884098a                         setbit          10,g0,g0
2850  90403000 01017000                ld              flags,r8  /* set flags */
2858  5842098a                         setbit          10,r8,r8
285c  92403000 01017000                st              r8,flags
285c
2864                           watchdog_clk_init_end:
2864  8c203000 003f3f3f                ldconst         0x003f3f3f,r4 /* page 0 reg set 0 */
286c  92203000 03000000                st              r4,clock_msr
286c
286c
286c                                   /* modify the process controls to a priority of 3 */
2874  5988de10                         ldconst 0x00030000,g1
2878  5997de10                         ldconst 0x001f0000,g2  /* load mask for priority */
287c  658c8292                         modpc   g2,g2,g1  /* update process controls */
287c
287c
287c                                   /* goto the warmstart routine */
2880  0012ac 08                        b       warmstart
2880
2880
2880                                   /* not used */
2884                           watchdog_return:
2884  0a000000                         ret
2884
2884
```

```
2884                    #line 1 "hap.src:trig_int.src"
2884                    /*****************************************************************************/
2884                    /*    TRIG_INT.SRC                                                           */
2884                    /*    Trigger interrupt hanling routine                                      */
2884                    /*                                                                           */
2884                    /*    9/10/91  r.h.                                                          */
2884                    /*    10/3/91 added packet stuff                                             */
2884                    /*    3/5/92 modify flag gathering                                           */
2884                    /*    3/12/92 force attitude frame flags to attitude frames                  */
2884                    /*****************************************************************************/
2884
2888                            .globl clock_msr
2888                            .globl clock_irr
2888                          .globl trigger_intr
2888
2888                    /*****************************************************************************/
2888                    /* read majority vote and fix the trigger time */
2888                    /*****************************************************************************/
2888                            .globl get_check_fix_time
2888
2888                    trigger_intr:
2888  90203000 01017004                 ld      more_flags,r4  /* check for set time function */
2890  30412038                          bbc     8,r4,get_trigger_time
2894  30492034                          bbc     9,r4,get_trigger_time
2894
2898  85703000 000034d4                 balx    set_clocks,r14
2898
28a0  59505e16                          ldconst 0x00400000,r10  /* mask */
28a4  5c581e00                          ldconst 0x00000000,r11  /* value */
28a8  8c603000 01017000                 ldconst flags,r12
28b0  615a800c                          atmod   r12,r10,r11  /* clear bit 22 of flags */
28b0
28b4  5950de08                          ldconst 0x00000300,r10  /* mask */
28b8  5c581e00                          ldconst 0x00000000,r11  /* value */
28bc  8c603000 01017004                 ldconst more_flags,r12
28c4  615a800c                          atmod   r12,r10,r11
28c4                                      /* clear bits 8,9 of more_flags */
28c4
28c4
28c8                    get_trigger_time:
28c8  8c703000 01018900                 ldconst trigger_time,r14
28d0  85783000 000031f4                 balx    get_check_fix_time,r15
28d0
28d0
28d0                    /*****************************************************************************/
28d0                    /* set the trigger flags */
28d0                    /*****************************************************************************/
28d8                            .globl flags
28d8
28d8                    set_trigger_flags:
28d8  8c403000 01017000                 ldconst flags,r8
28e0  5948de0f                          ldconst 0x00018000,r9  /* mask */
28e4  5950de0f                          ldconst 0x00018000,r10  /* value */
28e8  61524008                          atmod   r8,r9,r10  /* set bits 15 & 16 of flags */
28e8
28ec  8c403000 01017004                 ldconst more_flags,r8
28f4  5948de05                          ldconst 0x00000060,r9  /* mask */
28f8  5c501e00                          ldconst 0x00000000,r10  /* value */
28fc  61524008                          atmod   r8,r9,r10  /* clear bits 5 & 6 of more_flags */
28fc
28fc
28fc                    /*****************************************************************************/
28fc                    /* put together the trigger packet */
28fc                    /*****************************************************************************/
2900                          .globl trigger_packet
2900                          .globl frame_count
2900                          .globl frame_sync
2900                          .globl _current_solution
2900
2900                    make_trigger_packet:
2900  8c203000 5fa0f0a5                ldconst frame_sync,r4
2908  92203000 01018100                 st      r4,trigger_packet /* store frame sync */
2908
2908
2908                                    /* store flags, let exec add frame count */
2910  90283000 01017000                 ld      flags,r5
2918  8c303000 ffffff07                 ldconst 0xffffff07,r6
2920  58298085                          and     r5,r6,r5  /* take out intr enable bits */
2924  58301506                          not     r6,r6
2928  90383000 01017004                 ld      more_flags,r7
2930  5839c086                          and     r6,r7,r7  /* leave packet bits */
```

```
2934  5829c385                        or      r5,r7,r5  /* combine to get packet status */
2934
2938  82283000 01018106               stob    r5,trigger_packet + 6  /* store flags */
2940  59294c08                        shro    8,r5,r5
2944  82283000 01018105               stob    r5,trigger_packet + 5
294c  59294c08                        shro    8,r5,r5
2950  82283000 01018104               stob    r5,trigger_packet + 4
2950
2950
2958  5c401e00                        ldconst 0x0,r8
295c                          move_trigger_time:
295c  80483808 01018900               ldob    trigger_time [r8 * 1],r9
2964  82483808 01018109               stob    r9,trigger_packet + 9 [r8 * 1]
296c  59420801                        addo    0x01,r8,r8
2970  31423fec                        cmpobg  8,r8,move_trigger_time
2970
2974  5c481e00                        ldconst 0x0,r9  /* 1/10000s always 0 for now */
2978  82483000 01018111               stob    r9,trigger_packet + 17
2978
2978                                  /* load current solution */
2980  b0403000 0101a500               ldq     _current_solution,r8
2980                                  /* store q0, q1 */
2988  b2403000 01018112               stq     r8,trigger_packet + 18
2988
2990  b0403000 0101a510               ldq     _current_solution + 16,r8
2990                                  /* store q2, q3 */
2998  b2403000 01018122               stq     r8,trigger_packet + 34
2998
2998
29a0  0a000000                        ret
29a0
29a0
29a0                          #line 1 "hap.src:iae_int.src"
29a0                          /*****************************************************************************/
29a0                          /*    IAE_INT.SRC                                                            */
29a0                          /*    IAE interrupt hanling routine                                          */
29a0                          /*                                                                           */
29a0                          /*    9/10/91  r.h.                                                          */
29a0                          /*    10/15/91 add iae data processing                                       */
29a0                          /*    11/11/91 slow the rate to 2 per second                                 */
29a0                          /*    12/31/91 fix the iae interrupt count check                             */
29a0                          /*    4/5/92 change flag gathering and fix label useage                      */
29a0                          /*****************************************************************************/
29a0
29a4                          iae_intr:
29a4
29a4                          #line 1 "hap.src:iae_int.src:iae.src"
29a4                          #     FE version           : 1.20
29a4                          #     BE version           : V3.0
29a4                          #     Time of compilation  : Tue Mar 17 20:09:51 1992
29a4                          #     Command line                 :
                                    C:\INTEL960\BIN\IC960.EXE /AMC /S /O2 /v /Z iae10.lst iae10.c
29a4                          #     .ident     "ic960 V3.0 DOSX",0x29c6c29c
29a4                          #     .file      "IAE10.C"
29a4                                .text
29a4                          #     .align     4
29a4                                .globl     _main
29a4                          _main:
29a4                          #     .def       _main; .val _main; .scl 2; .type 0x44; .endef
29a4  8c086140                      lda        320(sp),sp
29a4                          #     stq        g8,64(fp)
29a4                          #     st         g12,80(fp)
29a4                          #     .file      "IAE10.C"
29a8  88583000 02000040             ldos       _iae_a1,r11
29b0  8c403000 00007fff             lda        32767,r8
29b8  8c483000 ffff8001             lda        -32767,r9
29c0  88183000 02000044             ldos       _iae_a2,r3
29c8  925fe054                      st         r11,0x54(fp)
29cc  921fe058                      st         r3,0x54+4(fp)
29d0  8c7800ff                      lda        255,r15
29d4  88183000 02000048             ldos       _iae_a3,r3
29dc  5c681e00                      mov        0,r13
29e0  58751805                      notbit     5,20,r14
29e4  921fe05c                      st         r3,0x54+8(fp)
29e8                          .I1:
29e8  901ffd0d 00000054             ld         0x54(fp)[r13*4],r3
29f0  9058390d 0101a000             ld         _pin[r13*4],r11
29f8  59e8c18b                      subi       r11,r3,g13
29fc  5a02209d                      cmpi       g13,r8
2a00  92effd0d 00000060             st         g13,0x60(fp)[r13*4]
2a08  000028 16                     ble        .I4
```

```
2a0c  90effd0d 00000054          ld        0x54(fp)[r13*4],g13
2a14  9050390d 0101a000          ld        _pin[r13*4],r10
2a1c  58181810                   notbit    T6,0,r3
2a20  595f418a                   subi      r10,g13,r11
2a24  5952c183                   subi      r3,r11,r10
2a28  9257fd0d 00000060          st        r10,0x60(fp)[r13*4]
2a30                       .I4:
2a30  905ffd0d 00000060          ld        0x60(fp)[r13*4],r11
2a38  3b5a4028                   cmpibge   r11,r9,.I6
2a3c  9057fd0d 00000054          ld        0x54(fp)[r13*4],r10
2a44  9058390d 0101a000          ld        _pin[r13*4],r11
2a4c  58e81810                   notbit    T6,0,g13
2a50  591a818b                   subi      r11,r10,r3
2a54  5918c09d                   addi      g13,r3,r3
2a58  921ffd0d 00000060          st        r3,0x60(fp)[r13*4]
2a60                       .I6:
2a60  901ffd0d 00000054          ld        0x54(fp)[r13*4],r3
2a68  7063808d                   mulo      r13,r14,r12
2a6c  9218390d 0101a000          st        r3,_pin[r13*4]
2a74  90eb3400 0101a114          ld        _a+20(r12),g13
2a7c  905b3400 0101a124          ld        _a+36(r12),r11
2a84  901b3400 0101a128          ld        _a+40(r12),r3
2a8c  595ac09d                   addi      g13,r11,r11
2a90  591ac083                   addi      r3,r11,r3
2a94  67183203                   cvtir     r3,fp3
2a98  6d801c83                   movrl     fp3,g0
2a9c  9a87fd8d 00000070          stl       g0,0x70(fp)[r13*8]
2aa4  90eb3400 0101a124          ld        _a+36(r12),g13
2aac  92eb3400 0101a128          st        g13,_a+40(r12)
2ab4  90533400 0101a114          ld        _a+20(r12),r10
2abc  92533400 0101a124          st        r10,_a+36(r12)
2ac4  905b3400 0101a108          ld        _a+8(r12),r11
2acc  901b3400 0101a118          ld        _a+24(r12),r3
2ad4  90533400 0101a11c          ld        _a+28(r12),r10
2adc  5918c08b                   addi      r11,r3,r3
2ae0  59e8c08a                   addi      r10,r3,g13
2ae4  92eb3400 0101a114          st        g13,_a+20(r12)
2aec  90533400 0101a118          ld        _a+24(r12),r10
2af4  92533400 0101a11c          st        r10,_a+28(r12)
2afc  905b3400 0101a108          ld        _a+8(r12),r11
2b04  925b3400 0101a118          st        r11,_a+24(r12)
2b0c  901b3400 0101a100          ld        _a(r12),r3
2b14  90eb3400 0101a10c          ld        _a+12(r12),g13
2b1c  905b3400 0101a110          ld        _a+16(r12),r11
2b24  59ef4083                   addi      r3,g13,g13
2b28  5957408b                   addi      r11,g13,r10
2b2c  92533400 0101a108          st        r10,_a+8(r12)
2b34  905b3400 0101a10c          ld        _a+12(r12),r11
2b3c  925b3400 0101a110          st        r11,_a+16(r12)
2b44  901b3400 0101a100          ld        _a(r12),r3
2b4c  921b3400 0101a10c          st        r3,_a+12(r12)
2b54  90effd0d 00000060          ld        0x60(fp)[r13*4],g13
2b5c  90533400 0101a104          ld        _a+4(r12),r10
2b64  901ffd0d 00000060          ld        0x60(fp)[r13*4],r3
2b6c  5952809d                   addi      g13,r10,r10
2b70  92533400 0101a100          st        r10,_a(r12)
2b78  921b3400 0101a104          st        r3,_a+4(r12)
2b80  9887fd8d 00000070          ldl       0x70(fp)[r13*8],g0
2b88  98a0398d 0101a200          ldl       _pfda[r13*8],g4
2b90  98e03000 000040d0          ldl       _gcrc,g12
2b98  98c7fd8d 00000070          ldl       0x70(fp)[r13*8],g8
2ba0  79840694                   subrl     g4,g0,g0
2ba4  9a87fd8d 00000090          stl       g0,0x90(fp)[r13*8]
2bac  9887fd8d 00000090          ldl       0x90(fp)[r13*8],g0
2bb4  98a7fd8d 00000070          ldl       0x70(fp)[r13*8],g4
2bbc  7984061c                   mulrl     g12,g0,g0
2bc0  79840798                   addrl     g8,g0,g0
2bc4  9a87fd8d 000000b0          stl       g0,0xb0(fp)[r13*8]
2bcc  9aa0398d 0101a200          stl       g4,_pfda[r13*8]
2bd4  5958508d                   addi      r13,1,r11
2bd8  586ac08f                   and       r15,r11,r13
2bdc  5a036882                   cmpi      2,r13
2be0  fffe08 13                  bge       .I1
2be4  9867e0b0                   ldl       0xb0(fp),r12
2be8  98403000 000040b0          ldl       _b,r8
2bf0  8ca03000 55555555          lda       1431655765,g4
2bf8  8ca83000 3fb55555          lda       1068848469,g5
2c00  79430688                   subrl     r8,r12,r8
2c04  9a47e0b0                   stl       r8,0xb0(fp)
2c08  9847e0b8                   ldl       0xb0+8(fp),r8
2c0c  98603000 000040b8          ldl       _b+8,r12
```

```
2c14  8c800000                lda     0,g0
2c18  8c883000 3fe00000       lda     1071644672,g1
2c20  7962068c                subrl   r12,r8,r12
2c24  9a67e0b8                stl     r12,0xb0+8(fp)
2c28  9867e0c0                ldl     0xb0+16(fp),r12
2c2c  98403000 000040c0       ldl     _b+16,r8
2c34  79b30688                subrl   r8,r12,g6
2c38  9ab7e0c0                stl     g6,0xb0+16(fp)
2c3c  9867e0b0                ldl     0xb0(fp),r12
2c40  98e7e0b8                ldl     0xb0+8(fp),g12
2c44  98403000 00004060       ldl     _mg,r8
2c4c  98c03000 00004068       ldl     _mg+8,g8
2c54  98d03000 00004070       ldl     _mg+16,g10
2c5c  79430608                mulrl   r8,r12,r8
2c60  79670618                mulrl   g8,g12,r12
2c64  79c5861a                mulrl   g10,g6,g8
2c68  79430788                addrl   r8,r12,r8
2c6c  79460788                addrl   r8,g8,r8
2c70  9a47e0d0                stl     r8,0xd0(fp)
2c74  9847e0b0                ldl     0xb0(fp),r8
2c78  98e7e0b8                ldl     0xb0+8(fp),g12
2c7c  98603000 00004078       ldl     _mg+24,r12
2c84  98c03000 00004080       ldl     _mg+32,g8
2c8c  98b7e0c0                ldl     0xb0+16(fp),g6
2c90  98d03000 00004088       ldl     _mg+40,g10
2c98  7962060c                mulrl   r12,r8,r12
2c9c  79470618                mulrl   g8,g12,r8
2ca0  79c5861a                mulrl   g10,g6,g8
2ca4  7962078c                addrl   r12,r8,r12
2ca8  7966078c                addrl   r12,g8,r12
2cac  9a67e0d8                stl     r12,0xd0+8(fp)
2cb0  9867e0b0                ldl     0xb0(fp),r12
2cb4  98e7e0b8                ldl     0xb0+8(fp),g12
2cb8  98403000 00004090       ldl     _mg+48,r8
2cc0  98c03000 00004098       ldl     _mg+56,g8
2cc8  98b7e0c0                ldl     0xb0+16(fp),g6
2ccc  98d03000 000040a0       ldl     _mg+64,g10
2cd4  79630608                mulrl   r8,r12,r12
2cd8  79470618                mulrl   g8,g12,r8
2cdc  79c5861a                mulrl   g10,g6,g8
2ce0  7942078c                addrl   r12,r8,r8
2ce4  79660788                addrl   r8,g8,r12
2ce8  9a67e0e0                stl     r12,0xd0+16(fp)
2cec  98e7e0d8                ldl     0xd0+8(fp),g12
2cf0  98403000 0101a308       ldl     _pb+8,r8
2cf8  98c03000 0101a310       ldl     _pb+16,g8
2d00  79430608                mulrl   r8,r12,r8
2d04  79670618                mulrl   g8,g12,r12
2d08  7962068c                subrl   r12,r8,r12
2d0c  98c7e0d0                ldl     0xd0(fp),g8
2d10  79430614                mulrl   g4,r12,r8
2d14  79420798                addrl   g8,r8,r8
2d18  9a47e0f0                stl     r8,0xf0(fp)
2d1c  98e7e0d0                ldl     0xd0(fp),g12
2d20  9847e0e0                ldl     0xd0+16(fp),r8
2d24  98603000 0101a300       ldl     _pb,r12
2d2c  98c03000 0101a310       ldl     _pb+16,g8
2d34  7962060c                mulrl   r12,r8,r12
2d38  79470618                mulrl   g8,g12,r8
2d3c  7962068c                subrl   r12,r8,r12
2d40  98c7e0d8                ldl     0xd0+8(fp),g8
2d44  79430614                mulrl   g4,r12,r8
2d48  79620798                addrl   g8,r8,r12
2d4c  9a67e0f8                stl     r12,0xf0+8(fp)
2d50  9867e0d8                ldl     0xd0+8(fp),r12
2d54  98e7e0d0                ldl     0xd0(fp),g12
2d58  98403000 0101a300       ldl     _pb,r8
2d60  98c03000 0101a308       ldl     _pb+8,g8
2d68  79430608                mulrl   r8,r12,r8
2d6c  79670618                mulrl   g8,g12,r12
2d70  7962068c                subrl   r12,r8,r12
2d74  9847e0e0                ldl     0xd0+16(fp),r8
2d78  79630614                mulrl   g4,r12,r12
2d7c  79430788                addrl   r8,r12,r8
2d80  9a47e100                stl     r8,0xf0+16(fp)
2d84  9867e0d0                ldl     0xd0(fp),r12
2d88  9a603000 0101a300       stl     r12,_pb
2d90  9847e0d8                ldl     0xd0+8(fp),r8
2d94  9a403000 0101a308       stl     r8,_pb+8
2d9c  9867e0e0                ldl     0xd0+16(fp),r12
2da0  9a603000 0101a310       stl     r12,_pb+16
```

```
2da8  9847e0f0                  ldl     0xf0(fp),r8
2dac  79620610                  mulrl   g0,r8,r12
2db0  9a67e158                  stl     r12,0x150+8(fp)
2db4  9847e0f8                  ldl     0xf0+8(fp),r8
2db8  79620610                  mulrl   g0,r8,r12
2dbc  9a67e160                  stl     r12,0x150+16(fp)
2dc0  9847e100                  ldl     0xf0+16(fp),r8
2dc4  79620610                  mulrl   g0,r8,r12
2dc8  9a67e168                  stl     r12,0x150+24(fp)
2dcc  9867e158                  ldl     0x150+8(fp),r12
2dd0  98403000 0101a408         ldl     _tho+8,r8
2dd8  7942078c                  addrl   r12,r8,r8
2ddc  79620610                  mulrl   g0,r8,r12
2de0  9a67e138                  stl     r12,0x130+8(fp)
2de4  9847e160                  ldl     0x150+16(fp),r8
2de8  98603000 0101a410         ldl     _tho+16,r12
2df0  79430788                  addrl   r8,r12,r8
2df4  79620610                  mulrl   g0,r8,r12
2df8  9a67e140                  stl     r12,0x130+16(fp)
2dfc  9867e168                  ldl     0x150+24(fp),r12
2e00  98403000 0101a418         ldl     _tho+24,r8
2e08  7942078c                  addrl   r12,r8,r8
2e0c  79a20610                  mulrl   g0,r8,g4
2e10  9aa7e148                  stl     g4,0x130+24(fp)
2e14  9867e138                  ldl     0x130+8(fp),r12
2e18  9887e140                  ldl     0x130+16(fp),g0
2e1c  7943060c                  mulrl   r12,r12,r8
2e20  79640610                  mulrl   g0,g0,r12
2e24  79850614                  mulrl   g4,g4,g0
2e28  79430788                  addrl   r8,r12,r8
2e2c  79440788                  addrl   r8,g0,r8
2e30  69043288                  cmprl   r8,0f0.0
2e34  00000c 13                 bge     .A100000
2e34                      #     callj     __thread_ptr   #returns addr ERRNO in g0
2e34                      #     ldconst   33,r13
2e34                      #     st        r13,(g0)       #errno = EDOM
2e38  6d601c90                  movrl   0f0.0,r12
2e3c  000008 08                 b       .A100001
2e40                      .A100000:
2e40  69601408                  sqrtrl  r8,r12          # inline expansion of sqrt
2e44                      .A100001:
2e44  6904328c                  cmprl   r12,0
2e48  9a67e130                  stl     r12,0x130(fp)
2e4c  000018 15                 bne     .I8
2e50  8ca00000                  lda     0,g4
2e54  8ca83000 3ff00000         lda     1072693248,g5
2e5c  5d801614                  movl    g4,g0
2e60  000018 08                 b       .I9
2e64                      .I8:
2e64  6980168c                  cosrl   r12,g0          # inline expansion of cos
2e68  9867e130                  ldl     0x130(fp),r12
2e6c  6940160c                  sinrl   r12,r8          # inline expansion of sin
2e70  9867e130                  ldl     0x130(fp),r12
2e74  79a2058c                  divrl   r12,r8,g4
2e78                      .I9:
2e78  9847e138                  ldl     0x130+8(fp),r8
2e7c  98c7e140                  ldl     0x130+16(fp),g8
2e80  98603000 0101a508         ldl     _current_solution+8,r12
2e88  98e03000 0101a510         ldl     _current_solution+16,g12
2e90  98d7e148                  ldl     0x130+24(fp),g10
2e94  98b03000 0101a518         ldl     _current_solution+24,g6
2e9c  79430608                  mulrl   r8,r12,r8
2ea0  79670618                  mulrl   g8,g12,r12
2ea4  79c5861a                  mulrl   g10,g6,g8
2ea8  79430788                  addrl   r8,r12,r8
2eac  98603000 0101a500         ldl     _current_solution,r12
2eb4  79460788                  addrl   r8,g8,r8
2eb8  7964060c                  mulrl   r12,g0,r12
2ebc  79420614                  mulrl   g4,r8,r8
2ec0  79630688                  subrl   r8,r12,r12
2ec4  9a67e110                  stl     r12,0x110(fp)
2ec8  9867e138                  ldl     0x130+8(fp),r12
2ecc  98c7e148                  ldl     0x130+24(fp),g8
2ed0  98403000 0101a500         ldl     _current_solution,r8
2ed8  98e03000 0101a510         ldl     _current_solution+16,g12
2ee0  98d7e140                  ldl     0x130+16(fp),g10
2ee4  98b03000 0101a518         ldl     _current_solution+24,g6
2eec  7942060c                  mulrl   r12,r8,r8
2ef0  79670618                  mulrl   g8,g12,r12
2ef4  79c5861a                  mulrl   g10,g6,g8
2ef8  79630788                  addrl   r8,r12,r12
```

```
2efc  98403000 0101a508          ldl     _current_solution+8,r8
2f04  79630698                   subrl   g8,r12,r12
2f08  79440608                   mulrl   r8,g0,r8
2f0c  79630614                   mulrl   g4,r12,r12
2f10  79430788                   addrl   r8,r12,r8
2f14  9a47e118                   stl     r8,0x110+8(fp)
2f18  9867e140                   ldl     0x130+16(fp),r12
2f1c  98c7e148                   ldl     0x130+24(fp),g8
2f20  98403000 0101a500          ldl     _current_solution,r8
2f28  98e03000 0101a508          ldl     _current_solution+8,g12
2f30  98d7e138                   ldl     0x130+8(fp),g10
2f34  98b03000 0101a518          ldl     _current_solution+24,g6
2f3c  7942060c                   mulrl   r12,r8,r8
2f40  79670618                   mulrl   g8,g12,r12
2f44  79c5861a                   mulrl   g10,g6,g8
2f48  7942068c                   subrl   r12,r8,r8
2f4c  98603000 0101a510          ldl     _current_solution+16,r12
2f54  79460788                   addrl   r8,g8,r8
2f58  7964060c                   mulrl   r12,g0,r12
2f5c  79420614                   mulrl   g4,r8,r8
2f60  7962078c                   addrl   r12,r8,r12
2f64  9a67e120                   stl     r12,0x110+16(fp)
2f68  9847e148                   ldl     0x130+24(fp),r8
2f6c  98c7e140                   ldl     0x130+16(fp),g8
2f70  98603000 0101a500          ldl     _current_solution,r12
2f78  98e03000 0101a508          ldl     _current_solution+8,g12
2f80  98d7e138                   ldl     0x130+8(fp),g10
2f84  98b03000 0101a510          ldl     _current_solution+16,g6
2f8c  79630608                   mulrl   r8,r12,r12
2f90  79470618                   mulrl   g8,g12,r8
2f94  79c5861a                   mulrl   g10,g6,g8
2f98  7962078c                   addrl   r12,r8,r12
2f9c  98403000 0101a518          ldl     _current_solution+24,r8
2fa4  79c30698                   subrl   g8,r12,g8
2fa8  79640608                   mulrl   r8,g0,r12
2fac  79460614                   mulrl   g4,g8,r8
2fb0  7942078c                   addrl   r12,r8,r8
2fb4  9a47e128                   stl     r8,0x110+24(fp)
2fb8  9847e110                   ldl     0x110(fp),r8
2fbc  9a403000 0101a500          stl     r8,_current_solution
2fc4  9847e118                   ldl     0x110+8(fp),r8
2fc8  9a403000 0101a508          stl     r8,_current_solution+8
2fd0  9847e120                   ldl     0x110+16(fp),r8
2fd4  9a403000 0101a510          stl     r8,_current_solution+16
2fdc  9847e128                   ldl     0x110+24(fp),r8
2fe0  9a403000 0101a518          stl     r8,_current_solution+24
2fe8  9847e158                   ldl     0x150+8(fp),r8
2fec  9a403000 0101a408          stl     r8,_tho+8
2ff4  9847e160                   ldl     0x150+16(fp),r8
2ff8  9a403000 0101a410          stl     r8,_tho+16
3000  9847e168                   ldl     0x150+24(fp),r8
3004  9a403000 0101a418          stl     r8,_tho+24
300c  88583000 0101a600          ldos    _solution_count,r11
3014  591ac881                   addi    1,r11,r3
3018  8a183000 0101a600          stos    r3,_solution_count
3018                             ldq     64(fp),g8
3018                       #     ld      80(fp),g12
3018                       #     ret
3018                       #     .def    _main; .val .; .scl -1; .endef
405d                       .data
405d                             .align  4
4060                             .globl  _mg
4060                       _mg:
4060  cea72d35 3e7ac617          .word   0xcea72d35,0x3e7ac617
4068  00000000 00000000          .word   0x00000000,0x00000000
4070  00000000 00000000          .word   0x00000000,0x00000000
4078  cf36a2c1 bddf72cb          .word   0xcf36a2c1,0xbddf72cb
4080  78ee3ef8 3e7ad25c          .word   0x78ee3ef8,0x3e7ad25c
4088  00000000 00000000          .word   0x00000000,0x00000000
4090  16c61f85 bdc5d3a3          .word   0x16c61f85,0xbdc5d3a3
4098  4c851e1a bdbf7abe          .word   0x4c851e1a,0xbdbf7abe
40a0  ecf9f622 3e7ac764          .word   0xecf9f622,0x3e7ac764
40a8                             .space  8
40b0                             .align  4
40b0                             .globl  _b
40b0                       _b:
40b0  bb0ccb80 bef421f7          .word   0xbb0ccb80,0xbef421f7
40b8  43fcd8c3 3f5d7312          .word   0x43fcd8c3,0x3f5d7312
40c0  07c949e7 3f2fe5b2          .word   0x07c949e7,0x3f2fe5b2
40c8                             .space  8
40d0                             .align  3
```

```
40d0                            .globl    _gcrc
40d0                    _gcrc:
40d0  014727dd 3feee0a2         .word     0x014727dd,0x3feee0a2
3020                    .text
3020
3020                              /* reset watchdog timers */
3020  8c203000 003f3f3f          ldconst 0x003f3f3f,r4
3028  92203000 03000000          st      r4,clock_msr  /* select pg 0 reg sel 0 */
3028
3030  8c203000 008f8f8f          ldconst 0x008f8f8f,r4  /* timer 0 control reg */
3038  92203000 03000004          st      r4,clock_timer0_cntl  /* mode 3, osc, start count */
3038
3038                              /* check IAE interrupt count */
3040  90203000 03000000          ld      clock_msr,r4  /* load interrupt counter value */
3048  59210e04                    shlo    4,r4,r4  /* get the 4 bits in the lsb */
304c  59210c1c                    shro    28,r4,r4
3050  80283000 0101a604          ldob    iae_interrupt_count,r5  /* load expected value */
3058  59294801                    addo    0x01,r5,r5
305c  5829488f                    and     0x0f,r5,r5  /* only look at the 4 lsb bits */
3060  82203000 0101a604          stob    r4,iae_interrupt_count  /* store current count */
3068  32214018                    cmpobe  r4,r5,iae_inc_solution_count
3068
3068                              /* set lost track of IMU flag */
306c  8c403000 01017000          ldconst flags,r8
3074  59485e15                    ldconst 0x00200000,r9
3078  59505e15                    ldconst 0x00200000,r10
307c  61524008                    atmod   r8,r9,r10  /* read modify write bit 21 of flags */
307c
3080                    iae_inc_solution_count:
3080  90203000 0101a600          ld      _solution_count,r4
3088  8c2803e8                    ldconst output_rate,r5
308c  33214008                    cmpobge r4,r5,make_normal_packet
308c
308c
3090  0a000000                    ret
3090
3090
3090
3090                    /*************************************************************************/
3090                    /* rst soln count, set copy packet flag, load flags into normal packet  */
3090                    /*************************************************************************/
3090
3094                          .globl flags
3094                          .globl solution_time
3094                         .globl get_check_fix_time
3094                          .globl normal_packet
3094                          .globl frame_sync
3094
3094                    make_normal_packet:
3094                              /* reset solution counter */
3094  5c201e00                    ldconst 0x00,r4
3098  92203000 0101a600          st      r4,_solution_count
3098
3098                              /* set copy packet flag */
30a0  8c403000 01017000          ldconst flags,r8
30a8  5948de1c                    ldconst 0x30000000,r9
30ac  5950de1c                    ldconst 0x30000000,r10
30b0  61524008                    atmod   r8,r9,r10  /* set bits 28 & 29 of flags */
30b0
30b0                              /* store flag in packet, let exec add frame count */
30b4  90283000 01017000          ld      flags,r5
30bc  8c303000 ffffff07          ldconst 0xffffff07,r6
30c4  58298085                    and     r5,r6,r5  /* take out intr enable bits */
30c8  58301506                    not     r6,r6
30cc  90383000 01017004          ld      more_flags,r7
30d4  5839c086                    and     r6,r7,r7  /* leave packet bits */
30d8  5829c385                    or      r5,r7,r5  /* combine to get packet status */
30d8
30dc  82283000 01018006          stob    r5,normal_packet + 6  /* store flags */
30e4  59294c08                    shro    8,r5,r5
30e8  82283000 01018005          stob    r5,normal_packet + 5
30f0  59294c08                    shro    8,r5,r5
30f4  82283000 01018004          stob    r5,normal_packet + 4
30f4
30f4
```

```
30f4                    /*************************************************************************/
30f4                    /* read majority vote and fix the solution time */
30f4                    /*************************************************************************/
30fc                            .globl get_check_fix_time
30fc
30fc                    solution_time_vote:
30fc  8c703000 01018800                 ldconst solution_time,r14
3104  85783000 000031f4                 balx    get_check_fix_time,r15
3104
3104
3104                    /*************************************************************************/
3104                    /* put together the rest of the normal packet */
3104                    /*************************************************************************/
310c                          .globl normal_packet
310c                          .globl frame_count
310c                          .globl frame_sync
310c                          .globl _current_solution
310c
310c  8c203000 5fa0f0a5                 ldconst frame_sync,r4
3114  92203000 01018000                 st      r4,normal_packet /* store frame sync */
3114
3114
311c  5c401e00                          ldconst 0x0,r8
3120                    move_solution_time:
3120  80483808 01018800                 ldob    solution_time [r8 * 1],r9
3128  82483808 01018009                 stob    r9,normal_packet + 9 [r8 * 1]
3130  59420801                          addo    0x01,r8,r8
3134  31423fec                          cmpobg  8,r8,move_solution_time
3134
3138  5c481e00                          ldconst 0x0,r9  /* 1/10000s always zero for now */
313c  82483000 01018011                 stob    r9,normal_packet + 17
313c
313c                                    /* load current solution */
3144  b0403000 0101a500                 ldq     _current_solution,r8
3144                                    /* store q0, q1 */
314c  b2403000 01018012                 stq     r8,normal_packet + 18
314c
3154  b0403000 0101a510                 ldq     _current_solution + 16,r8
3154                                    /* store q2, q3 */
315c  b2403000 01018022                 stq     r8,normal_packet + 34
315c
315c
315c                    /*************************************************************************/
315c                    /* make the heartbeat blink */
315c                    /*************************************************************************/
3164  90203000 01017000                 ld      flags,r4  /* load flags */
316c  58210e00                          clrbit  0,r4,r4  /* mask off heartbeat bit */
3170  80283000 0101a608                 ldob    heartbeat,r5  /* load heartbeat toggle */
3178  58294881                          and     0x01,r5,r5  /* mask off heartbeat bit */
317c  58294384                          or      r4,r5,r5  /* determine new ier word */
3180  82283000 05000000                 stob    r5,hap_ier  /* store new heartbeat */
3188  59294801                          addo    0x01,r5,r5  /* inc and store heartbeat counter */
318c  82283000 0101a608                 stob    r5,heartbeat
318c
318c
3194  0a000000                          ret
3194
3194
3194                    #line 1 "hap.src:ser_int.src"
3194                    /*************************************************************************/
3194                    /*    SER_INT.SRC                                                        */
3194                    /*    Serial interrupt hanling routine                                   */
3194                    /*                                                                       */
3194                    /*    9/10/91  r.h.                                                      */
3194                    /*    9/12/91 changed uart flag positions                                */
3194                    /*                                                                       */
3194                    /*************************************************************************/
3194
3198                          .globl flags
3198                          .globl uarta_iir
3198                          .globl uartb_iir
3198                          .globl serial_intr
3198
3198  80203000 04000008   serial_intr:    ldob           uarta_iir,r4
31a0  8c403000 01017000                 ldconst         flags,r8
31a0
31a8  30112010                          bbc             2,r4,other_a_int
31a8                                    /* set uart a rx full flag */
31ac  59485e1b                          ldconst         0x08000000,r9
31b0  59505e1b                          ldconst         0x08000000,r10
31b4  61524008                          atmod           r8,r9,r10
```

```
31b4
31b8  30092010             other_a_int:     bbc             1,r4,uartb_check
31b8                                  /* set uart a tx empty flag */
31bc  59485e1a                        ldconst         0x04000000,r9
31c0  59505e1a                        ldconst         0x04000000,r10
31c4  61524008                        atmod           r8,r9,r10
31c4
31c8  80203000 04000028    uartb_check:     ldob            uartb_iir,r4
31d0  30112010                        bbc             2,r4,other_b_int
31d0                                  /* set uart b rx full flag */
31d4  59485e19                        ldconst         0x02000000,r9
31d8  59505e19                        ldconst         0x02000000,r10
31dc  61524008                        atmod           r8,r9,r10
31dc
31e0  30092010             other_b_int:     bbc             1,r4,ser_int_return
31e0                                  /* set uart b tx empty flag */
31e4  59485e18                        ldconst         0x01000000,r9
31e8  59505e18                        ldconst         0x01000000,r10
31ec  61524008                        atmod           r8,r9,r10
31ec
31f0  0a000000             ser_int_return:   ret
31f0
31f0                                  /* uart interrupt ident register interrupt codes
31f0                                  0001 - no interrupt
31f0                                  0110 - receiver line status (disabled for hap)
31f0                                  0100 - received data available
31f0                                  1100 - character timeout indication
31f0                                  0010 - transmitter holding register empty
31f0                                  0000 - modem status (disabled for hap) */
31f0
31f0
31f0                                  #line 1 "hap.src:do_time.src"
31f0                                  /*****************************************************************************/
31f0                                  /*    DO_TIME.SRC                                                            */
31f0                                  /*    Routine to read, vote, and fix time.  Call using                       */
31f0                                  /*        the bal or balx instruction.                                       */
31f0                                  /*    The place that time is stored in memory is provided in reg r14         */
31f0                                  /*                                                                           */
31f0                                  /*    13 words store the uncorrected time, corrected time                    */
31f0                                  /*        is left in the packet format starting at address (r14)    */
31f0                                  /*                                                                           */
31f0                                  /*    9/10/91  r.h.                                                          */
31f0                                  /*   10/4/91 fix up register useage                                          */
31f0                                  /*    1/8/92 fix clock 0 bad problem, switch year msb lsb stuff              */
31f0                                  /*****************************************************************************/
31f0
31f4                                        .globl flags
31f4                                           .globl clock_msr
31f4                                           .globl clock_irr
31f4                                           .globl clock_pfr
31f4                                           .globl clock_ram_1E
31f4                                           .globl clock_years
31f4                                           .globl clock_date_msb
31f4                                           .globl clock_date_lsb
31f4                                           .globl clock_hours
31f4                                           .globl clock_minutes
31f4                                           .globl clock_seconds
31f4                                           .globl clock_hundredths
31f4                                        .globl get_check_fix_time
31f4
31f4
31f4                                  /*****************************************************************************/
31f4                                  /* read the time and store */
31f4                                  /*****************************************************************************/
31f4
31f4                                  get_check_fix_time:
31f4  8c203000 003f3f3f                       ldconst 0x003f3f3f,r4
31fc  92203000 03000000                       st      r4,clock_msr  /* select page 0 register 0 */
3204  5c201e00                        ldconst 0x0,r4  /* pass counter */
3204
3208  59210801             read_time:       addo    0x01,r4,r4
3208                                  /* read periodic flag reg to chk for hundredths rollover */
320c  90283000 0300000c               ld      clock_pfr,r5
320c
3214  90303000 0300002c               ld      clock_years,r6
321c  92339000                        st      r6,(r14)
321c
3220  90303000 03000034               ld      clock_date_msb,r6
3228  9233a004                        st      r6, 1*4 (r14)
3228
```

```
322c  90303000 03000030                ld      clock_date_lsb,r6
3234  9233a008                         st      r6, 2*4 (rT4)
3234
3238  90303000 03000020                ld      clock_hours,r6
3240  9233a00c                         st      r6, 3*4 (r14)
3240
3244  90303000 0300001c                ld      clock_minutes,r6
324c  9233a010                         st      r6, 4*4 (r14)
324c
3250  90303000 03000018                ld      clock_seconds,r6
3258  9233a014                         st      r6, 5*4 (r14)
3258
325c  90303000 03000014                ld      clock_hundredths,r6
3264  9233a018                         st      r6, 6*4 (r14)
3264
3264
3264                                   /* read periodic flag reg and check for rollover */
3268  90283000 0300000c                ld      clock_pfr,r5
3270  34812014                         cmpobl  16,r4,set_rollover_flag  /* only try n times */
3274  37297f94                         bbs     5,r5,read_time  /* check clk0 for rollover */
3278  37697f90                         bbs     13,r5,read_time  /* check clk1 for rollover */
327c  37a97f8c                         bbs     21,r5,read_time  /* check clk2 for rollover */
3280  000018 08                        b       ckfx_years
3280
3284                             set_rollover_flag:
3284  8c503000 01017004                ldconst more_flags,r10  /* set more than n rollover flg */
328c  5c581e04                         ldconst 0x00000004,r11
3290  5c601e04                         ldconst 0x00000004,r12
3294  6162c00a                         atmod   r10,r11,r12
3294
3294
3294                             /*************************************************************************/
3294                             /* check and fix time */
3294                             /*************************************************************************/
3294
3298  8c383000 003f3f3f          ckfx_years:   ldconst 0x003f3f3f,r7  /* select page 0 reg set 0 */
32a0  92383000 03000000                st      r7,clock_msr
32a0
32a8  80239000                         ldob    (r14),r4  /* read clock 0 data */
32ac  802ba001                         ldob    1 (r14),r5  /* read clock 1 data */
32b0  8033a002                         ldob    2 (r14),r6  /* read clock 2 data */
32b0
32b0                                   /* vote and identify error */
32b4  85683000 00003404                balx    vote,r13
32b4
32bc  32f9e00c                         cmpobe  0x1f,r7,store_year_lsb
32bc                                   /* fix years */
32c0  92383000 0300002c                st      r7,clock_years  /* add rollover detection */
32c0
32c8  8223a001             store_year_lsb:  stob  r4, 1 (r14)
32c8
32c8
32cc  8c383000 00ffffff    ckfx_year_base:  ldconst  0x00ffffff,r7  /* get year base, nvram */
32d4  92383000 03000000                st      r7,clock_msr  /* select page 1 */
32d4
32dc  90383000 03000078                ld      clock_ram_1E,r7  /* load year base */
32dc
32e4  8c6800ff                         ldconst 0xff,r13  /* isolate clock 0 data */
32e8  58234087                         and     r7,r13,r4
32ec  5939cc08                         shro    8,r7,r7  /* isolate clock 1 data */
32f0  582b4087                         and     r7,r13,r5
32f4  5939cc08                         shro    8,r7,r7  /* isolate clock 2 data */
32f8  58334087                         and     r7,r13,r6
32f8
32f8                                   /* vote and identify errors */
32fc  85683000 00003404                balx    vote,r13
32fc
3304  32f9e00c                         cmpobe  0x1f,r7,store_year_msb
3304
3304                                   /* fix year base */
3308  92383000 03000078                st      r7,clock_ram_1E
3308
3310  82239000                 store_year_msb:  stob  r4,(r14)
3310
3314  8c383000 003f3f3f                ldconst 0x003f3f3f,r7  /* select page 0 reg set 0 */
331c  92383000 03000000                st      r7,clock_msr
331c
331c
```

```
331c                    /*************************************************************************/
3324  8023a004          ckfx_date_msb:  ldob  1*4 (r14),r4  /* read clock 0 data */
3328  802ba005                          ldob    1*4 + 1 (r14),r5  /* read clock 1 data */
332c  8033a006                          ldob    1*4 + 2 (r14),r6  /* read clock 2 data */
332c
332c                                    /* vote and identify error */
3330  85683000 00003404                balx    vote,r13
3330
3338  32f9e00c                          cmpobe  0x1f,r7,store_date_msb
3338                                    /* fix date msb */
333c  92383000 03000034                st      r7,clock_date_msb  /* add rollover detection */
333c
3344  8223a002          store_date_msb:  stob  r4, 2 (r14)
3344
3344
3344                    /*************************************************************************/
3348  8023a008          ckfx_date_lsb:  ldob  2*4 (r14),r4  /* read clock 0 data */
334c  802ba009                          ldob    2*4 + 1 (r14),r5  /* read clock 1 data */
3350  8033a00a                          ldob    2*4 + 2 (r14),r6  /* read clock 2 data */
3350
3350                                    /* vote and identify error */
3354  85683000 00003404                balx    vote,r13
3354
335c  32f9e00c                          cmpobe  0x1f,r7,store_date_lsb
335c                                    /* fix date lsb */
3360  92383000 03000030                st      r7,clock_date_lsb  /* add rollover detection */
3360
3368  8223a003          store_date_lsb:  stob  r4, 3 (r14)
3368
3368
3368                    /*************************************************************************/
336c  8023a00c          ckfx_hour:      ldob    3*4 (r14),r4  /* read clock 0 data */
3370  802ba00d                          ldob    3*4 + 1 (r14),r5  /* read clock 1 data */
3374  8033a00e                          ldob    3*4 + 2 (r14),r6  /* read clock 2 data */
3374
3374                                    /* vote and identify error */
3378  85683000 00003404                balx    vote,r13
3378
3380  32f9e00c                          cmpobe  0x1f,r7,store_hour
3380                                    /* fix hour */
3384  92383000 03000020                st      r7,clock_hours  /* add rollover detection */
3384
338c  8223a004          store_hour:     stob    r4, 4 (r14)
338c
338c
338c                    /*************************************************************************/
3390  8023a010          ckfx_minute:    ldob    4*4 (r14),r4  /* read clock 0 data */
3394  802ba011                          ldob    4*4 + 1 (r14),r5  /* read clock 1 data */
3398  8033a012                          ldob    4*4 + 2 (r14),r6  /* read clock 2 data */
3398
3398                                    /* vote and identify error */
339c  85683000 00003404                balx    vote,r13
339c
33a4  32f9e00c                          cmpobe  0x1f,r7,store_minute
33a4                                    /* fix minute */
33a8  92383000 0300001c                st      r7,clock_minutes  /* add rollover detection */
33a8
33b0  8223a005          store_minute: stob    r4, 5 (r14)
33b0
33b0
33b0
33b0                    /*************************************************************************/
33b4  8023a014          ckfx_second:    ldob    5*4 (r14),r4  /* read clock 0 data */
33b8  802ba015                          ldob    5*4 + 1 (r14),r5  /* read clock 1 data */
33bc  8033a016                          ldob    5*4 + 2 (r14),r6  /* read clock 2 data */
33bc
33bc                                    /* vote and identify error */
33c0  85683000 00003404                balx    vote,r13
33c0
33c8  32f9e00c                          cmpobe  0x1f,r7,store_second
33c8                                    /* fix second */
33cc  92383000 03000018                st      r7,clock_seconds  /* add rollover detection */
33cc
33d4  8223a006          store_second: stob    r4, 6 (r14)
33d4
33d4
33d4
```

```
33d4                                /*************************************************************************/
33d8                                ckfx_hundredths:
33d8  8023a018                      ldob    6*4 (r14),r4  /* read clock 0 data */
33dc  802ba019                      ldob    6*4 + 1 (r14),r5  /* read clock 1 data */
33e0  8033a01a                      ldob    6*4 + 2 (r14),r6  /* read clock 2 data */
33e0
33e0                                /* vote and identify error */
33e4  85683000 00003404             balx    vote,r13
33e4
33ec  32f9e00c                      cmpobe  0x1f,r7,store_hundredths
33ec                                /* fix hundredths */
33f0  92383000 03000014             st      r7,clock_hundredths  /* add rollover detection */
33f0
33f8                                store_hundredths:
33f8  8223a007                      stob    r4, 7 (r14)
33f8
33f8                                /* for now make the last part always zero */
33fc  8243a008                      stob    r8, 8 (r14)
33fc
33fc
33fc
3400  8403d000                      bx      (r15)
3400
3400
3400                                /*************************************************************************/
3400                                /* subroutine to check time                                              */
3400                                /* inputs are time in r4,r5,r6                                           */
3400                                /* outputs are corrected time in r4 and 0/time to send in r7             */
3400                                /*************************************************************************/
3400
3404  58394304                      vote:         xor     r4,r5,r7  /* clk0 xor clk1 */
3408  58418304                                    xor     r4,r6,r8  /* clk0 xor clk2 */
340c  58498305                                    xor     r5,r6,r9  /* clk1 xor clk2 */
340c
340c                                /* check for no errors */
3410  58520387                      or      r7,r8,r10
3414  58528389                      or      r9,r10,r10
3418  3502a00c                      cmpobne 0x0,r10,chk_clk2
3418
341c  5c381e1f                      ldconst 0x1f,r7
341c
3420  84035000                      bx      (r13)
3420
3420
3420                                /* check for clk2 wrong */
3424  3501e02c                      chk_clk2:     cmpobne 0x0,r7,chk_clk1
3424                                /* set bit 13 of flags */
3428  8c503000 01017000             ldconst flags,r10
3430  59585e0d                      ldconst 0x00002000,r11
3434  59605e0d                      ldconst 0x00002000,r12
3438  6162c00a                      atmod   r10,r11,r12
3438
3438                                /* get good time tripled in r7 */
343c  59390e08                      shlo    8,r4,r7
3440  5839c384                      or      r4,r7,r7
3444  5939ce08                      shlo    8,r7,r7
3448  5839c384                      or      r4,r7,r7
3448
344c  84035000                      bx      (r13)
344c
344c
344c                                /* check for clk1 wrong */
3450  3502202c                      chk_clk1:     cmpobne 0x0,r8,chk_clk0
3450                                /* set bit 12 of flags */
3454  8c503000 01017000             ldconst flags,r10
345c  59585e0c                      ldconst 0x00001000,r11
3460  59605e0c                      ldconst 0x00001000,r12
3464  6162c00a                      atmod   r10,r11,r12
3464
3464                                /* get good time tripled in r7 */
3468  59390e08                      shlo    8,r4,r7
346c  5839c384                      or      r4,r7,r7
3470  5939ce08                      shlo    8,r7,r7
3474  5839c384                      or      r4,r7,r7
3474
3478  84035000                      bx      (r13)
3478
3478
```

```
3478                                    /* check for clk0 wrong */
347c  35026030              chk_clk0:   cmpobne 0x0,r9,all_wrong
347c                                    /* set bit 11 of flags */
3480  8c503000 01017000                ldconst flags,r10
3488  59585e0b                         ldconst 0x00000800,r11
348c  59605e0b                         ldconst 0x00000800,r12
3490  6162c00a                         atmod   r10,r11,r12
3490
3494  5c201605                         mov     r5,r4
3494                                    /* get good time tripled in r7 */
3498  59390e08                         shlo    8,r4,r7
349c  5839c384                         or      r4,r7,r7
34a0  5939ce08                         shlo    8,r7,r7
34a4  5839c384                         or      r4,r7,r7
34a4
34a8  84035000                         bx      (r13)
34a8
34a8
34ac                        all_wrong:
34ac                                    /* set bit 14 of flags, time uncorrectable */
34ac  8c503000 01017000                ldconst flags,r10
34b4  59585e0e                         ldconst 0x00004000,r11
34b8  59605e0e                         ldconst 0x00004000,r12
34bc  6162c00a                         atmod   r10,r11,r12
34bc
34bc                                    /* get clk0 time tripled in r7 */
34c0  59390e08                         shlo    8,r4,r7
34c4  5839c384                         or      r4,r7,r7
34c8  5939ce08                         shlo    8,r7,r7
34cc  5839c384                         or      r4,r7,r7
34cc
34d0  84035000                         bx      (r13)
34d0
34d0
34d0                        /*************************************************************************/
34d0                        /* subroutine to set the time in the clock                               */
34d0                        /* time is stored in memory starting at desired_time                     */
34d0                        /* each word contains trippled time that will go directly into the clks */
34d0                        /* uses registers ....                                                   */
34d0                        /*      r14 contains return IP                                           */
34d0                        /*                                                                       */
34d0                        /*    R. Higgins                                                         */
34d0                        /*    2/19/92                                                            */
34d0                        /*    3/9/92 zero valid byte in setting data, set leap year too          */
34d0                        /*    3/14/92 rearrange meaning of setting data                          */
34d0                        /*    3/16/92 fix invalidation of setting data                           */
34d0                        /*************************************************************************/
34d0
34d4                        set_clocks:
34d4  8c583000 00ffffff                ldconst 0x00ffffff,r11  /* use to zero valid byte */
34d4                                    /* ok to reset clocks */
34dc  8c503000 007f7f7f                ldconst 0x007f7f7f,r10  /* select page 0 reg 1 */
34e4  92503000 03000000                st      r10,clock_msr
34ec  5c501e00                         ldconst 0x00000000,r10
34f0  92503000 03000004                st      r10,clock_rtm  /* stop clock */
34f0
34f0                                    /* set everything */
34f8  90503000 01018f04                ld      desired_time + 1*4,r10
3500  92503000 0300002c                st      r10,clock_years  /* set clock */
3508  5852c08a                         and     r10,r11,r10  /* zero valid byte */
350c  92503000 01018f04                st      r10,desired_time + 1*4
350c
3514  90503000 01018f08                ld      desired_time + 2*4,r10
351c  92503000 03000034                st      r10,clock_date_msb  /* set clock */
3524  5852c08a                         and     r10,r11,r10  /* zero valid byte */
3528  92503000 01018f08                st      r10,desired_time + 2*4
3528
3530  90503000 01018f0c                ld      desired_time + 3*4,r10
3538  92503000 03000030                st      r10,clock_date_lsb  /* set clock */
3540  5852c08a                         and     r10,r11,r10  /* zero valid byte */
3544  92503000 01018f0c                st      r10,desired_time + 3*4
3544
354c  90503000 01018f10                ld      desired_time + 4*4,r10
3554  92503000 03000020                st      r10,clock_hours  /* set clock */
355c  5852c08a                         and     r10,r11,r10  /* zero valid byte */
3560  92503000 01018f10                st      r10,desired_time + 4*4
3560
3568  90503000 01018f14                ld      desired_time + 5*4,r10
3570  92503000 0300001c                st      r10,clock_minutes  /* set clock */
3578  5852c08a                         and     r10,r11,r10  /* zero valid byte */
357c  92503000 01018f14                st      r10,desired_time + 5*4
```

```
357c
3584  90503000 01018f18              ld      desired_time + 6*4,r10
358c  92503000 03000018              st      r10,clock_seconds  /* set clock */
3594  5852c08a                       and     r10,r11,r10  /* zero valid byte */
3598  92503000 01018f18              st      r10,desired_time + 6*4
3598
35a0  90503000 01018f1c              ld      desired_time + 7*4,r10
35a8  92503000 03000014              st      r10,clock_hundredths  /* set clock */
35b0  5852c08a                       and     r10,r11,r10  /* zero valid byte */
35b4  92503000 01018f1c              st      r10,desired_time + 7*4
35b4
35bc  8c503000 00ffffff              ldconst 0x00ffffff,r10  /* select page 1 */
35c4  92503000 03000000              st      r10,clock_msr
35cc  90503000 01018f00              ld      desired_time + 0*4,r10
35d4  92503000 03000078              st      r10,clock_ram_1E  /* set years msb */
35dc  5852c08a                       and     r10,r11,r10  /* zero valid byte */
35e0  92503000 01018f00              st      r10,desired_time + 0*4
35e0
35e8  8c503000 003f3f3f              ldconst 0x003f3f3f,r10
35f0  92503000 03000000              st      r10,clock_msr  /* select page 0 reg 0 */
35f8  8c503000 00404040              ldconst 0x00404040,r10
3600  92503000 0300000c              st      r10,clock_pfr  /* sngl supply mode, no test mode */
3600
3608  8c503000 007f7f7f              ldconst 0x007f7f7f,r10  /* select page 0 reg 1 */
3610  92503000 03000000              st      r10,clock_msr
3610                                 /* clk manual suggests to set modes then start clock */
3610
3610                                 /* set clks: 32.768khz osc,timer stdby,pf dis, 24 hr */
3618  90503000 01018f24              ld      desired_time + 9*4,r10  /* load leap year status */
3620  58528985                       setbit  5,r10,r10
3624  5852898d                       setbit  13,r10,r10
3628  58528995                       setbit  21,r10,r10
362c  92503000 03000004              st      r10,clock_rtm
3634  5852c08a                       and     r10,r11,r10  /* zero valid byte */
3638  92503000 01018f24              st      r10,desired_time + 9*4
3638
3640  8c503000 00282828              ldconst 0x00282828,r10  /* start clock */
3648  92503000 03000004              st      r10,clock_rtm
3648
3650  8c503000 003f3f3f              ldconst 0x003f3f3f,r10
3658  92503000 03000000              st      r10,clock_msr  /* select page 0 reg 0 */
3658
3658
3660  84039000                       bx      (r14)
3660
3660
3660                           #line 1 "hap.src:faults.src"
3660                           /* ************************************************************************/
3660                           /*    Fault handling routines                                             */
3660                           /*                                                                        */
3660                           /*    R. Higgins                                                          */
3660                           /*    10/15/92                                                            */
3660                           /*    1/3/92  add system error interrupt shell                            */
3660                           /*    2/4/92  add fault data storage and new fault flag stuff             */
3660                           /* ************************************************************************/
3660
3664                           override_fault:
3664  90803000 01017000              ld      flags,g0  /* load flags into g0 */
3664
366c  90883000 01017004              ld      more_flags,g1  /* load more_flags into g1, set fault flag */
3674  588c4983                       setbit  3,g1,g1
3678  588c4991                       setbit  17,g1,g1
3678
3678                                 /* add captured fault data to global registers */
367c  90907400 ffffffec              ld      - 48 + 28 (sp),g2
3684  90987400 fffffff0              ld      - 48 + 32 (sp),g3
368c  90a07400 fffffff4              ld      - 48 + 36 (sp),g4
3694  90a87400 fffffff8              ld      - 48 + 40 (sp),g5
369c  90b07400 fffffffc              ld      - 48 + 44 (sp),g6
369c
36a4  ffdc6c 08                      b       restart  /* go to the beginning of the software */
36a4
36a8  0a000000                       ret
36a8
36a8
```

```
36a8                        /* ********************************************************************** */
36a8
36ac                        trace_fault:
36ac  8c183000 01017004         ldconst  more_flags,r3
36b4  59205e12                  ldconst  ( 1<<18 ),r4
36b8  59285e12                  ldconst  ( 1<<18 ),r5
36bc  61290003                  atmod    r3,r4,r5  /* set the trace fault flag */
36bc
36bc                            /* add captured fault data to memory */
36c0  90907400 ffffffec         ld       - 48 + 28 (sp),g2
36c8  90987400 fffffff0         ld       - 48 + 32 (sp),g3
36d0  90a07400 fffffff4         ld       - 48 + 36 (sp),g4
36d8  90a87400 fffffff8         ld       - 48 + 40 (sp),g5
36e0  90b07400 fffffffc         ld       - 48 + 44 (sp),g6
36e8  92903000 01017008         st       g2,fault_record
36f0  92983000 0101700c         st       g3,fault_record + 1*4
36f8  92a03000 01017010         st       g4,fault_record + 2*4
3700  92a83000 01017014         st       g5,fault_record + 3*4
3708  92b03000 01017018         st       g6,fault_record + 4*4
3708
3710  00041c 08                 b        warmstart  /* go to the warmstart routine */
3710
3714  0a000000                  ret
3714
3714
3714                        /* ********************************************************************** */
3714
3718                        operation_fault:
3718  8c183000 01017004         ldconst  more_flags,r3
3720  59205e13                  ldconst  ( 1<<19 ),r4
3724  59285e13                  ldconst  ( 1<<19 ),r5
3728  61290003                  atmod    r3,r4,r5  /* set the operation fault flag */
3728
3728                            /* add captured fault data to memory */
372c  90907400 ffffffec         ld       - 48 + 28 (sp),g2
3734  90987400 fffffff0         ld       - 48 + 32 (sp),g3
373c  90a07400 fffffff4         ld       - 48 + 36 (sp),g4
3744  90a87400 fffffff8         ld       - 48 + 40 (sp),g5
374c  90b07400 fffffffc         ld       - 48 + 44 (sp),g6
3754  92903000 01017008         st       g2,fault_record
375c  92983000 0101700c         st       g3,fault_record + 1*4
3764  92a03000 01017010         st       g4,fault_record + 2*4
376c  92a83000 01017014         st       g5,fault_record + 3*4
3774  92b03000 01017018         st       g6,fault_record + 4*4
3774
377c  0003b0 08                 b        warmstart  /* go to the warmstart routine */
377c
3780  0a000000                  ret
3780
3780
3780                        /* ********************************************************************** */
3780
3784                        arithmetic_fault:
3784  90803000 01017000         ld       flags,g0  /* load flags into g0 */
3784
378c  90883000 01017004         ld       more_flags,g1  /* load more_flags into g1, set fault flag */
3794  588c4983                  setbit   3,g1,g1
3798  588c4994                  setbit   20,g1,g1
3798
3798                            /* add captured fault data to global registers */
379c  90907400 ffffffec         ld       - 48 + 28 (sp),g2
37a4  90987400 fffffff0         ld       - 48 + 32 (sp),g3
37ac  90a07400 fffffff4         ld       - 48 + 36 (sp),g4
37b4  90a87400 fffffff8         ld       - 48 + 40 (sp),g5
37bc  90b07400 fffffffc         ld       - 48 + 44 (sp),g6
37bc
37c4  ffdb4c 08                 b        restart  /* go to the beginning of the software */
37c4
37c8  0a000000                  ret
37c8
37c8
37c8                        /* ********************************************************************** */
37c8
37cc                        floating_fault:
37cc  90803000 01017000         ld       flags,g0  /* load flags into g0 */
37cc
37d4  90883000 01017004         ld       more_flags,g1  /* load more_flags into g1, set fault flag */
37dc  588c4983                  setbit   3,g1,g1
37e0  588c4995                  setbit   21,g1,g1
37e0
```

```
37e0                            /* add captured fault data to global registers */
37e4  90907400 ffffffec        ld       - 48 + 28 (sp),g2
37ec  90987400 fffffff0        ld       - 48 + 32 (sp),g3
37f4  90a07400 fffffff4        ld       - 48 + 36 (sp),g4
37fc  90a87400 fffffff8        ld       - 48 + 40 (sp),g5
3804  90b07400 fffffffc        ld       - 48 + 44 (sp),g6
3804
380c  ffdb04 08                b        restart  /* go to the beginning of the software */
380c
3810  0a000000                 ret
3810
3810
3810                          /* ********************************************************************** */
3810
3814                          constraint_fault:
3814  8c183000 01017004        ldconst  more_flags,r3
381c  59205e16                 ldconst  ( 1<<22 ),r4
3820  59285e16                 ldconst  ( 1<<22 ),r5
3824  61290003                 atmod    r3,r4,r5  /* set the trace fault flag */
3824
3824                            /* add captured fault data to memory */
3828  90907400 ffffffec        ld       - 48 + 28 (sp),g2
3830  90987400 fffffff0        ld       - 48 + 32 (sp),g3
3838  90a07400 fffffff4        ld       - 48 + 36 (sp),g4
3840  90a87400 fffffff8        ld       - 48 + 40 (sp),g5
3848  90b07400 fffffffc        ld       - 48 + 44 (sp),g6
3850  92903000 01017008        st       g2,fault_record
3858  92983000 0101700c        st       g3,fault_record + 1*4
3860  92a03000 01017010        st       g4,fault_record + 2*4
3868  92a83000 01017014        st       g5,fault_record + 3*4
3870  92b03000 01017018        st       g6,fault_record + 4*4
3870
3878  0002b4 08                b        warmstart  /* go to the warmstart routine */
3878
387c  0a000000                 ret
387c
387c
387c                          /* ********************************************************************** */
387c
3880                          virtual_memory_fault:
3880  8c183000 01017004        ldconst  more_flags,r3
3888  59205e17                 ldconst  ( 1<<23 ),r4
388c  59285e17                 ldconst  ( 1<<23 ),r5
3890  61290003                 atmod    r3,r4,r5  /* set the trace fault flag */
3890
3890                            /* add captured fault data to memory */
3894  90907400 ffffffec        ld       - 48 + 28 (sp),g2
389c  90987400 fffffff0        ld       - 48 + 32 (sp),g3
38a4  90a07400 fffffff4        ld       - 48 + 36 (sp),g4
38ac  90a87400 fffffff8        ld       - 48 + 40 (sp),g5
38b4  90b07400 fffffffc        ld       - 48 + 44 (sp),g6
38bc  92903000 01017008        st       g2,fault_record
38c4  92983000 0101700c        st       g3,fault_record + 1*4
38cc  92a03000 01017010        st       g4,fault_record + 2*4
38d4  92a83000 01017014        st       g5,fault_record + 3*4
38dc  92b03000 01017018        st       g6,fault_record + 4*4
38dc
38e4  000248 08                b        warmstart  /* go to the warmstart routine */
38e4
38e8  0a000000                 ret
38e8
38e8
38e8                          /* ********************************************************************** */
38e8
38ec                          protection_fault:
38ec  90803000 01017000        ld       flags,g0  /* load flags into g0 */
38ec
38f4  90883000 01017004        ld       more_flags,g1  /* load more_flags into g1, set fault flag */
38fc  588c4983                 setbit   3,g1,g1
3900  588c4998                 setbit   24,g1,g1
3900
3900                            /* add captured fault data to global registers */
3904  90907400 ffffffec        ld       - 48 + 28 (sp),g2
390c  90987400 fffffff0        ld       - 48 + 32 (sp),g3
3914  90a07400 fffffff4        ld       - 48 + 36 (sp),g4
391c  90a87400 fffffff8        ld       - 48 + 40 (sp),g5
3924  90b07400 fffffffc        ld       - 48 + 44 (sp),g6
3924
392c  ffd9e4 08                b        restart  /* go to the beginning of the software */
392c
3930  0a000000                 ret
```

```
3930
3930
3930                    /* ********************************************************************** */
3930
3934                    machine_fault:
3934  8c183000 01017004         ldconst  more_flags,r3
393c  59205e19                  ldconst  ( 1<<25 ),r4
3940  59285e19                  ldconst  ( 1<<25 ),r5
3944  61290003                  atmod    r3,r4,r5  /* set the trace fault flag */
3944
3944                            /* add captured fault data to memory */
3948  90907400 ffffffec         ld       - 48 + 28 (sp),g2
3950  90987400 fffffff0         ld       - 48 + 32 (sp),g3
3958  90a07400 fffffff4         ld       - 48 + 36 (sp),g4
3960  90a87400 fffffff8         ld       - 48 + 40 (sp),g5
3968  90b07400 fffffffc         ld       - 48 + 44 (sp),g6
3970  92903000 01017008         st       g2,fault_record
3978  92983000 0101700c         st       g3,fault_record + 1*4
3980  92a03000 01017010         st       g4,fault_record + 2*4
3988  92a83000 01017014         st       g5,fault_record + 3*4
3990  92b03000 01017018         st       g6,fault_record + 4*4
3990
3998  000194 08                 b        warmstart  /* go to the warmstart routine */
3998
399c  0a000000                  ret
399c
399c
399c                    /* ********************************************************************** */
399c
39a0                    structural_fault:
39a0  90803000 01017000         ld       flags,g0  /* load flags into g0 */
39a0
39a8  90883000 01017004         ld       more_flags,g1  /* load more_flags into g1, set fault flag */
39b0  588c4983                  setbit   3,g1,g1
39b4  588c499a                  setbit   26,g1,g1
39b4
39b4                            /* add captured fault data to global registers */
39b8  90907400 ffffffec         ld       - 48 + 28 (sp),g2
39c0  90987400 fffffff0         ld       - 48 + 32 (sp),g3
39c8  90a07400 fffffff4         ld       - 48 + 36 (sp),g4
39d0  90a87400 fffffff8         ld       - 48 + 40 (sp),g5
39d8  90b07400 fffffffc         ld       - 48 + 44 (sp),g6
39d8
39e0  ffd930 08                 b        restart  /* go to the beginning of the software */
39e0
39e4  0a000000                  ret
39e4
39e4
39e4                    /* ********************************************************************** */
39e4
39e8                    type_fault:
39e8  90803000 01017000         ld       flags,g0  /* load flags into g0 */
39e8
39f0  90883000 01017004         ld       more_flags,g1  /* load more_flags into g1, set fault flag */
39f8  588c4983                  setbit   3,g1,g1
39fc  588c499b                  setbit   27,g1,g1
39fc
39fc                            /* add captured fault data to global registers */
3a00  90907400 ffffffec         ld       - 48 + 28 (sp),g2
3a08  90987400 fffffff0         ld       - 48 + 32 (sp),g3
3a10  90a07400 fffffff4         ld       - 48 + 36 (sp),g4
3a18  90a87400 fffffff8         ld       - 48 + 40 (sp),g5
3a20  90b07400 fffffffc         ld       - 48 + 44 (sp),g6
3a20
3a28  ffd8e8 08                 b        restart  /* go to the beginning of the software */
3a28
3a2c  0a000000                  ret
3a2c
3a2c
3a2c                    /* ********************************************************************** */
3a2c
3a30                    process_fault:
3a30  90803000 01017000         ld       flags,g0  /* load flags into g0 */
3a30
3a38  90883000 01017004         ld       more_flags,g1  /* load more_flags into g1, set fault flag */
3a40  588c4983                  setbit   3,g1,g1
3a44  588c499c                  setbit   28,g1,g1
3a44
```

```
3a44                                /* add captured fault data to global registers */
3a48  90907400 fffffffec            ld      - 48 + 28 (sp),g2
3a50  90987400 fffffff0             ld      - 48 + 32 (sp),g3
3a58  90a07400 fffffff4             ld      - 48 + 36 (sp),g4
3a60  90a87400 fffffff8             ld      - 48 + 40 (sp),g5
3a68  90b07400 fffffffc             ld      - 48 + 44 (sp),g6
3a68
3a70  ffd8a0 08                     b       restart  /* go to the beginning of the software */
3a70
3a74  0a000000                      ret
3a74
3a74
3a74                              /* ********************************************************************** */
3a74
3a78                              descriptor_fault:
3a78  90803000 01017000             ld      flags,g0  /* load flags into g0 */
3a78
3a80  90883000 01017004             ld      more_flags,g1  /* load more_flags into g1, set fault flag */
3a88  588c4983                      setbit  3,g1,g1
3a8c  588c499d                      setbit  29,g1,g1
3a8c
3a8c                                /* add captured fault data to global registers */
3a90  90907400 fffffffec            ld      - 48 + 28 (sp),g2
3a98  90987400 fffffff0             ld      - 48 + 32 (sp),g3
3aa0  90a07400 fffffff4             ld      - 48 + 36 (sp),g4
3aa8  90a87400 fffffff8             ld      - 48 + 40 (sp),g5
3ab0  90b07400 fffffffc             ld      - 48 + 44 (sp),g6
3ab0
3ab8  ffd858 08                     b       restart  /* go to the beginning of the software */
3ab8
3abc  0a000000                      ret
3abc
3abc
3abc                              /* ********************************************************************** */
3abc
3ac0                              event_fault:
3ac0  90b07400 fffffffc             ld      - 48 + 44 (sp),g6  /* address of faulting instruction */
3ac0
3ac0                                /* restart processor */
3ac8  8c883000 ff000010             ldconst  proc_iac,g1
3ad0  8c903000 000040f0             ldconst  restart_data,g2
3ad0
3ad8  6004a111                      synmovq  g1,g2
3ad8
3ad8                                /* branch to faulting instruction */
3adc  84019000                      bx      (r6)
3adc
3ae0  0a000000                      ret
3ae0
3ae0
3ae0                              /* ********************************************************************** */
3ae0
3ae4                              system_err_intr:
3ae4  90803000 01017000             ld      flags,g0  /* load flags into g0 */
3ae4
3aec  90883000 01017004             ld      more_flags,g1  /* load more_flags into g1, set fault flag */
3af4  588c4983                      setbit  3,g1,g1
3af8  588c499f                      setbit  31,g1,g1
3af8
3af8                                /* add captured fault data to global registers */
3afc  90907400 fffffffec            ld      - 48 + 28 (sp),g2
3b04  90987400 fffffff0             ld      - 48 + 32 (sp),g3
3b0c  90a07400 fffffff4             ld      - 48 + 36 (sp),g4
3b14  90a87400 fffffff8             ld      - 48 + 40 (sp),g5
3b1c  90b07400 fffffffc             ld      - 48 + 44 (sp),g6
3b1c
3b24  ffd7ec 08                     b       restart  /* go to the beginning of the software */
3b24
3b28  0a000000                      ret
3b28
3b28
3b28                              /* ********************************************************************** */
3b28
3b2c                                .globl warmstart
3b2c                              warmstart:
3b2c  5c801e00                      ldconst  0,g0
3b30  82803000 05000000             stob     g0,hap_ier  /* disable all interrupts */
3b30
3b38  8c183000 01017000             ldconst  flags,r3
3b40  5c201e08                      ldconst  (1<<3),r4
3b44  5c281e08                      ldconst  (1<<3),r5
```

```
3b48  61290003                atmod    r3,r4,r5  /* set the warmstart fault flag */
3b48
3b48
3b48                        /* ********************************************************************** */
3b48                         /* make the system data structures in RAM, restart processor using new
3b48                            data structures, bind to the processor */
3b48
3b4c  85783000 00003b58       balx     make_data_structures,r15
3b4c
3b4c                          /* go to the executive warmstart */
3b54  ffdf90 08               b        exec_warmstart
3b54
3b54                        #line 1 "hap.src:movdata.src"
3b54                        /************************************************************************/
3b54                        /*    MOVDATA.SRC                                                       */
3b54                        /*    This routine makes the system data structures in ram, restarts    */
3b54                        /*    processor using the new data structures, binds only process to    */
3b54                        /*    processor, and returns to the calling location                    */
3b54                        /*                                                                      */
3b54                        /*    1/7/92  r.h.                                                      */
3b54                        /*    1/23/92 redo restart, prcb, pcb, intr stack                       */
3b54                        /*    1/29/92 set interruput table with warmstart in unused locations   */
3b54                        /*                                                                      */
3b54                        /************************************************************************/
3b54
3b58                          .globl make_data_structures
3b58                        make_data_structures:
3b58
3b58                        /************************************************************************/
3b58                         /* make the interrupt stack in RAM */
3b58                        /************************************************************************/
3b58
3b58                          .globl intr_stack /* on 64 byte boundary */
3b58
3b58  5c901e00                   ldconst 0,g2
3b5c  92903000 0101f000          st      g2,intr_stack           /* pfp */
3b5c
3b64  8c903000 0101f040          ldconst intr_stack + 64,g2      /* sp */
3b6c  92903000 0101f004          st      g2,intr_stack + 4
3b6c
3b74  8c903000 000019e4          ldconst exec_init,g2            /* rip */
3b7c  92903000 0101f008          st      g2,intr_stack + 8
3b7c
3b7c
3b7c                        /************************************************************************/
3b7c                         /* make the system procedure stack in RAM */
3b7c                        /************************************************************************/
3b7c
3b84                          .globl sys_proc_stack /* on 64 byte boundary */
3b84
3b84  5c901e00                   ldconst 0,g2
3b88  92903000 0101f500          st      g2,sys_proc_stack       /* pfp */
3b88
3b90  8c903000 0101f540          ldconst sys_proc_stack + 64,g2  /* sp */
3b98  92903000 0101f504          st      g2,sys_proc_stack + 4
3b98
3ba0  8c903000 000019e4          ldconst exec_init,g2            /* rip */
3ba8  92903000 0101f508          st      g2,sys_proc_stack + 8
3ba8
3ba8
3ba8                        /************************************************************************/
3ba8                         /* create the operating processor control block (PRCB) in RAM */
3ba8                        /************************************************************************/
3ba8
3bb0                          .globl startup_prcb
3bb0                          .globl operation_prcb
3bb0
3bb0  599b581f                   ldconst 172/4 + 1,g3  /* prcb is 172 bytes long */
3bb0
3bb4                        move_prcb:
3bb4  90a03913 00000ffc          ld      startup_prcb - 4 [g3 * 4],g4
3bbc  92a03913 0101f8fc          st      g4,operation_prcb - 4 [g3 * 4]
3bbc
3bc4  599cc901                   subo    0x01,g3,g3
3bc8  3504ffec                   cmpobne 0x00,g3,move_prcb
3bc8
3bc8
3bc8                             /* put in the correct processor controls word */
3bcc  8c98000c                   lda     0x0000000c,g3  /* process executing */
3bd0  92983000 0101f904          st      g3,operation_prcb + 1*4
3bd0
```

```
3bd0                                    /* put in the correct PCB Segment Selector */
3bd8  8c98037f                          lda     operation_pcb_ss,g3
3bdc  92983000 0101f90c                 st      g3,operation_prcb + 3*4
3bdc
3bdc                                    /* put in the correct fault table pointer */
3be4  8c983000 0101ff00                 lda     fault_table,g3
3bec  92983000 0101f928                 st      g3,operation_prcb + 10*4
3bec
3bec
3bec                                /************************************************************************/
3bec                                 /* make the process control block (PCB) in RAM */
3bec                                /************************************************************************/
3bec
3bf4                                  .globl startup_pcb
3bf4                                  .globl operation_pcb
3bf4
3bf4  599f981f                          ldconst 240/4 + 1,g3
3bf4
3bf8                                move_pcb:
3bf8  90903913 000011fc                 ld      startup_pcb - 4 [g3 * 4],g2
3c00  92903913 0101f9fc                 st      g2,operation_pcb - 4 [g3 * 4]
3c00
3c08  599cc901                          subo    0x01,g3,g3
3c0c  3504ffec                          cmpobne 0x00,g3,move_pcb
3c0c
3c0c
3c0c                                /************************************************************************/
3c0c                                 /* make the interrupt table in RAM */
3c0c                                /************************************************************************/
3c0c
3c10                                  .globl intr_table
3c10                                  .globl serial_intr
3c10                                  .globl iae_intr
3c10                                  .globl trigger_intr
3c10                                  .globl watchdog_intr
3c10
3c10                                set_intr_table:
3c10  8c903000 00003b2c                         ldconst warmstart,g2  /* send unused vectors to warmstart */
3c18  8c980101                                  ldconst 257,g3  /* 256 entries */
3c18
3c1c                                fill_intr_table:
3c1c  92903913 0101fafc                         st      g2,intr_table - 4 [g3 * 4]
3c1c
3c24  599cc901                                  subo    0x01,g3,g3
3c28  3504fff4                                  cmpobne 0,g3,fill_intr_table
3c28
3c28                                            /* initialize reserved entries to 0 */
3c2c  5c901e00                                  ldconst 0,g2
3c30  92903000 0101fed4                         st      g2,intr_table + 244*4 + 4
3c38  92903000 0101fed8                         st      g2,intr_table + 245*4 + 4
3c40  92903000 0101fedc                         st      g2,intr_table + 246*4 + 4
3c48  92903000 0101fee0                         st      g2,intr_table + 247*4 + 4
3c50  92903000 0101fee8                         st      g2,intr_table + 249*4 + 4
3c58  92903000 0101feec                         st      g2,intr_table + 250*4 + 4
3c60  92903000 0101fef0                         st      g2,intr_table + 251*4 + 4
3c60
3c60                                            /* initialize pending priorities to 0 */
3c68  92903000 0101fb00                         st      g2,intr_table + 0*4
3c68
3c68                                            /* initialize pending interrupts to 0 */
3c70  92903000 0101fb04                         st      g2,intr_table + 1*4
3c78  92903000 0101fb08                         st      g2,intr_table + 2*4
3c80  92903000 0101fb0c                         st      g2,intr_table + 3*4
3c88  92903000 0101fb10                         st      g2,intr_table + 4*4
3c90  92903000 0101fb14                         st      g2,intr_table + 5*4
3c98  92903000 0101fb18                         st      g2,intr_table + 6*4
3ca0  92903000 0101fb1c                         st      g2,intr_table + 7*4
3ca0
3ca0                                            /* set up the interrupt vectors */
3ca0                                            /* priority = vector / 8 */
3ca0                                            /* vector 244-247, 249-251 rsvd, do not use */
3ca0                                            /* vector 248 reserved for system error interrupt */
3ca0
3ca0                                             /* serial intr priority 5, vector 40 (28)h */
3ca8  8c903000 00003198                         ldconst serial_intr,g2
3cb0  92903000 0101fba4                         st      g2,intr_table + 40*4 + 4
3cb0
3cb0                                             /* iae intr priority 7, vector 56 (38)h */
3cb8  8c903000 000029a4                         ldconst iae_intr,g2
3cc0  92903000 0101fbe4                         st      g2,intr_table + 56*4 + 4
3cc0
```

```
3cc0                                    /* trigger intr priority 7, vector 58 (3a)h */
3cc8  8c903000 00002888                 ldconst trigger_intr,g2
3cd0  92903000 0101fbec                 st      g2,intr_table + 58*4 + 4
3cd0
3cd0                                    /* sys error interrupt, priority 31 vector fixed at 248 */
3cd8  8c903000 00003ae4                 ldconst system_err_intr,g2
3ce0  92903000 0101fee4                 st      g2,intr_table + 248*4 + 4
3ce0
3ce0                                    /* watchdog timer priority 31, vector 254 (fc)h */
3ce8  8c903000 000026e0                 ldconst watchdog_intr,g2
3cf0  92903000 0101fef4                 st      g2,intr_table + 252*4 + 4
3cf0
3cf0
3cf0                        /*************************************************************************/
3cf0                         /* set the processor interrupt control register */
3cf0                        /*************************************************************************/
3cf0
40d8                        .data
40d8
40d8                          .align 4
40d8
40e0                        proc_icr_value:
40e0  283a38fc                          .word 0x283a38fc
40e0
3cf8                        .text
3cf8
3cf8                          .globl proc_icr_addr
3cf8
3cf8                        init_proc_icr:
3cf8  8c883000 ff000004                 ldconst proc_icr_addr,g1
3d00  90903000 000040e0                 ld      proc_icr_value,g2
3d08  6004a011                          synmov  g1,g2
3d08
3d08
3d08                        /*************************************************************************/
3d08                         /* put the fault table in ram */
3d08                        /*************************************************************************/
3d08
3d0c                          .globl prom_fault_table
3d0c                          .globl fault_table
3d0c
3d0c                        copy_fault_table:
3d0c  8c980041                          ldconst 256/4 + 1,g3
3d0c
3d10                        move_fault_table:
3d10  90903913 000010ac                 ld      prom_fault_table - 4 [g3 * 4],g2
3d18  92903913 0101fefc                 st      g2,fault_table - 4 [g3 * 4]
3d18
3d20  599cc901                          subo    0x01,g3,g3
3d24  3504ffec                          cmpobne 0,g3,move_fault_table
3d24
3d24
3d24                        /*************************************************************************/
3d24                         /* restart the processor using the new data structures and
3d24                                 bind it to the processor                                 */
3d24                        /*************************************************************************/
3d24
3d24
40e4                        .data
40e4
40e4                          .align 4
40e4
40f0                        restart_data:
40f0  81000000                          .word 0x81000000        /* restart processor iac */
40f4  00000000                          .word seg_table         /* pointer to segment table */
40f8  0101f900                          .word operation_prcb    /* pointer to new prcb */
40fc  00000000                          .word 0x0               /* unused */
40fc

3d28                        .text
3d28
3d28                        /* restart the processor with the new operation_prcb, and operation_pcb */
3d28
3d28  8c883000 ff000010                 ldconst proc_iac,g1
3d30  8c903000 000040f0                 ldconst restart_data,g2
3d30
3d38  6004a111                          synmovq g1,g2
3d38
3d3c  8403d000                          bx      (r15)
3d3c
3d3c
```

```
3d40                    .align 4  /* start the data area on a four word boundary */

Number of errors:    0
Number of warnings:  0
```

Appendix F - HAP IAE Data Processing C-code Listing

```
/*************************************************************************/
/*  iae10.c  gyro data processing, melvin algorithm #2                   */
/*            Honeywell 3/17/1992 constants                              */
/*  R. Higgins                                                           */
/*  Nov 11, 1991                                                         */
/*  Nov 15, 1991 changed the mg matrix, sf"a" mul by col "a"             */
/*  Mar 17, 1992 use new Honeywell constants                             */
/*************************************************************************/

#include <fpsl.h>
#include <math.h>
#include <float.h>


#pragma inline

          /* define iae read ports */
          extern volatile unsigned short iae_a1;
          extern volatile unsigned short iae_a2;
          extern volatile unsigned short iae_a3;

          /* define current solution positon */
          extern double current_solution[4];

          /* define solution counter positon */
          extern unsigned short solution_count;

          /* define previous input */
          extern long pin[3];

          /* define filter array */
          extern long a[3][13];

          /* define previous delta angle value */
          extern double pfda[3];

          /* define last compensated and coning recovered angle value */
          extern double pb[3];

          /* define old theata values */
          extern double tho[4];


          /*************************************************************************/

    /* Honeywell values provided on Mar 17,1992
gyro nonorthogonality (arc seconds)
      59.102500000000000       82.173500000000000      236.599500000000000

nonorthogonality matrix constants
       1.000000000000000   0.000000000000000E+000   0.000000000000000E+000
 -1.147067248675788E-003       1.000000657881420   0.000000000000000E+000
 -3.980597305055931E-004  -2.865372022644481E-004       1.000000120408385

gyro scale factor (arc seconds/count)
       1.110936003500000       1.112923749000000       1.111146780000000
```

```
adjusted alignment matrix constants (radians/filter count)
  9.974018026005139E-008   0.000000000000000E+000   0.000000000000000E+000
 -1.144086941533243E-010   9.991870640450003E-008   0.000000000000000E+000
 -3.970254927489533E-011  -2.863040775161492E-011   9.975911585167462E-008

gyro bias corrections (deg/hr)
 -7.900000000000000E-004   7.409000000000000E-002   1.001500000000000E-002

adjusted bias constants (counts/0.0005 sec)
 -1.920002586359602E-005   1.797454678990771E-003   2.433566877636094E-004
*/

        const double mg[3][3] =
                    { {  9.974018026005139E-008,
                         0.000000000000000E+000,
                         0.000000000000000E+000 },
                      { -1.144086941533243E-010,
                         9.991870640450003E-008,
                         0.000000000000000E+000 },
                      { -3.970254927489533E-011,
                        -2.863040775161492E-011,
                         9.975911585167462E-008 } };

        const double b[3] =
        { -1.920002586359602E-005,
           1.797454678990771E-003,
           2.433566877636094E-004 };

            const double gcrc = 9.64921e-1;

          /*********************************************************************/

main()

{
    long in[3];
    long da[3];
    double fda[3];
    double dfda[3];
    unsigned char i;
    double fc[3];
    double fcgb[3];
    double angle[3];
    double q[4];
    double cth;
    double sth;
    double th[4];
    double thn[4];
    double epsilon;


          /* read iae data */
          in[0] = iae_a1;
          in[1] = iae_a2;
          in[2] = iae_a3;


     /*  Filtering
          Perform the following algorithm for each of the three axis, where
          i specifies the axis: */

    i = 0;

    while (i<=2)
      {

          /* figure out delta angles and correct for overflow */
                    da[i] = in[i] - pin[i];

                    /* overflow */
                    if ( da[i] > 32767 )  da[i] = in[i] - pin[i] - 65536;
                    /* underflow */
                    if ( da[i] < -32767 )  da[i] = in[i] - pin[i] + 65536;

                    pin[i] = in[i];

       /* (4) FILTER 4, 3 Stage sum and last output difference */

          fda[i]   = a[i][5] + a[i][9] + a[i][10];
          a[i][10] = a[i][9];
          a[i][9]  = a[i][5];
```

```
        /* (3) FILTER 3, 3 Stage sum. */

          a[i][5]  = a[i][2] + a[i][6] + a[i][7];
          a[i][7]  = a[i][6];
          a[i][6]  = a[i][2];

        /* (2) FILTER 2, 3 Stage sum. */

          a[i][2] = a[i][0] + a[i][3] + a[i][4];
          a[i][4] = a[i][3];
          a[i][3] = a[i][0];

        /*  (1)      FILTER 1, 2 Stage sum. */

          a[i][0] = da[i] + a[i][1];
          a[i][1] = da[i];


        /* coning recovery compensator */

             dfda[i] = fda[i] - pfda[i];

             pfda[i] = fda[i];


             fc[i] = fda[i] + gcrc * dfda[i];


           i = i + 1;

           }


           /* COMPENSATION using test data
           DEFINITIONS:
                          MG(X,Y) is a 3 by 3 matrix of constants
                               provided to compensate for scale factor and
                               and RLG misalignment.
                          B(X) is a three element matrix of constants
                               provided to compensate for gyro bias.
                          DT represents the sample time of the IAE.
                          FCGB(X) represents an array of filtered, crc
                               compensated, scale factor and misalignment
                               compensated, and bias compensated delta angle
                               data for each of the three IMU axes. */

      fc[0] = fc[0] - b[0];
      fc[1] = fc[1] - b[1];
      fc[2] = fc[2] - b[2];

      fcgb[0] = mg[0][0] * fc[0] +
                mg[0][1] * fc[1] +
                mg[0][2] * fc[2];

      fcgb[1] = mg[1][0] * fc[0] +
                mg[1][1] * fc[1] +
                mg[1][2] * fc[2];

      fcgb[2] = mg[2][0] * fc[0] +
                mg[2][1] * fc[1] +
                mg[2][2] * fc[2];


           /* attitude progagation using quaternion representation,
           Honeywell method
           DEFINITONS:
                angle[x] coning compensated angle
                      pb[x] storage variable */

           /* coning compensation */

angle[0] = fcgb[0] + ( 1.0 / 12.0 ) * ( pb[1] * fcgb[2] - pb[2] * fcgb[1] );
angle[1] = fcgb[1] + ( 1.0 / 12.0 ) * ( pb[2] * fcgb[0] - pb[0] * fcgb[2] );
angle[2] = fcgb[2] + ( 1.0 / 12.0 ) * ( pb[0] * fcgb[1] - pb[1] * fcgb[0] );

pb[0]    = fcgb[0];
pb[1]    = fcgb[1];
pb[2]    = fcgb[2];
```

```
        /* quaternion, attitude propagation, Melvin #2 */

    thn[1] = 0.5 * angle[0];
    thn[2] = 0.5 * angle[1];
    thn[3] = 0.5 * angle[2];

    th[1] = 0.5 * ( thn[1] + tho[1] );
    th[2] = 0.5 * ( thn[2] + tho[2] );
    th[3] = 0.5 * ( thn[3] + tho[3] );

    th[0] = sqrt ( th[1] * th[1] + th[2] * th[2] + th[3] * th[3] );

    if ( th[0] == 0.0 )
      {
          cth = 1.0;
          sth = 1.0;
      }
    else
      {
          cth = cos ( th[0] );
          sth = (sin ( th[0] ) ) / th[0];
      }

    q[0] = current_solution[0] * cth - sth * ( th[1] * current_solution[1] +
                                               th[2] * current_solution[2] +
                                               th[3] * current_solution[3] );

    q[1] = current_solution[1] * cth + sth * ( th[1] * current_solution[0] +
                                               th[3] * current_solution[2] -
                                               th[2] * current_solution[3] );

    q[2] = current_solution[2] * cth + sth * ( th[2] * current_solution[0] -
                                               th[3] * current_solution[1] +
                                               th[1] * current_solution[3] );

    q[3] = current_solution[3] * cth + sth * ( th[3] * current_solution[0] +
                                               th[2] * current_solution[1] -
                                               th[1] * current_solution[2] );

    current_solution[0] = q[0];
    current_solution[1] = q[1];
    current_solution[2] = q[2];
    current_solution[3] = q[3];

    tho[1] = thn[1];
    tho[2] = thn[2];
    tho[3] = thn[3];


    solution_count = solution_count + 1;

}
```

VITA

Robert F. Higgins Jr. was the sixth born into the Higgins family in Badkreuznach, W. Germany on October 29, 1962. He graduated from Oxon Hill High School, Oxon Hill, Maryland in 1980. He then attended the University of Maryland in College Park, Maryland. During studies at the University of Maryland, he entered the cooperative education program and worked with the Naval Research Laboratory. In December 1984, he graduated with the degree of Bachelor of Science in Electrical Engineering under the Cooperative Education Program. He continued to work at the Naval Research Laboratory after graduation. After a year off from school, he entered the masters program at the Virginia Polytechnic and State University. He attended classes at the Northern Virginia Telestar campus, leading to the degree of Master of Science.