Digital Signal Processor Software for the High-Energy Transient Experiment Satellite: Video Data Filtering and Storage

by

Scott Anthony Aquinas McDermott

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Master of Engineering

and

Bachelor of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1994

Author .....
Department of Electrical Engineering and Computer Science
May 16, 1994

Certified by .....
Roland K. Vanderspek
Research Physicist
Thesis Supervisor

Accepted by .....
F.R. Morgenthaler
Chair, Department Committee on Graduate Theses

Digital Signal Processor Software for the High-Energy Transient Experiment Satellite: Video Data Filtering and Storage

by

Scott Anthony Aquinas McDermott

Submitted to the Department of Electrical Engineering and Computer Science
on May 16, 1994, in partial fulfillment of the
requirements for the degrees of
Master of Engineering
and
Bachelor of Science in Electrical Engineering and Computer Science

Abstract

The following Thesis is a description of software to control, filter, and store video data representing a field of stellar objects detected at the ultraviolet wavelengths. The High Energy Transient Experiment (HETE) satellite uses two Motorola 56001 Digital Signal Processors as a computational buffer between an Inmos T805 Transputer General Purpose Processor (GPP), running high-level algorithms, and low-level camera electronics, providing raw pixel data. One 56001, the Front-End Processor or FEP, relays command and status information between the GPP and the camera electronics, and filters the video data to provide the GPP with the loci of stars within the fields of view of two ultraviolet CCD cameras. The other 56001, the Back-End Processor or BEP, stores the camera images to answer the GPP's requests for detailed analysis.

Thesis Supervisor: Roland K. Vanderspek
Title: Research Physicist

Acknowledgments

I wish to express my appreciation & admiration to —

My parents, not only for making it possible for me to get this calibre of education, but more for always believing in me — when you were asked what you wanted me to be when I grew up, you responded simply "Happy". I certainly am; learned how from you.

Roland, for three years of support and praise, always encouraging while never pressuring. *Je te remercie.*

Laura, for listening, and smiling, and Being There. For whispering "Theeeeesiiiiiis" when I needed it. For all the love.

Hurl, the most caring, funny, intelligent, and talented bunch of folks I have ever had the honor and pleasure to call my Friends. You have made MIT not only bearable, but magical. All the Best.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The universe is full of magical things patiently waiting for our wits to grow sharper.

— *Eden Phillpots*

1.1 The High-Energy Transient Experiment Satellite

The High-Energy Transient Experiment Satellite, or HETE[1], is the product of an international effort to understand the origin and characteristics of cosmic gamma-ray bursts. It utilizes a distributed-processing architecture to identify *transient events* in the gamma, X, and ultraviolet wavelengths, and then to associate those events which have a common origin. A transient event is a circumstance where a stellar object — a star, or other cosmic entity which gives off radiation — suddenly increases its photonic output, to wit, "becomes very bright very fast". All data relevant to these events are passed down to researchers on Earth.

Gamma-ray bursts offer significant insight into the workings of stars and other stellar sources; however, detectors sensitive to gamma photons (8 keV to 1 MeV)

[1] commonly pronounced "*heh*-tee"

have difficulty accurately determining the origin of the radiation. It is thus nontrivial to identify what stellar object produced a given burst. If however another transient event occurred, at about the same time and from the same general region as the gamma burst, then they could be associated: the two transients could be assumed to have come from the same object. And if the second event could be better localized, then the source of the gamma-ray burst would be better identified than is possible with the gamma detector alone. This is the purpose of the HETE project.

Space science laboratories in the Unites States, France, and Japan are working together to create the HETE satellite for a launch in 1995. Each team is responsible for a different part of the satellite — gamma ray detection, ultraviolet imagery, spacecraft hardware, *et cetera* — and data from the mission will be widely disseminated to researchers interested in the results. This broad distribution of both responsibility and outcome is another unique aspect of the HETE project.

HETE has extensive on-board processor power, centered around four Inmos T805 Transputers, each of which has two Motorola 56001 Digital Signal Processors at its disposal. These twelve microprocessors work in conjunction to perform a great deal of computation within the spacecraft itself, maximizing its ability to deal intelligently with many (potentially time-critical) observation situations, and minimizing the need for extensive communication with the ground. Each of the Transputer-plus-two-DSP processor groups govern a single HETE system. There is one system to control the spacecraft — power, radio, attitude, and the like — one to perform gamma and X band observations, and two to perform ultraviolet band observations. The two ultraviolet (UV) systems are virtually identical, differing primarily in the portion of the sky which they observe.

1.2 Focus of this Thesis

The present Thesis is primarily a presentation of the software designed for the two DSPs within each UV system. These DSPs lie between a charge-coupled device (CCD) camera system sensitive to the ultraviolet wavelengths, and a governing Transputer

referred to as a General Purpose Processor or GPP. The two cameras per system are commanded by, and report their data to, the DSPs; the DSPs are in turn commanded by the GPP above them, which wishes to know information such as:

- The status of the camera electronics
- The loci of stars in the cameras' fields of view
- The pixel values in a section of an image previously read out from the cameras

Since the GPP is intended to focus its resources toward the end of identifying UV transient events and associating them with gamma-ray bursts, the nuts-and-bolts interaction with the camera electronics is left to the DSPs. That is to say, the DSPs take the raw data from the cameras, and process it to produce information more useful to the GPP's work. These DSPs must therefore be programmed to perform the functions the GPP requires. These functions primarily boil down to filtering and storage of video data, and it is a discussion the design and implementation of software to perform these functions that will occupy the majority of the coming pages.

The context of this work will also be described: the value of gamma-ray burst research and the unique abilities of HETE to contribute to it, the computational environment wherein the HETE UV DSPs reside, and this designer's thoughts regarding improvements on, additions to, and limitations of the software suite presented. This document should therefore be a reasonably complete introduction to the HETE project as well as a detailed analysis of the UV system DSP software.

This work was done at the MIT Center for Space Research CCD Lab as a contribution to MIT's part in creating the HETE spacecraft. It was both a joy and a privilege to work with this laboratory and its staff over the past three years.

1.3 Outline of this Document

Following this Introduction, the HETE mission will be described in further detail. The goal of the mission, and the benefits of achieving this goal, are presented; the unique abilities of the HETE satellite that help to gain this end are then offered.

Chapter 3 outlines the HETE processing system, the capabilities and responsibilities of each of its components, and their means of communication. This is followed by a chapter focusing on the UV system in particular, how it uses its resources to achieve its purpose, and how those resources interact. Each of the UV system DSPs is then discussed in detail, both in terms of their design (algorithms and algorithmic concerns, trade-offs, etc.) and their implementation (specific coding issues).

This Thesis concludes with thoughts on future possibilities for the software presented, and a summary of the information conveyed. The code developed is appended to this document.

Chapter 2

The HETE Project

"Going on an Expotition?" said Pooh eagerly. "I don't think I've ever been on one of those. Where are we going to on this Expotition?"

— *A.A. Milne*
Winnie-the-Pooh

2.1 Mission Background

The first gamma-ray burst was detected by Vela satellites designed to detect airborne nuclear tests. Scientists analyzing the Vela data noticed bursts of gamma rays coming from sources other than the Earth or Sun, and concluded that they were cosmic in nature. The bursts were detected in 1968–69, but were not published until 1974.[8] The unique and scientifically exciting characteristics of such high-energy transient events are described by S.E. Woosley *et al.* in the following passage:

> On anyone's list of intriguing astronomical phenomena, x-ray and γ-ray bursts would rank near the top, both because of their enigmatic nature and the enormous instantaneous powers they develop. The γ-ray event of March 5, 1979, for example, was briefly (for $\sim$0.12 s) as bright, in bolometric energy flux, as a star of apparent magnitude *minus* 5 (*i.e.*, brighter than the planet Venus at maximum elongation) and, if situated, as many believe, in the Large Magellanic Cloud, had a luminosity equal to roughly 100 supernovae or ten times the electromagnetic output of the

> entire Milky Way Galaxy! Yet this flux had a rise time of less than 0.2 ms indicating an origin within an extremely small volume. Many other γ-ray bursts have delivered energy fluxes about 100 times small than this, but have maintained their bright output for longer times. X-ray bursts too are known to generate power comparable to the most massive stars ($\sim 10^5$ solar luminosities), but originate from neutron stars that have radii of only $\sim$10 km and can develop these powers in a time less than one second.[11]

A great many questions remain unanswered regarding the origin, purpose, and mechanics of these gamma-ray burst events.[7] Although the consensus was once that they are transmitted from the surfaces of accreting neutron stars, scientists are presently skeptical of this theory, and no single new theory has yet gained wide acceptance. Experiments such as the Burst and Transient Source Experiment (BATSE) on the Gamma-Ray Observatory satellite attempt to provide data to test new models, but so far the results from BATSE have not led to conclusive proofs or disproofs of any hypotheses, and in fact has raised more questions than it has answered. BATSE and related experiments suffer from the *problem of localization*, which makes association of high-energy burst events and the object that cause them difficult. Without a definite idea of the cosmic source of gamma-ray bursts, formulating ideas about their creation is exceptionally troublesome.

2.2 The Problem of Localization

Unlike low-energy optical range photons which can be focused and recorded in a camera, high-energy gamma and X range photons are difficult to record and with present technology impossible to focus: to wit, a detector can tell if it has been hit by high-energy photons, but it is difficult to determine where they came from. The BATSE experiment on the Gamma-Ray Observatory satellite is capable of determining the origin of a gamma-ray burst to an accuracy of $\sim 2^o$ when bombarded with photons (10^{-4} erg/cm^2), losing accuracy exponentially as the flux decreases.

The first proposals to attempt to improve this figure involved coordination among several observation sites separated by vast distances. For instance, localization of six gamma-ray burst events down to 0.05–14 arcmin^2 have been accomplished using

satellites orbiting the Sun, Earth, and Venus.[3] However the timing accuracy required by such a scheme in order to provide good triangulation data is in the millisecond range, making this route both costly and prone to unreliability. Further, the ten satellites used all had missions of their own, and extrasolar gamma-ray observation was a secondary benefit.

The HETE satellite was conceived to be a single satellite focused on the mission of detecting *and localizing* high-energy transient events. To accomplish this end, it utilizes *burster counterparts*: transient events in lower energy bands that are correlated with the high-energy bursts. The case for using optical counterparts is well presented by B. Schaefer:

> It is generally felt that identification of bursters with objects at other wavelengths will probably be required before significant progress can be made in determining their origins. The reason that low energy counterparts are so desirable is that low energy observations are much easier, cheaper, and more sensitive (as energy flux detectors) than gamma-ray observations. In addition, they offer the promise of establishing a much-needed distance scale for bursters. Finally, large data bases already exist for low energy data that can be compared to GRB [Gamma Ray Burst] observations.
>
> Low energy counterparts can be identified when the GRB is bursting or when it is quiescent. ... [T]here are the optical bursts which are expected to be produced at or near the gamma burst source itself by a variety of processes ....
>
> There are several motivations in searching for optical flashes. A light curve could yield the optical duration, the delays with respect to gamma radiation, and the presence of precursors or afterglows. This would aid in choosing between models in which the optical radiation is produced by reprocessing in an accretion disk or in the atmosphere of a stellar companion on one hand, and those in which it is produced by cyclotron repressing. A precise location can be measured which will allow deep follow-up searches for the quiescent counterpart. The fraction of the energy emitted in the optical is another useful quantity for comparison with theory, as would be polarization and color of the flash. In addition, a recurrence time scale may be much easier to measure with optical techniques, although it is not yet clear whether optical and gamma ray recurrence times are the same. Each of the above mentioned quantities can be compared with theory and hence can serve to distinguish between models.[4]

2.3 Mission Goal

It was in this purpose that the HETE[1] project was conceived by the participants of the 1983 conference of the American Institute of Physics, "High Energy Transients in Astrophysics".[11] It was meant to be small and inexpensive, utilizing current and proven technology to provide a relatively quick creation time and reliable system performance. To accomplish its mission, the HETE spacecraft would possess a gamma-ray detector, an X-ray detector, and several optical cameras. It would perform the detection and localization of high-energy transients on-board, making time synchronization among detectors both simpler and less important (as it would not be using time data to triangulate, but only to associate) than with the multi-craft scheme; and the several astronomical units of detector separation required by the latter is rendered unnecessary by the use of optical burst counterparts as a localization device.

The scientific goal for the HETE mission, as defined by those who conceived it, is to answer the following questions:[11]

- What physical processes operate in a γ-ray burst that allow the production of non-thermal radiation for a period (typically seconds to minutes) that is so much longer than typical neutron star dynamic time scales ($\sim$1 ms)? Which conditions lead to γ-ray bursts and which lead to X-ray bursts? What physical circumstances produce a γ-ray spectrum as opposed to the softer spectrum of an X-ray burst or X-ray pulsar? How are line features, both absorption and emission, produced?

- What are the properties of those neutron stars that produce high-energy transients? What are their radii, masses, accretion rates, field strengths, and internal temperatures and how do these parameters carry for the different categories of transients? What is their evolutionary status?

[1] Originally "High Energy Transient Explorer"; "Explorer" has since been replaced by "Experiment"

- How do repeated thermonuclear explosions on the surface of a neutron star interact with the interior temperature and a variable accretion rate to produce, from the same source, X-ray bursts that are sometimes regular in occurrence, sometimes erratic, and that sometimes cease altogether. How common are instances of recurrent flashes with intervals so short as to be incompatible with simple thermonuclear models?

- Do observable optical flashes always accompany γ-ray bursts? How is the optical flash produced? Is it an extension of the γ-ray mechanism or fluorescence of a disk or companion star? How does the time history of the optical flash compare with that of the γ-ray burst itself? Can the optical emission be used to obtain the properties of the binary star system or to indicate the presence of a strong magnetic field? Are there other classes of short lived astronomical phenomena that emit chiefly in the optical (or ultraviolet or near infrared)? *Ultraviolet was chosen for the HETE mission.*

- Are the basic processes that produce γ-rays in solar flares also operating elsewhere in the cosmos where highly magnetized plasmas are present?

2.4 HETE Instrumentation

HETE has one omnidirectional gamma-ray spectrometer, one wide-field X-ray monitor, and four ultraviolet cameras. It is also an autonomous spacecraft, meaning it has attitude control capability, a (solar) power acquisition and regulation system, a radio link with the Earth, and monitors for potentially dangerous situations such as depressions in the protective Terran radiation belts.

The omnidirectional γ-ray spectrometer is sensitive to photons in the 6keV to >1MeV range. It can resolve transient events down to a separation of 4ms. It is based on proven technology developed in France for the Soviet *Phobos* probes, which uses an array of four detectors, covering a region of the sky 2π steradians in area.

The wide-field X-ray monitor is sensitive to photons in the 2 to 25keV range. It

can resolve transient events down to a separation of 1ms. Note that this energy range partially overlaps that of the γ-ray detector, but this device offers a higher temporal resolution. The monitor observes a total of $\sim$2 steradians of sky area using two units, and can determine an X-ray source's origin to $\pm$6 arcminutes of accuracy.

Both the γ-ray spectrometer and the wide-field X-ray monitor communicate with two Motorola 56001 Digial Signal Processors (DSPs), which use their high-speed mathematical calculation abilities to process this raw detector data to find salient information. This information is provided to an Inmos T805 Transputer, called a General Purpose Processor or GPP. This GPP determines whether a burst has occurred in either of these bandwidths. The GPP, two DSPs, γ-ray spectometer, and X-ray monitor comprise the *Gamma / X system.*

The four ultraviolet cameras, divided into two pairs, are sensitive to photons in the 5 to 7eV range. They uses state-of-the-art charge-coupled device technology similar to that in consumer use to provide inexpensive, reliable detection. The UV system can resolve events from $\sim$0.5s to $\sim$6s, depending on how much of each camera's image is read out. Normal operations read out full frames from each camera in a data-compressed fashion every 2.0s. The four cameras, in total, view $\sim$1.7 steradians of the sky, and have $\pm$3 arcseconds of precision of localization (1σ).

Each pair of cameras is connected to readout electronics named the *Lasagna Box.* The Lasagna Box is commanded by, and reports its data to, two Motorola 56001 DSPs, called the *Front-End Processor* (FEP) and the *Back-End Processor* (BEP). They too use their ability at high-speed calculation to extract useful information from the raw pixel data, and provide this information to a governing Inmos T805 Transputer GPP. This GPP searches for transient events in the ultraviolet region and responds to burst announcements from the Gamma / X system to provide integrated reports on Gamma / X / UV burst activity. A GPP, two DSPs, and two ultraviolet cameras compose a *UV system*; there are two UV systems on-board HETE.

The spacecraft itself is largely cylindrical, 48cm in diameter and 90cm in height. It uses a single momentum wheel for attitude control. Its attitude is kept so the science instruments are always on the anti-Sun side of the vehicle, with the CCD

cameras radiatively cooled to –60°C for improved quantum efficiency. It carries four solar panels which extend after deployment to provide $0.9m^2$ of surface area, providing the 30 watts necessary for the scientific instruments and 16 watts necessary for spacecraft infrastructure. It is expected to transceive 10 kilobits of data per second with the Earth (orbit average), over an S-band radio link. It is projected to weigh approximately 250 pounds.

A fourth Inmos T805 Transputer GPP oversees two more Motorola 56001 DSPs, which in turn interact with the electronics controlling the spacecraft's power, attitude, communications with Earth, and so forth. These three processors and the spacecraft hardware they govern are collectively called the *Spaceraft Control system.*

HETE will be inserted by a Pegasus launch vehicle (shared with one other satellite) into 550nm high, 38° nominal inclination circular orbit above the Earth. It is expected to perform its mission for one to two years.

2.5 A Day in the Life of HETE

This section relates how the mission described above is accomplished by HETE, by describing the typical daily behavior of the spacecraft, in particular the UV system. The flow of requests and replies among the spacecraft processors is shown in Figure 2-1. The following chapters detail how the actions described here are performed.

HETE's detectors — the omnidirectional γ-ray spectrometer, the wide-field X-ray monitor, and the four ultraviolet cameras — always point in the anti-Solar direction. Thus for half of its orbit, these detectors are pointed toward the sunlit Earth, and for the other half, toward deep space. During the former, HETE is performing *daytime operations*, and during the latter, *nighttime operations.* Daytime operations are minimal: the detectors and most of the DSPs (those not in the Spacecraft Control system) are shut down to conserve power, and the GPPs spend most of their time idling. Once nighttime operations commence, however, HETE "wakes up" to work.

The GPPs in the Gamma / X and UV systems request the Spacecraft Control system provide power to their respective DSPs and detectors. The DSPs are booted

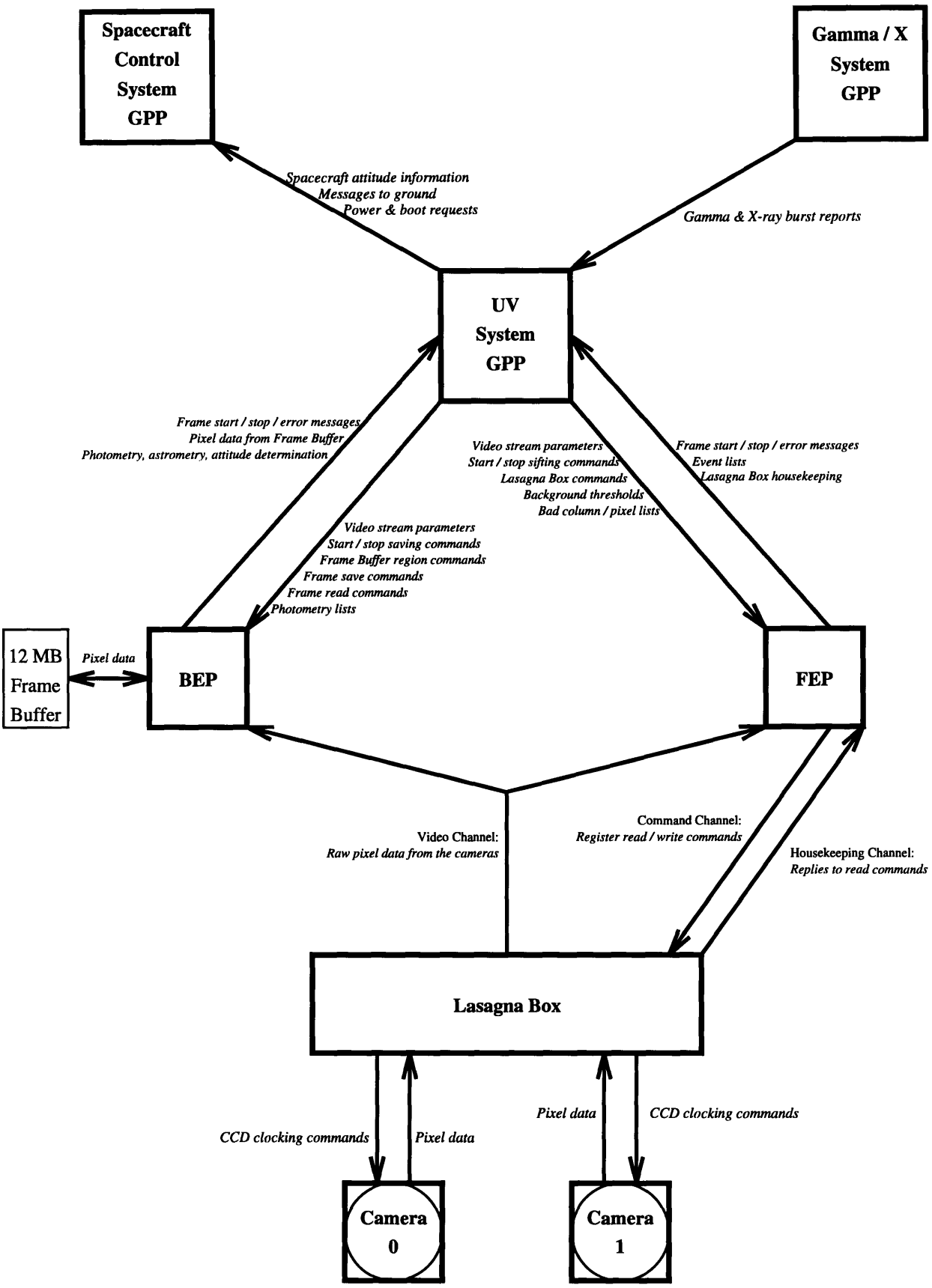


Figure 2-1: Information Flow In and Around the UV System

and the detector electronics initialized by the GPPs above them. All four HETE systems are now active and looking for burst events. We now focus on the behavior of one UV system.

The UV system GPP uses its FEP to command the Lasagna Box below it, and to receive status ("housekeeping") information back from this Lasagna Box. With the system initialized for nighttime operations, the GPP commands the Lasagna Box to begin reading out pixel data from both cameras; both the FEP and the BEP receive this pixel data.

The FEP processes this data as it arrives, searching for stellar objects (*events*). An event is defined as a pixel which is not part of the background — the dark sky — and has the highest pixel value in a 3-by-3-pixel area around it. Hence each ultraviolet-emitting stellar object visible to the HETE cameras should be recorded as a single event by the FEP. When an entire frame of data has been received by the FEP from the Lasagna Box, it reports the coordinates of all the events it found (the *event list*) to the GPP.

The BEP does not process pixel data as it arrives, but rather stores it in a *Frame Buffer*. Several full-size frames and many *subarrays* — rectangular portions of a camera's full image — can be stored in the Frame Buffer, with new data overwriting old as necessary. The BEP uses the 56001's strengths in mathematical calculation to perform astrometry, 3x3-summation photometry, and spacecraft attitude determination, using pixel data in the Frame Buffer.

These are the resources now available to the GPP, once it has commanded the Lasagna Box to begin generating frames. With every frame, the GPP receives an event list from the FEP. The GPP relays this list to the BEP, requesting photometry on each of these event coordinates, as well as an updated spacecraft attitude determination. The latter is forwarded to the Spacecraft Control system; the former is used to detect any *new sources* — events which were reported by the FEP for the first time this frame — and brightenings in *field sources* — sources which have become brighter since previous frames. These are potential UV burst sources.

Frame after frame, this process repeats. The Lasagna Box is reading out frame n;

the FEP is processing frame n for events; the BEP is storing frame n, while performing photometry on the events in frame $n-1$. The GPP receives the photometry from frame $n-1$ from the BEP and compares it to previous photometry to determine new sources and field sources. The GPP then receives the event list for frame n from the FEP, which it passes to the BEP for photometry on frame n. Now the Lasagna Box is reading out frame $n+1$; the FEP is processing frame $n+1$ for events; and the BEP is storing frame $n+1$, while performing photometry on the events in frame n.

This process can be interrupted by one of two occurrences within the Gamma / X system: the detection of either a gamma- or an X-ray burst. For both types, the full frames already in the Frame Buffer are of interest, as well as the frames about to be read out for the next few minutes. All of the frames currently in the Frame Buffer — that is to say, all of the frames preceding the gamma- or X-ray burst report — are sent to the ground immediately, and each frame following the report is sent to the ground after the frame is read out by the Lasagna Box and stored by the BEP.

For a gamma-ray burst, these following frames are still full-size, because the γ-ray spectrometer cannot provide localization information, meaning the burst may have come from anywhere within all four UV cameras' fields of view. An X-ray burst, however, can be localized down to ± 6 arcminutes of area: either a tall narrow box, or a short wide box, or a small squarish box. The GPP then commands that only a *subarray* of the full frame be read out and saved, potentially decreasing the time the Lasagna Box takes to read out a frame, and increasing the number of frames that can fit in the Frame Buffer.

The GPP resumes normal operation — saving full frames, not reporting them to the ground — some minutes after the Gamma / X system reports that the gamma- or X-ray burst has ceased. Note that the UV GPP can compare its own list of field sources to the reports from the Gamma / X system, correlating the information and making an on-board estimate of the stellar object which originated the high-energy burst. It can communicate this information to the ground for potentially immediate reaction from ground-based observatories.

Nighttime operations proceed in this fashion, with the UV system processing

full frames until the Gamma / X system reports a burst, at which point the UV GPP dumps the burst-preceding and burst-following camera frames to the ground; the latter frames may be smaller if the X-ray monitor was able to perform some localization that the UV system could take advantage of. As daytime approaches, the UV GPP dumps the photometry information it gathered through the night, as well as any previously saved frames of interest. The detectors and DSPs shut down, and daytime operations resume.

It is important to note the following unique strengths of HETE:

- Gamma, X, and UV detectors are all on the same spacecraft. Thus communication among them can be immediate, compared to disparate spacecraft trying to coordinate their observations.
- All of the detectors are aimed at the same portion of the sky. If a burst is seen in the *overlap region* of one detector, any counterparts this burst has in the other wavelengths will be seen by the other detectors.
- *Predecessor information* — information on the state of the sky leading up to a burst — is held in the Frame Buffer.

2.6 The HETE Team

HETE is being jointly created by five organizations:

- At the Centre d'Étude Spatiale de Rayonnements, in Toulouse, France: Gamma-ray detectors and software
- At the Institute of Physical and Chemical Research, near Tokyo, Japan: X-ray detectors and software
- At Los Alamos National Laboratories, in Los Alamos, New Mexico, USA: X-ray transient detection software and coded aperture
- At MIT's Center for Space Research, in Cambridge, Massachusetts, USA: Ultraviolet system components (cameras and electronics) and software

- At AeroAstro Corporation, in Herndon, Virginia, USA: Spacecraft body and computer hardware

MIT is also responsible for the archival of information downloaded from the satellite.

This level of distributed responsibility, both conceptual and geographical, is unique in satellite design. Team members make extensive use of current communication technology — primarily electronic mail, but also teleconferencing, remote file transfers, and the like — to keep the project coordinated.

Distribution of resultant data is also unique in the HETE project, as ground-based researchers can use the Internet to transfer data from MIT's archive to their own facility for analysis. They can further set up a small, inexpensive, omnidirectional VHF receiver to acquire data summaries directly from the satellite, for possible real-time coordination with local observatories.

This large array of persons from around the globe — the design team, from France, Japan, and the United States, as well as all the physicists researching high-energy transient phenomena — compose the group that will take HETE from its conception to its goal.

Chapter 3

The HETE Computer System

"Hello, computer?"

— *Montgomery Scott*
Star Trek IV

3.1 Overall Structure and Relations

The HETE computer system is centered around four Inmos T805 Transputers, each governing an independent HETE system, and communicating as peers to accomplish the satellite's mission. Each Transputer, termed a General Purpose Processor or GPP, has two Motorola 56001 Digital Signal Processors (DSPs) at its disposal; these DSPs in turn interface with the spacecraft externals — detectors, cameras, radio links, and the like. The DSPs thus provide a computation buffer between the GPP, and the low-level electronics which run the spacecraft and perform the actual astronomical observations. Figure 3-1 shows the architecture connecting these Transputers, DSPs, and low-level electronics.

The four systems on board the spacecraft are:

1. Spacecraft Control. Responsible for power management, attitude control, radio communication, and downlink message prioritization and buffering.

2. Gamma / X. Responsible for detecting transient events in the Gamma and X bands, and localizing them as well as possible. Passes this information to the UV systems for further localization, and to ground-based researchers.

3. UV System I. Responsible for detecting transient events in the ultraviolet band, and associating them with events reported by the Gamma / X system. Passes this information to ground-based researchers. Uses two UV cameras with partially overlapping fields of view.

4. UV System II. Same responsibilities as UV System I, but with cameras that observe a different portion of the sky.

Each system is on a separate processor board; the four boards are connected by a Host Interface Bus. Each board contains one GPP, two DSPs, Host Bus Interface logic to allow these processors to communicate over the Host Interface Bus, and Host Control Interface logic to control communication between the GPP, and its DSPs and the Host Bus Interface. These systems communicate over the Host Interface Bus: any GPP can communicate directly with the other three. Communications are conducted under the Inter-Process Protocol, or IPP, described in section 3.3.

Communication with the Earth takes place via a line-of-sight radio link with a set of *primary* and *secondary ground stations.* Three primary ground stations, in Arizona, Japan, and Italy, both receive from and transmit to the satellite; numerous (22 at last count) secondary ground stations scattered across the planet use simple omnidirectional receivers to accept selected downlink data as well, associating them with local observatories seeking curiosities in the visible, infrared, and radio bands. This is the first step in the dissemination of satellite information: those who wish to receive data as soon as possible when the satellite is overhead, need only acquire a small receiving system. The second step is prodigious use of the Internet, whereby data received from the satellite by the primary ground stations is sent to MIT for archival, and project researchers use standard Internet tools to acquire this information from MIT.

These four processor boards governing their respective systems, together with communication with and around Earth, define the HETE processing system.

3.2 Processor Capabilities

The Inmos T805 Transputer is designed to be capable of running a set of independent processes simultaneously, as well as communicating easily and efficiently with other devices of the same type. It is a 32-bit device running at 20 MHz. It has 500 kilobytes (131,072 words) of ROM, and 4 Megabytes (1,048,576 words) of RAM, both of which are two-bit error-checked. That is, words with 0 or 1 bits in error will be corrected in hardware and will not affect program execution; words with 2 bits in error will generate an exception, being a recognized and localized but not immediately correctable problem.

The ease of inter-processor communication facilitates coordination among systems. A gamma-ray burst occurs, possibly partially localizable by the X-ray detector, and is detected by the Gamma / X system; the Gamma / X GPP sends a message to the UV system viewing this portion of the sky, and this UV GPP attempts to associate the gamma- and X-ray bursts with a UV transient. An association is found, and the UV GPP decides to inform ground researchers. It sends a message destined for the ground, which is first routed to the Spacecraft Control GPP, on its way down the radio link to Earth. A land-based researcher wishes to get the UV frames corresponding to this transient: a message goes up the radio, into the Spacecraft Control GPP, over to the UV GPP, which responds with a set of messages headed back toward the Spacecraft Control GPP on the way down to the ground. And so on: we see that this Transputer-to-Transputer message passing is a major component of the HETE spacecraft's ability to perform its mission.

In normal operation, one GPP governs the two DSPs "beneath" it in the processing chain. If however one GPP should fail in flight, another can take over and command these DSPs remotely (*i.e.* from another processor board within the spacecraft), by using the Host Control Interface to "lock out" the faulty GPP from communicating

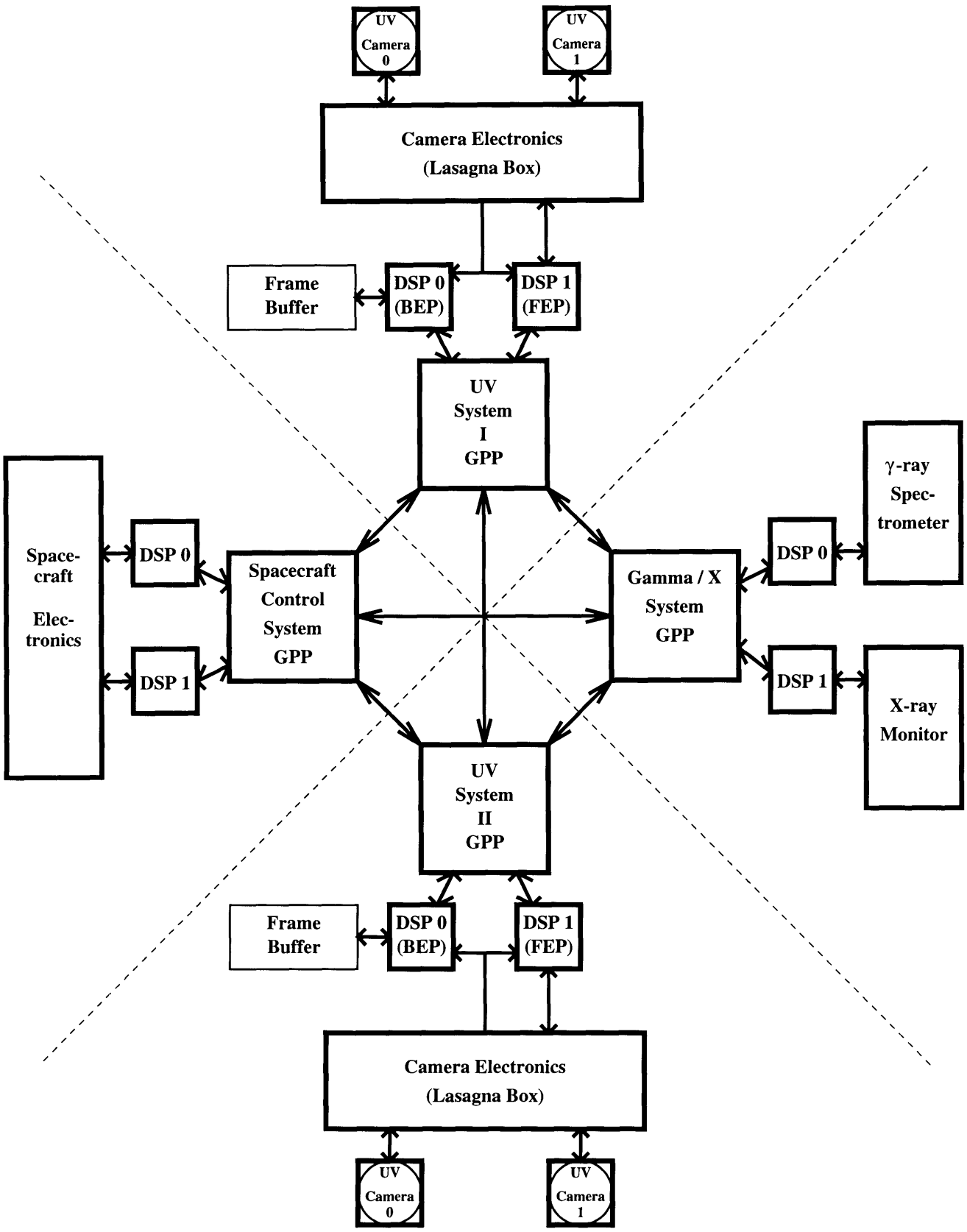


Figure 3-1: The HETE Processing System

with its DSPs. Direct Memory Access (DMA) will not be available between the remote GPP and the orphaned DSPs, meaning communication will be markedly slower; and the spacecraft bus will have much more traffic than it ought to under normal circumstances; but an entire system (UV, Gamma / X, or even Spacecraft Control, theoretically) need not fail because its Transputer has died. This is particularly applicable to the UV system, which has two identical instantiations on HETE: each UV GPP is *de facto* capable of performing the other's functions. All it need do is launch a new set of processes identical to the ones it already has running, with a few parameters changed — specifically, those relating to the orientation of the cameras, and the IPP addresses of the new DSPs. The DSPs, in turn, are programmed to send messages as replies to processes which request various kinds of information, and never actually know with whom they are communicating. This takes advantage of the "location ignorance" feature of IPP (described in section 3.3): neither the remote GPP nor its new charges need to know anything explicit about the physical or electronic location of the other; the former needs only know the latter's addresses, and the underlying IPP system takes care of routing commands and replies where they need to go.

The Motorola 56001 Digital Signal Processor is a single-process device with extensive on-board serial and host interface capabilities. It is a 24-bit device running at 20 MHz. It has a small (512 word) boot ROM, 32 kilowords of program RAM (referred to as "P:" space), and 64 kilowords of data RAM (32K in "X:" data space, and 32K in "Y:" data space). Two DSPs on HETE — one each for the two UV systems — have a 4 Megaword *Frame Buffer* as well. HETE DSP memory is not error-checked.

The DSPs are intended to take raw low-level data — Gamma or X detector outputs, pixel readouts from the UV cameras, signals from spacecraft utility electronics — and turn them into more high-level information that is immediately useful to the GPPs controlling them. Two serial communication systems are available to the DSP: a synchronous serial interface, where there is an explicit and distinct synchronization signal passed from the transmitter to the receiver; and an asynchronous serial

interface, where the synchronization is included in the data stream itself, as in the EIA232 (RS232) protocol. Each of these systems is capable of independent transmission and reception. It is through these two transmission channels and two reception channels per DSP, that communication is established with the low-level electronics. Communication with the GPP is accomplished via the DSP's 8-bit-wide host interface system.

3.3 The Inter-Process Protocol

Processes communicate via IPP, the Inter-Process Protocol. Recall that the Transputers (GPPs) run several processes each, and each DSP is considered a single process. Processes can also be present on Earthside computers.

The crucial element of IPP is *location ignorance.* No sending process need know anything specific about the receiving process — what kind of device it is running on, its location, how to get the message there — the method of transmission is the same, whether the destination is on the same on-board processor, or down on Earth halfway around the globe. Each processor has an underlying IPP system which completely abstracts away these concerns; this underlying system knows, for example, that messages to off-satellite processes should be routed first to the Spacecraft Control system, and messages to a process on the same processor board need not go out onto the Host Interface Bus.

When one process wishes to send an *IPP message* to another, it needs the following information:

- The recipient's *IPP address*
- The sender's own IPP address
- The message *type*
- The message *priority* (0–255, 0 being highest priority)
- The message itself: up to 512 16-bit unsigned words

A *Constellation Name Server* keeps track of each process's address. When a process becomes active, it registers with the Constellation Name Server, and gets its own IPP address in return. When such a process wishes to communicate with another, it asks the Constellation Name Server for that process's address, given its name (number actually): rather like copying information from a White Pages directory into a personal address book.

Two processes which communicate with each other agree on a set of message types so the recipient will know what the data in a message means. For instance, the Gamma / X and UV systems would both recognize a X_EVENT_FOUND message type: the former would know how to insert the relevant information — the time of the transient event, the best estimate of its source location, and so forth — into the message, and the latter would in turn be able to extract this information and use it.

The underlying IPP routing system will occasionally find itself with several, even many, messages requiring transmission. This is most likely true for inter-system messages when the Host Interface Bus is experiencing a lot of traffic, and almost definitely true for the Spacecraft Control system holding on to downlink messages until a primary ground station comes into view. To ensure that time-critical messages do not get delayed by less important communiqués, each message has a priority number associated with it, and higher priority messages (those with lower numbers) are promoted to the head of any transmission queues. In this fashion a X_EVENT_FOUND message can get from the Gamma / X system to one of the UV systems, even if the other UV system is generating a great deal of traffic on the Host Interface Bus, say by sending a large frame of image data to the Spacecraft Control system for eventual transmission to the ground.

The data within the message is a list of 512 or fewer "unsigned shorts", that is, 16-bit words representing the numbers from 0 to 65535. Data that are greater than 16 bits wide are customarily sent least-significant-word-first, so a number such as \$12345678 is packaged as message[n] = \$5678, message[$n+1$] = \$1234. If a long list of data that may span several messages needs to be sent, one of the words in each message is often a pair of flags indicating whether this message is the first or last

in the series. So if bit 0 ($0001) of message[0] is the "first message" flag, and bit 1 ($0002) of message[0] is the "last message" flag, then the recipient will usually receive a series of messages (of the same type) where message[0] reads 1—0—0...0—2. If the data can be fit into a single message, message[0] would be $0003 (= $0001 OR $0002).

With this information collected, the sender may compose an IPP message and send it, trusting that the information will eventually get to its destination, regardless of where it is. The recipient, further, can offer a reply to the sender without ever having to know the sender's name, because the sender's address is included in the message. This takes location ignorance to an even higher level, as a process may respond to a command without knowing or needing to know who sent it. This feature is especially useful in the HETE DSPs, as indicated in section 3.2 and chapters 5 and 7.

Chapter 4

The UV System

[Scientists] are Peeping Toms at the keyhole of eternity.

— *Arthur Koestler*

4.1 UV System Components

Arguably, the first component of the UV system are the stellar objects themselves, any and everything in the observable universe which sends out photons between 220nm and 310nm (5 to 7 eV, 9.7×10^{14} to 1.4×10^{15} Hz). On the HETE spacecraft, four charge-coupled device (CCD) cameras stand ready to receive these photons. These four cameras are divided into two pairs; each pair is connected to an identical set of electronics. For simplicity, future references to "the UV system" and its components will refer to the system associated with one of these pairs, with the understanding that there is a duplicate system on the spacecraft, differing only in the fact that its two cameras are aimed elsewhere.

The two UV cameras have a $\sim$10% overlapping field of view. That is, $\sim$10% of the image reported by one camera is also reported by the other. The overlapping regions are oriented to cover the areas of the sky to which the γ and X detectors are most sensitive. Charge-coupled devices require a good deal of support electronics to pull out pixel information. On HETE, the support electronics are contained in the

Lasagna Box. The Lasagna Box is a highly pipelined and parallelized combination of digital/analog converters, serial/parallel multiplexers and decoders, and command interpretation logic, designed to make camera images available to the rest of the UV system as rapidly, accurately, and robustly as possible. The Lasagna Box can read out all the data from the two 1024x1024-pixel cameras to 12-bit accuracy in 6.3 seconds; or, from a single camera to 16-bit accuracy in the same amount of time. Usually, the Lasagna Box will be configured to read out both cameras to 12-bit accuracy in a "2x2-summed" fashion, where the 1024x1024 pixel array is grouped into an grid of 2x2 squares, and the pixels within each square are summed to form a new image 512 pixels on a side. The Lasagna Box thus reports a 512x512-pixel[1] image from each of the two cameras, to 12-bit accuracy, every 1.6 seconds: this is referred to in following chapters as a "normal-mode" frame.

The Lasagna Box sends these pixel values to a pair of Motorola 56001 digital signal processors (DSPs) named the Front-End Processor (FEP) and the Back-End Processor (BEP). These DSPs report to the General-Purpose Processor (GPP), an Inmos T805 transputer which heads the HETE UV system. The GPP relies on the DSPs to perform a great deal of the initial computation on the video data sent by the Lasagna Box, transforming the long stream of pixel values into a collection of more useful information, such as the coordinates of stars within the image, the pixel intensities of some interesting portion of the image, and the like. The GPP itself is responsible for the highest level of on-board UV data analysis, most importantly the identification and localization of transient events, coordinated with HETE's Gamma / X system; and also for communication of important data to ground-based researchers through the HETE Spacecraft Control system, which has jurisdiction over the radio link. This division of labor takes advantage of the DSPs' ability to perform long and complex mathematical calculations quickly and efficiently, and the Transputer's ability to run several processes simultaneously while communicating easily with other Transputers.

[1]Hardware specifics within the Lasagna Box and CCD cameras may change the dimensions slightly

Channel	Name	Source	Rate	Purpose
Command	STD	DSP	500 kHz (2μs/bit)	24-bit command words, MSB-first
	SCK	DSP	500 kHz (2μs/cycle)	Bit-wide clock
	SC2	DSP	20.8 kHz (48μs/sync)	Bit-wide sync pulse during last bit of command word
Housekeeping	RXD	LBox	31.25 kHz (32μs/bit)	8-bit status words, EIA232 format
	SCLK	LBox	500 kHz (2μs/cycle)	16x sampling clock
Video	SRD	LBox	4 MHz (250ns/bit)	24-bit video data, MSB-first
	SC0	LBox	4 MHz (250ns/cycle)	Bit-wide clock
	SC1	LBox	167 kHz (6μs/sync)	Bit-wide sync pulse during last bit of video word
Reset	TXD	DSP	Aperiodic	Master reset signal, active high

Table 4.1: Lasagna Box / DSP Communication Signals

The GPP and the DSPs are electronically linked by an 8-bit-wide DMA bus; however, they communicate via the Inter-Process Protocol (see section 3.3), which means the software developer need not be concerned with the hardware specifics at this level. The hardware specifics are of great importance, however, with regard to communication with the Lasagna Box. The Lasagna Box communicates with the DSPs via three serial links:

- A *command channel*, a 24-bit-wide synchronous serial interface running at 500 kHz (48μs per word) from which the Lasagna Box accepts instructions;
- A *housekeeping channel*, an 8-bit-wide asynchronous serial interface running at 31.25 kHz (320μs per word, including start and end bits) through which the Lasagna Box sends responses (two words per response) to queries;
- A *video channel*, a 24-bit-wide synchronous serial interface running at 4 MHz (6μs per word) through which the Lasagna Box sends pixel data from the cameras.

Table 4.1 describes the actual signals which travel between the DSPs and the Lasagna Box.

The essential links in the UV system are thus Ground Researchers $\rightleftharpoons^{IPP/radio}$ Satellite $\rightleftharpoons^{IPP}$ UV GPP $\rightleftharpoons^{IPP}$ DSPs (FEP + BEP) $\rightleftharpoons^{serial}$ Lasagna Box $\rightleftharpoons^{parallel}$ cameras (x2) $\leftarrow^{photons}$ UV-transmitting stellar objects, with commands flowing down the chain left to right, and information flowing up the chain right to left. The FEP is in charge of relaying commands and status reports between the GPP and the Lasagna

Box, as well as producing event lists for the GPP; the BEP stores frame data for later analysis. See section 2.5, especially Figure 2-1, for an overview description of the use of these devices.

4.2 Controlling Transputer: The GPP

Please see section 3.2 for information on the Inmos T805's capabilities, and the interrelation of these devices on the HETE spacecraft. In the UV system, these capabilities are put to use to identify transient events — stellar objects whose ultraviolet emissions increase by a statistically significant amount on timescales ranging from 2 to 1000 seconds — in the ultraviolet band, and associating them with transients reported by the Gamma / X system. To this end it uses the DSPs, and through them the Lasagna Box and the cameras, to

- Locate all stellar objects (viz. stars) within the field of view of the two cameras. *(FEP function)*
- Perform photometry on these stellar objects in each frame. *(BEP function)*
- Compare the photometry of stellar objects from successive frames to determine if any of the objects are displaying transient behavior. *(GPP function)*
- Narrow the field of interest within the camera images and grab a succession of frames, if a localized X-ray burst was reported by the Gamma / X system. *(GPP function via the FEP, impacting the Lasagna Box and BEP)*
- Monitor the status of, and modify the parameters used by, the DSPs or the Lasagna Box to keep the video processing operating efficiently *(GPP function via the FEP, impacting the Lasagna Box and BEP)*

In normal operation, the GPP commands the UV subsystems to observe the entire field of view available to the two cameras. It takes the list of star locations for each frame, and attempts to associate each one with a star located in previous frames;

it can then compare the apparent magnitude of these stars at different times, to determine if a transient event has occurred.

The Gamma / X system will from time to time report that a burst has occurred in these wavelengths. Included in the message will be the best estimate of the portion of the sky where the burst originated (within ± 6 arcmin if the X-ray monitor could localize the event). The GPP then narrows the field of interest within the cameras' fields of view to include only this portion of the sky. This buys the UV system higher pixel-intensity accuracy, and the ability to save more frames separated by a smaller camera readout time. With this high-resolution (in both time and pixel intensity) series of frames, the GPP can gather a great deal of information on the exact location (within ± 3 arcsec) and brightness-change characteristics of this event.

The GPP is then responsible for relaying important information — details on the occurrence of transient events, relevant images, data on system status, *et cetera* — to researchers and project staff on the ground.

4.3 Dual DSP System: The FEP and BEP

This Thesis is focused on this component of the HETE UV system. In particular, a software suite to govern these two processors has been developed, and the following chapters will present the algorithmic and implentation concerns encountered along the way, with the solutions arrived at by this developer and his colleagues. To set the stage: the DSPs both have the following responsibilities —

- Function autonomously once booted, reporting regularly that it is still operating
- Communicate with the GPP and beyond via IPP, following the principle of "location ignorance": a message from a DSP is sent to the process which requested the type of information the message contains, without the DSP being aware of any specifics about the recipient[2]

[2]For convenience, in this document the recipient is generally assumed to be the GPP, as this is the normal situation in-flight. So statements like "report to the GPP" actually mean "report to the process which asked for this kind of information".

- Monitor the video channel from the Lasagna Box
- Advise the GPP when a frame is about to be processed, and again when processing is complete
- Warn the GPP of any recognized error situations, such as a hardware exception, a problem with the data from the Lasagna Box, and so forth

In addition, each DSP has these specific tasks:

The FEP

- Send commands from the GPP to the Lasagna Box, and relay responses back from the Lasagna Box to the GPP
- Report *events* as requested. An event is a feature within a camera frame which is determined to be a stellar object; so the FEP tells the GPP where all of the stars are in an image.

The BEP

- Store frames or portions of frames as requested
- Retrieve previously stored pixel data, presenting it in a form useful for analysis
- Analyze pixel data: perform astrometry, photometry, and spacecraft attitude determination using previously stored pixel data[3]

The BEP has a 12 Megabyte *Frame Buffer*, organized as 4,194,304 24-bit words, in which to store frames. This means that up to 16 normal-mode frames (32 camera images) may be stored in the Frame Buffer; or 8 normal-mode frames plus, say, 209 100x100-pixel camera images at 16-bit resolution, or 30 50x500 camera images followed by 17 200x500 two-camera frames at 12-bit resolution; and so on. In addition, both DSPs have 32 kilowords available in each of the 56001's memory spaces, P: (program), X: (data), and Y: (data). All of these spaces utilize a 24-bit word. This

[3]Not implemented in this release, and not discussed further in this Thesis

means that since the FEP does not have a Frame Buffer, it has only 64 kilowords of data memory to work with, and hence cannot store an entire frame: its event-finding system must be able to operate in real time, as the pixels are arriving, without the option of putting a frame aside when it's busy and going back later.

The code for the DSPs had to be, in descending priority order: fast, to keep up with the stream of incoming video data; robust, because a satellite must deal intelligently with unexpected circumstances without the benefit of a human supervisor; and readable, so that it may be understood by the members of the team who will use, augment, and support it. The DSPs must know how to communicate with the GPP on one side, and the Lasagna Box on the other: the former requires the software to be flexible, to respond to a variety of requests, and "intelligent", to take raw camera data and create information useful to the GPP; the latter requries the software to be not only fast, as stated above, but also efficient, to recognize the "meaning" of the data in real-time and deal with it accordingly.

To this end, three software techniques are used extensively in both DSPs. Robustness and readability are aided by the use of a *mainloop*, where a single "main" function calls a series of functions — say, **try_get_message()** to check if the GPP has sent a command, then **try_proc_vid()** to check if something should be done with the video data that just arrived, then **check_buddy_time()** to inform the GPP that the DSP is still alive and functioning — over and over in an endless loop. Each of these functions is constructed so that it returns within a reasonable amount of time to let the other functions operate; if it has not finished what it was doing, it picks up again the next time the mainloop calls it. In this way individual functions are less apt to swallow up all of the DSP's time should an unexpected situation occur — for instance, if the FEP were to send a command to the Lasagna Box and then sit in a wait loop until the response came back, a malfunctioning Lasagna Box could prevent the FEP from dealing with an important message from the GPP. Under the mainloop scheme, however, the FEP's Lasagna Box communication functions would send the command, and simply return. Then each time around the mainloop, these functions would check if a response had arrived yet, without interfering with the rest

of the FEP functions, like communication with the GPP.

The 56001 has several features which make *circular buffers* easy to implement. When a stream of data needs to be passed from one routine to another, a circular buffer may be used to limit the need for coordination between the sender and receiver, improving speed and efficiency. It is like two runners jogging around a track, where the sender is dropping crumbs, and the receiver, some paces behind, picking up the crumbs and consuming them. Around and around the track they go: the receiver need only be concerned with (1) not bumping into the back of the sender and (2) not getting a full lap behind the sender (which would mean the sender would start trampling the crumbs he dropped before, that is to say, the sender would overwrite data that the receiver hadn't processed yet). The sender does not need to worry about the receiver at all.

The most common reason for one routine to want to pass a stream of data to another in this fashion is communication. The 56001 hardware takes care of much of the mechanics of communication with the Lasagna Box and the GPP: when it has received a word from one of these devices, or it is ready to transmit a new word to them, it generates an interrupt to either read the word received or supply a new word to be transmitted. Interrupt handlers, however, ought to be as fast as possible, doing some minimal amount of work and then allowing the main code (*i.e.* code called by the mainloop, generally in a higher-level language than the interrupt handler, and thus more readable and possibly more robust) to deal with the data. And so an *interrupt-plus-main-code* system is used in these situations, where the handler for a receive interrupt simply places the new word into a circular buffer, and the handler for a transmit interrupt grabs the next word from a circular buffer and sends it. The main code is then responsible for filling the circular buffer used by a transmit handler, and for processing the data placed in a circular buffer by a receive handler. This takes the speed and efficiency advantages of a circular buffer and augments them with the speed of a short interrupt handler and the readability and robustness of high-level code.

The DSP code for HETE is written in Motorola GNU C, with time-critical routines (mostly relating to processing video data) coded in Motorola 56001 assembly.

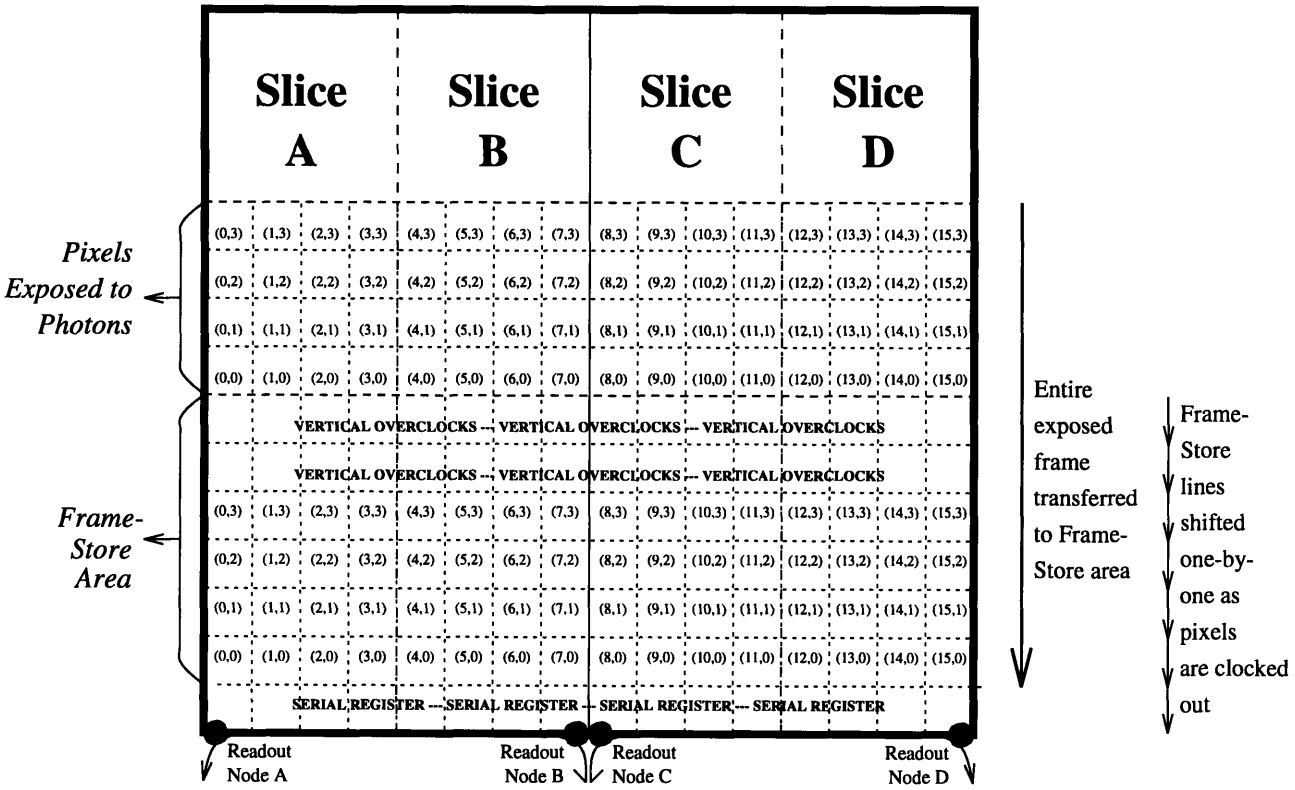
4.4 Camera Electronics: The Lasagna Box

The Lasagna Box, as indicated before, communicates via a command channel, a housekeeping channel, and a video channel. Words over the command channel correspond to either a write to or a read from a register within the Lasagna Box. Read commands generate a single response from the Lasanga Box over the housekeeping channel. There is no arbitration or regulation on housekeeping responses, so no more than one read command ought to be pending at any one time. A single active-high master reset signal initializes the device.

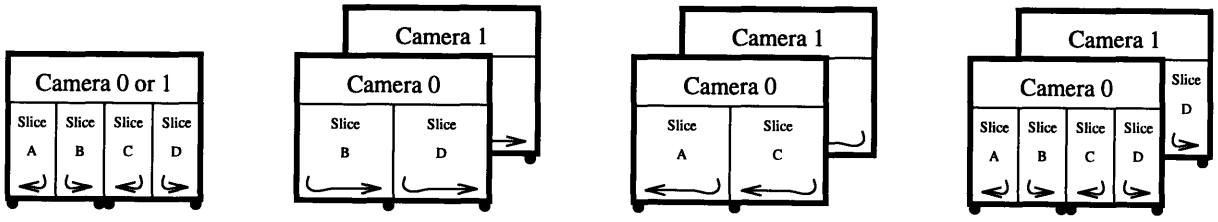
A charge-coupled device (CCD) camera uses semiconductor technology to transduce light energy into electrical signals. A CCD is a silicon wafer, one to two centimeters on a side, which produces free electrons when hit by photons within a given range of wavelengths. Placing focusing optics in front of such a device, such that light is sensibly aimed at the wafer instead of hitting it from all directions, creates a potentially useful camera. All that remains is to get the free electrons off the wafer in a manner that indicates how many photons hit a given part of the CCD. Figure 4-1 diagrams how this is accomplished in the HETE UV system.

A CCD is often pictured as an array of buckets which catch photons; the Lasagna Box measures the contents of these buckets and reports them as pixel values. Since this measuring process can take a significant amount of time, the CCD devices used on HETE are divided into an *exposed area* and a *frame-store area*: the former is exposed to photons arriving through the camera optics, which are then converted into an electrical charge; the latter is not exposed to photons, and therefore holds its charge until read out. When the exposed area has received sufficient exposure, its entire contents are shifted down into the frame-store area. The exposed area is now empty of charge and ready to receive the next image, and as it is doing so, the Lasagna Box is reading the pixel data from the previous image out of the frame-store

Single-Camera CCD Chip Layout



Readout Options



Mode 0 Readout **Mode 1 Readout** **Mode 2 Readout** **Mode 3 Readout**

Figure 4-1: Structure and Readout of the UV CCDs

area.

This reading is performed one bucket at a time, in a pixel-by-pixel line-by-line fashion. Say the CCD has been exposed to a new image, and the data transferred into the frame-store area: the Lasagna Box commands the first line in the frame-store area to dump its contents into the *serial register*, where the Lasagna Box can examine the pixel values in this line at one of four *readout nodes*. With the first line of data in the serial register, the second line can be transferred into the first, the third into the second, and so on. This poises the second line for transfer into the serial register for readout.

The four readout nodes are lettered A, B, C, and D. Since there are two cameras, each with four readout nodes, there are eight nodes total in the UV system: A0, B0, C0, D0, A1, B1, C1, and D1. Each readout node is capable of reading or *clocking out* pixels from a certain region of the CCD, depending on the *transmission mode* (often abbreviated "Mode") of the Lasagna Box. Clocking out pixels entails measuring how "full" the bucket at the readout node is, giving this fullness a number called the *pixel value* or *pixel intensity*, and then shifting the buckets in the serial register toward the readout node to read the next pixel value in the line. The region of the CCD whose pixels are clocked out by a given readout node is called a *slice*, and so there are potentially eight slices across the two cameras, A0 through D1, again depending on the Mode. There are four possible Modes:

- **Mode 0**: 16-bit pixel values clocked out from all the readout nodes in a single camera. With four readout nodes per camera, each slice (A, B, C, and D) is one-fourth the total width of the CCD. A Submode determines which camera (0 or 1).

- **Mode 1**: 16-bit pixel values clocked out from readout nodes B and D of both cameras. With two readout nodes utilized per camera, Slice B is the left half of a camera, and Slice D the right half.

- **Mode 2**: 16-bit pixel values clocked out from readout nodes A and C of both cameras. With two readout nodes utilized per camera, Slice A is the left half

of a camera, and Slice C the right half.

- **Mode 3**: 12-bit pixel values clocked out from all the slices in both cameras. Each of the eight slices is one-fourth the total width of a CCD. Two pixels are packed into a word.

For the 16-bit transmission modes (0 through 2), the lower 8 bits of each 24-bit word arriving over the video channel are all 1's, so pixel data is 0xPPPPFF, with the pixel value in PPPP. For the 12-bit Mode 3, two pixel values $0xP_1P_1P_1$ and $0xP_2P_2P_2$ are stacked as $0xP_1P_1P_1P_2P_2P_2$ in each 24-bit video channel word.

Note from Figure 4-1 that while readout nodes A and C clock pixels out left-to-right, readout nodes B and D clock pixels out right-to-left. Thus while readout node A would read pixel (0,0), shift its portion of the serial register to the left, read pixel (1,0), shift again, and read (2,0), readout node B would read pixel (7,0), shift its portion of the serial register to the right, read pixel (6,0), shift again, and read (5,0). This "backward" readout for Slices B and D complexifies reconstruction of the camera image.

When a full line of pixel values have been clocked out, the next line is transferred into the serial register, and all of the lines above it are transferred down one line. It is important to note that the specifics involved in transferring lines and clocking out pixels produces three types of *non-image pixels.* When a readout node begins to clock out a new line, the actual pixel values are preceded by a set of *horizontal underclocks*, and when all of the pixel values have been clocked out, the readout nodes continue to supply *horizontal overclocks* before the next line is shifted into the serial register. When all of the lines containing image data have been clocked out, a few *vertical overclock* lines remain, and they are clocked out as well. These non-image pixels represent a baseline "dark" response of the CCD, and are included in the video data in the order they are produced.

The Lasagna Box simultaneously reads all of the active readout nodes in its present Mode. This means that the Lasagna Box produces four pixel values at a time in Modes 0 through 2, and eight pixel values at a time in Mode 3. It uses these values to

Mode, [Submode]	First word from slice(s)	Second word from slice(s)	Third word from slice(s)	Fourth word from slice(s)
0, 0	A0	B0	C0	D0
0, 1	A1	B1	C1	D1
1	B0	D0	B1	D1
2	A0	C0	A1	C1
3	A0:B0	C0:D0	A1:B1	C1:D1

Table 4.2: Construction of Package-of-Four Video Words

(P1:P2 indicates pixel P1 is in the top 12 bits, P2 in the bottom 12 bits)

construct a *package-of-four-video-words*, or package-of-four, where each word contains the pixel value(s) clocked out from one (two in Mode 3) of the readout nodes, as indicated in Table 4.4.

These packages-of-four comprise the majority of the video stream. The video stream, transmitted to the FEP and the BEP over the video channel, contains three types of words: words containing pixel values; sync words; and blank words. A row sync pattern precedes every row of pixel data, and a frame sync pattern (which includes a frame identification number) precedes the set of rows representing a new frame. A small fixed number of blank words precedes each row sync pattern, and a large variable number of blank words precedes each frame sync pattern.

The sync patterns are also organized in packages-of-four. At time of development, the frame sync was two consecutive packages-of-four containing all 0x00FFFF's, followed by 24 packages-of-four encoding a 24-bit frame ID (four 0x00FFFF's was a one bit, four 0x000000's was a zero bit; most significant bit was sent first). The row sync was four 0x000000's followed by four 0x00FFFF's.

Let us posit a camera whose four slices each have two horizontal underclock columns, three image pixel columns, one horizontal overclock column, two image rows, and one vertical overclock row. Note that the actual camera image is 12x2. With the Lasagna Box in Mode 0, the video channel would look like this:

(Frame sync)
(Blank words)

(Row sync)

1. First horizontal underclock of slice A, first row
2. First horizontal underclock of slice B, first row
3. First horizontal underclock of slice C, first row
4. First horizontal underclock of slice D, first row
5. Second horizontal underclock of slice A, first row
6. Second horizontal underclock of slice B, first row
7. Second horizontal underclock of slice C, first row
8. Second horizontal underclock of slice D, first row
9. First image pixel of slice A, first row: image coordinate (0,0)
10. First image pixel of slice B, first row: image coordinate (5,0)
11. First image pixel of slice C, first row: image coordinate (6,0)
12. First image pixel of slice D, first row: image coordinate (11,0)
13. Second image pixel of slice A, first row: image coordinate (1,0)
14. Second image pixel of slice B, first row: image coordinate (4,0)
15. Second image pixel of slice C, first row: image coordinate (7,0)
16. Second image pixel of slice D, first row: image coordinate (10,0)
17. Third image pixel of slice A, first row: image coordinate (2,0)
18. Third image pixel of slice B, first row: image coordinate (3,0)
19. Third image pixel of slice C, first row: image coordinate (8,0)
20. Third image pixel of slice D, first row: image coordinate (9,0)
21. Horizontal underclock of slice A, first row
22. Horizontal underclock of slice B, first row
23. Horizontal underclock of slice C, first row
24. Horizontal underclock of slice D, first row

    (Blank words)

    (Row sync)
25. First horizontal underclock of slice A, second row
26. First horizontal underclock of slice B, second row
27. First horizontal underclock of slice C, second row

28. First horizontal underclock of slice D, second row
29. Second horizontal underclock of slice A, second row
30. Second horizontal underclock of slice B, second row
31. Second horizontal underclock of slice C, second row
32. Second horizontal underclock of slice D, second row
33. First image pixel of slice A, second row: image coordinate (0,1)
34. First image pixel of slice B, second row: image coordinate (5,1)
35. First image pixel of slice C, second row: image coordinate (6,1)
36. First image pixel of slice D, second row: image coordinate (11,1)
37. Second image pixel of slice A, second row: image coordinate (1,1)
38. Second image pixel of slice B, second row: image coordinate (4,1)
39. Second image pixel of slice C, second row: image coordinate (7,1)
40. Second image pixel of slice D, second row: image coordinate (10,1)
41. Third image pixel of slice A, second row: image coordinate (2,1)
42. Third image pixel of slice B, second row: image coordinate (3,1)
43. Third image pixel of slice C, second row: image coordinate (8,1)
44. Third image pixel of slice D, second row: image coordinate (9,1)
45. Horizontal underclock of slice A, second row
46. Horizontal underclock of slice B, second row
47. Horizontal underclock of slice C, second row
48. Horizontal underclock of slice D, second row
    (Blank words)
    (Row sync)
49. First pixel of slice A, vertical overclock row
50. First pixel of slice B, vertical overclock row
51. First pixel of slice C, vertical overclock row
52. First pixel of slice D, vertical overclock row
53. Second pixel of slice A, vertical overclock row
54. Second pixel of slice B, vertical overclock row
55. Second pixel of slice C, vertical overclock row

56. Second pixel of slice D, vertical overclock row
57. Third pixel of slice A, vertical overclock row
58. Third pixel of slice B, vertical overclock row
59. Third pixel of slice C, vertical overclock row
60. Third pixel of slice D, vertical overclock row
61. Fourth pixel of slice A, vertical overclock row
62. Fourth pixel of slice B, vertical overclock row
63. Fourth pixel of slice C, vertical overclock row
64. Fourth pixel of slice D, vertical overclock row
65. Fifth pixel of slice A, vertical overclock row
66. Fifth pixel of slice B, vertical overclock row
67. Fifth pixel of slice C, vertical overclock row
68. Fifth pixel of slice D, vertical overclock row
69. Sixth pixel of slice A, vertical overclock row
70. Sixth pixel of slice B, vertical overclock row
71. Sixth pixel of slice C, vertical overclock row
72. Sixth pixel of slice D, vertical overclock row

Chapter 5

Front-End Processor Functionality

When it is dark enough you can see the stars.

— *Ralph W. Emerson*

5.1 Camera Electronics Command and Housekeeping

The GPP relies on the FEP to relay instructions to the Lasagna Box, and to report back any replies from the Lasagna Box. The former is the Lasagna Box *command channel*; the latter, the *housekeeping channel*. The FEP reconciles the differing protocols and requirements of the GPP and the Lasagna Box. It is also sensitive to potential errors, reporting to the GPP should they occur.

The requirements of the GPP in this regard are:

- Communication must be via IPP
- Commands for the Lasagna Box are sent to the FEP as a single IPP message, a list of up to 128 read and write commands
- Each response from the Lasagna Box must be associated with the read command that generated it

- When a complete set of responses (and their associated commands) is gathered, it must be sent *en masse* to the GPP
- Responses must be sent back to the process which sent the associated commands

On the other hand, the Lasagna Box has the following restrictions:

- Communication must be in the appropriate serial format (see section 4.4)
- A word must be sent over the command channel every 48 microseconds
- No more than one read command may be pending at any time

5.1.1 Lasagna Box Requirements

Lasagna Box commands are 24-bit words, which either write to or read from a register within the device. Read commands generate a 16-bit response over the housekeeping channel, composed of 2 8-bit words, with the more significant word arriving first. The command channel is a synchronous serial interface, and the FEP is responsible for providing the 500 kHz serial clock and the one-bit-per-word synchronizing signal as well as the command data. The Lasagna Box provides the serial clock for the housekeeping channel, with synchronization included in the data stream. See section 4.4 for details.

For the Lasagna Box to operate properly, it must be constantly in the state of receiving a command: that is to say, transmission of a single command word over the command channel takes 48 μs (24 bits at $\frac{1}{500kHz} = 2\mu s$ per bit), and the Lasagna Box must receive a command word every 48 μs. The vast majority (99+%) of these words will be the "no-op" or no-operation command, which does not cause a read or a write within the Lasagna Box. The FEP supplies this regular stream of no-ops over the command channel, inserting more consequential commands as the GPP instructs it. This is accomplished by the standard DSP communication technique of low-level interrupt routine plus high-level "main" code, connected by a buffer: one routine in the FEP's main loop fills a circular buffer (the *transmit queue*) with no-ops; another

inserts "real" commands into the same buffer as they are called for; and an interrupt-driven routine simply sends one word after another from this buffer out the command channel, once every 48 μs.

When the Lasagna Box receives a read command, it sets about performing the read and relaying the result over the housekeeping channel. Different registers take different times to respond to a read command: a correlation error is therefore possible. For instance — the Lasagna Box receives read command for register A, which takes 200 μs to respond, immediately followed by a read command for register B, which takes 50 μs to respond. The FEP will thus receive the contents of register B first over the housekeeping channel, followed by the contents of register A, the opposite order of how the FEP requested this information. Further, there is no bus arbitration on the housekeeping channel within the Lasagna Box: in the above example, if the data from register A arrived while the data from B was still in the process of being transmitted, the result would be a useless jumble of bits.

The FEP therefore assumes the responsibility of ensuring that no more than one read command is pending in the Lasagna Box at any time. When the FEP places a read command in the transmit queue, it holds off from placing any more read/write commands there (sending no-ops in the meantime) until it has the reply to that read safely in hand. Once the reply is received, the FEP may resume sending commands. This precludes any bus collisions within the Lasagna Box, as well as guaranteeing that the FEP can reliably associate each read command with its response.

5.1.2 GPP Requirements

All communication between the FEP and the GPP is via the Inter-Process Protocol, or IPP, described in section 3.3. The FEP thus receives in an IPP message a list of commands to be sent to the Lasagna Box, and must compose an IPP message in response with a list of the replies from the Lasagna Box.

Up to 128 Lasagna Box commands may be sent to the FEP in a single IPP message (512 16-bit words per message, and packing of each 24-bit Lasagna Box command takes 2 words). The FEP is able to store 3 such messages, providing a buffer that

shields the GPP from having to worry about the specifics of feeding the commands to the Lasagna Box. This buffer allows any process within the GPP to send a Lasagna Box commands message as it deems necessary without needing to check whether the FEP has finished sending a previous set of commands.

The FEP, upon receiving such a message, begins to place the commands on the transmit queue. It checks each one to see whether or not it is a read command; if it is, it does not send any more of the commands from this message until the reply is received over the housekeeping channel, as per the system described in the previous subsection. Note that the FEP continues performing its other functions during this waiting period, in keeping with the mainloop concept described in section 4.3.

When a 16-bit housekeeping word is received, it is paired with the read command that prompted it, and the two pieces of information are saved. Placement of commands from the Lasagna Box commands message into the transmit queue recommences, continuing until the next read command is encountered, in a repeating cycle until all of the commands from this message have been placed in the queue. When this Lasagna Box commands message is exhausted, the housekeeping data and their associated read commands are placed in an IPP message and sent to the originator of the commands message. If another Lasagna Box commands message has arrived in the meantime, the transmission process begins anew.

In this fashion a GPP process which wishes to communicate with the Lasanga Box merely constructs a list of commands (reads and writes), and sends them in an IPP message to the FEP. Some time later this process receives a message in reply, which contains all the read commands the process sent, along with the results of these reads.

5.2 Event Finding

When the Lasagna Box has sent a complete frame of data over the video channel, the FEP reports to the GPP the locations of stars within that frame. A star is defined as a pixel whose value exceeds a given *background threshold* and which is maximal

within a 3-by-3 area centered on the pixel. These pixels are termed "events", and event finding is the FEP's second major responsibility.

This process is made difficult primarily by the peculiarities of the CCD camera system and its readout over the video channel, described in section 4.4. Recall that each of the two cameras governed by the Lasagna Box is composed of four CCD slices, each with their own photon response characteristics, and each having image-pixel and non-image-pixel regions. This poses the following additional constraints on event finding:

- Only image pixels are valid event candidates.
- Because slices have differing response characteristics, comparison of pixel values from different slices is invalid. Thus an event cannot be on the boundary between two slices, *i.e.*, the first or last image pixel in a slice row.
- For the same reason, background thresholds must be slice-specific.
- Because CCD devices can have "bad columns" and "bad pixels", these locations, as well as those which use them in the 3x3 local-maximum comparison, are not valid event candidates.
- The high rate of the video channel (at maximum, 2 pixels arriving every 6 μs) demands an exceptionally speed-optimized event finding algorithm.

Several of these constraints are actually turned to advantage in the FEP: specifically, the fact that any given 3x3 comparison occurs strictly within a single slice allows the FEP largely to ignore the jumbled slice order which arrives over the video channel; and the limitation to image pixels (which are easily located, even within the video channel pixel stream) significantly reduces the amount of computation required to find events.

5.2.1 Arrival of Video Data

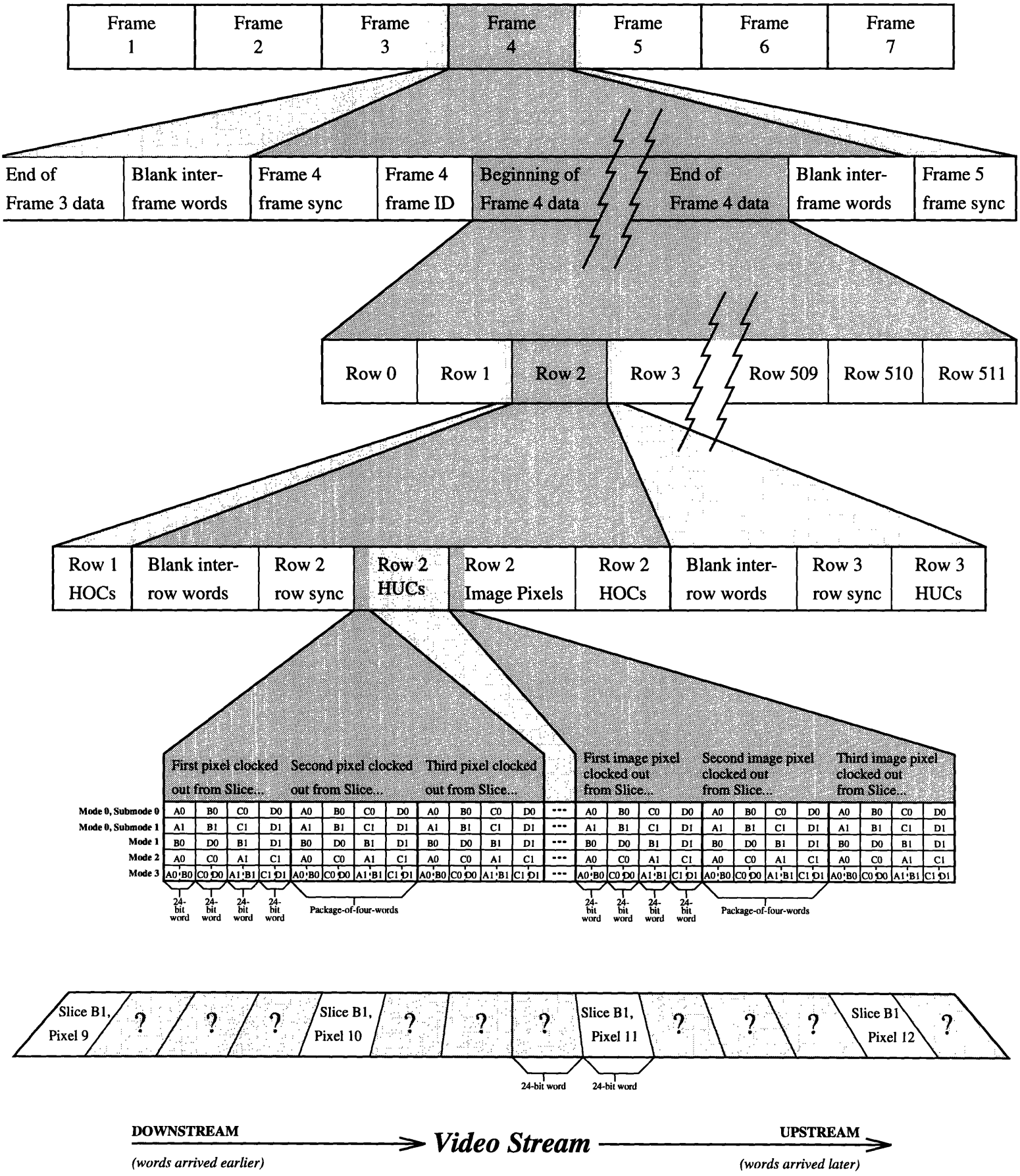


Figure 5-1: Levels of Detail Within the Video Data Stream

Let us break down the data which arrives over the video channel, as shown in Figure 5-1. On the grandest scale, the data represent an ongoing series of frames, camera images taken at successive times. Narrowing our scope, we regard a single frame, which arrives between two other frames: this single frame's beginning is signalled by the arrival of a frame sync over the video channel, and its end by the arrival of the next frame's frame sync. This frame itself is composed of a set of lines or rows, all identical in structure — that is, if the twenty-seventh word in row 1 represents coordinate (100, 1) on camera 0, then the twenty-seventh word in row 2 represents coordinate (100, 2), and the twenty-seventh word in row r represents the coordinate (100, r). Taking a closer look at this row structure, we note some blank words followed by a row sync at the beginning, followed in turn by a set of pixel values which represent information about the CCD "buckets" along that row. Narrowing our scope still further to attempt to determine the meaning of these pixel values, we note that they come, conceptually, in packages of four 24-bit words. Each of these four words represents data from either one or two CCD slices; and adjacent four-word packages contain horizontally adjacent data from each of these slices.

From another perspective: you are standing at a certain point in the video stream, and you know the number under your feet represents the value of the eleventh pixel in slice B, camera 1. If you want to find the twelfth pixel in slice B, camera 1, you go four words upstream (toward the later-arriving words); if you want to find the ninth pixel in slice B, camera 1, you go eight words downstream (toward the earlier-arrived words). This is true for all of the readout modes and submodes the Lasagna Box is capable of.

Further: for each row every slice reads out a set of horizontal underclocks, followed by a set of image pixels, followed by a set of horizontal overclocks. Given the previous observation, we conclude that for every row, the video stream contains a set of packages-of-four-words which are horizontal underclocks (from each of the four or eight slices represented in the given readout mode), followed by a set of packages-of-four-words which are image pixels, followed by a set of packages-of-four-words which are horizontal overclocks.

5.2.2 The FEP Event Finding Algorithm

A review is in order, with a focus on the FEP's use of the above observations. The FEP needs to compare each pixel to one of eight thresholds (one threshold for each slice in the two-camera system); but if it approaches the pixel stream as arriving in packages of four words, it can select the thresholds applicable to the origin of each of these four words, and then cycle through these preselected thresholds repeatedly, without truly being cognizant of the specific meaning of each word it compares.

For instance, let us say the Lasagna Box is operating in Mode 1. The packages-of-four will therefore represent Slices (that is to say, readout nodes) B0, D0, B1, D1. The FEP, upon syncing a new frame, loads variable **thresh0** with the threshold for B0, **thresh1** with the threshold for D0, **thresh2** with the threshold for B1, and **thresh3** with the threshold for D1. The actual threshold comparison loop then simply consists of:

1. Compare this word in the pixel stream to **thresh0**; go to next word in the pixel stream.
2. Compare this word in the pixel stream to **thresh1**; go to next word in the pixel stream.
3. Compare this word in the pixel stream to **thresh2**; go to next word in the pixel stream.
4. Compare this word in the pixel stream to **thresh3**; go to next word in the pixel stream.
5. Go to step 1.

If the Lasagna Box switches to Mode 0, camera 0, **thresh0** through **thresh3** are set to the thresholds for A0 through D0, respectively. The threshold comparison loop is unaffected. This "mode ignorance" allows for a great deal of optimization, in both speed and program memory. Table 5.2.2 summarizes the possible assignments of these tt thresh$n$ variables.

Mode, Submode	thresh0 set to the thresh. for	thresh1 set to the thresh. for	thresh2 set to the thresh. for	thresh3 set to the thresh. for	thresh4 set to the thresh. for	thresh5 set to the thresh. for	thresh6 set to the thresh. for	thresh7 set to the thresh. for
0, 0	A0	B0	C0	D0	unused	unused	unused	unused
0, 1	A1	B1	C1	D1	unused	unused	unused	unused
1, 0/1	B0	D0	B1	D1	unused	unused	unused	unused
2, 0/1	A0	C0	A1	C1	unused	unused	unused	unused
3, 0/1	A0	B0	C0	D0	A1	B1	C1	D1

Table 5.1: Assignment of Event-Finding **thresh***n* Variables

Mode ignorance is carried through to the local-maximum check. Since comparisons are only performed intra-slice, and intra-slice neighbors are always a fixed distance apart in the pixel stream, the FEP neither knows nor cares what slice it is in when it performs a local-maximum comparison. The FEP has found a pixel value that exceeds threshold: it can compare this event candidate to the value four words upstream, and the value four words downstream — the candidate's horizontal neighbors. It can then go one row upstream and repeat the process, comparing the candidate to the three neighbors "above" it; similarly one row downstream, for comparison with the three neighbors "below". It is only when an event is found that its true on-camera coordinate is calculated and recorded.

Note that there is the potential for error when a "plateau" is encountered, that is to say, a connected region of same-valued above-threshold pixels. If the local-maximum check searches for a pixel strictly greater in value than its neighbors, a plateau will generate no event; if on the other hand the search is for a greater-or-equal value, several of the pixels on the plateau may be recorded as distinct events. To alleviate this, the eight-neighbor comparison is divided into four greater-or-equal comparisons on one side, and four strictly-greater on the other. Specifically, a candidate must be $\geq$ the pixels below and to the left[1] of it, and $>$ the pixels above and to the right; or from another perspective, $\geq$ the pixels which precede the candidate in the pixel stream, and $>$ the pixels which follow it. This means that a summit plateau (where the pixel values decrease past the edges of the same-valued region) will cause a single

[1] "Left" and "right" are meant here in a slice-relative sense: the fourth pixel in slice B is considered to be left of the fifth pixel in slice B, even though the reverse readout of slices B and D means that the fourth pixel is on the right of the fifth pixel in an absolute, camera-relative sense.

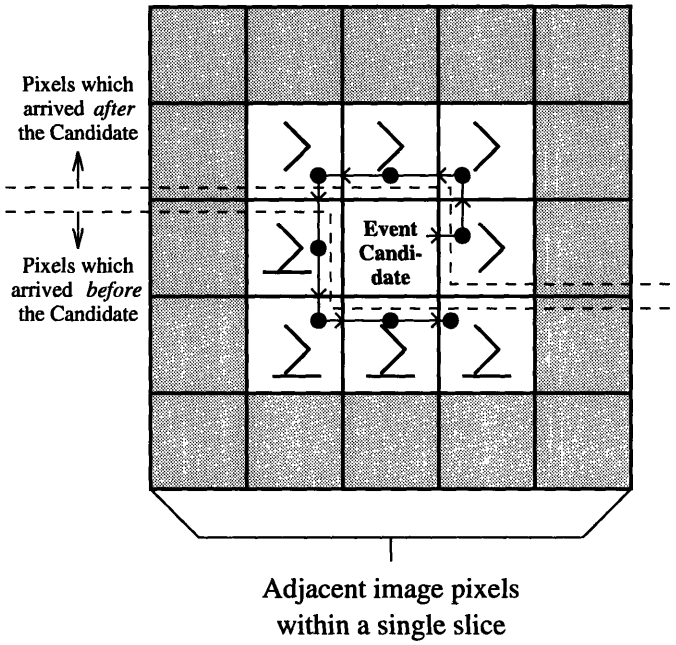


Figure 5-2: Local–Maximum Check

event to be recorded, in the upper-right corner of the plateau. This results in a local-maximum check of the kind shown in Figure 5-2; the arrowed lines indicate the order in which the comparisons are made, as described below in **Check for Local Maximum.**

These are the key elements in the event finding algorithm. Bringing it all together, we have the following system:

- **Process Line** (16 bit)

  Needs three full lines of data — processes the middle one

  1. If word meets or exceeds `thresh0`, **Check for Local Maximum**
  2. Go to next word
  3. If word meets or exceeds `thresh1`, **Check for Local Maximum**
  4. Go to next word
  5. If word meets or exceeds `thresh2`, **Check for Local Maximum**
  6. Go to next word

7. If word meets or exceeds **thresh3**, **Check for Local Maximum**
8. Go to next word
9. Repeat until end of line

- **Process Line** (12 bit)

  Needs three full lines of data — processes the middle one

  1. If upper pixel in word $\geq$ **thresh0**, **Check for Local Maximum**
  2. If lower pixel in word $\geq$ **thresh1**, **Check for Local Maximum**
  3. Go to next word
  4. If upper pixel in word $\geq$ **thresh2**, **Check for Local Maximum**
  5. If lower pixel in word $\geq$ **thresh3**, **Check for Local Maximum**
  6. Go to next word
  7. If upper pixel in word $\geq$ **thresh4**, **Check for Local Maximum**
  8. If lower pixel in word $\geq$ **thresh5**, **Check for Local Maximum**
  9. Go to next word
  10. If upper pixel in word $\geq$ **thresh6**, **Check for Local Maximum**
  11. If lower pixel in word $\geq$ **thresh7**, **Check for Local Maximum**
  12. Go to next word
  13. Repeat until end of line

- **Check for Local Maximum**

  1. Go four words upstream (candidate's right neighbor)
  2. Mask relevant bits from this word
  3. If the candidate is less than or equal to this word, return
  4. Go one row upstream (candidate's upper-right neighbor)
  5. Mask relevant bits from this word
  6. If the candidate is less than or equal to this word, return
  7. Go four words downstream (candidate's upper neighbor)
  8. Mask relevant bits from this word
  9. If the candidate is less than or equal to this word, return
  10. Go four words downstream (candidate's upper-left neighbor)
  11. Mask relevant bits from this word
  12. If the candidate is less than or equal to this word, return
  13. Go one row upstream (candidate's left neighbor)
  14. Mask relevant bits from this word

15. If the candidate is less than this word, return
16. Go one row downstream (candidate's lower-left neighbor)
17. Mask relevant bits from this word
18. If the candidate is less than this word, return
19. Go four words upstream (candidate's lower neighbor)
20. Mask relevant bits from this word
21. If the candidate is less than this word, return
22. Go four words upstream (candidate's lower-right neighbor)
23. Mask relevant bits from this word
24. If the candidate is less than this word, return
25. **Record Candidate as Event**

- **Record Candidate as Event**

  1. Determine the candidate's on-camera coordinate given its location in the pixel stream
  2. If this coordinate is on or adjacent to (*i.e.* was compared to) a bad pixel or bad column, return
  3. Record candidate's on-camera coordinate and intensity (X, Y, Z) as an event

5.2.3 Video Processing

All that remains is to set up circumstances such that **Process Line** can operate sensibly. The FEP must keep track of the parameters that govern the Lasagna Box and the video stream, specifically:

- The video transmission mode (M) and submode (SM)
- The number of horizontal underclocks in a slice ($HUCS$)
- The number of image pixels per line in a slice ($IMPS$)
- The number of horizontal overclocks in a slice ($HOCS$)
- The number of active rows in a frame ($ROWS$)
- The number of vertical overclocks in a frame ($VOCS$)

With the horizontal parameters (the number of underclocks, image pixels, and overclocks), plus awareness of the number of inter-slice words that will arrive along the video channel (the row sync pattern, plus any inter-row "blank" words introduced by the Lasagna Box for hardware reasons), the distance from row to row in the video stream may be calculated. That is, the number of words in the video stream between the pixel representing (*C*, *R*) and (*C*, *R+1*) is given by

$$\Delta Row = 4*(HUCS+IMPS+HOCS)+ROW\_SYNC\_LEN+INTER\_ROW\_WORDS$$

This is one variable the event finding algorithm requires. Also, **thresh0** through **thresh3** (or, for Mode 3, **thresh7**) must be set. The FEP keeps a list of the thresholds for each slice (A0 through D1), updated as required by the GPP, and based on *M* and *SM*, loads them according to Table 5.2.2.

And finally, the algorithm must have a means of observing the video stream itself. The FEP utilizes a circular *video buffer*, at least four times the largest possible ΔRow, into which data from the video channel is continuously poured. With a buffer of this size, three full rows of data are available for **Process Line**, and incoming data may continue to be dropped into the fourth row's worth of space. All that remains is to keep track of each row as it arrives.

This is accomplished via the sync patterns inserted into the pixel stream by the Lasagna Box. Assume the FEP is waiting for a frame sync to arrive, so it can start processing a frame: as described in section 4.4, it will observe a long string of blank words, followed by a set frame sync pattern (within which a frame identification number is encoded). Once the FEP has found this pattern and noted the frame ID, it checks if this frame should be processed. If so, it reports to the GPP that it has synced this frame, and begins to search for row sync patterns. If not, it returns to wait-for-frame-sync mode, ignoring the frame data.

The FEP first checks if a row's worth of data (ΔRow words) has arrived since the last row sync (or frame sync if this is the first row). If so, it checks for a row sync pattern at the beginning of this data. If there is a problem with the row sync, the FEP reports this to the GPP, and discards the remainder of this frame by going into wait-for-frame-sync mode.

If the row sync is valid, however, the FEP increments its running counter of the number of rows received so far in this frame. If this number indicates there are at least three image rows present in the video buffer, the middle row — that is, the row preceding the one just received and synced — may be processed. Recalling from Figure 5-1 that for each row, the video stream has a set of packages-of-four-words representing horizontal underclocks, a set of packages-of-four-words representing image pixels, and a set of packages-of-four-words representing horizontal overclocks, we note that we are only interested in the middle section, and may easily avoid processing the other two. In fact, since pixels on slice boundaries are not to be considered event candidates, the packages-of-four-words that begin and end the image-pixel region in the video stream may be skipped as well. So if we start with a pointer PTR into the video buffer that points to the first word following the row sync just found, we want to go downstream one row to get to the line we want to analyze, then go back upstream past the underclock and slice-boundary words:

$$PROCESS\_START\_PTR = PTR - \Delta Row + 4 * (HUCS + 1)$$

... and we want to process all the image pixels except the ones on the slice boundaries:

$$WORDS\_TO\_PROCESS = (4 * IMPS) - 2$$

In summary, a frame with 5 active rows and 2 vertical overclocks will be processed as follows:

1. FEP has current Lasagna Box parameters
2. FEP calculates ΔRow, WORDS_TO_PROCESS
3. FEP is in wait-for-frame-sync mode
4. Frame sync arrives and is placed in video buffer
5. FEP finds frame sync
6. FEP records frame ID and reports to GPP
7. FEP initializes **thresh0** through **thresh7**
8. First row of data arrives and is placed in video buffer

9. FEP finds row sync preceding first row: 1 line received
10. Second row of data arrives and is placed in video buffer
11. FEP finds row sync preceding second row: 2 lines received
12. Third row of data arrives and is placed in video buffer
13. FEP finds row sync preceding third row: 3 lines received
14. PROCESS_START_PTR calculated for processing of second row
15. **Process Line** called: second row is checked for events
16. Fourth row of data arrives and is placed in video buffer
17. FEP finds row sync preceding fourth row: 4 lines received
18. PROCESS_START_PTR calculated for processing of third row
19. **Process Line** called: third row is checked for events
20. Fifth row of data arrives and is placed in video buffer
21. FEP finds row sync preceding fifth row: 5 lines received
22. PROCESS_START_PTR calculated for processing of fourth row
23. **Process Line** called: fourth row is checked for events
24. Event finding complete: event list sent to GPP (with or without "Z" intensity values, as instructed by the GPP)
25. First vertical-overclock row arrives and is placed in video buffer
26. FEP finds row sync preceding first vertical-overclock row: 6 lines received
27. Second vertical-overclock row arrives and is placed in video buffer
28. FEP finds row sync preceding second vertical-overclock row: 7 lines received
29. FEP reports end-of-frame to GPP
30. FEP is in wait-for-frame-sync mode

5.3 Command & Response List

The FEP responds to the following commands from the GPP:

PROCESS_VIDEO

The FEP is instructed to process the next frame to arrive along the video channel. A parameter indicates whether the FEP should process only this next frame (and report intensity, or "Z", values), or to begin processing frame after frame until told to stop (and not report Z values to save communication time).

DONT_PROCESS_VIDEO

The FEP is to stop processing frames until told to start again. If a frame is currently being processed, the FEP is to finish doing so, but then ignore frames that follow it. There are no parameters.

BAD_COLUMN_LIST

The FEP is informed of the columns which are deemed faulty by the GPP. The FEP is not to report any events which are on or immediately adjacent to these columns. The FEP's bad-column list may be replaced or appended by the data in this message.

BAD_PIXEL_LIST

The FEP is informed of the pixels which are deemed faulty by the GPP. The FEP is not to report any events which are on or within a 3x3 area around these pixels. The FEP's bad-pixel list may be replaced or appended by the data in this message.

CCD_CONFIGURATION

The FEP is informed of: the transmission mode and cameras used (submode) by the Lasagna Box; the number of horizontal underclocks, active image columns, and horizontal overclocks per slice; the number of active image rows and vertical

overclocks per frame; and the on-camera row number of the first row to arrive over the video stream. The FEP may be instructed to abort processing the current frame in favor of these new parameters, or to institute these parameters when this frame has ended.

EVENT_THRESHOLDS

The FEP is told the background threshold values for slices A0, B0, C0, D0, A1, B1, C1, and D1.

LB_COMMANDS

The FEP is given a list of commands to send to the Lasagna Box.

The FEP generates the following messages for the GPP:

DSP_FRAME_STARTED

The FEP reports that a frame sync has been found. The frame ID is included. Message goes to the most recent sender of PROCESS_VIDEO.

DSP_FRAME_ENDED

The FEP reports that all of the rows of this frame, including vertical overclocks, have been received. The frame ID is included. Message goes to the most recent sender of PROCESS_VIDEO.

DSP_FRAME_ERROR

The FEP reports that there has been some error (such as an absent row sync) while processing a frame. The frame ID and a number indicating the type of error are included. Message goes to the most recent sender of PROCESS_VIDEO.

FEP_XY_EVENT_LIST

One of a set of messages reporting the (X, Y) coordinates of all events found in a frame. The frame ID is included, as well as a set of flags indicating whether this message is the first, last, or in the middle of the set. Message goes to the most recent sender of PROCESS_VIDEO.

FEP_XYZ_EVENT_LIST

One of a set of messages reporting the (X, Y, Z) coordinates and intensities of all events found in a frame. The frame ID is included, as well as a set of flags indicating whether this message is the first, last, or in the middle of the set. Message goes to the most recent sender of PROCESS_VIDEO.

LB_RESPONSES

The FEP reports the responses from the Lasagna Box generated by the commands in a previous LB_COMMANDS message. Each response is associated with the command that generated it. Message goes to the sender of the LB_COMMANDS message.

DSP_EXCEPTION

The FEP warns of a hardware exception, such as a video channel receive overrun, or command channel transmit underrun. For errors relating to Lasagna Box command and housekeeping, messages go to the most recent sender of LB_COMMANDS; for errors relating to the video channel, messages go to the most recent sender of PROCESS_VIDEO.

Chapter 6

Front-End Processor Implementation

The last 5% of a project takes 95% of the time.

— Murphy's Laws on Technology

The FEP C code is compiled to use the 56001's Y: data space, not to use register R5, and to employ parallel move operations when possible. FEP Assembly code uses parallel move operations heavily. Although efforts have been made in this chapter to point out when the FEP uses some subtle aspect of the compiler or the processor, readers unfamiliar with the Motorola 56001 or its GNU C compiler are referred to the documents on these cited in the Bibilography.[5, 6]

This chapter also uses terms such as "video buffer" and "package-of-four" defined in the previous chapter. Readers unfamiliar with the algorithms that this code implements are referred there.

Variables related to set beginnings are inclusive, and those related to set endings, exclusive. That is, start_col = 5 *and* end_col = 10 *defines columns 5, 6, 7, 8, and 9 as included in a set. This means that the number of members in the set is always the difference between the end and the beginning values.*

Numeric indices — coordinates, array references, bit locations, et cetera *— start with 0. Cardinal references start with "first". Hence the "first bit" of a 24-bit word is "bit 0", and the most significant "twenty-fourth" bit is "bit 23". Confusing as this appears, I believe it is a reasonably natural pattern, and consistent with related literature.*

6.1 Functional Breakdown

The FEP software is composed of the following modules:

- **FEPmain.c**: Performs initializations (with **FEPsetvars.c**), engages mainloop (see section 4.3). Also contains simulation- and memory-related test routines.
- **FEPsetvars.c**: Contains initialization routines.
- **FEPcom.c**: Handles high-level (*i.e.* above Assembly-level) communications with the GPP and Lasagna Box.
- **FEPvid.c**: Sets up event finding. Has routines for frame and row sync'ing, event recording, GPP updating; passes to **FEPproc.asm** for actual event-finding algorithm.
- **FEPproc.asm**: Performs event-finding algorithm described in section 5.2. Highly time-optimized code.
- **FEPasmsupport.asm**: Contains various small support routines which are best implemented in Assembly language, such as serial communications hardware control, interrupt vector initialization, *et cetera.*
- **FEPintr.asm**: Long interrupt handlers, for command transmission and exception, housekeeping receive and exception, and video receive exception.
- **crtrzp.asm**: Startup file which controls the DSP's behavior on reset. Initializes the stack pointer, DMA communications with the host GPP, and the like; then passes off to **FEPmain.c**.

- **FEPincl.h**: Contains constants, type and structure definitions, and external function declarations used by the above C modules. #include's other infrastructure .h files, such as those related to IPP communication.

- **FEPglobals.h**: Global variable declarations. #include'd once by **FEPmain.c**.

- **FEPequ.asm**: Constant definitions and variable allocations for the above Assembly modules.

- **machspec.asm**: Constant definitions for Assembly modules, specific to HETE hardware (as opposed to the NeXT development system).

- **ioequ.asm**: Constant definitions for communications hardware.

FEPincl.h defines several structures which contain vital information on the state of the FEP, in particular its video processing functions. Of primary importance is vidbuf_param_struct[1], which contains the following variables:

- id: 24-bit Frame Identification number of the most recently synced frame. When a new frame is synced, this number is set to the new frame's ID as encoded in the frame sync pattern.

- ccd: Structure containing the relevant parameters governing the cameras and Lasagna Box (see below for description).

- bad: Pointer to a structure containing information on the CCD pixels and columns which the GPP has deemed "bad", *i.e.*, pixels at or adjacent to these locations are not to be reported as events. See below for description.

- thresh[0..7]: The background thresholds for all eight slices in the two-camera system. Assumed to be right-justified but otherwise applicable to the current Lasagna Box transmission mode: *i.e.*, a 12-bit number (0–4095) for Mode 3, a 16-bit number (0-65535) otherwise. The relation of index to slice is given by the SLICE_*xx* constants definitions earlier in **FEPincl.h**.

[1] "vidbuf" is an abbreviation for the VIDeo BUFfer. The values in this structure relate either directly to the video buffer, or to the processing of the data in it.

- **do_sifting:** Flag telling the FEP whether or not this frame ought to be processed (*i.e.* searched for events).

- **send_z:** Flag telling the FEP whether or not the GPP is interested in the intensity ("Z") values of the events found in this frame. Also indicates whether or not the FEP is in single- or multi-frame mode: if the GPP is interested in Z values, then the FEP should assume that this is the only frame to be processed; otherwise the FEP should keep sending down (X, Y) event lists for every frame it sees until told to stop.

- **events_ptr:** Pointer into the region of X: data space where event information (coordinate and intensity) should be recorded. Note that dereferencing of this pointer must be done in (presumably in-line) Assembly, as the C code is compiled to use Y: data space.

- **numevents:** The number of events recorded so far. If this number exceeds the maximum (and hence **events_ptr** threatens to leave the space allocated for event recording), future events for this frame are not recorded.

- **one_line_delta:** The number of words a single row occupies in the pixel stream. If you are standing on a word in the pixel stream which corresponds to coordinate (c, r), going **one_line_delta** words downstream (toward earlier-arrived data) puts you on $(c, r-1)$, and going **one_line_delta** words upstream (toward later-to-arrive data) puts you on $(c, r+1)$. See subsection 5.2.1 for a discussion of this phenomenon.

- **left_off:** Pointer into the video buffer indicating the next word to be read. (The "receiver runner" from section 4.3's description of circular buffers.)

- **this_line_start:** Pointer into the video buffer indicating the first word in the row currently being processed, *i.e.*, the word immediately following the row sync.

- **lines_rcvd:** The number of lines confirmed to have been received into the video buffer for this frame. When a full row's worth of data (**one_line_delta** words) has

been read into the video buffer since the end of the last row, the FEP checks for a row sync at the beginning of these new data; if found, **lines_rcvd** is incremented by one.

- **waiting_for_frame_sync**: Flag indicating whether the FEP is currently searching the incoming video data for a frame sync, or whether it is processing a frame. When all of the rows in a frame have been received (including vertical overclocks), **waiting_for_frame_sync** is set to search for the next frame's sync. Note that setting **waiting_for_frame_sync** while the current frame's rows are still being received effectively aborts processing of this frame.

The variable **ccd** is a **ccd_param_struct**, which embodies the state of the Lasagna Box as it applies to the FEP.

- **mode**: The transmission mode of the Lasagna Box (0, 1, 2, or 3).
- **used_cameras**: Two-bit value representing which cameras have data in the pixel stream. 0x1 means camera 0; 0x2 means camera 1; and so 0x3 means both cameras. This is another encoding of the "Submode" referenced in section 4.4: **used_cameras** = 0x1 means Submode 0, = 0x2 means Submode 1.
- **num_hucs**: The number of horizontal underclocks read out per slice.
- **num_cols**: The number of image pixels read out per slice.
- **num_hocs**: The number of horizontal underclocks read out per slice.
- **total_cols**: The total number of columns per slice (= **num_hucs** + **num_cols** + **num_hocs**).
- **num_rows**: The number of rows containing image pixels read out per frame.
- **num_vocs**: The number of rows which are vertical overclocks read out per frame.
- **total_rows**: The total number of rows per frame (= **num_rows** + **num_vocs**).

- **start_row:** The "Y" value of the first row received after the frame sync. Non-zero if the Lasagna Box begins reading out at a line higher than the first CCD row in the camera. Event Y coordinates are thus a combination of **lines_rcvd** and **start_row.**

The variable **bad** points to a **bad_locs_struct**, which keeps track of the number of bad columns and pixels, the X coordinate of the bad columns, and the X and Y coordinates of the bad pixels (held in separate variables instead of a single two-dimensional array for faster access times). Camera number is encoded in bit 15 of applicable X coordinates, the most significant bit of these unsigned short values. A bad tenth column in camera 0 is thus encoded as **bad_columns**[n] = 0x0009, and a faulty pixel at camera 1's (5, 32) is **bad_pixels_x**[n] = 0x8005 and **bad_pixels_y**[n] = 0x0020. See Figure 6-1 for more information.

Two variables of type **vidbuf_param_struct** are kept: the **vidbuf_params** of the current frame (where a frame lasts from its frame sync to the next frame's sync), and the **next_vidbuf_params** which become current upon the arrival of the next frame sync. When the GPP sends a command to change any of the frame-processing parameters, the FEP modifies **next_vidbuf_params**; thus changes in the Lasagna Box state, slice thresholds, and the character of event reporting, result in modification of **ccd** and **one_line_delta**, **thresh**, and **do_sifting** and **send_z**, respectively *within* **next_vidbuf_params.** In addition, within **next_vidbuf_params**, **id** is zero to reflect that these parameters are not current; **lines_rcvd**, **waiting_for_frame_sync**, and **numevents** are also zero; and **events_ptr** points to the beginning of the event storage space. Then when a new frame sync comes along, the **left_off** pointer in **vidbuf_params** is copied to **next_vidbuf_params**, and then **next_vidbuf_params** is simply transferred *en masse* into **vidbuf_params**, reflecting that these values are now current. If the GPP has changed any of the parameters, these new parameters now take effect; otherwise, the result is simply the resetting of **id**, **lines_rcvd**, **waiting_for_frame_sync**, **numevents**, and **events_ptr.**

This bears repeating: **vidbuf_params** contains the parameters applicable to the *current* frame; **next_vidbuf_params** contains the parameters applicable to the *next* frame to come along the video channel, parameters which go into effect upon the next frame

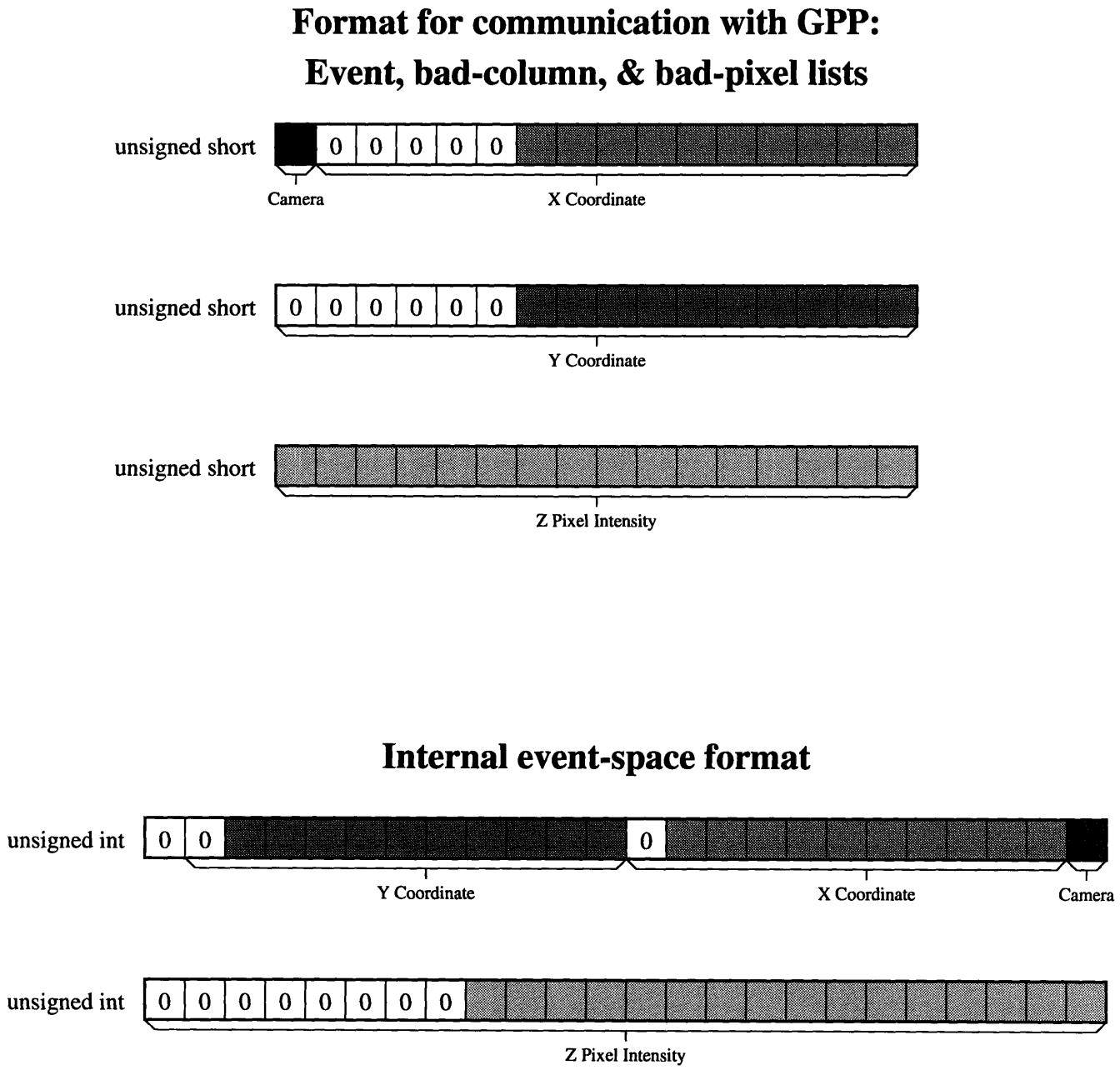


Figure 6-1: Encoding Formats for Events & Other Coordinate-Related Data

sync.

There is one caveat to this pattern. To save memory space, there is only one bad_param_struct allocated; the bad variables in both vidbuf_params and next_vidbuf_params point to this single structure. Thus when the GPP orders a change to the bad column or bad pixel list, the effect is immediate, as any new events found in the frame currently being processed will be compared to this new list. This is reasonable, and perhaps even advantageous.

6.1.1 Memory & Resource Allocation

The DSP's three memory spaces — X:, Y:, and P:, each spanning 32 kilowords — are divided by the FEP into definite regions, illustrated in Figure 6-2. P: space has reset and interrupt vectors from P:$0000 to P:$003F, short-addressable space from P:$0040 to P:$01FF, and long-addressable space from P:$0200 to P:$7FFF. In addition, P:$0000→$0AAA is initially set to point to boot ROM, so the loader cannot access this area.[2] So the majority of the code lies in P:$0AAB→$7FFF, with the **crtrzp.asm** reset code starting at P:$0AAB. Since short-addressable space allows for faster jump instructions, time-critical code — specifically, that in **FEPintr.asm** and **FEPproc.asm** — is placed in this space. Because of the ROM boot area coverage, however, the code for these routines, as well as the interrupt vectors, must be explicitly written to this area by the FEP itself.

X: space currently consists only of event recording, IPP variable storage, and the video buffer. With an 8K video buffer — ~four lines at the maximum expected line length, which occurs when all 1024 columns of both cameras are read out in Modes 1 or 2 — and just a few words set aside for IPP, there is space for over 12,000 events (at two words apiece) to be recorded per frame.

Y: space is heavily divided. The first bunch of words are allocated in **FEPequ.asm** to be short-addressable variables for Assembly routines; these variables must end up in Y:$0000→$00FF to be short-addressable, or preferably →$003F for short-addressable

[2]Future versions of the spacecraft hardware may change this.

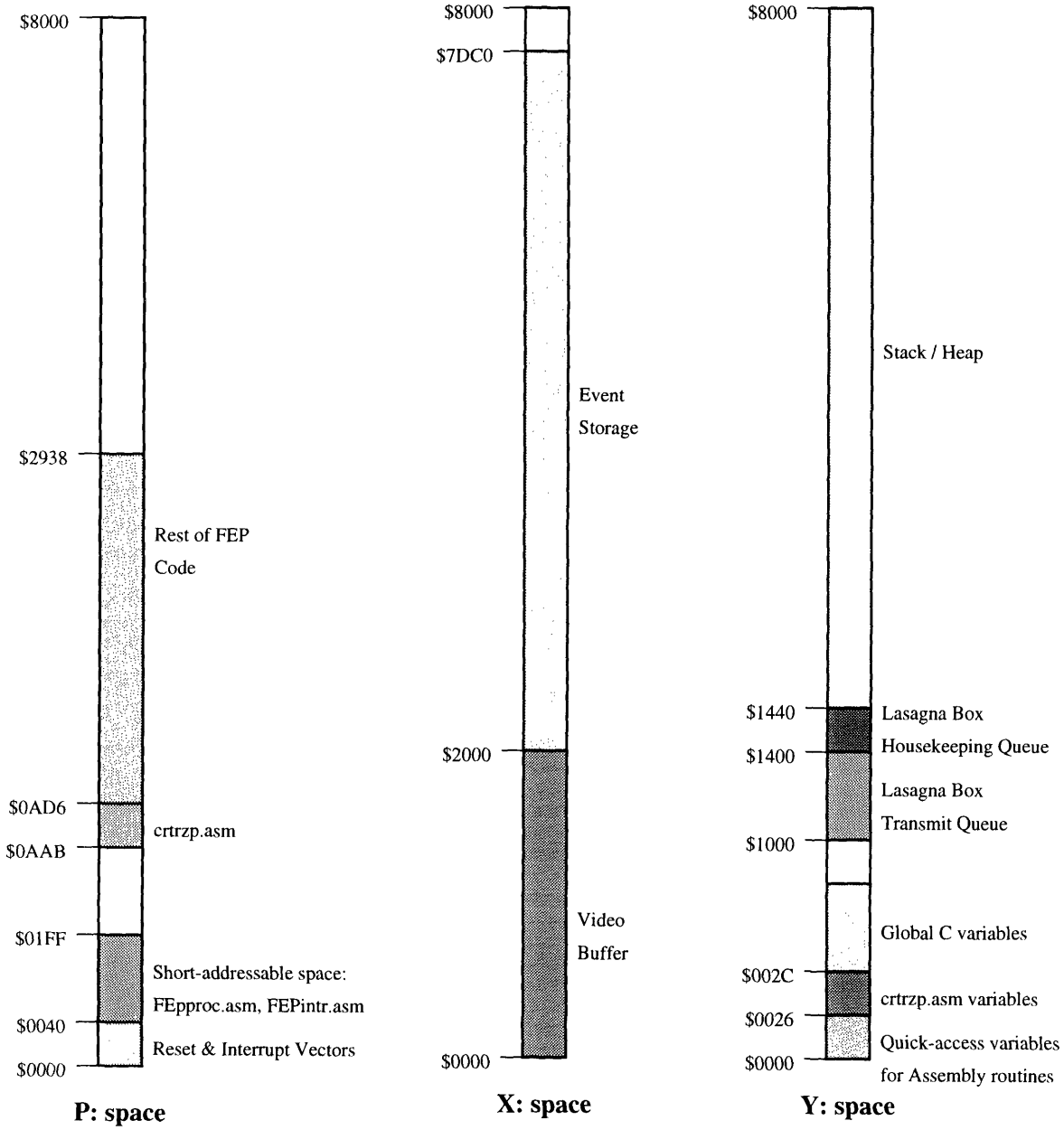


Figure 6-2: FEP RAM Memory Map

internal RAM (faster access times than short-addressable external RAM when external P: RAM is being used). After this comes compiler variables (memory limit, stack safety margin, *et cetera*), allocated in **crtrzp.asm**, and then C global variables. The Lasagna Box command / housekeeping queues (buffers) are placed above the global space, page-aligned so that the 56001's modulus registers M*n* may be used to facilitate operation of these circular buffers. 1K is alloted for the transmit queue SSI_TXQ, and 64 words for the housekeeping queue SCI_RXQ. The stack pointer R6 is initialized to start above these buffers, and the remainder of Y: space (Y:$1440→$7FFF) is the stack / heap.

Since the C compiler is not allowed to use register R5, this register is allocated to the video channel receive interrupt routine. R5 is declared to be the video buffer write pointer (the "sender runner" from section 4.3's description of circular buffers). Hence the SSI receive interrupt (indicating a new word received over the video channel) is a short interrupt which simply reads movep x:<<m_rx,x:(R5)+, that is, "Take the word just received, put it in X: space at the address indicated by R5, and then increment R5 in anticipation of the next word". The modulus register M5 is set once to keep R5 within the video buffer; after that, the 56001 hardware takes care of the behavior of R5 automatically.

6.2 Initialization and Normal Function

Upon reset, the FEP starts running at address P:$0000. A jump instruction here pushes program flow to P:$0AAB, above the ROM boot area, governed by **crtrzp.asm**. Hardware initialization — resetting the DSP stack pointer (SP), C code stack pointer (R6), peripheral hardware — is performed, and program flow is then switched to the **main()** routine in **FEPmain.c**.

main() declares vidbuf_params and next_vidbuf_params; they are generally passed by reference (*i.e.* pointers to these structures are used by most other routines), not only to allow for modification, but to save calling time. The former is generally abbreviated vbp, and the latter vbpnew. **main()** also allocates space for the bad_locs_struct

in the stack. Note that since these three structures are declared in **main()**, this space will not been overwritten. Control passes to **onetime_init()** in **FEPsetvars.c**, which performs the following initializations:

1. Initialize P:$40–AAA
2. Initialize serial communications interrupt vectors: SSI transmit and exception (command channel); SSI receive and exception (video channel); and SCI receive and exception (housekeeping channel)
3. Register the FEP with the Constellation Name Server, and get default destination addresses
4. Set up variables for buddy-check (the regular reports to the GPP stating the FEP is still alive)
5. Set up variables for command / housekeeping functions
6. Make R5 point to the beginning of the video buffer, and set M5 to keep R5 within the video buffer
7. Set up **vbpnew** to sensible default values. In particular, ensure **id**, **lines_rcvd**, **waiting_for_frame_sync**, **numevents**, **bad->num_bad_columns**, and **bad->num_bad_pixels** are all zero, and that **events_ptr** points to the beginning of event-recording space. Set **vbp->left_off** to the beginning of the video buffer, and "abort the incoming stream" (set **vbp->waiting_for_frame_sync**) to set the FEP looking for the next frame sync to come over the video channel, at which time these default parameters in **vbpnew** will become current.
8. Initialize serial communications variables (serial communications control registers, and pointers into the command / housekeeping queues)
9. Turn the serial communications interrupts off or on, according to whether we are or are not in simulation mode, respectively

Back in **main()**, control passes into the mainloop. The mainloop performs the following steps in repeated succession forever:

1. **check_buddy_time()**: Tell the GPP the FEP is alive
2. **try_get_message()**: See if an IPP message has arrived, and if so, process it
3. **simulate_*xx*()**: If the FEP is in simulation mode, fabricate pixel data and place it in the video buffer as though it arrived over the video channel; also empty the command channel transmit queue, as the SSI transmit interrupt would be doing in non-simulation mode
4. **try_proc_vid()**: Try to sync with the data which arrived during the previous mainloop; check a single row of data for events if possible
5. **parse_tdb_command()**: Insert consequential (*i.e.* non no-op) commands into the transmit queue if appropriate
6. **try_fill_lbcmd()**: Fill empty space in the transmit queue with no-ops
7. **try_relay_hk()**: Check the housekeeping queue for responses from the Lasagna Box; if a full set of responses has been received, send it to the process which sent the associated commands
8. **try_relay_errors()**: If any exceptions have occurred during the previous mainloop (such as an overrun or underrun in the serial communications system), warn the GPP

Note that for testing purposes, the FEP can be instructed to skip steps 3 through 8, effectively freezing much of the state of the FEP while letting it continue to report its live status and receive commands. The user can then instruct the FEP to execute a single mainloop (*i.e.* run through steps 3–8 once), and then observe how the state of the FEP has changed.

6.2.1 Serial Communication Interrupts

As outlined in section 4.3, the FEP works under a "interrupt-plus-main-code" system; thus, operating concurrently with the mainloop described above, is a set of interrupt-driven processes. Aside from the interrupts used by the underlying IPP system (host interface, direct memory access, and the like), the FEP utilizes six other hardware interrupts for use in serial communication with the Lasagna Box. There is one regular and one exceptional interrupt each for the command channel (SSI[3] transmit), video channel (SSI receive), and housekeeping channel (SCI[4] receive). The SCI transmit line (TXD) is used as the master reset line for the Lasagna Box. For information on this and other electronic signals used by the DSP for communication with the Lasagna Box, see table 4.1.

Interrupt vectors are initialized by **init_intr_vectors()** in **FEPasmsupport.asm**. As described above, the SSI receive interrupt is the fastest, as it is assumed to be the one called most often (once every $6\mu s$). It reads simply `movep x:<<m_rx,x:(R5)+`, that is, "Take the word just received, put in X: space at the address indicated by R5, and then increment R5 in anticipation of the next word". The other five (including SSI receive exception) are jumps to long interrupt handlers, coded in **FEPintr.asm**.

The two regular-interrupt handlers for SSI transmit and SCI receive are similar in structure. They put aside the values in the registers they will be using (R1 and M1), then reload these registers with the pointer and modulus (respectively) for their circular buffers. The SCI receive interrupt takes the word just received and saves it in the housekeeping queue `SCI_RXQ`; the SSI transmit interrupt takes the next word in the transmit queue `SSI_TXQ` and puts it in `m_tx` to be transmitted to the Lasagna Box. Both routines post-increment R1, and save this updated pointer to be re-loaded the next time this routine is called. They then update variables indicating how many times the interrupt has been called, so the main-code routines have an easier time figuring out how many new words have been written or read since the last time these variables were checked. The pre-interrupt values of R1 and M1 are restored, and the

[3]Synchronous Serial Interface
[4]Serial Communications Interface

handler returns.

The exception handlers for these routines simply read the appropriate status register into a variable, clearing the 56001's exception flag, and pass control to the regular handler to try to re-establish normal operation. The SSI receive exception handler is similar, in that it also reads the applicable status register into a variable, and then performs the **movep** x:<<m_rx,x:(R5)+ instruction. The main-code routine **try_relay_errors()** checks if any of these exception status variables are non-zero, and if so, sends a warning message to the GPP stating that the error occurred and including the status register value. It then zeros the exception status variable to make sure only one message is sent per error.

6.3 Communications

The FEP communicates with the GPP[5] on the one side, and with the Lasagna Box on the other. Low-level interaction for the former is performed by the underlying IPP system, and for the latter by the interrupt handlers described above. High-level communication is governed by **FEPcom.c**.

6.3.1 GPP Communication

All procedures in the FEP wishing to send information to the GPP use the function **Send_Host_Message()**, which packages the information given it into an IPP message and sends the message. **try_get_message()** uses **Get_Host_Message()** to receive messages from the GPP. If a message is available, **try_get_message()** then checks the message type, and processes the data accordingly. Note again that modifications to video processing parameters affect **vbpnew**, that is, **next_vidbuf_params**, and hence go into effect upon the arrival of the next frame sync along the video channel.

The FEP keeps track of the IPP addresses of three processes: the most recent

[5]As noted in section 4.3, statements like "communicate with the GPP" actually mean "communicate with processes using IPP in the HETE computer system". For the DSPs, this is usually the GPP above them.

sender of PROCESS_VIDEO is saved in **video_dest**; the most recent sender of a command containing Lasagna Box commands is saved in **lb_dest**; and the most recent sender of a test or debugging message is saved in **testmsg_dest**. The FEP responds to a great deal of test messages — including "peek" and "poke" commands, parameter dump requests, and commands concerning holding off execution of the mainloop — under the message type **viddsp_command**. These test messages are handled by **parse_viddsp_cmd()**.

The Lasagna Box command buffering described in section 5.1 adds a piece of complexity to this operation. Global space is allocated for three IPP messages, to be used in a circular fashion. The "free" message, *i.e.* the space into which new messages may be written, is indexed by **IPPbufTail**; if the new message contains Lasagna Box commands, **IPPbufTail** is incremented, and the mainloop function **parse_tdb_command()** recognizes that there is a message containing Lasagna Box commands (indexed by **IPPbufHead**) waiting to be transmitted. **parse_tdb_command()** then follows the algorithm outlined in subsection 5.1.2, placing commands into the transmit queue, while ensuring that no more than one read command is pending at any time. If too many Lasagna Box command messages are being stored, such that **IPPbufTail** threatens to overtake **IPPbufHead**, **try_get_message()** declines to accept any new messages until space for a new message is freed.

6.3.2 Lasagna Box Communication

parse_tdb_command() — so named after the original Lasagna Box command program, tdb — first checks if there is a message containing Lasagna Box commands waiting to be transmitted. If so, it picks up where it left off in this message during the previous mainloop (or starts at the beginning if this is a new message), and goes through the message one command at a time. Note that since commands are 24 bits, two IPP message spaces are used to encode them, with the least significant 16 bits preceding the 8 most significant bits.

For each command, **parse_tdb_command()** first checks if it is a special word — a command that is not to be sent to the Lasagna Box, but rather parsed by

the FEP itself to have an ultimate effect on the Lasagna Box. The most important of these special words (or rather, the only one expected to be used in-flight) are TXD_ON_CMD and TXD_OFF_CMD, which control the master reset line into the Lasagna Box.

For all other words, **parse_tdb_command()** tries to write the command into the transmit queue. If the write was successful, the procedure checks if the command was a read command, *i.e.* one which will generate a response over the housekeeping channel: if so, it sets a flag noting that there is a read command pending; otherwise it proceeds to the next command in the message and repeats the write process. **parse_tdb_command()** returns when (1) it has run out of commands, (2) a read command is pending (no response has been noted from the housekeeping channel), or (3) a command-write was unsuccessful (the transmit queue is full).

try_relay_hk() monitors the housekeeping queue. It holds an IPP message which it fills with read commands and their responses as they arrive, and sends this message when all of the commands in the current Lasagna Box command message have been sent. If a read command is pending and two 8-bit words (= 1 16-bit response) have arrived in the housekeeping queue, **try_relay_hk()** adds the command which prompted this response and the response itself to the IPP message it is holding, updates variables to note that this word has been read from the queue, and clears the read_pending flag. **try_relay_hk()** will report an error to the GPP if: (1) a response arrived over the housekeeping channel when no read command was pending; (2) more than one response arrived over the housekeeping channel when only a single read command was pending; or (3) a read command has been pending for too long without response from the Lasagna Box.

The video channel receive system is discussed in the following section.

6.4 Video Processing

Processing of data in the video buffer has two components: synchronization, and event-finding. Synchronization is handled by **FEPvid.c**; the core of event-finding is

in **FEPproc.asm**, which then uses **FEPvid.c** to calculate and record the coordinates of an event once located.

try_proc_vid is the center of video processing, being the mainloop routine which calls the rest of the synchronization and event-finding routines. This procedure begins by checking if the FEP is in waiting-for-frame-sync mode. If so, it calls **sync_frame()** to see if the frame sync has arrived in the video buffer yet; if the new frame is synced, the **next_vidbuf_params** are made current, and the ID encoded in the frame sync is recorded; if this frame is to be procesed, the GPP is informed of the frame sync's arrival, and variables used by **proc_line()** are set.

switch_params() takes care of making **next_vidbuf_params** current. It first preserves **vbp->left_off**, then sets all the values in **vidbuf_params** to the corresponding values in **next_vidbuf_params**. The new parameters are now current, and **next_vidbuf_params** now represent the *next* frame's parameters. **switch_params()** now checks for the kind of processing this frame entails: if this is a one-frame-only processing situation (evidenced by **send_z** being set), then clear the *next* frame's **do_sifting** flag; if the present frame is not to be processed, revert immediately back into waiting-for-frame-sync mode.

If the FEP is not in waiting-for-frame sync mode, it is in find-events ("sift") mode. **try_proc_vid** checks if there is a full row's worth of new words in the video buffer. If so, it checks for a row sync at the beginning of this new data, warning the GPP and aborting processing if the sync is absent. Assuming the sync is found, however, **vbp->lines_rcvd** is incremented, and if this means there are three image rows available in the video buffer, the middle row is processed. **vbp->this_line_start** is set to the beginning of the image-pixel region of the row being processed (*i.e.* the row immediately preceding the one just synced), **procstart** is set to the second package-of-four-words in this image-pixel region, and the row is processed by **proc_line()** in **FEPproc.asm**. See section 5.2 for more description of the reasons for these calculations.

Once the row is processed, **vbp->left_off** is updated to point just past the end of the row just synced. If all of the image rows have been processed, and hence the

event list is complete, the event list is sent to the GPP. If all of the rows have been received, including vertical overclocks, inform the GPP that the FEP has completed processing this frame, and return to waiting-for-frame-sync mode.

Note that while processing a frame, **vbp->left_off** is in one of two positions in **try_proc_vid()**: it is either immediately before the expected position of the next row's row sync, including preceding blank words (*i.e.* one word beyond the previous row's last pixel word, or one word beyond the frame sync); or immediately after the row sync just located (*i.e.* at the first word of the first package-of-four for this row).

FEPvid.c uses certain basic utilities to move around the video buffer. They are implemented as macros for improved speed, and as such *can modify variables in the argument list which appear to be "passed by value"*. That is, macros can change variable values given only the variable name; this is different than functions, which require pointers to variables in order to modify them.

- **read_vb(int *ad, int val)**: Read the video buffer at address **ad**, and put the value into **val**. Conceptually identical to **val** = ***ad**, but reads from X: instead of Y: space.

- **vb_add(int *ad1, int ad2)**: Return the address resulting from starting at address **ad1** and moving **ad2** words in the video buffer. Let's say the video buffer extends from 0x6000→0x7000: x = **vb_add(0x6002, -4)** sets x to 0x6FFE — start at 0x6002, go back 1 to 0x6001, back 2 to 0x6000, back 3 loops around the circular video buffer to 0x6FFF, then back 4 to 0x6FFE.

- **vb_distance(int *from, int *to)**: Return the number of words in the video buffer which separate **to** from **from**. That is, how many words would one have to **vb_add()** to **from** to get to **to**. So in the previous example, x = **vb_distance(0x6005,0x6007)** sets x to 2, while x = **vb_distance(0x6007,0x6005)** sets x to 0x0FFE.

- **have_words(int *read, int *write, int num)**: Returns TRUE if there are **num** words separating **write** from **read**, FALSE otherwise. Hence x = **have_words(0x6005,0x6007,2)** sets x to 1 (TRUE), while x = **have_words(0x6005,0x6007,3)** sets x to 0 (FALSE).

vb_add() and **vb_distance()** rely on a particular orchestration of the video buffer. They require that the length of the video buffer is a power of 2 — or more to the point, that there is only a single 1 in the binary representation of this length. With this scenario, the "relevant bits" in a video buffer address are all the bits less significant than this singular 1 — these bits are the offset from the bottom of the video buffer for **vb_add**, and the only nonzero bits **vb_distance** can produce. Thus binary AND operations can be used to simplify these calculations.

6.4.1 Synchronization

At time of code development, sync patterns were believed to be composed of *zero marks* and *one marks*, where a zero mark is four consecutive words of value 0, and a one mark is four consecutive words of value 0x00FFFF. The frame sync was thus:

1. A large but unpredictable number of blank words (0's)
2. A one mark
3. Another one mark
4. 24 packages-of-four, each one either a zero mark or a one mark, to encode the Frame Identification number ("frame ID"). A zero mark meant a 0 in the 24-bit frame ID; a one mark, a 1. The first package-of-four represented the most significant bit of the frame ID.

A row sync was a zero mark followed by a one mark. There was to be a small but fixed number of blank words (value 0) preceding every row sync; this number is defined as ROW_SYNC_SKIP in **FEPincl.h**. With these patterns in mind, the **check4()** routine was created, which returned a number indicating whether the next four words in the video buffer represented a zero mark, a one mark, or neither, returning ZEROS4, ONES4, or MIX4, respectively.

sync_frame() was designed as a state machine:

STATE 0 If a zero-to-0x00FFFF transition is located in the video buffer, indicating a possible start to the frame sync, go to STATE 1.

STATE 1 If the next four words are a one mark, go to STATE 2; else go back to STATE 0.

STATE 2 If the next four words are a one mark, the next 24 words are probably the frame ID: reset the variables concerned with reading the ID, mark this point in the video buffer in case something is wrong with the frame ID and we want to look at this part of the video buffer again, and go to STATE 3. Otherwise go back to STATE 0.

STATE 3 Check the next four words for either a zero mark or a one mark. If one of these is found, record it as part of the frame ID; if a MIX4 is found, what we thought was the frame sync actually was not, so go back to where we were in STATE 2 and look for the frame sync again (go back to STATE 0). If 24 one and zero marks are read, the ID has been found and the frame synced — report this and return to STATE 0 for the next frame.

sync_row() skips ROW_SYNC_SKIP (presumably blank) words, looks for the zero mark, and then for the one mark. If it finds these, it has found the row sync; if not, there is something wrong.

Both sync functions return if they run out of new data in the video buffer, ready to try again during the next mainloop.

6.4.2 Event-Finding

This is where the algorithm presented in section 5.2 is implemented. **FEPvid.c** sets up several variables for use by **proc_line()**:

- asm_thresh*n*: Background thresholds for each slice, as defined in table 5.2.2. These values are justified to match the location of the pixel being compared within its word (see below).

- asm_proclen: Set to the number of active image columns per slice, minus two. This reflects that all of the line's packages-of-four containing image pixels are to be processed, with the exception of the first and last.

- **procstart:** Address within the video buffer, set to the second package-of-four within the image-pixel portion of the line being processed.

- **up1, down1:** Number of words in the video buffer to go one line upstream (to the line which followed the line being processed, *i.e.*, the line just synced) and one line downstream (to the line which preceded the line being processed), respectively.

proc_line() begins by saving the registers it will be using on the stack. It then reads the parameters passed to it and stores them for quick access. R3 is used as the read pointer into the video buffer; M3 is set to keep R3 within the video buffer. **proc_line()** then checks whether the pixel data in the video buffer is 12- or 16-bit (*i.e.* Mode 3 or not), and calls the appropriate processing routine.

There are a few pratfalls that this code must be aware of, and a few tricks it uses liberally:

- The 56001 assumes 24-bit data are in 2's-complement format, so calculations and more importantly comparisons are conducted accordingly. To ensure sensible comparisons with unsigned data of 23 bits or fewer, the data is right-shifted by one bit, forcing the MSB to zero. Thus for left-justified 16-bit data, **asm_threshn** needs to be shifted left eight bits and then right one; for 12-bit pixels in the high half of the word, **asm_threshn** needs to be shifted left twelve bits and then right one; and for 12-bit pixels in the low half of the word, no shifting is necessary.

- For parallel moves involving a variable used in the primary instruction, such as `clr A A1,y:(R6)+` or `cmp X0,A x:(R3)+,A1`, *sources* are taken *before* the primary instruction, while *desinations* are modified *after* the primary instruction. Hence in these two examples, A1 would be saved to y:(R6) *before* A is cleared, but A1 is loaded with the value from x:(R3) *after* it is compared to X0.

- Since DO loops cannot end in a jump instruction, loops that logically ought to end with one are "rotated" so that the logically first instructions in the loop is

"pulled around" to be the last instructions encountered. This means that these "pulled around" instructions need to be repeated before entrance to the loop for the first iteration, and that when the final in-loop instructions are encountered for the last time, they must have no ultimate effect.

proc_16_bit() is for processing data in Modes 0, 1, or 2. It assumes pixels will be in the high 16 bits of the 24-bit words, with the lower 8 bits being $FF. Y0, N3, and the SHIFT flag are loaded for use in the **seek_event()** routine. Two "pulled-around" instructions precede the loop. Each loop iteration dispatches one package-of-four, so R3 being started at **procstart** and **asm_proclen** being set to **num_cols** - 2 means all of the image pixels in the line save those on slice boundaries will be processed. Each word is compared to the appropriate threshold, as described in **Process Line** (16 bit), in section 5.2. Note that since the pixels are left-justified, and then shifted one bit to the right to clear the MSB, the thresholds loaded into **asm_threshn** must be shifted up by 7 bits. This is accomplished in **load_thresh()**. Note also that since pixels are allowed to *meet* their thresholds and still be considered "above threshold", the extra 1's in the lower bits of the word do not affect the algorithm. Pixels which exceed threshold trigger a jump to **ge_thresh_16()**, which falls directly through to **seek_event()**.

proc_12_bit() is for processing data in Mode 3. It assumes pixels will be in both the high and the low 12 bits of the 24-bit words. Y0, and N3 are loaded for use in the **seek_event()** routine, with Y0 defaulting to "low-pixel" condition (that is, it needs to be changed when the event candidate is a high pixel). Three "pulled-around" instructions precede the loop. Each loop iteration again dispatches one package-of-four, as in **proc_16_bit**, so again all of the image pixels in the line save those on slice boundaries will be processed. The high and low pixels in each word of these packages-of-four are shifted or masked as necessary, and compared to the appropriate thresholds, as described in **Process Line** (12 bit) from the same section. Note that again shifting of the high-pixel thresholds is necessary, and performed by **load_thresh()**. Testing of high pixels is also similar to the testing of 16-bit pixels in that the extra bits from the low pixel do not impact the algorithm. Since the high

12 bits of the word are zeroed before comparison, there are no extraneous bits in the comparison of low pixels.

Threshold-exceeding pixels found by **proc_12_bit()** are sent to either **ge_thresh_hi()** or **ge_thresh_lo()**: the former, for high pixels, resets Y0 to mask the high pixel (shifted one bit to the right, recall) and sets the SHIFT flag to note that values should be shifted one bit to the right before comparison to avoid negative-value concerns; the latter, for low pixels, clears the SHIFT flag — Y0 is defaulted to mask the low pixel. These function then pass to **seek_event()**.

The **seek_event()** routine is what checks if a pixel is a local maximum, following the **Check for Local Maximum** algorithm, again described in section 5.2. It uses Y0 and the SHIFT flag to be a general-purpose routine, not caring whether it is comparing 12 or 16 bit pixels.

seek_event() starts by saving A1 (the next word to be compared in the **proc_*n*_word()** routine) and the R3 pointer, for clean return into the threshold-comparison loop. It then moves R3 back by N3, so it points to the pixel which exceeded its threshold and prompted the call to **seek_event()**. It loads this word into A1, shifting it one bit to the right if necessary. It then masks only the bits belonging to this pixel — note that since both greater-or-equal and strictly-greater tests need to be performed, the "extra bits don't matter" trick can't be used — and prepares to compare the pixel in question (hereafter referred to as the Center pixel) to its Right neighbor by setting N3 to 4.

The core of the **seek_event()** procedure is the **compare()** routine, which assumes A contains the Center pixel, N3 contains the offset from R3's present location to the pixel against which it should be compared, and Y0 and the SHIFT flag are set to reflect the position of the pixel in the word. **compare()** moves R3 by N3, loads B1 with the word found at this new location, shifts it one bit to the right if necessary, and masks out the relevant bits. It then compares B with A, setting the status bits accordingly, and returns. Status bits report the Center pixel with respect to the neighbor: if the status indicates "greater-than", then the Center pixel is strictly greater than the neighbor; conversely a status indicating "less-than-or-equal" indicates the Center

pixel is *not* strictly greater than the neighbor.

seek_event() then proceeds around the Center pixel's eight neighbors as indicated in Figure 5-2, adjusting N3 to move from one pixel to the next, and calls **compare()** each time. After each **compare()** call, **seek_event()** checks if this neighbor precludes the Center pixel from being a local maximum, and if so, it restores R3 and A1 and returns to the threshold-comparison loop that called it. If the Center pixel passes all eight comparisons, it is deemed an event, and all relevant data is passed to **record_event()** in **FEPvid.c**.

First order of business in **record_event()** is to determine the in-image coordinates of the event. **get_coords()** checks how far into the image-pixel portion of the line the event is, by comparing it to vbp->this_line_start. It then checks how many packages-of-four this represents, and which word within the package-of-four held this pixel (and for Mode 3, whether it is the high or low pixel in this word). With this information, plus the transmission mode and submode, the X coordinate and camera number may be calculated. The Y coordinate is two less than the number of lines received: for instance, when three lines have been received, the line being processed is the line Y=1. This number is offset by the start_row given by the GPP.

This coordinate is compared to the bad column and pixel lists; if it is deemed a valid event, the pixel value is right-justified, and the event information is saved in compressed form. Each event takes two words to save: the first word contains the coordinate and camera number, the second word the pixel value. The camera number is saved in bit 0 of the first word, the X coordinate in bits 1–11, and the Y coordinate in bits 12–22: thus coordinates from (0,0) on camera 0 to (2047,2047) on camera 1 may be saved (which far exceeds the possible range). The pixel value takes up at most the lower 16 bits of the second word. Figure 6-1 shows this format. The event is saved in X: memory space, vbp->numevents incremented and tested against its maximum, and control returns to **seek_event** in **FEPproc.asm**.

Note that this algorithm relies heavily on the projected characteristic of HETE UV camera images: specifically, only around 1% of the pixels should exceed threshold, and only a portion of these should actually be locally-maximal events. In this vein,

the threshold-comparison loop is very tight and time-optimized, the local-maximum check almost as dense (though it sacrifices some speed for program space and ease of readability, by jumping to the **compare()** routine instead of doing the comparison in-line), and the event-recording is much less time-optimized for relative ease of readability. Thus *setting of the thresholds is crucial to the operation of the FEP.* A threshold set too low will generate far too many calls to **seek_event()**, and with a new word arriving over the video channel every 6μs, the FEP could easily lose ground to the incoming data. It will most likely report this by experiencing a row sync error, as the row sync would have been overwritten by new data as the FEP fell behind. Hence a row sync error reported by the FEP but not the BEP, is the first hint that the background thresholds ought to be changed.

The FEP reports the events it has found in a frame to the GPP using **dump_events()** in **FEPcom.c**, which sends the information — including or excluding the pixel intensities, as previously requested by the GPP and noted in **vbp->send_z** — in a series of IPP messages.

Chapter 7

Back-End Processor Functionality

The engineering task involves trading off among various alternatives until a solution is identified which fits within the various constraints and still satisfies the functional requirement.

— *Robert M. McDermott*
Computer-Aided Logic Design

7.1 The Frame Buffer

Integral to the BEP's work is its interaction with the 4 Megaword (12 Megabyte) Frame Buffer. It is into this frame buffer that video data is saved and from it retrieved according to the wishes of the GPP. Very little can be traded off to conform to the optimization constraints the BEP needs to follow:

- Writing to the Frame Buffer must be fast, to keep up with the data arriving over the video channel
- Frame Buffer space must be used efficiently, to maximize the number of individual frames that can be held simultaneously
- Reading from the Frame Buffer must be simple to use, because other developers will be adding capabilities to the BEP that rely heavily on this function

- Reading from the Frame Buffer must be fast, because this function will be used very frequently

With time, storage space, and ease of interface all claiming optimization priority, program space becomes the only readily apparent resource that can "give way" to these concerns. The end result is a relatively large set of small, very fast low-level routines, to which more general-purpose (and hence more flexible, easier to interface with) routines dispatch. Another optimization step is pre-computation: useful values (the total number of pixels in an image, for instance) which are derivable from basic parameters (number of pixels per slice row, number of slices, number of rows) are calculated once when new parameters arrive, and stored for quick access as the various routines require them.

The BEP interacts with the Frame Buffer on two levels. At a low level, read/write operations are performed on the Frame Buffer via eight independently oriented 4-kiloword windows. At a higher level, the Frame Buffer is considered to be composed of a set of *regions* within which a number of same-parameter frames are stored.

7.1.1 Low-Level Frame Buffer Interaction: Windows

The Frame Buffer is evenly divided into 1024 4-kiloword pages. The BEP assigns eight 4-kiloword regions of memory to be windows into the Frame Buffer: each of these windows may be set to view a single page. Read and write operations to a window then become read and write operations to that page in the Frame Buffer.

Example: a window that extends from $8000 to $8FFF in BEP X memory space is set to page $555. I/O to this window is therefore I/O to Frame Buffer addresses $555000 to $555FFF. Let's say you write $123456 to X:$8111: now Frame Buffer address $555111 holds $123456. Now let us set another window, say Y:$A000→$AFFF, to page $555 as well. Reading Y:$A111 would produce the value $123456.

As Figure 8-1 shows, four windows are placed in the BEP's X memory space, starting at $8000, $9000, $A000, and $B000, and the remaining four are in Y space, starting at the same addresses. The adjacency of the windows within a memory space

allows larger aggregate windows to be formed: setting the X:$8000 region to page $555 creates a 4-kiloword window; setting the X:$9000 region to page $556 expands this to an 8-kiloword window; then setting the X:$A000 region to page $557, a 12-kiloword window; and finally setting the X:$B000 region to page $558, a 16-kiloword window. The BEP uses this ability when dealing with a block of data. A single line of video data, for example, is no more than two kilowords in length. If the BEP wishes to dump this data directly into the Frame Buffer, the write will involve at most two pages: so the BEP can set window w to page p where it wishes to start writing, and then window $w + 1$ to page $p + 1$; it may now perform a memory copy with no further concern about Frame Buffer pages and windows. Similarly a multi-word read (spanning under 4 kilowords) will require at most three windows: if it starts in page p, it can either need to move forward into page $p + 1$, or backward into page $p - 1$, or neither. Setting three adjacent windows accordingly allows for these eventualities without the actual read operation being concerned about crossing from one window to the next.

7.1.2 High-Level Frame Buffer Interaction: Regions

The BEP divides the Frame Buffer into a number of regions (presently three), wherein frames are stored. Each of the frames within a region has a similar structure — the same size, shape, storage mode, *et cetera* — and differ only in starting point, frame identification number, and sequential order within the region. The sequential order of frames is kept by organizing them as a linked list, whereby each frame points to a "next" frame, in such a way that following the next-frame pointers from one frame to the next eventually forms a closed loop. When the BEP completes saving one frame, it proceeds to that frame's next-frame in the Frame Buffer and saves the new incoming frame's pixels there, thereby proceeding around the closed loop with the newest data overwriting the oldest. Frames only point to next-frames within the same region.

Note that this linked-list organization allows frames to be "skipped over" in the closed loop. Let us suppose a four-frame region, where upon initialization frame 1 points to frame 2, frame to 2 to 3, frame 3 to 4, and 4 to 1: the closed loop

then proceeds $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \hookrightarrow 1 \rightarrow$..., as expected. If frame 1 were now to point to frame 3, frame 2 would be skipped, as the closed loop would proceed $1 \rightarrow 3 \rightarrow 4 \hookrightarrow 1 \rightarrow$.... Now frame 2 will not be overwritten by incoming frames: *i.e.*, if all of the frames which point to frame n are made to point to frame n's next-frame, frame n is made inviolate. Note that the wording "*all of the frames* which point to frame n" insures that every frame's next-frame will point to a frame within the closed loop for its region. This simplifies reinsertion of inviolate frames into the closed loop: let us say we now want to allow frame n to be overwritten. We start at frame n and proceed to its next-frame, frame z — which we know is in the closed loop. We continue following the next-frame pointers until we encounter a frame a which points to frame z. We change frame a to point to frame n, and frame n is once again in the closed loop.

Normally, the BEP will be storing full frames of data, frame after frame around a region's closed loop, until the GPP has reason to change the frame-save parameters — presumably shrinking the field of interest, possibly increasing the pixel resolution, to identify a transient event. The GPP will send the BEP the new parameters, along with the region to which these new frames should be saved. The BEP then saves the incoming data according to the new parameters in this new region, frame after frame, until the GPP again has reason to change the parameters, or simply revert back to the normal full-frame mode. This is the scenario for which the BEP region system is optimized.

Two notes regarding region switches and parameter-changes: the BEP keeps track of which frame it had been writing to in each region, so that it may return to that point and "pick up where it left off" should the GPP instruct it to switch to that region. If the GPP commands a parameter-change, however, all of the data previously in that region is discarded — closed-loop frames, inviolate frames, everything. The BEP reconstructs this region with the new parameters, fitting as many frames in as it can, and by default assigning the closed loop to include all of these frames. If the modified region is the region into which incoming frames are currently being written, the presently incoming frame is discarded, and the BEP goes into wait-for-frame-sync

mode to start afresh with the next frame.

7.2 Frame Buffer I/O

It is here that most of the optimization constraints presented in the previous section are addressed. The primary issue is when to "deal" the pixels: that is, when to take the jumbled pixel order that arrives over the video channel and reorganize it to create usable line-by-line raster data. Ought it be done as soon as possible, when the pixels are being written to the Frame Buffer? This way read operations are simple, and therefore fast. Or ought it be done when data is being read from the Frame Buffer, assuming that many fewer pixels will be read than the total written? The BEP is in fact capable of doing both, as the specifics of the frame parameters will dictate which is more optimal.

7.2.1 Deal on Write vs. Deal on Read

Taking the BEP as a whole, dealing is dealing, wherever it happens; to a broad approximation, it takes the same time either way. If we then make the assumption that the total number of pixel-read operations per frame is on average less than the total number of pixels per frame, deal-on-read is the way to go.

There is a snag, however, in the case where you would like to store a *subarray* of the total camera image. While the Lasagna Box sequencer can be programmed to transmit a subset of the total number of rows in a camera, it cannot send a subset of columns. A deal-on-read system, which would deposit the pixel stream as it appears on the video channel into the Frame Buffer, could waste an inordinate amount of Frame Buffer space — imagine the region of interest is a tall, thin box: deal-on-read would save a tall and fully-wide box. With 1024-column cameras, a region of 100-pixel-wide images would be over 90+% wasted space. Or, to put it worse, the BEP would be able to save less than one-tenth as many frames as it ought to.

So now we consider deal-on-write the better option. Unfortunately, even a highly time-optimized deal-on-write system requires up to 140μs of overhead computation

per slice to orchestrate the write, plus up to 0.25μs per pixel: in normal operation, with 512x512 images coming in transmission mode 3 (all eight slices operating), we are faced with $(140 * 8) + (0.25 * 512 * 2) = 1376\mu s$ of time to deal a single line. With 2 pixels arriving every 6μs, a new line will be arriving every $\frac{6}{2} * 512 * 2 = 3072\mu s$ — so 45% of the BEP's time will be spent in the very basic task of writing normal-mode frames into the Frame Buffer.[1] And note that in this case, the deal-on-write is not saving any Frame Buffer space, because we want all of the pixels to be saved anyway.

The solution is a hybrid. When we want to save entire lines, we follow deal-on-read; when we are interested only in a portion of the columns, we deal-on-write. The BEP deal-on-write system is able to skip the undesired columns in the pixel stream *en masse*, instead of considering the utility of each pixel individually: so in our hundred-column case, at least two of the slices per camera can be skipped entirely, and only the relevant pixels in the remaining slices undergo the 0.25μs copy loop. This results in a $(140 * 4) + (0.25 * 100 * 2) = 610\mu s$ write time per line. This example was for a 1024-column image, which takes $\frac{6}{2} * 1024 * 2 = 6144\mu s$ per line to transmit — so now our deal-on-write is swallowing only 10% of the BEP's time, for a 90% improvement in Frame Buffer space utilization. Note that this hybrid scheme necessitates that the Frame Buffer read system be able to identify how a frame was saved — dealt or not — and respond to both possibilities.

This means that there are two patterns that frames in the Frame Buffer may follow. Each region is declared either deal-on-read or deal-on-write: therefore all of the frames within a region will either be a set of lines in pixel-stream order, or several sets of lines in a more raster-like order. The former is relatively simple — or rather, it is no more complex than the video stream, as described in section 4.4 and Figure 5-1. The latter is also simple, in its own way.

A deal-on-write frame is divided into an image pixel portion and a non-image pixel

[1] For comparison, dumping a full line into the Frame Buffer without dealing takes 20μs of overhead per line plus 0.2μs per word, which is two pixels in this case: $20 + (\frac{0.2}{2} * 512 * 2) = 122.4\mu s$, or 4% of the time it takes the line to arrive over the video channel. For more details on this, as well as an indication of the fallacy of our assumption that dealing takes the same time no matter where you do it, see section 8.4.

portion (underclocks and overclocks); these portions are duplicated for each camera which has data in the pixel stream. Each portion is meant to be read out in a raster fashion, pixel after pixel, line after line.

An image pixel portion may be thought of as: all of the image data for the first line of this camera; followed by all of the image data for the second line of this camera; and so on, up to the last active image row. A non-image pixel portion is divided a little further: consider it, The horizontal underclocks followed by the horizontal overclocks of the first line of slice A; then the horizontal underclocks followed by the horizontal overclocks of the first line of slice B; same for slice C; same for slice D; then the horizontal underclocks followed by the horizontal overclocks of the second line of slice A; and so on, up to the last active image row. This pile of horizontal non-image pixels is followed by all of the available data for the first vertical overclock line, then the second, up to the last. Figure 7-1 illustrates this.

Note that the two-pixel-per-word space optimization present in the video channel is not abandoned when the BEP sorts pixel data on write. When unpacking each slice in Mode 3, it grabs two pixels from that slice and re-packs them into a single 24-bit word before saving them in the Frame Buffer. Note that all relevant horizontal parameters — the number of underclocks, image pixels, and overclocks per slice, as well as the start and end columns for an image — are required to be even numbers to allow this unpacking routine always to grab two pixels at a time. However the Lasagna Box hardware cannot send an odd number of any type of pixel (image pixel, horizontal underclock, horizontal overclock, or vertical overclock), so this requirement boils down to extending the set of columns defined by the start and end columns to ensure that every slice contributes an even number of pixels to the image.

7.2.2 Frame Buffer Write Operations

As with the FEP, the BEP uses a circular *video buffer* to store data arriving over the video channel. A BEP in wait-for-frame-sync mode searches this video buffer for a frame sync pattern, indicating the start of a new frame. If this frame is not to be saved, the BEP simply reverts back into wait-for-frame-sync mode.

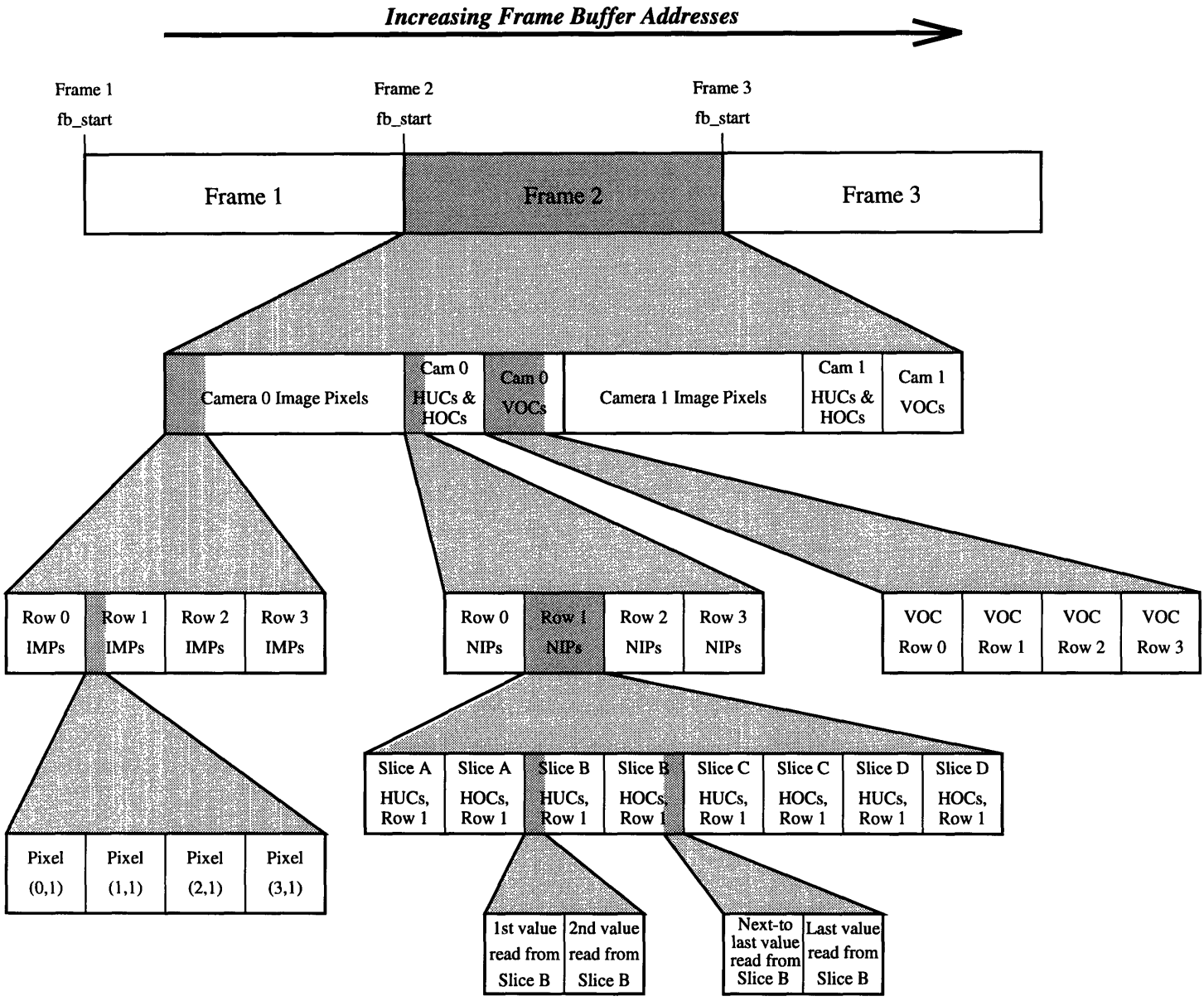


Figure 7-1: Deal-on-Write Frame Organization

If the GPP has commanded that this frame be saved, however, the BEP begins to search for row-sync patterns in the video buffer. If a full row's worth of new information has arrived, the BEP checks for a row sync at the beginning of it. If it is absent, the BEP warns the GPP of the error and abandons this frame. If the row sync is present, however, the BEP saves the row in the Frame Buffer.

If this frame is deal-on-read, the words in the video buffer corresponding to this row's pixel data — underclocks, image pixels, and overclocks alike — are copied verbatim into the Frame Buffer. A deal-on-write frame, of course, is a rather more involved procedure. Horizontal underclocks, image pixels, horizontal overclocks, and vertical overclocks, for each of the eight slices, must be considered and potentially saved for every new line of data.

The actual pixel-save routine is remarkably similar for each of these combinations. It relies on the observation made in subsection 5.2.1 and illustrated in Figure 5-1 that horizontally adjacent pixels in a slice are always four words apart in the pixel stream (assuming the slice is present in the given Lasagna Box transmission mode). The routine is given a start and an end address within the video buffer, and copies all of the pixels from the desired slice that are within this address range into the Frame Buffer. That is:

- Deal and Save Slice
    1. Calculate **offset**, the location of the slice's data within a package-of-four-words (see Table 7.2.2). If the slice is absent from the pixel stream, return.
    2. Calculate **num_pof4**, the number of packages-of-four between **start_address** and **end_address**.
    3. If the slice is B or D and the data being copied are image pixels, begin at **end_address** – 4 in the video buffer and go backward; otherwise begin at **start_address** and go forward.
    4. Add **offset** to the beginning address.
    5. Copy this word into the Frame Buffer. (Note that 16-bit data is written left-justified, as it is sent by the Lasagna Box; see section 4.4.)
    6. Go 4 words forward or backward in the video buffer; go 1 word forward in the Frame Buffer.
    7. Repeat **num_pof4** times from step 5.

Mode, Submode	A0 offset	B0 offset	C0 offset	D0 offset	A1 offset	B1 offset	C1 offset	D1 offset
0, 0	0	1	2	3	N/A	N/A	N/A	N/A
0, 1	N/A	N/A	N/A	N/A	0	1	2	3
1, 0/1	N/A	0	N/A	1	N/A	2	N/A	3
2, 0/1	0	N/A	1	N/A	2	N/A	3	N/A
3, 0/1	0(hi)	0(lo)	1(hi)	1(lo)	2(hi)	2(lo)	3(hi)	3(lo)

Table 7.1: Calculation of `offsets` Given Transmission Mode

If the pixel data is in the compressed Mode 3, `num_pof4` is halved, and step 5 is modified to read:

5a If the slice is A or C: mask the top 12 bits of this word; go four words forward or backward[2] in the video buffer; move the top 12 bits of this word into the bottom 12 bits (zeroing the top 12) and combine it with the first pixel; save this two-pixel word in the Frame Buffer.

5b If the slice is B or D: move the bottom 12 bits of this word into the top 12 bits (zeroing the bottom 12); go four words forward or backward in the video buffer; mask the bottom 12 bits of this word and combine it with the first pixel; save this two-pixel word in the Frame Buffer.

The deal-on-write routine thus considers in order each of the eight slices that may be represented in the pixel stream: A0 → B0 → C0 → D0 → A1 → B1 → C1 → D1. For each slice, the following tests are done:

1. Is this row composed of vertical overclocks? If so, and if non-image pixels are to be saved, save all of the pixels for this slice in the non-image pixel portion and return; if not, continue.
2. Are non-image pixels to be saved? If so, save the underclocks, followed by the overclocks, for this slice in the non-image pixel portion.
3. Save the segment of this slice's image pixels which is within the region of interest defined by the GPP in the image pixel portion.

Or put another way, with abbreviations as in subsection 5.2.3,

- If saving vertical overclocks:

[2]Actually, always forward; but this part of the algorithm doesn't need to know that.

start_address = **row_start**
end_address = **start_address** + [4 x ($HUCS + IMPS + HOCS$)]
NIP[3] = TRUE

- If saving horizontal underclocks:

  start_address = **row_start**
  end_address = **start_address** + (4 x $HUCS$)
  NIP = TRUE

- If saving horizontal overclocks:

  start_address = **row_start** + [4 x ($HUCS + IMPS$)]
  end_address = **start_address** + (4 x $HOCS$)
  NIP = TRUE

- If saving image pixels:

  start_address = **row_start** + **start_offset** *(precalculated value)*
  end_address = **row_start** + **end_offset** *(precalculated value)*
  NIP = FALSE

The "precalculated values" for the image pixel option is how the BEP discards unwanted columns, saving only the portion of the incoming data that is of interest to the GPP. When the BEP receives new video parameters from the GPP, it compares the range of columns governed by each slice to the range of desired columns defined by the GPP. If a slice is entirely outside the declared save region, its **start_offset** and **end_offset** are set to zero; otherwise its **start_offset** is set to $HUCS$, and its **end_offset** to $HUCS + IMPS$. If the **start_column** or **end_column** lies within a slice's domain, **start_offset** is increased or **end_offset** decreased (or both) so that the range of addresses offered to the pixel-save routine for this slice includes only the pixels that are in the desired columns.

7.2.3 Frame Buffer Read Operations

There are two general-purpose read functions at the lowest level of I/O: one handles 16-bit data, the other 12-bit data. Each of them are given a start location in the

[3] Abbreviation for "Non-Image Pixel"

Frame Buffer, the step in pixels between the values desired, and the total number of pixel values that should be read. These functions set up three adjoining windows to adjoining pages, where the middle page contains the start location; they then read the first pixel, move the specified number of steps (forward or backward) in the Frame Buffer — note that for 16-bit data, a pixel is a word, whereas a 12-bit pixel is half of a word — and read again, repeating until all the desired values have been acquired. Both routines right-justify the pixel values.

With these functions in hand, the higher-level read functions can read from a single pixel up to an entire slice for deal-on-read frames, and up to an entire line for deal-on-write frames. There are two general-purpose read utilities, **Read IMPs**[4] and **Read NIPs**. The read system divides each single-camera frame into 13 areas, each a connected region with its origin in the lower-left corner: the image area; a horizontal underclock area for each of the four slices; a horizontal overclock area for each slice; and a vertical overclock area for each slice. So (0, 0) in the image area is the lower-left corner of the camera image, *i.e.*, the first image pixel in slice A; (0, 0) in the slice B HUC area is the first horizontal underclock in slice B; (0, 0) in the slice D VOC area is the first slice D pixel from the first vertical overclock row. **Read IMPs** takes coordinates that define the corners of a box within the image area, and information on the frame to be read. **Read NIPs** takes the same type of information, plus an indication of the non-image pixel area in question. **Read NIPs** calculates the parameters needed by the low-level read utilities for the first line to be read, reads the first line, increments the start location in the Frame Buffer by one line, reads the second line, and continues until the entire box of data is read.

Because reads in the image area can cross slice boundaries, **Read IMPs** is a little more complex. It is also optimized slightly for faster operation on deal-on-read frames, since it is assumed that most frames will be of this type. **Read IMPs** first calculates what slices are spanned by the box to be read. Then it calculates the parameters needed by the low-level read utilities for the leftmost slice's contribution to the first line, and performs the read. It increments the start location in the

[4]IMage Pixels

Frame Buffer by one line and reads again, placing the data appropriately in the return array. It continues up the box, reading all the information contributed by this leftmost slice. It then begins again with the second-to-leftmost slice, reading all the data from this slice that contributes to the entire box; and so on until it has read from all the contributing slices. This line-by-line, slice-by-slice order takes advantage of the observation that it is time-consuming to calculate all of the parameters needed by the low-level routines (taking into account transmission mode, deal-on-write vs. deal-on-read, *et cetera*), but once these parameters are calculated, it is easy to use them repeatedly for successive lines.

Unfortunately, this general-purpose read system is not sufficiently fast to perform all of the reads the BEP ought to perform per frame. The most common read is a 3-by-3-pixel square in the image area, which for a normal-operation frame (Mode 3, deal-on-read) takes either 94.5μs if the square is contained in a single slice, or 124.7μs if the square spans two slices. A normal-operation frame, 512x512 pixels for each of two cameras with two pixels arriving every 6μs, takes $\frac{6}{2} * 512 * 512 * 2 = 1,572,864\mu$s = 1.57s: so the BEP would be able to perform at most $\frac{1572864}{94.5}$ = 16,644 3x3 reads per frame. This may appear to be a large number, but recall that the BEP needs to have time to perform calculations on the data it reads, as well as take care of its myriad other functions — writing to the Frame Buffer and communication with the GPP and the like.

In keeping with the principle that program memory may be sacrificed for speed, another read routine was created: a fast single-slice 3x3 read for normal-operation frames. This function does not have to check for transmission mode, or deal-on-write vs. deal-on-read, or how many slices it spans, or any of the other concerns that slow down the general-purpose system: its calculations are thus vastly simplified, and it performs its task in no more than 19.8μs.

7.3 Command & Response List

The BEP responds to the following commands from the GPP:

PROCESS_VIDEO

The BEP is instructed to save the next frame to arrive along the video channel. A parameter indicates whether the BEP should save only this next frame, or to begin saving frame after frame until told to stop.

DONT_PROCESS_VIDEO

The BEP is to stop saving frames until told to start again. If a frame is currently being processed, the BEP is to finish doing so, but then ignore frames that follow it. There are no parameters.

SWITCH_REGION

The BEP is told to switch into a new Frame Buffer region.

REALLOCATE_FB

The GPP defines the starting Frame Buffer address for each region. Each region ends where the next one starts; the GPP sends an explicit ending address for the last region.

CCD_CONFIGURATION

The BEP is informed of: the transmission mode and cameras used (submode) by the Lasagna Box; the number of horizontal underclocks, active image columns, and horizontal overclocks per slice; the number of active image rows and vertical overclocks per frame; the on-camera row numbers of the first and last rows to arrive over the video channel; the range of columns the BEP should save for each camera; whether to deal on write or deal on read; whether or not to save non-image pixels in deal-on-write mode; and the region to which these parameters apply. The BEP may

be instructed to abort processing the current frame in favor of these new parameters; if the region to be changed is the region currently being saved to, this abort happens automatically.

SAVE_FRAME

The GPP wishes a frame to be marked inviolate. The frame ID is sent.

DUMP_FRAME_RAW

The GPP wishes to receive an entire frame from the BEP, exactly as it appears in the Frame Buffer. The frame ID is sent.

DUMP_FRAME_DEAL

The GPP wishes to receive a frame's image data from the BEP, in raster format. The frame ID and camera number are sent.

DUMP_IMP_ARRAY

The GPP wishes to receive a box-shaped portion of a frame's image data from the BEP, in raster format. The frame ID, camera number, and coordinates of the box corners, are sent.

DUMP_TBT_ARRAY

The GPP wishes to receive a 3-by-3-pixel array from a frame's image data. The frame ID, camera number, and coordinates of the lower-left corner of the 3x3 region, are sent.

The BEP generates the following messages for the GPP:

DSP_FRAME_STARTED

The BEP reports that a frame sync has been found. The frame ID is included.

Message goes to the most recent sender of PROCESS_VIDEO.

DSP_FRAME_ENDED

The BEP reports that all of the rows of this frame, including vertical overclocks, have been received. The frame ID is included. Message goes to the most recent sender of PROCESS_VIDEO.

DSP_FRAME_ERROR

The BEP reports that there has been some error (such as an absent row sync) while processing a frame. The frame ID and a number indicating the type of error are included. Message goes to the most recent sender of PROCESS_VIDEO.

BEP_UNKNOWN_ID

The BEP has received a request concerning a frame which is not present in the Frame Buffer. Message goes to the sender of the request.

BEP_RAW_FRAME

One of a set of messages containing frame data exactly as it appears in the Frame Buffer. The frame ID is included, as well as a set of flags indicating whether this message is the first, last, or in the middle of the set. Message goes to the sender of the DUMP_FRAME_RAW request.

BEP_DEALT_IMPS

One of a set of messages containing image data from a frame, in raster format. The frame ID; camera number; a set of flags indicating whether this message is the first, last, or in the middle of the set; and the coordinates of the corners defining the box of data requested are included. Message goes to the sender of the DUMP_IMP_ARRAY or DUMP_FRAME_DEAL request.

BEP_TBT

A 3-by-3-pixel array of image data. The frame ID, camera number, and the coordinates of the lower-left corner of the array are included. Message goes to the sender of the DUMP_TBT_ARRAY request.

DSP_EXCEPTION

The BEP warns of a hardware exception, such as a video channel receive overrun. For errors relating to the video channel, messages go to the most recent sender of PROCESS_VIDEO.

Chapter 8

Back-End Processor Implementation

Memory
All alone in the Moonlight
I can smile at the old days
I was beautiful then

— *T.S. Eliot & A.L. Webber*

The BEP C code is compiled to use the 56001's Y: data space, not to use register R5, and to employ parallel move operations when possible. BEP Assembly code uses parallel move operations heavily. Although efforts have been made in this chapter to point out when the BEP uses some subtle aspect of the compiler or the processor, readers unfamiliar with the Motorola 56001 or its GNU C compiler are referred to the documents on these cited in the Bibilography.[5, 6]

This chapter also uses terms such as "video buffer" and "package-of-four" defined in the previous chapter. Readers unfamiliar with the algorithms that this code implements are referred there.

Variables related to set beginnings are inclusive, and those related to set endings, exclusive. That is, **start_col** = 5 *and* **end_col** = 10 *defines columns 5, 6, 7, 8, and 9 as included in a set. This means that the number of members in the set is always the difference between the end and the beginning values.*

Numeric indices — coordinates, array references, bit locations, et cetera *— start with 0. Cardinal references start with "first". Hence the "first bit" of a 24-bit word is "bit 0", and the most significant "twenty-fourth" bit is "bit 23". Confusing as this appears, I believe it is a reasonably natural pattern, and consistent with related literature.*

8.1 Functional Breakdown

The BEP software is composed of the following modules:

- **BEPmain.c**: Performs initializations (with **BEPsetvars.c**), engages mainloop (see section 4.3). Also contains simulation- and memory-related test routines.
- **BEPsetvars.c**: Contains initialization routines.
- **BEPcom.c**: Handles high-level (*i.e.* above Assembly-level) communications with the GPP.
- **BEPvidin.c**: Sets up copying from video buffer into Frame Buffer, including deal-on-write.
- **BEPvidout.c**: Mid-level pixel-read routines, used by high-level procedures when they wish to read information from the Frame Buffer.
- **BEPframe.c**: Functions related to frames and Frame Buffer regions: finding a frame in the Frame Buffer by its ID; marking a frame inviolate; setting up a region; and dumping frame data upon request.
- **BEPasmsupport.asm**: Contains various small support routines which are best implemented in Assembly language, such as serial communications hardware control, interrupt vector initialization, *et cetera.*

- **BEPfb.asm**: Lowest-level Frame Buffer I/O routines. Highly time-optimized.
- **BEPwindow.asm**: Fast routine for making a Frame Buffer window point to a specified page; similar routines for making two or three adjacent windows point to two or three adjacent pages.
- **BEPintr.asm**: Long interrupt handlers, currently only for video receive exception.
- **crtrzp.asm**: Startup file which controls the DSP's behavior on reset. Initializes the stack pointer, DMA communications with the host GPP, and the like; then passes off to **BEPmain.c**.
- **BEPincl.h**: Contains constants, type and structure definitions, and external function declarations used by the above C modules. #include's other infrastructure .h files, such as **BEPwindow.h** and those related IPP communication.
- **BEPwindow.h**: Constants related to Frame Buffer windows.
- **BEPglobals.h**: Global variable declarations. #include'd once by **BEPmain.c**.
- **BEPequ.asm**: Constant definitions and variable allocations for the above Assembly modules.
- **machspec.asm**: Constant definitions for Assembly modules, specific to HETE hardware (as opposed to the NeXT development system).
- **ioequ.asm**: Constant definitions for communications hardware.

BEPincl.h defines several structures which contain vital information on the state of the BEP, in particular on the frames stored in the Frame Buffer and the status of incoming video data. The former is governed by an **all_frames_struct**, the latter by a **vidbuf_param_struct**[1]. The **all_frames_struct** contains the following arrays:

[1] "vidbuf" is an abbreviation for the VIDeo BUFfer. The values in this structure relate either directly to the video buffer, or to the processing of the data in it.

- **region_param_struct rp**, one for each region. The parameters defining the characteristics of the frames within a given region.

- **frame_specs_struct fs**, one for each frame that can be kept track of. Small structure to keep track of only the parameters which change from frame to frame.

- **fb_ptr region_start**, one for each region, plus one. Defines the floor address of the region; the next region's floor is the present region's ceiling, so the extra **region_start** is the ceiling of the last region.

- **int current_framenum**, one for each region. Keeps track of the last frame number written to in each region, to facilitate region-switches.

A **region_param_struct** is composed of the following:

- **ccd**, a structure representing the Lasagna Box parameters

- **deal**, a flag indicating whether frames in this region are deal-on-write

- **save_nips**, a flag for deal-on-write frames indicating whether non-image pixels are saved

- **start_row** and **end_row**, one for each camera, indicating the on-camera number of the first and last image rows which arrive over the video channel

- **start_col** and **end_col**, one for each camera, indicating the first and last image columns which are saved in deal-on-write mode

- **imp_start_offset**, **nip_start_offset**, and **voc_start_offset**, one for each camera, indicating the offset from the base of the frame where the image pixels, non-image pixels, and vertical overclocks are stored in deal-on-write frames (see Figure 7-1)

- **linelen**, one for each camera, indicating the number of Frame Buffer words comprising a single line of (image) data — **end_col–start_col** for deal-on-write frames (half this number for Mode 3), 4*(**num_hocs**+**num_cols**+**num_hocs**) for deal-on-read frames (which use only **linelen[0]**)

- **size**, the total number of Frame Buffer words used to store a single frame

ccd, a **ccd_param_struct**, contains the transmission mode and cameras used; the number of horizontal underclocks, image pixels, and horizontal overclocks per slice; the total number of columns per slice; the number of rows containing image pixels, and vertical overclocks; and the total number of rows per image. A **frame_specs_struct** contains a pointer to the **region_param_struct** that represents this frame's parameters, the frame ID, the frame's base Frame Buffer address, and the number of the next-frame in the region's closed loop. In its present implementation, 1000 frames are reserved for each of three regions. The array of **frame_specs_structs** in the **all_frames_struct** thus has 3000 elements: indices 0–999 are for region 0, 1000-1999 are for region 1, and 2000-2999 are for region 2. See chapter 9 for other possible allocations of frames to regions.

The **vidbuf_param_struct** holds the following information on the incoming video data being placed in the video buffer:

- **region**: The number of the region where the frame is to be saved.
- **save_frame**: Flag telling the BEP whether or not this frame ought to be saved into the Frame Buffer.
- **single_frame**: Flag telling the BEP whether it is in single- or multi-frame mode, *i.e.*, if only this frame should be saved, or if frames should be continually saved until further notice
- **one_line_delta**: The number of words a single row occupies in the pixel stream. If you are standing on a word in the pixel stream which corresponds to coordinate (c, r), going **one_line_delta** words downstream (toward earlier-arrived data) puts you on $(c, r-1)$, and going **one_line_delta** words upstream (toward later-to-arrive data) puts you on $(c, r+1)$. See subsection 5.2.1 for a discussion of this phenomenon.
- **left_off**: Pointer into the video buffer indicating the next word to be read. (The "receiver runner" from section 4.3's description of circular buffers.)

- **lines_rcvd**: The number of lines confirmed to have been received into the video buffer for this frame. When a full row's worth of data (**one_line_delta** words) has been read into the video buffer since the end of the last row, the BEP checks for a row sync at the beginning of these new data; if found, **lines_rcvd** is incremented by one.

- **waiting_for_frame_sync**: Flag indicating whether the BEP is currently searching the incoming video data for a frame sync, or whether it is processing a frame. When all of the rows in a frame have been received (including vertical overclocks), **waiting_for_frame_sync** is set to search for the next frame's sync. Note that setting **waiting_for_frame_sync** while the current frame's rows are still being received effectively aborts processing of this frame.

- **lobound_delta**[*slice*], one for each of the eight slices — the **start_offsets** from subsection 7.2.2.

- **hibound_delta**[*slice*], one entry for each of the eight slices — the **end_offsets** from subsection 7.2.2.

- **imp_left_off**[*camera*], one entry for each of the two cameras. Marks where writing into the Frame Buffer left off at the last row, *i.e.*, the word immediately past the last word of the (image) line most recently written. Deal-on-read frames use **imp_left_off[0]** to keep track of its location in the Frame Buffer.

- **nip_left_off**[*camera*], one entry for each of the two cameras. Marks where writing into the non-image pixel portion of a deal-on-write frame left off at the last row, *i.e.*, the word immediately past the last non-image pixel written.

Two variables of type **vidbuf_param_struct** are kept: the **vidbuf_params** of the current frame (where a frame lasts from its frame sync to the next frame's sync), and the **next_vidbuf_params** which become current upon the arrival of the next frame sync. When the GPP sends a command to change any of the parameters in this structure, the BEP modifies **next_vidbuf_params**; thus changes in the region to be written to, and

the character of frame saving, result in modification of **region**, and **save_frame** and **single_frame**, respectively *within* **next_vidbuf_params**. In addition, within **next_vidbuf_params**, **lines_rcvd** and **waiting_for_frame_sync** are zero. Then when a new frame sync comes along, the **left_off** pointer in **vidbuf_params** is copied to **next_vidbuf_params**, and then **next_vidbuf_params** is simply transferred *en masse* into **vidbuf_params**, reflecting that these values are now current. If the GPP has changed any of the parameters, these new parameters now take effect; otherwise, the result is simply the resetting of **lines_rcvd** and **waiting_for_frame_sync**. The *xx*_**left_off** pointers are reset at this parameter-switching as well.

This bears repeating: **vidbuf_params** contains the parameters applicable to the *current* frame; **next_vidbuf_params** contains the parameters applicable to the *next* frame to come along the video channel, parameters which go into effect upon the next frame sync.

When the GPP wishes to reconfigure a region — most likely because it has reconfigured the Lasagna Box parameters — it sends the parameters needed for the **region_param_struct**, along with the number of the region to be changed. The BEP then reconfigures this region, effectively erasing any frames which were previously stored there by making them unreadable (the parameters for the region no longer match the parameters for the frames previously stored there). The frame IDs in the **frame_specs_structs** within this region are zeroed to reflect this. If the region modified is the region currently being written to, saving of the current frame is aborted, and the BEP waits until the next frame sync to begin saving frames under the new parameters.

8.1.1 Memory & Resource Allocation

The DSP's three memory spaces — X:, Y:, and P:, each spanning 32 kilowords — are divided by the BEP into definite regions, shown in Figure 8-1. P: space has reset and interrupt vectors from P:$0000 to P:$003F, short-addressable space from P:$0040 to P:$01FF, and long-addressable space from P:$0200 to P:$7FFF. In addition, P:$0000→$0AAA is initially set to point to boot ROM, so the loader cannot

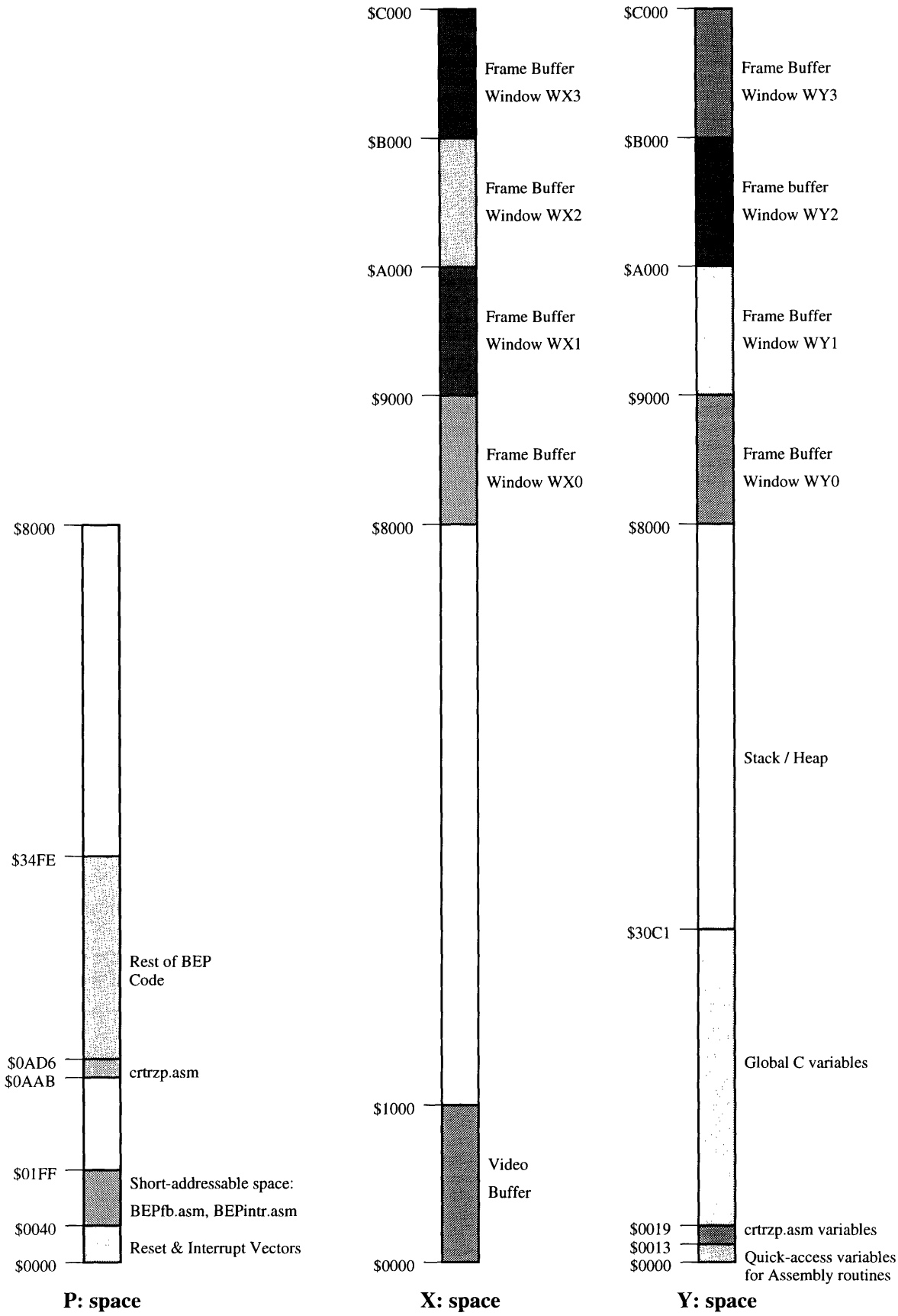


Figure 8-1: BEP RAM Memory Map

access this area.[2] So the majority of the code lies in P:$0AAB→$7FFF, with the **crtrzp.asm** reset code starting at P:$0AAB. Since short-addressable space allows for faster jump instructions, time-critical code — specifically, that in **BEPintr.asm** and **BEPfb.asm** — is placed in this space. Because of the ROM boot area coverage, however, the code for these routines, as well as the interrupt vectors, must be explicitly written to this area by the BEP itself.

X: space currently consists only of IPP variable storage and the video buffer. Since the video buffer can be as small as ~two maximum-size lines — and the longest expected line is 2K, when all 1024 columns of both cameras are read out in Modes 1 or 2 — and only a few words are allocated for IPP, the vast majority of X: space is available for future expansion.

Y: space is heavily divided. The first bunch of words are allocated in **BEPequ.asm** to be short-addressable variables for Assembly routines; these variables must end up in Y:$0000→$00FF to be short-addressable, or preferably →$003F for short-addressable internal RAM (faster access times than short-addressable external RAM when external P: RAM is being used). After this comes compiler variables (memory limit, stack safety margin, *et cetera*), allocated in **crtrzp.asm**, and then C global variables. The declaration of `all_frames_struct fb` in **BEPglobals.h** takes up 12,091 words in global space, 12,000 of which are the 3000 4-word `frame_specs_structs`. There is still almost 20K of usable stack / heap space above this, and the stack pointer R6 is initialized to start immediately above global space.

Since the C compiler is not allowed to use register R5, this register is allocated to the video channel receive interrupt routine. R5 is declared the video buffer write pointer (the "sender runner" from section 4.3's description of circular buffers). Hence the SSI receive interrupt (indicating a new word received over the video channel) is a short interrupt which simply reads `movep x:<<m_rx,x:(R5)+`, that is, "Take the word just received, put in X: space at the address indicated by R5, and then increment R5 in anticipation of the next word". The modulus register M5 is set once to keep R5 within the video buffer; after that, the 56001 hardware takes care of the behavior of

[2] Future versions of the spacecraft hardware may change this.

R5 automatically.

The Frame Buffer is divided into regions of adjustable size, as described in subsection 7.1.2. Each region is in turn divided into as many frames as will fit into it, given the characteristics of the region's frames (in this case, the **size** parameter in this region's **region_param_struct**), up to 1000 frames per region. By adjusting the first region's floor, or the last region's ceiling, space may be reserved at the beginning or the end of the 4 Megawords of Frame Buffer memory for uses other than frame storage.

8.2 Initialization and Normal Function

Upon reset, the BEP starts running at address P:$0000. A jump instruction here pushes program flow to P:$0AAB, above the ROM boot area, governed by **crtrzp.asm**. Hardware initialization — resetting the DSP stack pointer (SP), C code stack pointer (R6), peripheral hardware — is performed, and program flow is then switched to the **main()** routine in **BEPmain.c**.

main() declares **vidbuf_params** and **next_vidbuf_params**; they are generally passed by reference (*i.e.* pointers to these structures are used by most other routines), not only to allow for modification, but also to save calling time. The former is generally abbreviated **vbp**, and the latter **vbpnew**. Note that since these two structures are declared in **main()**, this space will not been overwritten. Control passes to **onetime_init()** in **BEPsetvars.c**, which performs the following initializations:

1. Initialize P:$40–AAA
2. Register the BEP with the Constellation Name Server, and get default destination addresses
3. Set up variables for buddy-check (the regular reports to the GPP stating the FEP is still alive)
4. Make R5 point to the beginning of the video buffer, and set M5 to keep R5 within the video buffer

5. Initialize serial communications variables (SSI control registers)

6. Initialize serial communications interrupt vectors: SSI receive and exception (video channel)

7. Turn the serial communications interrupts off or on, according to whether we are or are not in simulation mode, respectively

8. Set up the regions to reflect sensible default values. Make sure the **region_starts** are consecutive, so the ceiling of one is the floor of the next; divide the Frame Buffer equally among the regions by default. (For more information on setting up a region, see subsection 8.2.2.)

9. Set up **vbpnew** to sensible default values: in particular, ensure **lines_rcvd** and **waiting_for_frame_sync** are zero. Set **vbp->left_off** to the beginning of the video buffer, and "abort the incoming stream" (set **vbp->waiting_for_frame_sync**) to set the BEP looking for the next frame sync to come over the video channel, at which time these default parameters in **vbpnew** will become current.

Back in **main()**, control passes into the mainloop. The mainloop performs the following steps in repeated succession forever:

1. **check_buddy_time()**: Tell the GPP the BEP is alive

2. **try_get_message()**: See if an IPP message has arrived, and if so, process it

3. **simulate_*xx*()**: If the BEP is in simulation mode, fabricate pixel data and place it in the video buffer as though it arrived over the video channel

4. **try_copy_line()**: Try to sync with the data which arrived during the previous mainloop; copy a single row of data into the Frame Buffer if possible

5. **try_relay_errors()**: If any exceptions have occurred during the previous mainloop (such as an overrun in the serial communications system), warn the GPP

Note that for testing purposes, the BEP can be instructed to skip steps 3 through 5, effectively freezing much of the state of the BEP while letting it continue to report its live status and receive commands. The user can then instruct the BEP to execute a single mainloop (*i.e.* run through steps 3–5 once), and then observe how the state of the BEP has changed.

8.2.1 Serial Communication Interrupts

As outlined in section 4.3, the BEP works under a "interrupt-plus-main-code" system; thus, operating concurrently with the mainloop described above, is a set of interrupt-driven processes. Aside from the interrupts used by the underlying IPP system (host interface, direct memory access, and the like), the BEP has only two other hardware interrupts: one regular and one exceptional interrupt for the video channel (SSI receive). For information on the electronic signals involved in this channel, see table 4.1.

Two interrupt vectors are initialized by **init_intr_vectors()** in **BEPasmsupport.asm**. As described above, the SSI receive interrupt is the faster, as it is assumed to be the one called most often (once every $6\mu s$). It reads simply `movep x:<<m_rx,x:(R5)+`, that is, "Take the word just received, put in X: space at the address indicated by R5, and then increment R5 in anticipation of the next word".

The exception handler for this channel simply reads the appropriate status register into a variable, clearing the 56001's exception flag, and then performs the `movep x:<<m_rx,x:(R5)+` instruction to try to re-establish normal operation. The main-code routine **try_relay_errors()** checks if any of this or any other exception status variables are non-zero, and if so, sends a warning message to the GPP stating that the error occurred and including the status register value. It then zeros the exception status variable to make sure only one message is sent per error.

8.2.2 Region Configuration

When a region is reconfigured — most likely as a result of a command from the GPP — a great many computations take place, not only to reallocate the use of

this portion of the Frame Buffer, but to precalculate values that will be of use in time-critical portions of the code. Thus when the BEP receives a message such as CCD_CONFIGURATION, it collects the fundamental parameters such as transmission mode and the number of active columns per slice and the like, and then calls **fill_out_params()** in **FEPframe.c**.

The first thing **fill_out_params()** does is checks if the the incoming video data is in compressed format or not — that is, whether or not the Lasagna Box is using transmission mode 3. If it is in Mode 3, and these frames will be deal-on-write, then the BEP ensures that the write routine will always be able to grab two pixels at a time by (1) using the fact that the Lasagna Box hardware forces num_hucs, num_cols, and num_hocs to be even numbers, (2) decrementing start_col if it is odd, and (3) incrementing end_col if it is odd. The BEP has now guaranteed that there will always be an even number of pixels read from any slice into the Frame Buffer.

fill_out_params() then calculates size parameters — linelen, size, and the *xx*_start_offsets — and here we see how deal-on-write frames are more complex than deal-on-read. Deal-on-read frames are simple: since rows are copied verbatim from the video buffer into the Frame Buffer, the length of a line in words is just four times the number of columns per slice, no matter the transmission mode or anything else; the *xx*_start_offsets are all zero, again, since there is no separation of NIPs from IMPs; and the size of a frame is just the total number of rows times linelen.

A deal-on-write frame, however, has to worry about distinguishing NIPs from IMPs, and seeing what pixels the GPP wants saved versus thrown away. It uses the size parameter as a running count of how many words it has used so far. For each camera, the routine first checks how many actual rows will be saved[3], and then checks to make sure this camera's data is okay, *i.e.*, whether it is to be saved at all. It then computes linelen, the number of image columns stored per line, which is simply the number of columns the GPP asked to be stored, end_col - start_col, divided by compression, the number of pixels per word. **fill_out_params()** then decides whether or not it should keep this value, according to whether or not this camera is okay. It

[3]Presently assumed to be equal to the number of rows arriving

then allocates space for this camera's image pixels by setting **imp_start_offset** to **size**, the running count of how many words have been allocated so far. If this camera's data is not **okay**, **imp_start_offset** is reset to zero. **size** is incremented by the number of words taken up by the image pixels for this camera. In a similar fashion, if the GPP has instructed that non-image pixels are to be saved, space is allocated for the horizontal NIPs and the vertical overclocks by setting their ***xx*_start_offsets** to **size** and then incrementing **size** by the number of words these pixels will take up.

Once these values have been computed, the region itself must be configured — that is to say, the **frame_specs_structs** that are associated with this region must have their contents reset. **set_up_region()** takes care of this by deciding how many frames can fit into this region (up to 1000), and what **frame_specs_struct** indices are involved. It then sets up the first frame's **frame_specs_struct**: **rp** points to this region's **region_param_struct**, **frame_start** is at the region's floor, the **next_frame** is the second frame in the region, and the ID is reset to reflect that this frame has not been filled with video data yet. These values are copied to all of the other frames in the region, changing only the **frame_start** and **next_frame** parameters. The last frame in the region has its **next_frame** set to point to the first frame in the region; and the rest of the 1000 **frame_specs_structs** allocated for this region are set to be identical to this last entry. The region now has a definite single-closed-loop linked-list character. The **current_framenum** for this region is set to the first frame in the region, and the region is ready to receive frame data. Note again that all video data previously stored in this region is effectively erased when the region is reconfigured.

When the GPP orders the BEP to switch to a new region, several parameters in the **next_vidbuf_params** need to be modified. The **region** number itself is changed first, and then the **set_vbp_new_config()** routine in **FEPsetvars.c** takes care of the rest. For each of the eight slices, the **lobound_delta** and **hibound_deltas** are set to reflect which image pixels in a given slice are to be saved. If all of a slice is to be saved, **lobound_delta** is set to skip the packages-of-four containing horizontal underclocks, and **hibound_delta** set to stop copying before the packages-of-four containing horizontal overclocks. If only a portion of the slice is to be saved, however, these values need to be modified, as

If you're in Slice...	And start_col is in this slice...	And end_col is in this slice...
A or C	Increase lobound_delta	Decrease hibound_delta
B or D	Decrease hibound_delta	Increase lobound_delta

Table 8.1: Modification of *xx*bound_deltas for Deal-on-Write Frames

indicated in Table 8.1. Note the discrepancies with respect to slice are due to Slices B and D being read out "backwards", as described in section 4.4; Figures 4-1 and 5-1 may be useful in determining what the *xx*bound_deltas are interested in.

When the sizes of regions are modified by a REALLOCATE_FB command from the GPP, the set_up_region() function is called for each region. This means that *all of the frames previously stored in the Frame Buffer are rendered inaccessible.*

8.2.3 Assembly Language Techniques

There are quite a number of Assembly routines to write to and read from the Frame Buffer. However the tricks they use to save time, and the pratfalls they must watch out for, are very similar:

- The 56001 assumes 24-bit data are in 2's-complement format, so calculations and more importantly comparisons are conducted accordingly. To ensure sensible calculations with unsigned data (23 bits or less) that occupy the most significant bit, the data is right-shifted by one bit, forcing the MSB to zero.

- Fast multiple-bit shifts may be accomplished using the 56001's MULtiply (and Multiply-ACcumulate) instructions. Note that, as above, the MSB must be set to zero by shifting or masking for this operation to work consistently with unsigned data. This allows data in the top 12 or 16 bits of a 24-bit word to be placed in the bottom bits of A1 or B1, or conversely the bottom 12 bits to be placed in the top bits of A0 or B0, in a single instruction.

- For parallel moves involving a variable used in the primary instruction, such as clr A A1,y:(R6)+ or cmp X0,A x:(R3)+,A1, *sources* are taken *before* the primary

instruction, while *destinations* are modified *after* the primary instruction. Hence in these two examples, A1 would be saved to y:(R6) *before* A is cleared, but A1 is loaded with the value from x:(R3) *after* it is compared to X0.

- Since DO loops cannot end in a jump instruction, loops that logically ought to end with one are "rotated" so that the logically first instructions in the loop is "pulled around" to be the last instructions encountered. This means that these "pulled around" instructions need to be repeated before entrance to the loop for the first iteration, and that when the final in-loop instructions are encountered for the last time, they must have no ultimate effect.

8.3 Frame Data Storage

Processing of data in the video buffer has two components: synchronization, and pixel storage. Both functions are performed by **BEPvidin.c**, with the core copy routines in **BEPfb.asm**. Synchronization is handled in the same fashion as described in subsection 6.4.1.

try_copy_line() is the center of video storage, being the mainloop routine which calls the rest of the synchronization and pixel-save routines. This procedure begins by checking if the BEP is in waiting-for-frame-sync mode. If so, it calls **sync_frame()** to see if the frame sync has arrived in the video buffer yet; if the new frame is synced, the next_vidbuf_params are made current, and the ID encoded in the frame sync is recorded; if this frame is to be procesed (*i.e.* saved), the GPP is informed of the frame sync's arrival.

switch_params() takes care of making next_vidbuf_params current. It first notes whether the frame just past was saved, and if so, the current_framenum is set to this frame's next_frame. vbp->left_off is preserved, and then all the values in vidbuf_params are set to the corresponding values in next_vidbuf_params. The new parameters are now current, and next_vidbuf_params now represent the *next* frame's parameters. **switch_params()** now sets the *xx*_left_off parameters in vidbuf_params, and then checks for the kind of processing this frame entails: if this is a one-frame-

only processing situation, then clear the *next* frame's **save_frame** flag; if the present frame is not to be saved, revert immediately back into waiting-for-frame-sync mode.

If the BEP is not in waiting-for-frame sync mode, it is in save-lines mode. **try_copy_line()** checks if there is a full row's worth of new words in the video buffer. If so, it checks for a row sync at the beginning of this new data, warning the GPP and aborting processing if the sync is absent. Assuming the sync is found, however, **vbp->lines_rcvd** is incremented, and the line is saved into the Frame Buffer by **copy_line()**. Once the line is saved, **vbp->left_off** is updated to point just past the end of this line. If all of the rows have been received and saved, including vertical overclocks, the BEP informs the GPP that it has completed processing this frame, and returns to waiting-for-frame-sync mode.

Note that while processing a frame, **vbp->left_off** is in one of two positions in **try_copy_line()**: it is either immediately before the expected position of the next row's row sync, including preceding blank words (*i.e.* one word beyond the previous row's last pixel word, or one word beyond the frame sync); or immediately after the row sync just located (*i.e.* at the first word of the first package-of-four for this row).

BEPvidin.c uses the same basic utilities as **FEPvid.c** to move around the video buffer — **vb_add()**, **vb_distance()**, **read_vb()**, and **have_words()** — although the first two are implemented as functions in this release. They may be replaced with the macros found in **FEPvid.c** for a substantial improvement in speed, as long as the "relevant bits" orchestration of the video buffer is followed. See section 6.4 for further information.

The **copy_line()** procedure must first decide whether this frame is deal-on-write or not. If not, it simply sets up two consecutive Frame Buffer windows from X:$8000→$A000 (windows WX0 and WX1) such that the first window includes the address where writing in this frame left off after the last row was written. The Assembly routine **copy_words()** is then called to copy the entire line verbatim from the video buffer into the Frame Buffer.

A deal-on-write frame, of course, takes quite a bit more work. The algorithm governing this operation is offered in subsection 7.2.2. **copy_line()** checks each camera

to see if its data is to be saved, and if so, it cycles through each of the four slices in that camera. It then checks if non-image pixels are to be saved: if so, and if the line in question is composed of vertical overclocks, save the whole slice in the non-image pixel area; or, if the line in question is below the vertical overclocks, save the horizontal non-image pixels. Then, save the desired image pixels from this slice, where "desired" is defined as the pixels in this slice contained in the video buffer between **vbp->left_off** + **vbp->lobound_delta** and **vbp->left_off** + **vbp->hibound_delta**.

copy_line() passes off to **deal_into_fb()** to set up the variables required by the Assembly copy routines. **deal_into_fb()** performs steps 1 through 4 in the **Deal and Save Slice** algorithm from subsection 7.2.2, and updates the *xx*_**left_off** values in **vidbuf_params** to reflect the number of words written into the Frame Buffer. **deal_into_fb()** in turn passes off to **copy_words()** for 16-bit data, or **copy_hi_pixels()** or **copy_lo_pixels()** for 12-bit compressed data; these are the lowest-level pixel-write routines, found in **BEPfb.asm**.

copy_words() is relatively simple. It relies on the calling function to have set up two X: Frame Buffer windows in consecutive order, and to pass as arguments the starting point in the video buffer (placed in R2) and the starting point in the lower X: Frame Buffer window (placed in R1). Another parameter tells **copy_words** the "skip" distance, *i.e.*, how many words and in which direction R2 should move in order to point to the "next" word to be copied. This number will be –4 for dealing image pixels from slices B or D, +4 for dealing all other pixels, or +1 for a straight verbatim copy. The final parameter indicates how many words are to be written into the Frame Buffer. With these parameters in hand, copying into the Frame Buffer takes a two-line loop: the first line reads the value from the video buffer into A1, and moves R2 by the skip distance N2; the second line writes A1 into the Frame Buffer window and increments R1.

copy_hi_pixels() and **copy_lo_pixels()** take the same arguments, but dispatch with two pixels at a time, grabbing two consecutive same-slice pixels and combining them to form a single 24-bit word before saving to the Frame Buffer. Note that the final argument is *words saved to the Frame Buffer*, which in this case would be half

the number of *pixels* saved. Again, reference to the **Deal and Save Slice** algorithm in subsection 7.2.2 is useful, especially steps 5a (for **copy_hi_pixels()**) and 5b (for **copy_lo_pixels()**).

8.4 Frame Data Retrieval

Subsection 7.2.3 describes in relative detail the methods followed by the Frame Buffer read functions in **BEPvidout.c** and **BEPfb.asm. read_imp()** takes the start and end column and row defining the box of image pixels desired, the **frame_specs_struct** and the camera number indicating where this box should be taken from, and a pointer to an array into which the pixels (read out left-to-right, bottom-to-top, one right-justified pixel per word) should be written. It determines which slices are included in this box, and the image columns contained in these slices. It then calculates **bls**, the frame buffer address of the beginning of the image pixels in the bottom line in the desired box (**srow**).

At this point **read_imp()** enters a loop which cycles through the slices included in the box, counting by columns. That is, it starts with the first image column of the first slice involved, and increments by **num_cols** until it reaches the first image column of the last slice involved. It determines the columns within this slice which are included in the box, and dispatches to **get_imp_offset()** to get the rest of the parameters needed by the **FEPfb.asm** read utilities: the **offset** (in words) from **bls** where reading should start, a flag **startlo** indicating whether the read should start with the low pixel in this word, and the **step** (in pixels) from this pixel to the next one to be read. **get_imp_offset()** calculates these values using compressed calculations based on the transmission mode of the frame, and whether it is deal-on-write or deal-on-read. These calculations must take the following concerns into account:

- If there is no data saved for this slice, the read is invalid.
- If the frame is deal-on-write, **step** is +1; if the frame is deal-on-read, **step** is +4 (+8 if Mode 3) for slices A and C, or –4 (–8 if Mode 3) for slices B and D.

- If Mode 3 and deal-on-write, **offset** is half of the number of pixels between the beginning of the slice and **scol**, and **startlo** is 1 if that number of pixels is odd.

- If Mode 3 and deal-on-read, **startlo** is 1 for slices B and D.

It is notable that the actual calculations for **offset** are sufficiently compressed for both deal-on-read and deal-on-write frames, that there is not a significant time advantage to reading from the latter. The largest difference is in Mode 3, where the **offset**, **step**, and **startlo** calculations take 15 instructions for deal-on-write, and 70 for deal-on-read. Since the remainder of the **get_imp_offset()** routine, common to both types of frames, takes 122 instructions, this is an improvement of only $100 - \frac{122+15}{122+70} = 29\%$ — compared to a $\frac{3072\mu s}{122.4\mu s} = 2510\%$ loss during the write stage (using the numbers calculated in subsection 7.2.1).

With these values, **read_imp()** reads all of the pixels that this slice contributes to the desired box, using **read_fb12()** for Mode 3, or **read_fb16()** for other Modes. These two utilities from **BEPfb.asm** take the Frame Buffer address **fb_start** at which point the read should start, the flag **startlo**, the pixel-to-pixel **step**, the number of pixels to be read **dist**, and a pointer after which the right-justified pixels should be written consecutively. Note that **read_fb16()** does not use the **startlo** flag, but accepts it so that its calling sequence is the same as for **read_fb12**. This allows for a small time improvement, as **read_imp()** need only decide once which of these two routines it will use, storing its address in **readfunc**; calls to the read routine are then made indirectly.

The **read_fb*nn*()** routines use **setup_triple_window()** to make three consecutive X: windows (WX0, WX1, and WX2) point to three consecutive Frame Buffer pages, with **fb_start** "visible" through WX1. The **read_fb16()** read loop is preceded by one pulled-around instruction, which reads the first word from the Frame Buffer into A1, and increments R1 by **step** (in N1). The loop begins by shifting A1 to clear the MSB, then using the multiply-as-shift trick to right-justify the pixel value. It saves this value, increments the save pointer, and gets the next word from the Frame Buffer.

read_fb12() has a somewhat different system. It holds the current offset *in pixels*

from **fb_start** in A1 — so an even value in A1 means the next pixel to be read is in the high 12 bits of the word $\frac{A1}{2}$ words away from **fb_start**, and an odd value in A1 means the next pixel to be read is in the low 12 bits of that word. The read loop first saves the pixel offset, then right-shifts A1 by one, putting the LSB (indicating an odd or even offset) in the Carry bit, and leaving half of the pixel offset (*i.e.* the offset in words) in A1, which is then copied to N1. Program flow branches depending on the state of the Carry bit: for a low pixel, the value in **fb_start** + offset_in_pixels (R1+N1) is read into B1, and the top 12 bits zeroed; for a high pixel, this same value is read, but it is shifted-and-multiplied to place the high pixel in the low 12 bits of B1. The right-justified single-pixel value in B1 saved, the pixel offset is restored to A1 and incremented by **step**, and the loop repeats.

The **read_fb***nn*() utilities can thus read an entire line's worth of one slice's contribution to the box desired. To use this effectively, **read_imp()** reads the first line of the first slice's contribution; it then increments the **fb_start** passed to **read_fb***nn*() by one full line, and increments the pointer into which the pixels will be saved by one "box line" (*i.e.* the number of columns in the box to be read, **pixsnc**). This allows the next line to be read into the correct location of the return array. The two pointers are again incremented, repeating for the height of the box. Once the first slice's contribution has been read into the correct portions of the return array, the pointers and other variables are reset to receive the first line of the second slice's contribution to the box, and the line-by-line loop repeats. Again, this is advantageous because the time-consuming **get_imp_offset()** routine is only called once for each slice that contributes any data to the box.

As noted in subsection 7.2.3, a separate routine is available for highly optimized 3-by-3-pixel reads from a single slice saved as a Mode 3 deal-on-read frame. **tbt()** in **BEPvidout.c** may be called to perform this task: it takes the (X, Y) coordinate defining the lower-left corner of this 3x3 square, the **frame_specs_struct** and camera number to find the desired data, and a pointer to the nine-element return array. **tbt()** first calculates **ad**, the Frame Buffer address of the beginning of the line containing the lower-left corner; it then decides whether this 3x3 region is in slice A, B, C, or D.

When it has determined the slice, it increments **ad** to point to the address containing the lower-left pixel, and calls **tbt_hi()** for slices A or C, or **tbt_lo()** for slices B or D. These Assembly routines in **FEPtbt.asm** perform the rest of the work. They are both written completely sequentially, not using any jumps or loops, to save time. They accept **ad**, the pointer to the return array, and the number of words per line **linelen**, as arguments.

tbt_hi() orients WX1 to the page which includes **ad**, and WX2 to the following page: since the routine must increase Frame Buffer addresses both to go "right" and "up" *for slices A and C*, the preceding page need never be accessed. Further, since three lines fit comfortably in a single 4K window (the longest expected Mode 3 line is around 1K of data), these two adjoining windows contain all the data necessary for the 3x3 read operation. With R1 containing the Frame Buffer window address of the lower-left corner, the read operation begins: the lower-left corner is read, right-justified, and saved; R1 is incremented by 4 to get to the next pixel; this lower-center pixel is read, right-justified, and saved; R1 is incremented by 4 again; the lower-right pixel is read, right-justified, and saved; R1 is incremented by **linelen**–8, to point to the center-right pixel, and this middle line is read; similarly for the top line.

tbt_lo() is very similar, except that it must also set WX0 to the page preceding the one containing the lower-left corner, because although Frame Buffer addresses must increase to get to higher lines, they must *decrease* to go "right" in an image-region sense, due to the "backward" readout of slices B and D. So now when a pixel is read, R1 is *decremented* by 4 to get to the next pixel horizontally, and then incremented by **linelen**+8 to get to the left side of the next line.

If the GPP requests a frame dump, or a portion of one — in either event a raster-format array of image pixels — **dump_imp_array()** in **BEPframe.c** is used. This function simply grabs the tallest fully-wide box that will fit in an IPP message, collects pixel values from the following line to fill the rest of the message, and sends the message; it then puts the rest of the partially-read line into a new message, and again fills this new message with a box of data plus a portion of the following line. This process repeats until all of the data the GPP requested has been downloaded.

Chapter 9

Future Possibilities

For I dipt into the future, far as human eye could see,
Saw the Vision of the world, and all the wonder that would be.

Alfred, Lord Tennyson
Locksley Hall

This chapter outlines possible improvements and expansions to the included FEP and BEP code. If the algorithms and principles outlined in the preceding chapters are followed — use of the mainloop; ignorance of specifics concerning IPP recipients; utilization of the 56001's time-optimization capacities such as parallel move operations — augmenting the software should be a simple operation, keeping the DSPs both efficient and robust. Both of these software suites have been moderately tested; far more extensive testing is called for to ensure proper operation before launch.

The FEP as written performs all the functions it was designed for. The single largest augmentation I can see is optimization of the routine which compares an event to the bad-column and bad-pixel lists. If the lists were sorted when received, so that they were numerically ordered (by increasing column, let's say), comparisons need only be done until the member of the list being compared had a column greater than the event, instead of searching the entire list. That's an immediate 50% average improvement in searching speed. If the comparison proceeded in a binary-search fashion, there is a theoretical improvement from $O(n)$ operation to $O(\ln n)$, but the

specifics of implementing this scheme mean that it could well take more time for lists with less than several thousand members.

The BEP included here provides only the infrastructure for further capabilities — astrometry, photometry, and attitude determination. This infrastructure itself, however, bears improvement. The synchronization of frames and rows could be sped up, to match or even exceed the FEP's optimizations; the read functions, though they are optimized to a substantial level, are still uncomfortably slow; and there are still big question marks about deal-on-write, specifically:

- Do the **start_row** and **end_row** define the limits of what interests the GPP *within* the data that arrives over the video channel, or the limits of what actually does arrive over the video channel (*i.e.* the GPP commands the Lasagna Box not to transmit uninteresting rows)? The latter is currently assumed, but the former allows for more flexibility.

- Similarly, does **used_cameras** define the cameras that have data *present* on the video channel, or the cameras whose data we want to save *if* present?

- Is deal-on-write really necessary? If the science team determines that the BEP is never going to be requested to save a tall, narrow box of data, then the advantage of this scheme disappears, and both the read and the write system become greatly simplified.

The most benefit, though, would I believe result from a revamping of the Frame Buffer region system. To wit, most of the constants used ought by right be variables, specifically

- The number of regions. Currently fixed at 3, but why not make it adjustable to suit changing needs? Note that allowing for some large number of regions, say ten or more, and then only allocating Frame Buffer space to a portion of them, can accomplish this; but this would require that the number of frames, or more to the point the number of **frame_specs_structs**, alloted to a region must also be made variable, as noted in the next item.

- The number of frames per region. Definitely the biggest potential advantage lies here, but also possibly the biggest can of worms. Theoretically, since regions are orchestrated as closed-loop linked lists with some occasional offshoots (inviolate frames), it shouldn't matter what the actual indices into the frame specifics structures **fb->fs[*framenum*]** are. Hence instead of 1000 consecutive indices being allocated to each successive region, the BEP could go out into **fb->fs** and grab as many free **frame_specs_structs** as it needs *when a region is being configured.* And then when a region is reconfigured such that it holds fewer frames, it can release its claim on these structures, to be used as other regions become reconfigured. With this dynamic form of frame allocation, region 0 could hold 4 frames while region 1 holds 2000, without wasting space unnecessarily by allocating two thousand frames to *every* region. Hence the 12,000 words of global space taken up by 3000 **frame_specs_structs** could be drastically reduced. The problem here lies in changing all of the simple calculations, especially in **BEPframe.c**, that rely on an absolute assignment of frame indices to regions.

- The parameters governing regions. Some more intelligent way to reconfigure regions, especially their floors and ceilings, that does not render the frames within them unreadable, would be nice. Note that each **frame_specs_struct** points to a **region_params_struct**, and these two structures are all that is needed to pull information out of a frame. The trick would lie in saving this previous-configuration information, while still responding to the GPP's new-configuration command; and from there, to keep track of which frames have been overwritten by new data, and which are still valid.

Again I stress that the first order of business is exhaustive testing of the current system; and I encourage future developers to follow the patterns established in this document and in the software it describes.

Chapter 10

Conclusion

If we shadows have offended,
Think but this — and all is mended —
That you have but slumber'd here
While these visions did appear.
And this weak and idle theme,
No more yieldling but a dream.

— *William Shakespeare*
A Midsummer Night's Dream, V.ii

Software for the Front End Processor and the Back End Processor, the two Digital Signal Processors in the HETE spacecraft UV system, has been created. The design questions and implementation concerns encountered have been presented in this document, and the code itself is appended.

To understand the environment within which these Processors will work, the responsibilities of the UV system, and the architecture of the HETE computer system (both on-board and ground-based), were described. As the DSPs functioned as a computational buffer between the low-level charge-coupled device camera electronics and the high-level General Purpose Processor, the specifics of each of these devices were offered as they relate to and impact on the DSPs and their software.

To put the endeavour in context, the history of the HETE mission was recounted, including a description of what gamma-ray bursts are, why they are of scientific interest, and how the HETE satellite is intended to assist in the understanding of these phenomena. Sources for further research in this area are cited in the bibliography.

HETE is scheduled for launch on a Pegasus rocket on 30 April 1995, and will serve the scientific community by gathering data for one to two years. The uniquely wide data-dissemination characteristics of the mission will ensure that many researchers are able to make use of the information HETE will gather.

It was an honor to contribute to the HETE project.

Appendix A

IPP Messages Associated with the FEP

GPP <--> FEP COMMUNICATION

GPP -> FEP MESSAGES

Commands:

PROCESS_VIDEO

DONT_PROCESS_VIDEO

Parameter-changes:

BAD_COLUMN_LIST

BAD_PIXEL_LIST

CCD_CONFIGURATION

EVENT_THRESHOLDS

LB-direct:

LB_COMMANDS

```
Test routines:
TDB_COMMAND
VIDDSP_COMMAND


FEP -> GPP MESSAGES
-------------------
Video-related: (Responses to sender of PROCESS_VIDEO)
DSP_FRAME_STARTED
DSP_FRAME_ENDED
DSP_FRAME_ERROR
FEP_XY_EVENT_LIST
FEP_XYZ_EVENT_LIST

LB-related:
LB_RESPONSES (Responses to sender of LB_COMMANDS)

Errors & Exceptions:
DSP_EXCEPTION (LBox cmd/HK errors to sender of LB_COMMANDS,
video errors to sender of PROCESS_VIDEO)

Test routines: (Responses to sender of any Test routine message)
TDB_RESPONSE
VIDDSP_RESPONSE


================================================================================

Message array associations
```

Format:

IPP_MESSAGE_TYPE

0: What the value in message[0] means

1: What the value in message[1] means

2-3ff: "Same as 0-1:" indicates the continuation of a list

Values which are greater than 16 bits are send LSB-first, i.e. the low 16 bits in message[n], the high 16 bits in message[n+1].

PROCESS_VIDEO

0: 0 for a single frame with Z values, 1 for continuous frames sans Z

DONT_PROCESS_VIDEO

No parameters

BAD_COLUMN_LIST

0: 1 to append previous list, 2 to replace it

1: Number of bad columns in list

2: Bad column number, with camera # in bit 15 (0x8000)

3ff: Same as 2:

BAD_PIXEL_LIST

0: 1 to append previous list, 2 to replace it

1: Number of bad pixels in list

2: Bad pixel X, with camera # in bit 15 (0x8000)

3: Bad pixel Y

4-5ff: Same as 2-3:

CCD_CONFIGURATION:

0: 1 to install config immediately, 2 to wait until next frame

1: Transmission mode (0-3)

2: Cameras used (0x1 for Cam 0, 0x2 for Cam 1, 0x3 for both)

3: Number of horizontal underclocks per slice

4: Number of active columns per slice

5: Number of horizontal overclocks per slice

6: Number of active rows

7: Number of vertical overclocks

8: Number of first row read out

EVENT_THRESHOLDS:

0: Event-check threshold for Slice A, Camera 0

1: Event-check threshold for Slice B, Camera 0

2: Event-check threshold for Slice C, Camera 0

3: Event-check threshold for Slice D, Camera 0

4: Event-check threshold for Slice A, Camera 1

5: Event-check threshold for Slice B, Camera 1

6: Event-check threshold for Slice C, Camera 1

7: Event-check threshold for Slice D, Camera 1

LB_COMMANDS:

As-yet undefined

TDB_COMMAND

0: Low 16 bits of command

1: High 8 bits of command

VIDDSP_COMMAND

0: VIDDSP command type

1ff: Parameters specific to command

DSP_FRAME_STARTED

0: Low 16 bits of Frame ID

1: High 8 bits of Frame ID

DSP_FRAME_ENDED

0: Low 16 bits of Frame ID

1: High 8 bits of Frame ID

DSP_FRAME_ERROR

0: Low 16 bits of Frame ID

1: High 8 bits of Frame ID

2: Reason for error

FEP_XY_EVENT_LIST

0: Low 16 bits of Frame ID

1: High 8 bits of Frame ID

2: 0x1 if first list for this frame, 0x2 if last (0x0 if neither,
0x3 if both, i.e. only one list for this frame)

3: Event X, with camera # in bit 15 (0x8000)

4: Event Y

5-6ff: Same as 3-4:

FEP_XYZ_EVENT_LIST

0: Low 16 bits of Frame ID

1: High 8 bits of Frame ID

2: 0x1 if first list for this frame, 0x2 if last (0x0 if neither,

0x3 if both, i.e. only one list for this frame)

3: Event X, with camera # in bit 15 (0x8000)

4: Event Y

5: Event Z

6-8ff: Same as 3-5:

LB_RESPONSES

0ff: Housekeeping word received from Lasagna Box

DSP_EXCEPTION

0: Exception type (hk receive error, command transmit error, etc.)

1: Exception status code (reason)

TDB_RESPONSE

As-yet undefined

VIDDSP_RESPONSE

0: VIDDSP response type

1ff: Parameters specific to response

Appendix B

IPP Messages Associated with the BEP

```
GPP <--> BEP COMMUNICATION


GPP -> BEP MESSAGES
-------------------
Commands:
PROCESS_VIDEO
DONT_PROCESS_VIDEO

Parameter-changes:
SWITCH_REGION
REALLOCATE_FB
CCD_CONFIGURATION
SAVE_FRAME

Data-requests:
DUMP_FRAME_RAW
```

```
DUMP_FRAME_DEAL
DUMP_IMP_ARRAY
DUMP_TBT_ARRAY

Test routines:
VIDDSP_COMMAND


FEP -> GPP MESSAGES
-------------------
Video-related: (Responses to sender of PROCESS_VIDEO)
DSP_FRAME_STARTED
DSP_FRAME_ENDED
DSP_FRAME_ERROR

Dump-related: (Responses to sender of dump request)
BEP_UNKNOWN_ID
BEP_RAW_FRAME
BEP_DEALT_IMPS
BEP_TBT

Errors & Exceptions: (Responses to sender of PROCESS_VIDEO)
DSP_EXCEPTION

Test routines: (Responses to sender of any Test routine message)
VIDDSP_RESPONSE


================================================================================
```

Message array associations

Format:

IPP_MESSAGE_TYPE

0: What the value in message[0] means

1: What the value in message[1] means

2-3ff: "Same as 0-1:" indicates the continuation of a list

Values which are greater than 16 bits are send LSB-first, i.e. the low 16 bits in message[n], the high 16 bits in message[n+1].

PROCESS_VIDEO

0: 0 for a single frame, 1 for continuous frames

DONT_PROCESS_VIDEO

No parameters

SWITCH_REGION

0: FB region in which succeeding frames should be stored

REALLOCATE_FB

0: Region 0 start address (FB space): low 16 bits

1: Region 0 start address (FB space): high 8 bits

2-3ff: Same as 0-1:

CCD_CONFIGURATION:

0: 1 to install config immediately, 2 to wait until next frame

1: Transmission mode (0-3)

2: Cameras used (0x1 for Cam 0, 0x2 for Cam 1, 0x3 for both)

3: Number of horizontal underclocks per slice

4: Number of active columns per slice

5: Number of horizontal overclocks per slice

6: Number of active rows

7: Number of vertical overclocks

8: Number of first image row read out, cam 0

9: Number of last image row read out, cam 0

10: Number of first desired image column, cam 0

11: Number of last desired image column, cam 0

12-15: Same as 8-11: for cam 1

16: 1 to deal pixels on input, 0 otherwise

17: 1 to save non-image pixels in deal-on-input mode, 0 otherwise

30: FB region to which these parameters apply

SAVE_FRAME

0: ID of frame to mark inviolate: low 16 bits

1: ID of frame to mark inviolate: high 8 bits

DUMP_FRAME_RAW

0: ID of frame to dump: low 16 bits

1: ID of frame to dump: high 8 bits

DUMP_FRAME_DEAL

0: ID of frame to dump: low 16 bits

1: ID of frame to dump: high 8 bits

3: Camera

```
DUMP_IMP_ARRAY
0: ID of frame to dump: low 16 bits
1:  ID of frame to dump: high 8 bits
3: Camera
4: Start column
5: End column
6: Start row
7: End row

DUMP_TBT_ARRAY
0: ID of frame to dump: low 16 bits
1:  ID of frame to dump: high 8 bits
3: Camera
4: Lower-left X
5: Lower-left Y

VIDDSP_COMMAND
0: VIDDSP command type
1ff: Parameters specific to command


DSP_FRAME_STARTED
0: Low 16 bits of Frame ID
1: High 8 bits of Frame ID

DSP_FRAME_ENDED
0: Low 16 bits of Frame ID
1: High 8 bits of Frame ID

DSP_FRAME_ERROR
```

0: Low 16 bits of Frame ID

1: High 8 bits of Frame ID

2: Reason for error

BEP_UNKNOWN_ID

0: Low 16 bits of Frame ID

1: High 8 bits of Frame ID

BEP_RAW_FRAME

0: Low 16 bits of Frame ID

1: High 8 bits of Frame ID

2: 0x1 if first list for this frame, 0x2 if last (0x0 if neither,

0x3 if both, i.e. only one list for this frame)

3: Low 16 bits of the first word in this packet

4: High 8 bits of the first word in this packet

5-6ff: Same as 3-4:

BEP_DEALT_IMPS

0: Low 16 bits of Frame ID

1: High 8 bits of Frame ID

2: 0x1 if first list for this frame, 0x2 if last (0x0 if neither,

0x3 if both, i.e. only one list for this frame)

4: Start column

5: End column

6: Start row

7: End row

8: Low 16 bits of the first word in this packet

9: High 8 bits of the first word in this packet

10-11ff: Same as 8-9:

```
BEP_TBT
0: Low 16 bits of Frame ID
1: High 8 bits of Frame ID
3: Camera
4: Lower-left X
5: Lower-left Y
6ff: Pixel values (left to right, bottom to top)

DSP_EXCEPTION
0: Exception type (video receive error, etc.)
1: Exception status code (reason)

VIDDSP_RESPONSE
0: VIDDSP response type
1ff: Parameters specific to response
```

Bibliography

[1] Tye Brady. *Lasagna Box User's Guide.* Center for Space Research CCD Lab, 1993.

[2] George R. Ricker *et al.* The high energy transient experiment (hete): An international multiwavelength mission. *Proposal*, 1990.

[3] K. Hurley. Gamma ray burst positions. *Gamma Ray Transients and Related Astrophysical Phenomona (AIP)*, 1981.

[4] K. Hurley. Astronomical issues: Optical flashes. *Gamma-Ray Bursts (AIP)*, 1984.

[5] Motorola Inc. *DSP56000/DSP56001 User's Manual.* Motorola Inc., 1989.

[6] Motorola Inc. *G56KCC User's Manual.* Motorola Inc., 1991.

[7] Edison P. Liang and Vahé Petrosian. Foreword. *Gamma-Ray Bursts (AIP)*, 1984.

[8] R.W. Klebesadel; I.B. Strong; R.A. Olson. L85. *Astrophysics Journal 182*, 1973.

[9] Laurence E. Peterson. Gamma-ray spectroscopy: An historical perspective. *Nuclear Spectroscopy of Astrophysical Sources (AIP)*, 1987.

[10] Stanford E. Woosley. Foreword. *High Energy Transients in Astrophysics (AIP)*, 1983.

[11] Stanford E. Woosley. The high energy transient explorer. *High Energy Transients in Astrophysics (AIP)*, 1983.

FRONT-END PROCESSOR (FEP) SOFTWARE

gmake PBFEP for Pizza-Box load file pb_FEP.hld. PB-specific files read
from, and module cln's written to, pbcln/.
gmake ASMFEP for assembly compilation of FEP modules (put in asm/).

Map, interim load, and list files are put in obj/.

Relies on ipp.h / buddy_chk.h and subsequent files to define the following:

VARIABLE TYPES
==============
- IPP_address
- IPP_msg_type
- IPP_priority
- IPP_msg_buffer
- IPP_header

ROUTINES / MACROS
=================
- check_buddy_time()
- prepare_IPP_headers()
- IPP_make_header()
- IPP_send_message()
- IPP_get_message()
- IPP_register_address()
- IPP_get_address()
- IPP_length()
- IPP_source()
- IPP_type()

IPP NAMES & VARIABLES
=====================
- FEP
- DEFAULT_VIDEO_DEST_FEP
- DEFAULT_LB_DEST
- DEFAULT_TESTMSG_DEST
- IPP_DATASIZE
- my_ipp_addr

MESSAGE TYPES & PRIORITIES
==========================

DSP_FRAME_STARTED	DSP_FRAME_STARTED_PRIORITY
DSP_FRAME_ENDED	DSP_FRAME_ENDED_PRIORITY
DSP_FRAME_ERROR	DSP_FRAME_ERROR_PRIORITY
DSP_EXCEPTION	DSP_EXCEPTION_PRIORITY
FEP_XYZ_EVENT_LIST	FEP_EVENT_LIST_PRIORITY
FEP_XY_EVENT_LIST	
LB_RESPONSES	LB_PRIORITY
VIDDSP_RESPONSE	VIDDSP_PRIORITY
UNKNOWN_TYPE	DEFAULT_PRIORITY
PROCESS_VIDEO	
DONT_PROCESS_VIDEO	
BAD_COLUMN_LIST	
BAD_PIXEL_LIST	
CCD_CONFIGURATION	
EVENT_THRESHOLDS	
LB_COMMANDS	
TDB_COMMAND	
VIDDSP_COMMAND	

FLAGS, MASKS, ETC.
==================

FRAME_ERROR_ROWFAIL	FRAME_ERROR = row sync error
CAMERA_MASK	0x8000, for encoding camera number in coordinate
LBCMD_READ_MASK	0x800000, the R/W bit in an LB command word
LIST_APPEND	Does msg supplant or augment current data?
DSP_INSTALL_NEW_CCD_CONFIGURATION_NOW	New params take effect immediately
SSI_RX_EXCEPTION	DSP_EXCEPTION = SSI receive error (video)
SSI_TX_EXCEPTION	DSP_EXCEPTION = SSI transmit error (LB cmd)
SCI_RX_EXCEPTION	DSP_EXCEPTION = SCI receive error (LB hk)
EVENTS_FIRST_LIST	Flag indicating message is first in a set
EVENTS_LAST_LIST	Flag indicating message is last in a set
EVENTS_MIDDLE_LIST	0: Absence of above flags
MAX_BAD_COLUMNS	Maximum number of bad columns recordable
MAX_BAD_PIXELS	Maximum number of hot pixels recordable

LASAGNA BOX SPECIAL COMMANDS / WORDS
====================================
- LB_IDLE_COMMAND
- REPEATMODE
- TXD_ON_CMD
- TXD_OFF_CMD
- SC1_ON_CMD
- SC1_OFF_CMD

VIDDSP TEST MESSAGES
====================

VD_MEMDUMP	Memory peek
VD_TEST_WORD	16-bit word without specific association
VD_TEST_INT	24-bit word without specific association
VD_SIMULATE	Engage / Disengage simulation mode
VD_FORCENEW	Force vbpnew to become current
VD_ECHO	Echo back the word enclosed
VD_MEMPOKE	Memory poke
VD_VPDUMP_BEP	vidbuf_params dump
VD_NVPDUMP_BEP	next_vidbuf_params dump
VD_GVDUMP	global variables dump (R5, R6, simulating, etc.)
VD_BCDUMP	bad-columns dump
VD_BPDUMP	bad-pixels dump
VD_SINGLELOOP	Engage / Disengage single-mainloop mode
VD_DOLOOP	Perform a single mainloop
VD_UNREC_CMD	Unrecognized VIDDSP command

```
/* Module handling high-level communications with the GPP and Lasagna Box */

#include "FEPincl.h"

/* Macro for reading an address in X space */
#define read_eventspace(ad, val) __asm("move x:(%1),%0" : "=S"(val) : "A"(ad));

  extern IPP_address me, video_dest, lb_dest, testmsg_dest;

/* Command that should be stuffed into the LBox command buffer */
int lb_fill_command = LB_IDLE_COMMAND;

/* Get number of data words in status input queue */
#define scirxdata() ((sci_rx_intrs-sci_rxq_xfers)&0xffff)

/* Get number of free words on command output queue */
#define ssitxfree() ((ssi_tx_intrs-ssi_txq_xfers)&0xffff)

extern int simulating, doloop, single_loop;
extern int ssi_rx_exception, ssi_tx_exception, sci_rx_exception;

#define  lbcmd_read(cmd)   (((cmd) & LBCMD_READ_MASK) == 0)
#define  sizeIPPbuf   3
#define  MAX_LBREAD_TIME   20000

struct IPPmsg {
  IPP_header        hdr;
  IPP_msg_buffer    msg;
};
struct IPPmsg IPPbuf[sizeIPPbuf];   /* Buffer of TDB IPP messages.  */
int           IPPbufHead, IPPbufTail;  /* Pointers for IPPbuf.  */
int           currentCmd;   /* currentCmd points to next cmd to be written.  */
int           currentResponse;  /* currentResponse points to next response to be written
 */
IPP_msg_buffer    IPPresponse;  /* IPP_msg_buffer with responses to current command.  */
int           read_pending;     /* If a read command has been sent out, and no answer re
ceived. */


void Send_Host_Message(IPP_address dest, IPP_msg_type type,
               IPP_priority priority, unsigned short data_len, IPP_msg_buffer msg)
{
  IPP_header head;

  IPP_make_header(&head, priority, dest, me, type, data_len);
  IPP_send_message(&head, msg);
}

int Get_Host_Message(IPP_header *head, IPP_msg_buffer msg)
{
  int got_msg;
  int retval;

  got_msg = IPP_get_message(head, msg);
  if (got_msg == -1) retval = 0;
  else retval = 1;

  return retval;
}

void send_test_word(unsigned short test_word)
{
  IPP_msg_buffer msg;

  msg[0] = VD_TEST_WORD;
  msg[1] = test_word;

  Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE, VIDDSP_PRIORITY,
            2, msg);
}

void send_test_int(int test_int)
{
  IPP_msg_buffer msg;

  msg[0] = VD_TEST_INT;
  msg[1] = (test_int & 0xFFFF);
  msg[2] = (test_int >> 16);

  Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE, VIDDSP_PRIORITY,
            3, msg);
}

int write_lbcmd(int command)
{
  int room_on_q;

  room_on_q = (ssitxfree() > 0);

  if (room_on_q)
    {
      *ssi_tx_inptr++ = command;
      ssi_txq_xfers++;
      if (ssi_tx_inptr > ssi_txq_end) ssi_tx_inptr = ssi_txq;
    }

  return room_on_q;
}

void param_dump(p, which)
     vidbuf_param_struct p;
     unsigned short which;
{
  IPP_msg_buffer msg;
  int i;

  msg[0] = which;
  msg[1] = p.ccd.mode;
  msg[2] = p.ccd.used_cameras;
  msg[3] = p.ccd.num_hucs;
  msg[4] = p.ccd.num_cols;
  msg[5] = p.ccd.num_hocs;
  msg[6] = p.ccd.total_cols;
  msg[7] = p.ccd.num_rows;
  msg[8] = p.ccd.num_vocs;
  msg[9] = p.ccd.start_row;
  msg[10] = p.ccd.total_rows;
  for (i=0; i<TOTALSLICES; i++) msg[i+11] = p.thresh[i];
  msg[TOTALSLICES+11] = p.bad->num_bad_columns;
  msg[TOTALSLICES+12] = p.bad->num_bad_pixels;
  msg[TOTALSLICES+13] = (unsigned short)(p.id & 0x00FFFF);
  msg[TOTALSLICES+14] = (unsigned short)(p.id >> 16);
  msg[TOTALSLICES+15] = (unsigned short)p.left_off;
  msg[TOTALSLICES+16] = (unsigned short)p.this_line_start;
  msg[TOTALSLICES+17] = (unsigned short)p.one_line_delta;
  msg[TOTALSLICES+18] = (unsigned short)p.lines_rcvd;
  msg[TOTALSLICES+19] = (unsigned short)p.waiting_for_frame_sync;
  msg[TOTALSLICES+20] = (unsigned short)p.do_sifting;
  msg[TOTALSLICES+21] = (unsigned short)p.send_z;
  msg[TOTALSLICES+22] = (unsigned short)p.numevents;
  msg[TOTALSLICES+23] = (unsigned short)p.events_ptr;

  Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE, VIDDSP_PRIORITY,
```

```
            TOTALSLICES+24, msg);
}

void parse_viddsp_cmd(IPP_msg_buffer msg, vidbuf_param_struct *vbp,
              vidbuf_param_struct *vbpnew)
{
  int i;

  switch(msg[0])
    {
    case VD_SIMULATE: simulating = msg[1];
      init_portc_intrs();
      break;
    case VD_FORCENEW: abort_stream(vbp, vbpnew);
      break;
    case VD_ECHO:  Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE,
                        VIDDSP_PRIORITY, 2, msg);
      break;
    case VD_MEMDUMP: memdump_cmd(msg);
      break;
    case VD_MEMPOKE: mempoke_cmd(msg);
      break;
    case VD_VPDUMP:  param_dump(*vbp, VD_VPDUMP);
      break;
    case VD_NVPDUMP: param_dump(*vbpnew, VD_NVPDUMP);
      break;
    case VD_GVDUMP:  msg[1] = (unsigned short)asm_inptr;
      msg[2] = (unsigned short)simulating;
      msg[3] = (unsigned short)single_loop;
      __asm("move R6,%0" : "=S"(msg[4]) : );
      Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE,
            VIDDSP_PRIORITY, 5, msg);
      break;
    case VD_BCDUMP:  for (i = 0; i < vbp->bad->num_bad_columns; i++)
      msg[i+1] = vbp->bad->bad_columns[i];
      Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE,
            VIDDSP_PRIORITY, vbp->bad->num_bad_columns + 1,
            msg);
      break;
    case VD_BPDUMP:  for (i = 0; i < vbp->bad->num_bad_pixels; i++)
      {
    msg[(2*i)+1] = vbp->bad->bad_pixels_x[i];
    msg[(2*i)+2] = vbp->bad->bad_pixels_y[i];
      }
      Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE,
            VIDDSP_PRIORITY, (2*vbp->bad->num_bad_pixels)
            + 1, msg);
      break;
    case VD_SINGLELOOP: single_loop = msg[1];
      doloop = !single_loop;
      break;
    case VD_DOLOOP:  doloop = 1;
      break;
    default: msg[1] = msg[0];
      msg[0] = VD_UNREC_CMD;
      Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE,
            VIDDSP_PRIORITY, 2, msg);
    }
}

/* The following parse_tdb_command has been modified to work with a
 * ring buffer of IPP messages, which contain unknown numbers of tdb
 * commands.  It has been modified 11Mar94 by aka.
 */

void parse_tdb_command(void)
{
  int    tdb_command;
  static int repeatmode = 0;

/* If the following if statement is true, then all commands in the
 * current message have been sent, and all the responses have been
 * received.  In this case, a response message should be sent, and the
 * current message should be removed from the queue.
 */
  if (IPPbufHead == IPPbufTail)  /* The buffer queue is empty! */
    {
      try_fill_lbcmd();
      return;
    }
  if ((currentCmd == IPP_length(&(IPPbuf[IPPbufHead].hdr)))
      && (!read_pending))
    {
/*      Send_Host_Message(IPP_source(&(IPPbuf[IPPbufHead].hdr)),
            LB_RESPONSES+4, LB_PRIORITY, 0, IPPresponse); */
      /* Send filled Response buffer.  */
      if (currentResponse > 0)
    Send_Host_Message(IPP_source(&(IPPbuf[IPPbufHead].hdr)),
              LB_RESPONSES, LB_PRIORITY, currentResponse, IPPresponse);
      currentResponse = 0;
      IPPbufHead = ((IPPbufHead + 1) % sizeIPPbuf);  /* inc Head to remove current msg *
/
      currentCmd = 0;
    }
  if (IPPbufHead == IPPbufTail)  /* The buffer queue is empty! */
    {
      try_fill_lbcmd();
      return;
    }

  tdb_command = ((0xFFFF & IPPbuf[IPPbufHead].msg[currentCmd]) +
        (0xFF0000 & (0x10000 * (0xFF & IPPbuf[IPPbufHead].msg[currentCmd+1]))));

  while (!read_pending)    /* A read command has not been written yet.  */
    {
      switch(tdb_command)
    {
    case REPEATMODE: lb_fill_command = 0;
      repeatmode = !repeatmode;
      break;
    case TXD_ON_CMD: txd_on();
      break;
    case TXD_OFF_CMD: txd_off();
      break;
    case SC1_ON_CMD: sc1_on();
      break;
    case SC1_OFF_CMD: sc1_off();
      break;
    default:if (repeatmode) lb_fill_command = tdb_command;
      if (write_lbcmd(tdb_command)){
        if (lbcmd_read(tdb_command))
          {
        read_pending = 1;
/*      Send_Host_Message(IPP_source(&(IPPbuf[IPPbufHead].hdr)),
                LB_RESPONSES+2, LB_PRIORITY, 0, IPPresponse); */
          }
      }
      else
        return;  /* Can't write anything to the queue. */
    }
      currentCmd += 2;  /* Increment since this command was sent. */
      if (currentCmd == IPP_length(&IPPbuf[IPPbufHead].hdr))
    {
      try_fill_lbcmd();
```

```
/*    Send_Host_Message(IPP_source(&(IPPbuf[IPPbufHead].hdr)),
                LB_RESPONSES+3, LB_PRIORITY, 0, IPPresponse); */
      return;
    }
      tdb_command = ((0xFFFF & IPPbuf[IPPbufHead].msg[currentCmd]) +
            (0xFF0000 & (0x10000 * (0xFF & IPPbuf[IPPbufHead].msg[currentCmd+1]))));
    }
  try_fill_lbcmd();
}

void try_get_message(vbp, vbpnew)
     vidbuf_param_struct *vbp, *vbpnew;
{
  int   i, istart, iend, index;

  doloop = !single_loop;

  if (((IPPbufTail + 1) == IPPbufHead) ||
      ((IPPbufTail == (sizeIPPbuf - 1)) && (IPPbufHead == 0)))  /* IPPbuf is full!! */
    {
      parse_tdb_command();
      return;
    }
  /* Buffer is not full!  */

  if (Get_Host_Message(&(IPPbuf[IPPbufTail].hdr), IPPbuf[IPPbufTail].msg))
    {
      switch(IPP_type(&(IPPbuf[IPPbufTail].hdr)))
    {
    case PROCESS_VIDEO:
      vbpnew->do_sifting = 1;
      vbpnew->send_z = !IPPbuf[IPPbufTail].msg[0];
      video_dest = IPP_source(&(IPPbuf[IPPbufTail].hdr));
      break;
    case DONT_PROCESS_VIDEO:
      vbpnew->do_sifting = 0;
      break;
    case BAD_COLUMN_LIST:
      if (IPPbuf[IPPbufTail].msg[0] == LIST_APPEND)
        {
          istart = vbpnew->bad->num_bad_columns;
          iend = min((istart + IPPbuf[IPPbufTail].msg[1]), MAX_BAD_COLUMNS);
        }
      else
        {
          istart = 0;
          iend = min(IPPbuf[IPPbufTail].msg[1], MAX_BAD_COLUMNS);
        }
      index = 2;
      for (i = istart; i < iend; i++)
        {
          vbpnew->bad->bad_columns[i] = IPPbuf[IPPbufTail].msg[index];
          index++;
        }
      vbpnew->bad->num_bad_columns = iend;
      break;
    case BAD_PIXEL_LIST:
      if (IPPbuf[IPPbufTail].msg[0] == LIST_APPEND)
        {
          istart = vbpnew->bad->num_bad_pixels;
          iend = min((istart + IPPbuf[IPPbufTail].msg[1]), MAX_BAD_PIXELS);
        }
      else
        {
          istart = 0;
          iend = min(IPPbuf[IPPbufTail].msg[1], MAX_BAD_COLUMNS);
        }
      index = 2;
      for (i = istart; i < iend; i++)
        {
          vbpnew->bad->bad_pixels_x[i] = IPPbuf[IPPbufTail].msg[index];
          vbpnew->bad->bad_pixels_y[i] = IPPbuf[IPPbufTail].msg[index+1];
          index += 2;
        }
      vbpnew->bad->num_bad_pixels = iend;
      break;
    case CCD_CONFIGURATION:
      vbpnew->ccd.mode = IPPbuf[IPPbufTail].msg[1];
      vbpnew->ccd.used_cameras = IPPbuf[IPPbufTail].msg[2];
      vbpnew->ccd.num_hucs = IPPbuf[IPPbufTail].msg[3];
      vbpnew->ccd.num_cols = IPPbuf[IPPbufTail].msg[4];
      vbpnew->ccd.num_hocs = IPPbuf[IPPbufTail].msg[5];
      vbpnew->ccd.num_rows = IPPbuf[IPPbufTail].msg[6];
      vbpnew->ccd.num_vocs = IPPbuf[IPPbufTail].msg[7];
      vbpnew->ccd.start_row = IPPbuf[IPPbufTail].msg[8];
      vbpnew->ccd.total_cols = vbpnew->ccd.num_hucs +
        vbpnew->ccd.num_cols + vbpnew->ccd.num_hocs;
      vbpnew->ccd.total_rows = vbpnew->ccd.num_rows +
        vbpnew->ccd.num_vocs;
      vbpnew->one_line_delta = ROW_SYNC_LEN +
        (4 * vbpnew->ccd.total_cols);
      if ((IPPbuf[IPPbufTail].msg[0] & DSP_INSTALL_NEW_CCD_CONFIGURATION_NOW) != 0)
        {
          vbpnew->do_sifting = 1;
          abort_stream(vbp, vbpnew);
        }
      break;
    case EVENT_THRESHOLDS:
      for (i=0; i<TOTALSLICES; i++) vbpnew->thresh[i] = IPPbuf[IPPbufTail].msg[i];
      break;
    case LB_COMMANDS:
      lb_dest = IPP_source(&(IPPbuf[IPPbufTail].hdr));
      IPPbufTail = ((IPPbufTail + 1) % sizeIPPbuf);  /* Only if a TDB command is message
 saved. */
      parse_tdb_command();
      break;
    case TDB_COMMAND:
      /*  Send_Host_Message(IPP_source(&(IPPbuf[IPPbufTail].hdr)),
          LB_RESPONSES+1, LB_PRIORITY, 0, IPPbuf[IPPbufTail].msg); */
      lb_dest = IPP_source(&(IPPbuf[IPPbufTail].hdr));
      IPPbufTail = ((IPPbufTail + 1) % sizeIPPbuf);  /* Only if a TDB command is message
 saved. */
      parse_tdb_command();
      break;
    case VIDDSP_COMMAND:
      testmsg_dest = IPP_source(&(IPPbuf[IPPbufTail].hdr));
      parse_viddsp_cmd(IPPbuf[IPPbufTail].msg, vbp, vbpnew);
      break;
    default:
      IPPbuf[IPPbufTail].msg[0] = IPP_type(&(IPPbuf[IPPbufTail].hdr));
      Send_Host_Message(IPP_source(&(IPPbuf[IPPbufTail].hdr)), UNKNOWN_TYPE,
                DEFAULT_PRIORITY, 1, IPPbuf[IPPbufTail].msg);
      break;
    }
    }
}


void try_fill_lbcmd()
{
  while (write_lbcmd(lb_fill_command)) ;
}

/* Try to put data received over the SCI (Housekeeping) line into the
```

```
 * IPPresponse, if expected, and send an error message otherwise.
 * Modified on 11Mar94 by aka.
 */

void try_relay_hk()
{
  int                   num_on_q;
  IPP_msg_buffer        msg;
  int                   i;
  int                   msglen = 0;
  static unsigned int   timeout = 0;
/*  IPP_address           errordest;*/

  /* Get how many words are waiting to be sent to the GPP */
  num_on_q = scirxdata() / 2;

  /* If we've got something to send (from this time or last time), do it */
  if (num_on_q > 0)
    {
      if ((!read_pending) || (num_on_q > 1))  /* Too many responses.  Send error! */
    {
      for (i = 0; i < num_on_q; i++)
        {
          msg[msglen++] = -1;
          msg[msglen++] = -1;
          msg[msglen++] = ((*sci_rx_outptr << 8) + (0xFF & (*(sci_rx_outptr+1))));
          sci_rx_outptr += 2;
          sci_rxq_xfers += 2;
          if (sci_rx_outptr >= sci_rxq_end) sci_rx_outptr = sci_rxq;
        }
/*    errordest = IPP_get_address(5121);
      Send_Host_Message(errordest,
                LB_UNREQUESTED, LB_PANIC_PRIORITY, msglen, msg); */
    }
      else {
    read_pending = 0;
    if (IPP_type(&(IPPbuf[IPPbufHead].hdr)) == TDB_COMMAND)
      {
        IPPresponse[currentResponse++] = ((*sci_rx_outptr << 8) +
                          (0xFF & (*(sci_rx_outptr+1))));
      }
    else if (IPP_type(&(IPPbuf[IPPbufHead].hdr)) == LB_COMMANDS)
      {
        IPPresponse[currentResponse++] = IPPbuf[IPPbufHead].msg[currentCmd - 2];
        IPPresponse[currentResponse++] = IPPbuf[IPPbufHead].msg[currentCmd - 1];
        IPPresponse[currentResponse++] = ((*sci_rx_outptr << 8) +
                          (0xFF & (*(sci_rx_outptr+1))));
      }
    sci_rx_outptr += 2;
    sci_rxq_xfers += 2;
    if (sci_rx_outptr >= sci_rxq_end) sci_rx_outptr = sci_rxq;
    timeout = 0;
      }
    }
  else if ((num_on_q == 0) && read_pending)
    {
      timeout++;
      if (timeout >= MAX_LBREAD_TIME)
    {
      /*  msg[msglen++] =  IPPbuf[IPPbufHead].msg[currentCmd - 2];
          msg[msglen++] =  IPPbuf[IPPbufHead].msg[currentCmd - 1];
          msg[msglen++] =  -1;
          Send_Host_Message(IPP_source(&(IPPbuf[IPPbufHead].hdr)),
          LB_TIMEOUT, LB_PANIC_PRIORITY, msglen, msg); */
      timeout = 0;
      read_pending = 0;
      if (IPP_type(&(IPPbuf[IPPbufHead].hdr)) == TDB_COMMAND)
        {
          IPPresponse[currentResponse++] = -1;
        }
      else if (IPP_type(&(IPPbuf[IPPbufHead].hdr)) == LB_COMMANDS)
        {
          IPPresponse[currentResponse++] = IPPbuf[IPPbufHead].msg[currentCmd - 2];
          IPPresponse[currentResponse++] = IPPbuf[IPPbufHead].msg[currentCmd - 1];
          IPPresponse[currentResponse++] = -1;
        }
    }
    }
}

void try_relay_errors()
{
  IPP_msg_buffer msg;

  if (ssi_rx_exception != 0)
    {
      msg[0] = SSI_RX_EXCEPTION;
      msg[1] = (unsigned short)ssi_rx_exception;
      ssi_rx_exception = 0;
      Send_Host_Message(video_dest, DSP_EXCEPTION,
            DSP_EXCEPTION_PRIORITY, 2, msg);
    }
  if (ssi_tx_exception != 0)
    {
      msg[0] = SSI_TX_EXCEPTION;
      msg[1] = (unsigned short)ssi_tx_exception;
      ssi_tx_exception = 0;
      Send_Host_Message(lb_dest, DSP_EXCEPTION,
            DSP_EXCEPTION_PRIORITY, 2, msg);
    }
  if (sci_rx_exception != 0)
    {
      msg[0] = SCI_RX_EXCEPTION;
      msg[1] = (unsigned short)sci_rx_exception;
      sci_rx_exception = 0;
      Send_Host_Message(lb_dest, DSP_EXCEPTION,
            DSP_EXCEPTION_PRIORITY, 2, msg);
    }
}

void signal_frame_started(unsigned int id)
{
  IPP_msg_buffer msg;

  msg[0] = (unsigned short)(id & 0xFFFF);
  msg[1] = (unsigned short)(id >> 16);

  Send_Host_Message(video_dest, DSP_FRAME_STARTED,
            DSP_FRAME_STARTED_PRIORITY, 2, msg);
}

void signal_frame_ended(unsigned int id)
{
  IPP_msg_buffer msg;

  msg[0] = (unsigned short)(id & 0xFFFF);
  msg[1] = (unsigned short)(id >> 16);

  Send_Host_Message(video_dest, DSP_FRAME_ENDED,
            DSP_FRAME_ENDED_PRIORITY, 2, msg);
}

void signal_frame_error(unsigned int id, unsigned short reason)
{
```

```
    IPP_msg_buffer msg;

    msg[0] = (unsigned short)(id & 0xFFFF);
    msg[1] = (unsigned short)(id >> 16);
    msg[2] = reason;

    Send_Host_Message(video_dest, DSP_FRAME_ERROR, DSP_FRAME_ERROR_PRIORITY,
              3, msg);
  }

  void send_packet_of_events(unsigned int **event_ptr, int *events_to_send,
                 int total_events, flag send_z, unsigned int id)
  {
    IPP_msg_buffer msg;
    IPP_msg_type type;
    unsigned int val;
    int   i, event_len, msg_len;

    if (send_z)
      {
        event_len = 3;
        type = FEP_XYZ_EVENT_LIST;
      }
    else
      {
        event_len = 2;
        type = FEP_XY_EVENT_LIST;
      }

    msg_len = min(event_len*(*events_to_send) + 3, IPP_DATASIZE-2);

    msg[0] = (unsigned short)(id & 0xFFFF);
    msg[1] = (unsigned short)(id >> 16);
    msg[2] = EVENTS_MIDDLE_LIST;
    if (*events_to_send == total_events) msg[2] |= EVENTS_FIRST_LIST;

    for (i = 3; i < msg_len; i += event_len)
      {
        read_eventspace((*event_ptr), val);
        msg[i] = ((val & 0x0007FE) >> 1) + ((val & 0x000001) * CAMERA_MASK);
        msg[i+1] = val >> 12;
        if (send_z)
      {
        read_eventspace(((*event_ptr) + 1), val);
        msg[i+2] = val;
      }
        (*event_ptr) += 2;
        (*events_to_send)--;
      }

    if (*events_to_send == 0) msg[2] |= EVENTS_LAST_LIST;

    Send_Host_Message(video_dest, type, FEP_EVENT_LIST_PRIORITY, msg_len, msg);
  }

  void dump_events(vidbuf_param_struct *vbp)
  {
    int   events_to_send;
    unsigned int *dump_events_ptr;

    events_to_send = vbp->numevents;
    dump_events_ptr = EVENTS_BOTTOM;

    while (events_to_send > 0)
      {
        send_packet_of_events(&dump_events_ptr, &events_to_send,
                  vbp->numevents, vbp->send_z, vbp->id);
      }
  }
```

```
/* Main module of FEP.  Initializes and engages forever-loop.  Contains */
/* simulation- and memory-related test routines.      */

#include <stdlib.h>
#include "FEPincl.h"
#include "FEPglobals.h"


#define vb_write(val) __asm("move %0,x:(R5)+" : : "S"(val));

/* How many sets of 4 zeros to drop into the video buffer before  */
/* writing a frame sync                                           */
#define BUNCH_OF_ZEROS 1
/* Flag to indicate whether a frame sync should be inserted into the */
/* simulated video data stream                                    */
int drop_frame_sync=0;

/* Simulated video data */
/*int ccd0sim[112] =
    { 97, 98, 99,100, 101,102,103,104, 104,103,102,101, 100, 99, 98, 97,
      98, 99,100,101, 102,103,104,105, 105,104,103,102, 101,100, 99, 98,
      99,100,101,102, 103,104,105,106, 106,105,104,103, 102,101,100, 99,
      98, 99,100,101, 102,103,104,105, 105,104,103,102, 101,100, 99, 98,
      97, 98, 99,100, 101,102,103,104, 104,103,102,101, 100, 99, 98, 97,
      96, 97, 98, 99, 100,101,102,103, 103,102,101,100,  99, 98, 97, 96,
      95, 96, 97, 98,  99,100,101,102, 102,101,100, 99,  98, 97, 96, 95};*/

int ccd0sim[112] =
    { 97, 98, 99,100, 101,102,103,100, 100,103,102,101, 100, 99, 98, 97,
      98, 99,100,101, 102,103,104,100, 100,104,103,102, 101,100, 99, 98,
      99,100,101,102, 103,104,105,100, 100,105,104,103, 102,101,100, 99,
      98, 99,100,101, 102,103,104,100, 100,104,103,102, 101,100, 99, 98,
      97, 98, 99,100, 101,102,103,100, 100,103,102,101, 100, 99, 98, 97,
      96, 97, 98, 99, 100,101,102,103, 103,102,101,100,  99, 98, 97, 96,
      95, 96, 97, 98,  99,100,101,102, 102,101,100, 99,  98, 97, 96, 95};

int ccd1sim[112] =
    {100,100,100,100, 100,100,100,100, 100,100,100,100, 100,100,100,100,
     100,200,200,100, 100,100,100,200, 100,100,200,100, 100,100,200,100,
     100,100,100,100, 100,100,100,100, 100,100,100,100, 100,100,100,100,
     200,100,100,100, 200,100,100,100, 200,100,100,100, 200,100,100,200,
     200,100,100,100, 100,100,100,100, 100,100,100,100, 100,100,100,200,
     100,100,100,100, 100,100,100,100, 100,100,100,100, 100,100,100,100,
     100,100,100,200, 200,100,100,200, 200,100,100,200, 200,100,100,200};



void sim4(pat4)
int pat4;
{
    int i;

    if (pat4 == ZEROS4)
    {
        for (i=0; i<4; i++) vb_write(ZERO_MARK);
    }
    else
    {
        for (i=0; i<4; i++) vb_write(ONE_MARK);
    }


oid simulate_frame_sync()

    static int id = 0x555555;
    int i;

    for (i=0; i<BUNCH_OF_ZEROS; i++) sim4(ZEROS4);
    sim4(FRAME_SYNC1);
    sim4(FRAME_SYNC2);
    for (i=NUMIDBITS-1; i>=0; i--)
    {
        if ((id & (1<<i)) == 0) sim4(ZEROS4);
        else sim4(ONES4);
    }
    id++;
}

void simulate_row_sync()
{
    int i;

    for (i=0; i<ROW_SYNC_SKIP; i++) vb_write(0);
    sim4(ROW_SYNC1);
    sim4(ROW_SYNC2);
}

void simulate_word_rcvd()
{
    static int ABCDcnt=0, Aline=0, Bline=7, Cline=8, Dline=15;
    int sim_cntr;
/*  printf("sim_cntr = %x\n",sim_cntr);*/

    if (drop_frame_sync)
    {
        drop_frame_sync=0;
        simulate_frame_sync();
        ABCDcnt=0;
        Aline=0;  Bline=7;  Cline=8;  Dline=15;
    }
    if (ABCDcnt==0) simulate_row_sync();

    sim_cntr = (ccd0sim[Aline+ABCDcnt]*0x1000) + ccd0sim[Bline-ABCDcnt];
    vb_write(sim_cntr);
    sim_cntr = (ccd0sim[Cline+ABCDcnt]*0x1000) + ccd0sim[Dline-ABCDcnt];
    vb_write(sim_cntr);
    sim_cntr = (ccd1sim[Aline+ABCDcnt]*0x1000) + ccd1sim[Bline-ABCDcnt];
    vb_write(sim_cntr);
    sim_cntr = (ccd1sim[Cline+ABCDcnt]*0x1000) + ccd1sim[Dline-ABCDcnt];
    vb_write(sim_cntr);

    ABCDcnt++;
    if (ABCDcnt > 3)
    {
        Aline += 16;
        if (Aline > 96)
        {
            Aline = 0; Bline = 7;
            Cline = 8; Dline = 15;
        }
        else
        {
            Bline += 16;
            Cline += 16;
            Dline += 16;
        }
        ABCDcnt = 0;
    }
}

void alt_simulate_word_rcvd()
{
    static int sim_cntr = 0;
```

```
    int i;

    for (i=0; i<500; i++)
    {
        vb_write(sim_cntr);
        sim_cntr += 0x1001;
        if (sim_cntr >= (101*0x1000)) sim_cntr = 0;
    }
}

void simulate_lbcmd_sent()
{
    int *ssi_tx_outptr;

    __asm("move y:<ssi_tx_outptr,%0" : "=A"(ssi_tx_outptr) : );
    ssi_tx_outptr++;
    if (ssi_tx_outptr > ssi_txq_end) ssi_tx_outptr = ssi_txq;
    __asm("move %0,y:<ssi_tx_outptr" : : "A"(ssi_tx_outptr));
    ssi_tx_intrs++;
    if (ssi_tx_outptr > ssi_txq_end) ssi_tx_outptr = ssi_txq;
}

void xmemcpy(start, end, copy_space)
unsigned short start,end;
unsigned short *copy_space;
{
    unsigned short i;
    int     *ad, val;

    for (i=start; i<=end; i++)
    {
        ad = (int *)i;
        __asm("move x:(%1),%0" : "=S"(val) : "A"(ad));
        *copy_space++ = val & 0xFFFF;
        *copy_space++ = val >> 16;
    }
}


void ymemcpy(start, end, copy_space)
unsigned short start,end;
unsigned short *copy_space;
{
    unsigned short i;
    int     val;

    for (i=start; i<=end; i++)
    {
        val = *((int *)i);
        *copy_space++ = val & 0xFFFF;
        *copy_space++ = val >> 16;
    }
}

void pmemcpy(start, end, copy_space)
unsigned short start,end;
unsigned short *copy_space;
{
    unsigned short i;
    int     *ad, val;

    for (i=start; i<=end; i++)
    {
        ad = (int *)i;
        __asm("movem p:(%1),%0" : "=S"(val) : "A"(ad));
        *copy_space++ = val & 0xFFFF;
        *copy_space++ = val >> 16;
    }
}

void memdump_cmd(IPP_msg_buffer msg)
{
    unsigned short start, end;
    charspace;

    space = (char)msg[1];
    start = msg[2];
    end = msg[3];

/*  put_host((unsigned short)space);
    put_host(start);
    put_host(end);*/

    if ((end-start) > ((IPP_DATASIZE-10) / 2))
    {
        end = start + ((IPP_DATASIZE-10) / 2);
        msg[3] = end;
    }

    switch(space)
    {
        case 'x': xmemcpy(start, end, &(msg[4]));  break;
        case 'y': ymemcpy(start, end, &(msg[4]));  break;
        case 'p': pmemcpy(start, end, &(msg[4]));  break;
    }

    Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE, VIDDSP_PRIORITY,
        (2 * ((end-start)+1) ) + 4, msg);
}

void mempoke_cmd(IPP_msg_buffer msg)
{
    charspace;
    int     *ad, val;

    space = (char)msg[1];
    ad = (int *)msg[2];
    val = msg[3];
    val |= msg[4] << 16;
    switch(space)
    {
        case 'x': __asm volatile("move %0,x:(%1)" : : "S"(val), "A"(ad));
                  break;
        case 'y': *(ad) = val;
                  break;
        case 'p': __asm volatile("movem %0,p:(%1)" : : "S"(val), "A"(ad));
                  break;
    }
}

void memdump_auto(char space, unsigned short start, unsigned short end)
{
    IPP_msg_buffer msg;

    msg[0] = VD_MEMDUMP;
    msg[1] = (unsigned short)space;
    msg[2] = start;
    msg[3] = end;
    memdump_cmd(msg);
}


void main(void)
{
```

```
    vidbuf_param_struct vidbuf_params, next_vidbuf_params;

    next_vidbuf_params.bad =
        (bad_locs_struct *)malloc(sizeof(bad_locs_struct));

    onetime_init(&vidbuf_params, &next_vidbuf_params);

    for(;;)
    {
        check_buddy_time();
        try_get_message(&vidbuf_params, &next_vidbuf_params);
        if (doloop)
        {
            if (simulating)
            {
                simulate_lbcmd sent();
                if (vidbuf_params.waiting_for_frame_sync) drop_frame_sync = 1;
                simulate_word_rcvd();
            }
            try_proc_vid(&vidbuf_params, &next_vidbuf_params);
            parse_tdb_command();
            try_fill_lbcmd();
            try_relay_hk();
            try_relay_errors();
        }
    }
}
```

```
/* Initialization routines */

#include <stdlib.h>
#include "FEPincl.h"

extern int IPPbufHead, IPPbufTail, currentCmd, currentResponse, read_pending;

extern IPP_address me, video_dest, lb_dest, testmsg_dest;

extern int *sci_rxo_ptr;
extern int sci_rx_intrs, sci_rxo_xfers;

extern int simulating;


void init_portc_intrs()
{
/* If we're not in simulation mode, turn on Interrupts */
    if (simulating)
    {
        __asm("bclr #m_rie,x:<<m_scr");
        __asm("bclr #m_srie,x:<<m_crb");
        __asm("bclr #m_stie,x:<<m_crb");
    }
    else
    {
        __asm("bset #m_rie,x:<<m_scr");
        __asm("bset #m_srie,x:<<m_crb");
        __asm("bset #m_stie,x:<<m_crb");
    }
}



void onetime_init(vbp, vbpnew)
vidbuf_param_struct *vbp, *vbpnew;
{
    int mod, i;

    copy_lowp_routines();
    init_intr_vectors();

    me = IPP_register_address(FEP);
    video_dest = IPP_get_address(DEFAULT_VIDEO_DEST_FEP);
    lb_dest = IPP_get_address(DEFAULT_LB_DEST);
    testmsg_dest = IPP_get_address(DEFAULT_TESTMSG_DEST);

    my_ipp_addr = FEP;
    prepare_IPP_headers();

    IPPbufHead = 0;
    IPPbufTail = 0;
    currentCmd = 0;
    currentResponse = 0;
    read_pending = 0;

    asm_inptr = VIDBUF_BOTTOM;
    mod = VIDBUF_LEN-1;
    __asm("move %0,M5" : : "S"(mod));

    vbpnew->ccd.mode = 3;
    vbpnew->ccd.used_cameras = CAMERA_BOTH;
    vbpnew->ccd.num_hucs = 0;
    vbpnew->ccd.num_cols = 4;
    vbpnew->ccd.num_hocs = 0;
    vbpnew->ccd.total_cols = vbpnew->ccd.num_hucs + vbpnew->ccd.num_cols +
        vbpnew->ccd.num_hocs;
    vbpnew->ccd.num_rows = 20;
    vbpnew->ccd.num_vocs = 5;
    vbpnew->ccd.start_row = 0;
    vbpnew->ccd.total_rows = vbpnew->ccd.num_rows + vbpnew->ccd.num_vocs;

    for (i = 0; i < (TOTALSLICES); i++)
        vbpnew->thresh[i] = 100;

    (vbpnew->bad)->num_bad_columns = 0;
    (vbpnew->bad)->num_bad_pixels = 0;

    vbpnew->id = 0;
    vbp->left_off = VIDBUF_BOTTOM;
    vbpnew->one_line_delta = ROW_SYNC_LEN + (4 * vbpnew->ccd.total_cols);
    vbpnew->lines_rcvd = 0;
    vbpnew->waiting_for_frame_sync = 0;
    vbpnew->do_sifting = 1;
    vbpnew->send_z = 1;
    vbpnew->numevents = 0;
    vbpnew->events_ptr = EVENTS_BOTTOM;

    abort_stream(vbp, vbpnew);

    simulating = 0;

    init_communications();
    init_portc_intrs();
}
```

```
/* Event-finding: frame sync, row sync, event record routines; passes */
/* to proc_line() for actual event coordinate determination    */

#include "FEPincl.h"

/* Read from the video buffer (effectively val = *ad, but in X: space) */
#define read_vb(ad, val) __asm("move x:(%1),%0" : "=S"(val) : "A"(ad));

/* Go ad2 words from ad1, within the video buffer */
#define vb_add(ad1, ad2) ((((int)(ad1) + (ad2)) & VIDBUF_RELEVANT_BITS) + VIDBUF_BOTTOM)

/* The circular-buffer distance from from to to */
#define vb_distance(from, to) (((int)((to) - (from)) + VIDBUF_LEN) & VIDBUF_RELEVANT_BIT
S)

/* True if there are num words available in the video buffer between */
/* read (following) and write (leading)                              */
#define have_words(read, write, num) (vb_distance(read, write) > num)

/* Write to event space (effectively *ad = val, but in X: space) */
#define write_eventspace(ad, val) __asm("move %0,x:(%1)" : : "S"(val), "A"(ad));

extern IPP_address testmsg_dest;

int check4(ptr)
int *ptr;
/* Check the four words following ptr, and see if they */
/* all represent a one mark, a zero mark, or neither */
{
    int *ptrwork;
    int orig_val, test_val;
    int retval = MIX4;

    read_vb(ptr, orig_val);
    ptrwork = vb_add(ptr, 1);
    read_vb(ptrwork, test_val);
    if (test_val == orig_val)
    {
        ptrwork = vb_add(ptrwork, 1);
        read_vb(ptrwork, test_val);
        if (test_val == orig_val)
        {
            ptrwork = vb_add(ptrwork, 1);
            read_vb(ptrwork, test_val);
            if (test_val == orig_val)
            {
                if (orig_val == ZERO_MARK) retval = ZEROS4;
                else if (orig_val == ONE_MARK) retval = ONES4;
            }
        }
    }
    return retval;
}

void switch_params(current, new)
vidbuf_param_struct *current, *new;
{
    new->left_off = current->left_off;
    *current = *new;
    if (current->send_z) new->do_sifting = 0;
    if (!current->do_sifting) current->waiting_for_frame_sync = 1;
}

void get_coords(address, highpixel, vbp, x, y, cam)
int *address, highpixel;
vidbuf_param_struct *vbp;
unsigned short *x, *y, *cam;
/* Calculate the on-camera coordinates of a pixel given its address */
/* in the video buffer                                              */
{
    int imageslicelen, offset, div4, mod4, mod2;

/* Do we do anything here re. 2x2-summed mode? */

    imageslicelen = vbp->ccd.num_cols;
    offset = vb_distance(vbp->this_line_start, address);
    div4 = offset / 4;
    mod4 = offset % 4;
    mod2 = offset % 2;

    switch(vbp->ccd.mode)
    {
        case 0: if (mod2) *x = (mod4+1)*imageslicelen - div4 - 1;
                else *x = mod4*imageslicelen + div4;
                *cam = vbp->ccd.used_cameras - 1;
                break;
        case 1: *x = (mod2+1)*imageslicelen - div4 - 1;
                *cam = (mod4 >= 2);
                break;
        case 2: *x = mod2*imageslicelen + div4;
                *cam = (mod4 >= 2);
                break;
        case 3: if (highpixel) *x = (mod2*2)*imageslicelen + div4;
                else *x = ((mod2*2)+2)*imageslicelen - div4 - 1;
                *cam = (mod4 >= 2);
                break;
    }

    *y = (vbp->lines_rcvd - 2) + vbp->ccd.start_row;
}

void load_thresh(vbp)
vidbuf_param_struct vbp;
/* Load the asm_thresh variables used by proc_line(). */
/* Justify the thresholds as necessary: 16-bit pixels */
/* and high 12-bit pixels, left-justified and then one */
/* bit right-shifted to avoid negation confusion; low */
/* 12-bit pixels, right-justified (i.e. unchanged).  */
{
    switch(vbp.ccd.mode)
    {
        case 0: if (vbp.ccd.used_cameras == 0x01)
                {
                    asm_thresh0 = vbp.thresh[SLICE_A0] << 7;
                    asm_thresh1 = vbp.thresh[SLICE_B0] << 7;
                    asm_thresh2 = vbp.thresh[SLICE_C0] << 7;
                    asm_thresh3 = vbp.thresh[SLICE_D0] << 7;
                }
                else
                {
                    asm_thresh0 = vbp.thresh[SLICE_A1] << 7;
                    asm_thresh1 = vbp.thresh[SLICE_B1] << 7;
                    asm_thresh2 = vbp.thresh[SLICE_C1] << 7;
                    asm_thresh3 = vbp.thresh[SLICE_D1] << 7;
                }
                break;
        case 1: asm_thresh0 = vbp.thresh[SLICE_B0] << 7;
                asm_thresh1 = vbp.thresh[SLICE_D0] << 7;
                asm_thresh2 = vbp.thresh[SLICE_B1] << 7;
                asm_thresh3 = vbp.thresh[SLICE_D1] << 7;
                break;
        case 2: asm_thresh0 = vbp.thresh[SLICE_A0] << 7;
                asm_thresh1 = vbp.thresh[SLICE_C0] << 7;
```

```
                asm_thresh2 = vbp.thresh[SLICE_A1] << 7;
                asm_thresh3 = vbp.thresh[SLICE_C1] << 7;
                break;
        case 3: asm_thresh0 = vbp.thresh[SLICE_A0] << 11;
                asm_thresh1 = vbp.thresh[SLICE_B0];
                asm_thresh2 = vbp.thresh[SLICE_C0] << 11;
                asm_thresh3 = vbp.thresh[SLICE_D0];
                asm_thresh4 = vbp.thresh[SLICE_A1] << 11;
                asm_thresh5 = vbp.thresh[SLICE_B1];
                asm_thresh6 = vbp.thresh[SLICE_C1] << 11;
                asm_thresh7 = vbp.thresh[SLICE_D1];
                break;
    }
}

void load_init_proc_vars(vbp)
vidbuf_param_struct vbp;
/* Set asm_proclen so proc_line() processes all of the */
/* packages-of-four containing image pixels, except the */
/* first and last, which contain image pixels on slice */
/* boundaries.  Then load the asm_thresh variables.  */
{
    asm_proclen = vbp.ccd.num_cols - 2;

    load_thresh(vbp);
}

int sync_frame(vbp, id)
vidbuf_param_struct *vbp;
int *id;
{
    static int state=0;
    static int count=0;
    static int local_id=0;
    int *inptr, *idptr=0;
    int gotsync=0;
    int val1, val2;

    inptr = asm_inptr;

    while (have_words(vbp->left_off, inptr, 4) && !gotsync)
    {
        switch(state)
        {
            case 0: read_vb(vbp->left_off, val1);
                    vbp->left_off = vb_add(vbp->left_off, 1);
                    read_vb(vbp->left_off, val2);
                    if ((val1 == ZERO_MARK) && (val2 == ONE_MARK)) state = 1;
                    break;
            case 1: if (check4(vbp->left_off) == FRAME_SYNC1)
                    {
                        vbp->left_off = vb_add(vbp->left_off, 4);
                        state = 2;
                    } else state = 0;
                    break;
            case 2: if (check4(vbp->left_off) == FRAME_SYNC2)
                    {
                        vbp->left_off = vb_add(vbp->left_off, 4);
                        idptr = vbp->left_off;
                        count = 1;
                        local_id = 0;
                        state = 3;
                    } else state = 0;
                    break;
            case 3: val1 = check4(vbp->left_off);
                    if (val1 == MIX4)
                    {
                        vbp->left_off = idptr;
                        state = 0;
                    }
                    else
                    {
                        if (val1 == ONES4) local_id |= 1 << (NUMIDBITS-count);
                        vbp->left_off = vb_add(vbp->left_off, 4);
                        count++;
                        if (count > NUMIDBITS)
                        {
                            gotsync = 1;
                            *id = local_id;
                            state = 0;
                        }
                    }
                    break;
        }
    }
    return gotsync;
}

int sync_row(vbp)
vidbuf_param_struct *vbp;
{
    int *inptr;
    int val1, val2;
    int gotsync = 0;

    inptr = asm_inptr;
    if (have_words(vbp->left_off, inptr, ROW_SYNC_LEN) && !gotsync)
    {
        vbp->left_off = vb_add(vbp->left_off, ROW_SYNC_SKIP);
        val1 = check4(vbp->left_off);
        vbp->left_off = vb_add(vbp->left_off, 4);
        val2 = check4(vbp->left_off);
        vbp->left_off = vb_add(vbp->left_off, 4);
        if ((val1 == ROW_SYNC1) && (val2 == ROW_SYNC2)) gotsync = 1;
        else gotsync = -1;
    }
    return gotsync;
}

void try_proc_vid(vbp, vbpnew)
vidbuf_param_struct *vbp, *vbpnew;
{
    int row_syncd;
    int *inptr, *procstart;
    int up1, down1;
    int id;

    if (vbp->waiting_for_frame_sync)
    {
/* If we're in waiting-for-frame-sync mode, try to sync the frame */
        vbp->waiting_for_frame_sync = !sync_frame(vbp, &id);
        if (!vbp->waiting_for_frame_sync)
        {
/* If we've just found the frame sync, make vbpnew current, and save the ID */
            switch_params(vbp, vbpnew);
            vbp->id = id;
            if (vbp->do_sifting)
            {
/* If we're processing this frame, tell the GPP we've sync'd it, and set */
/* things up for proc_len() for this frame.                */
                signal_frame_started(vbp->id);
                load_init_proc_vars(*vbp);
            }
        }
```

```
    }
    else
    {
        inptr = asm_inptr;
        if (have_words(vbp->left_off, inptr, vbp->one_line_delta))
        {
/* If we've gotten a full row of data, try to find the row sync that */
/* precedes it.                                                      */
            row_syncd = sync_row(vbp);

            if (row_syncd == -1)
            {
/* If we couldn't find the row sync, warn the GPP and abort this frame */
                signal_frame_error(vbp->id, FRAME_ERROR_ROWFAIL);
                abort_stream(vbp,vbpnew);
            }
            else if (row_syncd == 1)
            {
/* If we found the row sync, note that we've received another line */
                vbp->lines_rcvd++;

                if ((vbp->lines_rcvd >= 3) &&
                    (vbp->lines_rcvd <= vbp->ccd.num_rows))
                {
/* If we can process the preceding line, do so: the line being processed */
/* starts one row ago, and the image portion of this line starts after the */
/* packages-of-four containing horizontal underclocks; skip the first  */
/* package-of-four in the image portion; calculate the number of words to  */
/* go "up" and "down" by one row in the vidbuf; dispatch to proc_line(). */
                    vbp->this_line_start = vb_add(vbp->left_off,
                        (4*vbp->ccd.num_hucs) - vbp->one_line_delta);
                    procstart = vb_add(vbp->this_line_start, 4);
                    up1 = vbp->one_line_delta;
                    down1 = -vbp->one_line_delta;
                    proc_line(procstart, vbp->ccd.mode, up1, down1, vbp);
                }

/* Leave off at the end of this line, ready to look for the next row sync */
                vbp->left_off = vb_add(vbp->left_off,
                    vbp->one_line_delta - ROW_SYNC_LEN);

/* If we've processed all the image lines, dump the event list to the GPP */
                if (vbp->lines_rcvd == vbp->ccd.num_rows)
                    dump_events(vbp);
                if (vbp->lines_rcvd == vbp->ccd.total_rows)
                {
/* If we've processed all the lines including VOCs, tell the GPP, and */
/* prepare to receive the next frame.                                 */
                    signal_frame_ended(vbp->id);
                    vbp->waiting_for_frame_sync = 1;
                }
            }
        }
    }
}

flag event_loc_okay(unsigned short x, unsigned short y, unsigned short cam,
    vidbuf_param_struct *vbp)
/* Check an event candidate against the bad-column and -pixel lists */
{
    int     i;
    flagokay = 1;

    x += (cam * CAMERA_MASK);

    i = 0;
    while (okay && (i < vbp->bad->num_bad_columns))
    {
        okay = (abs(x - vbp->bad->bad_columns[i]) > 1);
        i++;
    }

    i = 0;
    while (okay && (i < vbp->bad->num_bad_pixels))
    {
        okay = ((abs(x - vbp->bad->bad_pixels_x[i]) > 1) ||
                (abs(y - vbp->bad->bad_pixels_y[i]) > 1));
        i++;
    }

    return okay;
}

void record_event(address, value, hipixel, vbp)
int *address, value, hipixel;
vidbuf_param_struct *vbp;
{
    unsigned short x,y,cam;

/*
    IPP_msg_buffer msg;

    msg[0] = VD_RAWEVENT;
    msg[1] = (unsigned short)address;
    msg[2] = (unsigned short)(value & 0xFFFF);
    msg[3] = (unsigned short)(value >> 16);
    msg[4] = (unsigned short)hipixel;

    Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE, VIDDSP_PRIORITY, 5, msg);
*/

    get_coords(address, hipixel, vbp, &x, &y, &cam);

    if (event_loc_okay(x, y, cam, vbp))
    {
        if (vbp->ccd.mode != 3) value >>= 7;
        else if (hipixel) value >>= 11;

        write_eventspace(vbp->events_ptr, ((y << 12) + (x << 1) + cam));
        vbp->events_ptr++;
        write_eventspace(vbp->events_ptr, value);
        vbp->events_ptr++;
        vbp->numevents++;
        if (vbp->numevents >= MAXNUMEVENTS)
        {
            vbp->numevents = MAXNUMEVENTS-1;
            vbp->events_ptr -= 2;
        }
    }
}

void abort_stream(vbp, vbpnew)
vidbuf_param_struct *vbp, *vbpnew;
{
    vbp->waiting_for_frame_sync = 1;
}
```

```
; Various small support routines, best implemented in Assembly

    section asmsupport

    global Finit_communications
    global Fcopy_lowp_routines
    global Finit_intr_vectors
    global Fcheck_host
    global Fget_host
    global Fput_host
    global put555_host
    global Ftxd_on
    global Ftxd_off
    global Ftxd_tog
    global Fsc1_on
    global Fsc1_off

    org     p:


Finit_communications
    moveY0,y:(R6)+
    moveR1,y:(R6)+

    bset#15,x:<<m_ipr  ; Set SCI priority to 2
    bclr#14,x:<<m_ipr
    bset#13,x:<<m_ipr  ; Set SSI priority to 3
    bset#12,x:<<m_ipr

    clr     A
    movep A1,x:<<m_pcc  ; Force reset of Port C

    move#Fsci_rxq,R1  ; Init queue pointers, interface variables
    move#(SCI_RXQ_SIZE-1),Y0
    moveY0,y:<sci_rx_mod
    moveA1,y:<Fsci_rx_intrs
    moveA1,y:<Fsci_rxq_xfers
    moveR1,y:<sci_rx_inptr
    moveR1,y:<Fsci_rx_outptr
    move#Fssi_txq,R1
    move#(SSI_TXQ_SIZE-1),Y0
    moveY0,y:<ssi_tx_mod
    moveA1,y:<Fssi_tx_intrs
    moveA1,y:<Fssi_txq_xfers
    moveR1,y:<ssi_tx_outptr
    moveR1,y:<Fssi_tx_inptr
    rep     #SSI_TXQ_SIZE
    moveA1,y:(R1)+

; SSI Initialization
; 24 bits/word, no frame rate division, clock prescaling according
; to precalculated value
    movep #($6000+divider),x:<<m_cra
; Normal mode, continuous clock, asynchronous operation, one-bit clock
; for both transmit and receive, MSB-first; transmit clock is internal,
; receive clock is external.  Enable receiver, but leave transmitter
; disabled until clean startup is possible.
    movep #$2130,x:<<m_crb

; SCI Initialization
; 10-bit word (1 start, 8 data, 1 stop); enable receiver, disable
; transmitter; use pullups, don't send a break, MSB-first, uninverted clock.
    movep #$010A,x:<<m_scr
; Reveive clock is external, prescale clock
    movep #$6000,x:<<m_sccr

; Enable SSI & SCI pins on Port C, with the exception of TXD and SC1
    movep #$1FD,x:<<m_pcc
    bset#1,x:<<m_pcddr  ; Make TXD an output
    bclr#1,x:<<m_pcd    ; Clear TXD

; To cleanly start the SSI TX, one must first initialize the SSI.
; Then it must be given something to send.  Finally, turn on the
; TX, and its interrupt enable if desired.
    movep #0,x:<<m_tx       ; Give it something to send
    bset#m_ste,x:<<m_crb  ; Turn on transmitter

    movey:-(R6),R1
    movey:-(R6),Y0
    rts


Fcopy_lowp_routines
; Copy the space P:$7040-$7AAA to P:$0040-$0AAA
    moveR1,y:(R6)+
    moveR2,y:(R6)+

    move#$7040,R1
    move#$0040,R2

    do      #($AAB-$040),_end_copy
    movem p:(R1)+,A1
    movem A1,p:(R2)+
_end_copy

    movey:-(R6),R2
    movey:-(R6),R1
    rts

Finit_intr_vectors
; Initialize port C interrupt vectors.  Most vector instructions are of
; the form "jsr <[address]"; SSI receive (video data) is
; "movep x:<<m_rx,x:(R5)+".  Constants found in FEPequ.asm.
    moveY0,y:(R6)+

    clr     A
    move#>SHORT_JSR_INSTR,Y0

    move#SSI_RX_INSTR,A1
    movem A1,p:$000C
    movem A0,p:$000D

    move#ssi_rxe_handler,A1
    or      Y0,A
    movem A1,p:$000E
    movem A0,p:$000F

    move#ssi_tx_handler,A1
    or      Y0,A
    movem A1,p:$0010
    movem A0,p:$0011

    move#ssi_txe_handler,A1
    or      Y0,A
    movem A1,p:$0012
    movem A0,p:$0013

    move#sci_rx_handler,A1
    or      Y0,A
    movem A1,p:$0014
    movem A0,p:$0015

    move#sci_rxe_handler,A1
```

```
    or      Y0,A
    movem A1,p:$0016
    movem A0,p:$0017

    movey:-(R6),Y0
    rts


Fcheck_host
; Low-level non-blocking NeXT host read
    moveR0,y:(R6)+
    lua     (R6)+,R0
    movessh,y:(R6)+

    clr     A
    jclr#m_hrdf,x:<<m_hsr,_no_word

    moveN0,y:(R6)
    move#-3,N0
    moveR1,y:(R6)+
    movey:(R0+N0),R1
    move#>1,A1
    movep x:<<m_hrx,y:(R1)
    movey:(R6)-,R1
    movey:(R6),N0
_no_word
    move(R6)-
    movey:(R6)-,ssh
    movey:(R6),R0
    tst     A
    rts

Fget_host
; Low-level blocking NeXT host read
    jclr#m_hrdf,x:<<m_hsr,Fget_host
    clr     A
    movep x:<<m_hrx,A1
    tst     A
    rts

Fput_host
; Low-level blocking NeXT host write
    moveR0,y:(R6)+
    lua     (R6)+,R0
    movessh,y:(R6)+

    moveN0,y:(R6)+
    move#-3,N0
    moveA1,y:<y_a1_putaside_ph
    movey:(R0+N0),A
wait_for_host
    jclr#m_htde,x:<<m_hsr,wait_for_host
    movep A1,x:<<m_htx

    move(R6)-
    move  y:(R6)-,N0

    movey:(R6)-,ssh
    movey:(R6),R0

    movey:<y_a1_putaside_ph,A1

    rts

put555_host
; Quick block NeXT host write of the constant "555"
    moveA1,y:<y_a1_putaside_ph
    move#>555,A
wait_for_host_555
    jclr#m_htde,x:<<m_hsr,wait_for_host_555
    movep A1,x:<<m_htx

    movey:<y_a1_putaside_ph,A1

    rts

Ftxd_on
; Turn Port C TXD (LBox reset) on
    bset#1,x:<<m_pcd
    rts

Ftxd_off
; Turn Port C TXD (LBox reset) off
    bclr#1,x:<<m_pcd
    rts

Ftxd_tog
; Change the state of Port C TXD (LBox reset)
    bchg#1,x:<<m_pcd
    rts


Fsc1_on
; Turn Port C SC1 on
; NOTE: This is either ineffective or detrimental if SSI receive
; (Video data) is active!
    bset#4,x:<<m_pcd    ; Set SC1
    rts

Fsc1_off
; Turn Port C SC1 off
; NOTE: This is either ineffective or detrimental if SSI receive
; (Video data) is active!
    bclr#4,x:<<m_pcd    ; Clear SC1
    rts

    endsec
```

```
; EVENT- AND CPU-RELATED EQUATES DEFINED IN machspec.asm, IN *CLN DIRECTORY

; COORDINATE WITH FEPincl.h

; GLOSSARY: SSI_TX = LBox command transmit
;           SSI_TXQ = LBox command transmit queue
;           SCI_RX = LBox housekeeping receive
;           SCI_RXQ = LBox housekeeping queue
;           p-of-4 = package-of-four words (grouping of video data)
;           vbp = vidbuf_params
;           vidbuf = Video Buffer

; vidbuf allocated in X: space at the end of this file
VIDBUF_BOTTOM equ  $0
VIDBUF_LEN  equ     $2000
VIDBUF_TOP  equ     VIDBUF_BOTTOM+VIDBUF_LEN

; Lasagna Box queues allocated in Y: space at the end of this file
LBOX_QUEUE_BOTTOM equ  $1000
SSI_TXQ_SIZE  equ   1024
SCI_RXQ_SIZE  equ   64

; Bit location of "shift-to-avoid-negation-confusion" flag in y_flags_p1
SHIFT       equ     0

SHORT_JSR_INSTR  equ  $0D0000  ; jsr <[address]
SSI_RX_INSTR  equ   $085DAF ; movep x:<<m_rx,x:(R5)+


    org y:0
; ENSURE THESE VARIABLES END UP IN SHORT-ADDRESSABLE INTERNAL Y RAM (< $40) !

y_a1_putaside_ph  ds1   ; For saving A1 during put_host()

y_r1_putaside_ssi_tx ds 1 ; For saving R1 during SSI TX interrupt
y_m1_putaside_ssi_tx ds 1 ; For saving M1 during SSI TX interrupt

y_r1_putaside_sci_rx ds 1 ; For saving R1 during SCI RX interrupt
y_m1_putaside_sci_rx ds 1 ; For saving M1 during SCI RX interrupt

Fssi_txq_xfers  ds  1   ; # of words put in ssi_txq by main code
Fssi_tx_intrs   ds  1   ; # of words taken off ssi_txq by SSI TX intr
Fssi_tx_inptr   ds  1   ; Where main code puts words in ssi_txq
ssi_tx_outptr   ds  1   ; Where SSI TX intr reads words from ssi_txq
ssi_tx_mod          ds  1   ; M1 value to keep R1 inside ssi_txq

Fsci_rxq_xfers  ds  1   ; # of words taken off sci_rxq by main code
Fsci_rx_intrs   ds  1   ; # of words put in sci_rxq by SCI RX intr
Fsci_rx_outptr  ds  1   ; Where main code reads words from sci_rxq
sci_rx_inptr    ds  1   ; Where SCI RX intr puts words in sci_rxq
sci_rx_mod          ds  1   ; M1 value to keep R1 inside sci_rxq

y_unmod             dc  $FFFF ; Reset value for a Modulus register

Fasm_proclen    ds  1   ; # of p-of-4 proc_line() should process
y_flags_p1          ds  1   ; Flags used by proc_line() (like SHIFT)
y_vbp_ptr           ds  1   ; Pointer to vbp held by proc_line()
y_event_add         ds  1   ; Address of event candidate in the vidbuf

Fasm_thresh0    ds  1   ; Threshold for the 1st pixel in the p-of-4
Fasm_thresh1    ds  1   ; Threshold for the 2nd pixel in the p-of-4
Fasm_thresh2    ds  1   ; Threshold for the 3rd pixel in the p-of-4
Fasm_thresh3    ds  1   ; Threshold for the 4th pixel in the p-of-4
Fasm_thresh4    ds  1   ; Threshold for the 5th pixel in the p-of-4
Fasm_thresh5    ds  1   ; Threshold for the 6th pixel in the p-of-4
Fasm_thresh6    ds  1   ; Threshold for the 7th pixel in the p-of-4
Fasm_thresh7    ds  1   ; Threshold for the 8th pixel in the p-of-4

y_up1               ds  1   ; # of vidbuf words to "up" neighbor
y_down1             ds  1   ; # of vidbuf words to "down" neighbor
y_right1            dc  4   ; # of vidbuf words to "right" neighbor
y_left1             dc  -4  ; # of vidbuf words to "left" neighbor

y_plus1             dc  1   ; Quick-access constants
y_plus2             dc  2

y_proc_ptr_mod  dc  (VIDBUF_LEN-1) ; Mn val to keep Rn inside vidbuf

y_low12_mask    dc  $000FFF ; Mask out low pixel
y_high12_mask   dc  $7FF800 ; Mask out high pixel (right-shifted by 1)
y_high16_mask   dc  $7FFF80 ; Mask out 16bit pixel (right-shifted by 1)


; Allocate the vidbuf in X: space

    org x:VIDBUF_BOTTOM
xbuf_vidbuf dsm     VIDBUF_LEN


; Allocate the LBox queues after global space & before stack

    org y:LBOX_QUEUE_BOTTOM
Fssi_txq    dsm     SSI_TXQ_SIZE
Fssi_txq_end
Fsci_rxq    dsm     SCI_RXQ_SIZE
Fsci_rxq_end
```

```
;;!cc

; Port C interrupt handlers

    section interrupts

    global sci_rx_handler,sci_rxe_handler
    global ssi_tx_handler,ssi_txe_handler
    global ssi_rxe_handler

    org p:$40

sci_rx_handler  ; 15 words
    moveR1,y:<y_r1_putaside_sci_rx ; Save registers we'll be using
    moveM1,y:<y_m1_putaside_sci_rx
    movey:<sci_rx_inptr,R1  ; Get buffer receive-pointer
    movey:<sci_rx_mod,M1    ; and circular-buffer modulus
    movep x:<<m_srxl,y:(R1)+   ; Save the received word in buffer
    moveR1,y:<sci_rx_inptr  ; Update buffer receive-pointer
    movey:<Fsci_rx_intrs,R1   ; Get interrupt counter
    movey:<y_unmod,M1       ; Eliminate modulus
    move(R1)+                   ; Increment interrupt counter
    moveR1,y:<Fsci_rx_intrs   ; Update interrupt counter
    movey:<y_r1_putaside_sci_rx,R1 ; Restore registers
    movey:<y_m1_putaside_sci_rx,M1
    rti

sci_rxe_handler  ; 3 words
    movep x:<<m_ssr,y:Fsci_rx_exception ; Record what the problem is
    jmp     <sci_rx_handler     ; Read the word as though normal

ssi_tx_handler  ; 15 words
    moveR1,y:<y_r1_putaside_ssi_tx ; Save registers we'll be using
    moveM1,y:<y_m1_putaside_ssi_tx
    movey:<ssi_tx_outptr,R1   ; Get command-out pointer
    movey:<ssi_tx_mod,M1    ; and circular-buffer modulus
    movep y:(R1)+,x:<<m_tx  ; Transmit the next command word
    moveR1,y:<ssi_tx_outptr   ; Update command-out pointer
    movey:<Fssi_tx_intrs,R1   ; Get interrupt counter
    movey:<y_unmod,M1       ; Eliminate modulus
    move(R1)+                   ; Increment interrupt counter
    moveR1,y:<Fssi_tx_intrs   ; Update interrupt counter
    movey:<y_r1_putaside_ssi_tx,R1 ; Restore registers
    movey:<y_m1_putaside_ssi_tx,M1
    rti

ssi_txe_handler  ; 3 words
    movep x:<<m_sr,y:Fssi_tx_exception ; Record what the problem is
    jmp     <ssi_tx_handler     ; Send the word as though normal

ssi_rxe_handler  ; 3 words
    movep x:<<m_sr,y:Fssi_rx_exception ; Record what the problem is
    movep x:<<m_rx,x:(R5)+  ; Read the word as though normal
    rti

    endsec
```

```
;;!cc
; Quick routine to find local-maxima events

    section proc_line

    org p:$A0
    global Fproc_line

Fproc_line
    moveR0,y:(R6)+
    lua     (R6)+,R0
    movessh,y:(R6)+

    clr     A               #-3,N0
; Save X, Y, B, (RNM)3
    moveB2,y:(R6)+
    moveB1,y:(R6)+
    moveB0,y:(R6)+
    moveY1,y:(R6)+
    moveY0,y:(R6)+
    moveX1,y:(R6)+
    moveX0,y:(R6)+
    moveM3,y:(R6)+
    moveN3,y:(R6)+
    moveR3,y:(R6)+

    move(R0)+N0
    movey:(R0)-,R3              ; Start processing at R3
    movey:(R0)-,A1              ; Mode: check if 12- or 16-bit pixels
    movey:(R0)-,B1              ; Words-per-line
    moveB1,y:<y_up1
    movey:(R0)-,B1              ; -(Words-per-line)
    moveB1,y:<y_down1
    movey:(R0),B1               ; Pointer to vidbuf_params, for
    moveB1,y:<y_vbp_ptr      ; use when passing to record_event()

    movey:<y_proc_ptr_mod,M3  ; Set up R3 circularity

; Process line (check if Mode 3 or not)
    move#>3,X1
    cmp     X1,A
    jeq     <_mode_12_bit
    jsr     <proc_16_bit
    jmp     <_end
_mode_12_bit
    jsr     <proc_12_bit
_end
; Restore X, Y, B, (RNM)3
    move(R6)-
    movey:(R6)-,R3
    movey:(R6)-,N3
    movey:(R6)-,M3
    movey:(R6)-,X0
    movey:(R6)-,X1
    movey:(R6)-,Y0
    movey:(R6)-,Y1
    movey:(R6)-,B0
    movey:(R6)-,B1
    movey:(R6)-,B2

    movey:(R6)-,ssh
    movey:(R6),R0
    rts


proc_16_bit
; Set mask for seek_event routines
    movey:<y_high16_mask,Y0
; Note that when ge_thresh_16 is called, R3 will be pointing to
; TWO words past the word which triggered the call
    movey:<y_plus2,N3
; Note that the pixel data includes the MSB, and must therefore
; be shifted before calculation or comparison to avoid negative-
; value errors
    bset#SHIFT,y:<y_flags_pl

; Preset A1 to the first word in the first package-of-four;
; R3 points to the second word in the first package-of-four
    movex:(R3)+,A1
; Shift A one bit to the right to clear the MSB; set X0 to the
; threshold to which the first word in a package-of-four should
; be compared
    lsr     A               y:<Fasm_thresh0,X0

; Loop num_cols-2 times, each time testing four words
    do      y:<Fasm_proclen,_endloop
; Compare the first word in the package-of-four to its threshold;
; Set A1 to the second word in the package-of-four; R3 points to
; the third word in the package-of-four
    cmp     X0,A        x:(R3)+,A1
; If the first word in the package-of-four meets or exceeds threshold,
; check if it is a local maximum
    jsge<ge_thresh_16

; Shift A one bit to the right to clear the MSB; set X0 to the
; threshold to which the second word in a package-of-four should
; be compared
    lsr     A               y:<Fasm_thresh1,X0
; Compare the second word in the package-of-four to its threshold;
; Set A1 to the third word in the package-of-four; R3 points to
; the fourth word in the package-of-four
    cmp     X0,A        x:(R3)+,A1
; If the second word in the package-of-four meets or exceeds
; threshold, check if it is a local maximum
    jsge<ge_thresh_16

; Shift A one bit to the right to clear the MSB; set X0 to the
; threshold to which the third word in a package-of-four should
; be compared
    lsr     A               y:<Fasm_thresh2,X0
; Compare the third word in the package-of-four to its threshold;
; Set A1 to the fourth word in the package-of-four; R3 points to
; the first word in the next package-of-four
    cmp     X0,A        x:(R3)+,A1
; If the third word in the package-of-four meets or exceeds threshold,
; check if it is a local maximum
    jsge<ge_thresh_16

; Shift A one bit to the right to clear the MSB; set X0 to the
; threshold to which the first word in a package-of-four should
; be compared
    lsr     A               y:<Fasm_thresh3,X0
; Compare the fourth word in the package-of-four to its threshold;
; Set A1 to the first word in the next package-of-four; R3 points to
; the second word in the next package-of-four
    cmp     X0,A        x:(R3)+,A1
; If the fourth word in the package-of-four meets or exceeds
; threshold, check if it is a local maximum
    jsge<ge_thresh_16

; Shift A one bit to the right to clear the MSB; set X0 to the
; threshold to which the first word in a package-of-four should
; be compared
    lsr     A               y:<Fasm_thresh0,X0
```

```
_endloop
    rts

proc_12_bit
; Set mask for testing of low-bits pixel, in this loop & in seek_event
    movey:<y_low12_mask,Y0
; Note that when ge_thresh_16 is called, R3 will be pointing to
; ONE word past the word which triggered the call
    movey:<y_plus1,N3

; Preset X0 to the threshold to which the hi pixel in the first
; word in a package-of-four should be compared
    movey:<Fasm_thresh0,X0
; Preset A1 to first word in the first package-of-four; R3 still
; points to the first word in the package-of-four
    movex:(R3),A1
; Shift A one bit to the right to clear the MSB; set X1 to the
; threshold to which the lo pixel in the first word in a package-
; of-four should be compared
    lsr     A               y:<Fasm_thresh1,X1

; Loop num_cols-2 times, each time testing four words (8 pixels)
    do      y:<Fasm_proclen,_endloop
; Compare the first word in the package-of-four to the hi-pixel
; threshold; set A1 to the first word in the package of four (again);
; R3 points to the second word in the package-of-four
    cmp     X0,A        x:(R3)+,A1
; If the hi pixel in the first word in the package-of-four meets or
; exceeds threshold, check if it is a local maximum
    jsge<ge_thresh_hi
; Mask A1 so only the lo pixel remains; set X0 to the threshold to
; which the hi pixel in the second word in a package-of-four should
; be compared
    and     Y0,A        y:<Fasm_thresh2,X0
; Compare the lo pixel in the first word in the package-of-four to
; its threshold; set A1 to the second word in the package-of-four;
; R3 still points to the second word in the package-of-four
    cmp     X1,A        x:(R3),A1
; If the lo pixel in the first word in the package-of-four meets or
; exceeds threshold, check if it is a local maximum
    jsge<ge_thresh_lo

; Shift A one bit to the right to clear the MSB; set X1 to the
; threshold to which the lo pixel in the second word in a package-
; of-four should be compared
    lsr     A               y:<Fasm_thresh3,X1
; Compare the hi pixel in the second word in the package-of-four to
; its threshold; set A1 to the second word in the package-of-four
; (again); R3 points to the third word in the package-of-four
    cmp     X0,A        x:(R3)+,A1
; If the hi pixel in the second word in the package-of-four meets or
; exceeds threshold, check if it is a local maximum
    jsge<ge_thresh_hi
; Mask A1 so only the lo pixel remains; set X0 to the threshold to
; which the hi pixel in the third word in a package-of-four should
; be compared
    and     Y0,A        y:<Fasm_thresh4,X0
; Compare the lo pixel in the second word in the package-of-four to
; its threshold; set A1 to the third word in the package-of-four;
; R3 still points to the third word in the package-of-four
    cmp     X1,A        x:(R3),A1
; If the lo pixel in the second word in the package-of-four meets or
; exceeds threshold, check if it is a local maximum
    jsge<ge_thresh_lo

; Shift A one bit to the right to clear the MSB; set X1 to the
; threshold to which the lo pixel in the third word in a package-
; of-four should be compared
    lsr     A               y:<Fasm_thresh5,X1
; Compare the hi pixel in the third word in the package-of-four to
; its threshold; set A1 to the third word in the package-of-four
; (again); R3 points to the fourth word in the package-of-four
    cmp     X0,A        x:(R3)+,A1
; If the hi pixel in the third word in the package-of-four meets or
; exceeds threshold, check if it is a local maximum
    jsge<ge_thresh_hi
; Mask A1 so only the lo pixel remains; set X0 to the threshold to
; which the hi pixel in the fourth word in a package-of-four should
; be compared
    and     Y0,A        y:<Fasm_thresh6,X0
; Compare the lo pixel in the third word in the package-of-four to
; its threshold; set A1 to the fourth word in the package-of-four;
; R3 still points to the fourth word in the package-of-four
    cmp     X1,A        x:(R3),A1
; If the lo pixel in the third word in the package-of-four meets or
; exceeds threshold, check if it is a local maximum
    jsge<ge_thresh_lo

; Shift A one bit to the right to clear the MSB; set X1 to the
; threshold to which the lo pixel in the fourth word in a package-
; of-four should be compared
    lsr     A               y:<Fasm_thresh7,X1
; Compare the hi pixel in the fourth word in the package-of-four to
; its threshold; set A1 to the fourth word in the package-of-four
; (again); R3 points to the first word in the next package-of-four
    cmp     X0,A        x:(R3)+,A1
; If the hi pixel in the fourth word in the package-of-four meets or
; exceeds threshold, check if it is a local maximum
    jsge<ge_thresh_hi
; Mask A1 so only the lo pixel remains; set X0 to the threshold to
; which the hi pixel in the first word in a package-of-four should
; be compared
    and     Y0,A        y:<Fasm_thresh0,X0
; Compare the lo pixel in the fourth word in the package-of-four to
; its threshold; set A1 to the first word in the next package-of-four;
; R3 still points to the first word in the next package-of-four
    cmp     X1,A        x:(R3),A1
; If the lo pixel in the fourth word in the package-of-four meets or
; exceeds threshold, check if it is a local maximum
    jsge<ge_thresh_lo

; Shift A one bit to the right to clear the MSB; set X1 to the
; threshold to which the lo pixel in the first word in a package-
; of-four should be compared
    lsr     A               y:<Fasm_thresh1,X1
_endloop
    rts


ge_thresh_hi ; 8 + 210+ = 218+
; Reset Y0 to mask high-bits pixel
    movey:<y_high12_mask,Y0
; Note that shifting-to-avoid-negation-confusion is to be done
    bset#SHIFT,y:<y_flags_pl
    jsr     <seek_event
; Restore Y0
    movey:<y_low12_mask,Y0
    rts

ge_thresh_lo ; 4 + 210+ = 214+
; No shifting necessary for lower 12 bits
    bclr#SHIFT,y:<y_flags_pl
    jmp     <seek_event
```

```
ge_thresh_16
seek_event

; Compares pixel "X" to its eight nearest neighbors
;
;   UL  U   UR
;    L  X   R
;   DL  D   DR
; Note: "Left" and "Right" reversed for slices B & D;
;   but algorithm doesn't really care. THOUGH SCIENCE MIGHT!

    clr     A               A1,y:(R6)+
    moveR3,y:(R6)+
; R3 -> X
    move(R3)-N3
    moveR3,y:<y_event_add
; A = X
    movex:(R3),A1
    jclr#SHIFT,y:<y_flags_pl,_noshift_X
    lsr     A
_noshift_X
    and     Y0,A        y:<y_right1,N3

; R3 -> R; X ? R
    jsr     <compare
; Check X > R
    jle     <higher_exists
; R3 -> UR; X ? UR
    movey:<y_up1,N3
    jsr     <compare
; Check X > UR
    jle     <higher_exists
; R3 -> U; X ? U
    movey:<y_left1,N3
    jsr     <compare
; Check X > U
    jle     <higher_exists
; R3 -> UL; X ? UL
    jsr     <compare
; Check X > UL
    jle     <higher_exists
; R3 -> L; X ? L
    movey:<y_down1,N3
    jsr     <compare
; Check X >= L
    jlt     <higher_exists
; R3 -> DL; X ? DL
    jsr     <compare
; Check X >= DL
    jlt     <higher_exists
; R3 -> D; X ? D
    movey:<y_right1,N3
    jsr     <compare
; Check X >= D
    jlt     <higher_exists
; R3 -> DR; X ? DR
    jsr     <compare
; Check X >= DR
    jlt     <higher_exists

; Found Event!
; Call record_event(int *address, int value, int highpixel,
; vidbuf_param_struct *vbpnew);

    movey:<y_vbp_ptr,B1
    clr     B               B1,y:(R6)+
    jclr#SHIFT,y:<y_flags_pl,_noshift
    move#>1,B1
_noshift
    moveB1,y:(R6)+
    moveA1,y:(R6)+
    movey:<y_event_add,B1
    moveB1,y:(R6)+
    jsr     Frecord_event
    move(R6)-
    move(R6)-
    move(R6)-
    move(R6)-
higher_exists
    clr     A               (R6)-
    movey:(R6)-,R3
    movey:(R6),A1
    rts

compare ; 9
; Changes R3 by N3 words; compares the word found here to the pixel
; in question (shifting to avoid negation confusion if necessary);
; returns with the status bits set to the result of this comparison
    clr     B               (R3)+N3
    movex:(R3),B1
    jclr#SHIFT,y:<y_flags_pl,_noshift
    lsr     B
_noshift
    and     Y0,B
    cmp     B,A
    rts


    endsec
```

```
       opt so,xr
       page 132,66,3,3

;CRTRZP.ASM i920622 jek
;  RZP CRT0 replacement source -- Configuration for RZP DSP
;930615 org adjusted per size of ROM-booted area.
; Modified 1/94 by S.A.McD. to start stack above Y-space buffers

; From: $Id: crt056.asm,v 1.12 91/06/14 14:30:36 jeff Exp $
; Modifications to this file Copyright (C) 1992,1993 AeroAstro, Inc.

; Memory model based on a jammed -define for the symbol "mm" at command line.
; Example:
; asm56000 -bcrtRZPy.cln -lcrtRZPy.lst   -dmm y   crtRZP
;           dest=y cln^   y list file^  define^   ^same source, every model
; See MAKERZP.BAT

       include "ioequ.asm"
;GNU global declarations (not currently global)
;  global FRTC_Count
;  global FRF_Count
;  global FGPS_Count
;  global FDGlue_Stat
;  global FDGlue_Ctrl
; for these equ's
       include "dglue.a"

;---------------------------------------
; Externs (init driver calls ONLY)
; (GNU/asm56k do not require explicit
; EXTERN/EXTRN-type declaration, globals
; are all that need be declared, EXTRN is
; automatic)
;---------------------------------------
; Finitialize_dma_link
; Finit_RTC_driver
;---------------------------------------


;===========================================
TOP_OF_MEMORY  equ     $7FFF    ;last RAM alloc'able word in Y:/X: data RAM

STACK_START    equ     $1440    ;after LB Queues

; This value represents the 32k X & Y: internal plus external SRAM on the RZP
; P top of SRAM memory is also 32k.
;  (all addresses below are in hex)

;DSP Internal Program SRAM @ P:0000 to P:01FF  24 bits
;    External Program SRAM @ P:0200 to P:7FFF  24 bits
;   (w)External Boot EEROM @ P:8000 to P:FFFF   8 bits, WAITs, EEWE protected
;   (r)External Boot EEROM @ P:C000 to P:DFFF   8 bits, WAITs
; DSP Internal Y:Data SRAM @ Y:0000 to Y:00FF  24 bits
;     External Y:Data SRAM @ Y:0100 to Y:7FFF  24 bits
;          Frame buffer Y8 @ Y:8000 to Y:8FFF  24 bits
;          Frame buffer Y9 @ Y:8000 to Y:9FFF  24 bits
;          Frame buffer YA @ Y:8000 to Y:AFFF  24 bits
;          Frame buffer YB @ Y:8000 to Y:BFFF  24 bits
;                No memory @ Y:C000 to Y:EEFF
;   Frame buffer page regs @ Y:EF00 to Y:EF07  12 bits
;                No memory @ Y:EF08 to Y:FFBF
;       DGLUE Peripherals @ Y:FFC0 to Y:FFC3  varies (Fxx0-Fxx3)
;                No memory @ Y:FFC4 to Y:FFFF
; DSP Internal X:Data SRAM @ X:0000 to X:00FF  24 bits
;     External X:Data SRAM @ X:0100 to X:7FFF  24 bits
;          Frame buffer X8 @ X:8000 to X:8FFF  24 bits
;          Frame buffer X9 @ X:8000 to X:9FFF  24 bits
;          Frame buffer XA @ X:8000 to X:AFFF  24 bits
;          Frame buffer XB @ X:8000 to X:BFFF  24 bits
;                No memory @ X:C000 to X:FFBF
; DSP Internal Peripherals @ X:FFC0 to X:FFFF  varies

;-----------------------------------------------
; Block access to area used by bootstrap ROM code
;-----------------------------------------------

       section bootstrapper       ;Block GNU access
       org     p:$0               ; to the ROM booted area
       ds      2731               ; block out use of first 8k *BYTES*
       endsec                     ; (8192/3)=2730.7 words, last word is partial
                                  ; and is not stuffed by rom burner

CRT0ORG  equ   $0AAB              ;CRT0 + compiled code starts after
                                  ;ROM-booted code, which fills 0..AAA
                                  ;See ROMBOOT.ASM
;;#### HETE?
;;       section BlockXInternal  ;block internal DSP RAM in X:
;;       org     x:$0            ; to force GNU to use the more
;;       ds      $100            ; 'robust' external RAM on the
;;       endsec                  ; RZP.
;;       section BlockYInternal  ;same note - BOTH must be blocked,
;;       org     y:$0            ;regardles of "mm" model chosen
;;       ds      $100
;;       endsec


;===========================================
;  System variable area (GNU)
;===========================================

       section crt0

       org     mm:              ; per GNU model

; The following variables are used for dynamic memory allocation
; __stack_safty: Since dynamic memory and the stack grow towards each other
; This constant tells brk and sbrk what the minimum amount of space should
; be left between the top of stack during the brk or sbrk call and the end
; of any allocated memory.
; __mem_limit: a constant telling brk and sbrk where the end of available
; memory is.
; __break: pointer to the next block of memory that can be allocated
; The heap may be moved by changing the initial value of __break.
; This is the base of the heap.
; __y_size: the base of dynamic memory.
; errno. error type: set by some libraries
; __max_signal the last possible signal (interrupt #).


                global Ferrno
Ferrno          ds 1       ;init=0
                global F__stack_safety
F__stack_safety ds 1       ;init=1024      ;512 on the hack board
                global F__mem_limit
F__mem_limit    ds 1       ;init=TOP_OF_MEMORY
                global F__break
F__break        ds 1       ;init=TOP_OF_MEMORY
                global F__y_size
F__y_size       ds 1       ;init=DSIZE normally, modified for FEP
                global F__max_signal
F__max_signal   ds 1       ;init=$3e       ; what the quint monitor.


;===========================================
```

```
;       GNU startup
;=========================================

        org     p:CRT0ORG

        global  F__start
F__start

; wait state programming is done twice in case app is
; executed initially from here.

        move    #0,SP                           ; Clear hardware stack pointer
        movep   #0,X:<<$FFFE                    ; clear auto waits (BCR)
        ori     #$80,omr                        ; activate DSP wait state recog.
        movep   #WAITENA_IRQFST,Y:<<FDGlue_Ctrl   ; and in DGLUE

;-----------------------------------------------
; GNU pointer initialization code
;-----------------------------------------------

        clr     a
        move    a,mm:Ferrno
        move    #TOP_OF_MEMORY,a
        move    a,mm:F__mem_limit
        move    a,mm:F__break
        move    #STACK_START,a
        move    a,mm:F__y_size
        move    #>$3E,a
        move    a,mm:F__max_signal
        move    #1024,A
        move    a,mm:F__stack_safety

;       To change the base of the stack, change the
;       value loaded into the stack pointer here.

        move    mm:F__y_size,r6         ; initialize the stack pointer
        move    #$0,r0                  ; funny value to terminate a backtrace.

        and     #$F0,mr                 ; clear scaling mode bits i1,i0
                                        ; and set current IPL = 0 (AFTER setting R6)
        and     #$BF,ccr                ; init condition code register

;-----------------------------------------------
;       Call driver inits here
;-----------------------------------------------

        jsr     Finitialize_dma_link    ;Link / Init IPP driver (extern)
        jsr     Finit_RTC_driver        ;Link / Init RTC driver (extern)

        jsr     Fmain                   ; run user program


;===========================================
; This should never be encountered - indicates
; end of main() seen. #### Should invoke self-nuke?
;===========================================
        global  F__crt0_end
F__crt0_end
        move    #>-1,x0         ; set signal that we're done, and
        swi                     ; return to the monitor.
        stop                    ; just in case.

;===========================================
        global  F__printf_end   ; avoid loading humungous, useless hosted printf
;===========================================
F__printf_end                   ; dummy printf catch routine.
        swi                     ; invoke monitor
        rts                     ; return to execution after the 'printing'.

        endsec                  ;section crt0

;===========================================
;       section crt0__fp_shift
;===========================================
;
;       org     mm:
;
; floating point table setup   --NOT USED in this system, no floating point
;
;       global  F__fp_shift
;F__fp_shift    dc      $800000,$c00000,$e00000,$f00000,$f80000,$fc0000
;               dc      $fe0000,$ff0000,$ff8000,$ffc000,$ffe000,$fff000
;               dc      $fff800,$fffc00,$fffe00,$ffff00,$ffff80,$ffffc0
;               dc      $ffffe0,$fffff0,$fffff8,$fffffc,$fffffe
;
;       endsec
;-------------------------------------------




;===========================================
; Following section blocks the GNU linker
; from using reserved address spaces as RAM,
; and defines RZP memory map/peripheral space
;===========================================

        section RZP_Addresses

;------------- P: Boot EPROM area

                org     p:$8000
Boot_EEROM      ds      $8000       ;EEWE must be enabled for write

;------------- X: frame buffers

                org     x:$8000

                global  FFrame_X8
FFrame_X8       ds      $1000
                global  FFrame_X9
FFrame_X9       ds      $1000
                global  FFrame_XA
FFrame_XA       ds      $1000
                global  FFrame_XB
FFrame_XB       ds      $1000

;------------- Non-populated X area

                org     x:$C000
                ds      ($FFC0-$C000)   ;block unused area

;------------- Y: frame buffers

                org     y:$8000
                global  FFrame_Y8
FFrame_Y8       ds      $1000
                global  FFrame_Y9
FFrame_Y9       ds      $1000
                global  FFrame_YA
FFrame_YA       ds      $1000
```

```
              global  FFrame_YB
FFrame_YB     ds      $1000

;------------- Non-populated Y area 1

              org     y:$C000
              ds      ($EF00-$C000)   ;block unused area

;------------- Frame buffer registers

              org     y:$EF00
              global  FFrameReg_X8
FFrameReg_X8  ds      1
              global  FFrameReg_X9
FFrameReg_X9  ds      1
              global  FFrameReg_XA
FFrameReg_XA  ds      1
              global  FFrameReg_XB
FFrameReg_XB  ds      1
              global  FFrameReg_Y8
FFrameReg_Y8  ds      1
              global  FFrameReg_Y9
FFrameReg_Y9  ds      1
              global  FFrameReg_YA
FFrameReg_YA  ds      1
              global  FFrameReg_YB
FFrameReg_YB  ds      1

;------------- Non-populated Y area 2

              org     y:$EF08
              ds      ($FFC0-$EF08)   ;block unused area

;------------- DGLUE registers
; see DGLUE.A
;              org     y:$FFC0 ; FFCx so we can use short i/o addressing modes
;DGLUE Registers 0-3
;FRTC_Count     ds      1    ;12-bit r/o
;FRF_Count      ds      1    ;12-bit r/o
;FGPS_Count     ds      1    ;12-bit r/o
;FDGlue_Stat                 ;bits 0..5, 7..11 r/o (addr shared with ctrl reg:)
;FDGlue_Ctrl    ds      1    ;bits 0..6 w/o

;------------- Non-populated Y area 3

              org     y:$FFC4
              ds      ($FFFF-$FFC4)+1   ;block unused area

       endsec

       end     F__start
```

```
;
; This program originally available on the Motorola DSP bulletin board.
; It is provided under a DISCLAMER OF WARRANTY available from
; Motorola DSP Operation, 6501 Wm. Cannon Drive W., Austin, Tx., 78735.
;
; Motorola Standard I/O Equates (lower case).
;
; Last Update 25 Aug 87   Version 1.1  (fixed m_of)
;
;********************************************************************
;
;       EQUATES for DSP56000 I/O registers and ports
;
;********************************************************************

ioequlc ident   1,0

;--------------------------------------------------------------------
;
;       EQUATES for I/O Port Programming
;
;--------------------------------------------------------------------

;       Register Addresses

m_bcr   EQU     $FFFE           ; Port A Bus Control Register
m_pbc   EQU     $FFE0           ; Port B Control Register
m_pbddr EQU     $FFE2           ; Port B Data Direction Register
m_pbd   EQU     $FFE4           ; Port B Data Register
m_pcc   EQU     $FFE1           ; Port C Control Register
m_pcddr EQU     $FFE3           ; Port C Data Direction Register
m_pcd   EQU     $FFE5           ; Port C Data Register


;--------------------------------------------------------------------
;
;       EQUATES for Host Interface
;
;--------------------------------------------------------------------

;       Register Addresses

m_hcr   EQU     $FFE8           ; Host Control Register
m_hsr   EQU     $FFE9           ; Host Status Register
m_hrx   EQU     $FFEB           ; Host Receive Data Register
m_htx   EQU     $FFEB           ; Host Transmit Data Register

;       Host Control Register Bit Flags

m_hrie  EQU     0               ; Host Receive Interrupt Enable
m_htie  EQU     1               ; Host Transmit Interrupt Enable
m_hcie  EQU     2               ; Host Command Interrupt Enable
m_hf2   EQU     3               ; Host Flag 2
m_hf3   EQU     4               ; Host Flag 3

;       Host Status Register Bit Flags

m_hrdf  EQU     0               ; Host Receive Data Full
m_htde  EQU     1               ; Host Transmit Data Empty
m_hcp   EQU     2               ; Host Command Pending
m_hf    EQU     $18             ; Host Flag Mask
m_hf0   EQU     3               ; Host Flag 0
m_hf1   EQU     4               ; Host Flag 1
m_dma   EQU     7               ; DMA Status

;--------------------------------------------------------------------
;
;       EQUATES for Serial Communications Interface (SCI)
;
;--------------------------------------------------------------------

;       Register Addresses

m_srxl  EQU     $FFF4           ; SCI Receive Data Register (low)
m_srxm  EQU     $FFF5           ; SCI Receive Data Register (middle)
m_srxh  EQU     $FFF6           ; SCI Receive Data Register (high)
m_stxl  EQU     $FFF4           ; SCI Transmit Data Register (low)
m_stxm  EQU     $FFF5           ; SCI Transmit Data Register (middle)
m_stxh  EQU     $FFF6           ; SCI Transmit Data Register (high)
m_stxa  EQU     $FFF3           ; SCI Transmit Data Address Register
m_scr   EQU     $FFF0           ; SCI Control Register
m_ssr   EQU     $FFF1           ; SCI Status Register
m_sccr  EQU     $FFF2           ; SCI Clock Control Register

;       SCI Control Register Bit Flags

m_wds   EQU     $3              ; Word Select Mask
m_wds0  EQU     0               ; Word Select 0
m_wds1  EQU     1               ; Word Select 1
m_wds2  EQU     2               ; Word Select 2
m_sbk   EQU     4               ; Send Break
m_wake  EQU     5               ; Wake-up Mode Select
m_rwi   EQU     6               ; Receiver Wake-up Enable
m_woms  EQU     7               ; Wired-OR Mode Select
m_re    EQU     8               ; Receiver Enable
m_te    EQU     9               ; Transmitter Enable
m_ilie  EQU     10              ; Idle Line Interrupt Enable
m_rie   EQU     11              ; Receive Interrupt Enable
m_tie   EQU     12              ; Transmit Interrupt Enable
m_tmie  EQU     13              ; Timer Interrupt Enable

;       SCI Status Register Bit Flags

m_trne  EQU     0               ; Transmitter Empty
m_tdre  EQU     1               ; Transmit Data Register Empty
m_rdrf  EQU     2               ; Receive Data Register Full
m_idle  EQU     3               ; Idle Line
m_or    EQU     4               ; Overrun Error
m_pe    EQU     5               ; Parity Error
m_fe    EQU     6               ; Framing Error
m_r8    EQU     7               ; Received Bit 8

;       SCI Clock Control Register Bit Flags

m_cd    EQU     $FFF            ; Clock Divider Mask
m_cod   EQU     12              ; Clock Out Divider
m_scp   EQU     13              ; Clock Prescaler
m_rcm   EQU     14              ; Receive Clock Source
m_tcm   EQU     15              ; Transmit Clock Source

;--------------------------------------------------------------------
;
;       EQUATES for Synchronous Serial Interface (SSI)
;
;--------------------------------------------------------------------

;       Register Addresses

m_rx    EQU     $FFEF           ; Serial Receive Data Register
m_tx    EQU     $FFEF           ; Serial Transmit Data Register
m_cra   EQU     $FFEC           ; SSI Control Register A
m_crb   EQU     $FFED           ; SSI Control Register B
m_sr    EQU     $FFEE           ; SSI Status Register
m_tsr   EQU     $FFEE           ; SSI Time Slot Register
```

```
;       SSI Control Register A Bit Flags

m_pm    EQU     $FF             ; Prescale Modulus Select Mask
m_dc    EQU     $1F00           ; Frame Rate Divider Control Mask
m_wl    EQU     $6000           ; Word Length Control Mask
m_wl0   EQU     13              ; Word Length Control 0
m_wl1   EQU     14              ; Word Length Control 1
m_psr   EQU     15              ; Prescaler Range

;       SSI Control Register B Bit Flags

m_of    EQU     $3              ; Serial Output Flag Mask
m_of0   EQU     0               ; Serial Output Flag 0
m_of1   EQU     1               ; Serial Output Flag 1
m_scd   EQU     $1C             ; Serial Control Direction Mask
m_scd0  EQU     2               ; Serial Control 0 Direction
m_scd1  EQU     3               ; Serial Control 1 Direction
m_scd2  EQU     4               ; Serial Control 2 Direction
m_sckd  EQU     5               ; Clock Source Direction
m_fsl   EQU     8               ; Frame Sync Length
m_syn   EQU     9               ; Sync/Async Control
m_gck   EQU     10              ; Gated Clock Control
m_mod   EQU     11              ; Mode Select
m_ste   EQU     12              ; SSI Transmit Enable
m_sre   EQU     13              ; SSI Receive Enable
m_stie  EQU     14              ; SSI Transmit Interrupt Enable
m_srie  EQU     15              ; SSI Receive Interrupt Enable

;       SSI Status Register Bit Flags

m_if    EQU     $2              ; Serial Input Flag Mask
m_if0   EQU     0               ; Serial Input Flag 0
m_if1   EQU     1               ; Serial Input Flag 1
m_tfs   EQU     2               ; Transmit Frame Sync
m_rfs   EQU     3               ; Receive Frame Sync
m_tue   EQU     4               ; Transmitter Underrun Error
m_roe   EQU     5               ; Receiver Overrun Error
m_tde   EQU     6               ; Transmit Data Register Empty
m_rdf   EQU     7               ; Receive Data Register Full

;---------------------------------------------------------------------
;
;       EQUATES for Exception Processing
;
;---------------------------------------------------------------------

;       Register Addresses

m_ipr   EQU     $FFFF           ; Interrupt Priority Register

;       Interrupt Priority Register Bit Flags

m_ial   EQU     $7              ; IRQA Mode Mask
m_ial0  EQU     0               ; IRQA Mode Interrupt Priority Level (low)
m_ial1  EQU     1               ; IRQA Mode Interrupt Priority Level (high)
m_ial2  EQU     2               ; IRQA Mode Trigger Mode
m_ibl   EQU     $38             ; IRQB Mode Mask
m_ibl0  EQU     3               ; IRQB Mode Interrupt Priority Level (low)
m_ibl1  EQU     4               ; IRQB Mode Interrupt Priority Level (high)
m_ibl2  EQU     5               ; IRQB Mode Trigger Mode
m_hpl   EQU     $C00            ; Host Interrupt Priority Level Mask
m_hpl0  EQU     10              ; Host Interrupt Priority Level Mask (low)
m_hpl1  EQU     11              ; Host Interrupt Priority Level Mask (high)
m_ssl   EQU     $3000           ; SSI Interrupt Priority Level Mask
m_ssl0  EQU     12              ; SSI Interrupt Priority Level Mask (low)
m_ssl1  EQU     13              ; SSI Interrupt Priority Level Mask (high)
m_scl   EQU     $C000           ; SCI Interrupt Priority Level Mask
m_scl0  EQU     14              ; SCI Interrupt Priority Level Mask (low)
m_scl1  EQU     15              ; SCI Interrupt Priority Level Mask (high)
```

```
; COORDINATE WITH FEPincl.h

EVENTS_BOTTOM equ  $2000
MAXNUMEVENTS equ  12000

CPU_Mhz        equ     20.0    ;for HETE CPU

Mbps           equ     0.5     ;SSI clock rate, megabits per second
;clock division constant for SSI
divider        equ     @CVI(0.25*CPU_Mhz/Mbps+0.5)-1


; Allocate event-recording space after the vidbuf

    org x:EVENTS_BOTTOM
eventspace  ds      (2*MAXNUMEVENTS)
```

```
; crt file for FEP loaded into NeXT DSP

    opt so,xr
    page 132,66,3,3

TOP_OF_MEMORY equ $6000   ; 32K expanded memory
                          ; Leave room for VidBuf at the top

    ;   This section should be loaded at P:0 that way reset will
    ;   cause a jump to main.

;   The reset section is located at p:0 in the mapfile. This section is
;   intended to hold system reset code. This code should jump to start
;   when it is done.

    section reset

    org p:$0
    jmp F__start

    org p:$2
    dup $3E
    nop
    endm

    org p:$40 ; For pushing P to NeXT DSP common memory space $1440-$3FF9
savep ds($1440-*)

    endsec

    section crt0

    org y:

; The following varaibles are used for dynamic memory allocation
; __stack_safety: Since dynamic memory and the stack grow towards each other
; This constant tells brk and sbrk what the minimum amount of space should
; be left between the top of stack during the brk or sbrk call and the end
; of any allocated memory.
; __mem_limit: a constant telling brk and sbrk where the end of available
; memory is.
; __break: pointer to the next block of memory that can be allocated
; The heap may be moved by changing the initial value of __break.
; This is the base of the heap.
; __y_size: the base of dynamic memory.
; errno: error type: set by some libraries
; __max_signal the last possible signal.

    global Ferrno
Ferrno  dc  $0

    global F__stack_safety
F__stack_safety dc 256

    global F__mem_limit
F__mem_limit dc TOP_OF_MEMORY

    global F__break
F__breakdc  TOP_OF_MEMORY

    global F__y_size
F__y_size dc$4000 ; normally DSIZE

    global F__max_signal
F__max_signal dc $3e ; what the quint monitor.

    org p:

F__start
    global F__start

;   To change the base of the stack, change the value loaded into the
;   stack pointer here.

    and     #$f3,mr
    and     #$bf,ccr
    movey:F__y_size,r6 ; initialize the stack pointer
    move#$0,r0  ; funny value to terminate a backtrace.

; Do NeXT-DSP reset
    movec #0,omr  ; Disable data ROM, DSP into Mode 0
    bset#0,x:<<m_pbc ; Enable Host Interface via Port B
    bset#3,x:<<m_pcddr ; Enable External RAM via Port C
    bclr#3,x:<<m_pcd
    movep #>$000000,x:<<m_bcr ; No wait states on external SRAM

    jsr Fmain   ; run user program

F__crt0_end
    global F__crt0_end
    stop        ; all done

    endsec

    section crt0__time

    org y:0

    global F__time
F__time dc $0 ; used to time execution with simulator

    org p:

Fclock
    global Fclock

    movey:F__time,a
    tst a
    rts

    endsec

    section crt0__printf_end

;   org y:$3ff
;savemem_pf ds 2

    org p:$3ff  ; we must reserve this location so that
                ; the instruction fetch mechanism will not
                ; automatically trigger the printf.
    nop

    org p:$400  ; this location is recognized by the
                ; exec program.

    global F__printf_end
F__printf_end   ; dummy printf catch routine.
    rts

    endsec

    section crt0__fp_shift
```

```
    org y:

;
; floating point table setup
;

    global F__fp_shift
F__fp_shift dc      $800000,$c00000,$e00000,$f00000,$f80000,$fc0000
        dc      $fe0000,$ff0000,$ff8000,$ffc000,$ffe000,$fff000
        dc      $fff800,$fffc00,$fffe00,$ffff00,$ffff80,$ffffc0
        dc      $ffffe0,$fffff0,$fffff8,$fffffc,$fffffe

    endsec


    section protect_loader

    org     x:$100
x_protect_loader ds ($200-*)

    org     y:$100
y_protect_loader ds ($200-*)

    endsec
```

```
/* Flag to indicate video data should be generated, and not taken */
/* from the SSI line; and data should be taken off the LBox   */
/* transmit queue.                                            */
flag simulating;

/* Flag indicating a main-loop should be performed */
flag doloop;

/* Flag indicating main-loops should only be performed on command */
flag single_loop;

/* IPP addresses for self, and recipients of FEP messages */
IPP_address me, video_dest, lb_dest, testmsg_dest;

/* Values set when communications exceptions occur */
int ssi_rx_exception=0, ssi_tx_exception=0, sci_rx_exception=0;
```

```
#include "stdlib.h"
#include "ipp.h"
#include "buddy_chk.h"

#define min(x,y) (((x)<(y))?(x):(y))


/*** COORDINATE WITH FEPequ.asm ***/
#define VIDBUF_BOTTOM (int *)0
#define VIDBUF_LEN  0x2000
#define VIDBUF_TOP  (int *)(VIDBUF_BOTTOM+VIDBUF_LEN)
#define VIDBUF_RELEVANT_BITS 0x1FFF

#define EVENTS_BOTTOM (unsigned int *)0x2000

#define MAXNUMEVENTS 12000

#define ZERO_MARK 0
#define ONE_MARK 65535

#define ZEROS4 0
#define ONES4 1
#define MIX4 2

#define FRAME_SYNC1  ONES4
#define FRAME_SYNC2  ONES4

#define ROW_SYNC1  ZEROS4
#define ROW_SYNC2  ONES4
#define ROW_SYNC_SKIP 2  /* # of blank words preceding row sync */
#define ROW_SYNC_LEN 10  /* 2 blank words + 2x4 sync marks  */

#define NUMIDBITS  24

#define SLICES_PER_CAMERA 4
#define NUM_CAMERAS   2
#define TOTALSLICES   (SLICES_PER_CAMERA*NUM_CAMERAS)

#define SLICE_A0 0
#define SLICE_B0 1
#define SLICE_C0 2
#define SLICE_D0 3
#define SLICE_A1 4
#define SLICE_B1 5
#define SLICE_C1 6
#define SLICE_D1 7

#define CAMERA_0 0x01
#define CAMERA_1 0x02
#define CAMERA_BOTH 0x03

typedef unsigned char byte;
typedef byte    flag;

typedef struct {
                    unsigned short bad_columns[MAX_BAD_COLUMNS];
                    unsigned short num_bad_columns;
                    unsigned short bad_pixels_x[MAX_BAD_PIXELS];
                    unsigned short bad_pixels_y[MAX_BAD_PIXELS];
                    unsigned short num_bad_pixels;
               } bad_locs_struct;

typedef struct {
                    unsigned short mode;
                    unsigned short used_cameras;
                    unsigned short num_hucs;
                    unsigned short num_cols;
                    unsigned short num_hocs;
                    unsigned short total_cols;
                    unsigned short num_rows;
                    unsigned short num_vocs;
                    unsigned short start_row;
                    unsigned short total_rows;
               } ccd_param_struct;

typedef struct {
                    ccd_param_struct ccd;
                    unsigned short  thresh[TOTALSLICES];
                    bad_locs_struct  *bad;
                    unsigned int  id;
                    int     *left_off, *this_line_start;
                    int one_line_delta;
                    int     lines_rcvd;
                    flagwaiting_for_frame_sync;
                    flagdo_sifting;
                    flagsend_z;
                    int     numevents;
                    unsigned int *events_ptr;
               } vidbuf_param_struct;


/*** FEPasmsupport.asm: Global routines ***/
void init_communications();
void copy_lowp_routines();
void init_intr_vectors();
int check_host(unsigned short *value);
int get_host();
void put_host(unsigned short value);
void txd_on(void);
void txd_off(void);
void txd_tog(void);
void sc1_on(void);
void sc1_off(void);


/*** FEPcom.c: Global routines ***/
void Send_Host_Message(IPP_address dest, IPP_msg_type type,
    IPP_priority priority, unsigned short data_len, IPP_msg_buffer msg);
void send_test_word(unsigned short test_word);
void send_test_int(int test_int);
void try_get_message(vidbuf_param_struct *vbp, vidbuf_param_struct *vbpnew);
void try_fill_1bcmd();
void try_relay_hk();
void try_relay_errors();
void signal_frame_started(unsigned int id);
void signal_frame_ended(unsigned int id);
void signal_frame_error(unsigned int id, unsigned short reason);
void dump_events(vidbuf_param_struct *vbp);
void parse_tdb_command(void);


/*** FEPmain.c: Global routines ***/
void memdump_cmd(IPP_msg_buffer msg);
void mempoke_cmd(IPP_msg_buffer msg);
void memdump_auto(char space, unsigned short start, unsigned short end);


/*** FEPproc.asm: Global routines ***/
void proc_line(int *start, int mode, int up1, int down1,
    vidbuf_param_struct *vbp_ptr);


/*** FEPsetvars.c: Global routines ***/
```

```
void init_portc_intrs();
void onetime_init(vidbuf_param_struct *vbp, vidbuf_param_struct *vbpnew);


/*** FEPvid.c: Global routines ***/
void try_proc_vid(vidbuf_param_struct *vbp, vidbuf_param_struct *vbpnew);
void record_event(int *address, int value, int highpixel,
    vidbuf_param_struct *vbpnew);
void abort_stream(vidbuf_param_struct *vbp, vidbuf_param_struct *vbpnew);


/*** Assembly-declared variables ***/
register int *asm_inptr   __asm("r5"); /* vidbuf write pointer */

extern int asm_proclen;
extern int asm_thresh0, asm_thresh1, asm_thresh2, asm_thresh3;
extern int asm_thresh4, asm_thresh5, asm_thresh6, asm_thresh7;

extern int sci_rxq[], sci_rxq_end[];
extern int ssi_txq[], ssi_txq_end[];

extern int *sci_rx_outptr;
extern int *ssi_tx_inptr;

/* How many words have been put into the buffer from the SCI line, */
/* and how many have been sent down to the GPP      */
extern int sci_rx_intrs, sci_rxq_xfers;

/* How many words have been put into the buffer by us, and how many */
/* have been sent to the Lasagna Box            */
extern int ssi_tx_intrs, ssi_txq_xfers;
```

```
include $(HETEDIR)/Makefiles/global_defs

INCLUDE = $(HETEINCLUDE)
NAME= FEP
GAWK = /hete/bin/nextbin/cldlodfix.gawk
NEXTINCLUDE = $(HETEINCLUDE)/nextinclude

PBCLNs = pbcln/FEPmain.cln pbcln/FEPsetvars.cln pbcln/FEPvid.cln\
    pbcln/FEPasmsupport.cln pbcln/FEPproc.cln pbcln/FEPintr.cln\
    pbcln/FEPcom.cln
PBCRT = pbcln/crtrzpy.cln
PBOPTS = -g -c -Wall -alo -ffixed-r5 -DUSEIPP -I$(PBDSPINCLUDE) -I$(INCLUDE)
PBSUPPORT = machspec.asm FEPequ.asm
PBLIBCLNS = $(PBDSPLIB)/ippshell.cln

ASMs = asm/FEPmain.asm asm/FEPsetvars.asm asm/FEPvid.asm\
    asm/FEPcom.asm asm/FEPproc.asm\
    asm/FEPintr.asm asm/FEPasmsupport.asm
ASMOPTS = -g -c -Wall -alo -ffixed-r5 -S -DUSEIPP -I$(PBDSPINCLUDE) -I$(INCLUDE)


ifeq ($(PBDSP),YES)
PBDSPLODFILES := pb_$(NAME).hld
endif


PBFEP: $(PBCRT) $(PBDSPLODFILES)

ASMFEP: $(ASMs)



pb_$(NAME).hld: obj/pb_$(NAME).lod
    lod2hostload obj/pb_$(NAME).lod
    rm -f pb_$(NAME).hld
    mv obj/pb_$(NAME).hld .

obj/pb_$(NAME).lod: obj/pb_$(NAME).cld
    cldlod obj/pb_$(NAME).cld | gawk -f $(GAWK) > obj/pb_$(NAME).lod.realspace
    strip_last.sh obj/pb_$(NAME).lod.realspace
    ../utils/shiftlowp obj/pb_$(NAME).lod.realspace obj/pb_$(NAME).lod

obj/pb_$(NAME).cld: $(PBCLNs) $(SUPPORTFILES) Makefile
    g56k -j -mobj/pb_$(NAME).map -my-memory -mstack_check -crt $(PBCRT) $(PBSUPPORT) $(P
BCLNs) $(PBLIBCLNS) -o obj/pb_$(NAME).cld -l$(PBDSPLIB)/librzpy.clb -I$(PBDSPINCLUDE) -I
$(INCLUDE)



pbcln/FEPasmsupport.cln: FEPasmsupport.asm Makefile
    g56k FEPasmsupport.asm $(PBOPTS) -o pbcln/FEPasmsupport.cln

asm/FEPasmsupport.asm: FEPasmsupport.asm Makefile
    cp FEPasmsupport.asm asm/FEPasmsupport.asm


pbcln/FEPcom.cln: FEPcom.c Makefile
    g56k FEPcom.c $(PBOPTS) -o pbcln/FEPcom.cln

asm/FEPcom.asm: FEPcom.c Makefile
    g56k FEPcom.c $(ASMOPTS) -o asm/FEPcom.asm


pbcln/FEPintr.cln: FEPintr.asm Makefile
    g56k FEPintr.asm $(PBOPTS) -o pbcln/FEPintr.cln
`
asm/FEPintr.asm: FEPintr.asm Makefile
    cp FEPintr.asm asm/FEPintr.asm


pbcln/FEPmain.cln: FEPmain.c Makefile
    g56k FEPmain.c $(PBOPTS) -o pbcln/FEPmain.cln

asm/FEPmain.asm: FEPmain.c Makefile
    g56k FEPmain.c $(ASMOPTS) -o asm/FEPmain.asm


pbcln/FEPproc.cln: FEPproc.asm Makefile
    g56k FEPproc.asm $(PBOPTS) -o pbcln/FEPproc.cln

asm/FEPproc.asm: FEPasmsupport.asm Makefile
    cp FEPproc.asm asm/FEPproc.asm


pbcln/FEPsetvars.cln: FEPsetvars.c Makefile
    g56k FEPsetvars.c $(PBOPTS) -o pbcln/FEPsetvars.cln

asm/FEPsetvars.asm: FEPsetvars.c Makefile
    g56k FEPsetvars.c $(ASMOPTS) -o asm/FEPsetvars.asm


pbcln/FEPvid.cln: FEPvid.c Makefile
    g56k FEPvid.c $(PBOPTS) -o pbcln/FEPvid.cln

asm/FEPvid.asm: FEPvid.c Makefile
    g56k FEPvid.c $(ASMOPTS) -o asm/FEPvid.asm


pbcln/crtrzpy.cln: crtrzp.asm Makefile
    asm56000 -I$(HETEDIR)/lib/pbdsplib/src -bpbcln/crtrzpy.cln -lobj/crtrzpy.lst -dmm Y
crtrzp.asm


include $(HETEDIR)/Makefiles/global_rules
```

Section Link Map by Address

X Memory (default)

Start	End	Length	Section	
0000	1FFF	8192	GLOBAL	Abs Mod
8000	FFBF	32704	RZP_Addresses	Abs

X Memory (low)

Start	End	Length	Section

No sections

X Memory (high)

Start	End	Length	Section

No sections

Y Memory (default)

Start	End	Length	Section	
0000	0025	38	GLOBAL	Abs
0026	002B	6	crt0	
002C	0129	254	FEPmain_c	
012A	012C	3	FEPvid_c	
012D	097B	2127	FEPcom_c	
097C	098B	16	ippshell_c	
098C	098C	1	ipp_driver	
098D	098E	2	RTC_handler	
098F	09A0	18	buddy_chk_c	
09A1	09A1	1	maloc_c	
1000	143F	1088	GLOBAL	Abs Mod
8000	FFBF	32704	RZP_Addresses	Abs
FFC4	FFFF	60	RZP_Addresses	Abs

Y Memory (low)

Start	End	Length	Section

No sections

Y Memory (high)

Start	End	Length	Section

No sections

L Memory (default)

Start	End	Length	Section

No sections

L Memory (low)

Start	End	Length	Section

No sections

L Memory (high)

Start	End	Length	Section

No sections

P Memory (default)

Start	End	Length	Section	
0000	0AAA	2731	bootstrapper	Abs
0040	0067	40	interrupts	Abs *Overlap*
00A0	0141	162	proc_line	Abs
0AAB	0AD6	44	crt0	Abs
0AD7	0EC3	1005	FEPmain_c	
0EC4	0F94	209	FEPsetvars_c	
0F95	162A	1686	FEPvid_c	
162B	16C5	155	asmsupport	
16C6	231D	3160	FEPcom_c	
231E	25EB	718	ippshell_c	
25EC	2688	157	ipp_driver	
2689	2706	126	RTC_handler	
2707	2783	125	buddy_chk_c	
2784	290A	391	maloc_c	
290B	2932	40	mcpy_c	
2933	298F	93	brk_c	
8000	FFFF	32768	RZP_Addresses	Abs

P Memory (low)

Start	End	Length	Section

No sections

P Memory (high)

Start	End	Length	Section

No sections

Section Link Map by Name

Section	Memory	Start	End	Length
FEPcom_c	Y default	012D	097B	2127
	P default	16C6	231D	3160

FEPmain_c		Y	default	002C	0129	254
		P	default	0AD7	0EC3	1005
FEPsetvars_c		P	default	0EC4	0F94	209
FEPvid_c		Y	default	012A	012C	3
		P	default	0F95	162A	1686
GLOBAL	Mod	X	default	0000	1FFF	8192
	Abs	Y	default	0000	0025	38
	Mod	Y	default	1000	143F	1088
RTC_handler		Y	default	098D	098E	2
		P	default	2689	2706	126
RZP_Addresses	Abs	X	default	8000	FFBF	32704
	Abs	Y	default	8000	FFBF	32704
	Abs	Y	default	FFC4	FFFF	60
	Abs	P	default	8000	FFFF	32768
asmsupport		P	default	162B	16C5	155
bootstrapper	Abs	P	default	0000	0AAA	2731
brk_c		P	default	2933	298F	93
buddy_chk_c		Y	default	098F	09A0	18
		P	default	2707	2783	125
crt0		Y	default	0026	002B	6
	Abs	P	default	0AAB	0AD6	44
interrupts	Abs	P	default	0040	0067	40
ipp_driver		Y	default	098C	098C	1
		P	default	25EC	2688	157
ippshell_c		Y	default	097C	098B	16
		P	default	231E	25EB	718
maloc_c		Y	default	09A1	09A1	1
		P	default	2784	290A	391
mcpy_c		P	default	290B	2932	40
proc_line	Abs	P	default	00A0	0141	162

Symbol Listing by Name

Name	Type	Value	Section	Attributes
ABS_JSR_INSTR....	int	0BF080	ipp_driver	ABS LOCAL
ACK_DT.....	int	000004	ipp_driver	ABS LOCAL
ACK_PORT.....	int	00FFE8	ipp_driver	ABS LOCAL
ACK_TD.....	int	000003	ipp_driver	ABS LOCAL
ASM_APP_0.....	int	P:000AE3	FEPmain_c	REL LOCAL
ASM_APP_0.....	int	P:000EC9	FEPsetvars_c	REL LOCAL
ASM_APP_0.....	int	P:000FA6	FEPvid_c	REL LOCAL
ASM_APP_0.....	int	P:001955	FEPcom_c	REL LOCAL
ASM_APP_0.....	int	P:002940	brk_c	REL LOCAL
ASM_APP_1.....	int	P:000AE4	FEPmain_c	REL LOCAL
ASM_APP_1.....	int	P:000ECA	FEPsetvars_c	REL LOCAL
ASM_APP_1.....	int	P:000FA8	FEPvid_c	REL LOCAL
ASM_APP_1.....	int	P:001956	FEPcom_c	REL LOCAL
ASM_APP_1.....	int	P:002941	brk_c	REL LOCAL
ASM_APP_10.....	int	P:000C5F	FEPmain_c	REL LOCAL
ASM_APP_11.....	int	P:000C60	FEPmain_c	REL LOCAL
ASM_APP_12.....	int	P:000C6C	FEPmain_c	REL LOCAL
ASM_APP_13.....	int	P:000C6D	FEPmain_c	REL LOCAL
ASM_APP_14.....	int	P:000CA9	FEPmain_c	REL LOCAL
ASM_APP_15.....	int	P:000CAB	FEPmain_c	REL LOCAL
ASM_APP_16.....	int	P:000D30	FEPmain_c	REL LOCAL
ASM_APP_17.....	int	P:000D32	FEPmain_c	REL LOCAL
ASM_APP_18.....	int	P:000E21	FEPmain_c	REL LOCAL
ASM_APP_19.....	int	P:000E22	FEPmain_c	REL LOCAL
ASM_APP_1A.....	int	P:000E27	FEPmain_c	REL LOCAL
ASM_APP_1B.....	int	P:000E28	FEPmain_c	REL LOCAL
ASM_APP_2.....	int	P:000AEA	FEPmain_c	REL LOCAL
ASM_APP_2.....	int	P:000ECA	FEPsetvars_c	REL LOCAL
ASM_APP_2.....	int	P:000FB3	FEPvid_c	REL LOCAL
ASM_APP_2.....	int	P:002260	FEPcom_c	REL LOCAL
ASM_APP_3.....	int	P:000AEB	FEPmain_c	REL LOCAL
ASM_APP_3.....	int	P:000ECB	FEPsetvars_c	REL LOCAL
ASM_APP_3.....	int	P:000FB5	FEPvid_c	REL LOCAL
ASM_APP_3.....	int	P:002262	FEPcom_c	REL LOCAL
ASM_APP_4.....	int	P:000B4D	FEPmain_c	REL LOCAL
ASM_APP_4.....	int	P:000ECB	FEPsetvars_c	REL LOCAL
ASM_APP_4.....	int	P:000FC1	FEPvid_c	REL LOCAL
ASM_APP_4.....	int	P:00228E	FEPcom_c	REL LOCAL
ASM_APP_5.....	int	P:000B4E	FEPmain_c	REL LOCAL
ASM_APP_5.....	int	P:000ECC	FEPsetvars_c	REL LOCAL
ASM_APP_5.....	int	P:000FC3	FEPvid_c	REL LOCAL
ASM_APP_5.....	int	P:00228F	FEPcom_c	REL LOCAL
ASM_APP_6.....	int	P:000BAE	FEPmain_c	REL LOCAL
ASM_APP_6.....	int	P:000ECE	FEPsetvars_c	REL LOCAL
ASM_APP_6.....	int	P:000FCF	FEPvid_c	REL LOCAL
ASM_APP_7.....	int	P:000BAF	FEPmain_c	REL LOCAL
ASM_APP_7.....	int	P:000ECF	FEPsetvars_c	REL LOCAL
ASM_APP_7.....	int	P:000FD1	FEPvid_c	REL LOCAL
ASM_APP_8.....	int	P:000BCA	FEPmain_c	REL LOCAL
ASM_APP_8.....	int	P:000ECF	FEPsetvars_c	REL LOCAL
ASM_APP_8.....	int	P:001296	FEPvid_c	REL LOCAL
ASM_APP_9.....	int	P:000BCB	FEPmain_c	REL LOCAL
ASM_APP_9.....	int	P:000ED0	FEPsetvars_c	REL LOCAL
ASM_APP_9.....	int	P:001298	FEPvid_c	REL LOCAL
ASM_APP_A.....	int	P:000BDE	FEPmain_c	REL LOCAL
ASM_APP_A.....	int	P:000ED0	FEPsetvars_c	REL LOCAL

```
ASM_APP_A........int   P:0012A6           FEPvid_c          REL LOCAL
ASM_APP_B........int   P:000BDF           FEPmain_c         REL LOCAL
ASM_APP_B........int   P:000ED1           FEPsetvars_c      REL LOCAL
ASM_APP_B........int   P:0012A7           FEPvid_c          REL LOCAL
ASM_APP_C........int   P:000BF0           FEPmain_c         REL LOCAL
ASM_APP_C........int   P:000F15           FEPsetvars_c      REL LOCAL
ASM_APP_C........int   P:0015E8           FEPvid_c          REL LOCAL
ASM_APP_D........int   P:000BF1           FEPmain_c         REL LOCAL
ASM_APP_D........int   P:000F16           FEPsetvars_c      REL LOCAL
ASM_APP_D........int   P:0015E9           FEPvid_c          REL LOCAL
ASM_APP_E........int   P:000C45           FEPmain_c         REL LOCAL
ASM_APP_E........int   P:0015F4           FEPvid_c          REL LOCAL
ASM_APP_F........int   P:000C46           FEPmain_c         REL LOCAL
ASM_APP_F........int   P:0015F5           FEPvid_c          REL LOCAL
BASE_PROC_ID.....int     000004           ipp_driver        ABS LOCAL
BOARD_ID_MASK....int     000F00           GLOBAL            ABS GLOBAL
BOARD_ID_MASK....int     000F00           ipp_driver        ABS LOCAL
BOARD_ID_MASK....int     000F00           RTC_handler       ABS LOCAL
Boot_EEROM.......int   P:008000           RZP_Addresses     ABS LOCAL
CPU_Mhz..........fpt     2.000000E+001    GLOBAL            ABS GLOBAL
CRT0ORG..........int     000AAB           GLOBAL            ABS GLOBAL
DEBUG_BIT_MASK...int     000060           GLOBAL            ABS GLOBAL
DEBUG_BIT_MASK...int     000060           ipp_driver        ABS LOCAL
DEBUG_BIT_MASK...int     000060           RTC_handler       ABS LOCAL
DSIZE............int     0009A2           GLOBAL            ABS GLOBAL
DSP_ID_BITNO.....int     000007           GLOBAL            ABS GLOBAL
DSP_ID_BITNO.....int     000007           ipp_driver        ABS LOCAL
DSP_ID_BITNO.....int     000007           RTC_handler       ABS LOCAL
DT_MODE..........int     000001           ipp_driver        ABS LOCAL
DT_MODEBIT.......int     000000           ipp_driver        ABS LOCAL
EEPROM_WE_BITNO..int     000004           GLOBAL            ABS GLOBAL
EEPROM_WE_BITNO..int     000004           ipp_driver        ABS LOCAL
EEPROM_WE_BITNO..int     000004           RTC_handler       ABS LOCAL
EVENTS_BOTTOM....int     002000           GLOBAL            ABS GLOBAL
FDGlue_Ctrl......int     00FFC3           GLOBAL            ABS GLOBAL
FDGlue_Ctrl......int     00FFC3           ipp_driver        ABS LOCAL
FDGlue_Ctrl......int     00FFC3           RTC_handler       ABS LOCAL
FDGlue_Stat......int     00FFC3           GLOBAL            ABS GLOBAL
FDGlue_Stat......int     00FFC3           ipp_driver        ABS LOCAL
FDGlue_Stat......int     00FFC3           RTC_handler       ABS LOCAL
FFrameReg_X8.....int   Y:00EF00           RZP_Addresses     ABS GLOBAL
FFrameReg_X9.....int   Y:00EF01           RZP_Addresses     ABS GLOBAL
FFrameReg_XA.....int   Y:00EF02           RZP_Addresses     ABS GLOBAL
FFrameReg_XB.....int   Y:00EF03           RZP_Addresses     ABS GLOBAL
FFrameReg_Y8.....int   Y:00EF04           RZP_Addresses     ABS GLOBAL
FFrameReg_Y9.....int   Y:00EF05           RZP_Addresses     ABS GLOBAL
FFrameReg_YA.....int   Y:00EF06           RZP_Addresses     ABS GLOBAL
FFrameReg_YB.....int   Y:00EF07           RZP_Addresses     ABS GLOBAL
FFrame_X8........int   X:008000           RZP_Addresses     ABS GLOBAL
FFrame_X9........int   X:009000           RZP_Addresses     ABS GLOBAL
FFrame_XA........int   X:00A000           RZP_Addresses     ABS GLOBAL
FFrame_XB........int   X:00B000           RZP_Addresses     ABS GLOBAL
FFrame_Y8........int   Y:008000           RZP_Addresses     ABS GLOBAL
FFrame_Y9........int   Y:009000           RZP_Addresses     ABS GLOBAL
FFrame_YA........int   Y:00A000           RZP_Addresses     ABS GLOBAL
FFrame_YB........int   Y:00B000           RZP_Addresses     ABS GLOBAL
FGPS_Count.......int     00FFC2           GLOBAL            ABS GLOBAL
FGPS_Count.......int     00FFC2           ipp_driver        ABS LOCAL
FGPS_Count.......int     00FFC2           RTC_handler       ABS LOCAL
FGet_Host_Messag.int   P:00170C           FEPcom_c          REL GLOBAL
```



```
FIPP_collectmsg..int   P:00231E           ippshell_c        REL GLOBAL
FIPP_get_address.int   P:0025B1           ippshell_c        REL GLOBAL
FIPP_get_message.int   P:0023B7           ippshell_c        REL GLOBAL
FIPP_make_header.int   P:002439           ippshell_c        REL GLOBAL
FIPP_makeheader..int   P:0024A9           ippshell_c        REL GLOBAL
FIPP_register_ad.int   P:002527           ippshell_c        REL GLOBAL
FIPP_send_messag.int   P:0023F1           ippshell_c        REL GLOBAL
FIPPbuf..........int   Y:000358           FEPcom_c          REL GLOBAL
FIPPbufHead......int   Y:000357           FEPcom_c          REL GLOBAL
FIPPbufTail......int   Y:000356           FEPcom_c          REL GLOBAL
FIPPresponse.....int   Y:000154           FEPcom_c          REL GLOBAL
FRF_Count........int     00FFC1           GLOBAL            ABS GLOBAL
FRF_Count........int     00FFC1           ipp_driver        ABS LOCAL
FRF_Count........int     00FFC1           RTC_handler       ABS LOCAL
FRTC_Count.......int     00FFC0           GLOBAL            ABS GLOBAL
FRTC_Count.......int     00FFC0           ipp_driver        ABS LOCAL
FRTC_Count.......int     00FFC0           RTC_handler       ABS LOCAL
FSend_Host_Messa.int   P:0016C6           FEPcom_c          REL GLOBAL
F___ABCDcnt1.....int   Y:000111           FEPmain_c         REL LOCAL
F___Aline2.......int   Y:000112           FEPmain_c         REL LOCAL
F___Bline3.......int   Y:000113           FEPmain_c         REL LOCAL
F___Cline4.......int   Y:000114           FEPmain_c         REL LOCAL
F___Dline5.......int   Y:000115           FEPmain_c         REL LOCAL
F___count1.......int   Y:00012B           FEPvid_c          REL LOCAL
F___got_header0..int   Y:00097C           ippshell_c        REL LOCAL
F___header1......int   Y:00097D           ippshell_c        REL LOCAL
F___id0..........int   Y:000110           FEPmain_c         REL LOCAL
F___last_tick0...int   Y:000993           buddy_chk_c       REL LOCAL
F___local_id2....int   Y:00012C           FEPvid_c          REL LOCAL
F___repeatmode0..int   Y:00013C           FEPcom_c          REL LOCAL
F___sim_cntr6....int   Y:000116           FEPmain_c         REL LOCAL
F___state0.......int   Y:00012A           FEPvid_c          REL LOCAL
F___timeout1.....int   Y:000152           FEPcom_c          REL LOCAL
F__break.........int   Y:000029           crt0              REL GLOBAL
F__crt0_end......int   P:000AD1           crt0              ABS GLOBAL
F__max_signal....int   Y:00002B           crt0              REL GLOBAL
F__mem_limit.....int   Y:000028           crt0              REL GLOBAL
F__printf_end....int   P:000AD5           crt0              ABS GLOBAL
F__stack_safety..int   Y:000027           crt0              REL GLOBAL
F__start.........int   P:000AAB           crt0              ABS GLOBAL
F__y_size........int   Y:00002A           crt0              REL GLOBAL
Fabort_stream....int   P:00161D           FEPvid_c          REL GLOBAL
Falt_simulate_wo.int   P:000C38           FEPmain_c         REL GLOBAL
Fasm_proclen.....int   Y:000010           GLOBAL            ABS GLOBAL
Fasm_thresh0.....int   Y:000014           GLOBAL            ABS GLOBAL
Fasm_thresh1.....int   Y:000015           GLOBAL            ABS GLOBAL
Fasm_thresh2.....int   Y:000016           GLOBAL            ABS GLOBAL
Fasm_thresh3.....int   Y:000017           GLOBAL            ABS GLOBAL
Fasm_thresh4.....int   Y:000018           GLOBAL            ABS GLOBAL
Fasm_thresh5.....int   Y:000019           GLOBAL            ABS GLOBAL
Fasm_thresh6.....int   Y:00001A           GLOBAL            ABS GLOBAL
Fasm_thresh7.....int   Y:00001B           GLOBAL            ABS GLOBAL
Fbrk.............int   P:002933           brk_c             REL GLOBAL
Fbuddy_hdr.......int   Y:000994           buddy_chk_c       REL GLOBAL
Fccd0sim.........int   Y:000030           FEPmain_c         REL GLOBAL
Fccd1sim.........int   Y:0000A0           FEPmain_c         REL GLOBAL
Fcheck4..........int   P:000F95           FEPvid_c          REL GLOBAL
Fcheck_buddy_tim.int   P:002707           buddy_chk_c       REL GLOBAL
Fcheck_host......int   P:001688           asmsupport        REL GLOBAL
Fcopy_lowp_routi.int   P:001659           asmsupport        REL GLOBAL
```

```
FcurrentCmd.....int  Y:000355  FEPcom_c  REL GLOBAL
FcurrentResponse.int  Y:000354  FEPcom_c  REL GLOBAL
Fdoloop.........int  Y:000128  FEPmain_c  REL GLOBAL
Fdrop_frame_sync.int  Y:00002F  FEPmain_c  REL GLOBAL
Fdump_events.....int  P:0022E6  FEPcom_c  REL GLOBAL
Ferrno..........int  Y:000026  crt0  REL GLOBAL
Fevent_loc_okay..int  P:0014D2  FEPvid_c  REL GLOBAL
Ffree...........int  P:0028EE  maloc_c  REL GLOBAL
Fget_coords.....int  P:001026  FEPvid_c  REL GLOBAL
Fget_host.......int  P:00169D  asmsupport  REL GLOBAL
Fget_irqa_tick...int  P:0026A7  RTC_handler  REL GLOBAL
Fget_nsynch_time.int  P:0026AB  RTC_handler  REL GLOBAL
Fhead...........int  Y:0009A1  maloc_c  REL LOCAL
Finit_RTC_driver.int  P:002689  RTC_handler  REL GLOBAL
Finit_communicat.int  P:00162B  asmsupport  REL GLOBAL
Finit_intr_vecto.int  P:001665  asmsupport  REL GLOBAL
Finit_portc_intr.int  P:000EC4  FEPsetvars_c  REL GLOBAL
Finitialize_dma_.int  P:0025EC  ipp_driver  REL GLOBAL
Flb_dest........int  Y:00011B  FEPmain_c  REL GLOBAL
Flb_fill_command.int  Y:00012D  FEPcom_c  REL GLOBAL
Fload_init_proc_.int  P:00121C  FEPvid_c  REL GLOBAL
Fload_thresh.....int  P:001146  FEPvid_c  REL GLOBAL
Flocal_processor.int  Y:00098A  ippshell_c  REL GLOBAL
Fmain...........int  P:000E5F  FEPmain_c  REL GLOBAL
Fmajortag.......int  Y:00098F  buddy_chk_c  REL GLOBAL
Fmalloc.........int  P:002784  maloc_c  REL GLOBAL
Fmaster_node_id..int  Y:00098B  ippshell_c  REL LOCAL
Fme.............int  Y:000123  FEPmain_c  REL GLOBAL
Fmemcpy.........int  P:00290B  mcpy_c  REL GLOBAL
Fmemdump_auto....int  P:000E35  FEPmain_c  REL GLOBAL
Fmemdump_cmd.....int  P:000D4D  FEPmain_c  REL GLOBAL
Fmempoke_cmd.....int  P:000DEC  FEPmain_c  REL GLOBAL
Fminortag.......int  Y:000991  buddy_chk_c  REL GLOBAL
Fmy_ipp_addr.....int  Y:0009A0  buddy_chk_c  REL GLOBAL
Fmy_name........int  Y:000989  ippshell_c  REL LOCAL
Fonetime_init....int  P:000ED2  FEPsetvars_c  REL GLOBAL
Fparam_dump.....int  P:0017F1  FEPcom_c  REL GLOBAL
Fparse_tdb_comma.int  P:001A12  FEPcom_c  REL GLOBAL
Fparse_viddsp_cm.int  P:0018D8  FEPcom_c  REL GLOBAL
Fpmemcpy........int  P:000D09  FEPmain_c  REL GLOBAL
Fprepare_IPP_hea.int  P:002734  buddy_chk_c  REL GLOBAL
Fproc_line......int  P:0000A0  proc_line  ABS GLOBAL
Fput_host.......int  P:0016A3  asmsupport  REL GLOBAL
Fread_pending....int  Y:000153  FEPcom_c  REL GLOBAL
Frealloc........int  P:00288A  maloc_c  REL GLOBAL
Freceive_words...int  P:00260B  ipp_driver  REL GLOBAL
Frecord_event....int  P:00157A  FEPvid_c  REL GLOBAL
Fsbrk...........int  P:00296A  brk_c  REL GLOBAL
Fscl_off........int  P:0016C4  asmsupport  REL GLOBAL
Fscl_on.........int  P:0016C2  asmsupport  REL GLOBAL
Fsci_rx_exceptio.int  Y:00002E  FEPmain_c  REL GLOBAL
Fsci_rx_intrs....int  Y:00000B  GLOBAL  ABS GLOBAL
Fsci_rx_outptr...int  Y:00000C  GLOBAL  ABS GLOBAL
Fsci_rxq........int  Y:001400  GLOBAL  ABS GLOBAL BUFFER
Fsci_rxq_end.....int  Y:001440  GLOBAL  ABS GLOBAL
Fsci_rxq_xfers...int  Y:00000A  GLOBAL  ABS GLOBAL
Fsend_packet_of_.int  P:0021EC  FEPcom_c  REL GLOBAL
Fsend_test_int...int  P:001765  FEPcom_c  REL GLOBAL
Fsend_test_word..int  P:00172F  FEPcom_c  REL GLOBAL
Fsend_words.....int  P:00264B  ipp_driver  REL GLOBAL
```



```
Fsignal_frame_en.int  P:002167  FEPcom_c  REL GLOBAL
Fsignal_frame_er.int  P:0021A6  FEPcom_c  REL GLOBAL
Fsignal_frame_st.int  P:002128  FEPcom_c  REL GLOBAL
Fsim4...........int  P:000AD7  FEPmain_c  REL GLOBAL
Fsimulate_frame_.int  P:000AEF  FEPmain_c  REL GLOBAL
Fsimulate_lbcmd_.int  P:000C57  FEPmain_c  REL GLOBAL
Fsimulate_row_sy.int  P:000B47  FEPmain_c  REL GLOBAL
Fsimulate_word_r.int  P:000B5E  FEPmain_c  REL GLOBAL
Fsimulating.....int  Y:000129  FEPmain_c  REL GLOBAL
Fsingle_loop.....int  Y:000127  FEPmain_c  REL GLOBAL
Fssi_rx_exceptio.int  Y:00002C  FEPmain_c  REL GLOBAL
Fssi_tx_exceptio.int  Y:00002D  FEPmain_c  REL GLOBAL
Fssi_tx_inptr....int  Y:000007  GLOBAL  ABS GLOBAL
Fssi_tx_intrs....int  Y:000006  GLOBAL  ABS GLOBAL
Fssi_txq.........int  Y:001000  GLOBAL  ABS GLOBAL BUFFER
Fssi_txq_end.....int  Y:001400  GLOBAL  ABS GLOBAL
Fssi_txq_xfers...int  Y:000005  GLOBAL  ABS GLOBAL
Fswitch_params...int  P:000FF4  FEPvid_c  REL GLOBAL
Fsync_frame.....int  P:001247  FEPvid_c  REL GLOBAL
Fsync_row.......int  P:001369  FEPvid_c  REL GLOBAL
Ftestmsg_dest....int  Y:000117  FEPmain_c  REL GLOBAL
Ftry_fill_lbcmd..int  P:001EA7  FEPcom_c  REL GLOBAL
Ftry_get_message.int  P:001B75  FEPcom_c  REL GLOBAL
Ftry_proc_vid....int  P:0013D3  FEPvid_c  REL GLOBAL
Ftry_relay_error.int  P:00209B  FEPcom_c  REL GLOBAL
Ftry_relay_hk....int  P:001EB7  FEPcom_c  REL GLOBAL
Ftxd_off........int  P:0016BE  asmsupport  REL GLOBAL
Ftxd_on.........int  P:0016BC  asmsupport  REL GLOBAL
Ftxd_tog........int  P:0016C0  asmsupport  REL GLOBAL
Fvideo_dest.....int  Y:00011F  FEPmain_c  REL GLOBAL
Fwrite_lbcmd.....int  P:0017A8  FEPcom_c  REL GLOBAL
Fxmemcpy........int  P:000C82  FEPmain_c  REL GLOBAL
Fymemcpy........int  P:000CC6  FEPmain_c  REL GLOBAL
GPP_REQ_DT_ADR...int  000028  ipp_driver  ABS LOCAL
GPP_REQ_DT_NUM...int  000014  ipp_driver  ABS LOCAL
HFB_EVENT_BITNO..int  000005  GLOBAL  ABS GLOBAL
HFB_EVENT_BITNO..int  000005  ipp_driver  ABS LOCAL
HFB_EVENT_BITNO..int  000005  RTC_handler  ABS LOCAL
IRQA_L0_BIT.....int  000000  RTC_handler  ABS LOCAL
IRQA_L1_BIT.....int  000001  RTC_handler  ABS LOCAL
IRQA_SEN_BIT....int  000002  RTC_handler  ABS LOCAL
IRQA_SRCSTAT_BIT.int  000001  GLOBAL  ABS GLOBAL
IRQA_SRCSTAT_BIT.int  000001  ipp_driver  ABS LOCAL
IRQA_SRCSTAT_BIT.int  000001  RTC_handler  ABS LOCAL
IRQA_SRC_BITNO...int  000004  GLOBAL  ABS GLOBAL
IRQA_SRC_BITNO...int  000004  ipp_driver  ABS LOCAL
IRQA_SRC_BITNO...int  000004  RTC_handler  ABS LOCAL
IRQA_STAT_BITNO..int  000003  GLOBAL  ABS GLOBAL
IRQA_STAT_BITNO..int  000003  ipp_driver  ABS LOCAL
IRQA_STAT_BITNO..int  000003  RTC_handler  ABS LOCAL
IRQA_VEC_ADR....int  000008  RTC_handler  ABS LOCAL
IRQB_STAT_BITNO..int  000002  GLOBAL  ABS GLOBAL
IRQB_STAT_BITNO..int  000002  ipp_driver  ABS LOCAL
IRQB_STAT_BITNO..int  000002  RTC_handler  ABS LOCAL
IRQ_SRC_BIT.....int  000004  RTC_handler  ABS LOCAL
L1..............int  P:0023AE  ippshell_c  REL LOCAL
L1..............int  P:00272B  buddy_chk_c  REL LOCAL
L1..............int  P:00287C  maloc_c  REL LOCAL
L10.............int  P:000F4E  FEPsetvars_c  REL LOCAL
L10.............int  P:00101F  FEPvid_c  REL LOCAL
```



```
L10.............int  P:001799  FEPcom_c  REL LOCAL
L10.............int  P:0023E8  ippshell_c  REL LOCAL
```

```
L10.............int   P:00283E          maloc_c          REL LOCAL
L102............int   P:0015D7          FEPvid_c         REL LOCAL
L105............int   P:001D19          FEPcom_c         REL LOCAL
L106............int   P:001D54          FEPcom_c         REL LOCAL
L108............int   P:001DC2          FEPcom_c         REL LOCAL
L11.............int   P:000AE8          FEPmain_c        REL LOCAL
L11.............int   P:00100A          FEPvid_c         REL LOCAL
L11.............int   P:0023BE          ippshell_c       REL LOCAL
L112............int   P:001DD2          FEPcom_c         REL LOCAL
L113............int   P:001DEC          FEPcom_c         REL LOCAL
L114............int   P:001DEC          FEPcom_c         REL LOCAL
L115............int   P:001E28          FEPcom_c         REL LOCAL
L116............int   P:001E60          FEPcom_c         REL LOCAL
L117............int   Y:000149          FEPcom_c         REL LOCAL
L119............int   P:001DEA          FEPcom_c         REL LOCAL
L12.............int   P:0017E4          FEPcom_c         REL LOCAL
L12.............int   P:0023E8          ippshell_c       REL LOCAL
L12.............int   P:002822          maloc_c          REL LOCAL
L120............int   P:001D47          FEPcom_c         REL LOCAL
L121............int   P:001D46          FEPcom_c         REL LOCAL
L122............int   P:001CA1          FEPcom_c         REL LOCAL
L123............int   P:001CA0          FEPcom_c         REL LOCAL
L124............int   P:001E24          FEPcom_c         REL LOCAL
L125............int   P:001C68          FEPcom_c         REL LOCAL
L126............int   P:001D03          FEPcom_c         REL LOCAL
L127............int   P:001C0C          FEPcom_c         REL LOCAL
L128............int   P:001E04          FEPcom_c         REL LOCAL
L129............int   P:001E17          FEPcom_c         REL LOCAL
L13.............int   P:000AEB          FEPmain_c        REL LOCAL
L13.............int   P:00105B          FEPvid_c         REL LOCAL
L13.............int   P:0023E6          ippshell_c       REL LOCAL
L13.............int   P:002801          maloc_c          REL LOCAL
L130............int   P:001E42          FEPcom_c         REL LOCAL
L131............int   P:001E91          FEPcom_c         REL LOCAL
L133............int   P:001EAB          FEPcom_c         REL LOCAL
L136............int   P:001EDC          FEPcom_c         REL LOCAL
L137............int   P:001FF8          FEPcom_c         REL LOCAL
L138............int   P:001F47          FEPcom_c         REL LOCAL
L139............int   P:001EF0          FEPcom_c         REL LOCAL
L14.............int   P:00106C          FEPvid_c         REL LOCAL
L14.............int   P:0023D4          ippshell_c       REL LOCAL
L142............int   P:001F44          FEPcom_c         REL LOCAL
L144............int   P:001F05          FEPcom_c         REL LOCAL
L146............int   P:001F74          FEPcom_c         REL LOCAL
L147............int   P:001FD6          FEPcom_c         REL LOCAL
L149............int   P:001FF3          FEPcom_c         REL LOCAL
L15.............int   P:000AE4          FEPmain_c        REL LOCAL
L15.............int   P:001086          FEPvid_c         REL LOCAL
L15.............int   P:0023E6          ippshell_c       REL LOCAL
L15.............int   P:0027F0          maloc_c          REL LOCAL
L150............int   P:00208C          FEPcom_c         REL LOCAL
L153............int   P:002042          FEPcom_c         REL LOCAL
L157............int   P:001F45          FEPcom_c         REL LOCAL
L158............int   P:001FB7          FEPcom_c         REL LOCAL
L159............int   P:002088          FEPcom_c         REL LOCAL
L16.............int   P:001129          FEPvid_c         REL LOCAL
L16.............int   P:0023E1          ippshell_c       REL LOCAL
L16.............int   P:002860          maloc_c          REL LOCAL
```



```
L161............int   P:0020CE          FEPcom_c         REL LOCAL
L162............int   P:0020F5          FEPcom_c         REL LOCAL
L163............int   P:00211E          FEPcom_c         REL LOCAL
L164............int   P:0020C9          FEPcom_c         REL LOCAL
L165............int   P:0020F0          FEPcom_c         REL LOCAL
L166............int   P:002119          FEPcom_c         REL LOCAL
L168............int   P:002158          FEPcom_c         REL LOCAL
L17.............int   P:0010B7          FEPvid_c         REL LOCAL
L17.............int   P:0023E3          ippshell_c       REL LOCAL
L170............int   P:002197          FEPcom_c         REL LOCAL
L172............int   P:0021DC          FEPcom_c         REL LOCAL
L174............int   P:002212          FEPcom_c         REL LOCAL
L176............int   P:00222B          FEPcom_c         REL LOCAL
L178............int   P:00224B          FEPcom_c         REL LOCAL
L18.............int   P:0010CB          FEPvid_c         REL LOCAL
L18.............int   P:00184E          FEPcom_c         REL LOCAL
L18.............int   P:002846          maloc_c          REL LOCAL
L182............int   P:002292          FEPcom_c         REL LOCAL
L183............int   P:00225F          FEPcom_c         REL LOCAL
L184............int   P:0022B7          FEPcom_c         REL LOCAL
L185............int   P:0022AC          FEPcom_c         REL LOCAL
L186............int   P:002216          FEPcom_c         REL LOCAL
L187............int   P:00222D          FEPcom_c         REL LOCAL
L188............int   P:0022D1          FEPcom_c         REL LOCAL
L19.............int   P:002430          ippshell_c       REL LOCAL
L19.............int   P:0027D3          maloc_c          REL LOCAL
L192............int   P:002301          FEPcom_c         REL LOCAL
L193............int   P:002314          FEPcom_c         REL LOCAL
L2..............int   P:000AE6          FEPmain_c        REL LOCAL
L2..............int   P:000ECE          FEPsetvars_c     REL LOCAL
L2..............int   P:000FE7          FEPvid_c         REL LOCAL
L2..............int   P:0016E3          FEPcom_c         REL LOCAL
L2..............int   P:002369          ippshell_c       REL LOCAL
L2..............int   P:00272B          buddy_chk_c      REL LOCAL
L2..............int   P:0027A1          maloc_c          REL LOCAL
L20.............int   P:000B02          FEPmain_c        REL LOCAL
L20.............int   P:0010DC          FEPvid_c         REL LOCAL
L20.............int   P:001856          FEPcom_c         REL LOCAL
L20.............int   P:002423          ippshell_c       REL LOCAL
L20.............int   P:00281D          maloc_c          REL LOCAL
L21.............int   P:0010EC          FEPvid_c         REL LOCAL
L21.............int   P:0018C7          FEPcom_c         REL LOCAL
L21.............int   P:0024A0          ippshell_c       REL LOCAL
L21.............int   P:002878          maloc_c          REL LOCAL
L22.............int   P:0010F6          FEPvid_c         REL LOCAL
L22.............int   P:00246C          ippshell_c       REL LOCAL
L22.............int   P:0028E2          maloc_c          REL LOCAL
L23.............int   P:00110B          FEPvid_c         REL LOCAL
L23.............int   P:001A02          FEPcom_c         REL LOCAL
L23.............int   P:00247A          ippshell_c       REL LOCAL
L24.............int   P:000B2B          FEPmain_c        REL LOCAL
L24.............int   P:001902          FEPcom_c         REL LOCAL
L24.............int   P:00251E          ippshell_c       REL LOCAL
L24.............int   P:0028AC          maloc_c          REL LOCAL
L25.............int   P:00190A          FEPcom_c         REL LOCAL
L25.............int   P:0024DC          ippshell_c       REL LOCAL
L25.............int   P:0028C1          maloc_c          REL LOCAL
L26.............int   P:000B19          FEPmain_c        REL LOCAL
L26.............int   P:0024EA          ippshell_c       REL LOCAL
L26.............int   P:0028D0          maloc_c          REL LOCAL
```



```
L27.............int   P:000B2E          FEPmain_c        REL LOCAL
L27.............int   P:0010A5          FEPvid_c         REL LOCAL
L27.............int   P:001912          FEPcom_c         REL LOCAL
L27.............int   P:0025A7          ippshell_c       REL LOCAL
```

```
L27.............int   P:0028CD          maloc_c          REL LOCAL
L28.............int   P:000B21          FEPmain_c        REL LOCAL
L28.............int   P:0010D3          FEPvid_c         REL LOCAL
L28.............int   P:001918          FEPcom_c         REL LOCAL
L28.............int   P:002567          ippshell_c       REL LOCAL
L29.............int   P:00111E          FEPvid_c         REL LOCAL
L29.............int   P:00191E          FEPcom_c         REL LOCAL
L29.............int   P:002575          ippshell_c       REL LOCAL
L29.............int   P:002904          maloc_c          REL LOCAL
L3..............int   P:000ED1          FEPsetvars_c     REL LOCAL
L3..............int   P:0016F0          FEPcom_c         REL LOCAL
L3..............int   P:00235E          ippshell_c       REL LOCAL
L3..............int   P:00277C          buddy_chk_c      REL LOCAL
L3..............int   P:00295E          brk_c            REL LOCAL
L30.............int   P:00111B          FEPvid_c         REL LOCAL
L30.............int   P:001934          FEPcom_c         REL LOCAL
L30.............int   P:0025A5          ippshell_c       REL LOCAL
L31.............int   P:00194A          FEPcom_c         REL LOCAL
L31.............int   P:0025E0          ippshell_c       REL LOCAL
L32.............int   P:001217          FEPvid_c         REL LOCAL
L32.............int   P:00195D          FEPcom_c         REL LOCAL
L32.............int   P:0025D2          ippshell_c       REL LOCAL
L33.............int   P:000B4C          FEPmain_c        REL LOCAL
L33.............int   P:00116C          FEPvid_c         REL LOCAL
L33.............int   P:0025D6          ippshell_c       REL LOCAL
L34.............int   P:001191          FEPvid_c         REL LOCAL
L34.............int   P:0025DE          ippshell_c       REL LOCAL
L35.............int   P:000B4E          FEPmain_c        REL LOCAL
L36.............int   P:0011AC          FEPvid_c         REL LOCAL
L36.............int   P:00196A          FEPcom_c         REL LOCAL
L37.............int   P:000B85          FEPmain_c        REL LOCAL
L37.............int   P:0011C7          FEPvid_c         REL LOCAL
L37.............int   P:00198B          FEPcom_c         REL LOCAL
L38.............int   P:000B8C          FEPmain_c        REL LOCAL
L38.............int   P:0011E6          FEPvid_c         REL LOCAL
L39.............int   P:000C2A          FEPmain_c        REL LOCAL
L4..............int   P:00235A          ippshell_c       REL LOCAL
L4..............int   P:002766          buddy_chk_c      REL LOCAL
L4..............int   P:002922          mcpy_c           REL LOCAL
L40.............int   P:000C16          FEPmain_c        REL LOCAL
L41.............int   P:000C27          FEPmain_c        REL LOCAL
L41.............int   P:00115C          FEPvid_c         REL LOCAL
L41.............int   P:00199B          FEPcom_c         REL LOCAL
L42.............int   P:0011E0          FEPvid_c         REL LOCAL
L42.............int   P:0019CE          FEPcom_c         REL LOCAL
L43.............int   P:0019DA          FEPcom_c         REL LOCAL
L44.............int   P:001239          FEPvid_c         REL LOCAL
L44.............int   P:0019E0          FEPcom_c         REL LOCAL
L45.............int   P:000C4F          FEPmain_c        REL LOCAL
L45.............int   Y:00012E          FEPcom_c         REL LOCAL
L46.............int   P:001344          FEPvid_c         REL LOCAL
L46.............int   P:0019BD          FEPcom_c         REL LOCAL
L47.............int   P:000C43          FEPmain_c        REL LOCAL
L47.............int   P:001357          FEPvid_c         REL LOCAL
L47.............int   P:001980          FEPcom_c         REL LOCAL
L48.............int   P:0019E6          FEPcom_c         REL LOCAL
```



```
L49.............int   P:000C50          FEPmain_c        REL LOCAL
L49.............int   P:001293          FEPvid_c         REL LOCAL
L49.............int   P:0019E9          FEPcom_c         REL LOCAL
L5..............int   P:000FDE          FEPvid_c         REL LOCAL
L5..............int   P:001725          FEPcom_c         REL LOCAL
L5..............int   P:00235E          ippshell_c       REL LOCAL
L5..............int   P:002771          buddy_chk_c      REL LOCAL
L5..............int   P:0027CB          maloc_c          REL LOCAL
L5..............int   P:00295C          brk_c            REL LOCAL
L50.............int   P:00192D          FEPcom_c         REL LOCAL
L51.............int   P:000C6C          FEPmain_c        REL LOCAL
L51.............int   P:0012B8          FEPvid_c         REL LOCAL
L51.............int   P:001943          FEPcom_c         REL LOCAL
L52.............int   P:0019FD          FEPcom_c         REL LOCAL
L53.............int   P:001B66          FEPcom_c         REL LOCAL
L54.............int   P:0012D4          FEPvid_c         REL LOCAL
L55.............int   P:001A8D          FEPcom_c         REL LOCAL
L56.............int   P:001A69          FEPcom_c         REL LOCAL
L57.............int   P:000CA8          FEPmain_c        REL LOCAL
L57.............int   P:0012F7          FEPvid_c         REL LOCAL
L58.............int   P:000CB9          FEPmain_c        REL LOCAL
L58.............int   P:001308          FEPvid_c         REL LOCAL
L59.............int   P:000CB8          FEPmain_c        REL LOCAL
L6..............int   P:000AE2          FEPmain_c        REL LOCAL
L6..............int   P:001727          FEPcom_c         REL LOCAL
L6..............int   P:002369          ippshell_c       REL LOCAL
L6..............int   P:0027AA          maloc_c          REL LOCAL
L6..............int   P:002924          mcpy_c           REL LOCAL
L6..............int   P:002954          brk_c            REL LOCAL
L60.............int   P:001323          FEPvid_c         REL LOCAL
L60.............int   P:001B15          FEPcom_c         REL LOCAL
L61.............int   P:001AD8          FEPcom_c         REL LOCAL
L62.............int   P:001AE6          FEPcom_c         REL LOCAL
L63.............int   P:001AEA          FEPcom_c         REL LOCAL
L64.............int   P:000CEC          FEPmain_c        REL LOCAL
L64.............int   P:001287          FEPvid_c         REL LOCAL
L64.............int   P:001AEE          FEPcom_c         REL LOCAL
L65.............int   P:000CFC          FEPmain_c        REL LOCAL
L65.............int   P:001275          FEPvid_c         REL LOCAL
L65.............int   P:001AF2          FEPcom_c         REL LOCAL
L66.............int   P:000CFB          FEPmain_c        REL LOCAL
L66.............int   P:001AF6          FEPcom_c         REL LOCAL
L67.............int   P:001342          FEPvid_c         REL LOCAL
L67.............int   P:001B00          FEPcom_c         REL LOCAL
L68.............int   P:00131C          FEPvid_c         REL LOCAL
L7..............int   P:0023AA          ippshell_c       REL LOCAL
L7..............int   P:0027E3          maloc_c          REL LOCAL
L7..............int   P:002987          brk_c            REL LOCAL
L70.............int   P:0013C4          FEPvid_c         REL LOCAL
L71.............int   P:000D2F          FEPmain_c        REL LOCAL
L71.............int   P:0013C2          FEPvid_c         REL LOCAL
L71.............int   Y:00013D          FEPcom_c         REL LOCAL
L72.............int   P:000D40          FEPmain_c        REL LOCAL
L73.............int   P:000D3F          FEPmain_c        REL LOCAL
L73.............int   P:001AC7          FEPcom_c         REL LOCAL
L74.............int   P:001421          FEPvid_c         REL LOCAL
L74.............int   P:001B64          FEPcom_c         REL LOCAL
L75.............int   P:000D74          FEPmain_c        REL LOCAL
L75.............int   P:001A64          FEPcom_c         REL LOCAL
L76.............int   P:000DB6          FEPmain_c        REL LOCAL
```



```
L76.............int   P:001A7F          FEPcom_c         REL LOCAL
L77.............int   P:000D8B          FEPmain_c        REL LOCAL
L77.............int   P:0014C4          FEPvid_c         REL LOCAL
L77.............int   P:001E96          FEPcom_c         REL LOCAL
L78.............int   P:000D9A          FEPmain_c        REL LOCAL
L78.............int   P:001BA5          FEPcom_c         REL LOCAL
```

```
L79.............int   P:000DA9          FEPmain_c         REL LOCAL
L79.............int   P:00144C          FEPvid_c          REL LOCAL
L8..............int   P:000F4A          FEPsetvars_c      REL LOCAL
L8..............int   P:001758          FEPcom_c          REL LOCAL
L8..............int   P:0023A0          ippshell_c        REL LOCAL
L8..............int   P:002985          brk_c             REL LOCAL
L82.............int   P:000D83          FEPmain_c         REL LOCAL
L82.............int   P:001491          FEPvid_c          REL LOCAL
L82.............int   P:001BE3          FEPcom_c          REL LOCAL
L83.............int   P:000DDB          FEPmain_c         REL LOCAL
L83.............int   P:0014AF          FEPvid_c          REL LOCAL
L83.............int   P:001C0E          FEPcom_c          REL LOCAL
L84.............int   P:001C17          FEPcom_c          REL LOCAL
L85.............int   P:000E28          FEPmain_c         REL LOCAL
L85.............int   P:00141A          FEPvid_c          REL LOCAL
L85.............int   P:001C48          FEPcom_c          REL LOCAL
L86.............int   P:000E21          FEPmain_c         REL LOCAL
L87.............int   P:000E24          FEPmain_c         REL LOCAL
L88.............int   P:000E27          FEPmain_c         REL LOCAL
L88.............int   P:001521          FEPvid_c          REL LOCAL
L89.............int   P:001501          FEPvid_c          REL LOCAL
L89.............int   P:001C66          FEPcom_c          REL LOCAL
L9..............int   P:001014          FEPvid_c          REL LOCAL
L9..............int   P:0023AA          ippshell_c        REL LOCAL
L91.............int   P:000E19          FEPmain_c         REL LOCAL
L91.............int   P:00156A          FEPvid_c          REL LOCAL
L92.............int   P:00155E          FEPvid_c          REL LOCAL
L93.............int   P:00155C          FEPvid_c          REL LOCAL
L94.............int   P:000E8A          FEPmain_c         REL LOCAL
L94.............int   P:001534          FEPvid_c          REL LOCAL
L94.............int   P:001C7E          FEPcom_c          REL LOCAL
L95.............int   P:001CAD          FEPcom_c          REL LOCAL
L96.............int   P:001CE3          FEPcom_c          REL LOCAL
L97.............int   P:00160F          FEPvid_c          REL LOCAL
L97.............int   P:001CDF          FEPcom_c          REL LOCAL
L98.............int   P:000EA8          FEPmain_c         REL LOCAL
L98.............int   P:0015C9          FEPvid_c          REL LOCAL
L99.............int   P:000EA7          FEPmain_c         REL LOCAL
L99.............int   P:0015DA          FEPvid_c          REL LOCAL
LBOX_QUEUE_BOTTO.int     001000          GLOBAL            ABS GLOBAL
LSIZE...........int     0009A2          GLOBAL            ABS GLOBAL
MAXNUMEVENTS....int     002EE0          GLOBAL            ABS GLOBAL
M_BCR...........int     00FFFE          ipp_driver        ABS LOCAL
M_BCR...........int     00FFFE          RTC_handler       ABS LOCAL
M_CD............int     000FFF          ipp_driver        ABS LOCAL
M_CD............int     000FFF          RTC_handler       ABS LOCAL
M_COD...........int     00000C          ipp_driver        ABS LOCAL
M_COD...........int     00000C          RTC_handler       ABS LOCAL
M_CRA...........int     00FFEC          ipp_driver        ABS LOCAL
M_CRA...........int     00FFEC          RTC_handler       ABS LOCAL
M_CRB...........int     00FFED          ipp_driver        ABS LOCAL
M_CRB...........int     00FFED          RTC_handler       ABS LOCAL
M_DC............int     001F00          ipp_driver        ABS LOCAL
M_DC............int     001F00          RTC_handler       ABS LOCAL
M_GCK...........int     00000A          RTC_handler       ABS LOCAL
M_HCIE..........int     000002          ipp_driver        ABS LOCAL
M_HCIE..........int     000002          RTC_handler       ABS LOCAL
M_HCP...........int     000002          ipp_driver        ABS LOCAL
M_HCP...........int     000002          RTC_handler       ABS LOCAL
M_HCR...........int     00FFE8          ipp_driver        ABS LOCAL
M_HCR...........int     00FFE8          RTC_handler       ABS LOCAL
M_HF............int     000018          ipp_driver        ABS LOCAL
M_HF............int     000018          RTC_handler       ABS LOCAL
M_HF0...........int     000003          ipp_driver        ABS LOCAL
M_HF0...........int     000003          RTC_handler       ABS LOCAL
M_HF1...........int     000004          ipp_driver        ABS LOCAL
M_HF1...........int     000004          RTC_handler       ABS LOCAL
M_HF2...........int     000003          ipp_driver        ABS LOCAL
M_HF2...........int     000003          RTC_handler       ABS LOCAL
M_HF3...........int     000004          ipp_driver        ABS LOCAL
M_HF3...........int     000004          RTC_handler       ABS LOCAL
M_HPL...........int     000C00          ipp_driver        ABS LOCAL
M_HPL...........int     000C00          RTC_handler       ABS LOCAL
M_HPL0..........int     00000A          ipp_driver        ABS LOCAL
M_HPL0..........int     00000A          RTC_handler       ABS LOCAL
M_HPL1..........int     00000B          ipp_driver        ABS LOCAL
M_HPL1..........int     00000B          RTC_handler       ABS LOCAL
M_HRDF..........int     000000          ipp_driver        ABS LOCAL
M_HRDF..........int     000000          RTC_handler       ABS LOCAL
M_HRIE..........int     000000          ipp_driver        ABS LOCAL
M_HRIE..........int     000000          RTC_handler       ABS LOCAL
M_HRX...........int     00FFEB          ipp_driver        ABS LOCAL
M_HRX...........int     00FFEB          RTC_handler       ABS LOCAL
M_HSR...........int     00FFE9          ipp_driver        ABS LOCAL
M_HSR...........int     00FFE9          RTC_handler       ABS LOCAL
M_HTDE..........int     000001          ipp_driver        ABS LOCAL
M_HTDE..........int     000001          RTC_handler       ABS LOCAL
M_HTIE..........int     000001          ipp_driver        ABS LOCAL
M_HTIE..........int     000001          RTC_handler       ABS LOCAL
M_HTX...........int     00FFEB          ipp_driver        ABS LOCAL
M_HTX...........int     00FFEB          RTC_handler       ABS LOCAL
M_IAL...........int     000007          ipp_driver        ABS LOCAL
M_IAL...........int     000007          RTC_handler       ABS LOCAL
M_IAL0..........int     000000          ipp_driver        ABS LOCAL
M_IAL0..........int     000000          RTC_handler       ABS LOCAL
M_IAL1..........int     000001          ipp_driver        ABS LOCAL
M_IAL1..........int     000001          RTC_handler       ABS LOCAL
M_IAL2..........int     000002          ipp_driver        ABS LOCAL
M_IAL2..........int     000002          RTC_handler       ABS LOCAL
M_IBL...........int     000038          ipp_driver        ABS LOCAL
M_IBL...........int     000038          RTC_handler       ABS LOCAL
M_IBL0..........int     000003          ipp_driver        ABS LOCAL
M_IBL0..........int     000003          RTC_handler       ABS LOCAL
M_IBL1..........int     000004          ipp_driver        ABS LOCAL
M_IBL1..........int     000004          RTC_handler       ABS LOCAL
M_IBL2..........int     000005          ipp_driver        ABS LOCAL
M_IBL2..........int     000005          RTC_handler       ABS LOCAL
```



```
M_DMA...........int     000007          ipp_driver        ABS LOCAL
M_DMA...........int     000007          RTC_handler       ABS LOCAL
M_FE............int     000006          ipp_driver        ABS LOCAL
M_FE............int     000006          RTC_handler       ABS LOCAL
M_FSL...........int     000008          ipp_driver        ABS LOCAL
M_FSL...........int     000008          RTC_handler       ABS LOCAL
M_GCK...........int     00000A          ipp_driver        ABS LOCAL
```



```
M_IDLE..........int     000003          ipp_driver        ABS LOCAL
M_IDLE..........int     000003          RTC_handler       ABS LOCAL
M_IF............int     000002          ipp_driver        ABS LOCAL
M_IF............int     000002          RTC_handler       ABS LOCAL
M_IF0...........int     000000          ipp_driver        ABS LOCAL
M_IF0...........int     000000          RTC_handler       ABS LOCAL
M_IF1...........int     000001          ipp_driver        ABS LOCAL
M_IF1...........int     000001          RTC_handler       ABS LOCAL
```

M_ILIE	int	00000A	ipp_driver	ABS LOCAL	M_SCD1	int	000003	RTC_handler	ABS LOCAL
M_ILIE	int	00000A	RTC_handler	ABS LOCAL	M_SCD2	int	000004	ipp_driver	ABS LOCAL
M_IPR	int	00FFFF	ipp_driver	ABS LOCAL	M_SCD2	int	000004	RTC_handler	ABS LOCAL
M_IPR	int	00FFFF	RTC_handler	ABS LOCAL	M_SCKD	int	000005	ipp_driver	ABS LOCAL
M_MOD	int	00000B	ipp_driver	ABS LOCAL	M_SCKD	int	000005	RTC_handler	ABS LOCAL
M_MOD	int	00000B	RTC_handler	ABS LOCAL	M_SCL	int	00C000	ipp_driver	ABS LOCAL
M_OF	int	000002	ipp_driver	ABS LOCAL	M_SCL	int	00C000	RTC_handler	ABS LOCAL
M_OF	int	000002	RTC_handler	ABS LOCAL	M_SCL0	int	00000E	ipp_driver	ABS LOCAL
M_OF0	int	000000	ipp_driver	ABS LOCAL	M_SCL0	int	00000E	RTC_handler	ABS LOCAL
M_OF0	int	000000	RTC_handler	ABS LOCAL	M_SCL1	int	00000F	ipp_driver	ABS LOCAL
M_OF1	int	000001	ipp_driver	ABS LOCAL	M_SCL1	int	00000F	RTC_handler	ABS LOCAL
M_OF1	int	000001	RTC_handler	ABS LOCAL	M_SCP	int	00000D	ipp_driver	ABS LOCAL
M_OR	int	000004	ipp_driver	ABS LOCAL	M_SCP	int	00000D	RTC_handler	ABS LOCAL
M_OR	int	000004	RTC_handler	ABS LOCAL	M_SCR	int	00FFF0	ipp_driver	ABS LOCAL
M_PBC	int	00FFE0	ipp_driver	ABS LOCAL	M_SCR	int	00FFF0	RTC_handler	ABS LOCAL
M_PBC	int	00FFE0	RTC_handler	ABS LOCAL	M_SR	int	00FFEE	ipp_driver	ABS LOCAL
M_PBD	int	00FFE4	ipp_driver	ABS LOCAL	M_SR	int	00FFEE	RTC_handler	ABS LOCAL
M_PBD	int	00FFE4	RTC_handler	ABS LOCAL	M_SRE	int	00000D	ipp_driver	ABS LOCAL
M_PBDDR	int	00FFE2	ipp_driver	ABS LOCAL	M_SRE	int	00000D	RTC_handler	ABS LOCAL
M_PBDDR	int	00FFE2	RTC_handler	ABS LOCAL	M_SRIE	int	00000F	ipp_driver	ABS LOCAL
M_PCC	int	00FFE1	ipp_driver	ABS LOCAL	M_SRIE	int	00000F	RTC_handler	ABS LOCAL
M_PCC	int	00FFE1	RTC_handler	ABS LOCAL	M_SRXH	int	00FFF6	ipp_driver	ABS LOCAL
M_PCD	int	00FFE5	ipp_driver	ABS LOCAL	M_SRXH	int	00FFF6	RTC_handler	ABS LOCAL
M_PCD	int	00FFE5	RTC_handler	ABS LOCAL	M_SRXL	int	00FFF4	ipp_driver	ABS LOCAL
M_PCDDR	int	00FFE3	ipp_driver	ABS LOCAL	M_SRXL	int	00FFF4	RTC_handler	ABS LOCAL
M_PCDDR	int	00FFE3	RTC_handler	ABS LOCAL	M_SRXM	int	00FFF5	ipp_driver	ABS LOCAL
M_PE	int	000005	ipp_driver	ABS LOCAL	M_SRXM	int	00FFF5	RTC_handler	ABS LOCAL
M_PE	int	000005	RTC_handler	ABS LOCAL	M_SSL	int	003000	ipp_driver	ABS LOCAL
M_PM	int	0000FF	ipp_driver	ABS LOCAL	M_SSL	int	003000	RTC_handler	ABS LOCAL
M_PM	int	0000FF	RTC_handler	ABS LOCAL	M_SSL0	int	00000C	ipp_driver	ABS LOCAL
M_PSR	int	00000F	ipp_driver	ABS LOCAL	M_SSL0	int	00000C	RTC_handler	ABS LOCAL
M_PSR	int	00000F	RTC_handler	ABS LOCAL	M_SSL1	int	00000D	ipp_driver	ABS LOCAL
M_R8	int	000007	ipp_driver	ABS LOCAL	M_SSL1	int	00000D	RTC_handler	ABS LOCAL
M_R8	int	000007	RTC_handler	ABS LOCAL	M_SSR	int	00FFF1	ipp_driver	ABS LOCAL
M_RCM	int	00000E	ipp_driver	ABS LOCAL	M_SSR	int	00FFF1	RTC_handler	ABS LOCAL
M_RCM	int	00000E	RTC_handler	ABS LOCAL	M_STE	int	00000C	ipp_driver	ABS LOCAL
M_RDF	int	000007	ipp_driver	ABS LOCAL	M_STE	int	00000C	RTC_handler	ABS LOCAL
M_RDF	int	000007	RTC_handler	ABS LOCAL	M_STIE	int	00000E	ipp_driver	ABS LOCAL
M_RDRF	int	000002	ipp_driver	ABS LOCAL	M_STIE	int	00000E	RTC_handler	ABS LOCAL
M_RDRF	int	000002	RTC_handler	ABS LOCAL	M_STXA	int	00FFF3	ipp_driver	ABS LOCAL
M_RE	int	000008	ipp_driver	ABS LOCAL	M_STXA	int	00FFF3	RTC_handler	ABS LOCAL
M_RE	int	000008	RTC_handler	ABS LOCAL	M_STXH	int	00FFF6	ipp_driver	ABS LOCAL
M_RFS	int	000003	ipp_driver	ABS LOCAL	M_STXH	int	00FFF6	RTC_handler	ABS LOCAL
M_RFS	int	000003	RTC_handler	ABS LOCAL	M_STXL	int	00FFF4	ipp_driver	ABS LOCAL
M_RIE	int	00000B	ipp_driver	ABS LOCAL	M_STXL	int	00FFF4	RTC_handler	ABS LOCAL
M_RIE	int	00000B	RTC_handler	ABS LOCAL	M_STXM	int	00FFF5	ipp_driver	ABS LOCAL
M_ROE	int	000005	ipp_driver	ABS LOCAL	M_STXM	int	00FFF5	RTC_handler	ABS LOCAL
M_ROE	int	000005	RTC_handler	ABS LOCAL	M_SYN	int	000009	ipp_driver	ABS LOCAL
M_RWI	int	000006	ipp_driver	ABS LOCAL	M_SYN	int	000009	RTC_handler	ABS LOCAL
M_RWI	int	000006	RTC_handler	ABS LOCAL	M_TCM	int	00000F	ipp_driver	ABS LOCAL
M_RX	int	00FFEF	ipp_driver	ABS LOCAL	M_TCM	int	00000F	RTC_handler	ABS LOCAL
M_RX	int	00FFEF	RTC_handler	ABS LOCAL					

M_SBK	int	000004	ipp_driver	ABS LOCAL	M_TDE	int	000006	ipp_driver	ABS LOCAL
M_SBK	int	000004	RTC_handler	ABS LOCAL	M_TDE	int	000006	RTC_handler	ABS LOCAL
M_SCCR	int	00FFF2	ipp_driver	ABS LOCAL	M_TDRE	int	000001	ipp_driver	ABS LOCAL
M_SCCR	int	00FFF2	RTC_handler	ABS LOCAL	M_TDRE	int	000001	RTC_handler	ABS LOCAL
M_SCD	int	00001C	ipp_driver	ABS LOCAL	M_TE	int	000009	ipp_driver	ABS LOCAL
M_SCD	int	00001C	RTC_handler	ABS LOCAL	M_TE	int	000009	RTC_handler	ABS LOCAL
M_SCD0	int	000002	ipp_driver	ABS LOCAL	M_TFS	int	000002	ipp_driver	ABS LOCAL
M_SCD0	int	000002	RTC_handler	ABS LOCAL	M_TFS	int	000002	RTC_handler	ABS LOCAL
M_SCD1	int	000003	ipp_driver	ABS LOCAL	M_TIE	int	00000C	ipp_driver	ABS LOCAL
					M_TIE	int	00000C	RTC_handler	ABS LOCAL

```
M_TMIE.....int     00000D          ipp_driver      ABS LOCAL
M_TMIE.....int     00000D          RTC_handler     ABS LOCAL
M_TRNE.....int     000000          ipp_driver      ABS LOCAL
M_TRNE.....int     000000          RTC_handler     ABS LOCAL
M_TSR......int     00FFEE          ipp_driver      ABS LOCAL
M_TSR......int     00FFEE          RTC_handler     ABS LOCAL
M_TUE......int     000004          ipp_driver      ABS LOCAL
M_TUE......int     000004          RTC_handler     ABS LOCAL
M_TX.......int     00FFEF          ipp_driver      ABS LOCAL
M_TX.......int     00FFEF          RTC_handler     ABS LOCAL
M_WAKE.....int     000005          ipp_driver      ABS LOCAL
M_WAKE.....int     000005          RTC_handler     ABS LOCAL
M_WDS......int     000003          ipp_driver      ABS LOCAL
M_WDS......int     000003          RTC_handler     ABS LOCAL
M_WDS0.....int     000000          ipp_driver      ABS LOCAL
M_WDS0.....int     000000          RTC_handler     ABS LOCAL
M_WDS1.....int     000001          ipp_driver      ABS LOCAL
M_WDS1.....int     000001          RTC_handler     ABS LOCAL
M_WDS2.....int     000002          ipp_driver      ABS LOCAL
M_WDS2.....int     000002          RTC_handler     ABS LOCAL
M_WL.......int     006000          ipp_driver      ABS LOCAL
M_WL.......int     006000          RTC_handler     ABS LOCAL
M_WL0......int     00000D          ipp_driver      ABS LOCAL
M_WL0......int     00000D          RTC_handler     ABS LOCAL
M_WL1......int     00000E          ipp_driver      ABS LOCAL
M_WL1......int     00000E          RTC_handler     ABS LOCAL
M_WOMS.....int     000007          ipp_driver      ABS LOCAL
M_WOMS.....int     000007          RTC_handler     ABS LOCAL
Mbps.......fpt     5.000000E-001   GLOBAL          ABS GLOBAL
NO_MODE....int     000000          ipp_driver      ABS LOCAL
PSIZE......int     002990          GLOBAL          ABS GLOBAL
REQ_ABT....int     000003          ipp_driver      ABS LOCAL
REQ_PORT...int     00FFE9          ipp_driver      ABS LOCAL
REQ_TD.....int     000004          ipp_driver      ABS LOCAL
RX_even....int   P:002632          ipp_driver      REL LOCAL
SCI_RXQ_SIZE...int 000040          GLOBAL          ABS GLOBAL
SHIFT......int     000000          GLOBAL          ABS GLOBAL
SHORT_JSR_INSTR..int 0D0000        GLOBAL          ABS GLOBAL
SSD1BIT....int     000005          GLOBAL          ABS GLOBAL
SSD1BIT....int     000005          ipp_driver      ABS LOCAL
SSD1BIT....int     000005          RTC_handler     ABS LOCAL
SSD2BIT....int     000006          GLOBAL          ABS GLOBAL
SSD2BIT....int     000006          ipp_driver      ABS LOCAL
SSD2BIT....int     000006          RTC_handler     ABS LOCAL
SSI_RX_INSTR...int 085DAF          GLOBAL          ABS GLOBAL
SSI_TXQ_SIZE...int 000400          GLOBAL          ABS GLOBAL
STACK_START....int 001440          GLOBAL          ABS GLOBAL
TOP_OF_MEMORY..int 007FFF          GLOBAL          ABS GLOBAL
TX_even....int   P:00266E          ipp_driver      REL LOCAL
VIDBUF_BOTTOM..int 000000          GLOBAL          ABS GLOBAL
```



```
VIDBUF_LEN.....int 002000          GLOBAL          ABS GLOBAL
VIDBUF_TOP.....int 002000          GLOBAL          ABS GLOBAL
WAITDIS_IRQFST..int 000000         GLOBAL          ABS GLOBAL
WAITDIS_IRQFST..int 000000         ipp_driver      ABS LOCAL
WAITDIS_IRQFST..int 000000         RTC_handler     ABS LOCAL
WAITDIS_IRQSLW..int 000010         GLOBAL          ABS GLOBAL
WAITDIS_IRQSLW..int 000010         ipp_driver      ABS LOCAL
WAITDIS_IRQSLW..int 000010         RTC_handler     ABS LOCAL
WAITENA_IRQFST..int 000002         GLOBAL          ABS GLOBAL
WAITENA_IRQFST..int 000002         ipp_driver      ABS LOCAL
WAITENA_IRQFST..int 000002         RTC_handler     ABS LOCAL
WAITENA_IRQSLW..int 000012         GLOBAL          ABS GLOBAL
WAITENA_IRQSLW..int 000012         ipp_driver      ABS LOCAL
WAITENA_IRQSLW..int 000012         RTC_handler     ABS LOCAL
WAIT_DISABLE....int 000000         GLOBAL          ABS GLOBAL
WAIT_DISABLE....int 000000         ipp_driver      ABS LOCAL
WAIT_DISABLE....int 000000         RTC_handler     ABS LOCAL
WAIT_DISSTAT_BIT.int 000000        GLOBAL          ABS GLOBAL
WAIT_DISSTAT_BIT.int 000000        ipp_driver      ABS LOCAL
WAIT_DISSTAT_BIT.int 000000        RTC_handler     ABS LOCAL
WAIT_ENABLE.....int 000002         GLOBAL          ABS GLOBAL
WAIT_ENABLE.....int 000002         ipp_driver      ABS LOCAL
WAIT_ENABLE.....int 000002         RTC_handler     ABS LOCAL
XSIZE......int     000000          GLOBAL          ABS GLOBAL
YSIZE......int     0009A2          GLOBAL          ABS GLOBAL
abort_RX...int   P:00263C          ipp_driver      REL LOCAL
abort_TX...int   P:002678          ipp_driver      REL LOCAL
compare....int   P:00013A          proc_line       ABS LOCAL
divider....int     000009          GLOBAL          ABS GLOBAL
dma_state..int   Y:00098C          ipp_driver      REL LOCAL
eventspace.int   X:002000          GLOBAL          ABS GLOBAL
exit_RX....int   P:002642          ipp_driver      REL LOCAL
exit_TX....int   P:002680          ipp_driver      REL LOCAL
ge_thresh_16...int P:000109        proc_line       ABS LOCAL
ge_thresh_hi...int P:000102        proc_line       ABS LOCAL
ge_thresh_lo...int P:000107        proc_line       ABS LOCAL
getdata....int   P:002639          ipp_driver      REL LOCAL
gppReqDTISR....int P:002607        ipp_driver      REL LOCAL
higher_exists..int P:000136        proc_line       ABS LOCAL
its_dsp_zero...int P:0025FC        ipp_driver      REL LOCAL
m_bcr......int     00FFFE          GLOBAL          ABS GLOBAL
m_cd.......int     000FFF          GLOBAL          ABS GLOBAL
m_cod......int     00000C          GLOBAL          ABS GLOBAL
m_cra......int     00FFEC          GLOBAL          ABS GLOBAL
m_crb......int     00FFED          GLOBAL          ABS GLOBAL
m_dc.......int     001F00          GLOBAL          ABS GLOBAL
m_dma......int     000007          GLOBAL          ABS GLOBAL
m_fe.......int     000006          GLOBAL          ABS GLOBAL
m_fsl......int     000008          GLOBAL          ABS GLOBAL
m_gck......int     00000A          GLOBAL          ABS GLOBAL
m_hcie.....int     000002          GLOBAL          ABS GLOBAL
m_hcp......int     000002          GLOBAL          ABS GLOBAL
m_hcr......int     00FFE8          GLOBAL          ABS GLOBAL
m_hf.......int     000018          GLOBAL          ABS GLOBAL
m_hf0......int     000003          GLOBAL          ABS GLOBAL
m_hf1......int     000004          GLOBAL          ABS GLOBAL
m_hf2......int     000003          GLOBAL          ABS GLOBAL
m_hf3......int     000004          GLOBAL          ABS GLOBAL
m_hpl......int     000C00          GLOBAL          ABS GLOBAL
m_hpl0.....int     00000A          GLOBAL          ABS GLOBAL
```



```
m_hpl1.....int     00000B          GLOBAL          ABS GLOBAL
m_hrdf.....int     000000          GLOBAL          ABS GLOBAL
m_hrie.....int     000000          GLOBAL          ABS GLOBAL
m_hrx......int     00FFEB          GLOBAL          ABS GLOBAL
m_hsr......int     00FFE9          GLOBAL          ABS GLOBAL
m_htde.....int     000001          GLOBAL          ABS GLOBAL
m_htie.....int     000001          GLOBAL          ABS GLOBAL
m_htx......int     00FFEB          GLOBAL          ABS GLOBAL
m_ial......int     000007          GLOBAL          ABS GLOBAL
m_ial0.....int     000000          GLOBAL          ABS GLOBAL
m_ial1.....int     000001          GLOBAL          ABS GLOBAL
m_ial2.....int     000002          GLOBAL          ABS GLOBAL
```

m_ibl.....int	000038	GLOBAL	ABS GLOBAL
m_ibl0.....int	000003	GLOBAL	ABS GLOBAL
m_ibl1.....int	000004	GLOBAL	ABS GLOBAL
m_ibl2.....int	000005	GLOBAL	ABS GLOBAL
m_idle.....int	000003	GLOBAL	ABS GLOBAL
m_if.....int	000002	GLOBAL	ABS GLOBAL
m_if0.....int	000000	GLOBAL	ABS GLOBAL
m_if1.....int	000001	GLOBAL	ABS GLOBAL
m_ilie.....int	00000A	GLOBAL	ABS GLOBAL
m_ipr.....int	00FFFF	GLOBAL	ABS GLOBAL
m_mod.....int	00000B	GLOBAL	ABS GLOBAL
m_of.....int	000003	GLOBAL	ABS GLOBAL
m_of0.....int	000000	GLOBAL	ABS GLOBAL
m_of1.....int	000001	GLOBAL	ABS GLOBAL
m_or.....int	000004	GLOBAL	ABS GLOBAL
m_pbc.....int	00FFE0	GLOBAL	ABS GLOBAL
m_pbd.....int	00FFE4	GLOBAL	ABS GLOBAL
m_pbddr.....int	00FFE2	GLOBAL	ABS GLOBAL
m_pcc.....int	00FFE1	GLOBAL	ABS GLOBAL
m_pcd.....int	00FFE5	GLOBAL	ABS GLOBAL
m_pcddr.....int	00FFE3	GLOBAL	ABS GLOBAL
m_pe.....int	000005	GLOBAL	ABS GLOBAL
m_pm.....int	0000FF	GLOBAL	ABS GLOBAL
m_psr.....int	00000F	GLOBAL	ABS GLOBAL
m_r8.....int	000007	GLOBAL	ABS GLOBAL
m_rcm.....int	00000E	GLOBAL	ABS GLOBAL
m_rdf.....int	000007	GLOBAL	ABS GLOBAL
m_rdrf.....int	000002	GLOBAL	ABS GLOBAL
m_re.....int	000008	GLOBAL	ABS GLOBAL
m_rfs.....int	000003	GLOBAL	ABS GLOBAL
m_rie.....int	00000B	GLOBAL	ABS GLOBAL
m_roe.....int	000005	GLOBAL	ABS GLOBAL
m_rwi.....int	000006	GLOBAL	ABS GLOBAL
m_rx.....int	00FFEF	GLOBAL	ABS GLOBAL
m_sbk.....int	000004	GLOBAL	ABS GLOBAL
m_sccr.....int	00FFF2	GLOBAL	ABS GLOBAL
m_scd.....int	00001C	GLOBAL	ABS GLOBAL
m_scd0.....int	000002	GLOBAL	ABS GLOBAL
m_scd1.....int	000003	GLOBAL	ABS GLOBAL
m_scd2.....int	000004	GLOBAL	ABS GLOBAL
m_sckd.....int	000005	GLOBAL	ABS GLOBAL
m_scl.....int	00C000	GLOBAL	ABS GLOBAL
m_scl0.....int	00000E	GLOBAL	ABS GLOBAL
m_scl1.....int	00000F	GLOBAL	ABS GLOBAL
m_scp.....int	00000D	GLOBAL	ABS GLOBAL
m_scr.....int	00FFF0	GLOBAL	ABS GLOBAL
m_sr.....int	00FFEE	GLOBAL	ABS GLOBAL
m_sre.....int	00000D	GLOBAL	ABS GLOBAL

m_srie.....int	00000F	GLOBAL	ABS GLOBAL
m_srxh.....int	00FFF6	GLOBAL	ABS GLOBAL
m_srxl.....int	00FFF4	GLOBAL	ABS GLOBAL
m_srxm.....int	00FFF5	GLOBAL	ABS GLOBAL
m_ssl.....int	003000	GLOBAL	ABS GLOBAL
m_ssl0.....int	00000C	GLOBAL	ABS GLOBAL
m_ssl1.....int	00000D	GLOBAL	ABS GLOBAL
m_ssr.....int	00FFF1	GLOBAL	ABS GLOBAL
m_ste.....int	00000C	GLOBAL	ABS GLOBAL
m_stie.....int	00000E	GLOBAL	ABS GLOBAL
m_stxa.....int	00FFF3	GLOBAL	ABS GLOBAL
m_stxh.....int	00FFF6	GLOBAL	ABS GLOBAL
m_stxl.....int	00FFF4	GLOBAL	ABS GLOBAL
m_stxm.....int	00FFF5	GLOBAL	ABS GLOBAL
m_syn.....int	000009	GLOBAL	ABS GLOBAL
m_tcm.....int	00000F	GLOBAL	ABS GLOBAL
m_tde.....int	000006	GLOBAL	ABS GLOBAL
m_tdre.....int	000001	GLOBAL	ABS GLOBAL
m_te.....int	000009	GLOBAL	ABS GLOBAL
m_tfs.....int	000002	GLOBAL	ABS GLOBAL
m_tie.....int	00000C	GLOBAL	ABS GLOBAL
m_tmie.....int	00000D	GLOBAL	ABS GLOBAL
m_trne.....int	000000	GLOBAL	ABS GLOBAL
m_tsr.....int	00FFEE	GLOBAL	ABS GLOBAL
m_tue.....int	000004	GLOBAL	ABS GLOBAL
m_tx.....int	00FFEF	GLOBAL	ABS GLOBAL
m_wake.....int	000005	GLOBAL	ABS GLOBAL
m_wds.....int	000003	GLOBAL	ABS GLOBAL
m_wds0.....int	000000	GLOBAL	ABS GLOBAL
m_wds1.....int	000001	GLOBAL	ABS GLOBAL
m_wds2.....int	000002	GLOBAL	ABS GLOBAL
m_wl.....int	006000	GLOBAL	ABS GLOBAL
m_wl0.....int	00000D	GLOBAL	ABS GLOBAL
m_wl1.....int	00000E	GLOBAL	ABS GLOBAL
m_woms.....int	000007	GLOBAL	ABS GLOBAL
no_ov.....int	P:002702	RTC_handler	REL LOCAL
proc_12_bit.....int	P:0000E2	proc_line	ABS LOCAL
proc_16_bit.....int	P:0000CE	proc_line	ABS LOCAL
put555_host.....int	P:0016B4	asmsupport	REL GLOBAL
putdata.....int	P:002675	ipp_driver	REL LOCAL
rtc_isr.....int	P:0026ED	RTC_handler	REL LOCAL
sci_rx_handler...int	P:000040	interrupts	ABS GLOBAL
sci_rx_inptr.....int	Y:00000D	GLOBAL	ABS GLOBAL
sci_rx_mod.....int	Y:00000E	GLOBAL	ABS GLOBAL
sci_rxe_handler..int	P:00004F	interrupts	ABS GLOBAL
seek_event.....int	P:000109	proc_line	ABS LOCAL
simple_life.....int	P:0026E2	RTC_handler	REL LOCAL
ssi_rxe_handler..int	P:000064	interrupts	ABS GLOBAL
ssi_tx_handler...int	P:000052	interrupts	ABS GLOBAL
ssi_tx_mod.....int	Y:000009	GLOBAL	ABS GLOBAL
ssi_tx_outptr....int	Y:000008	GLOBAL	ABS GLOBAL
ssi_txe_handler..int	P:000061	interrupts	ABS GLOBAL
tickerhi.....int	Y:00098E	RTC_handler	REL LOCAL
tickerlo.....int	Y:00098D	RTC_handler	REL LOCAL
wait_for_DT.....int	P:002651	ipp_driver	REL LOCAL
wait_for_host....int	P:0016AB	asmsupport	REL LOCAL
wait_for_host_55.int	P:0016B7	asmsupport	REL LOCAL
wait_rq_dwn.....int	P:002619	ipp_driver	REL LOCAL
wait_rx_dma.....int	P:00261E	ipp_driver	REL LOCAL
wait_tx_dma.....int	P:002657	ipp_driver	REL LOCAL

waitnext_RX.....int	P:002634	ipp_driver	REL LOCAL
waitnext_TX.....int	P:002670	ipp_driver	REL LOCAL
xbuf_vidbuf.....int	X:000000	GLOBAL	ABS GLOBAL BUFFER
y_a1_putaside_ph.int	Y:000000	GLOBAL	ABS GLOBAL
y_down1.....int	Y:00001D	GLOBAL	ABS GLOBAL
y_event_add.....int	Y:000013	GLOBAL	ABS GLOBAL
y_flags_pl.....int	Y:000011	GLOBAL	ABS GLOBAL
y_high12_mask....int	Y:000024	GLOBAL	ABS GLOBAL
y_high16_mask....int	Y:000025	GLOBAL	ABS GLOBAL
y_left1.....int	Y:00001F	GLOBAL	ABS GLOBAL
y_low12_mask.....int	Y:000023	GLOBAL	ABS GLOBAL
y_m1_putaside_sc.int	Y:000004	GLOBAL	ABS GLOBAL
y_m1_putaside_ss.int	Y:000002	GLOBAL	ABS GLOBAL
y_plus1.....int	Y:000020	GLOBAL	ABS GLOBAL

```
y_plus2.....int   Y:000021          GLOBAL          ABS GLOBAL
y_proc_ptr_mod...int   Y:000022     GLOBAL          ABS GLOBAL
y_r1_putaside_sc.int   Y:000003     GLOBAL          ABS GLOBAL
y_r1_putaside_ss.int   Y:000001     GLOBAL          ABS GLOBAL
y_right1.....int   Y:00001E         GLOBAL          ABS GLOBAL
y_unmod.....int   Y:00000F          GLOBAL          ABS GLOBAL
y_up1.....int   Y:00001C            GLOBAL          ABS GLOBAL
y_vbp_ptr.....int   Y:000012        GLOBAL          ABS GLOBAL
```

Symbol Listing by Value

Value	Name	Value	Name	Value	Name
000000	DT_MODEBIT	000000	IRQA_L0_BIT	000000	M_HRDF
000000	M_HRDF	000000	M_HRIE	000000	M_HRIE
000000	M_IAL0	000000	M_IAL0	000000	M_IF0
000000	M_IF0	000000	M_OF0	000000	M_OF0
000000	M_TRNE	000000	M_TRNE	000000	M_WDS0
000000	M_WDS0	000000	NO_MODE	000000	SHIFT
000000	VIDBUF_BOTTOM	000000	WAITDIS_IRQFST	000000	WAITDIS_IRQFST
000000	WAITDIS_IRQFST	000000	WAIT_DISABLE	000000	WAIT_DISABLE
000000	WAIT_DISABLE	000000	WAIT_DISSTAT_BIT	000000	WAIT_DISSTAT_BIT
000000	WAIT_DISSTAT_BIT	000000	XSIZE	000000	m_hrdf
000000	m_hrie	000000	m_ial0	000000	m_if0
000000	m_of0	000000	m_trne	000000	m_wds0
5.000000E-001	Mbps	000001	DT_MODE	000001	IRQA_L1_BIT
000001	IRQA_SRCSTAT_BIT	000001	IRQA_SRCSTAT_BIT	000001	IRQA_SRCSTAT_BIT
000001	M_HTDE	000001	M_HTDE	000001	M_HTIE
000001	M_HTIE	000001	M_IAL1	000001	M_IAL1
000001	M_IF1	000001	M_IF1	000001	M_OF1
000001	M_OF1	000001	M_TDRE	000001	M_TDRE
000001	M_WDS1	000001	M_WDS1	000001	m_htde
000001	m_htie	000001	m_ial1	000001	m_if1
000001	m_of1	000001	m_tdre	000001	m_wds1
000002	IRQA_SEN_BIT	000002	IRQB_STAT_BITNO	000002	IRQB_STAT_BITNO
000002	IRQB_STAT_BITNO	000002	M_HCIE	000002	M_HCIE
000002	M_HCP	000002	M_HCP	000002	M_IAL2
000002	M_IAL2	000002	M_IF	000002	M_IF
000002	M_OF	000002	M_OF	000002	M_RDRF
000002	M_RDRF	000002	M_SCD0	000002	M_SCD0
000002	M_TFS	000002	M_TFS	000002	M_WDS2
000002	M_WDS2	000002	WAITENA_IRQFST	000002	WAITENA_IRQFST
000002	WAITENA_IRQFST	000002	WAIT_ENABLE	000002	WAIT_ENABLE
000002	WAIT_ENABLE	000002	m_hcie	000002	m_hcp

000002	m_ial2	000002	m_if	000002	m_rdrf
000002	m_scd0	000002	m_tfs	000002	m_wds2
000003	ACK_TD	000003	IRQA_STAT_BITNO	000003	IRQA_STAT_BITNO
000003	IRQA_STAT_BITNO	000003	M_HF0	000003	M_HF0
000003	M_HF2	000003	M_HF2	000003	M_IBL0
000003	M_IBL0	000003	M_IDLE	000003	M_IDLE
000003	M_RFS	000003	M_RFS	000003	M_SCD1
000003	M_SCD1	000003	M_WDS	000003	M_WDS
000003	REQ_ABT	000003	m_hf0	000003	m_hf2
000003	m_ibl0	000003	m_idle	000003	m_of
000003	m_rfs	000003	m_scd1	000003	m_wds
000004	ACK_DT	000004	BASE_PROC_ID	000004	EEPROM_WE_BITNO
000004	EEPROM_WE_BITNO	000004	EEPROM_WE_BITNO	000004	IRQA_SRC_BITNO
000004	IRQA_SRC_BITNO	000004	IRQA_SRC_BITNO	000004	IRQ_SRC_BIT
000004	M_HF1	000004	M_HF1	000004	M_HF3
000004	M_HF3	000004	M_IBL1	000004	M_IBL1
000004	M_OR	000004	M_OR	000004	M_SBK
000004	M_SBK	000004	M_SCD2	000004	M_SCD2
000004	M_TUE	000004	M_TUE	000004	REQ_TD
000004	m_hf1	000004	m_hf3	000004	m_ibl1
000004	m_or	000004	m_sbk	000004	m_scd2
000004	m_tue	000005	HFB_EVENT_BITNO	000005	HFB_EVENT_BITNO
000005	HFB_EVENT_BITNO	000005	M_IBL2	000005	M_IBL2
000005	M_PE	000005	M_PE	000005	M_ROE
000005	M_ROE	000005	M_SCKD	000005	M_SCKD

```
000005 M_WAKE 000005 M_WAKE 000005
  SSD1BIT
000005 SSD1BIT 000005 SSD1BIT 000005
  m_ibl2
000005 m_pe 000005 m_roe 000005
  m_sckd
000005 m_wake 000006 M_FE 000006
  M_FE
000006 M_RWI 000006 M_RWI 000006
  M_TDE
000006 M_TDE 000006 SSD2BIT 000006
  SSD2BIT
000006 SSD2BIT 000006 m_fe 000006
  m_rwi
000006 m_tde 000007 DSP_ID_BITNO 000007
  DSP_ID_BITNO
000007 DSP_ID_BITNO 000007 M_DMA 000007
  M_DMA
000007 M_IAL 000007 M_IAL 000007
  M_R8
000007 M_R8 000007 M_RDF 000007
  M_RDF
000007 M_WOMS 000007 M_WOMS 000007
  m_dma
000007 m_ial 000007 m_r8 000007
  m_rdf
000007 m_woms 000008 IRQA_VEC_ADR 000008
  M_FSL
000008 M_FSL 000008 M_RE 000008
  M_RE
000008 m_fsl 000008 m_re 000009
  M_SYN
000009 M_SYN 000009 M_TE 000009
  M_TE
000009 divider 000009 m_syn 000009
  m_te
00000A M_GCK 00000A M_GCK 00000A
  M_HPL0
00000A M_HPL0 00000A M_ILIE 00000A
  M_ILIE
00000A m_gck 00000A m_hpl0 00000A
  m_ilie
00000B M_HPL1 00000B M_HPL1 00000B
  M_MOD
00000B M_MOD 00000B M_RIE 00000B
  M_RIE
00000B m_hpl1 00000B m_mod 00000B
  m_rie
00000C M_COD 00000C M_COD 00000C
  M_SSL0
00000C M_SSL0 00000C M_STE 00000C
  M_STE
00000C M_TIE 00000C M_TIE 00000C
  m_cod
00000C m_ssl0 00000C m_ste 00000C
  m_tie
00000D M_SCP 00000D M_SCP 00000D
  M_SRE
00000D M_SRE 00000D M_SSL1 00000D
  M_SSL1
00000D M_TMIE 00000D M_TMIE 00000D
  M_WL0
00000D M_WL0 00000D m_scp 00000D
  m_sre
00000D m_ssl1 00000D m_tmie 00000D
  m_wl0
00000E M_RCM 00000E M_RCM 00000E
  M_SCL0
00000E M_SCL0 00000E M_STIE 00000E
  M_STIE
```



```
00000E M_WL1 00000E M_WL1 00000E
  m_rcm
00000E m_scl0 00000E m_stie 00000E
  m_wl1
00000F M_PSR 00000F M_PSR 00000F
  M_SCL1
00000F M_SCL1 00000F M_SRIE 00000F
  M_SRIE
00000F M_TCM 00000F M_TCM 00000F
  m_psr
00000F m_scl1 00000F m_srie 00000F
  m_tcm
000010 WAITDIS_IRQSLW 000010 WAITDIS_IRQSLW 000010
  WAITDIS_IRQSLW
000012 WAITENA_IRQSLW 000012 WAITENA_IRQSLW 000012
  WAITENA_IRQSLW
2.000000E+001 CPU_Mhz 000014 GPP_REQ_DT_NUM 000018
  M_HF
000018 M_HF 000018 m_hf 00001C
  M_SCD
00001C M_SCD 00001C m_scd 000028
  GPP_REQ_DT_ADR
000038 M_IBL 000038 M_IBL 000038
  m_ibl
000040 SCI_RXQ_SIZE 000040 sci_rx_handler 00004F
  sci_rxe_handler
000052 ssi_tx_handler 000060 DEBUG_BIT_MASK 000060
  DEBUG_BIT_MASK
000060 DEBUG_BIT_MASK 000061 ssi_txe_handler 000064
  ssi_rxe_handler
0000A0 Fproc_line 0000CE proc_16_bit 0000E2
  proc_12_bit
0000FF M_PM 0000FF M_PM 0000FF
  m_pm
000102 ge_thresh_hi 000107 ge_thresh_lo 000109
  ge_thresh_16
000109 seek_event 000136 higher_exists 00013A
  compare
000400 SSI_TXQ_SIZE 0009A2 DSIZE 0009A2
  LSIZE
0009A2 YSIZE 000AAB CRT0ORG 000AAB
  F__start
000AD1 F__crt0_end 000AD5 F__printf_end 000AD7
  Fsim4
000AE2 L6 000AE3 ASM_APP_0 000AE4
  ASM_APP_1
000AE4 L15 000AE6 L2 000AE8
  L11
000AEA ASM_APP_2 000AEB ASM_APP_3 000AEB
  L13
000AEF Fsimulate_frame_ 000B02 L20 000B19
  L26
000B21 L28 000B2B L24 000B2E
  L27
000B47 Fsimulate_row_sy 000B4C L33 000B4D
  ASM_APP_4
000B4E ASM_APP_5 000B4E L35 000B5E
  Fsimulate_word_r
```

000B85	L37	000B8C	L38	000BAE
ASM_APP_6				
000BAF	ASM_APP_7	000BCA	ASM_APP_8	000BCB
ASM_APP_9				
000BDE	ASM_APP_A	000BDF	ASM_APP_B	000BF0
ASM_APP_C				
000BF1	ASM_APP_D	000C00	M_HPL	000C00
M_HPL				
000C00	m_hpl	000C16	L40	000C27
L41				
000C2A	L39	000C38	Falt_simulate_wo	000C43
L47				
000C45	ASM_APP_E	000C46	ASM_APP_F	000C4F
L45				
000C50	L49	000C57	Fsimulate_lbcmd_	000C5F
ASM_APP_10				
000C60	ASM_APP_11	000C6C	ASM_APP_12	000C6C
L51				
000C6D	ASM_APP_13	000C82	Fxmemcpy	000CA8
L57				
000CA9	ASM_APP_14	000CAB	ASM_APP_15	000CB8
L59				
000CB9	L58	000CC6	Fymemcpy	000CEC
L64				
000CFB	L66	000CFC	L65	000D09
Fpmemcpy				
000D2F	L71	000D30	ASM_APP_16	000D32
ASM_APP_17				
000D3F	L73	000D40	L72	000D4D
Fmemdump_cmd				
000D74	L75	000D83	L82	000D8B
L77				
000D9A	L78	000DA9	L79	000DB6
L76				
000DDB	L83	000DEC	Fmempoke_cmd	000E19
L91				
000E21	ASM_APP_18	000E21	L86	000E22
ASM_APP_19				
000E24	L87	000E27	ASM_APP_1A	000E27
L88				
000E28	ASM_APP_1B	000E28	L85	000E35
Fmemdump_auto				
000E5F	Fmain	000E8A	L94	000EA7
L99				
000EA8	L98	000EC4	Finit_portc_intr	000EC9
ASM_APP_0				
000ECA	ASM_APP_1	000ECA	ASM_APP_2	000ECB
ASM_APP_3				
000ECB	ASM_APP_4	000ECC	ASM_APP_5	000ECE
ASM_APP_6				
000ECE	L2	000ECF	ASM_APP_7	000ECF
ASM_APP_8				
000ED0	ASM_APP_9	000ED0	ASM_APP_A	000ED1
ASM_APP_B				
000ED1	L3	000ED2	Fonetime_init	000F00
BOARD_ID_MASK				
000F00	BOARD_ID_MASK	000F00	BOARD_ID_MASK	000F15
ASM_APP_C				
000F16	ASM_APP_D	000F4A	L8	000F4E
L10				
000F95	Fcheck4	000FA6	ASM_APP_0	000FA8
ASM_APP_1				
000FB3	ASM_APP_2	000FB5	ASM_APP_3	000FC1
ASM_APP_4				
000FC3	ASM_APP_5	000FCF	ASM_APP_6	000FD1
ASM_APP_7				
000FDE	L5	000FE7	L2	000FF4
Fswitch_params				
000FFF	M_CD	000FFF	M_CD	000FFF
m_cd				
001000	LBOX_QUEUE_BOTTO	00100A	L11	001014
L9				
00101F	L10	001026	Fget_coords	00105B
L13				
00106C	L14	001086	L15	0010A5
L27				
0010B7	L17	0010CB	L18	0010D3
L28				
0010DC	L20	0010EC	L21	0010F6
L22				
00110B	L23	00111B	L30	00111E
L29				
001129	L16	001146	Fload_thresh	00115C
L41				
00116C	L33	001191	L34	0011AC
L36				
0011C7	L37	0011E0	L42	0011E6
L38				
001217	L32	00121C	Fload_init_proc_	001239
L44				
001247	Fsync_frame	001275	L65	001287
L64				
001293	L49	001296	ASM_APP_8	001298
ASM_APP_9				
0012A6	ASM_APP_A	0012A7	ASM_APP_B	0012B8
L51				
0012D4	L54	0012F7	L57	001308
L58				
00131C	L68	001323	L60	001342
L67				
001344	L46	001357	L47	001369
Fsync_row				
0013C2	L71	0013C4	L70	0013D3
Ftry_proc_vid				
00141A	L85	001421	L74	001440
STACK_START				
00144C	L79	001491	L82	0014AF
L83				
0014C4	L77	0014D2	Fevent_loc_okay	001501
L89				
001521	L88	001534	L94	00155C
L93				
00155E	L92	00156A	L91	00157A
Frecord_event				
0015C9	L98	0015D7	L102	0015DA
L99				
0015E8	ASM_APP_C	0015E9	ASM_APP_D	0015F4
ASM_APP_E				
0015F5	ASM_APP_F	00160F	L97	00161D
Fabort_stream				
00162B	Finit_communicat	001659	Fcopy_lowp_routi	001665
Finit_intr_vecto				
001688	Fcheck_host	00169D	Fget_host	0016A3
Fput_host				
0016AB	wait_for_host	0016B4	put555_host	0016B7
wait_for_host_55				
0016BC	Ftxd_on	0016BE	Ftxd_off	0016C0
Ftxd_tog				

0016C2 Fsc1_on 0016C4 Fsc1_off 0016C6
FSend_Host_Messa
0016E3 L2 0016F0 L3 00170C
FGet_Host_Messag
001725 L5 001727 L6 00172F
Fsend_test_word
001758 L8 001765 Fsend_test_int 001799
L10
0017A8 Fwrite_lbcmd 0017E4 L12 0017F1
Fparam_dump
00184E L18 001856 L20 0018C7
L21
0018D8 Fparse_viddsp_cm 001902 L24 00190A
L25
001912 L27 001918 L28 00191E
L29
00192D L50 001934 L30 001943
L51
00194A L31 001955 ASM_APP_0 001956
ASM_APP_1
00195D L32 00196A L36 001980
L47
00198B L37 00199B L41 0019BD
L46
0019CE L42 0019DA L43 0019E0
L44
0019E6 L48 0019E9 L49 0019FD
L52
001A02 L23 001A12 Fparse_tdb_comma 001A64
L75
001A69 L56 001A7F L76 001A8D
L55
001AC7 L73 001AD8 L61 001AE6
L62
001AEA L63 001AEE L64 001AF2
L65
001AF6 L66 001B00 L67 001B15
L60
001B64 L74 001B66 L53 001B75
Ftry_get_message
001BA5 L78 001BE3 L82 001C0C
L127
001C0E L83 001C17 L84 001C48
L85
001C66 L89 001C68 L125 001C7E
L94
001CA0 L123 001CA1 L122 001CAD
L95
001CDF L97 001CE3 L96 001D03
L126
001D19 L105 001D46 L121 001D47
L120
001D54 L106 001DC2 L108 001DD2
L112

001DEA L119 001DEC L113 001DEC
L114
001E04 L128 001E17 L129 001E24
L124
001E28 L115 001E42 L130 001E60
L116
001E91 L131 001E96 L77 001EA7
Ftry_fill_lbcmd
001EAB L133 001EB7 Ftry_relay_hk 001EDC
L136
001EF0 L139 001F00 M_DC 001F00
M_DC
001F00 m_dc 001F05 L144 001F44
L142
001F45 L157 001F47 L138 001F74
L146
001FB7 L158 001FD6 L147 001FF3
L149
001FF8 L137 002000 EVENTS_BOTTOM 002000
VIDBUF_LEN
002000 VIDBUF_TOP 002042 L153 002088
L159
00208C L150 00209B Ftry_relay_error 0020C9
L164
0020CE L161 0020F0 L165 0020F5
L162
002119 L166 00211E L163 002128
Fsignal_frame_st
002158 L168 002167 Fsignal_frame_en 002197
L170
0021A6 Fsignal_frame_er 0021DC L172 0021EC
Fsend_packet_of_
002212 L174 002216 L186 00222B
L176
00222D L187 00224B L178 00225F
L183
002260 ASM_APP_2 002262 ASM_APP_3 00228E
ASM_APP_4
00228F ASM_APP_5 002292 L182 0022AC
L185
0022B7 L184 0022D1 L188 0022E6
Fdump_events
002301 L192 002314 L193 00231E
FIPP_collectmsg
00235A L4 00235E L3 00235E
L5
002369 L2 002369 L6 0023A0
L8
0023AA L7 0023AA L9 0023AE
L1
0023B7 FIPP_get_message 0023BE L11 0023D4
L14
0023E1 L16 0023E3 L17 0023E6
L13
0023E6 L15 0023E8 L10 0023E8
L12
0023F1 FIPP_send_messag 002423 L20 002430
L19
002439 FIPP_make_header 00246C L22 00247A
L23
0024A0 L21 0024A9 FIPP_makeheader 0024DC
L25
0024EA L26 00251E L24 002527
FIPP_register_ad
002567 L28 002575 L29 0025A5
L30
0025A7 L27 0025B1 FIPP_get_address 0025D2
L32
0025D6 L33 0025DE L34 0025E0
L31
0025EC Finitialize_dma_ 0025FC its_dsp_zero 002607
gppReqDTISR
00260B Freceive_words 002619 wait_rq_dwn 00261E
wait_rx_dma

```
002632 RX_even 002634 waitnext_RX 002639
  getdata
00263C abort_RX 002642 exit_RX 00264B
  Fsend_words
002651 wait_for_DT 002657 wait_tx_dma 00266E
  TX_even
002670 waitnext_TX 002675 putdata 002678
  abort_TX
002680 exit_TX 002689 Finit_RTC_driver 0026A7
  Fget_irqa_tick
0026AB Fget_nsynch_time 0026E2 simple_life 0026ED
  rtc_isr
002702 no_ov 002707 Fcheck_buddy_tim 00272B
  L1
00272B L2 002734 Fprepare_IPP_hea 002766
  L4
002771 L5 00277C L3 002784
  Fmalloc
0027A1 L2 0027AA L6 0027CB
  L5
0027D3 L19 0027E3 L7 0027F0
  L15
002801 L13 00281D L20 002822
  L12
00283E L10 002846 L18 002860
  L16
002878 L21 00287C L1 00288A
  Frealloc
0028AC L24 0028C1 L25 0028CD
  L27
0028D0 L26 0028E2 L22 0028EE
  Ffree
002904 L29 00290B Fmemcpy 002922
  L4
002924 L6 002933 Fbrk 002940
  ASM_APP_0
002941 ASM_APP_1 002954 L6 00295C
  L5
00295E L3 00296A Fsbrk 002985
  L8
002987 L7 002990 PSIZE 002EE0
  MAXNUMEVENTS
003000 M_SSL 003000 M_SSL 003000
  m_ssl
006000 M_WL 006000 M_WL 006000
  m_wl
```



```
007FFF TOP_OF_MEMORY 008000 Boot_EEROM 00C000
  M_SCL
00C000 M_SCL 00C000 m_scl 00FFC0
  FRTC_Count
00FFC0 FRTC_Count 00FFC0 FRTC_Count 00FFC1
  FRF_Count
00FFC1 FRF_Count 00FFC1 FRF_Count 00FFC2
  FGPS_Count
00FFC2 FGPS_Count 00FFC2 FGPS_Count 00FFC3
  FDGlue_Ctrl
00FFC3 FDGlue_Ctrl 00FFC3 FDGlue_Ctrl 00FFC3
  FDGlue_Stat
00FFC3 FDGlue_Stat 00FFC3 FDGlue_Stat 00FFE0
  M_PBC
00FFE0 M_PBC 00FFE0 m_pbc 00FFE1
  M_PCC
00FFE1 M_PCC 00FFE1 m_pcc 00FFE2
  M_PBDDR
00FFE2 M_PBDDR 00FFE2 m_pbddr 00FFE3
  M_PCDDR
00FFE3 M_PCDDR 00FFE3 m_pcddr 00FFE4
  M_PBD
00FFE4 M_PBD 00FFE4 m_pbd 00FFE5
  M_PCD
00FFE5 M_PCD 00FFE5 m_pcd 00FFE8
  ACK_PORT
00FFE8 M_HCR 00FFE8 M_HCR 00FFE8
  m_hcr
00FFE9 M_HSR 00FFE9 M_HSR 00FFE9
  REQ_PORT
00FFE9 m_hsr 00FFEB M_HRX 00FFEB
  M_HRX
00FFEB M_HTX 00FFEB M_HTX 00FFEB
  m_hrx
00FFEB m_htx 00FFEC M_CRA 00FFEC
  M_CRA
00FFEC m_cra 00FFED M_CRB 00FFED
  M_CRB
00FFED m_crb 00FFEE M_SR 00FFEE
  M_SR
00FFEE M_TSR 00FFEE M_TSR 00FFEE
  m_sr
00FFEE m_tsr 00FFEF M_RX 00FFEF
  M_RX
00FFEF M_TX 00FFEF M_TX 00FFEF
  m_rx
00FFEF m_tx 00FFF0 M_SCR 00FFF0
  M_SCR
00FFF0 m_scr 00FFF1 M_SSR 00FFF1
  M_SSR
00FFF1 m_ssr 00FFF2 M_SCCR 00FFF2
  M_SCCR
00FFF2 m_sccr 00FFF3 M_STXA 00FFF3
  M_STXA
00FFF3 m_stxa 00FFF4 M_SRXL 00FFF4
  M_SRXL
00FFF4 M_STXL 00FFF4 M_STXL 00FFF4
  m_srxl
00FFF4 m_stxl 00FFF5 M_SRXM 00FFF5
  M_SRXM
00FFF5 M_STXM 00FFF5 M_STXM 00FFF5
  m_srxm
00FFF5 m_stxm 00FFF6 M_SRXH 00FFF6
  M_SRXH
00FFF6 M_STXH 00FFF6 M_STXH 00FFF6
  m_srxh
00FFF6 m_stxh 00FFFE M_BCR 00FFFE
  M_BCR
00FFFE m_bcr 00FFFF M_IPR 00FFFF
  M_IPR
00FFFF m_ipr 085DAF SSI_RX_INSTR 0BF080
  ABS_JSR_INSTR
0D0000 SHORT_JSR_INSTR 000000 xbuf_vidbuf 002000
  eventspace
008000 FFrame_X8 009000 FFrame_X9 00A000
  FFrame_XA
00B000 FFrame_XB 000000 y_a1_putaside_ph 000001
  y_r1_putaside_ss
000002 y_m1_putaside_ss 000003 y_r1_putaside_sc 000004
  y_m1_putaside_sc
000005 Fssi_txq_xfers 000006 Fssi_tx_intrs 000007
  Fssi_tx_inptr
```

```
000008 ssi_tx_outptr 000009 ssi_tx_mod 00000A
  Fsci_rxq_xfers
00000B Fsci_rx_intrs 00000C Fsci_rx_outptr 00000D
  sci_rx_inptr
00000E sci_rx_mod 00000F y_unmod 000010
  Fasm_proclen
000011 y_flags_pl 000012 y_vbp_ptr 000013
  y_event_add
000014 Fasm_thresh0 000015 Fasm_thresh1 000016
  Fasm_thresh2
000017 Fasm_thresh3 000018 Fasm_thresh4 000019
  Fasm_thresh5
00001A Fasm_thresh6 00001B Fasm_thresh7 00001C
  y_up1
00001D y_down1 00001E y_right1 00001F
  y_left1
000020 y_plus1 000021 y_plus2 000022
  y_proc_ptr_mod
000023 y_low12_mask 000024 y_high12_mask 000025
  y_high16_mask
000026 Ferrno 000027 F__stack_safety 000028
  F__mem_limit
000029 F__break 00002A F__y_size 00002B
  F__max_signal
00002C Fssi_rx_exceptio 00002D Fssi_tx_exceptio 00002E
  Fsci_rx_exceptio
00002F Fdrop_frame_sync 000030 Fccd0sim 0000A0
  Fccd1sim
000110 F___id0 000111 F___ABCDcnt1 000112
  F___Aline2
000113 F___Bline3 000114 F___Cline4 000115
  F___Dline5
000116 F___sim_cntr6 000117 Ftestmsg_dest 00011B
  Flb_dest
00011F Fvideo_dest 000123 Fme 000127
  Fsingle_loop
000128 Fdoloop 000129 Fsimulating 00012A
  F___state0
```



```
00012B F___count1 00012C F___local_id2 00012D
  Flb_fill_command
00012E L45 00013C F___repeatmode0 00013D
  L71
000149 L117 000152 F___timeout1 000153
  Fread_pending
000154 FIPPresponse 000354 FcurrentResponse 000355
  FcurrentCmd
000356 FIPPbufTail 000357 FIPPbufHead 000358
  FIPPbuf
00097C F___got_header0 00097D F___header1 000989
  Fmy_name
00098A Flocal_processor 00098B Fmaster_node_id 00098C
  dma_state
00098D tickerlo 00098E tickerhi 00098F
  Fmajortag
000991 Fminortag 000993 F___last_tick0 000994
  Fbuddy_hdr
0009A0 Fmy_ipp_addr 0009A1 Fhead 001000
  Fssi_txq
001400 Fsci_rxq 001400 Fssi_txq_end 001440
  Fsci_rxq_end
008000 FFrame_Y8 009000 FFrame_Y9 00A000
  FFrame_YA
00B000 FFrame_YB 00EF00 FFrameReg_X8 00EF01
  FFrameReg_X9
00EF02 FFrameReg_XA 00EF03 FFrameReg_XB 00EF04
  FFrameReg_Y8
00EF05 FFrameReg_Y9 00EF06 FFrameReg_YA 00EF07
  FFrameReg_YB
```

BACK-END PROCESSOR (BEP) SOFTWARE

gmake PBBEP for Pizza-Box load file pb_BEP.hld. PB-specific files read from, and module cln's written to, pbcln/.
gmake ASMBEP for assembly compilation of BEP modules (put in asm/).

Map, interim load, and list files are put in obj/.

Relies on ipp.h / buddy_chk.h and subsequent files to define the following:

VARIABLE TYPES
==============
- IPP_address
- IPP_msg_type
- IPP_priority
- IPP_msg_buffer
- IPP_header

ROUTINES / MACROS
=================
- check_buddy_time()
- prepare_IPP_headers()
- IPP_make_header()
- IPP_send_message()
- IPP_get_message()
- IPP_register_address()
- IPP_get_address()
- IPP_length()
- IPP_source()
- IPP_type()

IPP NAMES & VARIABLES
=====================
- BEP
- DEFAULT_VIDEO_DEST_BEP
- DEFAULT_TESTMSG_DEST
- IPP_DATASIZE
- my_ipp_addr

MESSAGE TYPES & PRIORITIES
==========================

DSP_FRAME_STARTED	DSP_FRAME_STARTED_PRIORITY
DSP_FRAME_ENDED	DSP_FRAME_ENDED_PRIORITY
DSP_FRAME_ERROR	DSP_FRAME_ERROR_PRIORITY
DSP_EXCEPTION	DSP_EXCEPTION_PRIORITY
BEP_UNKNOWN_ID	BEP_UNKNOWN_ID_PRIORITY
BEP_RAW_FRAME	FRAME_DUMP_PRIORITY
BEP_DEALT_IMPS	
BEP_TBT	TBT_PRIORITY
VIDDSP_RESPONSE	VIDDSP_PRIORITY
UNKNOWN_TYPE	DEFAULT_PRIORITY
PROCESS_VIDEO	
DONT_PROCESS_VIDEO	
CCD_CONFIGURATION	
SWITCH_REGION	
REALLOCATE_FB	
SAVE_FRAME	
UNSAVE_FRAME	
DUMP_FRAME_RAW	
DUMP_FRAME_DEAL	
DUMP_IMP_ARRAY	
DUMP_TBT_ARRAY	
VIDDSP_COMMAND	

FLAGS, MASKS, ETC.
==================

FRAME_ERROR_ROWFAIL	FRAME_ERROR = row sync error
DSP_INSTALL_NEW_CCD_CONFIGURATION_NOW	New params take effect immediately
SSI_RX_EXCEPTION	DSP_EXCEPTION = SSI receive error (video)
DUMP_FIRST_MESSAGE	Flag indicating message is first in a set
DUMP_LAST_MESSAGE	Flag indicating message is last in a set
DUMP_MIDDLE_MESSAGE	0: Absence of above flags
NUMREGIONS	Number of regions in the Frame Buffer
HUC	Number indicating Horizontal Underclocks
HOC	Number indicating Horizontal Overclocks
VOC	Number indicating Vertical Overclocks

VIDDSP TEST MESSAGES
====================

VD_MEMDUMP	Memory peek
VD_TEST_WORD	16-bit word without specific association
VD_TEST_INT	24-bit word without specific association
VD_SIMULATE	Engage / Disengage simulation mode
VD_FORCENEW	Force vbpnew to become current
VD_ECHO	Echo back the word enclosed
VD_MEMPOKE	Memory poke
VD_VPDUMP_BEP	vidbuf_params dump
VD_NVPDUMP_BEP	next_vidbuf_params dump
VD_RPDUMP	region_params dump
VD_FSDUMP	frame_specs dump
VD_SINGLELOOP	Engage / Disengage single-mainloop mode
VD_DOLOOP	Perform a single mainloop
VD_UNREC_CMD	Unrecognized VIDDSP command

```
/* Module handling high-level communications with the GPP */

#include "BEPincl.h"

extern IPP_address me, video_dest, testmsg_dest;

extern int simulating, doloop, single_loop;

extern int ssi_rx_exception;

extern all_frames_struct fb;


void Send_Host_Message(IPP_address dest, IPP_msg_type type,
    IPP_priority priority, unsigned short data_len, IPP_msg_buffer msg)
{
    IPP_header head;

    IPP_make_header(&head, priority, dest, me, type, data_len);
    IPP_send_message(&head, msg);
}

int Get_Host_Message(IPP_header *head, IPP_msg_buffer msg)
{
    int got_msg;
    int retval;

    got_msg = IPP_get_message(head, msg);
    if (got_msg == -1) retval = 0;
    else retval = 1;

    return retval;
}

void send_test_word(unsigned short test_word)
{
    IPP_msg_buffer msg;

    msg[0] = VD_TEST_WORD;
    msg[1] = test_word;

    Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE, VIDDSP_PRIORITY,
        2, msg);
}

void send_test_int(int test_int)
{
    IPP_msg_buffer msg;

    msg[0] = VD_TEST_INT;
    msg[1] = (test_int & 0xFFFF);
    msg[2] = (test_int >> 16);

    Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE, VIDDSP_PRIORITY,
        3, msg);
}

void param_dump(vidbuf_param_struct p, unsigned short which)
{
    IPP_msg_buffer msg;
    int     i;

    msg[0] = which;
    msg[1] = (unsigned short)p.region;
    msg[2] = (unsigned short)p.left_off;
    msg[3] = (unsigned short)p.one_line_delta;
    msg[4] = (unsigned short)p.lines_rcvd;
    msg[5] = (unsigned short)p.waiting_for_frame_sync;
    msg[6] = (unsigned short)p.single_frame;
    msg[7] = (unsigned short)p.save_frame;
    for (i=0; i<TOTALSLICES; i++)
    {
        msg[2*i + 8] = (unsigned short)p.lobound_delta[i];
        msg[2*i + 9] = (unsigned short)p.hibound_delta[i];
    }
    for (i=0; i<NUM_CAMERAS; i++)
    {
        msg[4*i+2*TOTALSLICES+8]=(unsigned short)(p.imp_left_off[i]&0x00FFFF);
        msg[4*i+2*TOTALSLICES+9]=(unsigned short)(p.imp_left_off[i]>>16);
        msg[4*i+2*TOTALSLICES+10]=(unsigned short)(p.nip_left_off[i]&0x00FFFF);
        msg[4*i+2*TOTALSLICES+11]=(unsigned short)(p.nip_left_off[i]>>16);
    }

    Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE, VIDDSP_PRIORITY,
        2*TOTALSLICES+4*NUM_CAMERAS+8, msg);
}

void rp_dump(IPP_msg_buffer msg)
{
    int     region,i;

    region = msg[1];
    msg[2] = fb.rp[region].ccd.mode;
    msg[3] = fb.rp[region].ccd.used_cameras;
    msg[4] = fb.rp[region].ccd.num_hucs;
    msg[5] = fb.rp[region].ccd.num_cols;
    msg[6] = fb.rp[region].ccd.num_hocs;
    msg[7] = fb.rp[region].ccd.total_cols;
    msg[8] = fb.rp[region].ccd.num_rows;
    msg[9] = fb.rp[region].ccd.num_vocs;
    msg[10] = fb.rp[region].ccd.total_rows;
    msg[11] = fb.rp[region].deal;
    msg[12] = fb.rp[region].save_nips;
    msg[13] = fb.rp[region].size;
    for (i = 0; i < NUM_CAMERAS; i++)
    {
        msg[4*i+14] = fb.rp[region].start_row[i];
        msg[4*i+15] = fb.rp[region].end_row[i];
        msg[4*i+16] = fb.rp[region].start_col[i];
        msg[4*i+17] = fb.rp[region].end_col[i];
    }
    for (i = 0; i < NUM_CAMERAS; i++)
    {
        msg[3*i+4*NUM_CAMERAS+14] = fb.rp[region].imp_start_offset[i];
        msg[3*i+4*NUM_CAMERAS+15] = fb.rp[region].nip_start_offset[i];
        msg[3*i+4*NUM_CAMERAS+16] = fb.rp[region].voc_start_offset[i];
    }
    for (i = 0; i < NUM_CAMERAS; i++)
        msg[i+7*NUM_CAMERAS+14] = fb.rp[region].linelen[i];
    msg[8*NUM_CAMERAS+14] = fb.region_start[region] & 0xFFFF;
    msg[8*NUM_CAMERAS+15] = fb.region_start[region] >> 16;
    msg[8*NUM_CAMERAS+16] = fb.current_framenum[region];

    Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE, VIDDSP_PRIORITY,
        8*NUM_CAMERAS+17, msg);
}

void parse_viddsp_cmd(IPP_msg_buffer msg, vidbuf_param_struct *vbp,
    vidbuf_param_struct *vbpnew)
{
    switch(msg[0])
    {
        case VD_SIMULATE: simulating = msg[1];
```

```
                        init_portc_intrs();
                        break;
        case VD_FORCENEW: abort_stream(vbp, vbpnew);
                        break;
        case VD_ECHO:  Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE,
                            VIDDSP_PRIORITY, 2, msg);
                        break;
        case VD_MEMDUMP: memdump_cmd(msg);
                        break;
        case VD_MEMPOKE: mempoke_cmd(msg);
                        break;
        case VD_VPDUMP:  param_dump(*vbp, VD_VPDUMP_BEP);
                        break;
        case VD_NVPDUMP: param_dump(*vbpnew, VD_NVPDUMP_BEP);
                        break;
        case VD_RPDUMP:  rp_dump(msg);
                        break;
        case VD_FSDUMP:  msg[2] = fb.fs[msg[1]].id & 0xFFFF;
                        msg[3] = fb.fs[msg[1]].id >> 16;
                        msg[4] = fb.fs[msg[1]].frame_start & 0xFFFF;
                        msg[5] = fb.fs[msg[1]].frame_start >> 16;
                        msg[6] = fb.fs[msg[1]].next_frame;
                        Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE,
                            VIDDSP_PRIORITY, 7, msg);
                        break;
        case VD_GVDUMP:  msg[1] = (unsigned short)asm_inptr;
                        msg[2] = (unsigned short)simulating;
                        msg[3] = (unsigned short)single_loop;
                        __asm("move R6,%0" : "=S"(msg[4]) : );
                        Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE,
                            VIDDSP_PRIORITY, 5, msg);
                        break;
        case VD_SINGLELOOP: single_loop = msg[1];
                        doloop = !single_loop;
                        break;
        case VD_DOLOOP:  doloop = 1;
                        break;
        default:msg[1] = msg[0];
                    msg[0] = VD_UNREC_CMD;
                    Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE,
                        VIDDSP_PRIORITY, 2, msg);
    }
}

void try_get_message(vbp, vbpnew)
vidbuf_param_struct *vbp, *vbpnew;
{
    IPP_header  head;
    IPP_msg_buffer msg;
    int     i, region;

    doloop = !single_loop;

    if (Get_Host_Message(&head, msg))
    {
        switch(IPP_type(&head))
        {
            case PROCESS_VIDEO:
                vbpnew->save_frame = 1;
                vbpnew->single_frame = !msg[0];
                video_dest = IPP_source(&head);
                break;
            case DONT_PROCESS_VIDEO:
                vbpnew->save_frame = 0;
                break;
            case SWITCH_REGION:
                vbpnew->region = msg[0];
                set_vbp_new_config(vbpnew);
                break;
            case REALLOCATE_FB:
                for (i = 0; i < NUMREGIONS+1; i++)
                {
                    fb.region_start[i] = msg[2*i] + (msg[2*i+1] << 16);
                    if (i > 0) set_up_region(i-1);
                }
                break;
            case CCD_CONFIGURATION:
                region = msg[30];
                fb.rp[region].ccd.mode = msg[1];
                fb.rp[region].ccd.used_cameras = msg[2];
                fb.rp[region].ccd.num_hucs = msg[3];
                fb.rp[region].ccd.num_cols = msg[4];
                fb.rp[region].ccd.num_hocs = msg[5];
                fb.rp[region].ccd.num_rows = msg[6];
                fb.rp[region].ccd.num_vocs = msg[7];
                for (i = 0; i < NUM_CAMERAS; i++)
                {
                    fb.rp[region].start_row[i] = msg[8+(4*i)];
                    fb.rp[region].end_row[i] = msg[9+(4*i)];
                    fb.rp[region].start_col[i] = msg[10+(4*i)];
                    fb.rp[region].end_col[i] = msg[11+(4*i)];
                }
                fb.rp[region].deal = msg[8+(NUM_CAMERAS*4)];
                fb.rp[region].save_nips = msg[9+(NUM_CAMERAS*4)];
                fill_out_params(&fb.rp[region]);
                set_up_region(region);
                if (((msg[0] & DSP_INSTALL_NEW_CCD_CONFIGURATION_NOW) != 0)
                    || (region == vbp->region))
                {
                    abort_stream(vbp, vbpnew);
                }
                break;
            case SAVE_FRAME:
                save_frame(msg[0] + (msg[1] << 16));
                break;
            case UNSAVE_FRAME:
                unsave_frame(msg[0] + (msg[1] << 16));
                break;
            case DUMP_FRAME_RAW:
                dump_frame_raw(msg, IPP_source(&head));
                break;
            case DUMP_FRAME_DEAL:
                msg[4] = 0;
                msg[5] = 0xFFFF;
                msg[6] = 0;
                msg[7] = 0xFFFF;
                dump_imp_array(msg, IPP_source(&head));
                break;
            case DUMP_IMP_ARRAY:
                dump_imp_array(msg, IPP_source(&head));
                break;
            case DUMP_TBT_ARRAY:
                dump_tbt_array(msg, IPP_source(&head));
                break;
            case VIDDSP_COMMAND:
                testmsg_dest = IPP_source(&head);
                parse_viddsp_cmd(msg, vbp, vbpnew);
                break;
            default:
                msg[0] = IPP_type(&head);
                Send_Host_Message(IPP_source(&head), UNKNOWN_TYPE,
                    DEFAULT_PRIORITY, 1, msg);
                break;
        }
```

```
    }
}

void try_relay_errors()
{
  IPP_msg_buffer msg;

  if (ssi_rx_exception != 0)
    {
      msg[0] = SSI_RX_EXCEPTION;
      msg[1] = (unsigned short)ssi_rx_exception;
      ssi_rx_exception = 0;
      Send_Host_Message(video_dest, DSP_EXCEPTION,
            DSP_EXCEPTION_PRIORITY, 2, msg);
    }
}

void signal_frame_started(unsigned int id)
{
  IPP_msg_buffer msg;

  msg[0] = (unsigned short)(id & 0xFFFF);
  msg[1] = (unsigned short)(id >> 16);

  Send_Host_Message(video_dest, DSP_FRAME_STARTED,
           DSP_FRAME_STARTED_PRIORITY, 2, msg);
}

void signal_frame_ended(unsigned int id)
{
  IPP_msg_buffer msg;

  msg[0] = (unsigned short)(id & 0xFFFF);
  msg[1] = (unsigned short)(id >> 16);

  Send_Host_Message(video_dest, DSP_FRAME_ENDED,
           DSP_FRAME_ENDED_PRIORITY, 2, msg);
}

void signal_frame_error(unsigned int id, unsigned short reason)
{
  IPP_msg_buffer msg;

  msg[0] = (unsigned short)(id & 0xFFFF);
  msg[1] = (unsigned short)(id >> 16);
  msg[2] = reason;

  Send_Host_Message(video_dest, DSP_FRAME_ERROR, DSP_FRAME_ERROR_PRIORITY,
           3, msg);
}
```

```
/* Functions related to frames and FB regions: finding a frame in the FB by */
/* its ID; marking a frame inviolate; setting up a region; dumping frames */

#include <string.h>
#include "BEPincl.h"

#define make_even_plus(x) if (((x) & 1) != 0) (x)++
#define make_even_minus(x) if (((x) & 1) != 0) (x)--

extern all_frames_struct fb;


int find_frame_in_region(unsigned int id, int region)
{
    int framenum, return_frame = -2;

    framenum = region * FRAMES_PER_REGION;
    do
    {
        if (fb.fs[framenum].id == id) return_frame = framenum;
        else if (fb.fs[framenum].id == END_FRAME) return_frame = -1;
        framenum++;
    } while (return_frame < -1);

    return return_frame;
}

int find_frame(unsigned int id)
{
    int region = 0, return_frame;

    do
    {
        return_frame = find_frame_in_region(id, region);
        region++;
    } while ((return_frame == -1) && (region < NUMREGIONS));

    return return_frame;
}

void save_frame(unsigned int id)
/* Mark a frame inviolate by making all frames that point to it, point */
/* instead to its next_frame                                          */
{
    int     saveframenum, framenum, new_next_frame;

    saveframenum = find_frame(id);
    if (saveframenum >= 0)
    {
        new_next_frame = fb.fs[saveframenum].next_frame;
        framenum = (saveframenum / FRAMES_PER_REGION) * FRAMES_PER_REGION;
        while (fb.fs[framenum].id != END_FRAME)
        {
            if (fb.fs[framenum].next_frame == saveframenum)
                fb.fs[framenum].next_frame = new_next_frame;
            framenum++;
        }
    }
}

void unsave_frame(unsigned int id)
/* Re-mark frame A violate by tracing around the region's closed loop */
/* until it finds a frame Z with the same next_frame as frame A: it  */
/* then makes frame Z's next_frame point to frame A.      */
/* WARNING: WILL LOCK IF NEXT_FRAME DOES NOT LOOP PROPERLY IN REGION */
{
    int     saveframenum, framenum, old_next_frame;

    saveframenum = find_frame(id);
    if (saveframenum >= 0)
    {
        old_next_frame = fb.fs[saveframenum].next_frame;
        framenum = old_next_frame;
        while (fb.fs[framenum].next_frame != old_next_frame)
            framenum = fb.fs[framenum].next_frame;
        fb.fs[framenum].next_frame = saveframenum;
    }
}

void dump_frame_raw(IPP_msg_buffer msg, IPP_address dest)
/* Dump verbatim the contents of the Frame Buffer allocated to the */
/* requested frame.                                                */
{
    frame_specs_struct *fs;
    unsigned int val;
    unsigned int id;
    fb_ptr frame_ptr;
    int     framenum, size_left, size_sent, i;
    int     *win_ptr;

    id = msg[0] + (msg[1] << 16);
    framenum = find_frame(id);
    if (framenum < 0)
        Send_Host_Message(dest, BEP_UNKNOWN_ID, BEP_UNKNOWN_ID_PRIORITY,
            2, msg);
    else
    {
        fs = &(fb.fs[framenum]);
        size_left = fs->rp->size;
        frame_ptr = fs->frame_start;
        msg[2] = DUMP_FIRST_MESSAGE;
        while (size_left > 0)
        {
            win_ptr = setup_dual_window(WY1, frame_ptr);
            size_sent = min(size_left, (IPP_DATASIZE / 2) - 4);
            for (i = 0; i < size_sent; i++)
            {
                val = read_fb_win(win_ptr);
                msg[2*i+3] = val & 0xFFFF;
                msg[2*i+4] = val >> 16;
                win_ptr++;
            }
            size_left -= size_sent;
            frame_ptr += size_sent;
            if (size_left <= 0) msg[2] |= DUMP_LAST_MESSAGE;
            Send_Host_Message(dest, BEP_RAW_FRAME, FRAME_DUMP_PRIORITY,
                (size_sent*2)+3, msg);
            msg[2] = DUMP_MIDDLE_MESSAGE;
            check_buddy_time();
        }
    }
}

void dump_imp_array(IPP_msg_buffer msg, IPP_address dest)
/* Dump a raster-format array of image pixels from the requested frame */
{
    frame_specs_struct *fs;
    unsigned short *msg_ptr;
    unsigned int id;
    int     framenum, line, msgspc_left, line_left_off, left_in_line;
    int     cam, scol, ecol, erow, numcols, lines_get;

    id = msg[0] + (msg[1] << 16);
```

```
    cam = msg[3];
    framenum = find_frame(id);
    if (framenum < 0)
        Send_Host_Message(dest, BEP_UNKNOWN_ID, BEP_UNKNOWN_ID_PRIORITY,
            2, msg);
    else
    {
        fs = &(fb.fs[framenum]);
        msg[4] = max(msg[4], fs->rp->start_col[cam]);
        msg[5] = min(msg[5], fs->rp->end_col[cam]);
        msg[6] = max(msg[6], fs->rp->start_row[cam]);
        msg[7] = min(msg[7], fs->rp->end_row[cam]);
        scol = msg[4];
        ecol = msg[5];
        line = msg[6];
        erow = msg[7];
        numcols = ecol - scol;
        line_left_off = ecol;
        msg[2] = DUMP_FIRST_MESSAGE;
/* Loop until we've got all the lines requested */
        while (line < erow)
        {
/* msg[0-7] is header; msg[8-511] is data */
            msg_ptr = &(msg[8]);
            msgspc_left = IPP_DATASIZE - 8;

/* If there is data left in the current line, start the new message with it */
/* If the rest of the line fits in the message, go to the next line.  */
            left_in_line = ecol - line_left_off;
            if (left_in_line > 0)
            {
                left_in_line = min(left_in_line, msgspc_left);
                read_imp(cam, line_left_off, line_left_off + left_in_line,
                    line, line+1, (int *)msg_ptr, fs);
                if ((line_left_off + left_in_line) == ecol)
                {
                    line++;
                    line_left_off = 0;
                }
                else
                {
                    line_left_off += left_in_line;
                }
                msg_ptr += left_in_line;
                msgspc_left -= left_in_line;
            }

            if (msgspc_left > 0)
            {
/* If there is any room at all left in the message, see how many full */
/* lines we can fit in it, and read them as a box.      */
                lines_get = min((msgspc_left / numcols), (erow - line));
                read_imp(cam, scol, ecol, line, line + lines_get,
                    (int *)msg_ptr, fs);
                line += lines_get;
                msg_ptr += lines_get * numcols;
                msgspc_left -= lines_get * numcols;

/* If the last line has been read, note that this is the final message */
                if (line >= erow) msg[2] |= DUMP_LAST_MESSAGE;
                else
                {
/* Otherwise, read as much of the next line as will fit in the message, */
/* trusting that the next loop around will grab the rest of this line */
                    read_imp(cam, scol, scol+msgspc_left, line, line+1,
                        (int *)msg_ptr, fs);
                    line_left_off = msgspc_left;
                    msgspc_left = 0;
                }
            }

/* Send the message */
            Send_Host_Message(dest, BEP_DEALT_IMPS, FRAME_DUMP_PRIORITY,
                IPP_DATASIZE - msgspc_left, msg);
/* Following messages will be either middle or end messages */
            msg[2] = DUMP_MIDDLE_MESSAGE;
/* Tell the GPP we're still alive, just in case this dump is taking a while */
            check_buddy_time();
        }
    }
}

void dump_tbt_array(IPP_msg_buffer msg, IPP_address dest)
/* Dump a quick 3x3-pixel array */
{
    unsigned int id;
    int     framenum, cam, llx, lly;

    id = msg[0] + (msg[1] << 16);
    cam = msg[3];
    llx = msg[4];
    lly = msg[5];
    framenum = find_frame(id);
    if (framenum < 0)
        Send_Host_Message(dest, BEP_UNKNOWN_ID, BEP_UNKNOWN_ID_PRIORITY,
            2, msg);
    else
      {
        tbt(cam, llx, lly, (int *)(&(msg[6])), &(fb.fs[framenum]));
        Send_Host_Message(dest, BEP_TBT, TBT_PRIORITY, 15, msg);
        check_buddy_time();
      }
}

void set_up_region(int region)
/* Set up the frame_specs_structs within this newly-configured region */
{
    int i, istart, iend;

/* Currently an absolute allocation of frame indices to regions */
    istart = region * FRAMES_PER_REGION;
/* Fit as many frames as possible in the region */
    iend = min(istart + ((fb.region_start[region+1] - fb.region_start[region])
        / fb.rp[region].size), istart + FRAMES_PER_REGION);
/* First frame_specs_struct: rp points to this region's region_params_ */
/* struct, frame_start is at the floor of the region, the next frame is */
/* at the next frame index, and the ID is reset.      */
    fb.fs[istart].rp = &(fb.rp[region]);
    fb.fs[istart].frame_start = fb.region_start[region];
    fb.fs[istart].next_frame = istart + 1;
    fb.fs[istart].id = END_FRAME;
/* For each following frame, increase frame_start and next_frame */
    for (i = istart + 1; i < iend; i++)
    {
        fb.fs[i] = fb.fs[i-1];
        fb.fs[i].frame_start += fb.fs[i].rp->size;
        fb.fs[i].next_frame++;
    }
/* Make the last frame point to the first, forming a closed loop */
    fb.fs[iend-1].next_frame = istart;
/* Make all unused frame_specs_structs point back to the first frame */
    for (i = iend; i < istart + FRAMES_PER_REGION; i++)
        fb.fs[i] = fb.fs[i-1];
/* Start at the first frame */
```

```
    fb.current_framenum[region] = istart;
}

void fill_out_params(region_param_struct *rp)
/* Perform precalculations on region parameters, for quick access later */
{
    int     i, act_rows, h_nips, compression, slices_per_cam;
    flagokay;

/* Mode 3 has two pixels per word; all other modes have one */
    if (rp->ccd.mode == 3)
    {
        compression = 2;
/* Mode 3 deal-on-write frames must have even horizontal parameters,    */
/* because the write routines grab two pixels at a time and compress    */
/* them.  num_hucs, num_cols, and num_hocs will be even by nature of  */
/* the Lasagna Box, which is Good.  However we also need the number of */
/* pixels _saved per slice_ to be even, which we can guarantee by  */
/* making start_ and end_col even; so we expand the range of columns */
/* the GPP asked us to save by one or two columns if we have to.  */
        if (rp->deal)
        {
            for (i = 0; i < NUM_CAMERAS; i++)
            {
                make_even_minus(rp->start_col[i]);
                make_even_plus(rp->end_col[i]);
            }
        }
    }
    else compression = 1;

/* Modes 0 and 3 use four slices per camera; Modes 1 and 2  */
/* use two.                                                 */
    if ((rp->ccd.mode == 0) || (rp->ccd.mode == 3)) slices_per_cam = 4;
    else slices_per_cam = 2;

/* Calculate the total number of columns per slice, and rows per frame */
    rp->ccd.total_cols = rp->ccd.num_hucs+rp->ccd.num_cols+rp->ccd.num_hocs;
    rp->ccd.total_rows = rp->ccd.num_rows + rp->ccd.num_vocs;

/* Calculate the number of horizontal NIPs per slice */
    h_nips = rp->ccd.num_hucs + rp->ccd.num_hocs;
/* Use size as a running counter of the FB words allocated so far per frame */
    rp->size = 0;
    for (i = 0; i < NUM_CAMERAS; i++)
    {
        if (rp->deal)
        {
/* For each camera in a deal-on-write frame: calculate the actual number of */
/* rows to be saved; then check if this camera's data will be saved at all */
            act_rows = rp->end_row[i] - rp->start_row[i];

            if ((rp->ccd.used_cameras & (1 << i)) == 0) okay = 0;
            else okay = 1;

/* linelen is the distance from start_col to end_col, divided by two if  */
/* there are two pixels per word, zero'd if this camera's data is not   */
/* present or not to be saved.                                          */
        rp->linelen[i] = ((rp->end_col[i] - rp->start_col[i]) / compression)
            * okay;

/* Note where the image pixels for this frame will be stored relative  */
/* to the bottom of the frame in the Frame Buffer, then increment size */
/* to allocate space for these pixels                  */
            rp->imp_start_offset[i] = rp->size * okay;
            rp->size += rp->linelen[i] * act_rows * okay;
            if (rp->save_nips)
            {
/* If NIPs are to be saved, note where the horizontal NIPs will be stored; */
/* increment size to allocate space for them; then note where the VOCS will */
/* be stored; increment size to allocate space for them.     */
                rp->nip_start_offset[i] = rp->size * okay;
                rp->size += ((slices_per_cam * h_nips) / compression)
                    * act_rows * okay;
                rp->voc_start_offset[i] = rp->size * okay;
                rp->size += ((slices_per_cam * rp->ccd.total_cols)
                    / compression) * rp->ccd.num_vocs * okay;
            }
            else
            {
/* If NIPs are not to be saved, zero their offsets */
                rp->nip_start_offset[i] = 0;
                rp->voc_start_offset[i] = 0;
            }
        }
        else
        {
/* For deal-on-write frames, linelen is 4 slices' worth of words (or 8 */
/* slices in Mode 3, but at 2 pixels per word, it's the same thing); */
/* data is not divided, so everything starts at the bottom of the frame */
            rp->linelen[i] = 4*rp->ccd.total_cols;
            rp->imp_start_offset[i] = 0;
            rp->nip_start_offset[i] = 0;
            rp->voc_start_offset[i] = 0;
            rp->size = rp->linelen[i] * rp->ccd.total_rows;
        }
    }
}
```

```
/* Main module of BEP.  Initializes and engages forever-loop.  Contains */
/* simulation- and memory-related test routines.      */

#include <stdlib.h>
#include "BEPincl.h"
#include "BEPglobals.h"

#define vb_write(val) __asm("move %0,x:(R5)+" : : "S"(val));

/* How many sets of 4 zeros to drop into the video buffer before  */
/* writing a frame sync                                           */
#define BUNCH_OF_ZEROS 1
/* Flag to indicate whether a frame sync should be inserted into the */
/* simulated video data stream                                    */
int drop_frame_sync=0;

/* Simulated video data */
/*int ccd0sim[112] =
    { 97, 98, 99,100, 101,102,103,104, 104,103,102,101, 100, 99, 98, 97,
      98, 99,100,101, 102,103,104,105, 105,104,103,102, 101,100, 99, 98,
      99,100,101,102, 103,104,105,106, 106,105,104,103, 102,101,100, 99,
      98, 99,100,101, 102,103,104,105, 105,104,103,102, 101,100, 99, 98,
      97, 98, 99,100, 101,102,103,104, 104,103,102,101, 100, 99, 98, 97,
      96, 97, 98, 99, 100,101,102,103, 103,102,101,100,  99, 98, 97, 96,
      95, 96, 97, 98,  99,100,101,102, 102,101,100, 99,  98, 97, 96, 95};*/

int ccd0sim[112] =
    { 97, 98, 99,100, 101,102,103,100, 100,103,102,101, 100, 99, 98, 97,
      98, 99,100,101, 102,103,104,100, 100,104,103,102, 101,100, 99, 98,
      99,100,101,102, 103,104,105,100, 100,105,104,103, 102,101,100, 99,
      98, 99,100,101, 102,103,104,100, 100,104,103,102, 101,100, 99, 98,
      97, 98, 99,100, 101,102,103,100, 100,103,102,101, 100, 99, 98, 97,
      96, 97, 98, 99, 100,101,102,103, 103,102,101,100,  99, 98, 97, 96,
      95, 96, 97, 98,  99,100,101,102, 102,101,100, 99,  98, 97, 96, 95};

int ccd1sim[112] =
    {100,100,100,100, 100,100,100,100, 100,100,100,100, 100,100,100,100,
     100,200,200,100, 100,100,100,200, 100,100,200,100, 100,100,200,100,
     100,100,100,100, 100,100,100,100, 100,100,100,100, 100,100,100,100,
     200,100,100,100, 200,100,100,100, 200,100,100,100, 200,100,100,200,
     200,100,100,100, 100,100,100,100, 100,100,100,100, 100,100,100,200,
     100,100,100,100, 100,100,100,100, 100,100,100,100, 100,100,100,100,
     100,100,100,200, 200,100,100,200, 200,100,100,200, 200,100,100,200};



void sim4(pat4)
int pat4;
{
    int i;

    if (pat4 == ZEROS4)
    {
        for (i=0; i<4; i++) vb_write(ZERO_MARK);
    }
    else
    {
        for (i=0; i<4; i++) vb_write(ONE_MARK);
    }
}

void simulate_frame_sync()
{
    static int id = 0x555555;
    int i;

    for (i=0; i<BUNCH_OF_ZEROS; i++) sim4(ZEROS4);
    sim4(FRAME_SYNC1);
    sim4(FRAME_SYNC2);
    for (i=NUMIDBITS-1; i>=0; i--)
    {
        if ((id & (1<<i)) == 0) sim4(ZEROS4);
        else sim4(ONES4);
    }
    id++;
}

void simulate_row_sync()
{
    int i;

    for (i=0; i<ROW_SYNC_SKIP; i++) vb_write(0);
    sim4(ROW_SYNC1);
    sim4(ROW_SYNC2);
}

void simulate_word_rcvd()
{
    static int ABCDcnt=0, Aline=0, Bline=7, Cline=8, Dline=15;
    int sim_cntr;

    if (drop_frame_sync)
    {
        drop_frame_sync=0;
        simulate_frame_sync();
        ABCDcnt=0;
        Aline=0;  Bline=7;  Cline=8;  Dline=15;
    }
    if (ABCDcnt==0) simulate_row_sync();

    sim_cntr = (ccd0sim[Aline+ABCDcnt]*0x1000) + ccd0sim[Bline-ABCDcnt];
    vb_write(sim_cntr);
    sim_cntr = (ccd0sim[Cline+ABCDcnt]*0x1000) + ccd0sim[Dline-ABCDcnt];
    vb_write(sim_cntr);
    sim_cntr = (ccd1sim[Aline+ABCDcnt]*0x1000) + ccd1sim[Bline-ABCDcnt];
    vb_write(sim_cntr);
    sim_cntr = (ccd1sim[Cline+ABCDcnt]*0x1000) + ccd1sim[Dline-ABCDcnt];
    vb_write(sim_cntr);

    ABCDcnt++;
    if (ABCDcnt > 3)
    {
        Aline += 16;
        if (Aline > 96)
        {
            Aline = 0; Bline = 7;
            Cline = 8; Dline = 15;
        }
        else
        {
            Bline += 16;
            Cline += 16;
            Dline += 16;
        }
        ABCDcnt = 0;
    }
}

void alt_simulate_word_rcvd(int vb_linelen)
{
    static unsigned int sim_cntr = 1;
    static int line_written = 0;
    int i;
```

```
    if (drop_frame_sync)
    {
        drop_frame_sync = 0;
        line_written = 0;
        simulate_frame_sync();
    }

    for (i=0; i<10; i++)
    {
        if (line_written == 0) simulate_row_sync();
        vb_write(sim_cntr);
        sim_cntr += 0x2002;
/*      if (sim_cntr >= (0x700*0x1000)) sim_cntr = 1;*/
        line_written++;
        if (line_written >= vb_linelen-ROW_SYNC_LEN) line_written = 0;
    }
}

void xmemcpy(start, end, copy_space)
unsigned short start,end;
unsigned short *copy_space;
{
    unsigned short i;
    int     *ad, val;

    for (i=start; i<=end; i++)
    {
        ad = (int *)i;
        __asm("move x:(%1),%0" : "=S"(val) : "A"(ad));
        *copy_space++ = val & 0xFFFF;
        *copy_space++ = val >> 16;
    }
}

void ymemcpy(start, end, copy_space)
unsigned short start,end;
unsigned short *copy_space;
{
    unsigned short i;
    int     val;

    for (i=start; i<=end; i++)
    {
        val = *((int *)i);
        *copy_space++ = val & 0xFFFF;
        *copy_space++ = val >> 16;
    }
}

void pmemcpy(start, end, copy_space)
unsigned short start,end;
unsigned short *copy_space;
{
    unsigned short i;
    int     *ad, val;

    for (i=start; i<=end; i++)
    {
        ad = (int *)i;
        __asm("movem p:(%1),%0" : "=S"(val) : "A"(ad));
        *copy_space++ = val & 0xFFFF;
        *copy_space++ = val >> 16;
    }
}

void memdump_cmd(IPP_msg_buffer msg)
{
    unsigned short start, end;
    charspace;

    space = (char)msg[1];
    start = msg[2];
    end = msg[3];

/*  put_host((unsigned short)space);
    put_host(start);
    put_host(end);*/

    if ((end-start) > ((IPP_DATASIZE-10) / 2))
    {
        end = start + ((IPP_DATASIZE-10) / 2);
        msg[3] = end;
    }

    switch(space)
    {
        case 'x': xmemcpy(start, end, &(msg[4]));  break;
        case 'y': ymemcpy(start, end, &(msg[4]));  break;
        case 'p': pmemcpy(start, end, &(msg[4]));  break;
    }

    Send_Host_Message(testmsg_dest, VIDDSP_RESPONSE, VIDDSP_PRIORITY,
        (2 * ((end-start)+1) ) + 4, msg);
}

void mempoke_cmd(IPP_msg_buffer msg)
{
    charspace;
    int     *ad, val;

    space = (char)msg[1];
    ad = (int *)msg[2];
    val = msg[3];
    val |= msg[4] << 16;
    switch(space)
    {
        case 'x': __asm volatile("move %0,x:(%1)" : : "S"(val), "A"(ad));
                  break;
        case 'y': *(ad) = val;
                  break;
        case 'p': __asm volatile("movem %0,p:(%1)" : : "S"(val), "A"(ad));
                  break;
    }
}

void memdump_auto(char space, unsigned short start, unsigned short end)
{
    IPP_msg_buffer msg;

    msg[0] = VD_MEMDUMP;
    msg[1] = (unsigned short)space;
    msg[2] = start;
    msg[3] = end;
    memdump_cmd(msg);
}


void main(void)
{
    vidbuf_param_struct vidbuf_params, next_vidbuf_params;

    onetime_init(&vidbuf_params, &next_vidbuf_params);
```

```
    for(;;)
    {
        check_buddy_time();
        try_get_message(&vidbuf_params, &next_vidbuf_params);
        if (doloop)
        {
            if (simulating)
            {
                if (vidbuf_params.waiting_for_frame_sync) drop_frame_sync = 1;
                alt_simulate_word_rcvd(vidbuf_params.one_line_delta);
            }
            try_copy_line(&vidbuf_params, &next_vidbuf_params);
            try_relay_errors();
        }
    }
}
```

```
/* Initialization routines, including adjustment to vidbuf_params */
/* in response to changing frames / parameters      */

#include <stdlib.h>
#include "BEPincl.h"

extern IPP_address me, video_dest, testmsg_dest;

extern all_frames_struct fb;

extern int simulating;


void init_portc_intrs()
{
/* If we're not in simulation mode, turn on Interrupts */
    if (simulating)
    {
        __asm("bclr #m_rie,x:<<m_scr");
    }
    else
    {
        __asm("bset #m_rie,x:<<m_scr");
    }
}

void set_vbp_left_offs(vidbuf_param_struct *vbp)
/* Set the _left_off variables in vbp to the beginning of the */
/* current frame (i.e. the current_framenum of vbp->region)  */
{
    int i;

    for (i = 0; i < NUM_CAMERAS; i++)
    {
        vbp->imp_left_off[i] = f_by_vbp(vbp).frame_start +
            f_by_vbp(vbp).rp->imp_start_offset[i];
        vbp->nip_left_off[i] = f_by_vbp(vbp).frame_start +
            f_by_vbp(vbp).rp->nip_start_offset[i];
    }
}

void set_vbp_new_config(vidbuf_param_struct *vbp)
/* Set lobound_delta, hibound_delta, and one_line_delta in response to */
/* a change in region parameters                                       */
{
    int i, j, range_lo, range_hi, testcol, slice;

    for (i = 0; i < NUM_CAMERAS; i++)
    {
        for (j = 0; j < SLICES_PER_CAMERA; j++)
        {
            slice = (i << 2) + j;
/* By default, assume all of the image pixels in this slice will be  */
/* saved: so lobound_delta is set to skip the packages-of-four     */
/* containing HUCs, and hibound_delta is set to include the packages- */
/* of-four containing IMPs.                                          */
            vbp->lobound_delta[slice] = 4*f_by_vbp(vbp).rp->ccd.num_hucs;
            vbp->hibound_delta[slice] = vbp->lobound_delta[slice] +
                4*f_by_vbp(vbp).rp->ccd.num_cols;

/* Define the range of image columns included in this slice    */
            range_lo = j * f_by_vbp(vbp).rp->ccd.num_cols;
            range_hi = ((j+1) * f_by_vbp(vbp).rp->ccd.num_cols);

/* First check the start_col against this range       */
            testcol = f_by_vbp(vbp).rp->start_col[i];
            if (testcol >= range_hi)
            {
/* If start_col is after this slice, save nothing from this slice  */
                vbp->lobound_delta[slice] = 0;
                vbp->hibound_delta[slice] = 0;
            }
            else if ((testcol > range_lo) && (testcol < range_hi))
            {
/* If start_col is within this slice, adjust lobound_delta upward if it */
/* is Slice A or C, or hibound_delta downward if it is B or D.   */
                if ((j & 1) == 0)
                    vbp->lobound_delta[slice] += 4 * (testcol - range_lo);
                else vbp->hibound_delta[slice] -= 4 * (testcol - range_lo);
            }

/* Now check the end_col against this range         */
            testcol = f_by_vbp(vbp).rp->end_col[i];
            if (testcol < range_lo)
            {
/* If end_col is before this slice, save nothing from this slice  */
                vbp->lobound_delta[slice] = 0;
                vbp->hibound_delta[slice] = 0;
            }
            else if ((testcol >= range_lo) && (testcol < range_hi))
            {
/* If end_col is within this slice, adjust hibound_delta downward if it */
/* is Slice A or C, or lobound_delta upward if it is B or D.   */
                if ((j & 1) == 0)
                    vbp->hibound_delta[slice] -= 4 * (range_hi - testcol);
                else vbp->lobound_delta[slice] += 4 * (range_hi - testcol);
            }
        }
    }

/* One row in the video buffer is composed of the blank words preceding */
/* the row sync, the row sync itself, and total_cols packages-of-four */
    vbp->one_line_delta = ROW_SYNC_LEN + (4*f_by_vbp(vbp).rp->ccd.total_cols);
}


void onetime_init(vbp, vbpnew)
vidbuf_param_struct *vbp, *vbpnew;
{
    int mod, region, j;

    copy_lowp_routines();

    me = IPP_register_address(BEP);
    video_dest = IPP_get_address(DEFAULT_VIDEO_DEST_BEP);
    testmsg_dest = IPP_get_address(DEFAULT_TESTMSG_DEST);

    my_ipp_addr = BEP;
    prepare_IPP_headers();

    asm_inptr = VIDBUF_BOTTOM;
    mod = VIDBUF_LEN-1;
    __asm("move %0,M5" : : "S"(mod));

    simulating = 0;

    init_communications();
    init_intr_vectors();
    init_portc_intrs();

    fb.region_start[0] = 0;
    for (region = 0; region < NUMREGIONS; region++)
    {
```

```
        fb.rp[region].ccd.mode = 3;
        fb.rp[region].ccd.used_cameras = 3;
        fb.rp[region].ccd.num_hucs = 4;
        fb.rp[region].ccd.num_cols = 16;
        fb.rp[region].ccd.num_hocs = 4;
        fb.rp[region].ccd.num_rows = 16;
        fb.rp[region].ccd.num_vocs = 4;
        for (j = 0; j < NUM_CAMERAS; j++)
        {
            fb.rp[region].start_row[j] = 0;
            fb.rp[region].end_row[j] = 16;
            fb.rp[region].start_col[j] = 0;
            fb.rp[region].end_col[j] = 64;
        }
        fb.rp[region].deal = 0;
        fb.rp[region].save_nips = 1;
        fb.region_start[region+1] = fb.region_start[region] +
            (MAX_FB_ADDRESS / NUMREGIONS);
        fill_out_params(&fb.rp[region]);
        set_up_region(region);
    }

    vbpnew->region = 0;
    vbpnew->lines_rcvd = 0;
    vbpnew->waiting_for_frame_sync = 0;
    vbpnew->save_frame = 1;
    vbpnew->single_frame = 1;
    set_vbp_new_config(vbpnew);

    vbp->one_line_delta = vbpnew->one_line_delta;
    vbp->save_frame = 0;
    vbp->left_off = VIDBUF_BOTTOM;
    abort_stream(vbp, vbpnew);
}
```

```
/* Routines to synchronize with the data in the video buffer, and write */
/* it to the Frame Buffer, dealing on write if called for    */

#include "BEPincl.h"

/* Read from the video buffer (effectively val = *ad, but in X: space) */
#define read_vb(ad, val) __asm("move x:(%1),%0" : "=S"(val) : "A"(ad));

/* True if there are num words available in the video buffer between */
/* read (following) and write (leading)              */
#define have_words(read, write, num) (vb_distance(read, write) > num)

extern all_frames_struct fb;


int *vb_add(int *addend1, int addend2)
/* Go ad2 words from ad1, within the video buffer */
{
    int *result;

    result = addend1 + addend2;
    if (result >= VIDBUF_TOP) result -= VIDBUF_LEN;
    else if (result < VIDBUF_BOTTOM) result += VIDBUF_LEN;

    return (int *)((int)result & 0x00FFFF);
}

int vb_distance(int *from, int *to)
/* The circular-buffer distance from from to to */
{
    int result;

    result = (int)(to-from);
    if (result < 0) result += VIDBUF_LEN;

    return result;
}

int check4(int *ptr)
/* Check the four words following ptr, and see if they */
/* all represent a one mark, a zero mark, or neither */
{
    int *ptrwork;
    int orig_val, test_val;
    int retval = MIX4;

    read_vb(ptr, orig_val);
    ptrwork = vb_add(ptr, 1);
    read_vb(ptrwork, test_val);
    if (test_val == orig_val)
    {
        ptrwork = vb_add(ptrwork, 1);
        read_vb(ptrwork, test_val);
        if (test_val == orig_val)
        {
            ptrwork = vb_add(ptrwork, 1);
            read_vb(ptrwork, test_val);
            if (test_val == orig_val)
            {
                if (orig_val == ZERO_MARK) retval = ZEROS4;
                else if (orig_val == ONE_MARK) retval = ONES4;
            }
        }
    }
    return retval;
}

void switch_params(vidbuf_param_struct *current, vidbuf_param_struct *new)
{
    if (current->save_frame)
        fb.current_framenum[current->region] = f_by_vbp(current).next_frame;
    new->left_off = current->left_off;
    *current = *new;
    set_vbp_left_offs(current);
    if (current->single_frame) new->save_frame = 0;
    if (!current->save_frame) current->waiting_for_frame_sync = 1;
}

int sync_frame(vidbuf_param_struct *vbp, unsigned int *id)
{
    static int state=0;
    static int count=0;
    static int local_id=0;
    int *inptr, *idptr=0;
    int gotsync=0;
    int val1, val2;

    inptr = asm_inptr;

    while (have_words(vbp->left_off, inptr, 4) && !gotsync)
    {
        switch(state)
        {
            case 0: read_vb(vbp->left_off, val1);
                    vbp->left_off = vb_add(vbp->left_off, 1);
                    read_vb(vbp->left_off, val2);
                    if ((val1 == ZERO_MARK) && (val2 == ONE_MARK)) state = 1;
                    break;
            case 1: if (check4(vbp->left_off) == FRAME_SYNC1)
                    {
                        vbp->left_off = vb_add(vbp->left_off, 4);
                        state = 2;
                    } else state = 0;
                    break;
            case 2: if (check4(vbp->left_off) == FRAME_SYNC2)
                    {
                        vbp->left_off = vb_add(vbp->left_off, 4);
                        idptr = vbp->left_off;
                        count = 1;
                        local_id = 0;
                        state = 3;
                    } else state = 0;
                    break;
            case 3: val1 = check4(vbp->left_off);
                    if (val1 == MIX4)
                    {
                        vbp->left_off = idptr;
                        state = 0;
                    }
                    else
                    {
                        if (val1 == ONES4) local_id |= 1 << (NUMIDBITS-count);
                        vbp->left_off = vb_add(vbp->left_off, 4);
                        count++;
                        if (count > NUMIDBITS)
                        {
                            gotsync = 1;
                            *id = local_id;
                            state = 0;
                        }
                    }
                    break;
        }
```

```
    }
    return gotsync;
}

int sync_row(vidbuf_param_struct *vbp)
{
    int *inptr;
    int val1, val2;
    int gotsync = 0;

    inptr = asm_inptr;
    if (have_words(vbp->left_off, inptr, ROW_SYNC_LEN) && !gotsync)
    {
        vbp->left_off = vb_add(vbp->left_off, ROW_SYNC_SKIP);
        val1 = check4(vbp->left_off);
        vbp->left_off = vb_add(vbp->left_off, 4);
        val2 = check4(vbp->left_off);
        vbp->left_off = vb_add(vbp->left_off, 4);
        if ((val1 == ROW_SYNC1) && (val2 == ROW_SYNC2)) gotsync = 1;
        else gotsync = -1;
    }
    return gotsync;
}

void deal_into_fb(vidbuf_param_struct *vbp, int slice_num, int slice_cam,
    flag nip, int lobound_delta, int hibound_delta)
/* Deal the pixels from the specified slice which are between  */
/* vbp->left_off + lobound_delta and vbp->left_off + hibound_delta  */
/* in the video buffer, into the Frame Buffer.      */
{
    flaginvalid = 0, pix_offset = 0;
    fb_ptr *fb_ptr_add;
    int     *fb_win_ptr, *lobound, *hibound;
    int     *start, direction, distance;

/* Calculate which word in a package-of-four contains this slice's data */
    switch(f_by_vbp(vbp).rp->ccd.mode)
    {
        case 0: pix_offset = slice_num;
                break;
        case 1: if ((slice_num & 1) == 0) invalid = 1;
                else pix_offset = (slice_num >> 1) + 2*slice_cam;
                break;
        case 2: if ((slice_num & 1) != 0) invalid = 1;
                else pix_offset = (slice_num >> 1) + 2*slice_cam;
                break;
        case 3: pix_offset = (slice_num >> 1) + 2*slice_cam;
                break;
    }

    if (!invalid)
    {
/* Start writing where we left off with this type of pixel in the FB */
        if (nip) fb_ptr_add = &(vbp->nip_left_off[slice_cam]);
        else fb_ptr_add = &(vbp->imp_left_off[slice_cam]);
/* Setup WX0 and WX1 for the write */
        fb_win_ptr = setup_dual_window(WX0, *fb_ptr_add);

/* Get the bounding vidbuf addresses */
        lobound = vb_add(vbp->left_off,lobound_delta);
        hibound = vb_add(vbp->left_off,hibound_delta);

/* If we're saving image pixels from Slice B or D, start at the high */
/* bound and work backward by four words (to undo the "backwards"  */
/* readout of these slices); otherwise, start the low bound and work */
/* forward by four words.                                          */
        if (((slice_num & 1) == 0) || nip)
        {
            direction = 4;
            start = vb_add(lobound, pix_offset);
        }
        else
        {
            direction = -4;
            start = vb_add(hibound, pix_offset-4);
        }

/* "distance" is the number of *words to be written to the FB*. For 16- */
/* bit pixels, this is the distance between the bounds divided by four, */
/* as there is one pixel per word and one pixel per package-of-four.   */
/* For 12-bit pixels, this is the distance between the bounds divided */
/* by eight, because there is still one pixel per package-of-four, but */
/* two pixels are grabbed at a time to compose a single word to be  */
/* written into the FB.                                              */
        if (f_by_vbp(vbp).rp->ccd.mode == 3)
        {
            distance = vb_distance(lobound,hibound) >> 3;
/* Dispatch to copy_hi_pixels for Slice A and C, copy_lo_pixels for B and D */
            if ((slice_num & 1) == 0)
                    copy_hi_pixels(start, fb_win_ptr, direction, distance);
                else copy_lo_pixels(start, fb_win_ptr, direction, distance);
        }
        else
        {
            distance = vb_distance(hibound,lobound) >> 2;
            copy_words(start, fb_win_ptr, direction, distance);
        }

/* Increment the imp_left_off or nip_left_off to reflect the words */
/* just written                                                    */
        *fb_ptr_add += distance;
    }
}

void copy_line(vidbuf_param_struct *vbp)
/* Copy the line just received into the video buffer, into the FB. */
/* Deal the pixels if called for, otherwise copy them verbatim.  */
{
    int *fb_win_ptr;
    int cam, slicenum, srow, erow, lbd, hbd;

    if (f_by_vbp(vbp).rp->deal)
    {
        for (cam = 0; cam < NUM_CAMERAS; cam++)
        {
            if ((f_by_vbp(vbp).rp->ccd.used_cameras & (1 << cam)) != 0)
            {
                srow = f_by_vbp(vbp).rp->start_row[cam];
                erow = f_by_vbp(vbp).rp->end_row[cam];
/* For each camera: check if we're saving this camera's data; if so, */
/* cycle through each slice in the camera           */
                for (slicenum = 0; slicenum < SLICES_PER_CAMERA; slicenum++)
                {
                    if (f_by_vbp(vbp).rp->save_nips)
                    {
/* If we're saving NIPs, check if we're in the VOC area; if so, save */
/* the entire line read out from this slice into the NIP area.  If  */
/* we're not in the VOC area yet, save the HUCs, then the HOCs.    */
                        if (vbp->lines_rcvd > f_by_vbp(vbp).rp->ccd.num_rows)
                        {
                            deal_into_fb(vbp, slicenum, cam, 1, 0,
                                4 * f_by_vbp(vbp).rp->ccd.total_cols);
                        }
                        else if ((srow + vbp->lines_rcvd) <= erow)
```

```
                    {
                        deal_into_fb(vbp, slicenum, cam, 1, 0,
                            4 * f_by_vbp(vbp).rp->ccd.num_hucs);
                        lbd = 4 * (f_by_vbp(vbp).rp->ccd.num_hucs +
                            f_by_vbp(vbp).rp->ccd.num_cols);
                        hbd = lbd + 4*(f_by_vbp(vbp).rp->ccd.num_hocs);
                        deal_into_fb(vbp, slicenum, cam, 1, lbd, hbd);
                    }
                }
/* Grab the precalculated bound_deltas, and if this slice contributes */
/* to the image to be saved, deal the contributing part of the slice */
/* into the IMP area.                                                */
                lbd = vbp->lobound_delta[(cam << 2) + slicenum];
                hbd = vbp->hibound_delta[(cam << 2) + slicenum];
                if ((lbd != hbd) && ((srow + vbp->lines_rcvd) <= erow))
                    deal_into_fb(vbp, slicenum, cam, 0, lbd, hbd);
            }
        }
    }
    else
    {
/* If this is a deal-on-read frame, set up WX0 and WX1, copy the line */
/* from the video buffer to these windows verbatim, and increment  */
/* imp_left_off[0] to reflect the line just written.     */
        fb_win_ptr = setup_dual_window(WX0, vbp->imp_left_off[0]);
        copy_words(vbp->left_off, fb_win_ptr, 1,
            4*f_by_vbp(vbp).rp->ccd.total_cols);
        vbp->imp_left_off[0] += 4*f_by_vbp(vbp).rp->ccd.total_cols;
    }
}

void try_copy_line(vidbuf_param_struct *vbp, vidbuf_param_struct *vbpnew)
{
    unsigned int id;
    int row_syncd;
    int *inptr;

    if (vbp->waiting_for_frame_sync)
    {
/* If we're in waiting-for-frame-sync mode, try to sync the frame */
        vbp->waiting_for_frame_sync = !sync_frame(vbp, &id);
        if (!vbp->waiting_for_frame_sync)
        {
/* If we've just found the frame sync, make vbpnew current */
            switch_params(vbp, vbpnew);
            if (vbp->save_frame)
            {
/* If we're processing this frame, record the ID and tell the GPP */
/* we've sync'd it                                                */
                f_by_vbp(vbp).id = id;
                signal_frame_started(f_by_vbp(vbp).id);
            }
        }
    }
    else
    {
        inptr = asm_inptr;
        if (have_words(vbp->left_off, inptr, vbp->one_line_delta))
        {
/* If we've gotten a full row of data, try to find the row sync that */
/* precedes it.                                                      */
            row_syncd = sync_row(vbp);
            if (row_syncd == -1)
            {
/* If we couldn't find the row sync, warn the GPP and abort this frame */
                signal_frame_error(f_by_vbp(vbp).id, FRAME_ERROR_ROWFAIL);
                abort_stream(vbp, vbpnew);
            }
            else if (row_syncd == 1)
            {
/* If we found the row sync, note that we've received another line and */
/* save it into the Frame Buffer                                       */
                vbp->lines_rcvd++;

                copy_line(vbp);

/* Leave off at the end of this line, ready to look for the next row sync */
                vbp->left_off = vb_add(vbp->left_off,
                    vbp->one_line_delta - ROW_SYNC_LEN);

                if (vbp->lines_rcvd == f_by_vbp(vbp).rp->ccd.total_rows)
                {
/* If we've processed all the lines including VOCs, tell the GPP, and */
/* prepare to receive the next frame.                                 */
                    signal_frame_ended(f_by_vbp(vbp).id);
                    vbp->waiting_for_frame_sync = 1;
                }
            }
        }
    }
}

void abort_stream(vbp, vbpnew)
vidbuf_param_struct *vbp, *vbpnew;
{
    vbp->waiting_for_frame_sync = 1;
}
```

```
/* Mid-level routines to read pixel data from the Frame Buffer */

#include "BEPincl.h"

#define slice_flip(slice, num, num_cols) ((((slice) & 1) == 0) ? (num) : (((num_cols) -
(num)) - 1))
#define sdir(slice) ((((slice) & 1) == 0) ? (1) : (-1))


flag get_imp_offset(int cam, int col, int slice, int slicecol,
    region_param_struct *rp, int *offset, flag *startlo, int *step)
/* Calculate the parameters needed to read a line of image data from a */
/* slice: offset, the number of words from the beginning of the image  */
/* pixel portion of this line to the desired start pixel; startlo, a */
/* flag indicating this start pixel is a low-12-bit pixel; and step,  */
/* the number of words (including direction) to the pixel's right  */
/* neighbor.                                                          */
{
    int     num, nc, sc, co;
    flagvalid = 1;

    sc = rp->start_col[cam];
    nc = rp->ccd.num_cols;
    num = col - slicecol;
    co = col - sc;

    switch(rp->ccd.mode)
    {
        case 0: if ((cam << 1) == rp->ccd.used_cameras)
                {
                    if (rp->deal)
                    {
                        *offset = co;
                        *step = 1;
                    }
                    else
                    {
                        *offset = 4*slice_flip(slice, num, nc) + slice;
                        *step = 4*sdir(slice);
                    }
                    *startlo = 0;
                }
                else valid = 0;
                break;
        case 1: if ((slice & 1) != 0)
                {
                    if (rp->deal)
                    {
                        *offset = co;
                        *step = 1;
                    }
                    else
                    {
                        *offset = 4*((nc-num)-1) + 2*cam + (slice >> 1);
                        *step = -4;
                    }
                    *startlo = 0;
                }
                else valid = 0;
                break;
        case 2: if ((slice & 1) == 0)
                {
                    if (rp->deal)
                    {
                        *offset = co;
                        *step = 1;
                    }
                    else
                    {
                        *offset = 4*num + 2*cam + (slice >> 1);
                        *step = 4;
                    }
                    *startlo = 0;
                }
                else valid = 0;
                break;
        case 3: if (rp->deal)
                {
                    *offset = co >> 1;
                    *step = 1;
                    *startlo = co & 1;
                }
                else
                {
                    *offset = 4*slice_flip(slice, num, nc) + 2*cam +
                        (slice >> 1);
                    *step = 8*sdir(slice);
                    *startlo = slice & 1;
                }
    }
    return valid;
}

flag read_imp(int cam, int scol, int ecol, int srow, int erow, int *pixs,
    frame_specs_struct *fs)
/* Read a box of image pixels from the frame in fs */
{
    int     start, dist, nc, pixsnc;
    int     i, istart, iend, slice, j;
    int     offset, step;
    flagstartlo;
    int     *work_pixs;
    fb_ptr bls, ls;
    void(*readfunc)(fb_ptr, flag, int, int *, int);
    flagvalid = 1;

/* Determine which slices contribute to this box, noting the beginning */
/* column of the first and last contributing slice      */
    pixsnc = ecol - scol;
    nc = fs->rp->ccd.num_cols;
    if (scol < nc)
    {
/* If scol is in the first slice, but we're in Mode 1, that's Slice B */
        slice = SLICE_A0 + (fs->rp->ccd.mode == 1);
        istart = 0;
        if (ecol < nc) iend = 0;
        else if (ecol < 2*nc) iend = nc;
        else if (ecol < 3*nc) iend = 2*nc;
        else iend = 3*nc;
    }
    else if (scol < 2*nc)
    {
/* If scol is in the second slice, but we're in Mode 1, that's Slice D; */
/* and if we're in Mode 2, that's Slice C           */
        slice = SLICE_B0 + (fs->rp->ccd.mode == 2) + 2*(fs->rp->ccd.mode == 1);
        istart = nc;
        if (ecol < 2*nc) iend = nc;
        else if (ecol < 3*nc) iend = 2*nc;
        else iend = 3*nc;
    }
    else if (scol < 3*nc)
    {
        slice = SLICE_C0;
```

```
        istart = 2*nc;
        if (ecol < 3*nc) iend = 2*nc;
        else iend = 3*nc;
    }
    else
    {
        slice = SLICE_D0;
        istart = 3*nc;
        iend = 3*nc;
    }

/* If this is a deal-on-write frame, the image portion of the first  */
/* line of the box is simply the bottom of the frame, plus the offset */
/* to get to this camera's IMP area, plus linelen times the number of */
/* lines we need to skip to get to this first line.  For a deal-on-read */
/* frame, the image portion of the first line of the box is the bottom */
/* of the frame, plus linelen times the number of lines we need to skip */
/* to get to this first line, and then plus some more to skip the   */
/* packages-of-four containing this line's horizontal underclocks.  */
    if (fs->rp->deal) bls = fs->frame_start + fs->rp->imp_start_offset[cam] +
        fs->rp->linelen[cam] * (srow - fs->rp->start_row[cam]);
    else bls = fs->frame_start + fs->rp->linelen[0] *
        (srow-fs->rp->start_row[0]) + 4*fs->rp->ccd.num_hucs;

/* If we're in Mode 3, use read_fb12 to read; otherwise use read_fb16 */
    if (fs->rp->ccd.mode == 3) readfunc = (void *)(read_fb12);
    else readfunc = (void *)(read_fb16);

/* Loop through each contributing slice, counting by starting column */
    for (i = istart; i <= iend; i += nc)
    {
/* Get the read parameters for the first box line from this slice  */
        start = max(i, scol);
        dist = min(i+nc, ecol) - start;
        valid &= get_imp_offset(cam, start, slice, i, fs->rp,
            &offset, &startlo, &step);
        work_pixs = pixs;
        ls = bls;
        if (dist > 0)
        {
/* For each line in the box, get the pixels that this slice contri-  */
/* butes, and read them into the appropriate spot in the return array. */
            for (j = srow; j < erow; j++)
            {
                (*readfunc)(ls + offset, startlo, step, work_pixs, dist);
                ls += fs->rp->linelen[cam];
                work_pixs += pixsnc;
            }
        }
        pixs += dist;
        slice++;
    }
    return valid;
}

flag read_nip(int cam, int slice, int niptype, int scol, int ecol,
    int srow, int erow, int *pixs, frame_specs_struct *fs)
/* Read a box of non-image pixels from a given slice's HUC, HOC, or */
/* VOC region                                                       */
{
    fb_ptr ls = 0;
    int     offset = 0, hn = 0, step = 8;
    flagstartlo = 0;
    int     i, pixsnc;
    void(*readfunc)(fb_ptr, flag, int, int *, int) = (void *)(read_fb12);
    flagvalid = 1;

/* Calculate the line-start address given the type of NIP requested */
    hn = (fs->rp->ccd.num_hucs + fs->rp->ccd.num_hocs);
    switch(niptype)
    {
        case HUC: if (fs->rp->deal)
                  {
                      ls = fs->frame_start + fs->rp->nip_start_offset[cam] +
                          4 * hn * (srow - fs->rp->start_row[cam]);
                      offset = 0;
                  }
                  else
                  {
                      ls = fs->frame_start + fs->rp->linelen[0] *
                          (srow - fs->rp->start_row[0]);
                      offset = 0;
                  }
                  break;
        case HOC: if (fs->rp->deal)
                  {
                      ls = fs->frame_start + fs->rp->nip_start_offset[cam] +
                          4 * hn * (srow - fs->rp->start_row[cam]);
                      offset = fs->rp->ccd.num_hucs;
                  }
                  else
                  {
                      ls = fs->frame_start + fs->rp->linelen[0] *
                          (srow - fs->rp->start_row[0]);
                      offset = 4 *
                          (fs->rp->ccd.num_hucs + fs->rp->ccd.num_cols);
                  }
                  break;
        case VOC: if (fs->rp->deal)
                  {
                      ls = fs->frame_start + fs->rp->voc_start_offset[cam] +
                          4 * fs->rp->ccd.total_cols * srow;
                      offset = 0;
                  }
                  else
                  {
                      ls = fs->frame_start +
                          fs->rp->linelen[0] * (fs->rp->ccd.num_rows + srow);
                      offset = 0;
                  }
                  break;
    }

/* Calculate the parameters needed for read (see get_imp_offset above) */
    switch(fs->rp->ccd.mode)
    {
        case 0: if ((cam << 1) == fs->rp->ccd.used_cameras)
                {
                    if (fs->rp->deal)
                    {
                        offset += slice*hn + scol;
                        step = 1;
                    }
                    else
                    {
                        offset += 4*scol + slice;
                        step = 4;
                    }
                    readfunc = (void *)(read_fb16);
                }
                else valid = 0;
                break;
        case 1: if ((slice & 1) != 0)
                {
```

```
                if (fs->rp->deal)
                {
                    offset += (slice >> 1)*hn + scol;
                    step = 1;
                }
                else
                {
                    offset += 4*scol + 2*cam + (slice >> 1);
                    step = 4;
                }
                readfunc = (void *)(read_fb16);
            }
            else valid = 0;
            break;
        case 2: if ((slice & 1) == 0)
            {
                if (fs->rp->deal)
                {
                    offset += (slice >> 1)*hn + scol;
                    step = 1;
                }
                else
                {
                    offset += 4*scol + 2*cam + (slice >> 1);
                    step = 4;
                }
                readfunc = (void *)(read_fb16);
            }
            else valid = 0;
            break;
        case 3: if (fs->rp->deal)
            {
                offset += (slice*hn + scol) >> 1;
                step = 1;
                startlo = (scol & 1);
            }
            else
            {
                offset += 4*scol + 2*cam + (slice >> 1);
                startlo = (slice & 1);
            }
            break;
    }

    pixsnc = ecol - scol;

/* Use these parameters line after line to read the desired NIPs */
    if (pixsnc > 0)
    {
        for (i = srow; i <= erow; i++)
        {
            (*readfunc)(ls + offset, startlo, step, pixs, pixsnc);
            ls += fs->rp->linelen[cam];
            pixs += pixsnc;
        }
    }

    return valid;
}

void tbt(int cam, int llx, int lly, int *pixs, frame_specs_struct *fs)
/* Fast 3x3-pixel read utility, given lower-left corner of the 3x3: */
/* assumes Mode 3 deal-on-read, 3x3 lies in single slice   */
{
    int nc, ad;

    nc = fs->rp->ccd.num_cols;

/* Calculate FB address of line containing (llx,lly) */
    ad = fs->frame_start + fs->rp->linelen[0] * (lly - fs->rp->start_row[0])
        + 4*fs->rp->ccd.num_hucs + 2*cam;

/* Determine what slice the 3x3 is in, adjust ad to point to (llx,lly), */
/* and call tbt_hi for Slices A and C, or tbt_lo for Slices B and D  */
    if (llx < nc)
    {
        ad += 4*llx;
        tbt_hi(ad, fs->rp->linelen[0], pixs);
    }
    else if (llx < 2*nc)
    {
        ad += 4*(2*nc - (llx+1));
        tbt_lo(ad, fs->rp->linelen[0], pixs);
    }
    else if (llx < 3*nc)
    {
        ad += 4*(llx - 2*nc) + 1;
        tbt_hi(ad, fs->rp->linelen[0], pixs);
    }
    else
    {
        ad += 4*(4*nc - llx) - 3;
        tbt_lo(ad, fs->rp->linelen[0], pixs);
    }
}
```

```
; Various small support routines, best implemented in Assembly

    section asmsupport

    global Finit_communications
    global Fcopy_lowp_routines
    global Finit_intr_vectors
    global Fcheck_host
    global Fget_host
    global Fput_host
    global put555_host
    global Ftxd_on
    global Ftxd_off
    global Ftxd_tog
    global Fsc1_on
    global Fsc1_off

    org     p:


Finit_communications
    moveY0,y:(R6)+
    moveR1,y:(R6)+

    bset#13,x:<<m_ipr  ; Set SSI priority to 3
    bset#12,x:<<m_ipr

    clr     A
    movep A1,x:<<m_pcc  ; Force reset of Port C

; SSI Initialization
; 24 bits/word, no frame rate division
    movep #$6000,x:<<m_cra
; Normal mode, continuous clock, asynchronous operation, one-bit clock
; for both transmit and receive, MSB-first; receive clock is external.
; Enable receiver.
    movep #$2130,x:<<m_crb

    movey:-(R6),R1
    movey:-(R6),Y0
    rts


Fcopy_lowp_routines
; Copy the space P:$7040-$7AAA to P:$0040-$0AAA
    moveR1,y:(R6)+
    moveR2,y:(R6)+

    move#$7040,R1
    move#$0040,R2

    do      #($AAB-$040),_end_copy
    movem p:(R1)+,A1
    movem A1,p:(R2)+
_end_copy

    movey:-(R6),R2
    movey:-(R6),R1
    rts

Finit_intr_vectors
; Initialize port C interrupt vectors.  Most vector instructions are of
; the form "jsr <[address]"; SSI receive (video data) is
; "movep x:<<m_rx,x:(R5)+".  Constants found in FEPequ.asm.
    moveY0,y:(R6)+
    clr     A
    move#>SHORT_JSR_INSTR,Y0

    move#SSI_RX_INSTR,A1
    movem A1,p:$000C
    movem A0,p:$000D

    move#ssi_rxe_handler,A1
    or      Y0,A
    movem A1,p:$000E
    movem A0,p:$000F

    movey:-(R6),Y0
    rts


Fcheck_host
; Low-level non-blocking NeXT host read
    moveR0,y:(R6)+
    lua     (R6)+,R0
    movessh,y:(R6)+

    clr     A
    jclr#m_hrdf,x:<<m_hsr,_no_word

    moveN0,y:(R6)
    move#-3,N0
    moveR1,y:(R6)+
    movey:(R0+N0),R1
    move#>1,A1
    movep x:<<m_hrx,y:(R1)
    movey:(R6)-,N0
    movey:(R6),R1
_no_word
    move(R6)-
    movey:(R6)-,ssh
    movey:(R6),R0
    tst     A
    rts

Fget_host
; Low-level blocking NeXT host read
    jclr#m_hrdf,x:<<m_hsr,Fget_host
    clr     A
    movep x:<<m_hrx,A1
    tst     A
    rts

Fput_host
; Low-level blocking NeXT host write
    moveR0,y:(R6)+
    lua     (R6)+,R0
    movessh,y:(R6)+

    moveN0,y:(R6)+
    move#-3,N0
    moveA1,y:<y_a1_putaside_ph
    movey:(R0+N0),A
wait_for_host
    jclr#m_htde,x:<<m_hsr,wait_for_host
    movep A1,x:<<m_htx

    move(R6)-
    move  y:(R6)-,N0

    movey:(R6)-,ssh
    movey:(R6),R0
```

```
    movey:<y_a1_putaside_ph,A1

    rts

put555_host
; Quick block NeXT host write of the constant "555"
    moveA1,y:<y_a1_putaside_ph

    move#>555,A
wait_for_host_555
    jclr#m_htde,x:<<m_hsr,wait_for_host_555
    movep A1,x:<<m_htx

    movey:<y_a1_putaside_ph,A1

    rts

Ftxd_on
; Turn Port C TXD (LBox reset) on
    bset#1,x:<<m_pcd
    rts

Ftxd_off
; Turn Port C TXD (LBox reset) off
    bclr#1,x:<<m_pcd
    rts

Ftxd_tog
; Change the state of Port C TXD (LBox reset)
    bchg#1,x:<<m_pcd
    rts


Fsc1_on
; Turn Port C SC1 on
; NOTE: This is either ineffective or detrimental if SSI receive
; (Video data) is active!
    bset#4,x:<<m_pcd    ; Set SC1
    rts

Fsc1_off
; Turn Port C SC1 off
; NOTE: This is either ineffective or detrimental if SSI receive
; (Video data) is active!
    bclr#4,x:<<m_pcd    ; Clear SC1
    rts

    endsec
```

```
; CPU-RELATED EQUATES DEFINED IN machspec.asm, IN *CLN DIRECTORY

; COORDINATE WITH BEPincl.h

; GLOSSARY: p-of-4 = package-of-four words (grouping of video data)
;           vbp = vidbuf_params
;           vidbuf = Video Buffer
;           FB = Frame Buffer
;           3x3 = 3-by-3-pixel fast read

; vidbuf allocated in X: space a the end of this file
VIDBUF_BOTTOM equ  0
VIDBUF_LEN  equ     $1000
VIDBUF_TOP  equ     VIDBUF_BOTTOM+VIDBUF_LEN

SHORT_JSR_INSTR  equ  $0D0000  ; jsr <[address]
SSI_RX_INSTR  equ   $085DAF ; movep x:<<m_rx,x:(R5)+


    org y:0
; ENSURE THESE VARIABLES END UP IN SHORT-ADDRESSABLE INTERNAL Y RAM (< $40) !

y_a1_putaside_rfb  ds 1 ; For saving A1 during FB read loops
y_a1_putaside_ph  ds1   ; For saving A1 during put_host()
y_tbt_lineinc   ds  1   ; # of FB words per line of saved data for 3x3

y_vidbuf_mod    dc  (VIDBUF_LEN-1) ; Mn value to keep Rn inside vidbuf

y_hi12_mask         dc  $FFF000 ; Mask out high 12-bit pixel
y_lo12_mask         dc  $000FFF ; Mask out low 12-bit pixel
; Multiplier to put high 16 bits of 24-bit word, right-shifted by one, into
; the low 16 bits of A1 or B1
y_hi_to_lo_mul_16  dc $10000
; Multiplier to put high 12 bits of 24-bit word, right-shifted by one, into
; the low 12 bits of A1 or B1
y_hi_to_lo_mul  dc  $1000
; Multiplier to put low 12 bits of a word into the high 12 bits of A0 or B0
y_lo_to_hi_mul  dc  $800

y_read_fb_win   dc  1       ; FB read routines use FB window WX1
y_page_controls_addr dc $EFF0 ; Beginning of FB window control registers
y_x1_page_control_addr dc $EFF1 ; WX1 window control register
y_winstart          dc  $8000 ; Beginning of FB windows in X: or Y: space
y_x1_winstart   dc  $9000 ; Beginning of window WX1 in X: space
y_winptr_mask   dc  $3FFF ; Relevant bits in a FB window address

y_plus1             dc  1       ; Quick-access constants
y_plus4             dc  4
y_plus8             dc  8
y_minus3            dc  -3


; Allocate the vidbuf in X: space

    org x:VIDBUF_BOTTOM
xbuf_vidbuf dsm     VIDBUF_LEN
```

```
; Lowest-level Frame Buffer read and write routines

    section fb_io

    org p:$50

    global Fcopy_words
    global Fcopy_hi_pixels
    global Fcopy_lo_pixels
    global Fread_fb16
    global Fread_fb12
    global Ftbt_hi
    global Ftbt_lo

Fcopy_words
; Copy entire 24-bit words from the video buffer into the Frame Buffer
    moveR0,y:(R6)+
    lua     (R6)+,R0
    movessh,y:(R6)+

    clr     A                   y:<y_minus3,N0
; Save R1, (RNM)2
    moveR1,y:(R6)+
    moveM2,y:(R6)+
    moveN2,y:(R6)+
    moveR2,y:(R6)+

    move(R0)+N0
    movey:(R0)-,R2  ; 1st param: ptr into vidbuf
    movey:(R0)-,R1  ; 2nd param: ptr into framebuf (window)
    movey:(R0)-,N2  ; 3rd param: distance (+/-4) to next
                            ; pixel, or +1 for no-dealing
    movey:<y_vidbuf_mod,M2 ; Set up R2 (vidbuf ptr) circularity

    do      y:(R0),_end_copy ; last param: # of words put into FB
    movex:(R2)+N2,A1  ; A1 := word from vidbuf; R2 -> next
                              ; word to be copied from vidbuf
    moveA1,x:(R1)+      ; Write word into FB
_end_copy

; Restore R1, (RNM)2
    move(R6)-
    movey:(R6)-,R2
    movey:(R6)-,N2
    movey:(R6)-,M2
    movey:(R6)-,R1

    movey:(R6)-.ssh
    movey:(R6),R0
    rts


Fcopy_hi_pixels
; Copy a series of Mode 3 high-12-bit pixels into the Frame Buffer,
; re-compressing them by grabbing two pixels at a time and putting
; the second one in the low 12 bits under the first one.
    moveR0,y:(R6)+
    lua     (R6)+,R0
    movessh,y:(R6)+

    clr     A                   y:<y_minus3,N0
; Save X, Y, B, R1, (RNM)2
    moveB2,y:(R6)+
    moveB1,y:(R6)+
    moveB0,y:(R6)+
    moveY1,y:(R6)+
    moveY0,y:(R6)+
    moveX1,y:(R6)+
    moveX0,y:(R6)+
    moveR1,y:(R6)+
    moveM2,y:(R6)+
    moveN2,y:(R6)+
    moveR2,y:(R6)+

    clr     B                   (R0)+N0
    movey:(R0)-,R2  ; 1st param: ptr into vidbuf
    movey:(R0)-,R1  ; 2nd param: ptr into framebuf (window)
    movey:(R0)-,N2  ; 3rd param: distance (+/-4) to next pixel

    movey:<y_vidbuf_mod,M2 ; Set up R2 (vidbuf ptr) circularlity
    movey:<y_hi12_mask,Y1 ; Set Y1 to $FFF000
    movey:<y_hi_to_lo_mul,X1 ; Set X1 to the multiplier that
                ; places the "top" 12 bits of a positive multiplicand
                ; (i.e., bits 22-11, with the MSBit 23 = 0) into the
                ; bottom 12 bits of the 2nd word of the product

    movex:(R2)+N2,A1 ; A1 := first word from vidbuf; R2 -> word
                ; containing next pixel to be copied from vidbuf
    do      y:(R0),_end_copy ; last param: # of words (2 pixels
                                  ; each) to be put into FB
; Only hi pixel left in A1 (lo pixel zeroed); B1 := word from vidbuf;
; R2 -> word containing next pixel to be copied from vidbuf
    and     Y1,A            x:(R2)+N2,B1
; Shift B1 one bit to the right to force it to be a positive number
    lsr     B
; Move this positive word into X0 to be used as multiplicand
    moveB1,X0
; X0*X1 puts the hi pixel from the B1 word into the low 12 bits of the
; second word of the product; the "accumulate" into the destination A
; means the low 12 bits of A1 (previously zero) now equal the hi pixel
; of the B word, and the pixel already in the top 12 bits of A1 is
; unaffected.  B1 := word from vidbuf (temporarily); R2 -> word
; containing next pixel to be copied from vidbuf.
    mac     X0,X1,A         x:(R2)+N2,B1
; Save two-pixel word in FB; A1 := just-read word from vidbuf
    tfr     B,A                 A1,x:(R1)+
_end_copy

; Restore X, Y, B, R1, (RNM)2
    move(R6)-
    movey:(R6)-,R2
    movey:(R6)-,N2
    movey:(R6)-,M2
    movey:(R6)-,R1
    movey:(R6)-,X0
    movey:(R6)-,X1
    movey:(R6)-,Y0
    movey:(R6)-,Y1
    movey:(R6)-,B0
    movey:(R6)-,B1
    movey:(R6)-,B2

    movey:(R6)-,ssh
    movey:(R6),R0
    rts


Fcopy_lo_pixels
; Copy a series of Mode 3 low-12-bit pixels into the Frame Buffer,
; re-compressing them by grabbing two pixels at a time and putting
; the first one in the high 12 bits over the second one.
    moveR0,y:(R6)+
    lua     (R6)+,R0
```

```
    movessh,y:(R6)+

    clr     A                   y:<y_minus3,N0
; Save X, Y, B, R1, (RNM)2
    moveB2,y:(R6)+
    moveB1,y:(R6)+
    moveB0,y:(R6)+
    moveY1,y:(R6)+
    moveY0,y:(R6)+
    moveX1,y:(R6)+
    moveX0,y:(R6)+
    moveR1,y:(R6)+
    moveM2,y:(R6)+
    moveN2,y:(R6)+
    moveR2,y:(R6)+

    clr     B                   (R0)+N0
    movey:(R0)-,R2  ; 1st param: ptr into vidbuf
    movey:(R0)-,R1  ; 2nd param: ptr into framebuf (window)
    movey:(R0)-,N2  ; 3rd param: distance (+/-4) to next pixel

    movey:<y_vidbuf_mod,M2 ; Set up R2 (vidbuf ptr) circularity
    movey:<y_lo12_mask,Y1 ; Set Y1 to $000FFF
    movey:<y_lo_to_hi_mul,X1 ; Set X1 to the multiplier that
                ; places the bottom 12 bits of a positive multiplicand
                ; (i.e., bits 11-0, with the MSBit 23 = 0) into the
                ; top 12 bits of the 1st word the product

    movex:(R2)+N2,B1 ; B1 := first word from vidbuf; R2 -> next
                        ; word to be copied from vidbuf
    do      y:(R0),_end_copy ; last param: # of words (2 pixels
                               ; each) to be put into FB
; Only lo pixel left in B1 (hi pixel, including MSBit, zeroed);
; A1 := word from vidbuf; R2 -> word containing next pixel to be ; copied from vidbuf
    and     Y1,B        x:(R2)+N2,A1
; Only lo pixel left in A1 (hi pixel zeroed); move the positive word
; in B1 into X0 to be used as a multiplicand
    and     Y1,A        B1,X0
; A0 := lo pixel stored in A1 (top 12 bits zero)
    moveA1,A0
; X0*X1 puts the lo pixel from the B1 word into the hi 12 bits of the
; first word of the product; the "accumulate" into the destination A
; means the hi 12 bits of A0 (previously zero) now equal the lo pixel
; of the B word, and the pixel already in the bottom 12 bits of A0 is
; unaffected.  B1 := word from vidbuf; R2 -> word containing next
; pixel to be copied from vidbuf.
    mac     X0,X1,A     x:(R2)+N2,B1
; Save two-pixel word in FB
    moveA0,x:(R1)+
_end_copy

; Restore X, Y, B, R1, (RNM)2
    move(R6)-
    movey:(R6)-,R2
    movey:(R6)-,N2
    movey:(R6)-,M2
    movey:(R6)-,R1
    movey:(R6)-,X0
    movey:(R6)-,X1
    movey:(R6)-,Y0
    movey:(R6)-,Y1
    movey:(R6)-,B0
    movey:(R6)-,B1
    movey:(R6)-,B2

    movey:(R6)-,ssh
    movey:(R6),R0
    rts


Fread_fb16
; Read a series of 24-bit words from the Frame Buffer.
    moveR0,y:(R6)+
    lua     (R6)+,R0
    movessh,y:(R6)+

    clr     A                   y:<y_minus3,N0
; Save X0, (RN)1, R2
    moveX0,y:(R6)+
    moveN1,y:(R6)+
    moveR1,y:(R6)+
    moveR2,y:(R6)+

    move(R0)+N0
    movey:(R0)-,X0  ; 1st param: start FB address
    moveX0,y:(R6)+  ; Save on stack for setup_triple_window
    movey:<y_read_fb_win,A1  ; A1 := # for window WX1
    moveA1,y:(R6)+  ; Save on stack for setup_triple_window
    jsr     Fsetup_triple_window ; Set window WX1 to the page
                ; containing the start FB address, WX0 to the page
                ; before this, and WX2 to the page after
    movey:<y_minus3,N6 ; Preset N6 to speed up register restore
    moveA1,R1       ; R1 := start FB window address from
                            ; setup_triple_window
    move(R0)-       ; 2nd param: startlo flag (unused)
    movey:(R0)-,N1  ; 3rd param: FB address increment
    movey:(R0)-,R2  ; 4th param: address to write words

    movey:<y_hi_to_lo_mul_16,X1 ; Set X1 to the multiplier that
                ; places the "top" 16 bits of a positive multiplicand
                ; (i.e., bits 22-7, with the MSBit 23 = 0) into the
                ; bottom 16 bits of the 2nd word of the product
    movey:(R1)+N1,A1  ; A1 := first word from FB; R1 ->
                                ; next word to be read from FB

    do  y:(R0),_end_read ; last param: # of words to read
; Shift A1 one bit to the right to force it to be a positive number
    lsr     A
; Move this positive word into X0 to be used as a multiplicand
    moveA1,X0
; Right-justify the 16-bit pixel, zeroing the top 8 bits, and put the
; result in A1; X0 := word from FB (temporarily); R1 -> next word to
; be read from FB
    mpy     X0,X1,A         y:(R1)+N1,X0
; Save right-justified pixel stored in A1; R2 -> location to save
; next pixel; A1 := just-read word from FB
    tfr     X0,A            A1,y:(R2)+
_end_read

; Restore X0, (RN)1, R2
    move(R6)+N6
    movey:(R6)-,R2
    movey:(R6)-,R1
    movey:(R6)-,N1
    movey:(R6)-,X0

    movey:(R6)-,ssh
    movey:(R6),R0
    rts


Fread_fb12
; Read a series of 12-bit pixels from the Frame Buffer.
    moveR0,y:(R6)+
```

```
    lua     (R6)+,R0
    movessh,y:(R6)+

    clr     A                   y:<y_minus3,N0
; Save X, Y, B, R1, R2
    moveB2,y:(R6)+
    moveB1,y:(R6)+
    moveB0,y:(R6)+
    moveY1,y:(R6)+
    moveY0,y:(R6)+
    moveX1,y:(R6)+
    moveX0,y:(R6)+
    moveR1,y:(R6)+
    moveR2,y:(R6)+

    move(R0)+N0
    movey:(R0)-,X0  ; 1st param: start FB address
    moveX0,y:(R6)+  ; Save on stack for setup_triple_window
    movey:<y_read_fb_win,A1  ; A1 := # for window WX1
    moveA1,y:(R6)+  ; Save on stack for setup_triple_window
    jsr     Fsetup_triple_window ; Set window WX1 to the page
                ; containing the start FB address, WX0 to the page
                ; before this, and WX2 to the page after
    movey:<y_minus3,N6 ; Preset N6 to speed up register restore
    moveA1,R1       ; R1 := start FB window address from
                            ; setup_triple_window
    movey:(R0)-,A1  ; 2nd param: which pixel in the first word
                            ; to start with (0 = hi, 1 = lo)
    movey:(R0)-,Y0  ; 3rd param: pixel increment (0.5 word)
    movey:(R0)-,R2  ; 4th param: address to write pixels

    clr     B                   y:<y_lo12_mask,Y1 ; Y1 := $000FFF
    movey:<y_hi_to_lo_mul,X1 ; Set X1 to the multiplier that
                ; places the "top" 12 bits of a positive multiplicand
                ; (i.e., bits 22-11, with the MSBit 23 = 0) into the
                ; bottom 12 bits of the 2nd word of the product

    do      y:(R0),_end_read ; last param: # of pixels to read
; A1 is assumed to have the pixel offset from the start address:
; If this is an even number, read the hi pixel; if odd, the lo pixel;
; and (A1 / 2) represents the offset in words.
; Save the pixel offset. Shift A1 one bit to the right: A1 := offset
; in pixels, CARRY = flag indicating the pixel of interest
; (0 = hi, 1 = lo).
    lsr     A                   A1,y:<y_a1_putaside_rfb
; N1 = offset in words from start FB window address
    moveA1,N1
; If carry is clear, read the hi pixel...
    jcc     <_hi_pix
; ... otherwise read the lo pixel
; B1 := word containing pixel to be read
    movex:(R1+N1),B1
; Only lo pixel left in B1 (hi pixel zeroed); restore the pixel offset
; to A1
    and     Y1,B            y:<y_a1_putaside_rfb,A1
; Save this pixel
    jmp     <_save_pix
_hi_pix
; B1 := word containing pixel to be read
    movex:(R1+N1),B1
; Shift B1 one bit to the right to force it to be a positive number
    lsr     B
; Move this positive word into X0 to be used as a multiplicand
    moveB,X0
; Put the hi pixel from the word read from the FB into the low 12 bits
; of B1; restore the pixel offset to A1
    mpy     X0,X1,B         y:<y_a1_putaside_rfb,A1
_save_pix
; Save the pixel (right-justified, with the top 12 bits zero); R2 ->
; location to save next word; increment pixel offset (positively or
; negatively) to indicate next pixel to be read
    add     Y0,A            B1,y:(R2)+
_end_read

; Restore X, Y, B, R1, R2
    move(R6)+N6
    movey:(R6)-,R2
    movey:(R6)-,R1
    movey:(R6)-,X0
    movey:(R6)-,X1
    movey:(R6)-,Y0
    movey:(R6)-,Y1
    movey:(R6)-,B0
    movey:(R6)-,B1
    movey:(R6)-,B2

    movey:(R6)-,ssh
    movey:(R6),R0
    rts


Ftbt_hi
; Grab a quick 3x3 array of pixels assumed to be from slice A or C
; saved as a Mode 3 deal-on-read frame (i.e. pixels read out left-to-
; right and located in the top 12 bits of their words).
    moveR0,y:(R6)+
    lua     (R6)+,R0
    movessh,y:(R6)+

    clr     A                   y:<y_minus3,N0
; Save X, B, (RN)1, R2
    moveB2,y:(R6)+
    moveB1,y:(R6)+
    moveB0,y:(R6)+
    moveX1,y:(R6)+
    moveX0,y:(R6)+
    moveR1,y:(R6)+
    moveN1,y:(R6)+
    moveR2,y:(R6)+

    clr     B                   (R0)+N0
    movey:(R0)-,X0  ; 1st param: start FB address
    movey:(R0)-,A1  ; 2nd param: FB increment for next line
    movey:(R0),R2   ; 3rd param: Address to write pixels

    movey:<y_plus8,X1 ; X1 := 8
; We're going to read one pixel, go forward one pixel (4 words), read
; another pixel, go forward one pixel (4 words), read another pixel,
; then go up to the next line --- meaning up one full line, then back
; two pixels (8 words).
; A1 := FB increment for next line, minus 8; R1 := $EFF1
    sub     X1,A            y:<y_x1_page_control_addr,R1
; B1 := start FB address; save the line-to-line increment stored in A1
    tfr     X0,B            A1,y:<y_tbt_lineinc
; Shift B1 one bit to the right to force it to be a positive number;
; A1 := $000FFF
    lsr     B                   y:<y_lo12_mask,A1
; A1 := the low 12 bits of the start FB address, i.e., the in-window
; address; X1 := the multiplier that places the "top" 12 bits of a
; positive multiplicand (i.e., bits 22-11, with the MSBit 23 = 0) into
; the bottom 12 bits of the 2nd word of the product.
    and     X0,A            y:<y_hi_to_lo_mul,X1
; Move the positive word in B1 into X0 to be used as a multiplicand
    moveB1,X0
```

```
; Put the top 12 bits of the start FB address (i.e., the page number)
; into the bottom 12 bits of B1; X0 := 1
    mpy     X0,X1,B         y:<y_plus1,X0
; ($EFF1) = B1, i.e., window WX1 is set to the page including the
; start FB address; R1 -> $EFF2 (control register for window WX2);
; B1 := the page following the one including the start FB address
    add     X0,B            B1,y:(R1)+
; X0 := $9000, the beginning of window WX1
    movey:<y_x1_winstart,X0
; A1 := start FB window address ($9000 + in-window address);
; ($EFF2) = B1, i.e., window WX2 is set to the page following the one
; including the start FB address
    or      X0,A            B1,y:(R1)
; R1 := start FB window address
    moveA1,R1
; N1 := 4, the number of words between horizontally adjacent pixels
; in a deal-on-read frame
    movey:<y_plus4,N1

; A1 := word containing first pixel, first line; R1 -> word containing
; second pixel, first line
    movex:(R1)+N1,A1
; Shift A1 one bit to the right to force it to be a positive number;
; B1 := word containing second pixel, first line; R1 -> word
; containing third pixel, first line
    lsr     A                   x:(R1)+N1,B1
; Shift B1 one bit to the right to force it to be a positive number;
; Move the positive word in A1 into X0 to be used as a multiplicand
    lsr     B                   A1,X0
; N1 := the number of words from the end of one line of the 3x3 to
; the beginning of the next
    movey:<y_tbt_lineinc,N1
; Right-justify the first pixel, first line, zeroing the top 12 bits,
; and store it in A1; move the positive word in B1 into X0 to be
; used as a multiplicand
    mpy     X0,X1,A         B1,X0
; Right-justify the second pixel, first line, zeroing the top 12 bits,
; and store it in B1; save the first pixel, first line; R2 -> location
; to save second pixel, first line
    mpy     X0,X1,B         A1,y:(R2)+
; A1 := third pixel, first line; R1 -> first pixel, second line
    movex:(R1)+N1,A1
; Shift A1 one bit to the right to force it to be a positive number;
; N1 := 4, the number of words between horizontally adjacent pixels
    lsr     A                   y:<y_plus4,N1
; Move the positive word in A1 into X0 to be used as a multiplicand
    moveA1,X0
; Right-justify the third pixel, first line, zeroing the top 12 bits,
; and store it in A1; save the second pixel, first line; R2 ->
; location to save third pixel, first line
    mpy     X0,X1,A         B1,y:(R2)+
; Save the third pixel, first line; R2 -> location to save first
; pixel, second line
    moveA1,y:(R2)+

; A1 := word containing first pixel, second line, R1 -> word
; containing second pixel, second line
    movex:(R1)+N1,A1
; Shift A1 one bit to the right to force it to be a positive number;
; B1 := word containing second pixel, second line; R1 -> word
; containing third pixel, second line
    lsr     A                   x:(R1)+N1,B1
; Shift B1 one bit to the right to force it to be a positive number;
; Move the positive word in A1 into X0 to be used as a multiplicand
    lsr     B                   A1,X0
; N1 := the number of words from the end of one line of the 3x3 to
; the beginning of the next
    movey:<y_tbt_lineinc,N1
; Right-justify the first pixel, second line, zeroing the top 12 bits,
; and store it in A1; move the positive word in B1 into X0 to be
; used as a multiplicand
    mpy     X0,X1,A         B1,X0
; Right-justify the second pixel, second line, zeroing the top 12
; bits, and store it in B1; save the first pixel, second line;
; R2 -> location to save second pixel, second line
    mpy     X0,X1,B         A1,y:(R2)+
; A1 := third pixel, second line; R1 -> first pixel, third line
    movex:(R1)+N1,A1
; Shift A1 one bit to the right to force it to be a positive number;
; N1 := 4, the number of words between horizontally adjacent pixels
    lsr     A                   y:<y_plus4,N1
; Move the positive word in A1 into X0 to be used as a multiplicand
    moveA1,X0
; Right justify the third pixel, second line, zeroing the top 12 bits,
; and store it in A1; save the second pixel, second line; R2 ->
; location to save third pixel, second line
    mpy     X0,X1,A         B1,y:(R2)+
; Save the third pixel, second line; R2 -> location to save first
; pixel third line
    moveA1,y:(R2)+

; A1 := word containing first pixel, third line; R1 -> word containing
; second pixel, third line
    movex:(R1)+N1,A1
; Shift A1 one bit to the right to force it to be a positive number;
; B1 := word containing second pixel, third line; R1 -> word
; containing third pixel, third line
    lsr     A                   x:(R1)+N1,B1
; Shift B1 one bits to the right to force it to be a positive number;
; Move the positive word in A1 into X0 to be used as a multiplicand
    lsr     B                   A1,X0
; Right-justify the first pixel, third line, zeroing the top 12 bits,
; and store it in A1; move the positive word in B1 into X0 to be
; used as a multiplicand
    mpy     X0,X1,A         B1,X0
; Right-justify the second pixel, third line, zeroing the top 12 bits,
; and store it in B1; save the first pixel, third line; R2 -> location
; to save second pixel, third line
    mpy     X0,X1,B         A1,y:(R2)+
; A1 := third pixel, third line
    movex:(R1),A1
; Shift A1 one bit to the right to force it to be a positive number
    lsr     A
; Move the positive word in A1 into X0 to be used as a multiplicand
    moveA1,X0
; Right-justify the third pixel, third line, zeroing the top 12 bits,
; and store it in A1; save the second pixel, third line; R2 ->
; location to save third pixel, third line
    mpy     X0,X1,A         B1,y:(R2)+
; Save the third pixel, third line
    moveA1,y:(R2)

; Restore X, B, (RN)1, R2
    move(R6)-
    movey:(R6)-,R2
    movey:(R6)-,N1
    movey:(R6)-,R1
    movey:(R6)-,X0
    movey:(R6)-,X1
    movey:(R6)-,B0
    movey:(R6)-,B1
    movey:(R6)-,B2

    movey:(R6)-,ssh
```

```
    movey:(R6),R0
    rts

Ftbt_lo
; Grab a quick 3x3 array of pixels assumed to be from slice B or D
; saved as a Mode 3 deal-on-read frame (i.e. pixels read out right-to-
; left and located in the bottom 12 bits of their words).
    moveR0,y:(R6)+
    lua     (R6)+,R0
    movessh,y:(R6)+

    clr     A                   y:<y_minus3,N0
; Save X, B, (RN)1, R2
    moveB2,y:(R6)+
    moveB1,y:(R6)+
    moveB0,y:(R6)+
    moveX1,y:(R6)+
    moveX0,y:(R6)+
    moveR1,y:(R6)+
    moveN1,y:(R6)+
    moveR2,y:(R6)+

    clr     B                   (R0)+N0
    movey:(R0)-,X0  ; 1st param: start FB address
    movey:(R0)-,A1  ; 2nd param: FB increment for next line
    movey:(R0),R2   ; 3rd param: Address to write pixels

    movey:<y_plus8,X1 ; X1 := 8
; We're going to read one pixel, go forward one pixel (backward 4
; words), read another pixel, go forward one pixel (backward 4 words),
; read another pixel, then go up to the next line --- meaning up one
; full line, then back two pixels (forward 8 words).
; A1 := FB increment for next line, plus 8; R1 := $EFF1
    add     X1,A            y:<y_page_controls_addr,R1
; B1 := start FB address; save the line-to-line increment stored in A1
    tfr     X0,B            A1,y:<y_tbt_lineinc
; Shift B1 one bit to the right to force it to be a positive number;
; A1 := $000FFF
    lsr     B                   y:<y_lo12_mask,A1
; A1 := the low 12 bits of the start FB address, i.e., the in-window
; address; X1 := the multiplier that places the "top" 12 bits of a
; positive multiplicand (i.e., bits 22-11, with the MSBit 23 = 0) into
; the bottom 12 bits of the 2nd word of the product.
    and     X0,A            y:<y_hi_to_lo_mul,X1
; Move the positive word in B1 into X0 to be used as a multiplicand
    moveB1,X0
; Put the top 12 bits of the start FB address (i.e., the page number)
; into the bottom 12 bits of B1; X0 := 1
    mpy     X0,X1,B         y:<y_plus1,X0
; B1 := the page preceding the one including the start FB address;
; X0 := $9000, the beginning of window WX1
    sub     X1,B            y:<y_x1_winstart,X0
; ($EFF0) = B1, i.e., the window WX0 is set to the page preceding the
; start FB address; R1 -> $EFF1 (control register for window WX1);
; B1 := the page including the start FB address
    add     X1,B            B1,y:(R1)+
; ($EFF1) = B1, i.e., the window WX1 is set to the page including the
; start FB address; R1 -> $EFF2 (control register for window WX2);
; B1 := the page following the start FB address
    add     X1,B            B1,y:(R1)+
; A1 := start FB window address ($9000 + in-window address);
; ($EFF2) = B1, i.e., the window WX2 is set to the page following the
; start FB address
    or      X0,A            B1,y:(R1)
; R1 := start FB window address
    moveA1,R1
; N1 := 4, the number of words between horizontally adjacent pixels
; in a deal-on-read frame
    movey:<y_plus4,N1
; X0 := $000FFF
    movey:<y_lo12_mask,X0

; A1 := word containing 1st pixel, 1st line; R1 -> word containing
; 2nd pixel, 1st line
    movex:(R1)-N1,A1
; Zero the top 12 bits, leaving only the 1st pixel 1st line in A1;
; B1 := word containing 2nd pixel, 1st line; R1 -> word containing
; 3rd pixel, 1st line
    and     X0,A            x:(R1)-N1,B1
; N1 := the number of words from the end of one line of the 3x3 to
; the beginning of the next
    movey:<y_tbt_lineinc,N1
; Zero the top 12 bits, leaving only the 2nd pixel 1st line in B1;
; Save 1st pixel 1st line; R2 -> location to save 2nd pixel 1st line
    and     X0,B            A1,y:(R2)+
; A1 := 3rd pixel 1st line; R1 -> 1st pixel 2nd line
    movex:(R1)+N1,A1
; N1 := 4, the number of words between horizontally adjacent pixels
    movey:<y_plus4,N1
; Zero the top 12 bits, leaving only the 3rd pixel 1st line in A1;
; Save 2nd pixel 1st line; R2 -> location to save 3rd pixel 1st line
    and     X0,A            B1,y:(R2)+
; Save 3rd pixel 1st line; R2 -> location to save 1st pixel 2nd line
    moveA1,y:(R2)+

; A1 := word containing 1st pixel, 2nd line; R1 -> word containing
; 2nd pixel, 2nd line
    movex:(R1)-N1,A1
; Zero the top 12 bits, leaving only the 1st pixel 1st line in A1;
; B1 := word containing 2nd pixel, 2nd line; R1 -> word containing
; 3rd pixel, 2nd line
    and     X0,A            x:(R1)-N1,B1
; N1 := the number of words from the end of one line of the 3x3 to
; the beginning of the next
    movey:<y_tbt_lineinc,N1
; Zero the top 12 bits, leaving only the 2nd pixel 3nd line in B1;
; Save 1st pixel 2nd line; R2 -> location to save 2nd pixel 2nd line
    and     X0,B            A1,y:(R2)+
; A1 := 3rd pixel 2nd line; R1 -> 1st pixel 3rd line
    movex:(R1)+N1,A1
; N1 := 4, the number of words between horizontally adjacent pixels
    movey:<y_plus4,N1
; Zero the top 12 bits, leaving only the 3rd pixel 2nd line in A1;
; Save 2nd pixel 2nd line; R2 -> location to save 3rd pixel 2nd line
    and     X0,A            B1,y:(R2)+
; Save 3rd pixel 2nd line; R2 -> location to save 1st pixel 3rd line
    moveA1,y:(R2)+

; A1 := word containing 1st pixel, 3rd line; R1 -> word containing
; 2nd pixel, 3rd line
    movex:(R1)-N1,A1
; Zero the top 12 bits, leaving only the 1st pixel 3rd line in A1;
; B1 := word containing 2nd pixel, 3rd line; R1 -> word containing
; 3rd pixel, 3rd line
    and     X0,A            x:(R1)-N1,B1
; Zero the top 12 bits, leaving only the 2nd pixel 3rd line in B1;
; Save 1st pixel 3rd line; R2 -> location to save 2nd pixel 3rd line
    and     X0,B            A1,y:(R2)+
; A1 := 3rd pixel 3rd line
    movex:(R1),A1
; Zero the top 12 bits, leaving only the 3rd pixel 3rd line in A1;
; Save 2nd pixel 3rd line; R2 -> location to save 3rd pixel 3rd line
    and     X0,A            B1,y:(R2)+
; Save 3rd pixel 3rd line
```

```
    moveA1,y:(R2)

; Restore X, B, (RN)1, R2
    move(R6)-
    movey:(R6)-,R2
    movey:(R6)-,N1
    movey:(R6)-,R1
    movey:(R6)-,X0
    movey:(R6)-,X1
    movey:(R6)-,B0
    movey:(R6)-,B1
    movey:(R6)-,B2

    movey:(R6)-,ssh
    movey:(R6),R0
    rts

    endsec
```

```
; Port C interrupt handlers

    section interrupts

    global ssi_rxe_handler

    org p:$40


ssi_rxe_handler  ; 3 words
    movep x:<<m_sr,y:Fssi_rx_exception ; Record what the problem is
    movep x:<<m_rx,x:(R5)+  ; Read the word as though normal
    rti


    endsec
```

```
    section window

    global Fsetup_window
    global Fsetup_dual_window
    global Fsetup_triple_window

Fsetup_window
; Make a given window point to the Frame Buffer page which includes
; the desired Frame Buffer address; return the Frame Buffer window
; address which then corresponds to the desired FB address.
    moveR0,y:(R6)+
    lua     (R6)+,R0
    movessh,y:(R6)+

; Save X, B, (RN)1
    moveB2,y:(R6)+
    moveB1,y:(R6)+
    moveB0,y:(R6)+
    moveX1,y:(R6)+
    moveX0,y:(R6)+
    moveR1,y:(R6)+
    moveN1,y:(R6)+

    clr     A                   #-3,N0
    clr     B                   (R0)+N0
    movey:(R0)-,N1  ; 1st param: Window number
    movey:(R0),X0   ; 2nd param: FB address

; B1 := FB address; R1 := $EFF0 (beginning of window control
; register area)
    tfr     X0,B            y:<y_page_controls_addr,R1
; Shift B1 one bit to the right to force it to be a positive number;
; A1 := $000FFF
    lsr     B                   y:<y_lo12_mask,A1
; A1 := the low 12 bits of the FB address, i.e., the in-window
; address; X1 := the multiplier that places the "top" 12 bits of a
; positive multiplicand (i.e., bits 22-11, with the MSBit 23 = 0) into
; the bottom 12 bits of the 2nd word of the product
    and     X0,A            y:<y_hi_to_lo_mul,X1
; Move the positive word in B1 into X0 to be used as a multiplicand
    moveB1,X0
; Put the top 12 bits of the start FB address (i.e., the page number)
; into the bottom 12 bits of B1; X1 := the multiplier that places the
; bottom 12 bits of a positive multiplicand (i.e., bits 11-0, with the
; MSBit 23 = 0) into the top 12 bits of the 1st word of the product
    mpy     X0,X1,B         y:<y_lo_to_hi_mul,X1
; X0 := window number (positive number, 0-7)
    moveN1,X0
; A0 := in-window address stored in A1
    moveA1,A0
; Put the window number in the top 12 bits above the in-window
; address.  A0 now reads 0000:0000:0www:aaaa:aaaa:aaaa, where "w"
; is the window number, and "a" the in-window address.
; X1 := $003FFF.
    mac     X0,X1,A         y:<y_winptr_mask,X1
; Move the window number / in-window address stored in A0 into A1
    moveA0,A1
; Zero the top "w" bit in A1 (the bit that indicates whether this
; is an X or a Y window); X0 := $8000, the start of the windows
; in X and Y space
    and     X1,A            y:<y_winstart,X0
; Add X0 to the window number / in-window address combination stored
; in A1.  A1 now holds the Frame Buffer window address corresponding
; to the desired FB address.  Set the window's page by storing the
; page number (in B1) into the register controlling this window.
    add     X0,A            B1,y:(R1+N1)

; Restore X, B, (RN)1
    move(R6)-
    movey:(R6)-,N1
    movey:(R6)-,R1
    movey:(R6)-,X0
    movey:(R6)-,X1
    movey:(R6)-,B0
    movey:(R6)-,B1
    movey:(R6)-,B2

    movey:(R6)-,ssh
    movey:(R6),R0
    rts


Fsetup_dual_window
; Make a given window point to the Frame Buffer page which includes
; the desired Frame Buffer address, and make the next window point
; to the following page; return the Frame Buffer window address which
; then corresponds to the desired FB address.
    moveR0,y:(R6)+
    lua     (R6)+,R0
    movessh,y:(R6)+

; Save X, B, (RN)1
    moveB2,y:(R6)+
    moveB1,y:(R6)+
    moveB0,y:(R6)+
    moveX1,y:(R6)+
    moveX0,y:(R6)+
    moveR1,y:(R6)+
    moveN1,y:(R6)+

    clr     A                   #-3,N0
    clr     B                   (R0)+N0
    movey:(R0)-,N1  ; 1st param: Window number
    movey:(R0),X0   ; 2nd param: FB address

; B1 := FB address; R1 := $EFF0 (beginning of window control
; register area)
    tfr     X0,B            y:<y_page_controls_addr,R1
; Shift B1 one bit to the right to force it to be a positive number;
; A1 := $000FFF
    lsr     B                   y:<y_lo12_mask,A1
; A1 := the low 12 bits of the FB address, i.e., the in-window
; address; X1 := the multiplier that places the "top" 12 bits of a
; positive multiplicand (i.e., bits 22-11, with the MSBit 23 = 0) into
; the bottom 12 bits of the 2nd word of the product
    and     X0,A            y:<y_hi_to_lo_mul,X1
; Move the positive word in B1 into X0 to be used as a multiplicand
    moveB1,X0
; Put the top 12 bits of the start FB address (i.e., the page number)
; into the bottom 12 bits of B1; X1 := the multiplier that places the
; bottom 12 bits of a positive multiplicand (i.e., bits 11-0, with the
; MSBit 23 = 0) into the top 12 bits of the 1st word of the product
    mpy     X0,X1,B         y:<y_lo_to_hi_mul,X1
; X0 := window number (positive number, 0-7)
    moveN1,X0
; A0 := in-window address stored in A1
    moveA1,A0
; Put the window number in the top 12 bits above the in-window
; address.  A0 now reads 0000:0000:0www:aaaa:aaaa:aaaa, where "w"
; is the window number, and "a" the in-window address.
; X1 := $003FFF.
    mac     X0,X1,A         y:<y_winptr_mask,X1
; Move the window number / in-window address stored in A0 into A1
```

```
    moveA0,A1
; Zero the top "w" bit in A1 (the bit that indicates whether this
; is an X or a Y window); X0 := $8000, the start of the windows
; in X and Y space
    and     X1,A            y:<y_winstart,X0
; Add X0 to the window number / in-window address combination stored
; in A1.  A1 now holds the Frame Buffer window address corresponding
; to the desired FB address.  Set the window's page by storing the
; page number (in B1) into the register controlling this window.
    add     X0,A            B1,y:(R1+N1)
; X0 := 1
    movey:<y_plus1,X0
; B1 := the page following the page containing the desired FB address;
; R1 -> $EFF1, the beginning of the window control register area
; plus one
    add     X0,B            (R1)+
; Set the following window's page by storing the next page number
; (in B1) into the register controlling this window
    moveB1,y:(R1+N1)

; Restore X, B, (RN)1
    move(R6)-
    movey:(R6)-,N1
    movey:(R6)-,R1
    movey:(R6)-,X0
    movey:(R6)-,X1
    movey:(R6)-,B0
    movey:(R6)-,B1
    movey:(R6)-,B2

    movey:(R6)-,ssh
    movey:(R6),R0
    rts


Fsetup_triple_window
; Make a given window point to the Frame Buffer page which includes
; the desired Frame Buffer address, and the windows on either side
; of it point to the pages on either side; return the Frame Buffer
; window address which then corresponds to the desired FB address.
    moveR0,y:(R6)+
    lua     (R6)+,R0
    movessh,y:(R6)+

; Save X, B, (RN)1
    moveB2,y:(R6)+
    moveB1,y:(R6)+
    moveB0,y:(R6)+
    moveX1,y:(R6)+
    moveX0,y:(R6)+
    moveR1,y:(R6)+
    moveN1,y:(R6)+

    clr     A                   #-3,N0
    clr     B                   (R0)+N0
    movey:(R0)-,N1  ; 1st param: Window number
    movey:(R0),X0   ; 2nd param: FB address

; B1 := FB address; R1 := $EFF0 (beginning of window control
; register area)
    tfr     X0,B            y:<y_page_controls_addr,R1
; Shift B1 one bit to the right to force it to be a positive number;
; A1 := $000FFF
    lsr     B                   y:<y_lo12_mask,A1
; A1 := the low 12 bits of the FB address, i.e., the in-window
; address; X1 := the multiplier that places the "top" 12 bits of a
; positive multiplicand (i.e., bits 22-11, with the MSBit 23 = 0) into
; the bottom 12 bits of the 2nd word of the product
    and     X0,A            y:<y_hi_to_lo_mul,X1
; Move the positive word in B1 into X0 to be used as a multiplicand
    moveB1,X0
; Put the top 12 bits of the start FB address (i.e., the page number)
; into the bottom 12 bits of B1; X1 := the multiplier that places the
; bottom 12 bits of a positive multiplicand (i.e., bits 11-0, with the
; MSBit 23 - 0) into the top 12 bits of the 1st word of the product
    mpy     X0,X1,B         y:<y_lo_to_hi_mul,X1
; X0 := window number (positive number, 0-7)
    moveN1,X0
; A0 := in-window address stored in A1
    moveA1,A0
; Put the window number in the top 12 bits above the in-window
; address.  A0 now reads 0000:0000:0www:aaaa:aaaa:aaaa, where "w"
; is the window number, and "a" the in-window address.
; X1 := $003FFF.
    mac     X0,X1,A         y:<y_winptr_mask,X1
; Move the window number / in-window address stored in A0 into A1
    moveA0,A1
; Zero the top "w" bit in A1 (the bit that indicates whether this
; is an X or a Y window); X0 := $8000, the start of the windows
; in X and Y space
    and     X1,A            y:<y_winstart,X0
; Add X0 to the window number / in-window address combination stored
; in A1.  A1 now holds the Frame Buffer window address corresponding
; to the desired FB address.  Set the window's page by storing the
; page number (in B1) into the register controlling this window.
    add     X0,A            B1,y:(R1+N1)
; X0 := 1
    movey:<y_plus1,X0
; B1 := the page preceding the page containing the desired FB address;
; R1 -> $EFEF, the beginning of the window control register area
; minus one
    sub     X0,B            (R1)-
; Set the preceding window's page by storing the preceding page number
; (in B1) into the register controlling this window
    moveB1,y:(R1+N1)
; B1 := the page following the page containing the desired FB address;
; R1 -> $EFF1, the beginning of the window control register area
; plus one
    add     X0,B            (R1)+
    add     X0,B            (R1)+
; Set the following window's page by storing the next page number
; (in B1) into the register controlling this window
    moveB1,y:(R1+N1)

; Restore X, B, (RN)1
    move(R6)-
    movey:(R6)-,N1
    movey:(R6)-,R1
    movey:(R6)-,X0
    movey:(R6)-,X1
    movey:(R6)-,B0
    movey:(R6)-,B1
    movey:(R6)-,B2

    movey:(R6)-,ssh
    movey:(R6),R0
    rts

    endsec
```

```
      opt so,xr
      page 132,66,3,3

;CRTRZP.ASM i920622 jek
;  RZP CRT0 replacement source -- Configuration for RZP DSP
;930615 org adjusted per size of ROM-booted area.
; Modified 1/94 by S.A.McD. to potentially start stack at specific point

; From: $Id: crt056.asm,v 1.12 91/06/14 14:30:36 jeff Exp $
; Modifications to this file Copyright (C) 1992,1993 AeroAstro, Inc.

; Memory model based on a jammed -define for the symbol "mm" at command line.
; Example:
; asm56000 -bcrtRZPy.cln -lcrtRZPy.lst   -dmm y   crtRZP
;          dest=y cln^   y list file^  define^   ^same source, every model
; See MAKERZP.BAT

      include "ioequ.asm"
;GNU global declarations (not currently global)
;  global FRTC_Count
;  global FRF_Count
;  global FGPS_Count
;  global FDGlue_Stat
;  global FDGlue_Ctrl
; for these equ's
      include "dglue.a"

;----------------------------------------
; Externs (init driver calls ONLY)
; (GNU/asm56k do not require explicit
; EXTERN/EXTRN-type declaration, globals
; are all that need be declared, EXTRN is
; automatic)
;----------------------------------------
; Finitialize_dma_link
; Finit_RTC_driver
;----------------------------------------


;==========================================
TOP_OF_MEMORY  equ     $7FFF   ;last RAM alloc'able word in Y:/X: data RAM

STACK_START    equ     $1000 ; not currently used

; This value represents the 32k X & Y: internal plus external SRAM on the RZP
; P top of SRAM memory is also 32k.
;  (all addresses below are in hex)

;DSP Internal Program SRAM @ P:0000 to P:01FF  24 bits
;    External Program SRAM @ P:0200 to P:7FFF  24 bits
;   (w)External Boot EEROM @ P:8000 to P:FFFF   8 bits, WAITs, EEWE protected
;   (r)External Boot EEROM @ P:C000 to P:DFFF   8 bits, WAITs
; DSP Internal Y:Data SRAM @ Y:0000 to Y:00FF  24 bits
;     External Y:Data SRAM @ Y:0100 to Y:7FFF  24 bits
;          Frame buffer Y8 @ Y:8000 to Y:8FFF  24 bits
;          Frame buffer Y9 @ Y:8000 to Y:9FFF  24 bits
;          Frame buffer YA @ Y:8000 to Y:AFFF  24 bits
;          Frame buffer YB @ Y:8000 to Y:BFFF  24 bits
;                No memory @ Y:C000 to Y:EEFF
;   Frame buffer page regs @ Y:EF00 to Y:EF07  12 bits
;                No memory @ Y:EF08 to Y:FFBF
;        DGLUE Peripherals @ Y:FFC0 to Y:FFC3  varies (Fxx0-Fxx3)
;                No memory @ Y:FFC4 to Y:FFFF
; DSP Internal X:Data SRAM @ X:0000 to X:00FF  24 bits
;     External X:Data SRAM @ X:0100 to X:7FFF  24 bits
;          Frame buffer X8 @ X:8000 to X:8FFF  24 bits
;          Frame buffer X9 @ X:8000 to X:9FFF  24 bits
;          Frame buffer XA @ X:8000 to X:AFFF  24 bits
;          Frame buffer XB @ X:8000 to X:BFFF  24 bits
;                No memory @ X:C000 to X:FFBF
; DSP Internal Peripherals @ X:FFC0 to X:FFFF  varies

;----------------------------------------------
; Block access to area used by bootstrap ROM code
;----------------------------------------------

      section bootstrapper       ;Block GNU access
      org     p:$0               ; to the ROM booted area
      ds      2731               ; block out use of first 8k *BYTES*
      endsec                     ; (8192/3)=2730.7 words, last word is partial
                                 ; and is not stuffed by rom burner

CRT0ORG  equ   $0AAB             ;CRT0 + compiled code starts after
                                 ;ROM-booted code, which fills 0..AAA
                                 ;See ROMBOOT.ASM
;;#### HETE?
;;      section BlockXInternal  ;block internal DSP RAM in X:
;;      org     x:$0            ; to force GNU to use the more
;;      ds      $100            ; 'robust' external RAM on the
;;      endsec                  ; RZP.
;;      section BlockYInternal  ;same note - BOTH must be blocked,
;;      org     y:$0            ;regardles of "mm" model chosen
;;      ds      $100
;;      endsec


;==========================================
;  System variable area (GNU)
;==========================================

      section crt0

      org     mm:              ; per GNU model

; The following variables are used for dynamic memory allocation
; __stack_safty: Since dynamic memory and the stack grow towards each other
; This constant tells brk and sbrk what the minimum amount of space should
; be left between the top of stack during the brk or sbrk call and the end
; of any allocated memory.
; __mem_limit: a constant telling brk and sbrk where the end of available
; memory is.
; __break: pointer to the next block of memory that can be allocated
; The heap may be moved by changing the initial value of __break.
; This is the base of the heap.
; __y_size: the base of dynamic memory.
; errno: error type: set by some libraries
; __max_signal the last possible signal (interrupt #).


                global Ferrno
Ferrno          ds 1       ;init=0
                global F__stack_safety
F__stack_safety ds 1       ;init=1024      ;512 on the hack board
                global F__mem_limit
F__mem_limit    ds 1       ;init=TOP_OF_MEMORY
                global F__break
F__break        ds 1       ;init=TOP_OF_MEMORY
                global F__y_size
F__y_size       ds 1       ;init=DSIZE normally, modifiable
                global F__max_signal
F__max_signal   ds 1       ;init=$3e       ; what the quint monitor.


;==========================================
```

```
;       GNU startup
;===========================================

        org     p:CRT0ORG

        global  F__start
F__start

; wait state programming is done twice in case app is
; executed initially from here.

        move    #0,SP                           ; Clear hardware stack pointer
        movep   #0,X:<<$FFFE                    ; clear auto waits (BCR)
        ori     #$80,omr                        ; activate DSP wait state recog.
        movep   #WAITENA_IRQFST,Y:<<FDGlue_Ctrl   ; and in DGLUE

;-------------------------------------------------
; GNU pointer initialization code
;-------------------------------------------------

        clr     a
        move    a,mm:Ferrno
        move    #TOP_OF_MEMORY,a
        move    a,mm:F__mem_limit
        move    a,mm:F__break
;        move    #STACK_START,a
        move    #DSIZE,a
        move    a,mm:F__y_size
        move    #>$3E,a
        move    a,mm:F__max_signal
        move    #1024,A
        move    a,mm:F__stack_safety

;       To change the base of the stack, change the
;       value loaded into the stack pointer here.

        move    mm:F__y_size,r6         ; initialize the stack pointer
        move    #$0,r0                  ; funny value to terminate a backtrace.

        and     #$F0,mr                 ; clear scaling mode bits i1,i0
                                        ; and set current IPL = 0 (AFTER setting R6)
        and     #$BF,ccr                ; init condition code register

;-------------------------------------------------
;       Call driver inits here
;-------------------------------------------------

        jsr     Finitialize_dma_link    ;Link / Init IPP driver (extern)
        jsr     Finit_RTC_driver        ;Link / Init RTC driver (extern)

        jsr     Fmain                   ; run user program


;===========================================
; This should never be encountered - indicates
; end of main() seen. #### Should invoke self-nuke?
;===========================================
        global  F__crt0_end
F__crt0_end
        move   #>-1,x0          ; set signal that we're done, and
        swi                     ; return to the monitor.
        stop                    ; just in case.

;===========================================
        global  F__printf_end   ; avoid loading humungous, useless hosted printf
;===========================================
F__printf_end                   ; dummy printf catch routine.
        swi                     ; invoke monitor
        rts                     ; return to execution after the 'printing'.

        endsec                  ;section crt0


;===========================================
;       section crt0__fp_shift
;===========================================
;
;       org     mm:
;
; floating point table setup   --NOT USED in this system, no floating point
;
;       global  F__fp_shift
;F__fp_shift   dc      $800000,$c00000,$e00000,$f00000,$f80000,$fc0000
;             dc      $fe0000,$ff0000,$ff8000,$ffc000,$ffe000,$fff000
;             dc      $fff800,$fffc00,$fffe00,$ffff00,$ffff80,$ffffc0
;             dc      $ffffe0,$fffff0,$fffff8,$fffffc,$fffffe
;
;       endsec
;-------------------------------------------




;===========================================
; Following section blocks the GNU linker
; from using reserved address spaces as RAM,
; and defines RZP memory map/peripheral space
;===========================================

        section RZP_Addresses

;------------- P: Boot EPROM area

                org     p:$8000
Boot_EEROM      ds      $8000       ;EEWE must be enabled for write

;------------- X: frame buffers

                org     x:$8000

                global  FFrame_X8
FFrame_X8       ds      $1000
                global  FFrame_X9
FFrame_X9       ds      $1000
                global  FFrame_XA
FFrame_XA       ds      $1000
                global  FFrame_XB
FFrame_XB       ds      $1000

;------------- Non-populated X area

                org     x:$C000
                ds      ($FFC0-$C000)   ;block unused area

;------------- Y: frame buffers

                org     y:$8000
                global  FFrame_Y8
FFrame_Y8       ds      $1000
                global  FFrame_Y9
FFrame_Y9       ds      $1000
                global  FFrame_YA
```

```
FFrame_YA       ds      $1000
                global  FFrame_YB
FFrame_YB       ds      $1000

;------------- Non-populated Y area 1

                org     y:$C000
                ds      ($EF00-$C000)   ;block unused area

;------------- Frame buffer registers

                org     y:$EF00
                global  FFrameReg_X8
FFrameReg_X8    ds      1
                global  FFrameReg_X9
FFrameReg_X9    ds      1
                global  FFrameReg_XA
FFrameReg_XA    ds      1
                global  FFrameReg_XB
FFrameReg_XB    ds      1
                global  FFrameReg_Y8
FFrameReg_Y8    ds      1
                global  FFrameReg_Y9
FFrameReg_Y9    ds      1
                global  FFrameReg_YA
FFrameReg_YA    ds      1
                global  FFrameReg_YB
FFrameReg_YB    ds      1

;------------- Non-populated Y area 2

                org     y:$EF08
                ds      ($FFC0-$EF08)   ;block unused area

;------------- DGLUE registers
; see DGLUE.A
;               org     y:$FFC0 ; FFCx so we can use short i/o addressing modes
;DGLUE Registers 0-3
;FRTC_Count     ds      1    ;12-bit r/o
;FRF_Count      ds      1    ;12-bit r/o
;FGPS_Count     ds      1    ;12-bit r/o
;FDGlue_Stat                 ;bits 0..5, 7..11 r/o (addr shared with ctrl reg:)
;FDGlue_Ctrl    ds      1    ;bits 0..6 w/o

;------------- Non-populated Y area 3

                org     y:$FFC4
                ds      ($FFFF-$FFC4)+1   ;block unused area

        endsec

        end     F__start
```

```
;
; This program originally available on the Motorola DSP bulletin board.
; It is provided under a DISCLAMER OF WARRANTY available from
; Motorola DSP Operation, 6501 Wm. Cannon Drive W., Austin, Tx., 78735.
;
; Motorola Standard I/O Equates (lower case).
;
; Last Update 25 Aug 87   Version 1.1  (fixed m_of)
;
;********************************************************************
;
;       EQUATES for DSP56000 I/O registers and ports
;
;********************************************************************

ioequlc ident   1,0

;--------------------------------------------------------------------
;
;       EQUATES for I/O Port Programming
;
;--------------------------------------------------------------------

;       Register Addresses

m_bcr   EQU     $FFFE           ; Port A Bus Control Register
m_pbc   EQU     $FFE0           ; Port B Control Register
m_pbddr EQU     $FFE2           ; Port B Data Direction Register
m_pbd   EQU     $FFE4           ; Port B Data Register
m_pcc   EQU     $FFE1           ; Port C Control Register
m_pcddr EQU     $FFE3           ; Port C Data Direction Register
m_pcd   EQU     $FFE5           ; Port C Data Register


;--------------------------------------------------------------------
;
;       EQUATES for Host Interface
;
;--------------------------------------------------------------------

;       Register Addresses

m_hcr   EQU     $FFE8           ; Host Control Register
m_hsr   EQU     $FFE9           ; Host Status Register
m_hrx   EQU     $FFEB           ; Host Receive Data Register
m_htx   EQU     $FFEB           ; Host Transmit Data Register

;       Host Control Register Bit Flags

m_hrie  EQU     0               ; Host Receive Interrupt Enable
m_htie  EQU     1               ; Host Transmit Interrupt Enable
m_hcie  EQU     2               ; Host Command Interrupt Enable
m_hf2   EQU     3               ; Host Flag 2
m_hf3   EQU     4               ; Host Flag 3

;       Host Status Register Bit Flags

m_hrdf  EQU     0               ; Host Receive Data Full
m_htde  EQU     1               ; Host Transmit Data Empty
m_hcp   EQU     2               ; Host Command Pending
m_hf    EQU     $18             ; Host Flag Mask
m_hf0   EQU     3               ; Host Flag 0
m_hf1   EQU     4               ; Host Flag 1
m_dma   EQU     7               ; DMA Status

;--------------------------------------------------------------------
;
;       EQUATES for Serial Communications Interface (SCI)
;
;--------------------------------------------------------------------

;       Register Addresses

m_srxl  EQU     $FFF4           ; SCI Receive Data Register (low)
m_srxm  EQU     $FFF5           ; SCI Receive Data Register (middle)
m_srxh  EQU     $FFF6           ; SCI Receive Data Register (high)
m_stxl  EQU     $FFF4           ; SCI Transmit Data Register (low)
m_stxm  EQU     $FFF5           ; SCI Transmit Data Register (middle)
m_stxh  EQU     $FFF6           ; SCI Transmit Data Register (high)
m_stxa  EQU     $FFF3           ; SCI Transmit Data Address Register
m_scr   EQU     $FFF0           ; SCI Control Register
m_ssr   EQU     $FFF1           ; SCI Status Register
m_sccr  EQU     $FFF2           ; SCI Clock Control Register

;       SCI Control Register Bit Flags

m_wds   EQU     $3              ; Word Select Mask
m_wds0  EQU     0               ; Word Select 0
m_wds1  EQU     1               ; Word Select 1
m_wds2  EQU     2               ; Word Select 2
m_sbk   EQU     4               ; Send Break
m_wake  EQU     5               ; Wake-up Mode Select
m_rwi   EQU     6               ; Receiver Wake-up Enable
m_woms  EQU     7               ; Wired-OR Mode Select
m_re    EQU     8               ; Receiver Enable
m_te    EQU     9               ; Transmitter Enable
m_ilie  EQU     10              ; Idle Line Interrupt Enable
m_rie   EQU     11              ; Receive Interrupt Enable
m_tie   EQU     12              ; Transmit Interrupt Enable
m_tmie  EQU     13              ; Timer Interrupt Enable

;       SCI Status Register Bit Flags

m_trne  EQU     0               ; Transmitter Empty
m_tdre  EQU     1               ; Transmit Data Register Empty
m_rdrf  EQU     2               ; Receive Data Register Full
m_idle  EQU     3               ; Idle Line
m_or    EQU     4               ; Overrun Error
m_pe    EQU     5               ; Parity Error
m_fe    EQU     6               ; Framing Error
m_r8    EQU     7               ; Received Bit 8

;       SCI Clock Control Register Bit Flags

m_cd    EQU     $FFF            ; Clock Divider Mask
m_cod   EQU     12              ; Clock Out Divider
m_scp   EQU     13              ; Clock Prescaler
m_rcm   EQU     14              ; Receive Clock Source
m_tcm   EQU     15              ; Transmit Clock Source

;--------------------------------------------------------------------
;
;       EQUATES for Synchronous Serial Interface (SSI)
;
;--------------------------------------------------------------------

;       Register Addresses

m_rx    EQU     $FFEF           ; Serial Receive Data Register
m_tx    EQU     $FFEF           ; Serial Transmit Data Register
m_cra   EQU     $FFEC           ; SSI Control Register A
m_crb   EQU     $FFED           ; SSI Control Register B
m_sr    EQU     $FFEE           ; SSI Status Register
m_tsr   EQU     $FFEE           ; SSI Time Slot Register
```

```
;       SSI Control Register A Bit Flags

m_pm    EQU     $FF             ; Prescale Modulus Select Mask
m_dc    EQU     $1F00           ; Frame Rate Divider Control Mask
m_wl    EQU     $6000           ; Word Length Control Mask
m_wl0   EQU     13              ; Word Length Control 0
m_wl1   EQU     14              ; Word Length Control 1
m_psr   EQU     15              ; Prescaler Range

;       SSI Control Register B Bit Flags

m_of    EQU     $3              ; Serial Output Flag Mask
m_of0   EQU     0               ; Serial Output Flag 0
m_of1   EQU     1               ; Serial Output Flag 1
m_scd   EQU     $1C             ; Serial Control Direction Mask
m_scd0  EQU     2               ; Serial Control 0 Direction
m_scd1  EQU     3               ; Serial Control 1 Direction
m_scd2  EQU     4               ; Serial Control 2 Direction
m_sckd  EQU     5               ; Clock Source Direction
m_fsl   EQU     8               ; Frame Sync Length
m_syn   EQU     9               ; Sync/Async Control
m_gck   EQU     10              ; Gated Clock Control
m_mod   EQU     11              ; Mode Select
m_ste   EQU     12              ; SSI Transmit Enable
m_sre   EQU     13              ; SSI Receive Enable
m_stie  EQU     14              ; SSI Transmit Interrupt Enable
m_srie  EQU     15              ; SSI Receive Interrupt Enable

;       SSI Status Register Bit Flags

m_if    EQU     $2              ; Serial Input Flag Mask
m_if0   EQU     0               ; Serial Input Flag 0
m_if1   EQU     1               ; Serial Input Flag 1
m_tfs   EQU     2               ; Transmit Frame Sync
m_rfs   EQU     3               ; Receive Frame Sync
m_tue   EQU     4               ; Transmitter Underrun Error
m_roe   EQU     5               ; Receiver Overrun Error
m_tde   EQU     6               ; Transmit Data Register Empty
m_rdf   EQU     7               ; Receive Data Register Full

;------------------------------------------------------------------------
;
;       EQUATES for Exception Processing
;
;------------------------------------------------------------------------

;       Register Addresses

m_ipr   EQU     $FFFF           ; Interrupt Priority Register

;       Interrupt Priority Register Bit Flags

m_ial   EQU     $7              ; IRQA Mode Mask
m_ial0  EQU     0               ; IRQA Mode Interrupt Priority Level (low)
m_ial1  EQU     1               ; IRQA Mode Interrupt Priority Level (high)
m_ial2  EQU     2               ; IRQA Mode Trigger Mode
m_ibl   EQU     $38             ; IRQB Mode Mask
m_ibl0  EQU     3               ; IRQB Mode Interrupt Priority Level (low)
m_ibl1  EQU     4               ; IRQB Mode Interrupt Priority Level (high)
m_ibl2  EQU     5               ; IRQB Mode Trigger Mode
m_hpl   EQU     $C00            ; Host Interrupt Priority Level Mask
m_hpl0  EQU     10              ; Host Interrupt Priority Level Mask (low)
m_hpl1  EQU     11              ; Host Interrupt Priority Level Mask (high)
m_ssl   EQU     $3000           ; SSI Interrupt Priority Level Mask
m_ssl0  EQU     12              ; SSI Interrupt Priority Level Mask (low)
m_ssl1  EQU     13              ; SSI Interrupt Priority Level Mask (high)
m_scl   EQU     $C000           ; SCI Interrupt Priority Level Mask
m_scl0  EQU     14              ; SCI Interrupt Priority Level Mask (low)
m_scl1  EQU     15              ; SCI Interrupt Priority Level Mask (high)
```

```
; Machine-specific constants and allocations

CPU_Mhz         equ     20.0    ;for HETE CPU
```

```
/* Flag to indicate video data should be generated, and not taken */
/* from the SSI line                                              */
flag simulating;

/* Flag indicating a main-loop should be performed */
flag doloop;

/* Flag indicating main-loops should only be performed on command */
flag single_loop;

/* IPP addresses for self, and recipients of FEP messages */
IPP_address me, video_dest, testmsg_dest;

/* Values set when communications exceptions occur */
int ssi_rx_exception=0, ssi_tx_exception=0, sci_rx_exception=0;

/* Master structure that keeps track of the Frame Buffer's setup */
all_frames_struct fb;
```

```
#include "stdlib.h"
#include "ipp.h"
#include "buddy_chk.h"
#include "BEPwindow.h"

#define min(x,y) (((x)<(y))?(x):(y))
#define max(x,y) (((x)>(y))?(x):(y))

/* Quick macro to get the frame_specs_struct of the frame being written to */
#define f_by_vbp(vbp) fb.fs[fb.current_framenum[(vbp)->region]]


/*** COORDINATE WITH BEPequ.asm ***/
#define VIDBUF_BOTTOM (int *)0
#define VIDBUF_LEN  0x1000
#define VIDBUF_TOP  (int *)(VIDBUF_BOTTOM+VIDBUF_LEN)


#define ZERO_MARK 0
#define ONE_MARK 65535

#define ZEROS4 0
#define ONES4 1
#define MIX4 2

#define FRAME_SYNC1  ONES4
#define FRAME_SYNC2  ONES4

#define ROW_SYNC1  ZEROS4
#define ROW_SYNC2  ONES4
#define ROW_SYNC_SKIP 2  /* # of blank words preceding row sync */
#define ROW_SYNC_LEN 10  /* 2 blank words + 2x4 sync marks  */

#define NUMIDBITS  24

#define SLICES_PER_CAMERA 4
#define NUM_CAMERAS   2
#define TOTALSLICES   (SLICES_PER_CAMERA*NUM_CAMERAS)

#define SLICE_A0 0
#define SLICE_B0 1
#define SLICE_C0 2
#define SLICE_D0 3
#define SLICE_A1 4
#define SLICE_B1 5
#define SLICE_C1 6
#define SLICE_D1 7

#define CAMERA_0 0x01
#define CAMERA_1 0x02
#define CAMERA_BOTH 0x03

/* NOTE: NUMREGIONS is defined in vidcom.h, as it is a constant */
/* that both the BEP and the controlling process (GPP) need to */
/* be aware of.                                                */
#define FRAMES_PER_REGION 1000
#define TOTALFRAMES   (NUMREGIONS*FRAMES_PER_REGION)

#define END_FRAME  0

#define MAX_FB_ADDRESS 0xFFFFFF

typedef unsigned char byte;
typedef byte    flag;
typedef unsigned int fb_ptr;

typedef struct {
                    unsigned short mode;
                    unsigned short used_cameras;
                    unsigned short num_hucs;
                    unsigned short num_cols;
                    unsigned short num_hocs;
                    unsigned short total_cols;
                    unsigned short num_rows;
                    unsigned short num_vocs;
                    unsigned short total_rows;
                } ccd_param_struct;

typedef struct {
                    ccd_param_struct ccd;
                    flagdeal;
                    flagsave_nips;
                    int     start_row[NUM_CAMERAS], end_row[NUM_CAMERAS];
                    int     start_col[NUM_CAMERAS], end_col[NUM_CAMERAS];
                    int     imp_start_offset[NUM_CAMERAS];
                    int     nip_start_offset[NUM_CAMERAS];
                    int     voc_start_offset[NUM_CAMERAS];
                    int     linelen[NUM_CAMERAS];
                    int     size;
                } region_param_struct;

typedef struct {
                    region_param_struct *rp;
                    unsigned int id;
                    fb_ptr frame_start;
                    int     next_frame;
                } frame_specs_struct;

typedef struct {
                    int     region;
                    int     *left_off;
                    int     one_line_delta;
                    int     lines_rcvd;
                    int     lobound_delta[TOTALSLICES];
                    int     hibound_delta[TOTALSLICES];
                    fb_ptr imp_left_off[NUM_CAMERAS];
                    fb_ptr nip_left_off[NUM_CAMERAS];
                    flagwaiting_for_frame_sync;
                    flagsingle_frame;
                    flagsave_frame;
                } vidbuf_param_struct;

typedef struct {
                    region_param_struct rp[NUMREGIONS];
                    frame_specs_struct fs[TOTALFRAMES];
                    fb_ptr region_start[NUMREGIONS+1];
                    int     current_framenum[NUMREGIONS];
                } all_frames_struct;


/*** FEPasmsupport.asm: Global routines ***/
void init_communications();
void copy_lowp_routines();
void init_intr_vectors();
int check_host(unsigned short *value);
int get_host();
void put_host(unsigned short value);
void txd_on(void);
void txd_off(void);
void txd_tog(void);
void sc1_on(void);
void sc1_off(void);
```

```
/*** BEPcom.c: Global routines ***/
void Send_Host_Message(IPP_address dest, IPP_msg_type type,
    IPP_priority priority, unsigned short data_len, IPP_msg_buffer msg);
void send_test_word(unsigned short test_word);
void send_test_int(int test_int);
void try_get_message(vidbuf_param_struct *vbp, vidbuf_param_struct *vbpnew);
void try_relay_errors();
void signal_frame_started(unsigned int id);
void signal_frame_ended(unsigned int id);
void signal_frame_error(unsigned int id, unsigned short reason);


/*** BEPfb.asm: Global routines ***/
void copy_words(int *vb_ptr, int *fb_win_ptr, int step, int dist);
void copy_hi_pixels(int *vb_ptr, int *fb_win_ptr, int step, int dist);
void copy_lo_pixels(int *vb_ptr, int *fb_win_ptr, int step, int dist);
void read_fb16(fb_ptr fb_start, flag startlo, int step, int *pixs, int dist);
void read_fb12(fb_ptr fb_start, flag startlo, int step, int *pixs, int dist);
void tbt_lo(fb_ptr fb_start, int line_inc, int *pixs);
void tbt_hi(fb_ptr fb_start, int line_inc, int *pixs);


/*** BEPframe.c: Global routines ***/
int find_frame_in_region(unsigned int id, int region);
int find_frame(unsigned int id);
void save_frame(unsigned int id);
void unsave_frame(unsigned int id);
void dump_frame_raw(IPP_msg_buffer msg, IPP_address dest);
void dump_imp_array(IPP_msg_buffer msg, IPP_address dest);
void dump_tbt_array(IPP_msg_buffer msg, IPP_address dest);
void set_up_region(int region);
void fill_out_params(region_param_struct *rp);


/*** BEPmain.c: Global routines ***/
void memdump_cmd(IPP_msg_buffer msg);
void mempoke_cmd(IPP_msg_buffer msg);
void memdump_auto(char space, unsigned short start, unsigned short end);


/*** BEPsetvars.c: Global routines ***/
void set_vbp_new_config(vidbuf_param_struct *vbp);
void set_vbp_left_offs(vidbuf_param_struct *vbp);
void init_portc_intrs();
void onetime_init(vidbuf_param_struct *vbp, vidbuf_param_struct *vbpnew);


/*** BEPvidin.c: Global routines ***/
void try_copy_line(vidbuf_param_struct *vbp, vidbuf_param_struct *vbpnew);
void abort_stream(vidbuf_param_struct *vbp, vidbuf_param_struct *vbpnew);


/*** BEPvidout.c: Global routines ***/
flag read_imp(int cam, int scol, int ecol, int srow, int erow, int *pixs,
    frame_specs_struct *fs);
flag read_nip(int cam, int slice, int niptype, int scol, int ecol,
    int srow, int erow, int *pixs, frame_specs_struct *fs);
void tbt(int cam, int llx, int lly, int *pixs, frame_specs_struct *fs);


/*** BEPwindow.asm: Global routines ***/
int *setup_window(int win_num, fb_ptr frame_ptr);
int *setup_dual_window(int win_num, fb_ptr frame_ptr);
int *setup_triple_window(int win_num, fb_ptr frame_ptr);


/*** Assembly-declared variables ***/
register int *asm_inptr   __asm("r5");
```

```
/* Beginning of FB window control registers (in Y: peripheral space) */
#define PAGE_CONTROLS (int *)0xEF00

#define read_fb_win(win_ptr) *(win_ptr)

#define NUMWINDOWS 8
#define WINDOW_SIZE 0x1000

#define WX0 0       /* Window from X:$8000-$8FFF */
#define WX1 1       /* Window from X:$9000-$9FFF */
#define WX2 2       /* Window from X:$A000-$AFFF */
#define WX3 3       /* Window from X:$B000-$BFFF */
#define WY0 4       /* Window from Y:$8000-$8FFF */
#define WY1 5       /* Window from Y:$9000-$9FFF */
#define WY2 6       /* Window from Y:$A000-$AFFF */
#define WY3 7       /* Window from Y:$B000-$BFFF */

#define WX0_START (int *)0x8000
#define WX0_END  (int *)(WX0_START + WINDOW_SIZE)
#define WX1_START (int *)0x9000
#define WX1_END  (int *)(WX1_START + WINDOW_SIZE)
#define WX2_START (int *)0xA000
#define WX2_END  (int *)(WX2_START + WINDOW_SIZE)
#define WX3_START (int *)0xB000
#define WX3_END  (int *)(WX3_START + WINDOW_SIZE)
#define WY0_START (int *)0x8000
#define WY0_END  (int *)(WY0_START + WINDOW_SIZE)
#define WY1_START (int *)0x9000
#define WY1_END  (int *)(WY1_START + WINDOW_SIZE)
#define WY2_START (int *)0xA000
#define WY2_END  (int *)(WY2_START + WINDOW_SIZE)
#define WY3_START (int *)0xB000
#define WY3_END  (int *)(WY3_START + WINDOW_SIZE)
```

```
include $(HETEDIR)/Makefiles/global_defs

INCLUDE = $(HETEINCLUDE)
NAME= BEP
GAWK = /hete/bin/nextbin/cldlodfix.gawk
NEXTINCLUDE = $(HETEINCLUDE)/nextinclude

PBCLNs = pbcln/BEPmain.cln pbcln/BEPsetvars.cln pbcln/BEPvidin.cln\
    pbcln/BEPvidout.cln pbcln/BEPcom.cln pbcln/BEPframe.cln\
    pbcln/BEPintr.cln pbcln/BEPfb.cln pbcln/BEPwindow.cln\
    pbcln/BEPasmsupport.cln
PBCRT = pbcln/crtrzpy.cln
PBOPTS = -g -c -Wall -alo -ffixed-r5 -DUSEIPP -I$(PBDSPINCLUDE) -I$(INCLUDE)
PBSUPPORT = machspec.asm BEPequ.asm
PBLIBCLNS = $(PBDSPLIB)/ippshell.cln

ASMs = asm/BEPmain.asm asm/BEPsetvars.asm asm/BEPvidin.asm\
    asm/BEPvidout.asm asm/BEPcom.asm asm/BEPframe.asm\
    asm/BEPintr.asm asm/BEPfb.asm asm/BEPwindow.asm\
    asm/BEPasmsupport.asm
ASMOPTS = -g -c -Wall -alo -ffixed-r5 -S -DUSEIPP -I$(PBDSPINCLUDE) -I$(INCLUDE)


ifeq ($(PBDSP),YES)
PBDSPLODFILES := pb_$(NAME).hld
endif


PBBEP: $(PBCRT) $(PBDSPLODFILES)

ASMBEP: $(ASMs)


pb_$(NAME).hld: obj/pb_$(NAME).lod
    lod2hostload obj/pb_$(NAME).lod
    rm -f pb_$(NAME).hld
    mv obj/pb_$(NAME).hld .
    rm -f /benz/h1/cosmosct/visible/pb_$(NAME).hld
    cp pb_$(NAME).hld /benz/h1/cosmosct/visible/

obj/pb_$(NAME).lod: obj/pb_$(NAME).cld
    cldlod obj/pb_$(NAME).cld | gawk -f $(GAWK) > obj/pb_$(NAME).lod.realspace
    strip_last.sh obj/pb_$(NAME).lod.realspace
    ../utils/shiftlowp obj/pb_$(NAME).lod.realspace obj/pb_$(NAME).lod

obj/pb_$(NAME).cld: $(PBCLNs) $(SUPPORTFILES) Makefile
    g56k -j -mobj/pb_$(NAME).map -my-memory -mstack_check -crt $(PBCRT) $(PBSUPPORT) $(P
BCLNs) $(PBLIBCLNS)  o obj/pb_$(NAME).cld -l$(PBDSPLIB)/librzpy.clb -I$(PBDSPINCLUDE) -I
$(INCLUDE)


pbcln/BEPasmsupport.cln: BEPasmsupport.asm Makefile
    g56k BEPasmsupport.asm $(PBOPTS) -o pbcln/BEPasmsupport.cln

asm/BEPasmsupport.asm: BEPasmsupport.asm Makefile
    cp BEPasmsupport.asm asm/BEPasmsupport.asm


pbcln/BEPcom.cln: BEPcom.c Makefile
    g56k BEPcom.c $(PBOPTS) -o pbcln/BEPcom.cln

asm/BEPcom.asm: BEPcom.c Makefile
    g56k BEPcom.c $(ASMOPTS) -o asm/BEPcom.asm


pbcln/BEPfb.cln: BEPfb.asm Makefile
    g56k BEPfb.asm $(PBOPTS) -o pbcln/BEPfb.cln

asm/BEPfb.asm: BEPasmsupport.asm Makefile
    cp BEPfb.asm asm/BEPfb.asm


pbcln/BEPfb_s.cln: BEPfb_s.c Makefile
    g56k BEPfb_s.c $(PBOPTS) -o pbcln/BEPfb_s.cln

asm/BEPfb_s.asm: BEPfb_s.c Makefile
    g56k BEPfb_s.c $(ASMOPTS) -o asm/BEPfb_s.asm


pbcln/BEPframe.cln: BEPframe.c Makefile
    g56k BEPframe.c $(PBOPTS) -o pbcln/BEPframe.cln

asm/BEPframe.asm: BEPframe.c Makefile
    g56k BEPframe.c $(ASMOPTS) -o asm/BEPframe.asm


pbcln/BEPintr.cln: BEPintr.asm Makefile
    g56k BEPintr.asm $(PBOPTS) -o pbcln/BEPintr.cln

asm/BEPintr.asm: BEPasmsupport.asm Makefile
    cp BEPintr.asm asm/BEPintr.asm


pbcln/BEPmain.cln: BEPmain.c Makefile
    g56k BEPmain.c $(PBOPTS) -o pbcln/BEPmain.cln

asm/BEPmain.asm: BEPmain.c Makefile
    g56k BEPmain.c $(ASMOPTS) -o asm/BEPmain.asm


pbcln/BEPsetvars.cln: BEPsetvars.c Makefile
    g56k BEPsetvars.c $(PBOPTS) -o pbcln/BEPsetvars.cln

asm/BEPsetvars.asm: BEPsetvars.c Makefile
    g56k BEPsetvars.c $(ASMOPTS) -o asm/BEPsetvars.asm


pbcln/BEPvidin.cln: BEPvidin.c Makefile
    g56k BEPvidin.c $(PBOPTS) -o pbcln/BEPvidin.cln

asm/BEPvidin.asm: BEPvidin.c Makefile
    g56k BEPvidin.c $(ASMOPTS) -o asm/BEPvidin.asm


pbcln/BEPvidout.cln: BEPvidout.c Makefile
    g56k BEPvidout.c $(PBOPTS) -o pbcln/BEPvidout.cln

asm/BEPvidout.asm: BEPvidout.c Makefile
    g56k BEPvidout.c $(ASMOPTS) -o asm/BEPvidout.asm


pbcln/BEPwindow.cln: BEPwindow.asm Makefile
    g56k BEPwindow.asm $(PBOPTS) -o pbcln/BEPwindow.cln

asm/BEPwindow.asm: BEPasmsupport.asm Makefile
    cp BEPwindow.asm asm/BEPwindow.asm


pbcln/BEPwindow_s.cln: BEPwindow_s.c Makefile
    g56k BEPwindow_s.c $(PBOPTS) -o pbcln/BEPwindow_s.cln

asm/BEPwindow_s.asm: BEPwindow_s.c Makefile
    g56k BEPwindow_s.c $(ASMOPTS) -o asm/BEPwindow_s.asm
```

```
pbcln/crtrzpy.cln: crtrzp.asm Makefile
    asm56000 -I$(HETEDIR)/lib/pbdsplib/src -bpbcln/crtrzpy.cln -lobj/crtrzpy.lst -dmm Y
crtrzp.asm


include $(HETEDIR)/Makefiles/global_rules
```

Section Link Map by Address

X Memory (default)

Start	End	Length	Section	
0000	0FFF	4096	GLOBAL	Abs Mod
8000	FFBF	32704	RZP_Addresses	Abs

X Memory (low)

Start	End	Length	Section

No sections

X Memory (high)

Start	End	Length	Section

No sections

Y Memory (default)

Start	End	Length	Section	
0000	0012	19	GLOBAL	Abs
0013	0018	6	crt0	
0019	304E	12342	BEPmain_c	
304F	3051	3	BEPvidin_c	
3052	309B	74	BEPcom_c	
309C	30AB	16	ippshell_c	
30AC	30AC	1	ipp_driver	
30AD	30AE	2	RTC_handler	
30AF	30C0	18	buddy_chk_c	
8000	FFBF	32704	RZP_Addresses	Abs
FFC4	FFFF	60	RZP_Addresses	Abs

Y Memory (low)

Start	End	Length	Section

No sections

Y Memory (high)

Start	End	Length	Section

No sections

L Memory (default)

Start	End	Length	Section

No sections

L Memory (low)

Start	End	Length	Section

No sections

L Memory (high)

Start	End	Length	Section

No sections

P Memory (default)

Start	End	Length	Section	
0000	0AAA	2731	bootstrapper	Abs
0040	0043	4	interrupts	Abs *Overlap*
0050	01A4	341	fb_io	Abs
0AAB	0AD6	44	crt0	Abs
0AD7	0EC0	1002	BEPmain_c	
0EC1	117E	702	BEPsetvars_c	
117F	17BE	1600	BEPvidin_c	
17BF	1E02	1604	BEPvidout_c	
1E03	24A6	1700	BEPcom_c	
24A7	2BD4	1838	BEPframe_c	
2BD5	2C4F	123	window	
2C50	2CB5	102	asmsupport	
2CB6	2F83	718	ippshell_c	
2F84	3020	157	ipp_driver	
3021	309E	126	RTC_handler	
309F	311B	125	buddy_chk_c	
311C	3143	40	mcpy_c	
3144	31EB	168	sEc_longdiv709	
8000	FFFF	32768	RZP_Addresses	Abs

P Memory (low)

Start	End	Length	Section

No sections

P Memory (high)

Start	End	Length	Section

No sections

Section Link Map by Name

Section	Memory	Start	End	Length
BEPcom_c	Y default	3052	309B	74
	P default	1E03	24A6	1700

BEPframe_c		P	default	24A7	2BD4	1838
BEPmain_c		Y	default	0019	304E	12342
		P	default	0AD7	0EC0	1002
BEPsetvars_c		P	default	0EC1	117E	702
BEPvidin_c		Y	default	304F	3051	3
		P	default	117F	17BE	1600
BEPvidout_c		P	default	17BF	1E02	1604
GLOBAL	Mod	X	default	0000	0FFF	4096
	Abs	Y	default	0000	0012	19
RTC_handler		Y	default	30AD	30AE	2
		P	default	3021	309E	126
RZP_Addresses	Abs	X	default	8000	FFBF	32704
	Abs	Y	default	8000	FFBF	32704
	Abs	Y	default	FFC4	FFFF	60
	Abs	P	default	8000	FFFF	32768
asmsupport		P	default	2C50	2CB5	102
bootstrapper	Abs	P	default	0000	0AAA	2731
buddy_chk_c		Y	default	30AF	30C0	18
		P	default	309F	311B	125
crt0		Y	default	0013	0018	6
	Abs	P	default	0AAB	0AD6	44
fb_io	Abs	P	default	0050	01A4	341
interrupts	Abs	P	default	0040	0043	4
ipp_driver		Y	default	30AC	30AC	1
		P	default	2F84	3020	157
ippshell_c		Y	default	309C	30AB	16
		P	default	2CB6	2F83	718
mcpy_c		P	default	311C	3143	40
sEc_longdiv709		P	default	3144	31EB	168
window		P	default	2BD5	2C4F	123

Symbol Listing by Name

Name	Type	Value	Section	Attributes
ABS_JSR_INSTR....	int	0BF080	ipp_driver	ABS LOCAL
ACK_DT.....	int	000004	ipp_driver	ABS LOCAL
ACK_PORT.....	int	00FFE8	ipp_driver	ABS LOCAL
ACK_TD.....	int	000003	ipp_driver	ABS LOCAL
ASM_APP_0.....	int	P:000AE3	BEPmain_c	REL LOCAL
ASM_APP_0.....	int	P:000EC6	BEPsetvars_c	REL LOCAL
ASM_APP_0.....	int	P:0011D6	BEPvidin_c	REL LOCAL
ASM_APP_0.....	int	P:00215B	BEPcom_c	REL LOCAL
ASM_APP_1.....	int	P:000AE4	BEPmain_c	REL LOCAL
ASM_APP_1.....	int	P:000EC7	BEPsetvars_c	REL LOCAL
ASM_APP_1.....	int	P:0011D8	BEPvidin_c	REL LOCAL
ASM_APP_1.....	int	P:00215C	BEPcom_c	REL LOCAL
ASM_APP_10.....	int	P:000CB3	BEPmain_c	REL LOCAL
ASM_APP_11.....	int	P:000CB5	BEPmain_c	REL LOCAL
ASM_APP_12.....	int	P:000D3A	BEPmain_c	REL LOCAL
ASM_APP_13.....	int	P:000D3C	BEPmain_c	REL LOCAL
ASM_APP_14.....	int	P:000E2B	BEPmain_c	REL LOCAL
ASM_APP_15.....	int	P:000E2C	BEPmain_c	REL LOCAL
ASM_APP_16.....	int	P:000E31	BEPmain_c	REL LOCAL
ASM_APP_17.....	int	P:000E32	BEPmain_c	REL LOCAL
ASM_APP_2.....	int	P:000AEA	BEPmain_c	REL LOCAL
ASM_APP_2.....	int	P:000EC9	BEPsetvars_c	REL LOCAL
ASM_APP_2.....	int	P:0011E3	BEPvidin_c	REL LOCAL
ASM_APP_3.....	int	P:000AEB	BEPmain_c	REL LOCAL
ASM_APP_3.....	int	P:000ECA	BEPsetvars_c	REL LOCAL
ASM_APP_3.....	int	P:0011E4	BEPvidin_c	REL LOCAL
ASM_APP_4.....	int	P:000B4D	BEPmain_c	REL LOCAL
ASM_APP_4.....	int	P:0010C4	BEPsetvars_c	REL LOCAL
ASM_APP_4.....	int	P:0011EE	BEPvidin_c	REL LOCAL
ASM_APP_5.....	int	P:000B4E	BEPmain_c	REL LOCAL
ASM_APP_5.....	int	P:0010C5	BEPsetvars_c	REL LOCAL
ASM_APP_5.....	int	P:0011EF	BEPvidin_c	REL LOCAL
ASM_APP_6.....	int	P:000BAE	BEPmain_c	REL LOCAL
ASM_APP_6.....	int	P:0011F9	BEPvidin_c	REL LOCAL
ASM_APP_7.....	int	P:000BAF	BEPmain_c	REL LOCAL
ASM_APP_7.....	int	P:0011FA	BEPvidin_c	REL LOCAL
ASM_APP_8.....	int	P:000BCA	BEPmain_c	REL LOCAL
ASM_APP_8.....	int	P:0012B9	BEPvidin_c	REL LOCAL
ASM_APP_9.....	int	P:000BCB	BEPmain_c	REL LOCAL
ASM_APP_9.....	int	P:0012BB	BEPvidin_c	REL LOCAL
ASM_APP_A.....	int	P:000BDE	BEPmain_c	REL LOCAL
ASM_APP_A.....	int	P:0012C9	BEPvidin_c	REL LOCAL
ASM_APP_B.....	int	P:000BDF	BEPmain_c	REL LOCAL
ASM_APP_B.....	int	P:0012CA	BEPvidin_c	REL LOCAL
ASM_APP_C.....	int	P:000BF0	BEPmain_c	REL LOCAL
ASM_APP_D.....	int	P:000BF1	BEPmain_c	REL LOCAL
ASM_APP_E.....	int	P:000C68	BEPmain_c	REL LOCAL
ASM_APP_F.....	int	P:000C69	BEPmain_c	REL LOCAL
BASE_PROC_ID.....	int	000004	ipp_driver	ABS LOCAL
BOARD_ID_MASK....	int	000F00	GLOBAL	ABS GLOBAL
BOARD_ID_MASK....	int	000F00	ipp_driver	ABS LOCAL
BOARD_ID_MASK....	int	000F00	RTC_handler	ABS LOCAL
Boot_EEROM.....	int	P:008000	RZP_Addresses	ABS LOCAL
CPU_Mhz.....	fpt	2.000000E+001	GLOBAL	ABS GLOBAL
CRT0ORG.....	int	000AAB	GLOBAL	ABS GLOBAL
DEBUG_BIT_MASK...	int	000060	GLOBAL	ABS GLOBAL

DEBUG_BIT_MASK...int	000060	ipp_driver	ABS	LOCAL
DEBUG_BIT_MASK...int	000060	RTC_handler	ABS	LOCAL
DSIZE.....int	0030C1	GLOBAL	ABS	GLOBAL
DSP_ID_BITNO.....int	000007	GLOBAL	ABS	GLOBAL
DSP_ID_BITNO.....int	000007	ipp_driver	ABS	LOCAL
DSP_ID_BITNO.....int	000007	RTC_handler	ABS	LOCAL
DT_MODE.....int	000001	ipp_driver	ABS	LOCAL
DT_MODEBIT.....int	000000	ipp_driver	ABS	LOCAL
EEPROM_WE_BITNO..int	000004	GLOBAL	ABS	GLOBAL
EEPROM_WE_BITNO..int	000004	ipp_driver	ABS	LOCAL
EEPROM_WE_BITNO..int	000004	RTC_handler	ABS	LOCAL
FDGlue_Ctrl.....int	00FFC3	GLOBAL	ABS	GLOBAL
FDGlue_Ctrl.....int	00FFC3	ipp_driver	ABS	LOCAL
FDGlue_Ctrl.....int	00FFC3	RTC_handler	ABS	LOCAL
FDGlue_Stat.....int	00FFC3	GLOBAL	ABS	GLOBAL
FDGlue_Stat.....int	00FFC3	ipp_driver	ABS	LOCAL
FDGlue_Stat.....int	00FFC3	RTC_handler	ABS	LOCAL
FFrameReg_X8.....int	Y:00EF00	RZP_Addresses	ABS	GLOBAL
FFrameReg_X9.....int	Y:00EF01	RZP_Addresses	ABS	GLOBAL
FFrameReg_XA.....int	Y:00EF02	RZP_Addresses	ABS	GLOBAL
FFrameReg_XB.....int	Y:00EF03	RZP_Addresses	ABS	GLOBAL
FFrameReg_Y8.....int	Y:00EF04	RZP_Addresses	ABS	GLOBAL
FFrameReg_Y9.....int	Y:00EF05	RZP_Addresses	ABS	GLOBAL
FFrameReg_YA.....int	Y:00EF06	RZP_Addresses	ABS	GLOBAL
FFrameReg_YB.....int	Y:00EF07	RZP_Addresses	ABS	GLOBAL
FFrame_X8.....int	X:008000	RZP_Addresses	ABS	GLOBAL
FFrame_X9.....int	X:009000	RZP_Addresses	ABS	GLOBAL
FFrame_XA.....int	X:00A000	RZP_Addresses	ABS	GLOBAL
FFrame_XB.....int	X:00B000	RZP_Addresses	ABS	GLOBAL
FFrame_Y8.....int	Y:008000	RZP_Addresses	ABS	GLOBAL
FFrame_Y9.....int	Y:009000	RZP_Addresses	ABS	GLOBAL
FFrame_YA.....int	Y:00A000	RZP_Addresses	ABS	GLOBAL
FFrame_YB.....int	Y:00B000	RZP_Addresses	ABS	GLOBAL
FGPS_Count.....int	00FFC2	GLOBAL	ABS	GLOBAL
FGPS_Count.....int	00FFC2	ipp_driver	ABS	LOCAL
FGPS_Count.....int	00FFC2	RTC_handler	ABS	LOCAL
FGet_Host_Messag.int	P:001E49	BEPcom_c	REL	GLOBAL
FIPP_collectmsg..int	P:002CB6	ippshell_c	REL	GLOBAL
FIPP_get_address.int	P:002F49	ippshell_c	REL	GLOBAL
FIPP_get_message.int	P:002D4F	ippshell_c	REL	GLOBAL
FIPP_make_header.int	P:002DD1	ippshell_c	REL	GLOBAL
FIPP_makeheader..int	P:002E41	ippshell_c	REL	GLOBAL
FIPP_register_ad.int	P:002EBF	ippshell_c	REL	GLOBAL
FIPP_send_messag.int	P:002D89	ippshell_c	REL	GLOBAL
FRF_Count.....int	00FFC1	GLOBAL	ABS	GLOBAL
FRF_Count.....int	00FFC1	ipp_driver	ABS	LOCAL
FRF_Count.....int	00FFC1	RTC_handler	ABS	LOCAL
FRTC_Count.....int	00FFC0	GLOBAL	ABS	GLOBAL
FRTC_Count.....int	00FFC0	ipp_driver	ABS	LOCAL
FRTC_Count.....int	00FFC0	RTC_handler	ABS	LOCAL
FSend_Host_Messa.int	P:001E03	BEPcom_c	REL	GLOBAL
F___ABCDcnt1.....int	Y:0000FE	BEPmain_c	REL	LOCAL
F___Aline2.....int	Y:0000FF	BEPmain_c	REL	LOCAL
F___Bline3.....int	Y:000100	BEPmain_c	REL	LOCAL
F___Cline4.....int	Y:000101	BEPmain_c	REL	LOCAL
F___Dline5.....int	Y:000102	BEPmain_c	REL	LOCAL
F___count1.....int	Y:003050	BEPvidin_c	REL	LOCAL
F___got_header0..int	Y:00309C	ippshell_c	REL	LOCAL
F___header1.....int	Y:00309D	ippshell_c	REL	LOCAL
F___id0.....int	Y:0000FD	BEPmain_c	REL	LOCAL

F___last_tick0...int	Y:0030B3	buddy_chk_c	REL	LOCAL
F___line_written.int	Y:000104	BEPmain_c	REL	LOCAL
F___local_id2....int	Y:003051	BEPvidin_c	REL	LOCAL
F___sim_cntr6....int	Y:000103	BEPmain_c	REL	LOCAL
F___state0.....int	Y:00304F	BEPvidin_c	REL	LOCAL
F__break.....int	Y:000016	crt0	REL	GLOBAL
F__crt0_end.....int	P:000AD1	crt0	ABS	GLOBAL
F__max_signal....int	Y:000018	crt0	REL	GLOBAL
F__mem_limit.....int	Y:000015	crt0	REL	GLOBAL
F__printf_end....int	P:000AD5	crt0	ABS	GLOBAL
F__stack_safety..int	Y:000014	crt0	REL	GLOBAL
F__start.....int	P:000AAB	crt0	ABS	GLOBAL
F__y_size.....int	Y:000017	crt0	REL	GLOBAL
Fabort_stream....int	P:0017B1	BEPvidin_c	REL	GLOBAL
Falt_simulate_wo.int	P:000C38	BEPmain_c	REL	GLOBAL
Fbuddy_hdr.....int	Y:0030B4	buddy_chk_c	REL	GLOBAL
Fccd0sim.....int	Y:00001D	BEPmain_c	REL	GLOBAL
Fccd1sim.....int	Y:00008D	BEPmain_c	REL	GLOBAL
Fcheck4.....int	P:0011C5	BEPvidin_c	REL	GLOBAL
Fcheck_buddy_tim.int	P:00309F	buddy_chk_c	REL	GLOBAL
Fcheck_host.....int	P:002C78	asmsupport	REL	GLOBAL
Fcopy_hi_pixels..int	P:000069	fb_io	ABS	GLOBAL
Fcopy_line.....int	P:00154E	BEPvidin_c	REL	GLOBAL
Fcopy_lo_pixels..int	P:000096	fb_io	ABS	GLOBAL
Fcopy_lowp_routi.int	P:002C5D	asmsupport	REL	GLOBAL
Fcopy_words.....int	P:000050	fb_io	ABS	GLOBAL
Fdeal_into_fb....int	P:0013DA	BEPvidin_c	REL	GLOBAL
Fdoloop.....int	Y:00304D	BEPmain_c	REL	GLOBAL
Fdrop_frame_sync.int	Y:00001C	BEPmain_c	REL	GLOBAL
Fdump_frame_raw..int	P:0025F3	BEPframe_c	REL	GLOBAL
Fdump_imp_array..int	P:0026CC	BEPframe_c	REL	GLOBAL
Fdump_tbt_array..int	P:0028A6	BEPframe_c	REL	GLOBAL
Ferrno.....int	Y:000013	crt0	REL	GLOBAL
Ffb.....int	Y:000105	BEPmain_c	REL	GLOBAL
Ffill_out_params.int	P:002A43	BEPframe_c	REL	GLOBAL
Ffind_frame.....int	P:0024FC	BEPframe_c	REL	GLOBAL
Ffind_frame_in_r.int	P:0024A7	BEPframe_c	REL	GLOBAL
Fget_host.....int	P:002C8D	asmsupport	REL	GLOBAL
Fget_imp_offset..int	P:0017BF	BEPvidout_c	REL	GLOBAL
Fget_irqa_tick...int	P:00303F	RTC_handler	REL	GLOBAL
Fget_nsynch_time.int	P:003043	RTC_handler	REL	GLOBAL
Finit_RTC_driver.int	P:003021	RTC_handler	REL	GLOBAL
Finit_communicat.int	P:002C50	asmsupport	REL	GLOBAL
Finit_intr_vecto.int	P:002C69	asmsupport	REL	GLOBAL
Finit_portc_intr.int	P:000EC1	BEPsetvars_c	REL	GLOBAL
Finitialize_dma_.int	P:002F84	ipp_driver	REL	GLOBAL
Flocal_processor.int	Y:0030AA	ippshell_c	REL	GLOBAL
Fmain.....int	P:000E69	BEPmain_c	REL	GLOBAL
Fmajortag.....int	Y:0030AF	buddy_chk_c	REL	GLOBAL
Fmaster_node_id..int	Y:0030AB	ippshell_c	REL	LOCAL
Fme.....int	Y:003048	BEPmain_c	REL	GLOBAL
Fmemcpy.....int	P:00311C	mcpy_c	REL	GLOBAL
Fmemdump_auto....int	P:000E3F	BEPmain_c	REL	GLOBAL
Fmemdump_cmd.....int	P:000D57	BEPmain_c	REL	GLOBAL
Fmempoke_cmd.....int	P:000DF6	BEPmain_c	REL	GLOBAL
Fminortag.....int	Y:0030B1	buddy_chk_c	REL	GLOBAL
Fmy_ipp_addr.....int	Y:0030C0	buddy_chk_c	REL	GLOBAL
Fmy_name.....int	Y:0030A9	ippshell_c	REL	LOCAL
Fonetime_init....int	P:001097	BEPsetvars_c	REL	GLOBAL
Fparam_dump.....int	P:001EE5	BEPcom_c	REL	GLOBAL

```
Fparse_viddsp_cm.int  P:002087          BEPcom_c          REL GLOBAL
Fpmemcpy.........int  P:000D13          BEPmain_c         REL GLOBAL
Fprepare_IPP_hea.int  P:0030CC          buddy_chk_c       REL GLOBAL
Fput_host........int  P:002C93          asmsupport        REL GLOBAL
Fread_fb12.......int  P:0000E7          fb_io             ABS GLOBAL
Fread_fb16.......int  P:0000C3          fb_io             ABS GLOBAL
Fread_imp........int  P:001928          BEPvidout_c       REL GLOBAL
Fread_nip........int  P:001AEC          BEPvidout_c       REL GLOBAL
Freceive_words...int  P:002FA3          ipp_driver        REL GLOBAL
Frp_dump.........int  P:001F97          BEPcom_c          REL GLOBAL
Fsave_frame......int  P:002527          BEPframe_c        REL GLOBAL
Fsc1_off.........int  P:002CB4          asmsupport        REL GLOBAL
Fsc1_on..........int  P:002CB2          asmsupport        REL GLOBAL
Fsci_rx_exceptio.int  Y:00001B          BEPmain_c         REL GLOBAL
Fsend_test_int...int  P:001EA2          BEPcom_c          REL GLOBAL
Fsend_test_word..int  P:001E6C          BEPcom_c          REL GLOBAL
Fsend_words......int  P:002FE3          ipp_driver        REL GLOBAL
Fset_up_region...int  P:002932          BEPframe_c        REL GLOBAL
Fset_vbp_left_of.int  P:000ECB          BEPsetvars_c      REL GLOBAL
Fset_vbp_new_con.int  P:000F38          BEPsetvars_c      REL GLOBAL
Fsetup_dual_wind.int  P:002BFB          window            REL GLOBAL
Fsetup_triple_wi.int  P:002C24          window            REL GLOBAL
Fsetup_window....int  P:002BD5          window            REL GLOBAL
Fsignal_frame_en.int  P:002422          BEPcom_c          REL GLOBAL
Fsignal_frame_er.int  P:002461          BEPcom_c          REL GLOBAL
Fsignal_frame_st.int  P:0023E3          BEPcom_c          REL GLOBAL
Fsim4............int  P:000AD7          BEPmain_c         REL GLOBAL
Fsimulate_frame_.int  P:000AEF          BEPmain_c         REL GLOBAL
Fsimulate_row_sy.int  P:000B47          BEPmain_c         REL GLOBAL
Fsimulate_word_r.int  P:000B5E          BEPmain_c         REL GLOBAL
Fsimulating......int  Y:00304E          BEPmain_c         REL GLOBAL
Fsingle_loop.....int  Y:00304C          BEPmain_c         REL GLOBAL
Fssi_rx_exceptio.int  Y:000019          BEPmain_c         REL GLOBAL
Fssi_tx_exceptio.int  Y:00001A          BEPmain_c         REL GLOBAL
Fswitch_params...int  P:00121B          BEPvidin_c        REL GLOBAL
Fsync_frame......int  P:001276          BEPvidin_c        REL GLOBAL
Fsync_row........int  P:001377          BEPvidin_c        REL GLOBAL
Ftbt.............int  P:001D3C          BEPvidout_c       REL GLOBAL
Ftbt_hi..........int  P:00011C          fb_io             ABS GLOBAL
Ftbt_lo..........int  P:000165          fb_io             ABS GLOBAL
Ftestmsg_dest....int  Y:003040          BEPmain_c         REL GLOBAL
Ftry_copy_line...int  P:0016C2          BEPvidin_c        REL GLOBAL
Ftry_get_message.int  P:0021A8          BEPcom_c          REL GLOBAL
Ftry_relay_error.int  P:0023A6          BEPcom_c          REL GLOBAL
Ftxd_off.........int  P:002CAE          asmsupport        REL GLOBAL
Ftxd_on..........int  P:002CAC          asmsupport        REL GLOBAL
Ftxd_tog.........int  P:002CB0          asmsupport        REL GLOBAL
Funsave_frame....int  P:00259B          BEPframe_c        REL GLOBAL
Fvb_add..........int  P:00117F          BEPvidin_c        REL GLOBAL
Fvb_distance.....int  P:0011AA          BEPvidin_c        REL GLOBAL
Fvideo_dest......int  Y:003044          BEPmain_c         REL GLOBAL
Fxmemcpy.........int  P:000C8C          BEPmain_c         REL GLOBAL
Fymemcpy.........int  P:000CD0          BEPmain_c         REL GLOBAL
GPP_REQ_DT_ADR...int    000028          ipp_driver        ABS LOCAL
GPP_REQ_DT_NUM...int    000014          ipp_driver        ABS LOCAL
HFB_EVENT_BITNO..int    000005          GLOBAL            ABS GLOBAL
HFB_EVENT_BITNO..int    000005          ipp_driver        ABS LOCAL
HFB_EVENT_BITNO..int    000005          RTC_handler       ABS LOCAL
IRQA_L0_BIT......int    000000          RTC_handler       ABS LOCAL
IRQA_L1_BIT......int    000001          RTC_handler       ABS LOCAL
IRQA_SEN_BIT.....int    000002          RTC_handler       ABS LOCAL
IRQA_SRCSTAT_BIT.int    000001          GLOBAL            ABS GLOBAL
IRQA_SRCSTAT_BIT.int    000001          ipp_driver        ABS LOCAL
IRQA_SRCSTAT_BIT.int    000001          RTC_handler       ABS LOCAL
IRQA_SRC_BITNO...int    000004          GLOBAL            ABS GLOBAL
IRQA_SRC_BITNO...int    000004          ipp_driver        ABS LOCAL
IRQA_SRC_BITNO...int    000004          RTC_handler       ABS LOCAL
IRQA_STAT_BITNO..int    000003          GLOBAL            ABS GLOBAL
IRQA_STAT_BITNO..int    000003          ipp_driver        ABS LOCAL
IRQA_STAT_BITNO..int    000003          RTC_handler       ABS LOCAL
IRQA_VEC_ADR.....int    000008          RTC_handler       ABS LOCAL
IRQB_STAT_BITNO..int    000002          GLOBAL            ABS GLOBAL
IRQB_STAT_BITNO..int    000002          ipp_driver        ABS LOCAL
IRQB_STAT_BITNO..int    000002          RTC_handler       ABS LOCAL
IRQ_SRC_BIT......int    000004          RTC_handler       ABS LOCAL
L1...............int  P:002D46          ippshell_c        REL LOCAL
L1...............int  P:0030C3          buddy_chk_c       REL LOCAL
L10..............int  P:000F27          BEPsetvars_c      REL LOCAL
L10..............int  P:00185D          BEPvidout_c       REL LOCAL
L10..............int  P:001ED6          BEPcom_c          REL LOCAL
L10..............int  P:0024E5          BEPframe_c        REL LOCAL
L10..............int  P:002D80          ippshell_c        REL LOCAL
L100.............int  P:001C41          BEPvidout_c       REL LOCAL
L100.............int  P:00234A          BEPcom_c          REL LOCAL
L101.............int  P:002360          BEPcom_c          REL LOCAL
L102.............int  P:002390          BEPcom_c          REL LOCAL
L103.............int  P:002A91          BEPframe_c        REL LOCAL
L104.............int  P:0023D9          BEPcom_c          REL LOCAL
L104.............int  P:002A7F          BEPframe_c        REL LOCAL
L105.............int  P:001C55          BEPvidout_c       REL LOCAL
L105.............int  P:0023D4          BEPcom_c          REL LOCAL
L106.............int  P:001C9A          BEPvidout_c       REL LOCAL
L106.............int  P:002A72          BEPframe_c        REL LOCAL
L107.............int  P:001C7E          BEPvidout_c       REL LOCAL
L107.............int  P:002413          BEPcom_c          REL LOCAL
L107.............int  P:002A9D          BEPframe_c        REL LOCAL
L108.............int  P:001C96          BEPvidout_c       REL LOCAL
L108.............int  P:002AAF          BEPframe_c        REL LOCAL
L109.............int  P:002452          BEPcom_c          REL LOCAL
L109.............int  P:002AAB          BEPframe_c        REL LOCAL
L10_ab...........int  P:0031E5          sEc_longdiv709    REL LOCAL
L10_ba...........int  P:003195          sEc_longdiv709    REL LOCAL
L11..............int  P:000AE8          BEPmain_c         REL LOCAL
L11..............int  P:001205          BEPvidin_c        REL LOCAL
L11..............int  P:002D56          ippshell_c        REL LOCAL
L110.............int  P:001C9F          BEPvidout_c       REL LOCAL
L111.............int  P:001CC2          BEPvidout_c       REL LOCAL
L111.............int  P:002497          BEPcom_c          REL LOCAL
L113.............int  P:002BB4          BEPframe_c        REL LOCAL
L114.............int  P:002B9C          BEPframe_c        REL LOCAL
L115.............int  P:001BFF          BEPvidout_c       REL LOCAL
L115.............int  P:002B0A          BEPframe_c        REL LOCAL
L116.............int  P:001D29          BEPvidout_c       REL LOCAL
L117.............int  P:002B94          BEPframe_c        REL LOCAL
L12...............int  P:001866          BEPvidout_c       REL LOCAL
L12...............int  P:002D80          ippshell_c        REL LOCAL
L120..............int  P:001CFE          BEPvidout_c       REL LOCAL
L120..............int  P:002AE5          BEPframe_c        REL LOCAL
L122..............int  P:001BEB          BEPvidout_c       REL LOCAL
L122..............int  P:002BC4          BEPframe_c        REL LOCAL
L123..............int  P:001BE2          BEPvidout_c       REL LOCAL
L124..............int  P:001CDE          BEPvidout_c       REL LOCAL
```

L124	int	P:002A97	BEPframe_c	REL LOCAL	L23	int	P:001F41	BEPcom_c	REL LOCAL
L125	int	P:001C77	BEPvidout_c	REL LOCAL	L23	int	P:002557	BEPframe_c	REL LOCAL
L125	int	P:002AB1	BEPframe_c	REL LOCAL	L23	int	P:002E12	ippshell_c	REL LOCAL
L126	int	P:001C92	BEPvidout_c	REL LOCAL	L24	int	P:000B2B	BEPmain_c	REL LOCAL
L126	int	P:002B0C	BEPframe_c	REL LOCAL	L24	int	P:001F84	BEPcom_c	REL LOCAL
L127	int	P:001C67	BEPvidout_c	REL LOCAL	L24	int	P:002EB6	ippshell_c	REL LOCAL
L127	int	P:002BB1	BEPframe_c	REL LOCAL	L25	int	P:0012D6	BEPvidin_c	REL LOCAL
L128	int	P:002AFF	BEPframe_c	REL LOCAL	L25	int	P:0025E5	BEPframe_c	REL LOCAL
L129	int	P:001D88	BEPvidout_c	REL LOCAL	L25	int	P:002E74	ippshell_c	REL LOCAL
L129	int	P:002B21	BEPframe_c	REL LOCAL	L26	int	P:000B19	BEPmain_c	REL LOCAL
L13	int	P:000AEB	BEPmain_c	REL LOCAL	L26	int	P:001054	BEPsetvars_c	REL LOCAL
L13	int	P:002511	BEPframe_c	REL LOCAL	L26	int	P:0018EF	BEPvidout_c	REL LOCAL
L13	int	P:002D7E	ippshell_c	REL LOCAL	L26	int	P:002E82	ippshell_c	REL LOCAL
L130	int	P:001DF4	BEPvidout_c	REL LOCAL	L27	int	P:000B2E	BEPmain_c	REL LOCAL
L130	int	P:002B5C	BEPframe_c	REL LOCAL	L27	int	P:001906	BEPvidout_c	REL LOCAL
L131	int	P:001DA6	BEPvidout_c	REL LOCAL	L27	int	P:002F3F	ippshell_c	REL LOCAL
L133	int	P:001DD4	BEPvidout_c	REL LOCAL	L28	int	P:000B21	BEPmain_c	REL LOCAL
L135	int	P:001DCC	BEPvidout_c	REL LOCAL	L28	int	P:000F58	BEPsetvars_c	REL LOCAL
L136	int	P:001DE8	BEPvidout_c	REL LOCAL	L28	int	P:0012EF	BEPvidin_c	REL LOCAL
L14	int	P:00187D	BEPvidout_c	REL LOCAL	L28	int	P:001908	BEPvidout_c	REL LOCAL
L14	int	P:00251B	BEPframe_c	REL LOCAL	L28	int	P:0025C7	BEPframe_c	REL LOCAL
L14	int	P:002D6C	ippshell_c	REL LOCAL	L28	int	P:002EFF	ippshell_c	REL LOCAL
L15	int	P:000AE4	BEPmain_c	REL LOCAL	L29	int	P:000F53	BEPsetvars_c	REL LOCAL
L15	int	P:001247	BEPvidin_c	REL LOCAL	L29	int	P:001FED	BEPcom_c	REL LOCAL
L15	int	P:00189E	BEPvidout_c	REL LOCAL	L29	int	P:0025D9	BEPframe_c	REL LOCAL
L15	int	P:001F33	BEPcom_c	REL LOCAL	L29	int	P:002F0D	ippshell_c	REL LOCAL
L15	int	P:002D7E	ippshell_c	REL LOCAL	L2_ab	int	P:0031A9	sEc_longdiv709	REL LOCAL
L16	int	P:00125E	BEPvidin_c	REL LOCAL	L2_ba	int	P:003159	sEc_longdiv709	REL LOCAL
L16	int	P:002D79	ippshell_c	REL LOCAL	L3	int	P:000ECA	BEPsetvars_c	REL LOCAL
L17	int	P:001067	BEPsetvars_c	REL LOCAL	L3	int	P:00119A	BEPvidin_c	REL LOCAL
L17	int	P:001269	BEPvidin_c	REL LOCAL	L3	int	P:00181F	BEPvidout_c	REL LOCAL
L17	int	P:0018A1	BEPvidout_c	REL LOCAL	L3	int	P:001E2D	BEPcom_c	REL LOCAL
L17	int	P:00258D	BEPframe_c	REL LOCAL	L3	int	P:002CF6	ippshell_c	REL LOCAL
L17	int	P:002D7B	ippshell_c	REL LOCAL	L3	int	P:003114	buddy_chk_c	REL LOCAL
L18	int	P:000FC2	BEPsetvars_c	REL LOCAL	L30	int	P:002F3D	ippshell_c	REL LOCAL
L18	int	P:001251	BEPvidin_c	REL LOCAL	L31	int	P:00106B	BEPsetvars_c	REL LOCAL
L18	int	P:0018C7	BEPvidout_c	REL LOCAL	L31	int	P:00130E	BEPvidin_c	REL LOCAL
L19	int	P:000FFF	BEPsetvars_c	REL LOCAL	L31	int	P:00180D	BEPvidout_c	REL LOCAL
L19	int	P:0018B7	BEPvidout_c	REL LOCAL	L31	int	P:002632	BEPframe_c	REL LOCAL
L19	int	P:001F4E	BEPcom_c	REL LOCAL	L31	int	P:002F78	ippshell_c	REL LOCAL
L19	int	P:002DC8	ippshell_c	REL LOCAL	L32	int	P:00131F	BEPvidin_c	REL LOCAL
L2	int	P:000AE6	BEPmain_c	REL LOCAL	L32	int	P:001914	BEPvidout_c	REL LOCAL
L2	int	P:000EC9	BEPsetvars_c	REL LOCAL	L32	int	P:0026BD	BEPframe_c	REL LOCAL
L2	int	P:001915	BEPvidout_c	REL LOCAL	L32	int	P:002F6A	ippshell_c	REL LOCAL
L2	int	P:001E20	BEPcom_c	REL LOCAL	L33	int	P:000B4C	BEPmain_c	REL LOCAL
L2	int	P:002D01	ippshell_c	REL LOCAL	L33	int	P:00106E	BEPsetvars_c	REL LOCAL
L2	int	P:0030C3	buddy_chk_c	REL LOCAL	L33	int	P:0018C4	BEPvidout_c	REL LOCAL
L20	int	P:000B02	BEPmain_c	REL LOCAL	L33	int	P:002011	BEPcom_c	REL LOCAL
L20	int	P:001359	BEPvidin_c	REL LOCAL	L33	int	P:002F6E	ippshell_c	REL LOCAL
L20	int	P:00257D	BEPframe_c	REL LOCAL	L34	int	P:000FFE	BEPsetvars_c	REL LOCAL
L20	int	P:002DBB	ippshell_c	REL LOCAL	L34	int	P:001339	BEPvidin_c	REL LOCAL
L21	int	P:000FEC	BEPsetvars_c	REL LOCAL	L34	int	P:001912	BEPvidout_c	REL LOCAL
L21	int	P:001364	BEPvidin_c	REL LOCAL	L34	int	P:002F76	ippshell_c	REL LOCAL
L21	int	P:001F6C	BEPcom_c	REL LOCAL	L35	int	P:000B4E	BEPmain_c	REL LOCAL
L21	int	P:00256C	BEPframe_c	REL LOCAL	L35	int	P:001066	BEPsetvars_c	REL LOCAL
L21	int	P:002E38	ippshell_c	REL LOCAL	L35	int	P:002663	BEPframe_c	REL LOCAL
L22	int	P:0018CC	BEPvidout_c	REL LOCAL	L36	int	P:001986	BEPvidout_c	REL LOCAL
L22	int	P:002E04	ippshell_c	REL LOCAL					

L23	int	P:00102A	BEPsetvars_c	REL LOCAL	L36	int	P:002665	BEPframe_c	REL LOCAL
L23	int	P:0012B7	BEPvidin_c	REL LOCAL	L37	int	P:000B85	BEPmain_c	REL LOCAL
L23	int	P:0018DF	BEPvidout_c	REL LOCAL	L37	int	P:002037	BEPcom_c	REL LOCAL
					L38	int	P:000B8C	BEPmain_c	REL LOCAL

L38	int	P:0012AB	BEPvidin_c	REL	LOCAL	L53	int	P:002750	BEPframe_c	REL	LOCAL
L39	int	P:000C2A	BEPmain_c	REL	LOCAL	L54	int	P:0020FA	BEPcom_c	REL	LOCAL
L39	int	P:001298	BEPvidin_c	REL	LOCAL	L54	int	P:002759	BEPframe_c	REL	LOCAL
L39	int	P:002041	BEPcom_c	REL	LOCAL	L55	int	P:000CB2	BEPmain_c	REL	LOCAL
L3_ab	int	P:0031B5	sEc_longdiv709	REL	LOCAL	L55	int	P:00144C	BEPvidin_c	REL	LOCAL
L3_ba	int	P:003165	sEc_longdiv709	REL	LOCAL	L55	int	P:001A1D	BEPvidout_c	REL	LOCAL
L4	int	P:002CF2	ippshell_c	REL	LOCAL	L55	int	P:002100	BEPcom_c	REL	LOCAL
L4	int	P:0030FE	buddy_chk_c	REL	LOCAL	L55	int	P:00276F	BEPframe_c	REL	LOCAL
L4	int	P:003133	mcpy_c	REL	LOCAL	L56	int	P:000CC3	BEPmain_c	REL	LOCAL
L40	int	P:000C16	BEPmain_c	REL	LOCAL	L56	int	P:002150	BEPcom_c	REL	LOCAL
L40	int	P:001357	BEPvidin_c	REL	LOCAL	L56	int	P:002778	BEPframe_c	REL	LOCAL
L40	int	P:00266C	BEPframe_c	REL	LOCAL	L57	int	P:000CC2	BEPmain_c	REL	LOCAL
L41	int	P:000C27	BEPmain_c	REL	LOCAL	L57	int	P:001A3F	BEPvidout_c	REL	LOCAL
L41	int	P:001333	BEPvidin_c	REL	LOCAL	L57	int	P:002163	BEPcom_c	REL	LOCAL
L41	int	P:002025	BEPcom_c	REL	LOCAL	L57	int	P:00278E	BEPframe_c	REL	LOCAL
L41	int	P:002693	BEPframe_c	REL	LOCAL	L58	int	P:001419	BEPvidin_c	REL	LOCAL
L42	int	P:002650	BEPframe_c	REL	LOCAL	L58	int	P:001A41	BEPvidout_c	REL	LOCAL
L43	int	P:000C50	BEPmain_c	REL	LOCAL	L58	int	P:00216F	BEPcom_c	REL	LOCAL
L43	int	P:001102	BEPsetvars_c	REL	LOCAL	L58	int	P:002797	BEPframe_c	REL	LOCAL
L43	int	P:0013CD	BEPvidin_c	REL	LOCAL	L59	int	P:00153D	BEPvidin_c	REL	LOCAL
L43	int	P:001A00	BEPvidout_c	REL	LOCAL	L59	int	P:002175	BEPcom_c	REL	LOCAL
L43	int	P:002003	BEPcom_c	REL	LOCAL	L5_ab	int	P:0031C9	sEc_longdiv709	REL	LOCAL
L43	int	P:002681	BEPframe_c	REL	LOCAL	L5_ba	int	P:003179	sEc_longdiv709	REL	LOCAL
L44	int	P:0010D5	BEPsetvars_c	REL	LOCAL	L6	int	P:000AE2	BEPmain_c	REL	LOCAL
L44	int	P:0013CB	BEPvidin_c	REL	LOCAL	L6	int	P:0011BF	BEPvidin_c	REL	LOCAL
L44	int	P:0019D0	BEPvidout_c	REL	LOCAL	L6	int	P:001863	BEPvidout_c	REL	LOCAL
L44	int	P:002073	BEPcom_c	REL	LOCAL	L6	int	P:001E64	BEPcom_c	REL	LOCAL
L44	int	P:002680	BEPframe_c	REL	LOCAL	L6	int	P:0024C5	BEPframe_c	REL	LOCAL
L45	int	P:001115	BEPsetvars_c	REL	LOCAL	L6	int	P:002D01	ippshell_c	REL	LOCAL
L45	int	P:0019AC	BEPvidout_c	REL	LOCAL	L6	int	P:003135	mcpy_c	REL	LOCAL
L46	int	P:000C79	BEPmain_c	REL	LOCAL	L60	int	P:00146B	BEPvidin_c	REL	LOCAL
L46	int	P:001114	BEPsetvars_c	REL	LOCAL	L60	int	Y:003052	BEPcom_c	REL	LOCAL
L46	int	P:0019C2	BEPvidout_c	REL	LOCAL	L61	int	P:00217B	BEPcom_c	REL	LOCAL
L46	int	P:002197	BEPcom_c	REL	LOCAL	L61	int	P:0027FD	BEPframe_c	REL	LOCAL
L46	int	P:00262B	BEPframe_c	REL	LOCAL	L62	int	P:000CF6	BEPmain_c	REL	LOCAL
L47	int	P:000C66	BEPmain_c	REL	LOCAL	L62	int	P:0014B6	BEPvidin_c	REL	LOCAL
L47	int	P:00145A	BEPvidin_c	REL	LOCAL	L62	int	P:001A58	BEPvidout_c	REL	LOCAL
L47	int	P:0020B2	BEPcom_c	REL	LOCAL	L62	int	P:00217E	BEPcom_c	REL	LOCAL
L47	int	P:0026B1	BEPframe_c	REL	LOCAL	L62	int	P:0027CE	BEPframe_c	REL	LOCAL
L48	int	P:00142B	BEPvidin_c	REL	LOCAL	L63	int	P:000D06	BEPmain_c	REL	LOCAL
L48	int	P:0020BA	BEPcom_c	REL	LOCAL	L63	int	P:0014AE	BEPvidin_c	REL	LOCAL
L49	int	P:000C60	BEPmain_c	REL	LOCAL	L63	int	P:001A5C	BEPvidout_c	REL	LOCAL
L49	int	P:001433	BEPvidin_c	REL	LOCAL	L63	int	P:0020DD	BEPcom_c	REL	LOCAL
L49	int	P:002712	BEPframe_c	REL	LOCAL	L63	int	P:0027D0	BEPframe_c	REL	LOCAL
L4_ab	int	P:0031D9	sEc_longdiv709	REL	LOCAL	L64	int	P:000D05	BEPmain_c	REL	LOCAL
L4_ba	int	P:003189	sEc_longdiv709	REL	LOCAL	L64	int	P:0020F3	BEPcom_c	REL	LOCAL
L5	int	P:001836	BEPvidout_c	REL	LOCAL	L64	int	P:0027F0	BEPframe_c	REL	LOCAL
L5	int	P:001E62	BEPcom_c	REL	LOCAL	L65	int	P:00151C	BEPvidin_c	REL	LOCAL
L5	int	P:002CF6	ippshell_c	REL	LOCAL	L65	int	P:001A67	BEPvidout_c	REL	LOCAL
L5	int	P:003109	buddy_chk_c	REL	LOCAL	L65	int	P:002192	BEPcom_c	REL	LOCAL
L50	int	P:0020C2	BEPcom_c	REL	LOCAL	L65	int	P:0027F5	BEPframe_c	REL	LOCAL
L50	int	P:002895	BEPframe_c	REL	LOCAL	L66	int	P:00150C	BEPvidin_c	REL	LOCAL
L51	int	P:0019F5	BEPvidout_c	REL	LOCAL	L66	int	P:001AC4	BEPvidout_c	REL	LOCAL
L51	int	P:0020C8	BEPcom_c	REL	LOCAL	L66	int	P:00286D	BEPframe_c	REL	LOCAL
L51	int	P:002731	BEPframe_c	REL	LOCAL	L67	int	P:002395	BEPcom_c	REL	LOCAL
L52	int	P:00143F	BEPvidin_c	REL	LOCAL						

L52	int	P:0019ED	BEPvidout_c	REL	LOCAL
L52	int	P:0020CE	BEPcom_c	REL	LOCAL
L52	int	P:00273A	BEPframe_c	REL	LOCAL
L53	int	P:00144C	BEPvidin_c	REL	LOCAL
L53	int	P:0020E4	BEPcom_c	REL	LOCAL

L67	int	P:00281C	BEPframe_c	REL	LOCAL
L68	int	P:001534	BEPvidin_c	REL	LOCAL
L68	int	P:002821	BEPframe_c	REL	LOCAL
L69	int	P:000D39	BEPmain_c	REL	LOCAL
L69	int	P:001449	BEPvidin_c	REL	LOCAL
L69	int	P:0021E3	BEPcom_c	REL	LOCAL

```
L69.............int   P:002851          BEPframe_c        REL LOCAL
L6_ab...........int   P:0031C4          sEc_longdiv709    REL LOCAL
L6_ba...........int   P:003174          sEc_longdiv709    REL LOCAL
L7..............int   P:002D42          ippshell_c        REL LOCAL
L70.............int   P:000D4A          BEPmain_c         REL LOCAL
L70.............int   P:00146D          BEPvidin_c        REL LOCAL
L70.............int   P:001A99          BEPvidout_c       REL LOCAL
L70.............int   P:002202          BEPcom_c          REL LOCAL
L71.............int   P:000D49          BEPmain_c         REL LOCAL
L71.............int   P:0014C2          BEPvidin_c        REL LOCAL
L71.............int   P:001A49          BEPvidout_c       REL LOCAL
L71.............int   P:00220B          BEPcom_c          REL LOCAL
L71.............int   P:0027BA          BEPframe_c        REL LOCAL
L72.............int   P:00221B          BEPcom_c          REL LOCAL
L73.............int   P:000D7E          BEPmain_c         REL LOCAL
L73.............int   P:00166F          BEPvidin_c        REL LOCAL
L73.............int   P:00270B          BEPframe_c        REL LOCAL
L74.............int   P:000DC0          BEPmain_c         REL LOCAL
L74.............int   P:0019FD          BEPvidout_c       REL LOCAL
L74.............int   P:00280D          BEPframe_c        REL LOCAL
L75.............int   P:000D95          BEPmain_c         REL LOCAL
L75.............int   P:0019BD          BEPvidout_c       REL LOCAL
L75.............int   P:002240          BEPcom_c          REL LOCAL
L75.............int   P:002887          BEPframe_c        REL LOCAL
L76.............int   P:000DA4          BEPmain_c         REL LOCAL
L76.............int   P:001666          BEPvidin_c        REL LOCAL
L76.............int   P:0019EF          BEPvidout_c       REL LOCAL
L77.............int   P:000DB3          BEPmain_c         REL LOCAL
L77.............int   P:001A2E          BEPvidout_c       REL LOCAL
L77.............int   P:002224          BEPcom_c          REL LOCAL
L77.............int   P:0028EA          BEPframe_c        REL LOCAL
L78.............int   P:001AD3          BEPvidout_c       REL LOCAL
L78.............int   P:002247          BEPcom_c          REL LOCAL
L78.............int   P:002924          BEPframe_c        REL LOCAL
L79.............int   P:001ABD          BEPvidout_c       REL LOCAL
L79.............int   P:0028E3          BEPframe_c        REL LOCAL
L7_ab...........int   P:0031D3          sEc_longdiv709    REL LOCAL
L7_ba...........int   P:003183          sEc_longdiv709    REL LOCAL
L8..............int   P:000EEF          BEPsetvars_c      REL LOCAL
L8..............int   P:00120D          BEPvidin_c        REL LOCAL
L8..............int   P:001846          BEPvidout_c       REL LOCAL
L8..............int   P:001E95          BEPcom_c          REL LOCAL
L8..............int   P:002D38          ippshell_c        REL LOCAL
L80.............int   P:000D8D          BEPmain_c         REL LOCAL
L80.............int   P:00165C          BEPvidin_c        REL LOCAL
L80.............int   P:00291D          BEPframe_c        REL LOCAL
L81.............int   P:000DE5          BEPmain_c         REL LOCAL
L81.............int   P:001631          BEPvidin_c        REL LOCAL
L81.............int   P:001BED          BEPvidout_c       REL LOCAL
L82.............int   P:0015D7          BEPvidin_c        REL LOCAL
L82.............int   P:001B3A          BEPvidout_c       REL LOCAL
L82.............int   P:002282          BEPcom_c          REL LOCAL
L82.............int   P:00297D          BEPframe_c        REL LOCAL
L83.............int   P:000E32          BEPmain_c         REL LOCAL
```



```
L83.............int   P:001B62          BEPvidout_c       REL LOCAL
L83.............int   P:002983          BEPframe_c        REL LOCAL
L84.............int   P:000E2B          BEPmain_c         REL LOCAL
L84.............int   P:0022D1          BEPcom_c          REL LOCAL
L85.............int   P:000E2E          BEPmain_c         REL LOCAL
L85.............int   P:001B6B          BEPvidout_c       REL LOCAL
L85.............int   P:0022E0          BEPcom_c          REL LOCAL
```

```
L86.............int   P:000E31          BEPmain_c         REL LOCAL
L86.............int   P:0015B1          BEPvidin_c        REL LOCAL
L86.............int   P:001B96          BEPvidout_c       REL LOCAL
L86.............int   P:0022EF          BEPcom_c          REL LOCAL
L87.............int   P:001582          BEPvidin_c        REL LOCAL
L87.............int   P:0022FE          BEPcom_c          REL LOCAL
L87.............int   P:0029B9          BEPframe_c        REL LOCAL
L88.............int   P:0016B1          BEPvidin_c        REL LOCAL
L88.............int   P:001BB6          BEPvidout_c       REL LOCAL
L88.............int   P:002317          BEPcom_c          REL LOCAL
L89.............int   P:000E23          BEPmain_c         REL LOCAL
L89.............int   P:001BDB          BEPvidout_c       REL LOCAL
L89.............int   P:002324          BEPcom_c          REL LOCAL
L8_ab...........int   P:0031D0          sEc_longdiv709    REL LOCAL
L8_ba...........int   P:003180          sEc_longdiv709    REL LOCAL
L9..............int   P:00185B          BEPvidout_c       REL LOCAL
L9..............int   P:0024D6          BEPframe_c        REL LOCAL
L9..............int   P:002D42          ippshell_c        REL LOCAL
L90.............int   P:00233D          BEPcom_c          REL LOCAL
L91.............int   P:001623          BEPvidin_c        REL LOCAL
L91.............int   P:002356          BEPcom_c          REL LOCAL
L91.............int   P:002A0C          BEPframe_c        REL LOCAL
L92.............int   P:000E8A          BEPmain_c         REL LOCAL
L92.............int   P:001595          BEPvidin_c        REL LOCAL
L92.............int   P:002373          BEPcom_c          REL LOCAL
L92.............int   P:002A28          BEPframe_c        REL LOCAL
L93.............int   P:001B30          BEPvidout_c       REL LOCAL
L93.............int   Y:003067          BEPcom_c          REL LOCAL
L93.............int   P:002A27          BEPframe_c        REL LOCAL
L94.............int   P:001720          BEPvidin_c        REL LOCAL
L94.............int   P:001CE1          BEPvidout_c       REL LOCAL
L94.............int   P:0029EA          BEPframe_c        REL LOCAL
L95.............int   P:001C11          BEPvidout_c       REL LOCAL
L95.............int   P:0022A6          BEPcom_c          REL LOCAL
L95.............int   P:0029E9          BEPframe_c        REL LOCAL
L96.............int   P:000EAC          BEPmain_c         REL LOCAL
L96.............int   P:0029CF          BEPframe_c        REL LOCAL
L97.............int   P:000EA6          BEPmain_c         REL LOCAL
L97.............int   P:0017A1          BEPvidin_c        REL LOCAL
L97.............int   P:001C33          BEPvidout_c       REL LOCAL
L97.............int   P:002200          BEPcom_c          REL LOCAL
L97.............int   P:002A22          BEPframe_c        REL LOCAL
L98.............int   P:00230B          BEPcom_c          REL LOCAL
L99.............int   P:00175D          BEPvidin_c        REL LOCAL
L99.............int   P:002331          BEPcom_c          REL LOCAL
L99.............int   P:002A99          BEPframe_c        REL LOCAL
L9_ab...........int   P:0031DD          sEc_longdiv709    REL LOCAL
L9_ba...........int   P:00318D          sEc_longdiv709    REL LOCAL
LSIZE...........int     0030C1          GLOBAL            ABS GLOBAL
M_BCR...........int     00FFFE          ipp_driver        ABS LOCAL
M_BCR...........int     00FFFE          RTC_handler       ABS LOCAL
M_CD............int     000FFF          ipp_driver        ABS LOCAL
M_CD............int     000FFF          RTC_handler       ABS LOCAL
```



```
M_COD...........int     00000C          ipp_driver        ABS LOCAL
M_COD...........int     00000C          RTC_handler       ABS LOCAL
M_CRA...........int     00FFEC          ipp_driver        ABS LOCAL
M_CRA...........int     00FFEC          RTC_handler       ABS LOCAL
M_CRB...........int     00FFED          ipp_driver        ABS LOCAL
M_CRB...........int     00FFED          RTC_handler       ABS LOCAL
M_DC............int     001F00          ipp_driver        ABS LOCAL
M_DC............int     001F00          RTC_handler       ABS LOCAL
```

M_DMA	int	000007	ipp_driver	ABS LOCAL	M_IDLE	int	000003	RTC_handler	ABS LOCAL
M_DMA	int	000007	RTC_handler	ABS LOCAL	M_IF	int	000002	ipp_driver	ABS LOCAL
M_FE	int	000006	ipp_driver	ABS LOCAL	M_IF	int	000002	RTC_handler	ABS LOCAL
M_FE	int	000006	RTC_handler	ABS LOCAL	M_IF0	int	000000	ipp_driver	ABS LOCAL
M_FSL	int	000008	ipp_driver	ABS LOCAL	M_IF0	int	000000	RTC_handler	ABS LOCAL
M_FSL	int	000008	RTC_handler	ABS LOCAL	M_IF1	int	000001	ipp_driver	ABS LOCAL
M_GCK	int	00000A	ipp_driver	ABS LOCAL	M_IF1	int	000001	RTC_handler	ABS LOCAL
M_GCK	int	00000A	RTC_handler	ABS LOCAL	M_ILIE	int	00000A	ipp_driver	ABS LOCAL
M_HCIE	int	000002	ipp_driver	ABS LOCAL	M_ILIE	int	00000A	RTC_handler	ABS LOCAL
M_HCIE	int	000002	RTC_handler	ABS LOCAL	M_IPR	int	00FFFF	ipp_driver	ABS LOCAL
M_HCP	int	000002	ipp_driver	ABS LOCAL	M_IPR	int	00FFFF	RTC_handler	ABS LOCAL
M_HCP	int	000002	RTC_handler	ABS LOCAL	M_MOD	int	00000B	ipp_driver	ABS LOCAL
M_HCR	int	00FFE8	ipp_driver	ABS LOCAL	M_MOD	int	00000B	RTC_handler	ABS LOCAL
M_HCR	int	00FFE8	RTC_handler	ABS LOCAL	M_OF	int	000002	ipp_driver	ABS LOCAL
M_HF	int	000018	ipp_driver	ABS LOCAL	M_OF	int	000002	RTC_handler	ABS LOCAL
M_HF	int	000018	RTC_handler	ABS LOCAL	M_OF0	int	000000	ipp_driver	ABS LOCAL
M_HF0	int	000003	ipp_driver	ABS LOCAL	M_OF0	int	000000	RTC_handler	ABS LOCAL
M_HF0	int	000003	RTC_handler	ABS LOCAL	M_OF1	int	000001	ipp_driver	ABS LOCAL
M_HF1	int	000004	ipp_driver	ABS LOCAL	M_OF1	int	000001	RTC_handler	ABS LOCAL
M_HF1	int	000004	RTC_handler	ABS LOCAL	M_OR	int	000004	ipp_driver	ABS LOCAL
M_HF2	int	000003	ipp_driver	ABS LOCAL	M_OR	int	000004	RTC_handler	ABS LOCAL
M_HF2	int	000003	RTC_handler	ABS LOCAL	M_PBC	int	00FFE0	ipp_driver	ABS LOCAL
M_HF3	int	000004	ipp_driver	ABS LOCAL	M_PBC	int	00FFE0	RTC_handler	ABS LOCAL
M_HF3	int	000004	RTC_handler	ABS LOCAL	M_PBD	int	00FFE4	ipp_driver	ABS LOCAL
M_HPL	int	000C00	ipp_driver	ABS LOCAL	M_PBD	int	00FFE4	RTC_handler	ABS LOCAL
M_HPL	int	000C00	RTC_handler	ABS LOCAL	M_PBDDR	int	00FFE2	ipp_driver	ABS LOCAL
M_HPL0	int	00000A	ipp_driver	ABS LOCAL	M_PBDDR	int	00FFE2	RTC_handler	ABS LOCAL
M_HPL0	int	00000A	RTC_handler	ABS LOCAL	M_PCC	int	00FFE1	ipp_driver	ABS LOCAL
M_HPL1	int	00000B	ipp_driver	ABS LOCAL	M_PCC	int	00FFE1	RTC_handler	ABS LOCAL
M_HPL1	int	00000B	RTC_handler	ABS LOCAL	M_PCD	int	00FFE5	ipp_driver	ABS LOCAL
M_HRDF	int	000000	ipp_driver	ABS LOCAL	M_PCD	int	00FFE5	RTC_handler	ABS LOCAL
M_HRDF	int	000000	RTC_handler	ABS LOCAL	M_PCDDR	int	00FFE3	ipp_driver	ABS LOCAL
M_HRIE	int	000000	ipp_driver	ABS LOCAL	M_PCDDR	int	00FFE3	RTC_handler	ABS LOCAL
M_HRIE	int	000000	RTC_handler	ABS LOCAL	M_PE	int	000005	ipp_driver	ABS LOCAL
M_HRX	int	00FFEB	ipp_driver	ABS LOCAL	M_PE	int	000005	RTC_handler	ABS LOCAL
M_HRX	int	00FFEB	RTC_handler	ABS LOCAL	M_PM	int	0000FF	ipp_driver	ABS LOCAL
M_HSR	int	00FFE9	ipp_driver	ABS LOCAL	M_PM	int	0000FF	RTC_handler	ABS LOCAL
M_HSR	int	00FFE9	RTC_handler	ABS LOCAL	M_PSR	int	00000F	ipp_driver	ABS LOCAL
M_HTDE	int	000001	ipp_driver	ABS LOCAL	M_PSR	int	00000F	RTC_handler	ABS LOCAL
M_HTDE	int	000001	RTC_handler	ABS LOCAL	M_R8	int	000007	ipp_driver	ABS LOCAL
M_HTIE	int	000001	ipp_driver	ABS LOCAL	M_R8	int	000007	RTC_handler	ABS LOCAL
M_HTIE	int	000001	RTC_handler	ABS LOCAL	M_RCM	int	00000E	ipp_driver	ABS LOCAL
M_HTX	int	00FFEB	ipp_driver	ABS LOCAL	M_RCM	int	00000E	RTC_handler	ABS LOCAL
M_HTX	int	00FFEB	RTC_handler	ABS LOCAL	M_RDF	int	000007	ipp_driver	ABS LOCAL
M_IAL	int	000007	ipp_driver	ABS LOCAL	M_RDF	int	000007	RTC_handler	ABS LOCAL
M_IAL	int	000007	RTC_handler	ABS LOCAL	M_RDRF	int	000002	ipp_driver	ABS LOCAL
M_IAL0	int	000000	ipp_driver	ABS LOCAL	M_RDRF	int	000002	RTC_handler	ABS LOCAL
M_IAL0	int	000000	RTC_handler	ABS LOCAL	M_RE	int	000008	ipp_driver	ABS LOCAL
M_IAL1	int	000001	ipp_driver	ABS LOCAL	M_RE	int	000008	RTC_handler	ABS LOCAL
M_IAL1	int	000001	RTC_handler	ABS LOCAL	M_RFS	int	000003	ipp_driver	ABS LOCAL
M_IAL2	int	000002	ipp_driver	ABS LOCAL	M_RFS	int	000003	RTC_handler	ABS LOCAL
M_IAL2	int	000002	RTC_handler	ABS LOCAL					

M_IBL	int	000038	ipp_driver	ABS LOCAL	M_RIE	int	00000B	ipp_driver	ABS LOCAL
M_IBL	int	000038	RTC_handler	ABS LOCAL	M_RIE	int	00000B	RTC_handler	ABS LOCAL
M_IBL0	int	000003	ipp_driver	ABS LOCAL	M_ROE	int	000005	ipp_driver	ABS LOCAL
M_IBL0	int	000003	RTC_handler	ABS LOCAL	M_ROE	int	000005	RTC_handler	ABS LOCAL
M_IBL1	int	000004	ipp_driver	ABS LOCAL	M_RWI	int	000006	ipp_driver	ABS LOCAL
M_IBL1	int	000004	RTC_handler	ABS LOCAL	M_RWI	int	000006	RTC_handler	ABS LOCAL
M_IBL2	int	000005	ipp_driver	ABS LOCAL	M_RX	int	00FFEF	ipp_driver	ABS LOCAL
M_IBL2	int	000005	RTC_handler	ABS LOCAL	M_RX	int	00FFEF	RTC_handler	ABS LOCAL
M_IDLE	int	000003	ipp_driver	ABS LOCAL	M_SBK	int	000004	ipp_driver	ABS LOCAL
					M_SBK	int	000004	RTC_handler	ABS LOCAL

```
M_SCCR..........int     00FFF2          ipp_driver      ABS LOCAL
M_SCCR..........int     00FFF2          RTC_handler     ABS LOCAL
M_SCD...........int     00001C          ipp_driver      ABS LOCAL
M_SCD...........int     00001C          RTC_handler     ABS LOCAL
M_SCD0..........int     000002          ipp_driver      ABS LOCAL
M_SCD0..........int     000002          RTC_handler     ABS LOCAL
M_SCD1..........int     000003          ipp_driver      ABS LOCAL
M_SCD1..........int     000003          RTC_handler     ABS LOCAL
M_SCD2..........int     000004          ipp_driver      ABS LOCAL
M_SCD2..........int     000004          RTC_handler     ABS LOCAL
M_SCKD..........int     000005          ipp_driver      ABS LOCAL
M_SCKD..........int     000005          RTC_handler     ABS LOCAL
M_SCL...........int     00C000          ipp_driver      ABS LOCAL
M_SCL...........int     00C000          RTC_handler     ABS LOCAL
M_SCL0..........int     00000E          ipp_driver      ABS LOCAL
M_SCL0..........int     00000E          RTC_handler     ABS LOCAL
M_SCL1..........int     00000F          ipp_driver      ABS LOCAL
M_SCL1..........int     00000F          RTC_handler     ABS LOCAL
M_SCP...........int     00000D          ipp_driver      ABS LOCAL
M_SCP...........int     00000D          RTC_handler     ABS LOCAL
M_SCR...........int     00FFF0          ipp_driver      ABS LOCAL
M_SCR...........int     00FFF0          RTC_handler     ABS LOCAL
M_SR............int     00FFEE          ipp_driver      ABS LOCAL
M_SR............int     00FFEE          RTC_handler     ABS LOCAL
M_SRE...........int     00000D          ipp_driver      ABS LOCAL
M_SRE...........int     00000D          RTC_handler     ABS LOCAL
M_SRIE..........int     00000F          ipp_driver      ABS LOCAL
M_SRIE..........int     00000F          RTC_handler     ABS LOCAL
M_SRXH..........int     00FFF6          ipp_driver      ABS LOCAL
M_SRXH..........int     00FFF6          RTC_handler     ABS LOCAL
M_SRXL..........int     00FFF4          ipp_driver      ABS LOCAL
M_SRXL..........int     00FFF4          RTC_handler     ABS LOCAL
M_SRXM..........int     00FFF5          ipp_driver      ABS LOCAL
M_SRXM..........int     00FFF5          RTC_handler     ABS LOCAL
M_SSL...........int     003000          ipp_driver      ABS LOCAL
M_SSL...........int     003000          RTC_handler     ABS LOCAL
M_SSL0..........int     00000C          ipp_driver      ABS LOCAL
M_SSL0..........int     00000C          RTC_handler     ABS LOCAL
M_SSL1..........int     00000D          ipp_driver      ABS LOCAL
M_SSL1..........int     00000D          RTC_handler     ABS LOCAL
M_SSR...........int     00FFF1          ipp_driver      ABS LOCAL
M_SSR...........int     00FFF1          RTC_handler     ABS LOCAL
M_STE...........int     00000C          ipp_driver      ABS LOCAL
M_STE...........int     00000C          RTC_handler     ABS LOCAL
M_STIE..........int     00000E          ipp_driver      ABS LOCAL
M_STIE..........int     00000E          RTC_handler     ABS LOCAL
M_STXA..........int     00FFF3          ipp_driver      ABS LOCAL
M_STXA..........int     00FFF3          RTC_handler     ABS LOCAL
M_STXH..........int     00FFF6          ipp_driver      ABS LOCAL
M_STXH..........int     00FFF6          RTC_handler     ABS LOCAL
M_TDRE..........int     000001          RTC_handler     ABS LOCAL
M_TE............int     000009          ipp_driver      ABS LOCAL
M_TE............int     000009          RTC_handler     ABS LOCAL
M_TFS...........int     000002          ipp_driver      ABS LOCAL
M_TFS...........int     000002          RTC_handler     ABS LOCAL
M_TIE...........int     00000C          ipp_driver      ABS LOCAL
M_TIE...........int     00000C          RTC_handler     ABS LOCAL
M_TMIE..........int     00000D          ipp_driver      ABS LOCAL
M_TMIE..........int     00000D          RTC_handler     ABS LOCAL
M_TRNE..........int     000000          ipp_driver      ABS LOCAL
M_TRNE..........int     000000          RTC_handler     ABS LOCAL
M_TSR...........int     00FFEE          ipp_driver      ABS LOCAL
M_TSR...........int     00FFEE          RTC_handler     ABS LOCAL
M_TUE...........int     000004          ipp_driver      ABS LOCAL
M_TUE...........int     000004          RTC_handler     ABS LOCAL
M_TX............int     00FFEF          ipp_driver      ABS LOCAL
M_TX............int     00FFEF          RTC_handler     ABS LOCAL
M_WAKE..........int     000005          ipp_driver      ABS LOCAL
M_WAKE..........int     000005          RTC_handler     ABS LOCAL
M_WDS...........int     000003          ipp_driver      ABS LOCAL
M_WDS...........int     000003          RTC_handler     ABS LOCAL
M_WDS0..........int     000000          ipp_driver      ABS LOCAL
M_WDS0..........int     000000          RTC_handler     ABS LOCAL
M_WDS1..........int     000001          ipp_driver      ABS LOCAL
M_WDS1..........int     000001          RTC_handler     ABS LOCAL
M_WDS2..........int     000002          ipp_driver      ABS LOCAL
M_WDS2..........int     000002          RTC_handler     ABS LOCAL
M_WL............int     006000          ipp_driver      ABS LOCAL
M_WL............int     006000          RTC_handler     ABS LOCAL
M_WL0...........int     00000D          ipp_driver      ABS LOCAL
M_WL0...........int     00000D          RTC_handler     ABS LOCAL
M_WL1...........int     00000E          ipp_driver      ABS LOCAL
M_WL1...........int     00000E          RTC_handler     ABS LOCAL
M_WOMS..........int     000007          ipp_driver      ABS LOCAL
M_WOMS..........int     000007          RTC_handler     ABS LOCAL
NO_MODE.........int     000000          ipp_driver      ABS LOCAL
PSIZE...........int     0031EC          GLOBAL          ABS GLOBAL
REQ_ABT.........int     000003          ipp_driver      ABS LOCAL
REQ_PORT........int     00FFE9          ipp_driver      ABS LOCAL
REQ_TD..........int     000004          ipp_driver      ABS LOCAL
RX_even.........int   P:002FCA          ipp_driver      REL LOCAL
SHORT_JSR_INSTR..int    0D0000          GLOBAL          ABS GLOBAL
SSD1BIT.........int     000005          GLOBAL          ABS GLOBAL
SSD1BIT.........int     000005          ipp_driver      ABS LOCAL
SSD1BIT.........int     000005          RTC_handler     ABS LOCAL
SSD2BIT.........int     000006          GLOBAL          ABS GLOBAL
SSD2BIT.........int     000006          ipp_driver      ABS LOCAL
SSD2BIT.........int     000006          RTC_handler     ABS LOCAL
SSI_RX_INSTR.....int    085DAF          GLOBAL          ABS GLOBAL
```



```
STACK_START.....int     001000          GLOBAL          ABS GLOBAL
TOP_OF_MEMORY....int    007FFF          GLOBAL          ABS GLOBAL
TX_even.........int   P:003006          ipp_driver      REL LOCAL
VIDBUF_BOTTOM....int    000000          GLOBAL          ABS GLOBAL
VIDBUF_LEN......int     001000          GLOBAL          ABS GLOBAL
VIDBUF_TOP......int     001000          GLOBAL          ABS GLOBAL
WAITDIS_IRQFST...int    000000          GLOBAL          ABS GLOBAL
WAITDIS_IRQFST...int    000000          ipp_driver      ABS LOCAL
WAITDIS_IRQFST...int    000000          RTC_handler     ABS LOCAL
WAITDIS_IRQSLW...int    000010          GLOBAL          ABS GLOBAL
WAITDIS_IRQSLW...int    000010          ipp_driver      ABS LOCAL
WAITDIS_IRQSLW...int    000010          RTC_handler     ABS LOCAL
```

WAITENA_IRQFST...int	000002	GLOBAL	ABS GLOBAL
WAITENA_IRQFST...int	000002	ipp_driver	ABS LOCAL
WAITENA_IRQFST...int	000002	RTC_handler	ABS LOCAL
WAITENA_IRQSLW...int	000012	GLOBAL	ABS GLOBAL
WAITENA_IRQSLW...int	000012	ipp_driver	ABS LOCAL
WAITENA_IRQSLW...int	000012	RTC_handler	ABS LOCAL
WAIT_DISABLE.....int	000000	GLOBAL	ABS GLOBAL
WAIT_DISABLE.....int	000000	ipp_driver	ABS LOCAL
WAIT_DISABLE.....int	000000	RTC_handler	ABS LOCAL
WAIT_DISSTAT_BIT.int	000000	GLOBAL	ABS GLOBAL
WAIT_DISSTAT_BIT.int	000000	ipp_driver	ABS LOCAL
WAIT_DISSTAT_BIT.int	000000	RTC_handler	ABS LOCAL
WAIT_ENABLE......int	000002	GLOBAL	ABS GLOBAL
WAIT_ENABLE......int	000002	ipp_driver	ABS LOCAL
WAIT_ENABLE......int	000002	RTC_handler	ABS LOCAL
XSIZE............int	000000	GLOBAL	ABS GLOBAL
YSIZE............int	0030C1	GLOBAL	ABS GLOBAL
abort_RX.........int	P:002FD4	ipp_driver	REL LOCAL
abort_TX.........int	P:003010	ipp_driver	REL LOCAL
dma_state........int	Y:0030AC	ipp_driver	REL LOCAL
exit_RX..........int	P:002FDA	ipp_driver	REL LOCAL
exit_TX..........int	P:003018	ipp_driver	REL LOCAL
getdata..........int	P:002FD1	ipp_driver	REL LOCAL
gppReqDTISR......int	P:002F9F	ipp_driver	REL LOCAL
its_dsp_zero.....int	P:002F94	ipp_driver	REL LOCAL
ldiv_aa..........int	P:003144	sEc_longdiv709	REL GLOBAL
ldiv_ab..........int	P:00319C	sEc_longdiv709	REL GLOBAL
ldiv_ba..........int	P:00314C	sEc_longdiv709	REL GLOBAL
ldiv_bb..........int	P:003148	sEc_longdiv709	REL GLOBAL
m_bcr............int	00FFFE	GLOBAL	ABS GLOBAL
m_cd.............int	000FFF	GLOBAL	ABS GLOBAL
m_cod............int	00000C	GLOBAL	ABS GLOBAL
m_cra............int	00FFEC	GLOBAL	ABS GLOBAL
m_crb............int	00FFED	GLOBAL	ABS GLOBAL
m_dc.............int	001F00	GLOBAL	ABS GLOBAL
m_dma............int	000007	GLOBAL	ABS GLOBAL
m_fe.............int	000006	GLOBAL	ABS GLOBAL
m_fsl............int	000008	GLOBAL	ABS GLOBAL
m_gck............int	00000A	GLOBAL	ABS GLOBAL
m_hcie...........int	000002	GLOBAL	ABS GLOBAL
m_hcp............int	000002	GLOBAL	ABS GLOBAL
m_hcr............int	00FFE8	GLOBAL	ABS GLOBAL
m_hf.............int	000018	GLOBAL	ABS GLOBAL
m_hf0............int	000003	GLOBAL	ABS GLOBAL
m_hf1............int	000004	GLOBAL	ABS GLOBAL
m_hf2............int	000003	GLOBAL	ABS GLOBAL
m_hf3............int	000004	GLOBAL	ABS GLOBAL
m_hpl............int	000C00	GLOBAL	ABS GLOBAL

m_hpl0...........int	00000A	GLOBAL	ABS GLOBAL
m_hpl1...........int	00000B	GLOBAL	ABS GLOBAL
m_hrdf...........int	000000	GLOBAL	ABS GLOBAL
m_hrie...........int	000000	GLOBAL	ABS GLOBAL
m_hrx............int	00FFEB	GLOBAL	ABS GLOBAL
m_hsr............int	00FFE9	GLOBAL	ABS GLOBAL
m_htde...........int	000001	GLOBAL	ABS GLOBAL
m_htie...........int	000001	GLOBAL	ABS GLOBAL
m_htx............int	00FFEB	GLOBAL	ABS GLOBAL
m_ial............int	000007	GLOBAL	ABS GLOBAL
m_ial0...........int	000000	GLOBAL	ABS GLOBAL
m_ial1...........int	000001	GLOBAL	ABS GLOBAL
m_ial2...........int	000002	GLOBAL	ABS GLOBAL
m_ibl............int	000038	GLOBAL	ABS GLOBAL
m_ibl0...........int	000003	GLOBAL	ABS GLOBAL
m_ibl1...........int	000004	GLOBAL	ABS GLOBAL
m_ibl2...........int	000005	GLOBAL	ABS GLOBAL
m_idle...........int	000003	GLOBAL	ABS GLOBAL
m_if.............int	000002	GLOBAL	ABS GLOBAL
m_if0............int	000000	GLOBAL	ABS GLOBAL
m_if1............int	000001	GLOBAL	ABS GLOBAL
m_ilie...........int	00000A	GLOBAL	ABS GLOBAL
m_ipr............int	00FFFF	GLOBAL	ABS GLOBAL
m_mod............int	00000B	GLOBAL	ABS GLOBAL
m_of.............int	000003	GLOBAL	ABS GLOBAL
m_of0............int	000000	GLOBAL	ABS GLOBAL
m_of1............int	000001	GLOBAL	ABS GLOBAL
m_or.............int	000004	GLOBAL	ABS GLOBAL
m_pbc............int	00FFE0	GLOBAL	ABS GLOBAL
m_pbd............int	00FFE4	GLOBAL	ABS GLOBAL
m_pbddr..........int	00FFE2	GLOBAL	ABS GLOBAL
m_pcc............int	00FFE1	GLOBAL	ABS GLOBAL
m_pcd............int	00FFE5	GLOBAL	ABS GLOBAL
m_pcddr..........int	00FFE3	GLOBAL	ABS GLOBAL
m_pe.............int	000005	GLOBAL	ABS GLOBAL
m_pm.............int	0000FF	GLOBAL	ABS GLOBAL
m_psr............int	00000F	GLOBAL	ABS GLOBAL
m_r8.............int	000007	GLOBAL	ABS GLOBAL
m_rcm............int	00000E	GLOBAL	ABS GLOBAL
m_rdf............int	000007	GLOBAL	ABS GLOBAL
m_rdrf...........int	000002	GLOBAL	ABS GLOBAL
m_re.............int	000008	GLOBAL	ABS GLOBAL
m_rfs............int	000003	GLOBAL	ABS GLOBAL
m_rie............int	00000B	GLOBAL	ABS GLOBAL
m_roe............int	000005	GLOBAL	ABS GLOBAL
m_rwi............int	000006	GLOBAL	ABS GLOBAL
m_rx.............int	00FFEF	GLOBAL	ABS GLOBAL
m_sbk............int	000004	GLOBAL	ABS GLOBAL
m_sccr...........int	00FFF2	GLOBAL	ABS GLOBAL
m_scd............int	00001C	GLOBAL	ABS GLOBAL
m_scd0...........int	000002	GLOBAL	ABS GLOBAL
m_scd1...........int	000003	GLOBAL	ABS GLOBAL
m_scd2...........int	000004	GLOBAL	ABS GLOBAL
m_sckd...........int	000005	GLOBAL	ABS GLOBAL
m_scl............int	00C000	GLOBAL	ABS GLOBAL
m_scl0...........int	00000E	GLOBAL	ABS GLOBAL
m_scl1...........int	00000F	GLOBAL	ABS GLOBAL
m_scp............int	00000D	GLOBAL	ABS GLOBAL
m_scr............int	00FFF0	GLOBAL	ABS GLOBAL
m_sr.............int	00FFEE	GLOBAL	ABS GLOBAL

m_sre............int	00000D	GLOBAL	ABS GLOBAL
m_srie...........int	00000F	GLOBAL	ABS GLOBAL
m_srxh...........int	00FFF6	GLOBAL	ABS GLOBAL
m_srxl...........int	00FFF4	GLOBAL	ABS GLOBAL
m_srxm...........int	00FFF5	GLOBAL	ABS GLOBAL
m_ssl............int	003000	GLOBAL	ABS GLOBAL
m_ssl0...........int	00000C	GLOBAL	ABS GLOBAL
m_ssl1...........int	00000D	GLOBAL	ABS GLOBAL
m_ssr............int	00FFF1	GLOBAL	ABS GLOBAL
m_ste............int	00000C	GLOBAL	ABS GLOBAL
m_stie...........int	00000E	GLOBAL	ABS GLOBAL
m_stxa...........int	00FFF3	GLOBAL	ABS GLOBAL
m_stxh...........int	00FFF6	GLOBAL	ABS GLOBAL
m_stxl...........int	00FFF4	GLOBAL	ABS GLOBAL

m_stxm.....int	00FFF5	GLOBAL	ABS GLOBAL	
m_syn.....int	000009	GLOBAL	ABS GLOBAL	
m_tcm.....int	00000F	GLOBAL	ABS GLOBAL	
m_tde.....int	000006	GLOBAL	ABS GLOBAL	
m_tdre.....int	000001	GLOBAL	ABS GLOBAL	
m_te.....int	000009	GLOBAL	ABS GLOBAL	
m_tfs.....int	000002	GLOBAL	ABS GLOBAL	
m_tie.....int	00000C	GLOBAL	ABS GLOBAL	
m_tmie.....int	00000D	GLOBAL	ABS GLOBAL	
m_trne.....int	000000	GLOBAL	ABS GLOBAL	
m_tsr.....int	00FFEE	GLOBAL	ABS GLOBAL	
m_tue.....int	000004	GLOBAL	ABS GLOBAL	
m_tx.....int	00FFEF	GLOBAL	ABS GLOBAL	
m_wake.....int	000005	GLOBAL	ABS GLOBAL	
m_wds.....int	000003	GLOBAL	ABS GLOBAL	
m_wds0.....int	000000	GLOBAL	ABS GLOBAL	
m_wds1.....int	000001	GLOBAL	ABS GLOBAL	
m_wds2.....int	000002	GLOBAL	ABS GLOBAL	
m_wl.....int	006000	GLOBAL	ABS GLOBAL	
m_wl0.....int	00000D	GLOBAL	ABS GLOBAL	
m_wl1.....int	00000E	GLOBAL	ABS GLOBAL	
m_woms.....int	000007	GLOBAL	ABS GLOBAL	
no_ov.....int	P:00309A	RTC_handler	REL LOCAL	
put555_host.....int	P:002CA4	asmsupport	REL GLOBAL	
putdata.....int	P:00300D	ipp_driver	REL LOCAL	
rtc_isr.....int	P:003085	RTC_handler	REL LOCAL	
simple_life.....int	P:00307A	RTC_handler	REL LOCAL	
ssi_rxe_handler..int	P:000040	interrupts	ABS GLOBAL	
tickerhi.....int	Y:0030AE	RTC_handler	REL LOCAL	
tickerlo.....int	Y:0030AD	RTC_handler	REL LOCAL	
wait_for_DT.....int	P:002FE9	ipp_driver	REL LOCAL	
wait_for_host....int	P:002C9B	asmsupport	REL LOCAL	
wait_for_host_55.int	P:002CA7	asmsupport	REL LOCAL	
wait_rq_dwn.....int	P:002FB1	ipp_driver	REL LOCAL	
wait_rx_dma.....int	P:002FB6	ipp_driver	REL LOCAL	
wait_tx_dma.....int	P:002FEF	ipp_driver	REL LOCAL	
waitnext_RX.....int	P:002FCC	ipp_driver	REL LOCAL	
waitnext_TX.....int	P:003008	ipp_driver	REL LOCAL	
xbuf_vidbuf.....int	X:000000	GLOBAL	ABS GLOBAL	BUFFER
y_a1_putaside_ph.int	Y:000001	GLOBAL	ABS GLOBAL	
y_a1_putaside_rf.int	Y:000000	GLOBAL	ABS GLOBAL	
y_hi12_mask.....int	Y:000004	GLOBAL	ABS GLOBAL	
y_hi_to_lo_mul...int	Y:000007	GLOBAL	ABS GLOBAL	
y_hi_to_lo_mul_1.int	Y:000006	GLOBAL	ABS GLOBAL	
y_lo12_mask.....int	Y:000005	GLOBAL	ABS GLOBAL	
y_lo_to_hi_mul...int	Y:000008	GLOBAL	ABS GLOBAL	

y_minus3.....int	Y:000012	GLOBAL	ABS GLOBAL	
y_page_controls_.int	Y:00000A	GLOBAL	ABS GLOBAL	
y_plus1.....int	Y:00000F	GLOBAL	ABS GLOBAL	
y_plus4.....int	Y:000010	GLOBAL	ABS GLOBAL	
y_plus8.....int	Y:000011	GLOBAL	ABS GLOBAL	
y_read_fb_win....int	Y:000009	GLOBAL	ABS GLOBAL	
y_tbt_lineinc....int	Y:000002	GLOBAL	ABS GLOBAL	
y_vidbuf_mod.....int	Y:000003	GLOBAL	ABS GLOBAL	
y_winptr_mask....int	Y:00000E	GLOBAL	ABS GLOBAL	
y_winstart.....int	Y:00000C	GLOBAL	ABS GLOBAL	
y_x1_page_contro.int	Y:00000B	GLOBAL	ABS GLOBAL	
y_x1_winstart....int	Y:00000D	GLOBAL	ABS GLOBAL	

Symbol Listing by Value

Value	Name	Value	Name	Value
Name				
000000	DT_MODEBIT	000000	IRQA_L0_BIT	000000
M_HRDF				
000000	M_HRDF	000000	M_HRIE	000000
M_HRIE				
000000	M_IAL0	000000	M_IAL0	000000
M_IF0				
000000	M_IF0	000000	M_OF0	000000
M_OF0				
000000	M_TRNE	000000	M_TRNE	000000
M_WDS0				
000000	M_WDS0	000000	NO_MODE	000000
VIDBUF_BOTTOM				
000000	WAITDIS_IRQFST	000000	WAITDIS_IRQFST	000000
WAITDIS_IRQFST				
000000	WAIT_DISABLE	000000	WAIT_DISABLE	000000
WAIT_DISABLE				
000000	WAIT_DISSTAT_BIT	000000	WAIT_DISSTAT_BIT	000000
WAIT_DISSTAT_BIT				
000000	XSIZE	000000	m_hrdf	000000
m_hrie				
000000	m_ial0	000000	m_if0	000000
m_of0				
000000	m_trne	000000	m_wds0	000001
DT_MODE				
000001	IRQA_L1_BIT	000001	IRQA_SRCSTAT_BIT	000001
IRQA_SRCSTAT_BIT				
000001	IRQA_SRCSTAT_BIT	000001	M_HTDE	000001
M_HTDE				
000001	M_HTIE	000001	M_HTIE	000001
M_IAL1				
000001	M_IAL1	000001	M_IF1	000001
M_IF1				
000001	M_OF1	000001	M_OF1	000001
M_TDRE				
000001	M_TDRE	000001	M_WDS1	000001
M_WDS1				
000001	m_htde	000001	m_htie	000001
m_ial1				
000001	m_if1	000001	m_of1	000001
m_tdre				
000001	m_wds1	000002	IRQA_SEN_BIT	000002
IRQB_STAT_BITNO				
000002	IRQB_STAT_BITNO	000002	IRQB_STAT_BITNO	000002
M_HCIE				
000002	M_HCIE	000002	M_HCP	000002
M_HCP				
000002	M_IAL2	000002	M_IAL2	000002
M_IF				
000002	M_IF	000002	M_OF	000002
M_OF				
000002	M_RDRF	000002	M_RDRF	000002
M_SCD0				
000002	M_SCD0	000002	M_TFS	000002
M_TFS				
000002	M_WDS2	000002	M_WDS2	000002
WAITENA_IRQFST				
000002	WAITENA_IRQFST	000002	WAITENA_IRQFST	000002
WAIT_ENABLE				
000002	WAIT_ENABLE	000002	WAIT_ENABLE	000002
m_hcie				
000002	m_hcp	000002	m_ial2	000002
m_if				

```
000002          m_rdrf          000002          m_scd0          000002
  m_tfs
000002          m_wds2          000003          ACK_TD          000003
  IRQA_STAT_BITNO
000003          IRQA_STAT_BITNO 000003          IRQA_STAT_BITNO 000003
  M_HF0
000003          M_HF0           000003          M_HF2           000003
  M_HF2
000003          M_IBL0          000003          M_IBL0          000003
  M_IDLE
000003          M_IDLE          000003          M_RFS           000003
  M_RFS
000003          M_SCD1          000003          M_SCD1          000003
  M_WDS
000003          M_WDS           000003          REQ_ABT         000003
  m_hf0
000003          m_hf2           000003          m_ibl0          000003
  m_idle
000003          m_of            000003          m_rfs           000003
  m_scd1
```



```
000003          m_wds           000004          ACK_DT          000004
  BASE_PROC_ID
000004          EEPROM_WE_BITNO 000004          EEPROM_WE_BITNO 000004
  EEPROM_WE_BITNO
000004          IRQA_SRC_BITNO  000004          IRQA_SRC_BITNO  000004
  IRQA_SRC_BITNO
000004          IRQ_SRC_BIT     000004          M_HF1           000004
  M_HF1
000004          M_HF3           000004          M_HF3           000004
  M_IBL1
000004          M_IBL1          000004          M_OR            000004
  M_OR
000004          M_SBK           000004          M_SBK           000004
  M_SCD2
000004          M_SCD2          000004          M_TUE           000004
  M_TUE
000004          REQ_TD          000004          m_hf1           000004
  m_hf3
000004          m_ibl1          000004          m_or            000004
  m_sbk
000004          m_scd2          000004          m_tue           000005
  HFB_EVENT_BITNO
000005          HFB_EVENT_BITNO 000005          HFB_EVENT_BITNO 000005
  M_IBL2
000005          M_IBL2          000005          M_PE            000005
  M_PE
000005          M_ROE           000005          M_ROE           000005
  M_SCKD
000005          M_SCKD          000005          M_WAKE          000005
  M_WAKE
000005          SSD1BIT         000005          SSD1BIT         000005
  SSD1BIT
000005          m_ibl2          000005          m_pe            000005
  m_roe
000005          m_sckd          000005          m_wake          000006
  M_FE
000006          M_FE            000006          M_RWI           000006
  M_RWI
000006          M_TDE           000006          M_TDE           000006
  SSD2BIT
000006          SSD2BIT         000006          SSD2BIT         000006
  m_fe
000006          m_rwi           000006          m_tde           000007
  DSP_ID_BITNO
000007          DSP_ID_BITNO    000007          DSP_ID_BITNO    000007
  M_DMA
000007          M_DMA           000007          M_IAL           000007
  M_IAL
000007          M_R8            000007          M_R8            000007
  M_RDF
000007          M_RDF           000007          M_WOMS          000007
  M_WOMS
000007          m_dma           000007          m_ial           000007
  m_r8
000007          m_rdf           000007          m_woms          000008
  IRQA_VEC_ADR
000008          M_FSL           000008          M_FSL           000008
  M_RE
000008          M_RE            000008          m_fsl           000008
  m_re
000009          M_SYN           000009          M_SYN           000009
  M_TE
000009          M_TE            000009          m_syn           000009
  m_te
00000A          M_GCK           00000A          M_GCK           00000A
  M_HPL0
00000A          M_HPL0          00000A          M_ILIE          00000A
  M_ILIE
00000A          m_gck           00000A          m_hpl0          00000A
  m_ilie
00000B          M_HPL1          00000B          M_HPL1          00000B
  M_MOD
00000B          M_MOD           00000B          M_RIE           00000B
  M_RIE
00000B          m_hpl1          00000B          m_mod           00000B
  m_rie
00000C          M_COD           00000C          M_COD           00000C
  M_SSL0
00000C          M_SSL0          00000C          M_STE           00000C
  M_STE
00000C          M_TIE           00000C          M_TIE           00000C
  m_cod
00000C          m_ssl0          00000C          m_ste           00000C
  m_tie
00000D          M_SCP           00000D          M_SCP           00000D
  M_SRE
00000D          M_SRE           00000D          M_SSL1          00000D
  M_SSL1
00000D          M_TMIE          00000D          M_TMIE          00000D
  M_WL0
00000D          M_WL0           00000D          m_scp           00000D
  m_sre
00000D          m_ssl1          00000D          m_tmie          00000D
  m_wl0
00000E          M_RCM           00000E          M_RCM           00000E
  M_SCL0
00000E          M_SCL0          00000E          M_STIE          00000E
  M_STIE
00000E          M_WL1           00000E          M_WL1           00000E
  m_rcm
00000E          m_scl0          00000E          m_stie          00000E
  m_wl1
00000F          M_PSR           00000F          M_PSR           00000F
  M_SCL1
00000F          M_SCL1          00000F          M_SRIE          00000F
  M_SRIE
00000F          M_TCM           00000F          M_TCM           00000F
  m_psr
```

```
00000F          m_scl1          00000F          m_srie          00000F
  m_tcm
000010          WAITDIS_IRQSLW  000010          WAITDIS_IRQSLW  000010
  WAITDIS_IRQSLW
000012          WAITENA_IRQSLW  000012          WAITENA_IRQSLW  000012
  WAITENA_IRQSLW
2.000000E+001   CPU_Mhz         000014          GPP_REQ_DT_NUM  000018
  M_HF
000018          M_HF            000018          m_hf            00001C
  M_SCD
00001C          M_SCD           00001C          m_scd           000028
  GPP_REQ_DT_ADR



Motorola DSP Linker  Version 4.0.6  94-05-12  08:12:40  obj/pb_BEP.map  Page 24


000038          M_IBL           000038          M_IBL           000038
  m_ibl
000040          ssi_rxe_handler 000050          Fcopy_words     000060
  DEBUG_BIT_MASK
000060          DEBUG_BIT_MASK  000060          DEBUG_BIT_MASK  000069
  Fcopy_hi_pixels
000096          Fcopy_lo_pixels 0000C3          Fread_fb16      0000E7
  Fread_fb12
0000FF          M_PM            0000FF          M_PM            0000FF
  m_pm
00011C          Ftbt_hi         000165          Ftbt_lo         000AAB
  CRT0ORG
000AAB          F__start        000AD1          F__crt0_end     000AD5
  F__printf_end
000AD7          Fsim4           000AE2          L6              000AE3
  ASM_APP_0
000AE4          ASM_APP_1       000AE4          L15             000AE6
  L2
000AE8          L11             000AEA          ASM_APP_2       000AEB
  ASM_APP_3
000AEB          L13             000AEF          Fsimulate_frame_ 000B02
  L20
000B19          L26             000B21          L28             000B2B
  L24
000B2E          L27             000B47          Fsimulate_row_sy 000B4C
  L33
000B4D          ASM_APP_4       000B4E          ASM_APP_5       000B4E
  L35
000B5E          Fsimulate_word_r 000B85         L37             000B8C
  L38
000BAE          ASM_APP_6       000BAF          ASM_APP_7       000BCA
  ASM_APP_8
000BCB          ASM_APP_9       000BDE          ASM_APP_A       000BDF
  ASM_APP_B
000BF0          ASM_APP_C       000BF1          ASM_APP_D       000C00
  M_HPL
000C00          M_HPL           000C00          m_hpl           000C16
  L40
000C27          L41             000C2A          L39             000C38
  Falt_simulate_wo
000C50          L43             000C60          L49             000C66
  L47
000C68          ASM_APP_E       000C69          ASM_APP_F       000C79
  L46
000C8C          Fxmemcpy        000CB2          L55             000CB3
  ASM_APP_10
000CB5          ASM_APP_11      000CC2          L57             000CC3
  L56
000CD0          Fymemcpy        000CF6          L62             000D05
  L64
000D06          L63             000D13          Fpmemcpy        000D39
  L69
000D3A          ASM_APP_12      000D3C          ASM_APP_13      000D49
  L71
000D4A          L70             000D57          Fmemdump_cmd    000D7E
  L73
000D8D          L80             000D95          L75             000DA4
  L76
000DB3          L77             000DC0          L74             000DE5
  L81
000DF6          Fmempoke_cmd    000E23          L89             000E2B
  ASM_APP_14
000E2B          L84             000E2C          ASM_APP_15      000E2E
  L85
000E31          ASM_APP_16      000E31          L86             000E32
  ASM_APP_17
000E32          L83             000E3F          Fmemdump_auto   000E69
  Fmain
000E8A          L92             000EA6          L97             000EAC
  L96
000EC1          Finit_portc_intr 000EC6         ASM_APP_0       000EC7
  ASM_APP_1
000EC9          ASM_APP_2       000EC9          L2              000ECA
  ASM_APP_3
000ECA          L3              000ECB          Fset_vbp_left_of 000EEF
  L8
000F00          BOARD_ID_MASK   000F00          BOARD_ID_MASK   000F00
  BOARD_ID_MASK
000F27          L10             000F38          Fset_vbp_new_con 000F53
  L29
000F58          L28             000FC2          L18             000FEC
  L21
000FFE          L34             000FFF          L19             000FFF
  M_CD
000FFF          M_CD            000FFF          m_cd            001000
  STACK_START
001000          VIDBUF_LEN      001000          VIDBUF_TOP      00102A
  L23
001054          L26             001066          L35             001067
  L17
00106B          L31             00106E          L33             001097
  Fonetime_init
0010C4          ASM_APP_4       0010C5          ASM_APP_5       0010D5
  L44
001102          L43             001114          L46             001115
  L45
00117F          Fvb_add         00119A          L3              0011AA
  Fvb_distance
0011BF          L6              0011C5          Fcheck4         0011D6
  ASM_APP_0
0011D8          ASM_APP_1       0011E3          ASM_APP_2       0011E4
  ASM_APP_3
0011EE          ASM_APP_4       0011EF          ASM_APP_5       0011F9
  ASM_APP_6
0011FA          ASM_APP_7       001205          L11             00120D
  L8
00121B          Fswitch_params  001247          L15             001251
  L18
00125E          L16             001269          L17             001276
  Fsync_frame
001298          L39             0012AB          L38             0012B7
  L23
0012B9          ASM_APP_8       0012BB          ASM_APP_9       0012C9
  ASM_APP_A
0012CA          ASM_APP_B       0012D6          L25             0012EF
  L28
```

```
00130E          L31             00131F          L32             001333
  L41
001339          L34             001357          L40             001359
  L20
```



```
001364          L21             001377          Fsync_row       0013CB
  L44
0013CD          L43             0013DA          Fdeal_into_fb   001419
  L58
00142E          L48             001433          L49             00143F
  L52
001449          L69             00144C          L53             00144C
  L55
00145A          L47             00146B          L60             00146D
  L70
0014AE          L63             0014B6          L62             0014C2
  L71
00150C          L66             00151C          L65             001534
  L68
00153D          L59             00154E          Fcopy_line      001582
  L87
001595          L92             0015B1          L86             0015D7
  L82
001623          L91             001631          L81             00165C
  L80
001666          L76             00166F          L73             0016B1
  L88
0016C2          Ftry_copy_line  001720          L94             00175D
  L99
0017A1          L97             0017B1          Fabort_stream   0017BF
  Fget_imp_offset
00180D          L31             00181F          L3              001836
  L5
001846          L8              00185B          L9              00185D
  L10
001863          L6              001866          L12             00187D
  L14
00189E          L15             0018A1          L17             0018B7
  L19
0018C4          L33             0018C7          L18             0018CC
  L22
0018DF          L23             0018EF          L26             001906
  L27
001908          L28             001912          L34             001914
  L32
001915          L2              001928          Fread_imp       001986
  L36
0019AC          L45             0019BD          L75             0019C2
  L46
0019D0          L44             0019ED          L52             0019EF
  L76
0019F5          L51             0019FD          L74             001A00
  L43
001A1D          L55             001A2E          L77             001A3F
  L57
001A41          L58             001A49          L71             001A58
  L62
001A5C          L63             001A67          L65             001A99
  L70
001ABD          L79             001AC4          L66             001AD3
  L78
001AEC          Fread_nip       001B30          L93             001B3A
  L82
001B62          L83             001B6B          L85             001B96
  L86
001BB6          L88             001BDB          L89             001BE2
  L123
001BEB          L122            001BED          L81             001BFF
  L115
001C11          L95             001C33          L97             001C41
  L100
001C55          L105            001C67          L127            001C77
  L125
001C7E          L107            001C92          L126            001C96
  L108
001C9A          L106            001C9F          L110            001CC2
  L111
001CDE          L124            001CE1          L94             001CFE
  L120
001D29          L116            001D3C          Ftbt            001D88
  L129
001DA6          L131            001DCC          L135            001DD4
  L133
001DE8          L136            001DF4          L130            001E03
  FSend_Host_Messa
001E20          L2              001E2D          L3              001E49
  FGet_Host_Messag
001E62          L5              001E64          L6              001E6C
  Fsend_test_word
001E95          L8              001EA2          Fsend_test_int  001ED6
  L10
001EE5          Fparam_dump     001F00          M_DC            001F00
  M_DC
001F00          m_dc            001F33          L15             001F41
  L23
001F4E          L19             001F6C          L21             001F84
  L24
001F97          Frp_dump        001FED          L29             002003
  L43
002011          L33             002025          L41             002037
  L37
002041          L39             002073          L44             002087
  Fparse_viddsp_cm
0020B2          L47             0020BA          L48             0020C2
  L50
0020C8          L51             0020CE          L52             0020DD
  L63
0020E4          L53             0020F3          L64             0020FA
  L54
002100          L55             002150          L56             00215B
  ASM_APP_0
00215C          ASM_APP_1       002163          L57             00216F
  L58
002175          L59             00217B          L61             00217E
  L62
002192          L65             002197          L46             0021A8
  Ftry_get_message
0021E3          L69             002200          L97             002202
  L70
00220B          L71             00221B          L72             002224
  L77
002240          L75             002247          L78             002282
  L82
0022A6          L95             0022D1          L84             0022E0
  L85
```

0022EF	L86	0022FE	L87	00230B
L98				
002317	L88	002324	L89	002331
L99				
00233D	L90	00234A	L100	002356
L91				
002360	L101	002373	L92	002390
L102				
002395	L67	0023A6	Ftry_relay_error	0023D4
L105				
0023D9	L104	0023E3	Fsignal_frame_st	002413
L107				
002422	Fsignal_frame_en	002452	L109	002461
Fsignal_frame_er				
002497	L111	0024A7	Ffind_frame_in_r	0024C5
L6				
0024D6	L9	0024E5	L10	0024FC
Ffind_frame				
002511	L13	00251B	L14	002527
Fsave_frame				
002557	L23	00256C	L21	00257D
L20				
00258D	L17	00259B	Funsave_frame	0025C7
L28				
0025D9	L29	0025E5	L25	0025F3
Fdump_frame_raw				
00262B	L46	002632	L31	002650
L42				
002663	L35	002665	L36	00266C
L40				
002680	L44	002681	L43	002693
L41				
0026B1	L47	0026BD	L32	0026CC
Fdump_imp_array				
00270B	L73	002712	L49	002731
L51				
00273A	L52	002750	L53	002759
L54				
00276F	L55	002778	L56	00278E
L57				
002797	L58	0027BA	L71	0027CE
L62				
0027D0	L63	0027F0	L64	0027F5
L65				
0027FD	L61	00280D	L74	00281C
L67				
002821	L68	002851	L69	00286D
L66				
002887	L75	002895	L50	0028A6
Fdump_tbt_array				
0028E3	L79	0028EA	L77	00291D
L80				
002924	L78	002932	Fset_up_region	00297D
L82				
002983	L83	0029B9	L87	0029CF
L96				
0029E9	L95	0029EA	L94	002A0C
L91				
002A22	L97	002A27	L93	002A28
L92				
002A43	Ffill_out_params	002A72	L106	002A7F
L104				
002A91	L103	002A97	L124	002A99
L99				
002A9D	L107	002AAB	L109	002AAF
L108				
002AB1	L125	002AE5	L120	002AFF
L128				
002B0A	L115	002B0C	L126	002B21
L129				
002B5C	L130	002B94	L117	002B9C
L114				
002BB1	L127	002BB4	L113	002BC4
L122				
002BD5	Fsetup_window	002BFB	Fsetup_dual_wind	002C24
Fsetup_triple_wi				
002C50	Finit_communicat	002C5D	Fcopy_lowp_routi	002C69
Finit_intr_vecto				
002C78	Fcheck_host	002C8D	Fget_host	002C93
Fput_host				
002C9B	wait_for_host	002CA4	put555_host	002CA7
wait_for_host_55				
002CAC	Ftxd_on	002CAE	Ftxd_off	002CB0
Ftxd_tog				
002CB2	Fsc1_on	002CB4	Fsc1_off	002CB6
FIPP_collectmsg				
002CF2	L4	002CF6	L3	002CF6
L5				
002D01	L2	002D01	L6	002D38
L8				
002D42	L7	002D42	L9	002D46
L1				
002D4F	FIPP_get_message	002D56	L11	002D6C
L14				
002D79	L16	002D7B	L17	002D7E
L13				
002D7E	L15	002D80	L10	002D80
L12				
002D89	FIPP_send_messag	002DBB	L20	002DC8
L19				
002DD1	FIPP_make_header	002E04	L22	002E12
L23				
002E38	L21	002E41	FIPP_makeheader	002E74
L25				
002E82	L26	002EB6	L24	002EBF
FIPP_register_ad				
002EFF	L28	002F0D	L29	002F3D
L30				
002F3F	L27	002F49	FIPP_get_address	002F6A
L32				
002F6E	L33	002F76	L34	002F78
L31				
002F84	Finitialize_dma_	002F94	its_dsp_zero	002F9F
gppReqDTISR				
002FA3	Freceive_words	002FB1	wait_rq_dwn	002FB6
wait_rx_dma				
002FCA	RX_even	002FCC	waitnext_RX	002FD1
getdata				
002FD4	abort_RX	002FDA	exit_RX	002FE3
Fsend_words				

002FE9	wait_for_DT	002FEF	wait_tx_dma	003000
M_SSL				
003000	M_SSL	003000	m_ssl	003006
TX_even				
003008	waitnext_TX	00300D	putdata	003010
abort_TX				

```
003018          exit_TX         003021          Finit_RTC_driver  00303F
  Fget_irqa_tick
003043          Fget_nsynch_time 00307A         simple_life       003085
  rtc_isr
00309A          no ov           00309F          Fcheck_buddy_tim  0030C1
  DSIZE
0030C1          LSIZE           0030C1          YSIZE             0030C3
  L1
0030C3          L2              0030CC          Fprepare_IPP_hea  0030FE
  L4
003109          L5              003114          L3                00311C
  Fmemcpy
003133          L4              003135          L6                003144
  ldiv_aa
003148          ldiv_bb         00314C          ldiv_ba           003159
  L2_ba
003165          L3_ba           003174          L6_ba             003179
  L5_ba
003180          L8_ba           003183          L7_ba             003189
  L4_ba
00318D          L9_ba           003195          L10_ba            00319C
  ldiv_ab
0031A9          L2_ab           0031B5          L3_ab             0031C4
  L6_ab
0031C9          L5_ab           0031D0          L8_ab             0031D3
  L7_ab
0031D9          L4_ab           0031DD          L9_ab             0031E5
  L10_ab
0031EC          PSIZE           006000          M_WL              006000
  M_WL
006000          m_wl            007FFF          TOP_OF_MEMORY     008000
  Boot_EEROM
00C000          M_SCL           00C000          M_SCL             00C000
  m_scl
00FFC0          FRTC_Count      00FFC0          FRTC_Count        00FFC0
  FRTC_Count
00FFC1          FRF_Count       00FFC1          FRF_Count         00FFC1
  FRF_Count
00FFC2          FGPS_Count      00FFC2          FGPS_Count        00FFC2
  FGPS_Count
00FFC3          FDGlue_Ctrl     00FFC3          FDGlue_Ctrl       00FFC3
  FDGlue_Ctrl
00FFC3          FDGlue_Stat     00FFC3          FDGlue_Stat       00FFC3
  FDGlue_Stat
00FFE0          M_PBC           00FFE0          M_PBC             00FFE0
  m_pbc
00FFE1          M_PCC           00FFE1          M_PCC             00FFE1
  m_pcc
00FFE2          M_PBDDR         00FFE2          M_PBDDR           00FFE2
  m_pbddr
00FFE3          M_PCDDR         00FFE3          M_PCDDR           00FFE3
  m_pcddr
00FFE4          M_PBD           00FFE4          M_PBD             00FFE4
  m_pbd
00FFE5          M_PCD           00FFE5          M_PCD             00FFE5
  m_pcd
00FFE8          ACK_PORT        00FFE8          M_HCR             00FFE8
  M_HCR
00FFE8          m_hcr           00FFE9          M_HSR             00FFE9
  M_HSR
00FFE9          REQ_PORT        00FFE9          m_hsr             00FFEB
  M_HRX
00FFEB          M_HRX           00FFEB          M_HTX             00FFEB
  M_HTX
00FFEB          m_hrx           00FFEB          m_htx             00FFEC
  M_CRA
00FFEC          M_CRA           00FFEC          m_cra             00FFED
  M_CRB
00FFED          M_CRB           00FFED          m_crb             00FFEE
  M_SR
00FFEE          M_SR            00FFEE          M_TSR             00FFEE
  M_TSR
00FFEE          m_sr            00FFEE          m_tsr             00FFEF
  M_RX
00FFEF          M_RX            00FFEF          M_TX              00FFEF
  M_TX
00FFEF          m_rx            00FFEF          m_tx              00FFF0
  M_SCR
00FFF0          M_SCR           00FFF0          m_scr             00FFF1
  M_SSR
00FFF1          M_SSR           00FFF1          m_ssr             00FFF2
  M_SCCR
00FFF2          M_SCCR          00FFF2          m_sccr            00FFF3
  M_STXA
00FFF3          M_STXA          00FFF3          m_stxa            00FFF4
  M_SRXL
00FFF4          M_SRXL          00FFF4          M_STXL            00FFF4
  M_STXL
00FFF4          m_srxl          00FFF4          m_stxl            00FFF5
  M_SRXM
00FFF5          M_SRXM          00FFF5          M_STXM            00FFF5
  M_STXM
00FFF5          m_srxm          00FFF5          m_stxm            00FFF6
  M_SRXH
00FFF6          M_SRXH          00FFF6          M_STXH            00FFF6
  M_STXH
00FFF6          m_srxh          00FFF6          m_stxh            00FFFE
  M_BCR
00FFFE          M_BCR           00FFFE          m_bcr             00FFFF
  M_IPR
00FFFF          M_IPR           00FFFF          m_ipr             085DAF
  SSI_RX_INSTR
0BF080          ABS_JSR_INSTR   0D0000          SHORT_JSR_INSTR   000000
  xbuf_vidbuf
008000          FFrame_X8       009000          FFrame_X9         00A000
  FFrame_XA
00B000          FFrame_XB       000000          y_a1_putaside_rf  000001
  y_a1_putaside_ph
000002          y_tbt_lineinc   000003          y_vidbuf_mod      000004
  y_hi12_mask
000005          y_lo12_mask     000006          y_hi_to_lo_mul_1  000007
  y_hi_to_lo_mul
000008          y_lo_to_hi_mul  000009          y_read_fb_win     00000A
  y_page_controls_
```



```
00000B          y_x1_page_contro 00000C         y_winstart        00000D
  y_x1_winstart
00000E          y_winptr_mask   00000F          y_plus1           000010
  y_plus4
000011          y_plus8         000012          y_minus3          000013
  Ferrno
000014          F__stack_safety 000015          F__mem_limit      000016
  F__break
000017          F__y_size       000018          F__max_signal     000019
  Fssi_rx_exceptio
00001A          Fssi_tx_exceptio 00001B         Fsci_rx_exceptio  00001C
  Fdrop_frame_sync
00001D          Fccd0sim        00008D          Fccd1sim          0000FD
  F___id0
```

0000FE	F___ABCDcnt1	0000FF	F___Aline2	000100
F___Bline3				
000101	F___Cline4	000102	F___Dline5	000103
F___sim_cntr6				
000104	F___line_written	000105	Ffb	003040
Ftestmsg_dest				
003044	Fvideo_dest	003048	Fme	00304C
Fsingle_loop				
00304D	Fdoloop	00304E	Fsimulating	00304F
F___state0				
003050	F___count1	003051	F___local_id2	003052
L60				
003067	L93	00309C	F___got_header0	00309D
F___header1				
0030A9	Fmy_name	0030AA	Flocal_processor	0030AB
Fmaster_node_id				
0030AC	dma_state	0030AD	tickerlo	0030AE
tickerhi				
0030AF	Fmajortag	0030B1	Fminortag	0030B3
F___last_tick0				
0030B4	Fbuddy_hdr	0030C0	Fmy_ipp_addr	008000
FFrame_Y8				
009000	FFrame_Y9	00A000	FFrame_YA	00B000
FFrame_YB				
00EF00	FFrameReg_X8	00EF01	FFrameReg_X9	00EF02
FFrameReg_XA				
00EF03	FFrameReg_XB	00EF04	FFrameReg_Y8	00EF05
FFrameReg_Y9				
00EF06	FFrameReg_YA	00EF07	FFrameReg_YB	