THE FAR ULTRAVIOLET SPECTROSCOPIC EXPLORER (FUSE) INSTRUMENT DATA SYSTEM

Brian K. Heggestad and Robert C. Moore
The Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Road, Laurel, Maryland 20723-6099

Introduction

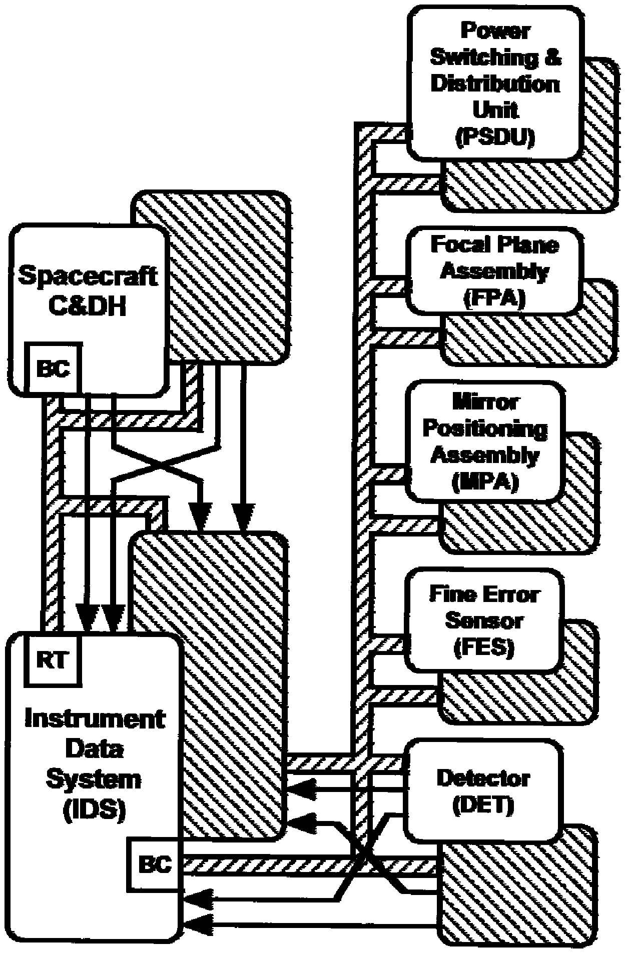
The Far Ultraviolet Spectroscopic Explorer (FUSE) instrument was designed to obtain high-resolution spectra of faint objects throughout the Galaxy. The FUSE satellite was launched on June 24, 1999, on its three-year mission to explore the universe using the technique of high-resolution spectroscopy in the far-ultraviolet spectral region. FUSE will be one of the most far-reaching scientific explorations of space to date. A central general-purpose computer, the Instrument Data System (IDS), controls the FUSE instrument. This is a fully redundant, programmable processor that provides command and telemetry functions for all other subsystems in the instrument. The IDS processor also provides science data processing and storage, star tracker fine pointing data processing, image processing, delayed commanding, rule-based autonomy and "safing," and instrument time and thermal management functions.

This paper describes the architecture for the IDS flight hardware and its real-time embedded flight software. The design uses commercial off-the-shelf (COTS) software components as much as possible, to reduce cost and software development time. The features of the IDS design that provide radiation hardness and fault tolerance are described. Implementation of software to meet the functional requirements is accomplished using a relatively small number of prioritized real-time tasks. A commercial real-time operating system kernel manages and supports these tasks. Inter-task communication is described, as are the software test and validation methods. The paper shows how custom ground support equipment was developed to facilitate software development and testing.

Reliable communications between the IDS and the FUSE spacecraft bus are accomplished using a MIL-STD-1553B bus that has an imposed, deterministic real-time protocol. Similarly, communication between the IDS and the other instrument subsystems uses a second MIL-STD-1553B bus that has its own time-division multiplex real-time protocol. The design of these real-time protocols is described, with particular attention to reliability and testability.

IDS Hardware

Figure 1 is a context diagram that shows how the IDS connects with the spacecraft and the other instrument subsystems. The FUSE IDS is a fully redundant design in which at most one of the two flight IDS boxes is powered at a time. In Figure 1 redundant units are shaded, as are the MIL-STD-1553B buses. There are redundant interfaces between the IDS and the FUSE spacecraft's redundant command and data handling (C&DH) processors. These comprise a MIL-STD-1553B spacecraft data bus (SDB) and redundant 1-Hz timing synchronization signals. A second MIL-STD-1553B instrument data bus (IDB) provides redundant communications between the IDS and all of the other subsystems of the FUSE instrument, each of which has redundant interfaces to the IDS. These other subsystems include the power switching and distribution unit (PSDU), focal plane assembly (FPA), mirror positioning assembly (MPA), fine error sensor (a visible-light star camera, FES), and detector (DET). In addition to the IDB, the detectors provide redundant science data interfaces to the IDS, each of which can supply the IDS with 32-bit data at rates up to 41,667 Hz.



Power Switching & Distribution Unit (PSDU)
Spacecraft C&DH
BC
Focal Plane Assembly (FPA)
Mirror Positioning Assembly (MPA)
Fine Error Sensor (FES)
RT
Instrument Data System (IDS)
Detector (DET)
BC

Figure 1. IDS Context Diagram

Figure 2 shows the inner components of the IDS. There is a MIL-STD-1553B protocol controller that implements a remote terminal (RT) for the SDB. The heart of the IDS is a radiation-tolerant 68020 central processing unit (CPU) that runs at 17 MHz and provides approximately 4 Mips of throughput. A 68882 floating-point unit (FPU) permits hardware floating-point calculations to be made. The CPU is supported by 128KB of programmable read-only memory (PROM), two 512KB banks of electrically erasable PROM (E^2PROM), and 1MB of radiation-tolerant static random-access memory (SRAM). In addition to the SRAM there is 48MB (two 24MB banks) of bulk SRAM that has built-in single-bit error correction, double-bit error detection (SEC-DED) error detection and correction (EDAC).

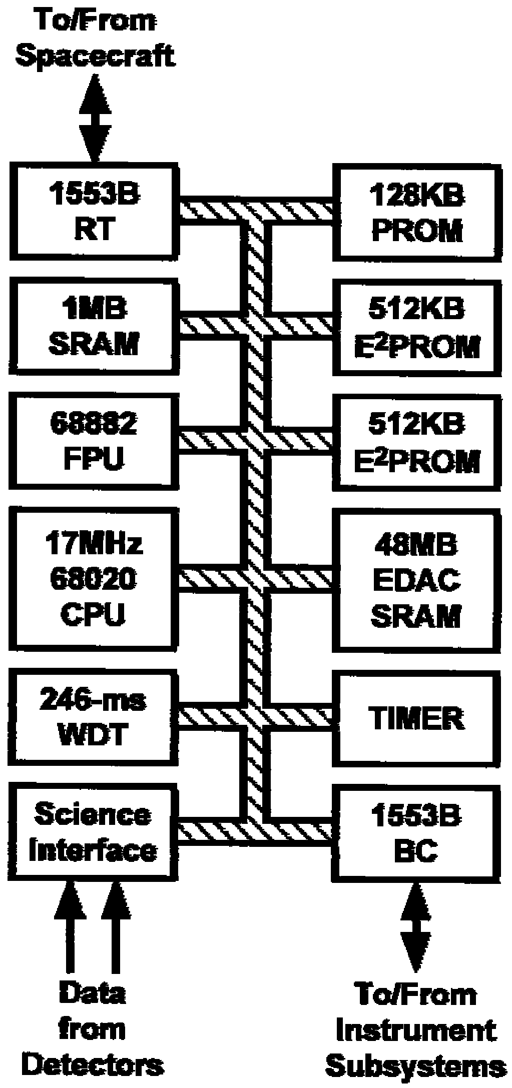


Figure 2. IDS Hardware Block Diagram

There is a watchdog timer (WDT) that must be reset within 246 milliseconds, or else the WDT will time out and reset the IDS. A programmable timer permits the IDS to implement a “tick timer” for the real-time operating system (RTOS), as well as 125 8-ms minor frames each second, which are the “slices” of real time to which most real-time IDS operations are synchronized. There is a custom science interface that accepts and merges science data from the two detector subsystems, and a MIL-STD-1553B protocol controller that implements a bus controller (BC) for the IDB.

IDS Software

Developing the IDS software was a significant challenge because many real-time and near-real-time tasks needed to be performed concurrently:

- process FES images or generate FPD packets from FES centroids
- collect, format and down-link housekeeping and memory dump data from instrument subsystems at rates of over 2000 bytes/second in nine packet types
- collect and sum detector pulse height analysis (PHA) data
- collect, process, and store detector science data at rates up to 32,000 samples (64,000 bytes) per second. Note that maintaining the science data storage and down-link rates is hampered by storage memory that operates at 8.5MHz, 1/2 of the processor bus speed.
- format and down-link science data at a rate of 14,000 bytes/second.
- allocate and de-allocate and initialize memory for future data collections
- process commands (execute or forward to subsystems) from the spacecraft at a rate of 15 commands/second
- monitor any set of incoming housekeeping data to autonomously detect and respond to anomalous/operational changes
- execute stored time-tagged and scripted commands
- manage the SDB (remote terminal) and IDB (bus controller) interfaces
- collect and process temperatures from 64 thermistors to control the temperatures of 32 instrument thermal zones to within +/- 0.5 degree Celsius
- maintain the WDT that needs to be reset at a rate greater than 4 Hz
- scrub bulk memory

The challenge was to implement time-critical tasks with others that are less urgent. For example, the BC, RT, and Detector managers must finish servicing the 1553 interfaces and processing science data within each 8-ms minor frame, while de-commutation of housekeeping data and subsequent evaluation of rules may occur over a 40-ms period, and allocation of memory or processing of an image may be allowed multiple seconds to complete.

Figure 3 shows the IDS software tasks with their interface to the spacecraft, the instrument subsystems, and to the other IDS software tasks. The highest priority function, time maintenance, is implemented as an interrupt service routine (ISR). This function sets the task schedule map to wake up each task in those minor frames in which the task is to run. Tasks are prioritized, with the most time-critical tasks having highest priority.

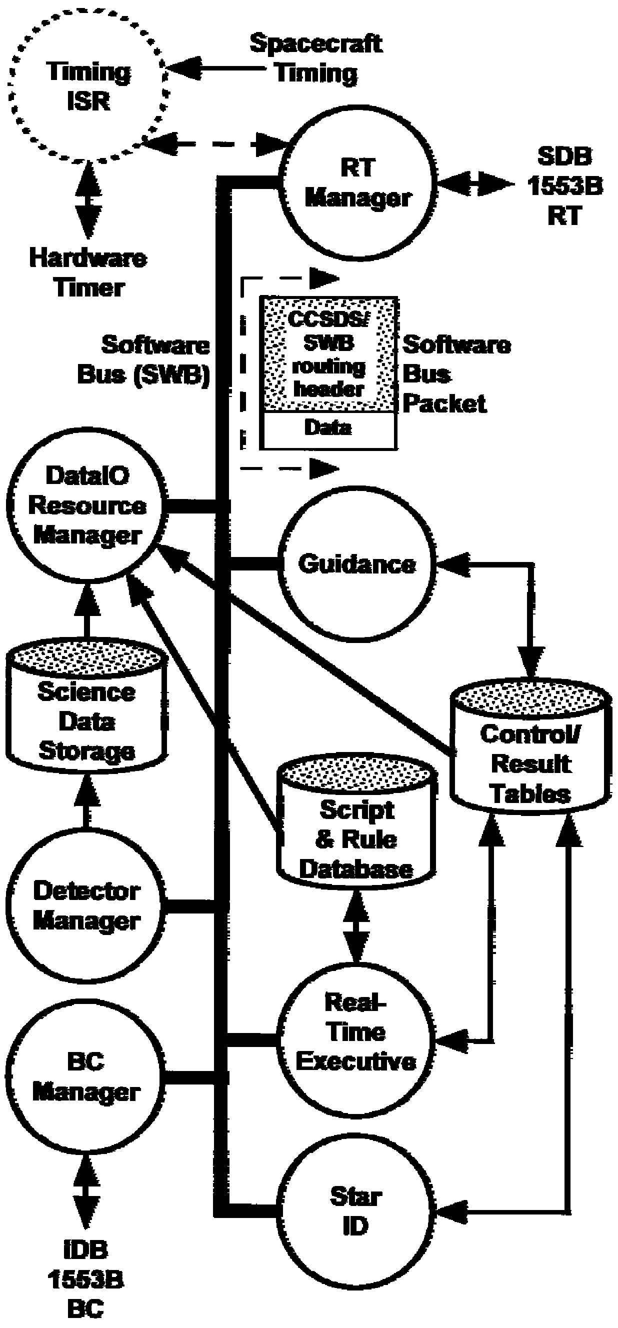


Figure 3. IDS Software Architecture

Inter-task communication is implemented with a software bus (SWB). A task places a given message on the SWB, which provides this message to any task registered for it. In some cases, such as the storage and retrieval of science data, information is communicated between tasks by direct access to memory, rather than by the software bus.

Real-Time Operating System

The Versatile Real-Time eXecutive kernel for embedded processors (VRTXsa), provided by Microtec Research, was chosen to provide proven COTS multitasking support, event-driven, priority-based scheduling, and real-time control. The use of events to schedule tasks allowed for easily mixing two types of scheduling. The first is the routine scheduling of tasks that happen in the same minor frame of each second. The IDS timing manager schedules these events. At each eight-ms interrupt the timing manager reads from an array the word that defines which tasks are to be scheduled, and posts this word to the RTOS task manager. Certain time-critical operations occur at indeterminate times. In these cases, when the data arrive, and are available for a second task, the first task may post an event to wake the first task.

Task priorities are assigned based on the degree to which each task needs to act as a real-time task. The order of the priorities, from highest to lowest, is: BC manager, Detector manager, RT Manager, DataIO, RTE, Guidance, and Star ID. BC Manager and Detector Manager must always finish their job within the minor frame. The other tasks are not awakened every minor frame, and have a longer period of time in which to finish.

Spacecraft Command Language

The Spacecraft Command Language (SCL) provided by Interface and Control Systems, Inc., is the second COTS product used in the IDS software. SCL comprises two tasks, DataIO and RTE. These tasks provide the IDS with scripting capability (time-tagged and relative-time command capability, conditional operations, etc). It also provides a rule engine that allows for extensive autonomous function capabilities based on evaluation of subsystem and IDS telemetry. Changes in system state or telemetry may trigger rules or launch scripts.

Software Bus

The majority of communication within the IDS occurs on the software bus (SWB). Commands received by the RT manager are forwarded to the target tasks or beyond to the instrument subsystems. Telemetry collected by the BC manager from instrument subsystems is forwarded to DataIO for rate limiting, and then on to the RT manager to be sent to the spacecraft. Guidance notifies DataIO of events and DataIO notifies the RTE of database changes.

The SWB is a set of support functions used by all tasks except the Detector Manager. A fixed set of packets is managed by the SWB. These are long and short telemetry packets and command packets. There is a common header on all of the packets that identifies the contents and thereby controls the routing of the packets. A set of function calls allows the tasks to register for a given packet ID, to request an empty packet in which to place data from the SWB, and to place a full packet on the SWB. The SWB then notifies all tasks registered for the packet, and provides them with a copy of the packet. When the receiving task is finished, it returns it to the SWB, which may then allocate the packet again.

The SWB interface helps to isolate the tasks, which do not know or care from whom a given packet came, but only what type of packet it is. In this way, the SWB provides a significant contribution to the development and testing of the software.

The IDS Bench Test Equipment (BTE) used for software development, test, and validation also used the SWB and SCL, and has its own RTE and DataIO. It is able to run the same scripts and maintain the same database as the IDS RTE. It also provides command, telemetry, and script compilers. Command and telemetry formats are maintained in a “C-like” language.

Once the SWB and RT Manager (the command and telemetry interface to the spacecraft) were in place, individual flight tasks could be tested against ground tasks. Guidance and Star ID tasks were in place and tested before the flight RTE. We were able to test these tasks using near-operational scripts running on the BTE RTE, and communicating across the SWB as if it were the flight RTE. The same scenario applied to the testing of other tasks.

Boot Code

Boot code provides a highly reliable interface to support the application code. The boot code resides in PROM and cannot be mistakenly destroyed either by an errant command from the ground or by malfunctioning software. This also means that it cannot be intentionally changed or fixed. The boot code is limited in scope to provide no more than the required support. This allowed very complete testing of all aspects of the boot code:

- communicating with the ground over the SDB using CCSDS packet format,
- running of a limited number of hardware self tests,
- supporting a limited number of commands that allow the boot code to coexist with the spacecraft; e.g., a command to select the source of the 1-Hz timing signal from the spacecraft,
- supporting commands to load memory (RAM and E^2PROM) and enable the upload of new application code,
- supporting commands to dump and checksum memory to provide verification of application code loads and provide debug information in the event of an IDS failure or suspected failure during either boot or application modes.

RT Manager

The RT Manager is the interface between the IDS and the spacecraft, and thereby between the IDS and the ground. Transfer of data across this interface is managed by the hardware protocol controller. The SDB follows a strictly regulated schedule that is synchronized to the 1-Hz and 125-Hz interrupts. Specific transactions are allowed in each of the minor frames. Interrupts notify the IDS when telemetry packets have been transferred from the IDS, or command packets have arrived.

The RT Manager implements several functions. It manages the MIL-STD-1553B protocol controller. On incoming messages it removes the protocol controller formatting and generates a CCSDS packet. It performs CCSDS level verification of the incoming command packets. On outgoing data it adds the protocol controller formatting. It maintains a SWB interface, forwards incoming commands to the appropriate tasks, and receives telemetry packets to be forwarded to the spacecraft via the SDB. It also provides a command interface for the IDS timing manager and provides a “reformat” service for several SDB packets that do not follow the standard IDS packet format.

BC Manager

With the exception of a high-speed science interface with each detector, all communication with the instrument subsystems is through the BC Manager. The dissemination of commands and the collection of telemetry across the IDB is strictly controlled by a set of tables governing the transactions that occur each minor frame. The IDB is heavily loaded. In the event of retries, less than 0.5 ms of extra time is available to complete all transactions in many of the minor frames. To insure a timely start of bus transactions each minor frame, the BC Manager is the highest priority task.

At the beginning of each minor frame a single write triggers the MIL-STD-1553B protocol controller to execute the control sequence assembled in the previous minor frame. The BC manager then builds the control sequence and formats the data for IDB transmission for the following minor frame and processes the data received in previous minor frames.

The BC Manager maintains a command queue for each of the subsystems, as well as one for itself. At the time in the schedule that a subsystem may receive a command, the BC Manager checks the queue for commands that may be sent. The BC Manager collects telemetry according to a fixed schedule. When a packet is complete, it formats the data and issues it to the software bus. Sometimes it posts a RTOS wake-up call for the receiving task.

BC Manager implements a MIL-STD-1553B interface that is tailored to each of the subsystems. Command and telemetry formats and rates vary among the subsystems. This served to greatly complicate the BC manager. Dedicated code is written to assemble some subsystem housekeeping where the IDS must search through the subsystem data in order to build a telemetry packet. The same is true for instances in which the IDS must interpret subsystem commands and send them on to the subsystem in different format and multiple steps. A series of tables was generated to control most BC manager functions.

The BC manager updates IDS telemetry with information concerning the state of the IDB and the subsystems, as well as the state of the BC manager itself and the commands it has processed.

```
rule DET1CURRMONITOR
  subsystem DET1
  category RT04
  priority 27
  activation YES
  continuous NO
  if (DET1CUR >= 184) then -- 184 => 350 mA
    msg "RULE DET04 I_DET1CUR=", I_DET1CUR
    deactivate DET1CURRMONITOR
    exec DET1CURRMONITOR_SCRIPT in 0 ticks
      priority = 27
  end if
end DET1CURRMONITOR
```

Figure 4. Example of a Rule

Real-Time Executive

The RTE task provides the IDS with scripting and rule-based autonomy capabilities. RTE scripts, rules, and database items are stored in the script and rule database (see Figure 3). An RTE rule is evaluated each time the RTE receives a notification from DataIO that a database item associated with the rule has changed. At this time, the RTE checks the premise of the rule, and executes associated commands if it is satisfied. See Figure 4 for an example of a rule.

RTE scripts may be scheduled to execute immediately, at a specific time, or at a relative time. They may be scheduled in real time or by rules or scripts. The scripts and rules may contain embedded packets that are placed on the SWB. This provides a way by which subsystem commands may be issued to the SWB for delivery to IDS tasks or instrument subsystems (the first way being the up-link to the IDS RT Manager). See Figure 5 for an example of a script.

```
script DET1CURRMONITOR_SCRIPT
  wait 2 seconds
  if (I_DET1CUR >= 184) then
    msg "PERSISTANCE MET I_DET1CUR=", I_DET1CUR
    cexl I_DET1HVPWROFF
    wait 1 second
    cexl I_DET1HVGRIDOFF
  else
    msg "PERSISTANCE NOT MET I_DET1CUR=", I_DET1CUR
  end if
  activate DET1CURRMONITOR
end DET1CURRMONITOR_SCRIPT
```

Figure 5. Example of a Script

The RTE is largely a COTS product; however, the interface to the SWB, some dedicated memory management functions, command handling and housekeeping telemetry updating capability are required to port the COTS product to the IDS. The RTE provides extensive and complicated software capability to the IDS with a minimal investment in testing beyond the interface software. Owing to the isolation of the RTE on the SWB, significant testing could be done using only DataIO, the RT Manager, and the BTE.

DataIO

DataIO supports the RTE by monitoring incoming telemetry, updating the flight database when changes in the incoming telemetry are significant, and notifying the RTE via SWB messages when these changes occur. The software that performs this is the COTS product together with a wrapper that integrates it with the rest of the IDS. The COTS DataIO does the actual de-commutation of the packet, based on the packet ID, the de-commutation record for the packet (see Figure 7), and the parameters for the database item (see Figure 6). The definition of the data includes the data type, size, conversion polynomials, and limits at which alarms are generated. The data type determines what other parameters are included in the record. For example, discrete database items will not have a conversion polynomial specified.

```
data COFM, 307, "I_DET1AUXCUR_COFM"
        numOfCoeffs = 6
        Coef = 1.99672
        Coef = 3.58939
        Coef = -0.0459486
        Coef = 0.000431326
        Coef = -1.74405e-006
        Coef = 2.63799e-009
end COFM
data ULAS, 306, "I_DET1AUXCUR"
-- DESCR: "DET1 Aux Power Current"
        flags = 8192
        rawValue = 0
        engValue = 0
        highRed = 200
        highYellow = 100
        lowYellow = -1
        lowRed = -2
        coeffID = 307
end ULAS
```

Figure 6. Flight Database Record

```
data DCMC, 57, "I_DET_T28_HSK"
  flags = 3
  numRecords =    3
    recordID = 304
    dataStructure = 2
    pointerIncrement = 30
    signExtend = 0
    mask = 65535
    lshift = 0
    order = 0
    recordID = 306
    dataStructure = 1
    pointerIncrement = 41
    signExtend = 0
    mask = 255
    lshift = 0
    order = 0
    recordID = 308
    dataStructure = 2
    pointerIncrement = 42
    signExtend = 0
    mask = 65535
    lshift = 0
    order = 0
end DCMC
```

Figure 7. Flight De-Commutation Record

Wrapped around the COTS product is software that provides an interface to the IDS. This includes an interface to the IDS SW bus, software to format the incoming telemetry packets such that the COTS DataIO can interpret it, and software to handle DataIO configuration commands. So long as the IDS flight DataIO has been configured to de-commutate a given packet, telemetry items within the packet may be added, deleted, or modified by the upload of new de-commutation and database records. Dedicated flight code changes are required to add new packet types to those that are being de-commutated.

Because DataIO is the task that handles telemetry, it was the obvious task in which to handle the control of the telemetry down-link modes and data rates. It was also a good task to place a number of smaller jobs that required access to telemetry, or that simply did not fit in any other location. Therefore, DataIO became a “catch-all” task that performs table-controlled telemetry down-link rate control, memory management, instrument thermal control, peak-up count level monitoring, IDS memory load and dump, table dumps, PHA calculations, memory scrubbing, and numerous other small tasks.

DataIO is awakened every 40 ms. It has its own internal schedule that governs what it does during that 40-ms frame. It must finish the tasks assigned to that frame during the allotted time.

Detector Manager

The detector manager is a compact task that is designed to read science data from the detectors and then process and store the data as fast as possible. The IDS has a requirement to handle 32,000 samples/second without losing science data. The hardware FIFO through which science data arrive is 256 samples deep. This means that the detector manager must be able to read and process 128 samples before the end of each minor frame. If it does not, science data may be lost. The detector manager is the second highest priority task.

The SWB interface (command interface) for the detector manager is handled by DataIO. DataIO also manages the memory into which the detector manager writes science data.

Guidance

The Guidance and Star Identification (StarID) tasks work together to fulfill the attitude processing requirements of the IDS. The Guidance task does the bulk of the attitude processing (slewing, image processing, guiding, and peak-up). The Star Identification task was formed to perform the star identification for the Guidance task. Star identification needs to run concurrently with the Guidance task's performance of unidentified tracking. By locating Star ID in a separate task, the RTOS could be used to provide the multitasking necessary without significant software development effort.

The Guidance and Star Identification tasks were designed to give the controllers of the instrument maximum script control over the fine pointing capabilities. A “tool box” approach was implemented. A typical acquisition of science data would occur as shown in Figure 8. After each step (coarse slew, image acquisition, etc.) the success/failure status is made available in the SCL database so that the script may, if so written, intelligently choose the next step. For example, if image acquisition fails, the script may try again before moving on to processing the image.

```
Target Acquisition
  Script command to Guidance:  Perform Coarse Slew
  Script command to Guidance: Get Image
  Script command to FES: Configure and get image
  Script command to Guidance: Process image
  Script command to Guidance: Guide on unidentified stars
  Script command to Star ID: Identify stars
  Script command to Guidance: Guide on identified stars
  Script command to Guidance: Perform peak-up (perform
    table-driven series of small slews, and with DataIO,
    select the position at which the highest count of
    detector events is collected)
  Script command to Guidance: Adjust FPA, fine slew
Exposure
  Scripted commands to detectors, IDS bulk memory manager,
    and detector manager to collect science data
```

Figure 8. Fixed Target Exposure Sequence

Further flexibility is achieved in the Guidance and Star ID tasks by using tables of parameters to control many of the functions. Among others, these tables include image processing parameters, attitude estimation parameters, optical distortion parameters, and a peak-up configuration table.

Guidance works in six modes: idle, slewing, image processing, unidentified tracking, identified tracking, and moving target tracking. In image processing, the Guidance task may be occupied for multiple seconds. Because the real-time constraints on guidance are not so limiting as on many of the other tasks, the guidance task is sixth in priority.

Star Identification

The Star Identification task performs only the identification of the stars. It has two modes, on and off. Once the Guidance task completes processing a FES full image, this task receives the centroids extracted from the full image. It identifies the star field by comparing the centroids to reference star positions specified in an uploaded star table. Uploaded parameters control aspects of the star identification, such as minimum and maximum angular separation of star pairs, the minimum number of stars that must be matched for a successful identification, and the desired number of matches (after which the search is stopped). When the identification is complete, an SCL database item is set to indicate to the controlling script the success or failure of the identification.

Star identification can take tens or hundreds of seconds to execute. It is given lowest priority of the major IDS tasks.

The isolation of the Star Identification and Guidance tasks on the SWB allowed for complete task-level testing on a development PC running a BTE version of the SWB. For these tasks, integration on the target system was required only for system-level testing.

Conclusion

We have presented a description of the IDS flight hardware and real-time embedded flight software. The IDS hardware design includes redundancy, highly radiation-tolerant static random-access memory (SRAM), EDAC, and fault-tolerant MIL-STD-1553B buses. The software is built on a proven RTOS and uses a small number of prioritized tasks to meet its functional requirements.

Acknowledgements

H. Landis Fisher, Patricia K. Murphy, and Micheal J. White of APL contributed the guidance and image processing software. Jim Van Gaasbeck (ICS) provided science processing software and the port of SCL to the IDS platform. Bruce A. Scherzinger of APL contributed the boot code and MIL-STD-1553B control software. Joshua F. Offutt (ICS) and Pat Cappeleare (ICS) were the primary BTE developers.

Robert C. Moore is the FUSE IDS lead system engineer. Brian K. Heggestad is the FUSE IDS lead software engineer.