CSR	MASSACHUSETTS INSTITUTE OF TECHNOLOGY CENTER FOR SPACE RESEARCH CAMBRIDGE, MASSACHUSETTS 02139	
REVISION LOG	TITLE: Boot BEP	DOC. NO. 36-53231

Revision	Date (mm/dd/yy)	ECO No.	Page(s) Affected	Reason	Approval
A	05/13/96	36-630	MOST See change Bars	UP DATE TO Rev A	[illegible] 5/31/96

24.0 Boot BEP (36-53231 A)

24.1 Purpose

The boot code provides the BEP with the capability to 1) boot from the Flight Software installed in memory; 2) to install code delivered from uplink and to begin executing at a selected location.

24.2 Uses

The Boot BEP provides the following feature:

Use 1:: Provides a means to initiate executable code on the BEP from the Flight Software.
Use 2:: To load and execute code from command packets.

24.3 Organization

24.3.1 Command Types

The booting of the BEP is normally controlled by external commands, either a power-on reset, or a discrete commanded reset. The latter command consists of two types, boot from memory and boot from uplinked command. An abnormal reset, triggered by the watchdog resetting, is expected when there has been an anomalous response from the running software. [1]

During boot, there is no capability for the software to echo commands to the ground. The setting of the BiLevel Discrete Downlink bits (LEDs on non flight boards) is used to indicate progress during the boot process. Their state is periodically sampled by the hardware at predetermined intervals. [2] The software "state" will continue to be reported by the main software after boot has transferred control. Refer to Section 24.6 Discrete Telemetry Status.

The standard mode of operation will use load from boot software in memory. The load from uplinked command is provided to allow ground to take over the BEP for maintenance, for debugging problems, for creating a functionality to perform a task not envisioned when the instrument was designed.

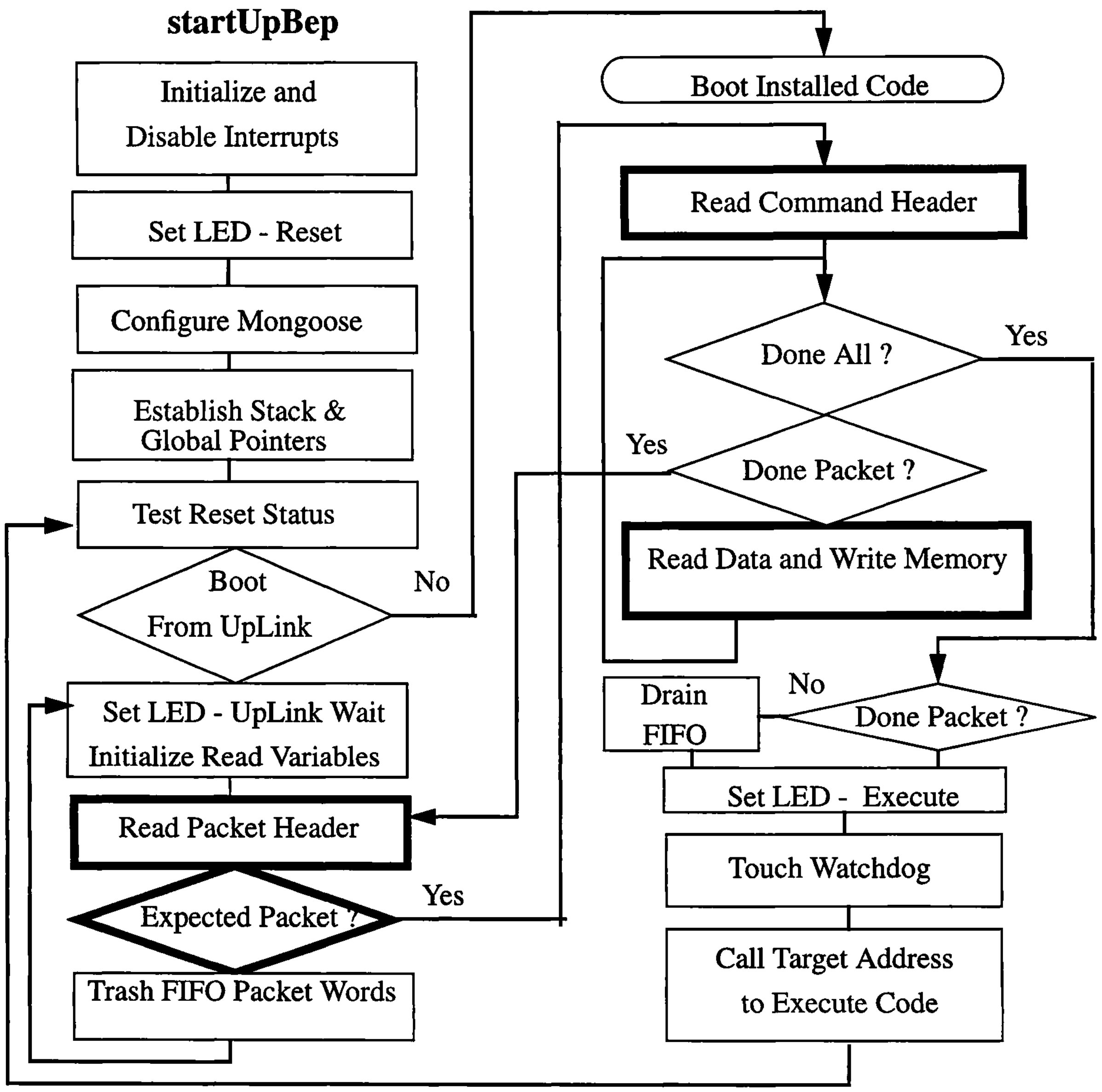
24.3.2 Boot Procedure

The boot procedure first manages the hardware interface including setting the BiLevel Discrete Downlink bits (LEDs) to indicate a reset, it then determines if a load from uplink is planned. If it is, the ground will have notified the instrument via the hardware command port, and the hardware will have set the BEP status register boot mode bit. If not loading from uplink, a standard boot

1. The hardware circuitry is described in *DPA Hardware Specification & System Description* Rev. B section 2.1.2.2
2. The Bilevel telemetry is described in *DPA Hardware Specification & System Description* Rev. B section 2.1.2.6.2

from memory will occur. Figure 114 illustrates the flow of control of the BEP boot procedure. The bolded boxes indicate functionality expanded in later figures.

FIGURE 114. BEP Boot Function Flow of Control

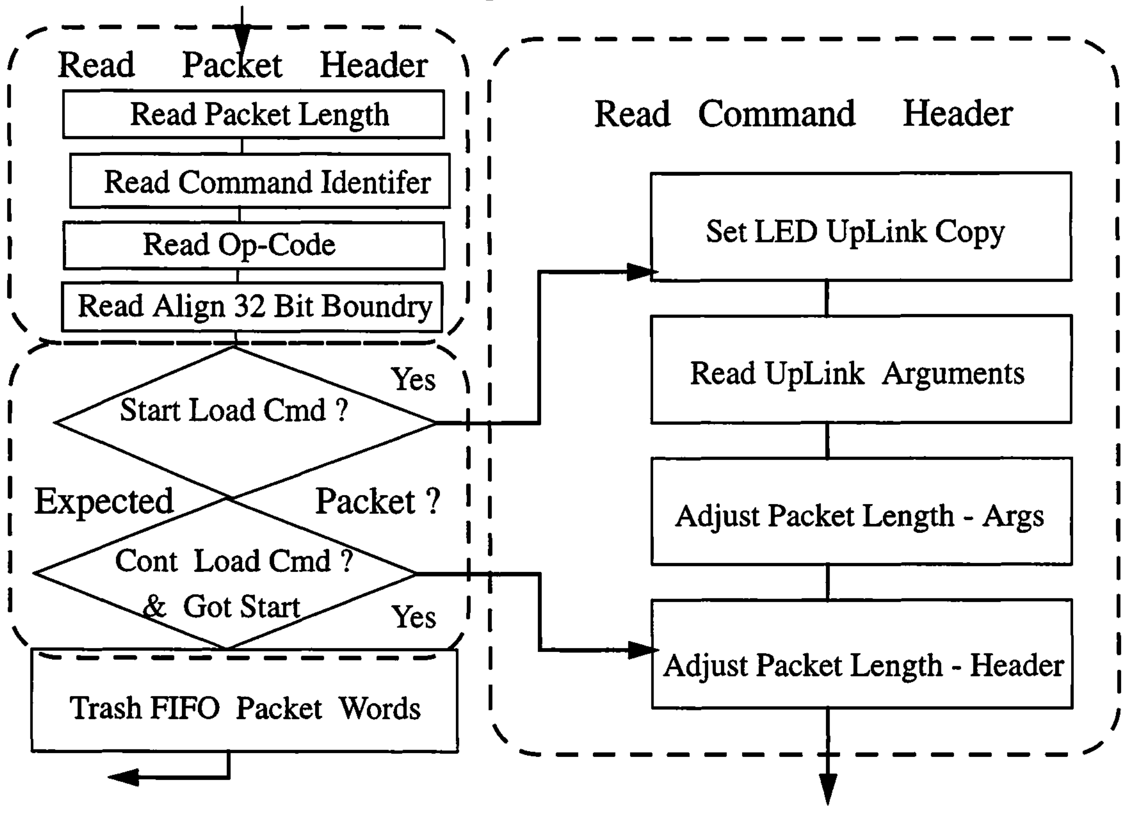


24.4 Scenario - Boot from Uplink Command

When booting from uplink the Uplink Wait LEDs are set, then the procedure will begin a loop which interrogates the FIFO as the commands arrive. During this cycle, the watchdog is handled. As the first command arrives the LEDs will be reset to Uplink Copy, indicating that data is being processed. The function will examine the packet header operation code and extract processing arguments from the command header. Unanticipated command packets will be discarded. The function will determine whether it should load data to memory or should execute code at an assigned address. It will load data to memory, cycling to obtain additional packets of data as necessary. Should the data load request be fulfilled before the last packet is empty, the remainder of that packet's data will be drained from the FIFO. The LEDS will be reset to Uplink Execute showing that execution is beginning. The watchdog will be touched, giving the new process the maximum time before it must handle that task. Should the executing process return, the boot process will branch back to reinitialize the mongoose, and re-examine the BEP Boot Mode state.

The process will read the FIFO until it obtains the first packet word, which will contain the packet total length. It will read and ignore the command identifier, then read the op-code and determine whether it is a Continue_Load or a Start_Load command. Other commands are read from the FIFO and ignored. The process will read the align word from the FIFO. Refer to Figure 115.

FIGURE 115. Read Packet Header With Expected Packet Decision and Read Command Header

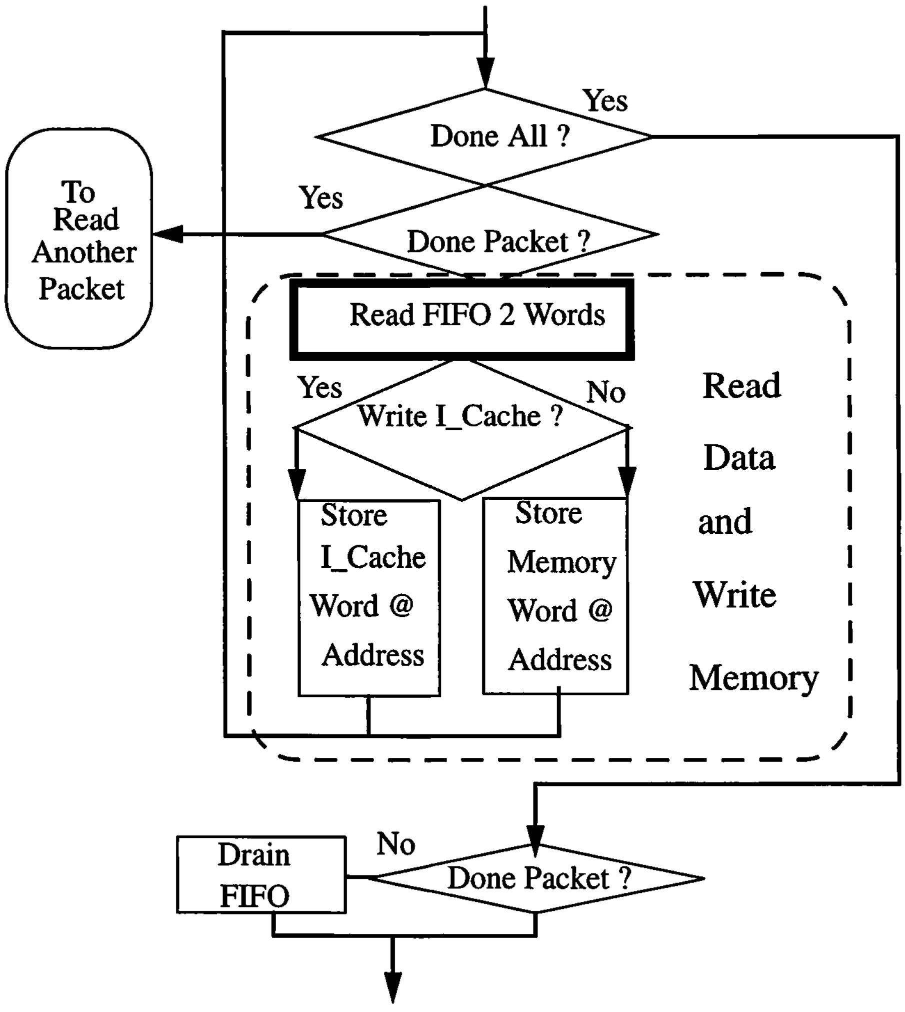


A Start_Load command will cause the LEDs to be set to indicate that an Uplink Copy is in progress. It will then read the total count of words to be written into memory and the address where execution is to begin after the data has been loaded. The packet length will be adjusted prior to advancing to a loop which reads the data and writes it to memory.

A Continue Load packet must be preceded by a Start_Load packet. When a Continue_Load command is recognized, the packet length is adjusted for the packet header size, before advancing to read the data and write it to memory. An unexpected packet op-code will result in trashing of that packet and restoration of the Uplink Wait (LED) state which requires a new Start_Load command before a Continue_Load command will be written to memory. An interviening Start_Load command takes precedence, and will load data as directed. This stratagem permits loading data to several locations prior to execution at its target address.

After having read the header, the process proceeds to the decisions concerning processing the remaining data. The total amount of data may have been loaded (including zero words). The amount of data in the packet may have been loaded, else the process will look for another packet. With more data to be loaded, the process will obtain the next word to be installed in memory. Refer to Figure 116. The bolded box indicates functionality expanded in the later figures.

FIGURE 116. Read Packet Data and Write It to Memory

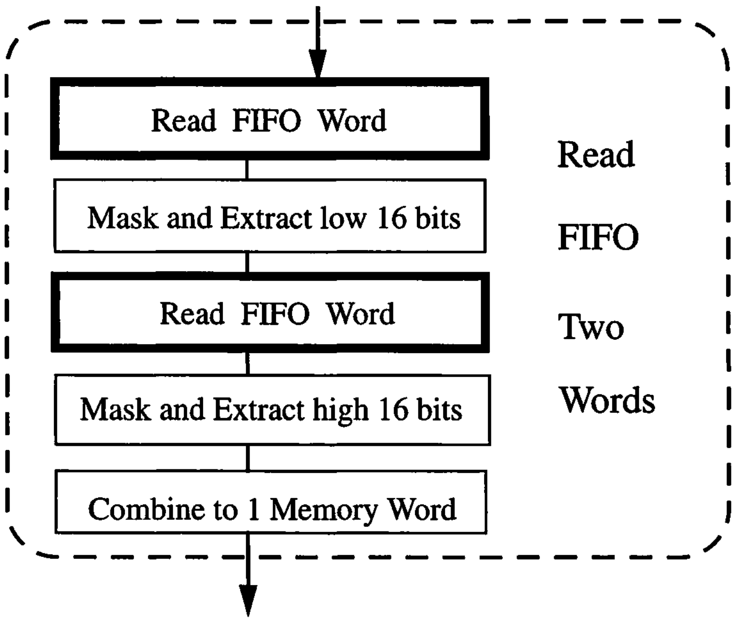


The I-Cache can not be written to directly. D-Cache and general memory can be written directly. To load to I-Cache the data to write, and the address to be written, must be delivered to hardware registers and the write will be accomplished. For this reason, the load address is tested and the data is then loaded in the appropriate manner. The process will return to evaluate the need to read and deposit another word.

After the Total number of words have been loaded, the process will check that the current packet has been emptied, else it will drain the FIFO of the residual words before advancing to begin execution.

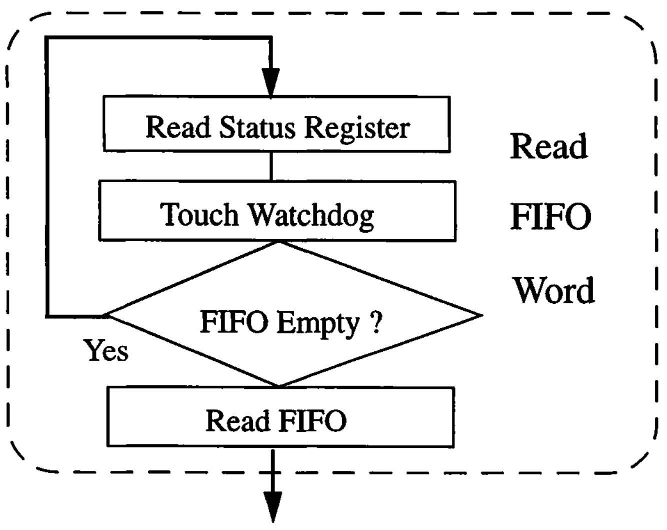
Each FIFO word contains 16 bits of data in the low two bytes, in addition to flags in the upper portion. When reading data to load into memory, two packet (FIFO) words are combined to create a single word to load into memory. Refer to Figure 117. The bolded box indicates functionality expanded in a later figure.

FIGURE 117. Read a Memory Word



The FIFO may or may not have an available word to be read every time one is requested. For this reason the process will read the FIFO status register. Then it will touch the watchdog to prevent a reset, and will test the Uplink-FIFO-Empty status bit to determine if a word may be transferred. It will continue to cycle until a word is available. It will read the word, then return to the calling function. Refer to Figure 118.

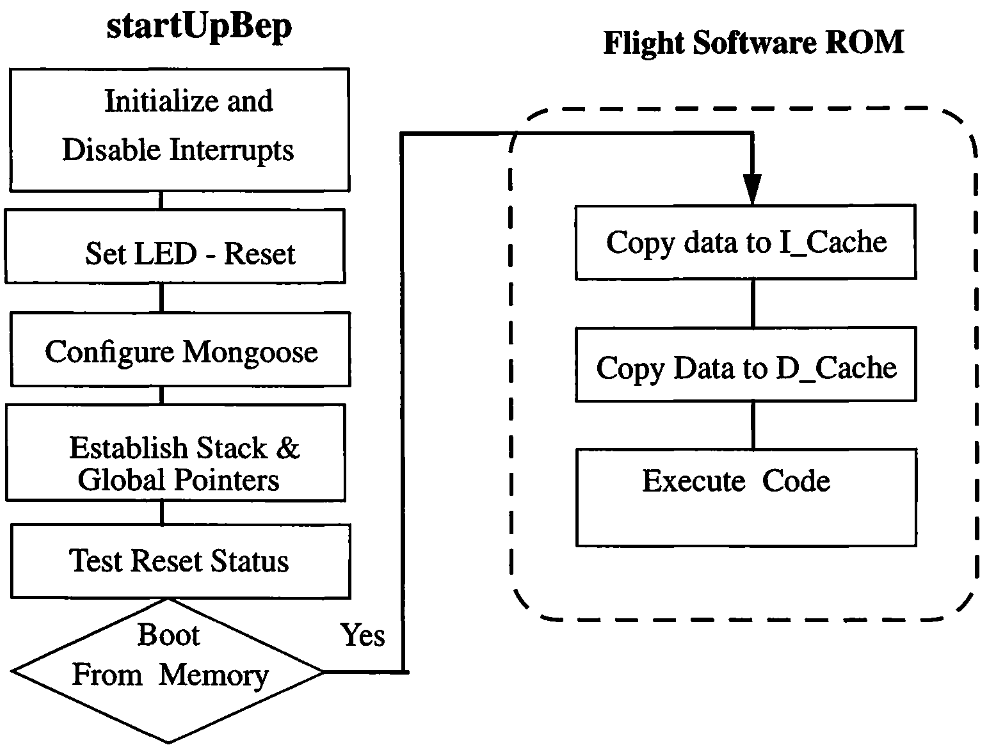
FIGURE 118. Read a FIFO Word



24.5 Scenario Boot from Flight Software in Memory

After initializing the mongoose, the process tests the reset status. If the mode indicates boot from FS in memory, control jumps to that memory location. Code stored in memory copies information into I-Cache (code) and D-Cache (data). It will then jump to the execution address. Execution from the FS memory is not expected to return. Refer to Figure 119.

FIGURE 119. Boot from Memory



24.6 Discrete Telemetry Status

The Back End's Boot from installed software and main application software use the four discrete telemetry bits to indicate the current status of the instrument software. These four bits allow the software to assign up to 16 different status values. These bits are sampled by the hardware once per Telemetry Major Frame.

The Boot software uses these bits to help a maintainer diagnose lockup problems during Boot. The Boot program asynchronously sets these bits to specific values at different stages during the boot. If the instrument hangs during the boot, the status bits will indicate which stage caused the hang. The stages and values are assigned as shown in Table 28.

TABLE 28. Boot Software Discrete Telemetry Value Assignments

Stage	Value	Description
Reset	15	The Back End Boot Code sets the status to this value upon reset of the Back End Processor. This value allows the maintainer to detect a lockup before the Boot process had a chance to detect the type of boot to be performed.
Memory Copy	14	The Back End Boot Code sets the status to this value prior to copying code and data from the Back End's Software. This value allows the maintainer to detect a failure while copying code and data from memory.
Memory Execute	13	The Back End Boot Software sets the status to this value prior to executing code loaded from the Back End's Software Memory. This value allows the maintainer to detect a lockup when the loaded code is executed.
Uplink Wait	12	The Back End Boot Code sets this value prior to polling the Command Interface for an "Start Load From Uplink" command. This value allows the maintainer to determine when the instrument is waiting for code and data from the uplink interface.
Uplink Copy	11	The Back End Boot Code sets this value after receiving the "Start Load From Uplink" command packet. This value allows the maintainer to determine that the instrument has received the first load command, and is in the process of copying its code and data, and waiting for subsequent "Continue Load From Uplink" commands.
Uplink Execute	10	The Back End Boot Software sets this value after receiving and loading all code and data from the Command Interface, and is about to execute the loaded code. This value allows the maintainer to detect a lockup when the loaded code is executed.

The loaded Back End software uses the status bits to help a maintainer determine the cur rent state of the instrument software. During its initialization stage, the Back End software sets the status bits to specific values to help a maintainer diagnose a hang during the initialization stage. After initialization, the Back End software periodically toggles the status between two values, depending on the current "state" of the software. Table 29 lists the values used by the Back End software.

TABLE 29. Main Software Discrete Telemetry Value Assignments

Stage	Value(s)	Description
Patch Application	9	The loaded Back End software sets the status bits to this value prior to applying any software patches. This allows the maintainer to determine if the Back End hangs while installing patches.
Startup	8	The loaded and patched Back End software sets the status bits to this value prior to executing its module initialization routines. This allows the maintainer to determine if the Back End hangs during its main initialization process.
Idle	7,6	The Back End software periodically toggles between these two values, while running when a Science Run is not being performed. This allows a maintainer to determine that the instrument is active, but not performing any science operations.
Science	5,4	The Back End software periodically toggles the status bits between these two values, rate, when executing a Science Run. This allows a maintainer to determine that the instrument is executing a Science Run.
WD Idle	3,2	If the Back End recovers after a Watchdog Reset, the software uses these values when it is in an "Idle" state. This allows a maintainer to determine that a Watchdog Reset occurred.
WD Science	1,0	If the Back End recovers after a Watchdog Reset, the software uses these values when it is performing a "Science Run." This allows a maintainer to determine that a Watchdog Reset occurred.