CSR

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CENTER FOR SPACE RESEARCH
CAMBRIDGE, MASSACHUSETTS 02139

REVISION LOG

TITLE: Boot FEP

DOC. NO. 36-53230

Revision	Date (mm/dd/yy)	ECO No.	Page(s) Affected	Reason	Approval
A	05/10/96	36-626	1169	UpGrade to RevA Fix typo Modify Register Store on Error (Bounce Exception)	5/23/96

40.0 Boot FEP (36-53230 A)

40.1 Purpose

The FEP is booted at the direction of a BEP function which commands and controls the process. The Boot FEP executable runs on the FEP. The FEP Manager function `loadRunProgram` might install `fepCtl` and `fepTimedBias` et. al. to initiate a science run. Refer to the FEP Manager *Section 25.5.2 Use 2: Reset, load, and run a program on a Front End Processor.*

Boot code is copied by the BEP into shared FEP bulk memory[1]. The boot code executing in the FEP bulk memory, will provide the BEP with the capability to deliver read, write, and execute commands to the FEP via a mailbox. With these commands the BEP may complete loading of the executable code into the FEP I-cache and begin execution of that Front End Processors stand alone code.

The procedures used to respond to the BEP directives are: `startUpFep`, `bootServerFep` and a (separate) copy of the FEP IO Library. No other code is active. When execution of a science run is commanded, the thread of control is passed to the function indicated and this boot code is effectively abandoned.

Note: This mechanism is not restricted to initiating science processing. Any code may be copied to the FEP, any code may be overwritten, and any code loaded may be executed and that code may or may not return. Since the thread of control is passed to an executing function, the responsibility to handle the watchdog is also passed to that function.

40.2 Uses

The Boot FEP provides the following feature:

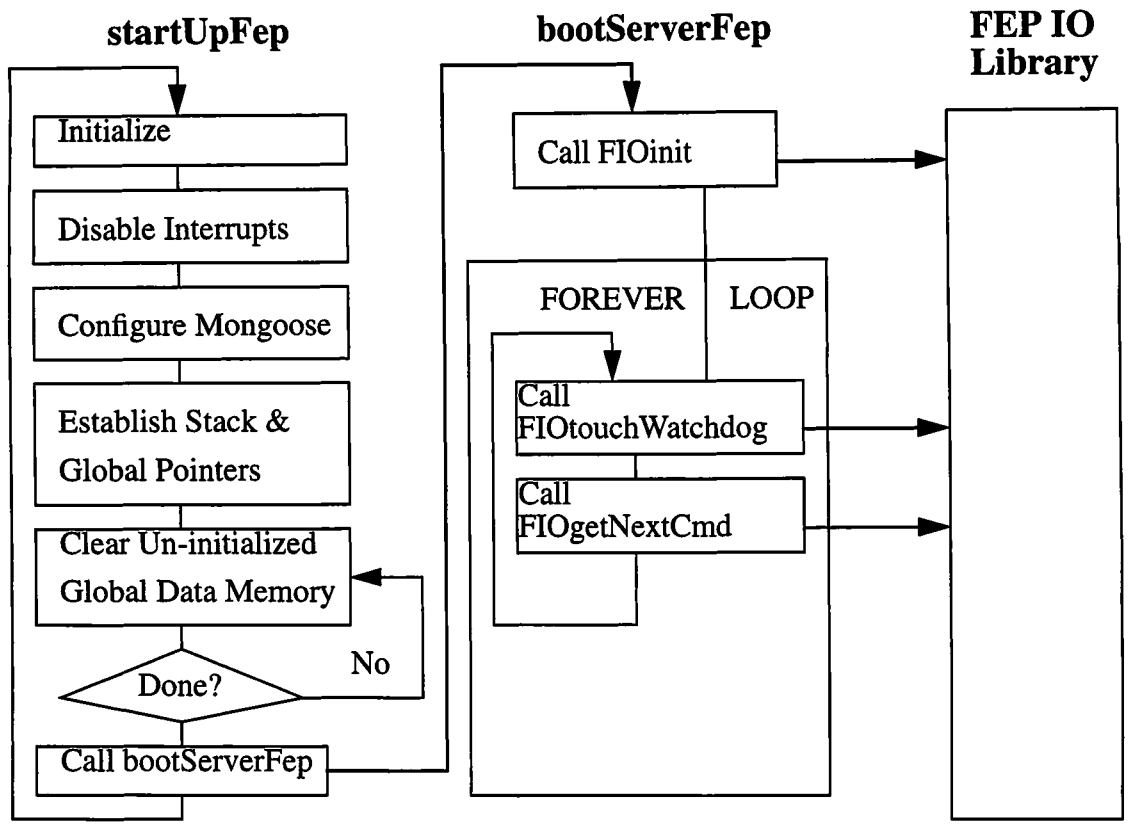
Use 1:: Provides a means of initiating executable code on the FEP.

40.3 Organization

Figure 176 illustrates the relationship between functions used by the FEP boot procedure.

1. The BEP/FEP interface is described in *DPA Hardware Specification & System Description* Rev. B section 2.1.2.10

FIGURE 176. FEP Boot Function Relationships



The FEP boot uses the FEP IO Library to fulfill commands entered into the incoming MailBox.

- The `startUpFep` routine establishes the mongoose hardware states needed to execute C code.
- The `bootServerFep` routine starts by calling `FIOinit` to set up the structures and pointers the FEP IO Library needs to provide communication from the BEP.
- It then loops FOREVER using `FIOtouchWatchdog` to reset the watchdog counter thus keep the watchdog from resetting the CPU, and using `FIOgetNextCmd` to respond to commands which will read, write, or to execute FEP memory.

40.3.1 Memory Map Requirements

The hardware locations of the MicroBoot Word (0xBFFFFFFC), reset vector (0xBFC00000), and general exception vector (0xBFC00180) are specified in section **2.2.2.3.1** of ***DPA Hardware Specification & System Description***, **Table 10. FEP Memory Map** (Rev B). (The MicroBoot Word is aliased to 0xA017FFFC.) The linker must be constrained to establish the beginning of this boot code at the reset vector. When the reset line is released, execution will begin at the reset address.

With interrupts disabled, should a hardware exception be detected, control will pass to the exception vector. Code has been provided to copy the contents of relevant registers to shared memory. The client process may interrogate it to discern and record the cause of failure.

The MicroBoot word contains a "sequence to initialize critical configuration parameters that are not accessible under software control."[1] Located at 0xBFFFFFFC, it is initiated upon reset.

40.4 Scenario

40.4.1 Pre-Boot Tasks

The functions `startUpFep` and `bootServerFep` are copied into FEP shared memory by the BEP while holding the FEP in a reset state. When released, the FEP will reference its microboot address and begin executing the `startUpFep` code.

40.4.2 Boot the FEP

`startUpFep` will initialize registers: setting kernel mode, disabling interrupts, mapping the general exception vector, setting co-processor 0 to usable, clearing the interrupt mask, and clearing the software interrupts in co-processor 0. (There are no additional co-processors.) The mongoose is then configured with no DMA and running with a single wait state. Because the stack size is declared during initial compilation, the stack pointer address is located at the maximum offset (minus this function's save area of 24 bytes). The global data pointer (gp) is assigned to the linker defined symbol _gp.

C program specifications expect the global data area (.bss) to be cleared. The address of the first word and the end have been provided by the linker. The process enters a loop which will write zeros into those memory locations. The process then calls the monitor function `bootServerFep` to service commands from the BEP. Should that function ever return, `startUpFep` will continue execution from its first statement.

In the event that an anomalous condition is encountered which transfers control to the exception vector; `startUpFep` will copy the FEP CPU cause register, and the Coprocessor status, cause, and exception program counter registers to FAULT_DATA in shared memory. The watchdog will be set to expire quickly.

40.4.3 Monitor the Watchdog and Fulfill BEP Commands

`bootServerFep` establishes a buffer for commands and for the FIO`getNextCmd` status variable. It first calls `FIOinit` to initialize the FEP IO Library pointers and structures used in communicating commands from the BEP mail box. Next it enters a forever loop in which it calls `FIOtouchWatchdog` to keep the mongoose watchdog counter from reaching zero and resetting

1. Mongoose ASIC Microcontroler Programming Guide; 8.5 MSF Microboot: Brian S. Smith

the CPU and calls `FIOgetNextCmd` to have the library functions interrogate the BEP mailbox for any read, write, or execute in memory commands, to which other library functions will respond and fulfill. Commands other than those mentioned are not honored. Usually the BEP uses these functions to complete loading of memory and to begin executing a loaded function. Refer to FEP IO Library Section 39.0 for a more detailed description of the library functions.

40.5 Specification

40.5.1 bootServerFep()

Scope: Initialization

Return Type: void

Documentation:

This function will use the FEP IO Library functions to establish communications from the BEP and to respond to BEP read, write, or execute in memory commands.

Preconditions:

The FEP mongoose and its co-processor must have been initialized and booted.

Semantics:

This function will use the FEP IO Library function `FIOinit` to establish mailbox communications from the BEP. It then enters a FOREVER loop in which it calls `FIOtouchWatchdog` to prevent the watchdog from counting down and resetting the CPU, and calls `FIOgetNextCmd` to respond to BEP commands to read, write, or to execute in memory.

Other BEP commands are ignored. They will not generate a mailbox response.

Postconditions:

This function never returns.

40.5.2 startUpFep()

Scope: Boot

Return Type: void

Documentation:

This assembly language function establishes the necessary initial conditions on the FEP.

Semantics:

This function initializes registers, loads the mongoose configuration, initializes the stack and global pointers, clears the global data memory (.bss), and transfers control to `bootServerFEP`. Should that function return, `startUpFep` will restart.

In the event that an anomalous condition is encountered which transfers control to the exception vector; `startUpFep` will copy some significant registers to FAULT_DATA in shared memory. The watchdog will be primed to expire.

Postconditions:

This function never returns.