CSR

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

REVISION LOG

TITLE: Software Detailed Design
Command Device

DOC. NO. 36-53208

Revision	Date (mm/dd/yy)	ECO No.	Page(s) Affected	Reason	Approval
-	4/23/95	-	-	Initial version for design walkthrough	
A	5/3/95	36-230	all	Incorporated review comments	5/4/95

8.0 Command Device (36-53208 A)

8.1 Purpose

The purpose of the Command Device is to provide access to Back End's Command Interface logic and to the Command FIFO.

8.2 Uses

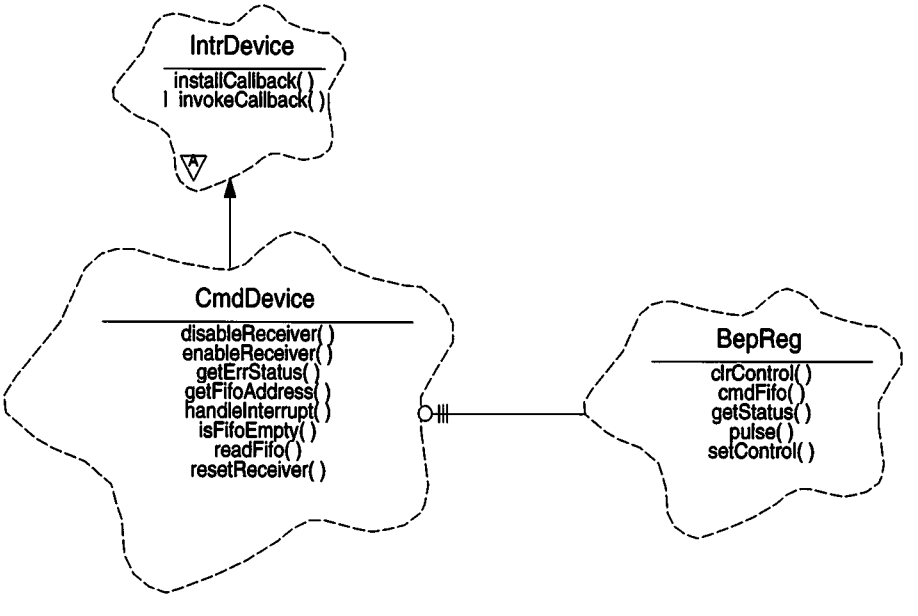
The Command Device class, **`CmdDevice`**, provides the following features:

Use 1:: Disable and reset the command logic and FIFO
Use 2:: Obtain the state of the command FIFO
Use 3:: Enable the command reception logic
Use 4:: Read words from the command FIFO
Use 5:: Obtain the virtual address of the command FIFO
Use 6:: Handle command interrupts, forward control to an installed callback instance.

8.3 Organization

The **`CmdDevice`** is an interruptible device, and is therefore a subclass of **`IntrDevice`** (see Section 6.0). This class relies on the **`BepReg`** class to provide access to the Command hardware control logic and command FIFO.

FIGURE 19. Command Device Class Relationships



`CmdDevice`- This class represents the Back End's Command Device logic and FIFO. It provides functions which control the state of the command reception hardware logic (`disableReceiver`, `enableReceiver`, `resetReceiver`), and provides access

to the Command FIFO (`getFifoAddress`, `isFifoEmpty`, `readFifo`). In addition to theses functions, it inherits the ability to install and invoke interrupt callback instances from **`IntrDevice`**.

`IntrDevice` - This is an abstract class which defines the common interface to all types of interruptible devices. It is used by the Interrupt Controller (not shown) to dispatch control to interruptible devices, and by client code to install callback functions. Child classes of **`IntrDevice`**, including **`CmdDevice`**, may use their parent's protected method, `invokeCallback`() to invoke the installed callback instance (see Section 6.0).

`BepReg`- This class represents the lowest level hardware access to the features provided by the Back End hardware control, status, and pulse registers, and to the Command FIFO. See Section 5.0 for a description of this class.

8.4 Command Logic Description

The Back End's command logic provides the ability to receive entire command packets into the command FIFO and generate an interrupt once the entire packet has been received. This significantly reduces the number of interrupts and the rate of interrupts that the Back End Processor must handle in order to receive a command packet.

Once enabled, the command logic interprets the first received 16-bit command word as the count of the total number of words in the packet, including the received length. It then writes the length into the Command FIFO, and proceeds to receive and store subsequent words from the packet until the last word of the packet has been received. Once the last word is in the FIFO, the hardware generates a command interrupt. The hardware then interprets the next 16-bit words as the length of the subsequent packet.

In the event that the length is out-of-range, or if the command FIFO fills when a packet is being received, the command logic will generate an interrupt, and let the software cope with the condition. The condition is indicated in the Back End's Status register.

Since command transfers and error handling require access to buffer pools and intelligent systems-level decision making, the **`CmdDevice`** class doesn't explicitly handle the transfer of commands from the FIFO into memory, or these error conditions, but instead, provides the low-level access routines used by the higher level protocol class (in this case, the **`CmdManager`**) to deal with these actions.

8.5 Scenarios

8.5.1 Use 1: Disable and reset command logic and FIFO

In order to support system initialization, and recovery from errors, the **CmdDevice** provides functions which allow the client to disable the command reception logic, `disableReceiver()`, and to reset the command FIFO, `resetReceiver()`. The client code uses these routines during initialization and error recovery to stop the hardware logic from generating command interrupts due to erroneous packet lengths, and to quickly discard the contents of the Command FIFO, respectively. In general, the client code should call `disableReceiver()` prior to resetting the contents of the Command FIFO.

8.5.2 Use 2: Obtain the state of the Command FIFO

In order to allow a client to ensure that the FIFO is empty prior to enabling command reception, or to ensure that the FIFO contains valid data prior to reading and interpreting its contents, the **CmdDevice** provides a function `isFifoEmpty()` which returns whether or not the FIFO currently contains any data. In order to allow client code to detect errors, it also provides a function, `getErrStatus()`, which returns the current error status of the Command Logic.

8.5.3 Use 3: Enable command reception

Once the client is ready to receive commands, it uses the **CmdDevice** `enableReceiver()` function to enable the command reception logic, which in turn will start generating command interrupts upon each reception of a command packet.

8.5.4 Use 4: Read Command FIFO

In order to allow a client to obtain the length of a command packet prior to starting a block transfer from the FIFO, the **CmdDevice** provides a function `readFifo()` which returns the next word(s) from the Command FIFO.

8.5.5 Use 5: Access Command FIFO Address

In order to allow the client code to perform block transfers from the Command FIFO (possibly by using the Mongoose DMA), the **CmdDevice** provides a function which returns the virtual address of the FIFO, `getFifoAddress()`.

8.5.6 Use 6: Handle command interrupts

Whenever a packet has been placed into the Command FIFO, or an error is detected by the Command Logic, a Command interrupt occurs. Once the Interrupt Controller (not shown in Figure 19) determines that the Command interface caused the interrupt, it masks off any lower priority interrupt causes and re-enables interrupts. It then dispatches control to the

CmdDevice class's `handleInterrupt()`. The **CmdDevice**'s `handleInterrupt()` function then resets the interrupt cause, and invokes the installed interrupt callback instance. The callback then performs any client specific operations.

8.6 Class CmdDevice

Documentation:

The Command Device is responsible for providing an interface to the ACIS BEP Command hardware.

Export Control: Public

Cardinality: 1

Hierarchy:

Superclasses: **`IntrDevice`**

Implementation Uses:

`BepReg`

Public Interface:

Operations:

```
disableReceiver()
enableReceiver()
getErrStatus()
getFifoAddress()
handleInterrupt()
isFifoEmpty()
readFifo()
resetReceiver()
```

Concurrency: Synchronous

Persistence: Persistent

8.6.1 disableReceiver()

Public member of: **CmdDevice**

Return Class: **void**

Documentation:

This function disables the hardware packet word counter and prevents subsequent "Command Available" interrupts.

Semantics:

Use **BepReg**::`clrControl()` to de-assert the Uplink Enable bit.

Concurrency: Synchronous

8.6.2 enableReceiver()

Public member of: **CmdDevice**

Return Class: **void**

Documentation:

This function enables the hardware packet word counter, and subsequent "Command Available" interrupts.

Semantics:

Use **BepReg**::`setControl()` to assert the Uplink Enable bit.

Concurrency: Synchronous

8.6.3 getErrStatus()

Public member of: **CmdDevice**

Return Class: **enum CmdDevStatus**

Documentation:

Return the current status of the command device. The currently defined values are as follows:

CMDDEV_NOERR - No errors
CMDDEV_ERRLENGTH - Illegal packet length
CMDDEV_ERRFIFOSPILL -Command FIFO filled

Semantics:

Use **BepReg**::`getStatus()` to read contents of status register. If the FIFO Full latch is set, return CMDDEV_ERRFIFOSPILL. If the Command Error bit is set, return CMDDEV_ERRLENGTH, otherwise, return CMDDEV_NOERR.

Concurrency: Synchronous

8.6.4 getFifoAddress()

Public member of: **CmdDevice**

Return Class: **volatile const unsigned short***

Documentation:

This function returns a pointer to the Command FIFO. Access to the FIFO address allows the caller to perform block copies or DMA transfers from the FIFO.

Semantics:

Get and return the address of the Command FIFO using **BepReg**::`cmdFifo()`.

Concurrency: Synchronous

8.6.5 handleInterrupt()

Public member of: **CmdDevice**

Return Class: **void**

Documentation:

This function handles an interrupt from the command hardware.

Semantics:

Reset the interrupt by pulsing the Clear Uplink Interrupt bit using **BepReg**::pulse(), and then use **IntrDevice**::invokeCallback() to pass control to the client code.

Concurrency: Synchronous

8.6.6 isFifoEmpty()

Public member of: **CmdDevice**

Return Class: **Boolean**

Documentation:

This function determines if the Command FIFO is empty. If so, it returns *BoolTrue*, else it returns *BoolFalse*.

Semantics:

Get the contents of the BEP's status register using **BepReg**::getStatus() and test the FIFO empty bit.

Concurrency: Synchronous

8.6.7 readFifo()

Public member of: **CmdDevice**

Return Class: **void**

Arguments:

unsigned* *dst*
unsigned *wordcnt*

Documentation:

This function reads *wordcnt* words from the Command FIFO into the buffer pointed to by *dst*. It is up to the caller to ensure that the FIFO contains at least *wordcnt* words prior to calling this function.

Semantics:

Get the address of the Command FIFO using **BepReg**::cmdFifo(), remove each word from the FIFO and store into *dst*.

Concurrency: Synchronous

8.6.8 resetReceiver()

Public member of: **CmdDevice**

Return Class: **void**

Documentation:

This function resets the command device hardware, including the command FIFO. All data in the FIFO is lost.

Semantics:

Using **BepReg**::pulse(), clear any pending interrupts and errors by asserting the Uplink Interrupt Reset bit in the BEP's Pulse Register and reset the FIFO by pulsing the FIFO reset bit and clear latched error bits. This function clears any errors reported via getErrStatus().

Concurrency: Synchronous