CSR

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

REVISION LOG

TITLE: Software Detailed Design
Bad Pixel and Column Map Classes

DOC. NO.
36-53240 Rev. A

Revision	Date (mm/dd/yy)	ECO No.	Page(s) Affected	Reason	Approval
A	4/10/96	36-580	all	Initial version. Incorporated comments from initial review.	4/22/96

32.0 Bad Pixel and Column Map Classes (36-53240 A)

32.1 Purpose

The purpose of the Bad Pixel and Column Map classes are to maintain a persistent list of bad CCD pixels and columns within the instrument. Within the instrument, there is one bad pixel map, and two bad column maps. The bad column maps are respectively associated with Timed Exposure mode and Continuous Clocking mode. These lists are used by science processing to prevent damaged pixels from saturating telemetry with non-X-ray event data.

32.2 Uses

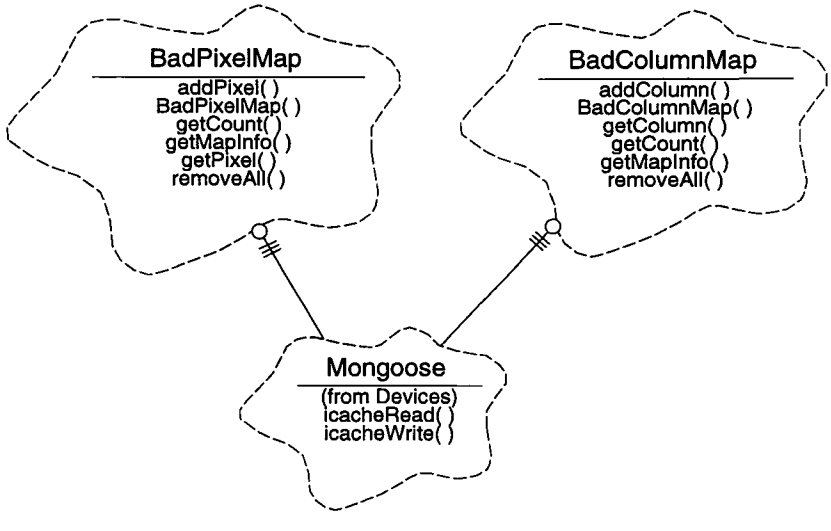
The following lists the use of the Bad Pixel and Column Map classes:

Use 1:: Append a bad pixel entry to the end of a bad pixel or column map
Use 2:: Remove all entries from a bad pixel or column map
Use 3:: Retrieve an entry from a bad pixel or column map
Use 4:: Retrieve the address and length of a map (for telemetry dump purposes)

32.3 Organization

Figure 140 illustrates the class relationships used by the Bad Pixel and Column Map classes. These classes include the **`BadPixelMap`** class, which is responsible for maintaining a list of bad pixels, and a **`BadColumnMap`** class, which is responsible for maintaining a list of bad column entries. In order to prevent stray pointers from corrupting the tables, their contents are maintained within Instruction Cache RAM (I-cache). The **`BadPixelMap`** and **`BadColumnMap`** classes both use the **`Mongoose`** class to read and write to this RAM.

FIGURE 140. Bad Pixel and Column Map Class Relationships



BadPixelMap- This class represents a list of bad pixels for all CCDs in the system. This class maintains the list in I-cache and provides functions to append new entries to the end of the list (`addPixel`), remove all entries from the list (`removeAll`), retrieve an entry from the list (`getPixel`) and get the total number of entries currently stored in the list (`getCount`). In order to support dumping the list to telemetry, the class also provides a function which returns the address of the start of the list, and the number of 32-bit words contained in the list (`getMapInfo`).

BadColumnMap- This class represents a list of bad columns for all CCDs in the system. This class maintains the list in I-cache and provides functions to append new entries to the end of the list (`addColumn`), remove all entries from the list (`removeAll`), retrieve an entry from the list (`getColumn`) and get the total number of entries currently stored in the list (`getCount`). In order to support dumping the list to telemetry, the class also provides a function which returns the address of the start of the list, and the number of 32-bit words contained in the list (`getMapInfo`).

Mongoose- This class is provided by the ***Devices*** class category, and is used by the **BadPixelMap** and **BadColumnMap** classes to write and read data to and from I-cache RAM (`icacheWrite`, `icacheRead`).

32.4 Memory Layouts

32.4.1 I-cache Memory Map

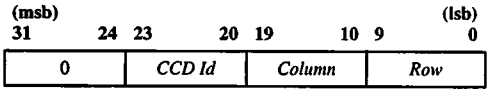
In order to reduce the opportunity that writes through a corrupted data pointer will corrupt the bad pixel and column maps, these maps are maintained within Instruction Cache RAM. Each map consists of a 32-bit entry count, followed by zero or more entries. For bad pixel maps, each entry is 32-bits wide. For bad column maps, each entry is 16-bits wide. Table 26 illustrates a proposed layout for the bad pixel and column maps.

TABLE 26. I-cache Bad Pixel and Column Map Layout (TBD)

Region		Address	Byte Size	Description
Patch Area		0x800fffff 0x800d7c00	0x30400	
Bad Pixel Map	Entries	0x800cdc04	0x9ffc	Array of bad pixel map entries (max. 10239)
	Count	0x800cdc00	0x4	Number of 32-bit bad pixel map entries in the table
Continuous Clocking Bad Column Map	Entries	0x800cc404	0x17fc	Array of bad column map entries (max. 3070 entries)
	Count	0x800cc400	0x4	Number of 16-bit bad column map entries in the table
Timed Exposure Bad Column Map	Entries	0x800cac04	0x17fc	Array of bad column map entries (max. 3070 entries)
	Count	0x800cac00	0x4	Number of 16 bit bad column map entries in the table
Compression Tables		0x800c2c00	0x8000	
Parameter Blocks		0x800c0400	0x2800	
System Configuration		0x800c0000	0x400	
Code		0x80080000	0x40000	

32.4.2 Bad Pixel Entry Format

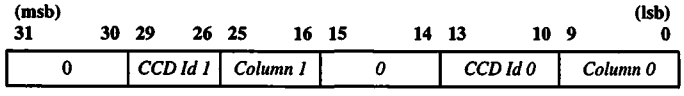
The bit layout for a bad pixel entry is as follows:



Where `row` and `column` specify the row and column position of the pixel within the CCD, and `CCD Id` specifies which CCD has the bad pixel. 0 indicates that the corresponding bits are set to 0 in a map entry.

32.4.3 Bad Column Entry Format

Bad column entries are bit-packed into each 32-bit entry location within I-cache, starting with the lower 16-bits. If there are an odd number of bad column entries in the table, the last 32-bit word in the table will contain 0 in its upper 16-bits. (NOTE: There is no ambiguity since he count at the beginning of the table reflects the number of column entries, not pairs of entries). The format of a 32-bit word containing two entries is as follows:



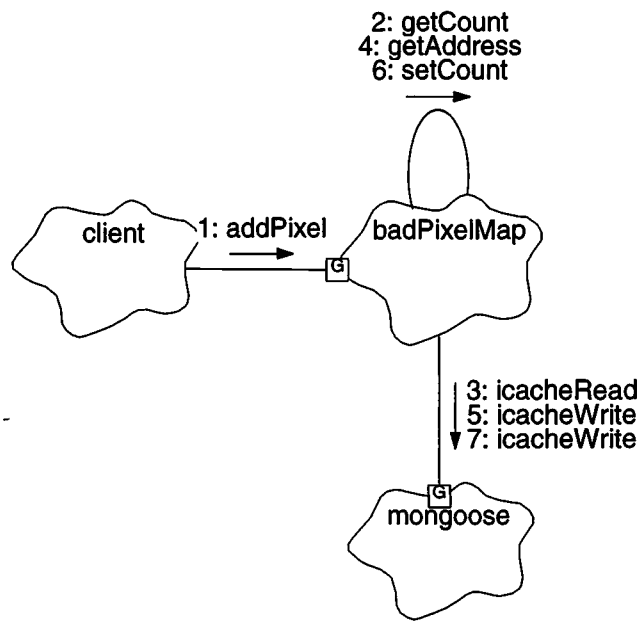
Where `Column` specifies the position of the column pixel within the CCD, and `CCD Id` specifies which CCD has the bad pixel. 0 indicates that the corresponding bits are set to 0 in a map entry.

32.5 Scenarios

32.5.1 Use 1: Append a bad pixel entry to the end of a map

Figure 141 illustrates the steps used by the client to append a bad pixel to the end of the bad pixel map. The steps to append a bad column are similar.

FIGURE 141. Append Bad Pixel to map

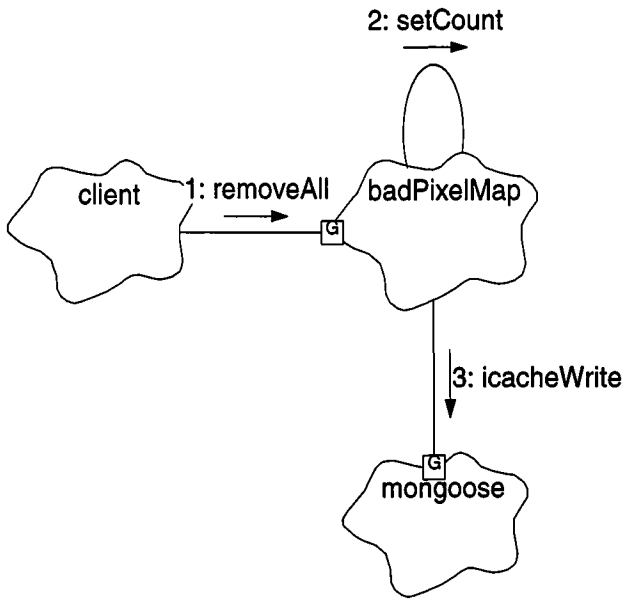


1. The *client* appends a pixel to the end of the bad pixel list by passing the pixel's CCD identifier, and row and column identifiers to *badPixelMap*.addPixel().
2. addPixel() gets the current end of the map using getCount().
3. getCount() uses *mongoose*.icacheRead() to read the entry count from within I-cache.
4. addPixel() checks to see if the table is full, and if so, returns the condition to the client. If the table is not full, addPixel() then retrieves the address of the entry slot just after the last entry using getAddress().
5. addPixel() then forms the entry value and write it to I-cache using *mongoose*.icacheWrite(). For the Bad Column map, if the entry slot index is odd, addColumn() (not illustrated) uses *mongoose*.icacheRead() to read the current 32-bit word from RAM, places the new entry value into the upper 16-bits of the word, and writes the value back out using *mongoose*.icacheWrite().
6. addPixel() then increments the current entry count and stores the value using setCount().
7. setCount() writes the new entry counter to I-cache using *mongoose*.icacheWrite().

32.5.2 Use 2: Remove all entries from a map

Figure 142 illustrates the steps used by the client to remove all entries from the bad pixel map. The steps to remove all entries from a bad column map are similar.

FIGURE 142. Delete contents of map

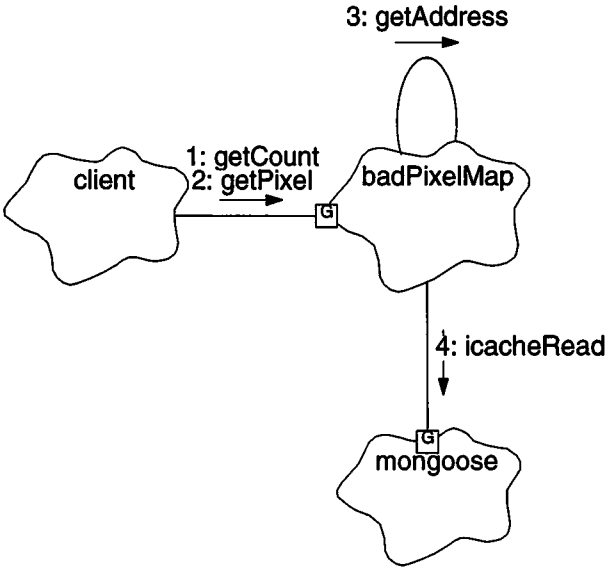


1. The *client* empties the bad pixel map by calling *badPixelMap*.removeAll().
2. removeAll() passes zero to setCount() to indicate the map is now empty.
3. setCount() stores the passed entry count into I-cache using *mongoose*.icacheWrite().

32.5.3 Use 3: Retrieve an entry from a map

Figure 143 illustrates the steps used by the client to retrieve an entry from the bad pixel map. The steps to obtain a bad column entry are similar.

FIGURE 143. Get Bad Pixel entry

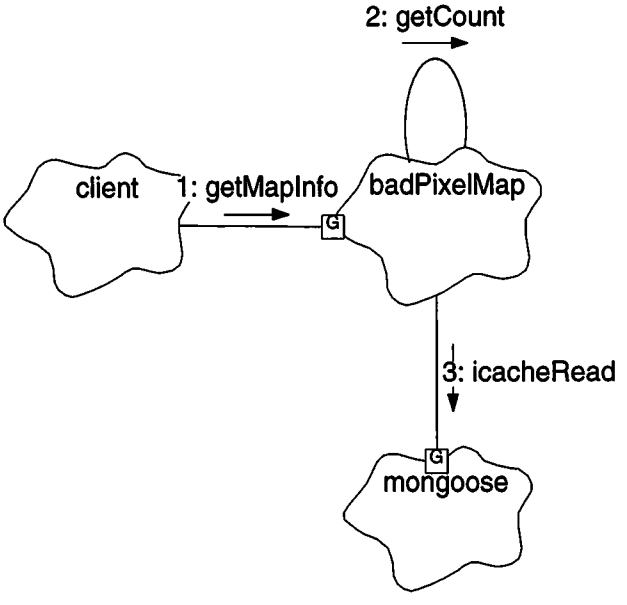


1. The client obtains the number of entries currently in the table using *badPixelMap*.`getCount()`.
2. The client then iteratively calls *badPixelMap*.`getPixel()` to obtain the contents of each bad pixel map entry.
3. `getPixel()` uses `getAddress()` to obtain the memory location of the desired map entry.
4. `getPixel()` then reads the entry using *mongoose*.`icacheRead()`, and returns the entry's CCD identifier, and row and column position to the caller. For the Bad Column Map, if the requested index is odd, the function shifts the upper 16-bits of the read word into the lower 16-bits, and then extracts the entry's fields from the lower 16-bits.

32.5.4 Use 4: Retrieve the address and length of a map

Figure 144 illustrates the steps used by a client to obtain the address and word length of the bad pixel map. The steps to obtain the corresponding address and length of a bad column map are similar.

FIGURE 144. Get Map Address and Length



1. The client gets the I-cache address of the map, and the number of words in the map using *badPixelMap*.`getMapInfo()`.
2. `getMapInfo()` retrieves the current number of entries using `getCount()`.
3. `getCount()` reads the entry count value from I-cache using *mongoose*.`icacheRead()`. `getMapInfo()` then returns the map's address and word count to the caller.

32.6 Class BadPixelMap

Documentation:

This class represents the Bad Pixel Map within the instrument. It is responsible for managing the list of bad CCD pixels within I-cache.

Export Control: Public

Cardinality: 1

Hierarchy:

Superclasses: **none**

Implementation Uses:

Mongoose

Public Interface:

Operations:
```
BadPixelMap()
addPixel()
getCount()
getMapInfo()
getPixel()
removeAll()
```

Protected Interface:

Operations:
```
getAddress()
setCount()
```

Private Interface:

Has-A Relationships:

unsigned* const *baseAddress*: This is the base address of the bad pixel map in i-cache.

const unsigned *maxCount*: This is the maximum number of entries in the bad pixel map table.

Concurrency: Guarded

Persistence: Persistent

32.6.1 BadPixelMap()

Public member of: **`BadPixelMap`**

Arguments:

`unsigned*` *`base`*
`unsigned` *`words`*

Documentation:

This is the constructor for the bad pixel map. *`base`* is the base address of the map in I-cache, and *`words`* is the maximum number of words that can be stored in the map. The initialization statements for the constructor set *`baseAddress`* to the passed *`base`*, and *`maxCount`* to the passed *`words`*.

Concurrency: Sequential

32.6.2 addPixel()

Public member of: **BadPixelMap**

Return Class: **Boolean**

Arguments:

CcdId *ccdid*
unsigned *row*
unsigned *column*

Documentation:

This function adds a pixel to the bad pixel map. *ccdid* is the CCD which contains the bad pixel. *row* and *column* identify the pixel's row and column location within the CCD. If successful, the function returns *BoolTrue*. If the map is full and the pixel cannot be stored, it returns *BoolFalse*.

Semantics:

This forms the bit-field entry using the passed arguments. It then uses `getCount()` to get the number of entries currently in the map. If the result is less than *maxCount*, the function uses `getAddress()` to get the virtual address of the last written entry, and uses *mongoose*.`icacheWrite()` to write the new entry onto the end of the map. It then uses `setCount()` to store the incremented entry count into I-cache.

Concurrency: Guarded

32.6.3 getAddress()

Protected member of: **BadPixelMap**

Return Class: **unsigned***

Arguments:

unsigned *index*

Documentation:

This function obtains the I-cache address for the table entry indicated by *index*. The function uses `getCount()` to obtain the number of entries in the table. If *index* is beyond the end of the table, this function returns 0, otherwise, it adds *index* + 1 to *baseAddress* (the plus 1 handles the count field at the beginning of the table) to form the address of the entry, and returns the computed table entry address.

Concurrency: Guarded

32.6.4 getCount()

Public member of: **BadPixelMap**

Return Class: **unsigned**

Documentation:

This function returns the number of bad pixel entries currently in the bad pixel map. This function uses *mongoose*.`icacheRead()` to read the count field from the map.

Concurrency: Guarded

32.6.5 getMapInfo()

Public member of: **BadPixelMap**

Return Class: **void**

Arguments:

const unsigned*& *addr*
unsigned& *wordcnt*

Documentation:

This function retrieves the base address of the bad pixel map, and the number of 32-bit words currently stored in the map. On return, *addr* contains the address of the map in I-cache (*baseAddress*), and *wordcnt* contains the number of words in the map (result of `getCount()` + 1).

Concurrency: Guarded

32.6.6 getPixel()

Public member of: **BadPixelMap**

Return Class: **Boolean**

Arguments:

unsigned *index*
CcdId& *ccdout*
unsigned& *rowout*
unsigned& *colout*

Documentation:

This function retrieves the bad pixel indexed by *index*. On return, *ccdout* contains the CCD which contains the bad pixel, and *rowout* and *colout* contain the row and column position within the CCD. If *index* is within range, the function returns *BoolTrue*. If *index* is beyond the last entry in the map, the function returns *BoolFalse*.

Semantics:

This starts by getting the address corresponding to the entry indicated by index using `getAddress()`. If the entry is valid (i.e. result is not zero), the function uses *mongoose*.`icacheRead()` to read the entry, and then extracts the bit-fields from the entry and stores the values into *ccdout*, *rowout* and *colout*.

Concurrency: Guarded

32.6.7 removeAll()

Public member of: **BadPixelMap**

Return Class: **void**

Documentation:

This function removes all pixels from the Bad Pixel Map by writing 0 to the entry count at the start of the map via `setCount()`.

Concurrency: Guarded

32.6.8 setCount()

Protected member of: **BadPixelMap**

Return Class: **void**

Arguments:

unsigned *count*

Documentation:

This function sets the number of pixels currently stored in the bad pixel map. This function calls *mongoose*.icacheWrite() to store count into the first location of the map.

Concurrency: Guarded

32.7 Class BadColumnMap

Documentation:

This class represents the Bad Column Map within the instrument. It is responsible for managing the list of bad CCD columns within I-cache.

Export Control: Public

Cardinality: 2

Hierarchy:

Superclasses: **none**

Implementation Uses:

Mongoose

Public Interface:

Operations:
```
BadColumnMap()
addColumn()
getColumn()
getCount()
getMapInfo()
removeAll()
```

Protected Interface:

Operations:
```
getAddress()
setCount()
```

Private Interface:

Has-A Relationships:

unsigned* const *baseAddress*: This is the base address of the bad column map in i-cache.

const unsigned *maxCount*: This is the maximum number of entries in the bad column map table.

Concurrency: Guarded

Persistence: Persistent

32.7.1 BadColumnMap()

Public member of: **BadColumnMap**

Arguments:

unsigned* *base*
unsigned *words*

Documentation:

This is the constructor for the bad column map. *base* is the base address of the map in I-cache, and *words* is the maximum number of words that can be stored in the map. The initialization statements for the constructor set *baseAddress* to the passed *base*, and *maxCount* to the passed *words*.

Concurrency: Sequential

32.7.2 addColumn()

Public member of: **BadColumnMap**

Return Class: **Boolean**

Arguments:

CcdId *ccdid*
unsigned *column*

Documentation:

This function adds a column to the bad column map. *ccdid* is the CCD which contains the bad column. *column* identifies the column's location within the CCD. If successful, the function returns *BoolTrue*. If the map is full and the column cannot be stored, it returns *BoolFalse*.

Semantics:

This forms a right-justified bit-field entry using the passed arguments. It then uses getCount() to get the number of entries currently in the map. If the result is less than *maxCount*, the function uses getAddress() to get the virtual address of the last written entry. If the result is odd, it then uses mongoose.icacheRead() to read the low order bits of the last entry, shifts the new entry into the upper 16-bits and combines the two. Otherwise, if the entry count is even, the function increments the address pointer to point to the next 32-bit word in the table. If then uses *mongoose*.icacheWrite() to write the new entry onto the end of the map. It then uses setCount() to store the incremented entry count into I-cache.

Concurrency: Guarded

32.7.3 getAddress()

Protected member of: **BadColumnMap**

Return Class: **unsigned***

Arguments:

unsigned *index*

Documentation:

This function obtains the I-cache address for the table entry indicated by *index*. The function uses `getCount()` to obtain the number of entries in the table. If *index* is beyond the end of the table, this function returns 0, otherwise, it adds (*index*/2) + 1 to *baseAddress* (NOTE: The divide by two is because there are two entries per 32-bit I-cache word, and the plus 1 handles the count field at the beginning of the table) to form the address of the entry, and returns the computed table entry address.

Concurrency: Guarded

32.7.4 getColumn()

Public member of: **BadColumnMap**

Return Class: **Boolean**

Arguments:

unsigned *index*
CcdId& *ccdout*
unsigned& *colout*

Documentation:

This function retrieves the bad column indexed by *index*. On return, *ccdout* contains the CCD which contains the bad column, and *colout* contains the column position within the CCD. If *index* is within range, the function returns *BoolTrue*. If *index* is beyond the last entry in the map, the function returns *BoolFalse*.

Semantics:

This starts by getting the address corresponding to the entry indicated by index using getAddress(). If the entry is valid (i.e. result is not zero), the function uses *mongoose*.icacheRead() to read word containing the entry. If *index* is odd, it then shifts the read word to put the desired entry into the lower 16-bits of the word. It then extracts the bit-fields from the entry in the lower 16-bits and stores the values into *ccdout*, *rowout* and *colout*.

Concurrency: Guarded

32.7.5 getCount()

Public member of: **BadColumnMap**

Return Class: **unsigned**

Documentation:

This function returns the number of bad column entries currently in the bad column map. This function uses *mongoose*.icacheRead() to read the count field from the map.

Concurrency: Guarded

32.7.6 getMapInfo()

Public member of: **BadColumnMap**

Return Class: **void**

Arguments:

const unsigned*& *addr*
unsigned& *wordcnt*

Documentation:

This function retrieves the base address of the bad column map, and the number of 32-bit words currently stored in the map. On return, *addr* contains the address of the map in I-cache (*baseAddress*), and *wordcnt* contains the number of words in the map. The *wordcnt* is computed by dividing the result of getCount() by two, rounding up to the nearest word, and adding 1 (for the count field at the start of the table).

Concurrency: Guarded

32.7.7 removeAll()

Public member of: **BadColumnMap**

Return Class: **void**

Documentation:

This function removes all columns from the Bad Column Map by writing 0 to the entry count at the start of the map via `setCount()`.

Concurrency: Guarded

32.7.8 setCount()

Protected member of: **BadColumnMap**

Return Class: **void**

Arguments:

unsigned *count*

Documentation:

This function sets the number of columns, *count*, currently stored in the bad column map using *mongoose*`.icacheWrite()`.

Concurrency: Guarded