Serial Interface Developers Guide for the ECS-320A Embeddable Camera Electronics System

(Document Number 700-00000040-R10)

10503 Timberwood Circle
Suite 120
Louisville, KY 40223
(502) 423-7225

Serial Interface Developers Guide
Document Number 700-00000040
April 2003

DISCLAIMER

NOTICE OF PROPRIETARY RIGHTS

The information in this document is subject to change without notice. Lumitron makes no representations or warranties with respect to the contents of this manual and specifically disclaims any implied warranties of merchantability or fitness for a particular purpose. Lumitron Inc. assumes no responsibility for errors or omissions in this document.

Lumitron
10503 Timberwood Circle
Suite 120
Louisville, KY 40223
(502) 423-7225
FAX (502) 423-7064

Table of Contents

1 Introduction

This guide has been written to help the developer become acquainted with and be able to develop around the Serial Interface Protocol requirements for the Embeddable Camera Electronics System hardware.

An overview of the system requirements and a detailed description of the protocol are provided. This guide also provides information on how the Lumitron Operational Manager Application software interacts with the ECS-320A hardware during system operation. This is done to provide examples on how the host application can communicate with the camera electronics.

The interface is based on a product developed by Motorola specifically for DSP integration. It consists of two components one that resides on the DSP (one of the camera electronics software drivers) and one that resides on a host (typically a PC). The protocol software that exists on the host can be custom developed, or the developer can integrate the Motorola provided software library. This document will deal only with the communications library provided by Motorola.

This document has been written with the assumption that the user is knowledgeable about Microsoft Windows OS based applications as well as how to create these applications.

This document is intended to encompass all camera application versions up to v11 b91 but is specific to that version. Earlier versions may not have all the features or commands listed in this document and there may be some differences in the data types/locations. Please contact Lumitron for specifics about previous camera software versions.

2 General Requirements

Software developed to interface with the camera electronics will need to incorporate the Motorola communications library version 1.2. This is the key component in development of an interface for a Windows based application. The files necessary to build an application can be obtained from Motorola. At the very least these files can be obtained by downloading or ordering (at no cost) the entire Motorola software development kit (MSW3SDK000AA: Embedded Software Development Kit for 56800/56800E), which will contain these files.

The files are as follows:

Mcbc12.dll	Dynamic Link Library
Mcbc12.lib	Library File
Mcbcom.chm	Compiled HTML help file
Mcbcom.h	Header File for library/dll
Mcberr.h	Header File with error codes

Along with the exposed interface provided by Motorola there are various defines, structures, and enumerations that are required to correctly transfer data to and from the camera electronics. This information can be obtained from Lumitron and portions of it may be listed in this document.

Host software requires an IBM PC or PC compatible with an available COM port. Lumitron's host application was designed for Windows 2000 and a minimum screen size setting of 1024 x 768.

3 Camera Boot Sequence

There are always two executables located on the camera electronics. The first is located in DSP boot flash and will be referred to as the "bootloader". This code provides a means for the camera to initially load or update the second executable (embedded camera application).

When a power on or reset occurs on the electronics: the DSP, via an interrupt, jumps to the boot flash and executes code located there. This code configures the DSP serial port peripheral registers to be

used for text/file transfer. The bootloader is JTAG loaded into the DSP boot flash, typically during the production of the camera electronics.

After the bootloader has configured the serial port it will output some text messages informing the host of its status and go into a wait state (approximately 5 seconds). During the wait state the host can initiate a file upload. Once a file upload is complete or the wait state times out, then execution is transferred to the application software loaded in DSP program flash memory.

3.1 Communication Configuration

The serial configuration of the camera while executing the bootloader is fixed:

- 115,200 Baud
- 8 Data Bits
- No Parity
- 1 Stop Bit

3.2 Boot Messages

During bootloader execution several text messages are output to the host. In a typical boot process where no file upload is attempted the following message is output.

Lumitron Bootloader v0003 for 49MHz Configuration.

© 2000-2001 Motorola Inc. S-Record loader. Version 1.3

Pause for transfer!

Application Started

As the host is monitoring the bootloader for text output it can key off of the 'Pause for Transfer!' message. At this point the host has a couple of seconds to begin the upload process. A typical boot process where a file is uploaded will have the following typical message.

Lumitron Bootloader v0003 for 49MHz Configuration.

© 2000-2001 Motorola Inc. S-Record loader. Version 1.3

Pause for transfer!

++

++++++++++++++++++++++++++++++++++++++

Loaded 0x8e1e Program and 0x0e70 Data words.

Application Started

The string of '+' characters in the message represent the file upload progress. Note that the message contains the Lumitron bootloader version, the controller board configuration oscillator rate, and the Motorola S-Record loader version.

3.3 Software Upload

The host application can be used to upgrade the camera electronics application software. To accomplish this task - the application needs to be 'ready' when the camera electronics receives a power on reset. This is the only way to initiate a file upload. The host application needs to be in the proper communications configuration (paragraph 3.1), have the file ready for upload (already stored in a buffer), and be prepared to trigger on the incoming message.

When the trigger occurs the host application can begin writing the file data to the COM port. The bootloader receives the incoming data stream, converts the text data, and writes the executable to the proper location in DSP program and data flash.

Once the file transfer is complete execution is transferred to the embedded application that was just loaded.

Lumitron v0003 Bootloader Version:

The latest bootloader version no longer uses a 'Xon/Xoff' protocol. This is due to the implementation of the RS-485 for specific camera configurations. Since, for this protocol we need to simulate half duplex communications, the host computer will send a single (complete) Motorola S-Record at a time. The data will be processed by the camera and then acknowledged with a '+' character in response. This tells the host that the camera is ready for another S-Record. Hosts that do not implement this 'send record : wait for response' will not function with this version of the bootloader.

Lumitron v0002/v0001 Bootloader Versions:

The upload of the embedded operational application is done using a standard "Xon/Xoff" protocol. Where the 'Xon' character is hexadecimal 0x11 and the 'Xoff' character is hexadecimal 0x13. The bootloader uploads files in the Motorola S-Record format.

4 Interface Protocol

The interface protocol is the set of simple binary structures and conventions, enabling data/code exchange between a host PC and a target camera electronics controller board. It uses raw 8-bits, no parity serial transfer at the controller board configured speed (115200 kbps by default).

The communication model is based on a master-slave basis. The PC computer sends a message with a command plus any arguments, and the target responds immediately (within specified time) with operation status code and return data. The target never initiates the communication. Its responses are exactly specified and always of fixed (known) length.

4.1 PC Master Information

PC Master is a protocol that was developed by Motorola for the real time test, debugging, and operation of DSP based hardware. It is made up of two parts: one that exists on the host and one that exists on the DSP controller.

The module that exists on the DSP controller is part of the software development kit (SDK) that was used in the design of the embedded camera application. Information specific to that SDK driver and its capabilities can be obtained from Motorola (*Embedded SDK: Targeting Motorola DSP56F80x Platform SDK126/D, Embedded SDK: PC Master User Manual SDK111/D*).

PC Master exists on the camera electronics as the only serial port driver for COM port 0. It operates in a polling mode and does not initiate a transmission but only responds to them. Using fixed memory mapped information about the DSP controller and the embedded application; it is possible to read/write DSP peripheral registers (paragraph 6.8), Xilinx FPGA registers (paragraph 0), and global variables (paragraph 6.2). Using Lumitron defined commands the capability expands to allow access to the real time clock, non-volatile RAM (paragraph TBD), serial data flash (paragraph 6.10), and coefficient data flash (paragraph TBD) parts via the DSP controller. This provides the host with a means to control/modify/check almost any configuration parameter that exists in the embedded software.

4.2 Communications Configuration

The configuration of the serial communications has two parts. The host will need to be able to configure which on-board port it will use and the baud rate of the port. It may also need to detect the configured baud rate of the camera electronics. This can easily be done by cycling through the

possible baud rates (9600, 38400, 115200) and verifying communication status. The remainder of the settings are automatically configured by the communications library.

5 Host Side Interface

5.1 Baud Rate

The baud rate of the host side can be fixed to the configuration of the camera electronics or it can be designed to detect the active baud rate. When opening the port on the host application side the function requires a baud rate parameter. See function definition below.

```
DWORD McbOpenCom(HMCBCOM* phCom, int port, int speed = 9600, LPWSTR remoteServer = NULL);
```

Using the open handle with the desired baud rate, the host application can call the function below.

```
DWORD McbDetect(HMCBCOM hCom, LPMCB_RESP_GETINFO pinfo);
```

By verifying the return of this function it can be determined if the baud rate setting is valid. Once it has been determined the connection is valid then data transfer can begin.

5.2 Function Subset

All of the functions contained in the Motorola communications library can be found in the ‘mcbcom.h’ file as well as the ‘Mcbcom.chm’ compiled HTML help file. Only those that are typical for interfacing with the camera electronics will be discussed here.

5.2.1 McbOpenCom

This function is used to open a PC Master Communications resource to begin data transfer. The pointer that is supplied is filled with a handle to the resource if successful. That handle is used in every subsequent call via the library to the camera electronics.

Function Definition: MCBCOM_API DWORD McbOpenCom(HMCBCOM* phCom, int port, int speed = 9600, LPWSTR remoteServer = NULL);

5.2.2 McbCloseCom

This function is used to close an existing PC Master communications resource. This function should be called when exiting the application, or when needing to change the resource configuration (baud rate).

Function Definition: MCBCOM_API void McbCloseCom(HMCBCOM hCom);

5.2.3 McbDetect

Call this function to detect if the communications link to the camera electronics is successful.

Function Definition: MCBCOM_API DWORD McbDetect(HMCBCOM hCom, LPMCB_RESP_GETINFO pinfo);

5.2.4 McbGetInfo

Call this function to obtain information about the protocol as it exists on the camera electronics. The supplied pointer is used to fill a data structure with the requested data.

Function Definition: MCBCOM MCBCOM_API DWORD McbGetInfo(HMCBCOM hCom, LPMCB_RESP_GETINFO pinfo);

5.2.5 McbReadDataMem

Call this function to read a block of memory from the DSP. At the lower level the call may be broken into several calls to the DSP controller to read the entire block. This function can also be used to read registers or a block of registers that are mapped into the DSP data memory space.

Function Definition: MCBCOM MCBCOM_API DWORD McbReadDataMem(HMCBCOM hCom, LPVOID dest, DWORD addr, WORD size);

Note that the function parameter for size is in bytes and the DSP base integer size is 16-bits, so to read a single memory location requires the parameter to be 2.

5.2.6 McbWriteDataMem

Call this function to write a block of memory to the DSP. At the lower level the call may be broken into several calls to the DSP controller to write the entire block. This function can also be used to write registers or a block of registers that are mapped into the DSP data memory space.

Function Definition: MCBCOM_API DWORD McbWriteDataMem(HMCBCOM hCom, LPCVOID src, DWORD addr, WORD size);

5.2.7 McbWriteDataMemMask

Call this function to write a block memory to the DSP with a mask parameter. At the lower level the call may be broken into several calls to the DSP controller to write the entire block. This function is useful when it is necessary to modify only a portion of a 16-bit register or location, thus providing a means to update a location without having to perform a read-modify-write at the higher level.

Function Definition: MCBCOM_API DWORD McbWriteDataMemMask(HMCBCOM hCom, LPCVOID src, LPCVOID mask, DWORD addr, WORD size);

5.2.8 McbSendAppCmd

Call this function to initiate a user defined command on the DSP controller. Typically commands are used when it is necessary to perform a sequence of events or to access resources that are outside of the DSP data memory space.

Function Definition: MCBCOM_API DWORD McbSendAppCmd(HMCBCOM hCom, BYTE code, DWORD argSize, LPCVOID argBuff);

An example of this function is shown below in paragraph 5.3.

5.2.9 McbGetAppCmdStatus

Call this function to retrieve the status of any outstanding commands. It is good practice to call this function to check the camera status before sending any new commands.

Function Definition: MCBCOM_API DWORD McbGetAppCmdStatus(HMCBCOM hCom, LPBYTE pCmdStatus);

5.3 Lumitron Defined Commands

Once the connection is verified a command can be sent to the camera electronics. A full list of available commands for the electronics is listed in. The paragraphs that follow will have a more detailed description of each command, the command response, and any arguments that are supplied with a specific command.

Below is an example of how to initiate a command to enable the focus motor in the ‘far’ direction. First check the status of the camera to ensure that a command is not already being executed. The ‘GetPCMStatus’ is a wrapper function that calls the McbGetAppCmdStatus function and includes a timeout.

```
…
// Wait until DSP is ready for next command
status = GetPCMStatus(m_hComPCM);

// Send Focus Motor Far Command
retVal = McbSendAppCmd(
      m_hComPCM,
      (BYTE)CMD_FOCUS_MOTOR_FAR,
      0,
      NULL
);

// If failed we need to break
if (FAILED(retVal))
{
      tStr.Format(IDS_PC_MSTR_ERROR, retVal);
      DoMessageBox(tStr, MB_OK, 0);
}
…
```

Notes:

- The third argument of the call ‘McbSendAppCmd’ is set to zero and the fourth argument is NULL. That is because this command requires no additional data for the command to perform its task.
- Several of the commands refer to the ‘scratch pad’ buffer. The scratch pad buffer is fixed at memory location 0x00C0 and has a length of 160 16-bit words (320 bytes). This gives the embedded application a place to temporarily store large amounts of data without having to break up serial flash, NUC flash, or utility memory reads/writes while trying to interface with the host.
- Some of the commands are part of a multi-command/function sequence to achieve the desired result. For example it takes two commands to read memory from the serial data flash. The first command takes the serial data flash address and the data transfer size as an argument, reads the appropriate flash page, and places the data read into a global ‘scratch pad’ buffer. The second command (function) calls the standard ‘McbReadDataMem’ to retrieve the data from the ‘scratch pad’ buffer into a local host buffer.
- The base data type for the PC side is 32-bit. The DSP on the other hand has a base of 16-bits. Bit fields are used as often as possible to provide efficient access to register values and configuration parameters. Also with the bit manipulation capability of the DSP it creates more efficient code for the embedded application. But the size difference between the base data types makes it difficult to use the same structures on both platforms. The structures that are listed in this document are pulled from the code on the PC side. This means that some data members are listed just as UWord16 instead of being cast to a 16-bit data structure. So bit fields within some data members will need a different method of extracting values (masking, cast after read, etc.).

5.3.1 CMD_COPY_SFLASH_PAGE

Description: Copy a page of serial flash memory to the DSP scratch pad buffer. Serial flash pages are 264 bytes long. The serial flash is used to store operational mode descriptor tables, NUC mode descriptor tables, vide palettes, overlay palettes, FPGA configuration files, and other additional items.

Command Code: Enumeration for CMD_COPY_SFLASH_PAGE

Argument Size: size of UWord16

Argument: Page number to be copied (0 - 4095)

Note: Use the McbReadDataMem function to read data from scratch pad.

5.3.2 CMD_PROG_SFLASH_FULL

Description: Program a full page to serial flash. The data to be programmed will need to be written to the scratch pad (DSP memory location 0x00C0) before initiating this command. The offset for the write into serial flash will be 0 regardless of the structure value.

Command Code: Enumeration for CMD_PROG_SFLASH_FULL

Argument Size: size of flash_XFER

Argument: See below.

```
struct _flash_XFER {
      WORD        eraseFlag;        // Status Flag to check if flash is erased
      WORD        page;             // Start Page in Serial flash (0 - 4095)
      WORD        offset;           // Offset for smaller than page size xfers ( < 264)
      WORD        size;             // Size in Bytes to program to flash (<= 264)
};
```

Note: Use the McbWriteDataMem function to move data into scratch pad.

5.3.3 CMD_PROG_SFLASH_PARTIAL

Description: Programs a partial page to serial flash. The data to be programmed will need to be written to the scratch pad (DSP memory location 0x00C0) before initiating this command.

Command Code: Enumeration for CMD_PROG_SFLASH_FULL

Argument Size: size of flash_XFER

Argument: See below.

```
struct _flash_XFER {
      WORD        eraseFlag;        // Status Flag to check if flash is erased
      WORD        page;             // Start Page in Serial flash (0 - 4095)
      WORD        offset;           // Offset for smaller than page size xfers ( < 264)
      WORD        size;             // Size in Bytes to program to flash (<= 264)
};
```

Note: Use the McbWriteDataMem function to move data into scratch pad.

5.3.4 CMD_PROG_PRODUCT_ID

Description: Program DSP Data flash with Product ID's. The product ID's will need to be written to the scratch pad (DSP memory location 0x00C0) before initiating this command. The product ID data is written to the DSP data flash starting at address 0x2000.

Command Code: Enumeration for CMD_PROG_PRODUCT_ID

Argument Size: 0

Argument: Null

Note: Use the McbWriteDataMem function to move data into scratch pad. Associated data structures listed below:

```
/* General Structure to hold component revision, type, and serial number */
struct _COMP_SER_NUM
{
      Word16            RevAndType;
      UWord32           SerNum;

};    /* Size = 3 Words */
typedef struct _COMP_SER_NUM COMP_SER_NUM, *PTR_COMP_SER_NUM;

/* Top level structure to hold info on each defined component */
struct _PRODUCT_ID
{
      COMP_SER_NUM      camera;
      COMP_SER_NUM      controllerBD;
      COMP_SER_NUM      camSupportBD;
```

```
      COMP_SER_NUM      fpaSupportBD;
      COMP_SER_NUM      calFlagAssy;
      COMP_SER_NUM      peripheral[4];
      COMP_SER_NUM      fpa;
      UWord16           ReserveBlk_A[2];

};    /* Size = 32 Words */
typedef struct _PRODUCT_ID PRODUCT_ID, *PTR_PRODUCT_ID;
```

5.3.5 CMD_READ_PRODUCT_ID

Description: Read Product ID from DSP Data flash and place into the scratch pad buffer.

Command Code: Enumeration for CMD_READ_PRODUCT_ID

Argument Size: 0

Argument: Null

Note: Use the McbReadDataMem function to read data from scratch pad. See data structures above.

5.3.6 CMD_PROG_STATIC_CFG

Description: Program DSP Data flash with the Static Configuration. The static configuration data will need to be written to the scratch pad (DSP memory location 0x00C0) before initiating this command. The static configuration data is written to the DSP data flash starting at address 0x2020.

Command Code: Enumeration for CMD_PROG_STATIC_CFG

Argument Size: 0

Argument: Null

Note: Use the McbWriteDataMem function to move data into scratch pad. Associated data structures listed below:

```
/* Cal Flag Servo Configuration Structure */
struct _CAL_FLAG_CFG
{
      UWord16           PwmPeriodFctr;
      UWord16           PwmCloseFctr;
      UWord16           PwmOpenFctr;

};
typedef struct _CAL_FLAG_CFG CAL_FLAG_CFG, *PTR_CAL_FLAG_CFG;

/* Lens Mode Configuration Structure */
struct _LENS_CFG
{
      /* 1st Word */
      unsigned          IDCode:8;         /* Lens ID Code (Bits 0 - 7) */
      unsigned          FocusMode:4;      /* Lens Focus Mode (Bits 8 - 11) */
      unsigned          ZoomMode:4;       /* Lens Zoom Mode (Bits 12 - 15) */

      /* 2nd Word */
      unsigned          BaseIndex:6;      /* NUC Base Index (Bits 0 - 5) */
      unsigned          LimitIndex:6;     /* NUC Limit Index (Bits 6 - 11) */
      unsigned          RsvdBits_A:4;     /* Reserved (Bits 12 - 15) */
};
typedef struct _LENS_CFG LENS_CFG, *PTR_LENS_CFG;

/* Static Configuration Structure */
struct _STATIC_CFG
{
      UWord16           RefClkRate;
      UWord16           comCfg;
      UWord16           dspVideo;
      UWord16           procVideo;
      CAL_FLAG_CFG      calFlag;
      UWord16           ReserveBlk_B[17];
```

```
	LENS_CFG		lens[4];

};
typedef struct _STATIC_CFG STATIC_CFG, *PTR_STATIC_CFG;
```

5.3.7 CMD_READ_STATIC_CFG

Description: Read Static Configuration from DSP Data flash and place into the scratch pad buffer.

Command Code: Enumeration for CMD_READ_STATIC_CFG

Argument Size: 0

Argument: Null

Note: Use the McbReadDataMem function to read data from scratch pad. See data structures above.

5.3.8 CMD_GET_CAMERA_TIME

Description: Read the Real Time Clock data and place into the scratch pad buffer. The time data is read from the serial peripheral Dallas real time clock chip.

Command Code: Enumeration for CMD_GET_CAMERA_TIME

Argument Size: 0

Argument: Null

Note: Use the McbReadDataMem function to read data from scratch pad. Associated data structures listed below:

```
/* Structure to hold time info from Real Time Clock */
struct _RTC_DATA
{
	UWord16		seconds;	/* 0 - 59 */
	UWord16		minutes;	/* 0 - 59 */
	UWord16		hours;		/* 0 - 23 */
	UWord16		day;		/* 1 - 7 */
	UWord16		date;		/* 1 - 31 */
	UWord16		month;		/* 1 - 12 */
	UWord16		year;		/* 0 - 99 (Add 2000 to get year) */

}; /* Size = 7 Words */
typedef struct _RTC_DATA RTC_DATA, *PTR_RTC_DATA;
```

5.3.9 CMD_SET_CAMERA_TIME

Description: Set the Real Time Clock data from values stored in the scratch pad buffer.

Command Code: Enumeration for CMD_SET_CAMERA_TIME

Argument Size: 0

Argument: Null

Note: Use the McbWriteDataMem function to move data into scratch pad. See data structures above.

5.3.10 CMD_GET_NVM_DATA

Description: Read a block of memory from the serial peripheral Dallas non-volatile memory (NVM) to the scratch pad buffer. The NVM block is used to store dynamic configuration data for the camera such as AGC/ALC mode, overlay mode, reticle position, active operation mode, active NUC table index, and many other settings. These settings are used during the boot process to restore the camera to a known state.

Command Code: Enumeration for CMD_GET_NVM_DATA

Argument Size: size of NVM_XFER

Argument: See below.

```
struct _NVM_XFER {
      UWord16     offset;     // Offset from byte 0
      UWord16     size;       // Size in bytes
};
typedef struct _NVM_XFER NVM_XFER, *PTR_NVM_XFER;
```

Note: Use the McbReadDataMem function to read data from scratch pad. The offset term in the data structure determines the offset address relative to the NVM part. This command allows the host to retrieve any portion or all of the current NVM contents.

Do not confuse access to this memory block with the portion of the global configuration data structure that has a structure member that contains the shadow version. On boot data from the NVM part is loaded into the global data structure for real time use by the software.

See Appendix D for information on how the memory is mapped on the serial NVM part.

5.3.11 CMD_SET_NVM_DATA

Description: Write a block of memory to the serial peripheral Dallas non-volatile memory (NVM) from the scratch pad buffer.

Command Code: Enumeration for CMD_SET_NVM_DATA

Argument Size: size of NVM_XFER

Argument: See paragraph 5.3.10

Note: Use the McbWriteDataMem function to move data into scratch pad. The offset term in the data structure determines the offset address relative to the NVM part.

Do not confuse access to this memory block with the portion of the global configuration data structure that has a structure member that contains the shadow version. On boot data from the NVM part is loaded into the global data structure for real time use by the software.

See Appendix D for information on how the memory is mapped on the serial NVM part.

5.3.12 CMD_FOCUS_MOTOR_FAR

Description: Calls the focus-out routine, which enables the focus motor in the forward direction for 200ms.

Command Code: Enumeration for CMD_FOCUS_MOTOR_FAR

Argument Size: 0

Argument: Null

Note: none.

5.3.13 CMD_FOCUS_MOTOR_NEAR

Description: Calls the focus-in routine, which enables the focus motor in the reverse direction for 200ms.

Command Code: Enumeration for CMD_FOCUS_MOTOR_NEAR

Argument Size: 0

Argument: Null

Note: none.

5.3.14 CMD_IMAGE_GRAB

Description: Perform an image grab into one of the utility memory buffers.

Command Code: Enumeration for CMD_IMAGE_GRAB

Argument Size: UWord16

Argument: Buffer to be selected (0 – data placed in image grab buffer A, and 1 - data placed in image grab buffer B).

Note: This command was included for debug and development and should not be used for normal operation since a write operation takes place. Buffers may be in use by real time camera operation and conflicts may occur.

5.3.15 CMD_READ_UTILITY_MEMORY

Description: Read a block of utility memory. The utility memory is external RAM that is interfaced through the Xilinx FPGA. It is used to store image grab data, histogram data, intensity transform tables, NUC refresh coefficients, and symbology overlay data.

Command Code: Enumeration for CMD_READ_UTILITY_MEMORY

Argument Size: size of UTIL_MEM_XFER

Argument: See below.

```
struct _UTIL_MEM_XFER{
      UWord32           addr;             // Address in utility memory to read/write
      UWord16           size;             // Size in WORDS (Limit to 160)
};
typedef struct _UTIL_MEM_XFER UTIL_MEM_XFER, *PTR_UTIL_MEM_XFER;
```

Base addresses of utility memory partitions are defined as follows:

```
/* Utility Memory MAR Base Addresses */
#define     MAR_ITT_LOW             0x00000000  /* Thru 0x00003FFF */
#define     MAR_ITT_HIGH            0x00004000  /* Thru 0x00007FFF */
#define     MAR_HISTO_GRAB          0x00008000  /* Thru 0x0000BFFF */
#define     MAR_IMAGE_GRAB_A        0x0000C000  /* Thru 0x0001FFFF */
#define     MAR_IMAGE_GRAB_B        0x00020000  /* Thru 0x00033FFF */
#define     MAR_SYMBOLOGY_OVLY      0x00034000  /* Thru 0x00047FFF */
#define     MAR_NUC_REFRESH         0x00048000  /* Thru 0x0005BFFF */
#define     MAR_EXT_DATA            0x0005C000  /* Thru 0x0007FFFF */
```

Note: Buffers may be in use by real time camera operation and conflicts may occur.

5.3.16 CMD_NUC_FLASH_RAMP

Description: Debug command to tell the camera to write ramp count to the parameter blocks (0 – 7) of the external NUC coefficient flash. The NUC coefficient flash is interfaced via the Xilinx FPGA.

Command Code: Enumeration for CMD_NUC_FLASH_RAMP

Argument Size: 0

Argument: Null

Note: Used to help with the development and testing of the flash memory interface to the FPGA and DSP.

5.3.17 CMD_NUC_FLASH_MEMORY

Description: Read a block of external NUC flash memory into the scratch pad buffer.

Command Code: Enumeration for CMD_NUC_FLASH_MEMORY

Argument Size: size of NUC_MEM_XFER

Argument: See below.

```
struct _NUC_MEM_XFER{
      UWord32     addr;       // Address in NUC flash memory to read
      UWord16     size;       // Size in WORDS (Limit to 160)
};
typedef struct _NUC_MEM_XFER NUC_MEM_XFER, *PTR_NUC_MEM_XFER;
```

Note: Use the McbReadDataMem function to read data from scratch pad. Multiple calls of this function would allow the host to read an entire set of NUC coefficients.

5.3.18 CMD_NUC_FLASH_TEST_PATTERN

Description: Debug command to tell the camera to write ramp count to the base blocks (10 – 14) of the external NUC coefficient flash. The NUC coefficient flash is interfaced via the Xilinx FPGA.

Command Code: Enumeration for CMD_NUC_FLASH_TEST_PATTERN

Argument Size: 0

Argument: Null

Note: Used to help with the development and testing of the flash memory interface to the FPGA and DSP.

5.3.19 CMD_TEC_DRV_ENABLE

Description: Enables or Disables the TEC Drive circuitry.

Command Code: Enumeration for CMD_TEC_DRV_ENABLE

Argument Size: UWord16

Argument: 0 – for disable, 1 – for enable

Note: Normally this operation would be controlled by settings in the NUC block descriptor table, but can be overridden for test purposes.

5.3.20 CMD_TEC_TEMP_SELECT

Description: Select TEC Temperature state to high or low.

Command Code: Enumeration for CMD_TEC_TEMP_SELECT

Argument Size: UWord16

Argument: 0 – for low temp state, 1 – for high temp state

Note: Normally this operation would be controlled by settings in the NUC block descriptor table, but can be overridden for test purposes.

5.3.21 CMD_CAL_FLAG_SERVO

Description: Calls the desired open servo, close servo, or set by factor routine. If the open or close mode is commanded then the values from data flash are used to set the servo position. If the factor mode is commanded then the factor supplied with the command packet will be used to set the position.

Command Code: Enumeration for CMD_CAL_FLAG_SERVO

Argument Size: Size of SERVO_MODE.

Argument: See below.

```
/* Commanded Servo Setting Structure */
```

```
struct _SERVO_MODE
{
      UWord16     mode;       /* Mode of Servo: Open, Closed, Factor */
      UWord16     factor;     /* PWM Factor for user specified setting */

};
typedef struct _SERVO_MODE SERVO_MODE, *PTR_SERVO_MODE;

/* Servo Mode Enumeration */
enum
{
      SERVO_OPEN = 0,
      SERVO_CLOSE,
      SERVO_FACTOR
};
```

Note: This command is to be used for test purposes. Once configured the servo will be controlled by the embedded application during normal operation

5.3.22 CMD_CAL_FLAG_REFERENCE

Description: Sets the calibration flag reference to the supplied state. If hot or cold reference is selected then the factors stored in the serial data flash (active NUC mode table) are used. If factor mode is selected then the value supplied with the command packet will be used to set the reference temperature.

Command Code: Enumeration for CMD_CAL_FLAG_REFERENCE

Argument Size: Size of CAL_FLAG_STATE.

Argument: See below.

```
/* Commanded Calibration Flag Setting Structure */
struct _CAL_FLAG_STATE
{
      UWord16     state;      /* State of Flag: Ambient, Cold, Hot, Factor */
      UWord16     factor;     /* Factor for user specified setting */

};
typedef struct _CAL_FLAG_STATE CAL_FLAG_STATE, *PTR_CAL_FLAG_STATE;

/* Calibration Reference Flag Enumeration */
enum
{
      CAL_FLAG_AMBIENT = 0,
      CAL_FLAG_HOT,
      CAL_FLAG_COLD,
      CAL_FLAG_FACTOR
};
```

Note: This command is to be used for test purposes. Once configured the calibration flag will be controlled by the embedded application during normal operation.

5.3.23 CMD_ONE_PT_REFRESH

Description: Initiates a 1-point refresh calibration.

Command Code: Enumeration for CMD_ONE_PT_REFRESH

Argument Size: UWord16

Argument: 0 – external reference, 1 – internal reference

Note: Since the 1 point refresh calibration using external flag requires placement of cold sources it is necessary to implement this command in ‘sub-protocol’ form. See paragraph 7.1 for details.

5.3.24 CMD_TWO_PT_NUC

Description: Initiates a 2-point calibration. The host is required to check/update the status of a global associated with the calibration process.

Command Code: Enumeration for CMD_TWO_PT_NUC

Argument Size: UWord16

Argument: 0 – external reference, 1 – internal reference

Note: Since the 2 point calibration requires placement of hot and cold sources it is necessary to implement this command in 'sub-protocol' form. See paragraph 7.5 for details.

5.3.25 CMD_LOAD_COLOR_PAL

Description: Load a video color palette from serial flash (make active).

Command Code: Enumeration for CMD_LOAD_COLOR_PAL

Argument Size: UWord16

Argument: index of desired palette (0 – 15)

Note: No check is performed to see if palette data in serial flash is valid.

5.3.26 CMD_LOAD_OVLY_PAL

Description: Load an overlay palette from serial flash (make active).

Command Code: Enumeration for CMD_LOAD_OVLY_PAL

Argument Size: UWord16

Argument: index of desired palette (0 – 7)

Note: No check is performed to see if palette data in serial flash is valid.

5.3.27 CMD_PIN_CHECK

Description: Supply a PIN for unlocking various memory locations in the camera for updating settings.

Command Code: Enumeration for CMD_PIN_CHECK

Argument Size: UWord16

Argument: PIN value (0 – 65535)

Note: This routine is supplied for advanced user operation. Portions of memory are locked to prevent inadvertent re-configuring of camera settings that could potentially put the camera in an unusable state.

5.3.28 CMD_FAN_SPEED_OPERATION

Description: Command to test fan operation.

Command Code: Enumeration for CMD_FAN_SPEED_OPERATION

Argument Size: UWord16

Argument: 0 – to disable fan, 1 – to enable fan

Note: Fan speed is normally controlled by software during operation, but can be overridden for test purposes.

5.3.29 CMD_GET_ADC_VALUES

Description: Command to retrieve the current value of all active ADC values. The data is placed into the scratch pad buffer.

Command Code: Enumeration for CMD_GET_ADC_VALUES

Argument Size: 0

Argument: Null

Note: Use the McbReadDataMem function to read data from scratch pad. This command was added for developmental purposes. The ADC channels read: ADC A Channels – 0 through 7, ADC B Channels 0 – 3, 7.

5.3.30 CMD_ENABLE_RETICLE

Description: Enables or disables the selected reticle on the overlay symbology.

Command Code: Enumeration for CMD_ENABLE_RETICLE

Argument Size: size of RETICLE_XFER

Argument: See below.

```
struct _RETICLE_XFER {
      unsigned    select:1;         // Select A (0) or B (1)
      unsigned    enable:1;         // Reticle Enable
      unsigned    horPos:9;         // Reticle Horizontal Position
      unsigned    size:4;           // Reticle Radius
      unsigned    empty:1;          // Open

      unsigned    emissivity:8;     // Reticle Emissivity
      unsigned    verPos:8;         // Reticle Vertical Position
};
typedef struct _RETICLE_XFER RETICLE_XFER, *PTR_RETICLE_XFER;
```

Note: The data contains all the data needed to enable the reticle on the desired screen location.

5.3.31 CMD_RETICLE_POSITION

Description: Move the selected reticle to desired location.

Command Code: Enumeration for CMD_RETICLE_POSITION

Argument Size: size of RETICLE_XFER

Argument: See paragraph 5.3.30.

Note: This command and CMD_ENABLE_RETICLE are functionally identical.

5.3.32 CMD_ONE_PT_UPDATE

Description: Perform a directed 1-point update calibration. Offset coefficients computed by the calibration are stored in NUC flash (become part of the permanent NUC coefficient set).

Command Code: Enumeration for CMD_ONE_PT_UPDATE

Argument Size: UWord16

Argument: 0 – do not use internal calibration flag, 1 – use internal calibration flag.

Note: Since the 1 point update calibration may require placement of reference source, it is necessary to implement this command in 'sub-protocol' form. See paragraph 7.3 for details.

5.3.33 CMD_WRITE_VID_ENC_REG

Description: Command to write value to video encoder register.

Command Code: Enumeration for CMD_WRITE_VID_ENC_REG

Argument Size: size of 2 * UWord16

Argument: 1st UWord16 – offset, 2nd UWord16 – data. The offset term is relative to the video encoder mode 0 register. In other words if a value of 0 is supplied as the offset then data will be written to the mode 0 register.

Note: This command is intended for development purposes and should not be used for normal operation.

5.3.34 CMD_PERFORM_TEST

Description: Perform test specified by argument in command packet.

Command Code: Enumeration for CMD_PERFORM_TEST

Argument Size: UWord16

Argument: Index from enumeration below.

```
enum
{
      TEST_REGISTERS = 0,
      TEST_OPERATIONAL,
      TEST_MEMORY,

      TEST_UNDEFINED
};
```

Note: After issuing a test command it is necessary to delay for several seconds until the camera has completed testing. Then the results can be obtained by reading the global configuration structure member 'CAMERA_ERRORS' (see Appendix A).

5.3.35 CMD_OVERLAY_REFRESH

Description: Command to perform a refresh of the symbology overlay.

Command Code: Enumeration for CMD_OVERLAY_REFRESH

Argument Size: 0

Argument: Null

Note: Erases contents of overlay memory and sets flag so that all objects are repainted.

5.3.36 CMD_FREEZE_IMAGE

Description: Command to (un)freeze display imagery.

Command Code: Enumeration for CMD_FREEZE_IMAGE

Argument Size: UWord16

Argument: 0 – normal imagery, 1 – freeze image

Note: none.

5.3.37 CMD_DETECT_BAD_PIXELS

Description: Command to initiate a defective pixel detection routine.

Command Code: Enumeration for CMD_DETECT_BAD_PIXELS

Argument Size: 0

Argument: Null

Note: Since the defective pixel requires placement of reference sources it is necessary to implement this command in 'sub-protocol' form. See paragraph 7.7 for details.

5.3.38 CMD_IRCON_LOAD_LUT

Description: Custom LUT table creation for radiometry. Contact Lumitron for specifics.

5.3.39 CMD_LOAD_RAD_PARAMS

Description: Custom load radiometric parameters for compile time software. Contact Lumitron for specifics.

5.3.40 CMD_RESET_PFV_COUNT

Description: Resets the processed FPA video frame count.

Command Code: Enumeration for CMD_RESET_PFV_COUNT

Argument Size: 0

Argument: Null

Note: none.

5.3.41 CMD_CLEAR_CONTINUE_FLAG

Description: Clears (resets) the idle while fault continue flag.

Command Code: Enumeration for CMD_CLEAR_CONTINUE_FLAG

Argument Size: 0

Argument: Null

Note: Errors that are trapped during the boot process or while operating under normal conditions cause the embedded software to enter an idle routine. While in this routine the host can check the error code and associated information. Once the fault has been acknowledged the continue flag can be cleared and the software will resume where it left off.

5.3.42 CMD_UNIFORMITY_TEST

Description: Command to initiate a uniformity test.

Command Code: Enumeration for CMD_UNIFORMITY_TEST

Argument Size: 0

Argument: Null

Note: This test is still under development.

5.3.43 CMD_WRITE_UTILITY_MEMORY

Description: Write data to the utility memory. The utility memory is external RAM that is interfaced through the Xilinx FPGA. The data to be programmed will need to be written to the scratch pad (DSP memory location 0x00C0) before initiating this command.

Command Code: Enumeration for CMD_WRITE_UTILITY_MEMORY

Argument Size: size of UTIL_MEM_XFER

Argument: See below.

```
struct _UTIL_MEM_XFER{
      UWord32     addr;       // Address in utility memory to read/write
      UWord16     size;       // Size in WORDS (Limit to 160)
```

```
};
typedef struct _UTIL_MEM_XFER UTIL_MEM_XFER, *PTR_UTIL_MEM_XFER;
```

Note: Use the McbWriteDataMem function to move data into scratch pad.

5.3.44 CMD_ADV_DETECT_BAD_PIXELS

Description: Command no longer used.

5.3.45 CMD_UPLOAD_NUC

Description: Command to upload a complete NUC table to the desired NUC index. The NUC table will be formatted for the embedded application and will contain pixel replace index values for defective pixels. Due to limited resources on the camera controller board, the command needs to be executed twice to upload both the gain and offset terms.

Command Code: Enumeration for CMD_UPLOAD_NUC

Argument Size: UWord16 parameter[2]

Argument:

parameter [0] – 0 for gain terms, 1 for offset terms.

parameter [1] – Base NUC address of the NUC table that is being uploaded. This value is the same value as that which would be written to the Xilinx NucTableBaseReg register and is computed as follows: Value = (NucTable * 5) + 3; where 0 >= NucTable <= 63. It translates to the MS bits of the NUC flash memory.

Note: See paragraph 7.9 for details on how to complete this process.

5.3.46 CMD_DOWNLOAD_NUC

Description: Command to download a complete NUC table to the desired NUC index. The NUC table will be formatted for the embedded application and will contain pixel replace index values for defective pixels. Due to limited resources on the camera controller board, the command needs to be executed twice to upload both the gain and offset terms.

Command Code: Enumeration for CMD_DOWNLOAD_NUC

Argument Size: UWord16 parameter[2]

Argument:

parameter [0] – 0 for gain terms, 1 for offset terms.

parameter [1] – Base NUC address of the NUC table that is being downloaded. This value is the same value as that which would be written to the Xilinx NucTableBaseReg register and is computed as follows: Value = (NucTable * 5) + 3; where 0 >= NucTable <= 63. It translates to the MS bits of the NUC flash memory.

Note: See paragraph 7.10 for details on how to complete this process.

5.3.47 CMD_COMPILE_DEFECT_LISTS

Description: Command to initiate a defective pixel list build for all valid NUC tables. The address range in NUC flash where each of the defect lists are stored will be erased prior to beginning the compile.

Command Code: Enumeration for CMD_COMPILE_DEFECT_LISTS

Argument Size: 0

Argument: None

Note: Once the command has been sent the embedded application will set the status code to BEGIN_PROCESS. The host can monitor this code until it is returned to HOST_READY which indicates the routine has completed compiling the defects.

5.3.48 CMD_RESTORE_FACTORY_DEFECTS

Description: Command to restore the active NUC table to the factory defect state. When complete, the coefficients of the NUC table will be erased, except for the locations where defects from the factory compiled list have been incorporated.

Command Code: Enumeration for CMD_RESTORE_FACTORY_DEFECTS

Argument Size: 0

Argument: None

Note: Once the command has been sent the embedded application will set the status code to BEGIN_PROCESS. The host can monitor this code until it is returned to DEFECT_COMPLETED or HOST_READY which indicates the routine has completed the restore process.

5.3.49 CMD_ENABLE_RANGE_RETICLE

Description: Enable or disable the range reticle.

Command Code: Enumeration for CMD_ENABLE_RANGE_RETICLE

Argument Size: UWord16

Argument: 0 – disable range reticle, 1 – enable range reticle

Note: The reticle will only enable if the lens type is set for 100mm.

5.3.50 CMD_INIT_NUC_TABLE

Description: Erases the active NUC table.

Command Code: Enumeration for CMD_INIT_NUC_TABLE

Argument Size: 0

Argument: None

Note: The Current NUC table information will be lost.

5.3.51 CMD_CAMERA_RECOVER

Description: Initiates a camera recover command (calls routine to reset the NUC mode and reload various FPGA registers).

Command Code: Enumeration for CMD_CAMERA_RECOVER

Argument Size: 0

Argument: None

Note: Intended to be used only if corruption of digital port data has been detected.

5.3.52 CMD_UPDATE_EXP_PORT

Description: Initiates a camera command that outputs current value of a shadow register to the write only expansion register. The shadow register is located in the global configuration data structure.

Command Code: Enumeration for CMD_UPDATE_EXP_PORT

Argument Size: 0

Argument: None

Note: Use of this command assumes that the host knows the current state of the shadow register in the global configuration structure

6 Camera Electronics Side Interface

The communications on the camera side is carried out by a PC Master Device driver that is part of the Motorola SDK targeted for the DSP 56F80X family. After transfer of execution from the bootloader, the software initializes a driver that in effect will 'take control' of the serial port. Once initialized the serial port will be in a listening mode, awaiting commands from the connected host. When an incoming command is detected, an interrupt occurs, and the command is parsed.

Standard commands (such as the reading and writing of DSP data memory) are executed and acknowledged immediately.

When a custom defined command (such as those listed in paragraph 5.3) is received, a command busy flag is set. This prevents the host from sending another command before the current one has been executed. The embedded software is responsible for polling the command status to establish if the host has requested some type of operation be performed. The command index is parsed, the task is performed, and then the command status flag is reset.

Initially the protocol baud rate is configured for 115200 bps. Shortly after the peripheral device drivers have been initialized, a check of the static configuration is done. If the camera electronics have been configured for a different baud rate then the serial configuration is modified appropriately.

6.1 DSP Data Memory

Using the PC Master Protocol direct access to the DSP data memory is permissible. There is however no direct access to the flash portion of the DSP data memory. This can be done indirectly to specific locations as described in paragraphs 5.3.4 - 5.3.7. The Motorola DSP56F807 has a data memory space as listed below.

Data Memory Map (Word Addresses):

0x0000 – 0x0FFF	*Data RAM*
0x1000 – 0x17FF	*DSP Peripherals*
0x1800 – 0x1FFF	*Reserved*
0x2000 – 0x3FFF	*Data Flash*
0x4000 – 0xFF7F	*External Memory*
0xFF80 – 0xFFFF	*Core Registers*

What Lumitron has done is to map various camera configuration and status data into pre-defined addresses so that the host can monitor/modify the camera electronics behavior. The paragraphs that follow will describe where the data is mapped and define the associated structure.

6.2 Global Configuration Structure (CAMERA_CONFIG)

The global configuration structure houses the current status of the camera electronics as well as information about the embedded application. This data structure has been mapped to address 0x0040 and has been allocated a length of 128 words. The structure is listed in Appendix A. As further updates to the embedded code are required the structure may be modified, but only by adding data members to the end of the structure. This way existing host applications can continue to access data members without updating code.

Any portion up to the entire structure can be read using the 'McbReadDataMem' routine. For example, if the host wanted to retrieve the current embedded application version, a read of the data member at address 0x0040 + offset of the 'swVersion', member could be performed.

The paragraphs below will provide information about each of the data members.

6.2.1 CameraConfig.nvmData

Type: NVM_GLOBAL_CFG

Size: 40 Words

Description: See paragraph TBD

6.2.2 CameraConfig.updateNVM

Type: bool

Size: 1 Word

Description: Set this value to non-zero when it is desired to update the non-volatile memory with the contents of the CameraConfig.nvmData. The setting is cleared by the embedded application.

6.2.3 CameraConfig.continueFlag

Type: bool

Size: 1 Word

Description: Under certain circumstances when the embedded application detects an error that it can recover from it goes into an idle state. In this idle routine the embedded application loops while checking for the state of this flag or a timeout to occur. Set the value of this flag to 1 (true or non-zero) for the code to return to normal operation.

6.2.4 CameraConfig.CmdsReceived

Type: UWord16

Size: 1 Word

Description: Debug value to keep track of user commands the camera has acknowledged since boot.

6.2.5 CameraConfig.camStats

Type: CAMERA_STATUS

```
/* Camera Status Code Structure */
struct _CAMERA_STATUS
{
      UWord16           ProcessCode;      /* Process Code Value */
      UWord16           ProgressCode;     /* Progress Code Value */
      UWord16           HostStatusCode;   /* Host Status Code */
      UWord16           HostData;         /* Status flag 2 */

};
typedef struct _CAMERA_STATUS CAMERA_STATUS, *PTR_CAMERA_STATUS;
```

Size: 4 Words

Description: Structure that contains four sub values for tracking boot progress, or interacting with the host during tests and calibrations. See paragraph 6.5 below for information on the progress code detection.

6.2.6 CameraConfig.camErrors

Type: CAMERA_ERRORS

```
/* Camera Error Code Structure */
struct _CAMERA_ERRORS
{
      UWord16           ErrorCode;        /* Error Code */
      UWord16           ErrorSubCode;     /* Error Code */
      UWord16           ErrorCount;       /* Additional errors after 1st */
      UWord16           ErrorData[5];     /* Error Data Array */

};
typedef struct _CAMERA_ERRORS CAMERA_ERRORS, *PTR_CAMERA_ERRORS;
```

Size: 8 Words

Description: Structure that contains various sub values for gathering information about errors that have occurred during testing and normal operation. See paragraph 6.6 for information on the error code detection.

6.2.7 CameraConfig.camTime

Type: RTC_DATA

```
/* Structure to hold time info from Real Time Clock */
struct _RTC_DATA
{
      UWord16           seconds;    /* 0 - 59 */
      UWord16           minutes;    /* 0 - 59 */
      UWord16           hours;      /* 0 - 23 */
      UWord16           day;        /* 1 - 7 */
      UWord16           date;       /* 1 - 31 */
      UWord16           month;      /* 1 - 12 */
      UWord16           year;       /* 0 - 99 (Add 2000 to get year) */

}; /* Size = 7 Words */
typedef struct _RTC_DATA RTC_DATA, *PTR_RTC_DATA;
```

Size: 7 Words

Description: Structure that contains the camera's current time and date information. Read only data member.

6.2.8 CameraConfig.expPort

Type: UWord16

Size: 1 Word

Description: Shadow value of the output expansion port used by the embedded application. Read only data member.

6.2.9 CameraConfig.swVersion

Type: UWord16

Size: 1 Word

Description: Software version ID of the loaded embedded application. Read only data member.

6.2.10 CameraConfig.swBuild

Type: UWord16

Size: 1 Word

Description: Software build ID of the loaded embedded application. Read only data member.

6.2.11 CameraConfig.fpgaVersion[2]

Type: UWord16

Size: 2 Words

Description: Xilinx FPGA version code of the stored configuration file. Read only data member.

6.2.12 CameraConfig.fpaSize

Type: IMAGE_SIZE

```
/* FPA Size Structure */
struct _IMAGE_SIZE
{
      UWord16           rows;       /* Rows (Vertical) */
      UWord16           cols;       /* Columns (Horizontal) */

};
typedef struct _IMAGE_SIZE    IMAGE_SIZE, *PTR_IMAGE_SIZE;
```

Size: 2 Words

Description: Structure that is filled at run time once it is determined the type of FPA that the embedded application will be configured to operate. Read only data member.

6.2.13 CameraConfig.agcLowIntensity

Type: UWord16

Size: 1 Word

Description: Value computed from any of the automatic gain and level control routines. Read only data member.

6.2.14 CameraConfig.agcHighIntensity

Type: UWord16

Size: 1 Word

Description: Value computed from any of the automatic gain and level control routines. Read only data member.

6.2.15 CameraConfig.actOpName[4]

Type: UWord16

Size: 4 Words (8 Bytes)

Description: Mnemonic for the active operational mode. Read only data member.

6.2.16 CameraConfig.actNucName[4]

Type: UWord16

Size: 4 Words (8 Bytes)

Description: Mnemonic for the active NUC mode. Read only data member.

6.2.17 CameraConfig.bitFieldIndex

Type: UWord16

Size: 1 Word

Description: The member is divided into the following bit fields.

Fan Speed Index: (Bits 0 - 2) Current internal fan speed index. Read only data member.

Base NUC table: (Bits 3 - 8) Current base NUC index. Read only data member.

Limit NUC table: (Bits 9 - 14) Current limit NUC index. Read only data member.

No Refresh: (Bit 15) Set to disable automatic NUC refreshes. Only use this bit during specific operations (defective pixel detection). Then clear when done.

6.2.18 CameraConfig.radSWInfo

Type: UWord16

Size: 1 Word

Description: OEM radiometric software version ID of the loaded embedded application. Read only data member.

6.2.19 CameraConfig.alarm

Type: UWord16

Size: 1 Word

Description: This member is divided into the following bit fields for alarm state monitoring and other miscellaneous settings. Read only data member(s).

Overtemp Alarm State: (Bit 0) Current overtemp alarm state.

Overtemp Acknowledge Alarm State: (Bit 1) Camera software acknowledge.

Battery Level State: (Bit 2) Low battery state value.

Battery Level Acknowledge State: (Bit 3) Camera software acknowledge.

FPA TEC Disable State: (Bit 4) TEC disable state (due to over temp condition).

Reserved Bits: (Bits 5 – 15).

6.2.20 CameraConfig.calFlagRefs

Type: UWord16

Size: 1 Word

Description: Calibration flag ADC reference settings as copied from the active NUC mode. These settings are used in subsequent calibrations that incorporate the internal calibration flag. The variable is divided as follows. Read only data member.

Cold Reference Setting: (Bits 0 – 7)

Hot Reference Setting: (Bits 8 – 15)

6.2.21 CameraConfig.fpaInfo

Type: UWord16

Size: 1 Word

Description: This value is broken into FPA type and sub-type as configured by the manufacturer/OEM. The value is read on boot from the DSP data flash and can be used by the host to configure host application settings. The variable is divided as follows. Read only data member.

FPA Type Identifier: (Bits 0 – 3)

FPA Sub-type Identifier: (Bits 4 – 7)

Range Reticle Enable: (Bit 8)

Reserved: (Bits 9 – 14)

Lock Memory Mirror Bit: (Bit 15)

The following enumerations are used for the type and subtype codes:

```
/* Fpa Type Enumerations */
enum
{
      FPA_NONE = 0,                 /* No FPA */
      FPA_DRS_U3000A,
      FPA_SOFRADIR,                 /* Ulis */
      FPA_RESERVED,
      FPA_INDIGO_9705,
      FPA_INDIGO_9809,
      FPA_INDIGO_0202,

      FPA_UNDEFINED                 /* x to 15 Reserved */
};

/* Fpa Sub-Type Enumerations */
enum
{
      FPA_9809_INSB = 0,
      FPA_9809_INGAAS,

      FPA_9809_UNDEFINED
};
```

6.2.22 CameraConfig.adcAFiltered[4]

Type: UWord16

Size: 4 Words

Description: Each word contains a filtered result of a corresponding ADC channel on the DSP. Read only data member. These filtered values have been shifted up one bit location for precision/rounding purposes. It is necessary to divide the value by 65535 to get a properly scaled value to use in voltage/temperature equations.

Index [0]: DSP Channel A0

Index [1]: DSP Channel A1

Index [2]: DSP Channel A2

Index [3]: DSP Channel A3

6.2.23 CameraConfig.adcBFiltered[4]

Type: UWord16

Size: 4 Words

Description: Each word contains a filtered result of a corresponding ADC channel on the DSP. Read only data member. These filtered values have been shifted up one bit location for precision/rounding purposes. It is necessary to divide the value by 65535 to get a properly scaled value to use in voltage/temperature equations.

Index [0]: DSP Channel B0

Index [1]: DSP Channel B1

Index [2]: DSP Channel B2

Index [3]: DSP Channel B3

6.2.24 CameraConfig.btnPanel

Type: BTN_PANEL

```
/* Button Panel State Structure */
struct _BTN_PANEL
{
      unsigned    menuUpdate:1;     /* Set if button was pressed and menu needs update.
      Should only be set by Host and only cleared by camera. */

      unsigned    actMenuID:7;      /* Active Menu or message ID from enumeration */

      unsigned    actCursor:4;      /* Current or active cursor location */
      unsigned    menuRows:4;       /* Number of rows in menu */

      /***** Camera Only Access *****/
      unsigned    reservedA:1;
      unsigned    prevMenuID:7;     /* Previous Menu or message ID from enumeration */

      unsigned    prevCursor:4;     /* Previous cursor location */
      unsigned    execState:2;      /* Execution state for main loop processing */
      unsigned    reservedB:2;

};    /* 2 Words */
typedef struct _BTN_PANEL     BTN_PANEL, *PTR_BTN_PANEL;
```

Size: 2 Words

Description: Structure that is modified by an attached custom button panel for menu control of camera. Do not modify this data.

6.3 Dynamic Configuration Structure (NVM_GLOBAL_CFG)

The dynamic configuration structure is the first structure member included in the global configuration. It contains settings that control the camera's 'look and feel'. Typical settings include selection of video and overlay palettes, AGC mode, reticle position, and additional features. The type definition of the structure is located in Appendix A.

On boot - the structure is filled with data that is read back from a subset of the nonvolatile RAM on the real time clock chip. This is the reason for the 'NVM' being a part of the member name. A complete description of the contents of the nonvolatile RAM is located in Appendix E.

During program flow the software will use the settings located in the NVM_GLOBAL_CFG structure to control behavior. The host can read these settings to determine the existing state and then use the 'McbWriteDataMem' routine to change a setting.

The paragraphs below will provide information about each of the data members (excluding reserved areas) in this structure.

6.3.1 nvmData.CamMode

Type: UWord16

Size: 1 Word

Description: See Appendix E for specifics on this value.

6.3.2 nvmData.FpaMode

Type: UWord16

Size: 1 Word

Description: See Appendix E for specifics on this value.

6.3.3 nvmData.ActMode

Type: UWord16

Size: 1 Word

Description: See Appendix E for specifics on this value.

6.3.4 nvmData.AutoNucData

Type: NVM_AUTO_NUC_MODE

```
/* Auto NUC Switch Parameters */
struct _NVM_AUTO_NUC_MODE
{
      unsigned    LowSatInt:14;     /* Auto NUC switch low saturation intensity */
      unsigned    ReservedA:2;      /* Reserved */
      unsigned    LowSatCount:16;   /* Auto NUC switch low saturation count */

      unsigned    HighSatInt:14;    /* Auto NUC switch high saturation intensity */
      unsigned    ReservedB:2;      /* Reserved */
      unsigned    HighSatCount:16;  /* Auto NUC switch high saturation count */

};
typedef struct _NVM_AUTO_NUC_MODE   NVM_AUTO_NUC_MODE, *PTR_NVM_AUTO_NUC_MODE;
```

Size: 4 Words

Description: See Appendix E for specifics on this value.

6.3.5 nvmData.AutoRfshTime

Type: UWord16

Size: 1 Word

Description: If the auto refresh time flag is enabled; then this value determines the time interval between automatic 1-point NUC refresh calibrations. Counter is reset upon any 1-point NUC refresh calibrations.

6.3.6 nvmData.AutoRfshTemp

Type: UWord16

Size: 1 Word

Description: If the auto refresh on temperature flag is enabled; then this value determines the temperature delta required to perform an automatic 1-point NUC refresh calibrations. This value is stored as an ADC count value. ADC count is saved upon any 1-point NUC refresh calibrations.

6.3.7 nvmData.ActPal

Type: UWord16

Size: 1 Word

Description: See Appendix E for specifics on this value.

6.3.8 nvmData.OvlMode

Type: UWord16

Size: 1 Word

Description: See Appendix E for specifics on this value.

6.3.9 nvmData.RtclXPos

Type: NVM_RETICLE_POS

```
/* Reticle Position & Emissivity */
struct _NVM_RETICLE_POS
{
      unsigned    RtclHorPos:9;     /* Reticle Horizontal Position */
      unsigned    Reserved:7;       /* Reserved */
      unsigned    RtclVerPos:8;     /* Reticle Vertical Position */
      unsigned    RtclEmiss:8;      /* Reticle Emissivity */

};
typedef struct _NVM_RETICLE_POS NVM_RETICLE_POS, *PTR_NVM_RETICLE_POS;
```

Size: 2 Words

Description: Same for both A and B reticles. See Appendix E for specifics on this value.

6.3.10 nvmData.RadMode

Type: UWord16

Size: 1 Word

Description: Bit 0 contains radiometric units (0 – Celsius, 1 – Fahrenheit).

6.3.11 nvmData.AgcMode

Type: UWord16

Size: 1 Word

Description: See Appendix E for specifics on this value.

6.3.12 nvmData.ManualITT

Type: NVM_MANUAL_ITT

```
/* Display Video Brightness & Contrast Data */
struct _NVM_MANUAL_ITT
{
      unsigned    LowInt:14;        /* Manual Low Intensity */
      unsigned    ReservedA:2;      /* Reserved */

      unsigned    HighInt:14;       /* Manual High Intensity */
      unsigned    ReservedB:2;      /* Reserved */

};
typedef struct _NVM_MANUAL_ITT NVM_MANUAL_ITT, *PTR_NVM_MANUAL_ITT;
```

Size: 2 Words

Description: See Appendix E for specifics on this value.

6.3.13 nvmData.AgcLimits

Type: NVM_AGC_LIMITS

```
/* AGC Intensity Limits */
struct _NVM_AGC_LIMITS
{
      unsigned    AgcLowLimit:14;   /* AGC Low Limit Intensity */
      unsigned    ReservedA:2;      /* Reserved */

      unsigned    AgcHighLimit:14;  /* AGC High Limit Intensity */
      unsigned    ReservedB:2;      /* Reserved */

};
```

```
typedef struct _NVM_AGC_LIMITS NVM_AGC_LIMITS, *PTR_NVM_AGC_LIMITS;
```

Size: 2 Words

Description: See Appendix E for specifics on this value.

6.3.14 nvmData.LinearMap

Type: UWord16

Size: 1 Word

Description: See Appendix E for specifics on this value.

6.3.15 nvmData.AgcBinLimit

Type: UWord16

Size: 1 Word

Description: See Appendix E for specifics on this value.

6.3.16 nvmData.ActZoneStat

Type: UWord16

Size: 1 Word

Description: Currently not used.

6.3.17 nvmData.VidScaleTemps

Type: NVM_VID_SCALE_TEMPS

```
/* Display Video Zero & Full Scale Temperatures (Addr: 100) */
struct _NVM_VID_SCALE_TEMPS
{
      UWord16     ZeroScaleTemp;    /* Display Video Zero Scale Temp */
      UWord16     FullScaleTemp;    /* Display Video Full Scale Temp */

};
typedef struct _NVM_VID_SCALE_TEMPS NVM_VID_SCALE_TEMPS, *PTR_NVM_VID_SCALE_TEMPS;
```

Size: 2 Words

Description: See Appendix E for specifics on this value.

6.3.18 nvmData.ImageParams

Type: UWord16

Size: 1 Word

Description: See Appendix E for specifics on this value.

6.3.19 nvmData.LensID

Type: UWord16

Size: 1 Word

Description: See Appendix E for specifics on this value.

Note: a change to the NVM_GLOBAL_CFG structure is only temporary. If the host wants that setting to be permanent (occur after next power on/reset), it is also necessary to set the ‘updateNVM’ flag. This tells the software to save the NVM_GLOBAL_CFG data to nonvolatile RAM at the next available

opportunity. In other words if the host intends on changing the current AGC mode and have that mode restored on subsequent boots – then two writes are required. The first to change the global data structure and then a second to force an update of the nonvolatile RAM.

6.4 Process Code Detection (CAMERA_STATUS)

During various routines, during normal operation, the embedded software will set a process codes. This code can be read by the host application to indicate why a camera may not be responding as desired. For example when the host requests a change in operational modes that triggers a change in the TEC setting, the camera may delay (freeze) for a minute or two while the TEC is allowed to stabilize. The code is located in the 'ProcessCode' member of the CAMERA_STATUS data structure.

The current process codes are listed below.

```
/* Process Code Enumerations */
enum
{
      PRC_UNDETERMINED = 0,
      PRC_FPGA_TESTS,
      PRC_OP_TESTS,
      PRC_MEM_TESTS,
      PRC_VIDENC_TESTS,
      PRC_TEC_STABILIZING,

      PRC_CAM_READY
};
```

6.5 Progress Code Detection (CAMERA_STATUS)

As the embedded software initializes hardware, performs built-in tests, and then enters the main operational loop, it sets progress codes. Reading the codes allows the host to detect the software's progress towards start of normal operation. The code is located in the 'ProgressCode' member of the CAMERA_STATUS data structure.

The current progress codes are listed below.

```
/* Progress Code Enumerations */
enum
{
      UNDETERMINED = 0,
      UNCONFIGURED_CONTROLLER,
      FAULT_DETECTED,
      MAIN_START,
      DRIVERS_INITIALIZED,
      GLOBALS_INITIALIZED,
      SYSTEM_RESET,
      BOARD_ID_CONFIRM,
      FPGA_PROGRAMMED,
      FPGA_REGISTER_TEST,
      CAMERA_OP_TEST_SETUP,
      INIT_FPA_SUPPORT_PWR,
      TEC_DRIVE_INITIALIZED,
      SYSTEM_TESTS_COMPLETE,
      CAL_FLAG_INITIALIZED,
      ENTERING_MAIN_LOOP,
      PALETTES_LOADED,
      OP_MODE_LOADED,
      OVERLAY_INITIALIZED,
      TEC_STABILIZED,

      CAMERA_READY
};
```

It is not required to take any action when reading these values but the host should not begin intensive communications with the camera until it has detected the CAMERA_READY progress code. This code is set just before entering the main processing loop (normal operation).

It is also recommended that the host check regularly (including the boot sequence) for the UNCONFIGURED_CONTROLLER and the FAULT_DETECTED codes.

The FAULT_DETECTED code is set by the embedded application when a built-in test fails or when a mismatch between the configured hardware and detected hardware occurs. The embedded software will jump to an idle loop to give the host a chance to read error data (see paragraph 6.6) and then set the continue flag (structure member 'continueFlag', paragraph 6.2.3). This idle routine has a timeout and will automatically return after time expires.

The UNCONFIGURED_CONTROLLER code is set by the embedded application when it detects erased product ID's, missing Xilinx FPGA configuration files, or an error programming the FPGA occurs. The embedded software will jump to an idle loop to give the host an opportunity to configure the controller (load product ID's, upload a Xilinx FPGA configuration file). This routine can not be exited, a power on/reset is required to return to normal operation.

6.6 Error Code Detection (CAMERA_ERRORS)

During the boot process several checks are made of the hardware. If a fault is detected then an error code is set along with any additional error data using the CAMERA_ERRORS data structure (see paragraph 6.2.5). Once the error data has been logged the code jumps to an idle state routine. Inside this routine the global configuration member progress code (paragraph 6.2.5) is set to FAULT_DETECTED and a timer is started.

While in this idle state the host has the opportunity to determine that a fault has occurred, read the error information and then set the continue flag (see paragraph 6.2.3). If the host does not acknowledge the error and the timer expires the code will continue operation. It should be noted that it may take a significantly longer time to boot if errors are present and the host does not acknowledge them appropriately.

The following is a list of possible error codes.

```
/* Camera Error Codes */
#define ERR_NONE                    0x0000
#define ERR_PC_MSTR                 0x8001
#define ERR_FPGA_LOAD               0x8002
#define ERR_FPGA_TEST               0x8003
#define ERR_MEM_TEST                0x8004
#define ERR_VID_ENCODER             0x8005
#define ERR_FORCE_ZERO              0x8006
#define ERR_FORCE_ONE               0x8007
#define ERR_FORCE_COUNT             0x8008
#define ERR_HISTO_GRAB              0x8009
#define ERR_GAIN_OFFSET             0x800A
#define ERR_FORCE_COUNT_COADD       0x800B
#define ERR_CONFIG_MISMATCH         0x800C
#define ERR_NUC_FLASH_PARAM         0x800D
#define ERR_TEST_SKIPPED            0x8FFF
```

6.6.1 Configuration ID Error

This check is of the configuration ID's stored in flash versus the ID's read back via the ADC.

Data from CameraConfig.camErrors:

ErrorCode: ERR_CONFIG_MISMATCH define.

ErrorSubCode: a more specific description of the error from the following enumeration.

```
/* Cfg Board ID Error SubCodes (#define ERR_CONFIG_MISMATCH 0x800C) */
enum
{
      ERR_LENS_MISMATCH = 1,
      ERR_FPA_SPRT_BD_MISMATCH,
      ERR_FPA_MISMATCH,
      ERR_CAM_CTRL_BD_MISMATCH,
```

```
        ERR_CAM_SPRT_BD_MISMATCH,

        ERR_SUBCODE_UNDEFINED
};
```

ErrorCount: a count of the total number of errors since boot.

ErrorData [0] Value stored in configuration flash.

ErrorData [1] Value read from the ADC.

ErrorData [2] – [4] Not used.

6.6.2 FPGA Load

This error is set during the configuration of the Xilinx FPGA.

Data from CameraConfig.camErrors:

ErrorCode: ERR_FPGA_LOAD define.

ErrorSubCode: a more specific description of the error from the following enumeration.

```
/* FPGA Load Error SubCodes (#define ERR_FPGA_LOAD 0x8002) */
enum
{
        ERR_FPGA_LOAD_UNDEFINED = 0,
        ERR_FPGA_NOT_DONE
};
```

ErrorCount: a count of the total number of errors since boot.

ErrorData [0] – [4] Not used.

6.6.3 FPGA Test

This error is set during the testing of the Xilinx FPGA registers.

Data from CameraConfig.camErrors:

ErrorCode: ERR_FPGA_TEST define.

ErrorSubCode: a more specific description of the error from the following enumeration.

```
/* FPGA Test Error SubCodes (#define ERR_FPGA_TEST 0x8003) */
enum
{
        ERR_FPGA_TEST_UNDEFINED = 0,
        ERR_WALKING_0_1,
        ERR_CROSSTALK,
        ERR_FPGA_MEMORY
};
```

ErrorCount: a count of the total number of errors since boot.

For SubCode ERR_WALKING_0_1:

ErrorData [0] Output pattern.

ErrorData [1] Register value read back.

ErrorData [2] Register test mask.

ErrorData [3] Address of register under test.

ErrorData [4] Not used.

For SubCode ERR_CROSSTALK:

ErrorData [0] Address of register being written to.

ErrorData [1] Address of register being tested for crosstalk.

ErrorData [2] Crosstalk register value.

ErrorData [3] Cross talk register test mask.

ErrorData [4] Not used.

For SubCode ERR_FPGA_MEMORY:

ErrorData [0] Value expected.

ErrorData [1] Value read from memory.

ErrorData [2] Test mask.

ErrorData [3] Memory address.

ErrorData [4] Not used.

6.6.4 Memory Test

This error is set during the testing of the controller utility memory.

Data from CameraConfig.camErrors:

ErrorCode: ERR_MEM_TEST define.

ErrorSubCode: a more specific description of the error from the following enumeration.

```
/* FPGA Test Error SubCodes (#define ERR_MEM_TEST 0x8004) */
enum
{
      ERR_MEM_TEST_UNDEFINED = 0,
      ERR_UTIL_RAMP_MAR_A,
      ERR_UTIL_ZERO_MAR_A,
      ERR_UTIL_RAMP_MAR_B,
      ERR_UTIL_ZERO_MAR_B
};
```

ErrorCount: a count of the total number of errors since boot.

ErrorData [0] Value expected.

ErrorData [1] Value read back.

ErrorData [2] Test mask.

ErrorData [3] Lower 16 bits of memory address.

ErrorData [4] Upper 16 bits of memory address.

6.6.5 Force ‘0’ Test

This error is set during operational tests.

Data from CameraConfig.camErrors:

ErrorCode: ERR_FORCE_ZERO define.

ErrorSubCode: No subcode.

ErrorCount: a count of the total number of errors since boot.

ErrorData [0] Value expected (0x0000).

ErrorData [1] Value read back.

ErrorData [2] Test mask.

ErrorData [3] Lower 16 bits of memory address.

ErrorData [4] Upper 16 bits of memory address.

6.6.6 Force ‘1’ Test

This error is set during operational tests.

Data from CameraConfig.camErrors:

ErrorCode: ERR_FORCE_ONE define.

ErrorSubCode: No subcode.

ErrorCount: a count of the total number of errors since boot.

ErrorData [0] Value expected (0x2000).

ErrorData [1] Value read back.

ErrorData [2] Test mask.

ErrorData [3] Lower 16 bits of memory address.

ErrorData [4] Upper 16 bits of memory address.

6.6.7 Force Count Test

This error is set during operational tests.

Data from CameraConfig.camErrors:

ErrorCode: ERR_FORCE_COUNT define.

ErrorSubCode: No subcode.

ErrorCount: a count of the total number of errors since boot.

ErrorData [0] Value expected.

ErrorData [1] Value read back.

ErrorData [2] Test mask.

ErrorData [3] Lower 16 bits of memory address.

ErrorData [4] Upper 16 bits of memory address.

6.6.8 Force Count Coadd Test

This error is set during operational tests.

Data from CameraConfig.camErrors:

ErrorCode: ERR_FORCE_COADD define.

ErrorSubCode: No subcode.

ErrorCount: a count of the total number of errors since boot.

ErrorData [0] Value expected.

ErrorData [1] Value read back.

ErrorData [2] Test mask.

ErrorData [3] Lower 16 bits of memory address.

ErrorData [4] Upper 16 bits of memory address.

6.6.9 Histogram Data Grab Test

This error is set during operational tests.

Data from CameraConfig.camErrors:

ErrorCode: ERR_HISTO_GRAB define.

ErrorSubCode: No subcode.

ErrorCount: a count of the total number of errors since boot.

ErrorData [0] Value expected.

ErrorData [1] Value read back.

ErrorData [2] Test mask.

ErrorData [3] Lower 16 bits of memory address.

ErrorData [4] Upper 16 bits of memory address.

6.6.10 NUC Gain and Offset Test

This error is set during operational tests.

Data from CameraConfig.camErrors:

ErrorCode: ERR_GAIN_OFFSET define.

ErrorSubCode: No subcode.

ErrorCount: a count of the total number of errors since boot.

ErrorData [0] Value expected.

ErrorData [1] Value read back.

ErrorData [2] Test mask.

ErrorData [3] Lower 16 bits of memory address.

ErrorData [4] Upper 16 bits of memory address.

6.6.11 Video Encoder Test

This error is set during operational tests.

Data from CameraConfig.camErrors:

ErrorCode: ERR_VID_ENCODER define.

ErrorSubCode: No subcode.

ErrorCount: a count of the total number of errors since boot.

ErrorData [0] Value output to encoder.

ErrorData [1] Value read back from encoder.

ErrorData [2] Test mask.

ErrorData [3] – [4] Not used.

6.7 Command Polling

During normal operation of the embedded application, a poll is done to check for user defined commands that have been issued by the host. When a command is detected the appropriate action is taken. Once the action is completed the command status is either reset or set to a status (see list below) to let the host know that it can proceed with the task at hand.

```
/* PC Master Command Code Response Enumerations */
enum
{
      CMDST_OK = 1,
      CMDST_SFLASH_PAGE_READY,
      CMDST_SFLASH_NOT_ERASED,
      CMDST_INVALID_PARAMETER,
      CMDST_X_DATA_READY,
      CMDST_P_DATA_READY,
      CMDST_XFLASH_RW_ERROR,
      CMDST_PFLASH_RW_ERROR,
      CMDST_NUC_FLASH_BUSY,
      CMDST_INVALID_PIN,
      CMDST_MEMORY_LOCKED,

      CMDST_TIMEOUT,
      CMDST_UNDEFINED
};
```

The use of a command status other than CMDST_OK is typically used for providing the host with an indication of the result of the latest command. For example, when reading a page of serial data flash, two commands are required. The first would be CMD_COPY_SFLASH_PAGE, which reads the data from flash, places the data in the scratch pad buffer, and then sets the command status to CMDST_SFLASH_PAGE_READY. When the host sees that the data is available (using the McbGetAppCmdStatus routine), it can execute a McbReadDataMem command to retrieve that data.

As long as the return status is not MCB_APPCMDRESULT_RUNNING a command may be issued to the camera.

6.8 Access to DSP Peripheral Registers (ArchIO)

The DSP peripheral registers are mapped into data memory space. These registers can be read from or written to using the standard 'McbReadDataMem/McbWriteDataMem' functions. Since the embedded application has built in drivers controlling the enabled peripherals, it is not recommended that the host application modifies any of these registers. It can be useful though to read various registers directly once the embedded application has initialized the peripherals. For example by reading the memory at the proper address, all of the current ADC count results can be obtained. Using the proper equations, these values can be converted to temperatures for the user application.

Information on these registers can be obtained in the Motorola *DSP56F80X User's Manual (DSP56F801-7UM/D)*.

The base register is mapped to memory address location 0x1000.

6.9 Access to Xilinx FPGA Registers (FpgaIO)

The Xilinx FPGA registers are mapped into DSP external memory data space. The data structure (or register map) is shown in Appendix B. These registers can be read from or written to using the standard 'McbReadDataMem/McbWriteDataMem' functions, although it is not recommended that these registers are written to unless the user fully understands the results of the register modification.

- The base register is mapped to address location 0x4000.
- The registers are only available after the FPGA has been configured by embedded application.

There are only a couple of registers that the host will typically modify. They are listed in the following paragraphs.

6.9.1 FPA Processor Operational Control Register Low

Address: 0x4000

Bit 0 (Unit Gain): 0 – Use NUC Coefficient Memory Gain; 1 – Force NUC Gain to Unity (1.0)

Bit 1 (Zero Ofst): 0 – Use NUC Coefficient Memory Offset; 1 – Force NUC Offset to Zero

Bit 2 (Zero Rf Ofst): 0 – Use Utility Memory NUC Refresh Offset Coefficient; 1 – Force NUC Refresh Offset to Zero

Bit 3 (Zero Px Rpl): 0 – Use NUC Coefficient Memory Pixel Replace Address; 1 – Force Defective Pixels to Zero

Bit 4 (Cam Pwr Dwn): 0 – Normal Camera Operation; 1 – Force Camera Timing into Power Down State (Set on power down detect interrupt)

Bit 6 (Act Itt Sel): 0 – Low ITT Applied to FPA Video; 1 – High ITT Applied to FPA Video

Bit 8 (Dspl Act): 0 – Blank FPA Image on Display; 1 – Enable FPA Image on Display

Bit 9 (Dspl Frz Mod): 0 – Normal (Live) FPA Image on Display; 1 –Freeze FPA Image on Display (Previously Captured in Image Grab Buffer B)

Bit 10 (Dspl Zm Mod): 0 – 1X FPA Image on Display; 1 –2X FPA Image on Display

Bits 11-12 (Dspl Byte Sel): 0 – ITT/FB Bits 7:0 Displayed; 1 – ITT/FB Bits 15:8 Displayed; 2 – FB Bits 17:10 Displayed (when in Freeze Mode); 3 - Reserved

Bit 13 (Ovl Act): 0 – Disable Overlays on Display; 1 – Enable Overlays on Display

Bit 15 (Pfv Act): 0 – Disable Processed FPA Video Port Signals; 1 – Enable Processed FPA Video Port Signals

6.9.2 FPA Processor Operational Control Register High

Address: 0x4001

Bit 0 (DFld Intr En): 0 – Disable Display Field Interrupt; 1 – Enable Display Field Interrupt

Bit 1 (FFrm Intr En): 0 – Disable FPA Frame Interrupt; 1 – Enable FPA Frame Interrupt

Bits 4-5 (EC Mem Dev Acc): 0 – No Memory Access; 1 - Instrumentation Header Memory Access; 2 – Color Palette Y Access; 3 – Color Palette Cr/Cb Access

Bits 8-9 (Tst Mux Sel): Test Mux Select: 0 – Digital FPA Video (Normal Operation); 1 – Test Count; 2 – Force 0; 3 – Force 1 (0x2000)

Bit 15 (Pfv FCnt En): 0 – Disable Processed FPA Video Frame Counter (Force to Zero); 1 – Enable Processed FPA Video Frame Counter

6.9.3 FPA Processor User Mode Control Register

Address: 0x4002

Bits 0-1 (Mstr Sync Mod): 0 – Internal (Use Programmable Sync Generator); 1 – Reserved; 2 – External (Field Toggle); 3 – External (Field Coherent)

Bit 6 (Dspl Vid Pol): 0 – Normal (“White Hot”); 1 – Inverted (“Black Hot”)

Bit 7 (Clr Bar En): 0 – Disable Display Color Bar; 1 – Enable Display Color Bar

Bit 8 (Clr Plt Y Sel): 0 – Low Byte of Color Palette Displayed (Normal Mode); 1 – High Byte of Color Palette Displayed (Gamma Corrected)

Bits 10-11 (Pfv Src Sel): Processed FPA Video Source Select: 0 – Digital FPA Video; 1 – NUC Corrected Video; 2 – Pixel Replaced Video; 3 – ITT Video

6.9.4 ATC Offset Coefficient Register

Address: 0x4007

16-Bit Signed Constant Added to Output of NUC Circuit (Prior to NUC Refresh)

-16,384 <= Atc Ofst <= 16,383.5

6.10 Access to Serial Data Flash

The Atmel serial data flash chip has 4096 pages of storage each with 264 bytes. The pages have been allocated as follows for the embedded application.

Page(s)	Allocated Use
0 - 7	Reserved
8 - 23	Op Mode Descriptor Tables (16x)
24 - 87	NUC Mode Descriptor Tables (64x)
88 - 127	Reserved
128 - 159	Zone Statistic Descriptor Tables (32x) (Currently not used)
160 - 190	Reserved
191	Overlay Palettes (8x)
192 - 223	Color Palette Y Tables (16x)
224 - 255	Color Palette Cr/Cb Tables (16x)
256 - 319	User Defect Pixel Lists (64x) (Currently not used)
320 - 511	Reserved
512 - 1145	FPA Processor FPGA Configuration File
1146 - 1535	Expansion FPGA Configuration File
1536 - 2175	Stored Image 0 (Currently not used)
2176 - 2815	Stored Image 1 (Currently not used)
2816 - 3455	Stored Image 2 (Currently not used)
3456 - 4095	Stored Image 3 (Currently not used)

The host can read/write to these locations through the use of the CMD_COPY_SFLASH_PAGE, CMD_PROG_SFLASH_FULL, CMD_PROG_SFLASH_PARTIAL commands. These commands are implemented in Lumitron's host applications to load palettes, FPGA configuration files, operational mode descriptors, and NUC mode descriptors.

7 Remote Calibration Process

The process to complete a non-uniformity calibration is initiated with a Lumitron defined command. After the calibration command has been issued, the embedded application communicates with the host based on the state of the "CameraConfig.camStats.HostStatusCode" global structure member. This variable, hereafter referred to as status code, has its state set by either the embedded application or the host application depending upon the next step in the process.

Status codes that are used in the calibration or defective pixel detection process are as follows:

```
/* Calibration and Defect Communication Status Enumerations */
enum
{
      HOST_READY = 0xFFFF,
      BEGIN_PROCESS = 0,
      CAL_PLACE_COLD_REF,
      CAL_COLD_REF_IN_PLACE,
      CAL_PLACE_HOT_REF,
      CAL_HOT_REF_IN_PLACE,
      CAL_DELTA_TOO_SMALL,
      CAL_COMPLETED,
      CAL_CALC_COEFFICIENTS,
      DEFECT_TOO_MANY,
      DEFECT_ERASE_FLASH,
      DEFECT_ERASE_ACK,
      DEFECT_COMPLETED,
      UNIFORMITY_TEST_IN_PROGRESS,
      NUC_FLASH_PROGRAMMED,

      HOST_UNDEFINED
};
```

7.1 One Point Refresh Calibration (Internal Flag):

(A) Send CMD_ONE_PT_REFRESH command with argument for using internal calibration flag.

(B) Embedded application will set the status code to BEGIN_PROCESS, and then begin execution of the calibration.

(C) Host may monitor (periodically read) the status code data member until it is set by the embedded application to CAL_COMPLETED.

(D) Calibration is complete

7.2 One Point Refresh Calibration (External Flag):

(A) Send CMD_ONE_PT_REFRESH command with argument for using external calibration flag.

(B) Embedded application will set the status code to BEGIN_PROCESS, and then begin execution of the calibration.

(C) Host will monitor (periodically read) the status code data member until it is set by the embedded application to CAL_PLACE_COLD_REF.

(D) Host/User automatically or manually places the cold reference in the sensor field of view.

(E) Host application sets the status code to CAL_COLD_REF_IN_PLACE.

(F) Host will monitor the status code data member until it is set by the embedded application to CAL_CALC_COEFFICIENTS or CAL_COMPLETED.

(G) The host/user may now remove the cold reference if desired.

(H) Calibration is complete

7.3 One Point Update Calibration (Internal Flag):

(A) Send CMD_ONE_PT_UPDATE command with argument for using internal calibration flag.

(B) Embedded application will set the status code to BEGIN_PROCESS, and then begin execution of the calibration.

(C) Host will monitor (periodically read) the status code data member until it is set by the embedded application to CAL_COMPLETED.

(D) Read the “CameraConfig.camStats.HostData” member to retrieve number of defects if desired.

(E) Calibration is complete.

7.4 One Point Update Calibration (External Flag):

(A) Send CMD_ONE_PT_UPDATE command with argument for using external calibration flag.

(B) Embedded application will set the status code to BEGIN_PROCESS, and then begin execution of the calibration.

(C) Host will monitor (periodically read) the status code data member until it is set by the embedded application to CAL_PLACE_COLD_REF.

(D) Host/User automatically or manually places the cold reference in the sensor field of view.

(E) Host application sets the status code to CAL_COLD_REF_IN_PLACE.

(F) Host will monitor the status code data member until it is set by the embedded application to CAL_CALC_COEFFICIENTS.

(G) The host/user may now remove the cold reference if desired.

(H) Host will monitor (periodically read) the status code data member until it is set by the embedded application to CAL_COMPLETED.

(I) Read the “CameraConfig.camStats.HostData” member to retrieve number of defects if desired.

(J) Calibration is complete.

7.5 Two Point Calibration (Internal Flags):

(A) Send CMD_TWO_PT_NUC command with argument for using internal calibration flags.

(B) Embedded application will set the status code to BEGIN_PROCESS, and then begin execution of the calibration.

(C) Host will monitor the status code data member until it is set by the embedded application to CAL_CALC_COEFFICIENTS or CAL_DELTA_TOO_SMALL. If the CAL_DELTA_TOO_SMALL code is received then the embedded application has aborted the calibration.

(D) Host will monitor (periodically read) the status code data member until it is set by the embedded application to CAL_COMPLETED.

(E) Read the “CameraConfig.camStats.HostData” member to retrieve number of defects if desired.

(F) Calibration is complete.

7.6 Two Point Calibration (External Flags):

(G) Send CMD_TWO_PT_NUC command with argument for using external calibration flags.

(H) Embedded application will set the status code to BEGIN_PROCESS, and then begin execution of the calibration.

(I) Host will monitor (periodically read) the status code data member until it is set by the embedded application to CAL_PLACE_COLD_REF.

(J) Host/User automatically or manually places the cold reference in the sensor field of view.

(K) Host application sets the status code to CAL_COLD_REF_IN_PLACE.

(L) Host will monitor (periodically read) the status code data member until it is set by the embedded application to CAL_PLACE_HOT_REF.

(M) Host/User automatically or manually places the hot reference in the sensor field of view.

(N) Host application sets the status code to CAL_HOT_REF_IN_PLACE

(O) Host will monitor the status code data member until it is set by the embedded application to CAL_CALC_COEFFICIENTS or CAL_DELTA_TOO_SMALL. If the CAL_DELTA_TOO_SMALL code is received then the embedded application has aborted the calibration.

(P) The host/user may now remove the hot reference if desired.

(Q) Host will monitor (periodically read) the status code data member until it is set by the embedded application to CAL_COMPLETED.

(R) Read the “CameraConfig.camStats.HostData” member to retrieve number of defects if desired.

(S) Calibration is complete.

7.7 Defective Pixel Detection

Lumitron has removed the automatic detection feature from the camera. Defective pixels are now solely identified through calibrations and external operations defined by the user.

7.8 User Defined Defective Pixel Map

This process can be followed to supply a user defined defective pixel map to the active NUC table. It assumes the host already has created a defective pixel map of the entire array. The pixels will marked as defective in the active NUC table, which will be erased in the process. A 2-point calibration is required after this process is complete.

(A) Disable the automatic NUC refresh operation by setting BIT-15 of the camera configuration bitFieldIndex member.

(B) Using the ‘McbWriteDataMem’ command - write a portion of the map to the scratch pad buffer. Remember it is sized at 160 words.

(C) Send the CMD_WRITE_UTILITY_MEMORY command with the proper address offset (base is MAR_IMAGE_GRAB_B) and data size (see paragraph 5.3.43) to move the data from the DSP scratch pad to the Xilinx utility memory.

(D) Repeat steps (B) and (C) until the entire map is loaded to utility memory.

(A) Send CMD_ADV_DETECT_BAD_PIXELS command (no arguments).

(B) Embedded application will set the status code to BEGIN_PROCESS, and then begin execution of the routine.

(C) Host will monitor (periodically read) the status code data member until it is set by the embedded application to DEFECT_ERASE_FLASH.

(D) If DEFECT_ERASE_FLASH status is detected then the embedded application needs to know whether the host wants the NUC flash erased or not.

(E) Host needs to set the “CameraConfig.camStats.HostData” member to 0xFFFF for erase flash or to 0x0000 to skip the erase process and exit.

(F) Host application sets the status code to DEFECT_ERASE_ACK.

(G) Host will monitor the status code data member until it is set by the embedded application to DEFECT_COMPLETED.

(H) Re-enable automatic NUC refresh bit.

(I) The user defective map now exists in the active NUC flash and is ready for a two point calibration.

7.9 Upload NUC Table from Host

This process can be followed to supply a host created set of NUC coefficients to the desired table index. It assumes the host already has created the offsets, gains, and replacement index (if needed)

for each pixel in the entire array. The NUC data will also need to be formatted for use in the hardware (see Appendix F). It will also be necessary to save NUC mode data to serial data flash that corresponds to the uploaded NUC data (see paragraph 5.3.2 for further details).

(A) Disable the automatic NUC refresh operation by setting BIT-15 of the camera configuration bitFieldIndex member.

(B) Set up for the first pass (gain terms only).

(C) Using the 'McbWriteDataMem' command - write a portion of the gain coefficients to the scratch pad buffer. Remember it is sized at 160 words.

(D) Send the CMD_WRITE_UTILITY_MEMORY command with the proper address offset (base is MAR_IMAGE_GRAB_B) and data size (see paragraph 5.3.43) to move the data from the DSP scratch pad to the Xilinx utility memory.

(E) Repeat steps (C) and (D) until the entire set of gain coefficients is loaded contiguously to utility memory.

(F) Send CMD_UPLOAD_NUC command with the arguments set for 'gain terms' and desired NUC base.

(G) Embedded application will set the status code to BEGIN_PROCESS, and then begin execution of the routine.

(H) The entire NUC Flash block is erased on the upload of gain terms only, which is why they are uploaded first.

(I) Host will monitor (periodically read) the status code data member until it is set by the embedded application to NUC_FLASH_PROGRAMMED or HOST_READY.

(J) Set up for the second pass (offset terms only).

(K) Using the 'McbWriteDataMem' command - write a portion of the offset coefficients to the scratch pad buffer. Remember it is sized at 160 words.

(L) Send the CMD_WRITE_UTILITY_MEMORY command with the proper address offset (base is MAR_IMAGE_GRAB_B) and data size (see paragraph 5.3.43) to move the data from the DSP scratch pad to the Xilinx utility memory.

(M) Repeat steps (K) and (L) until the entire set of offset coefficients is loaded to utility memory.

(N) Send CMD_UPLOAD_NUC command with the arguments for 'offset terms' and desired NUC base.

(O) Embedded application will set the status code to BEGIN_PROCESS, and then begin execution of the routine.

(P) Host will monitor (periodically read) the status code data member until it is set by the embedded application to NUC_FLASH_PROGRAMMED or HOST_READY.

(Q) Re-enable automatic NUC refresh bit.

(R) The host supplied NUC coefficient set now exists in the desired NUC flash table and is ready for use.

7.10 Download NUC Table to Host

This process can be followed to retrieve a set of NUC coefficients from the camera at the desired table index. It assumes the host already has created a buffer for the storage of the coefficients. The NUC data will be read in a form that is formatted for use in the camera hardware (see Appendix F), so the host may need to perform casting/conversion before using it locally.

(A) Set up for the first pass (gain terms only).

(B) Send CMD_DOWNLOAD_NUC command with the arguments set for ‘gain terms’ and desired NUC base. This will instruct the camera to read the NUC flash and store the gain terms in utility memory (at base of MAR_IMAGE_GRAB_B).

(C) Host will monitor (periodically read) the status code data member until it is set by the embedded application to HOST_READY.

(D) Send the CMD_READ_UTILITY_MEMORY command with the proper address offset (base is MAR_IMAGE_GRAB_B) and data size (see paragraph 5.3.15) to move the data from the Xilinx utility memory to the DSP scratch pad memory area.

(E) Using the ‘McbReadDataMem’ command - read a portion of the gain coefficients to the scratch pad buffer. Remember it is sized at 160 words.

(F) Repeat steps (D) and (E) until the entire set of gain coefficients is downloaded to local host memory.

(G) Set up for the second pass (offset terms only).

(H) Send CMD_DOWNLOAD_NUC command with the arguments set for ‘offset terms’ and desired NUC base. This will instruct the camera to read the NUC flash and store the offset terms in utility memory (at base of MAR_IMAGE_GRAB_B).

(I) Host will monitor (periodically read) the status code data member until it is set by the embedded application to HOST_READY.

(J) Send the CMD_READ_UTILITY_MEMORY command with the proper address offset (base is MAR_IMAGE_GRAB_B) and data size (see paragraph 5.3.15) to move the data from the Xilinx utility memory to the DSP scratch pad memory area.

(K) Using the ‘McbReadDataMem’ command - read a portion of the offset coefficients to the scratch pad buffer. Remember it is sized at 160 words.

(L) Repeat steps (J) and (K) until the entire set of offset coefficients is downloaded to local host memory.

(M) The host now has a local copy of the desired NUC flash table.

Appendix A - Camera Configuration Data Structures

```
/*

Current Camera Configuration Information Structure:

This structure will contain information about the camera that will be useful for standard
operation as well as for the remote user to have access.  Therefore this data structure is fixed
in memory so that the info is always available via PCMaster.

*/
struct _CAMERA_CONFIG
{
      /* NVM Config Data for Global Access */
      NVM_GLOBAL_CFG    nvmData;               /* 40 Words */

      /* NVM Data Update Flag */
      UWord16           updateNVM;             /* 1 Word */

      /* Debug Values for Development Platform */
      UWord16           continueFlag;          /* 1 Word  */
      UWord16           CmdsReceived;          /* 1 Word  */

      /* Camera Status Codes */
      CAMERA_STATUS     camStats;              /* 4 Words */

    /* Camera Error Codes */
      CAMERA_ERRORS     camErrors;             /* 8 Words */

      /* Current camera time */
      RTC_DATA          camTime;               /* 7 Words */

      /* Mirror of Output Port Expansion Register */
      UWord16           expPort;               /* 1 Word */

      /* Software Version */
      UWord16           swVersion;             /* 1 Word */

      /* Software Build */
      UWord16           swBuild;               /* 1 Word */

      /* FPGA Proc Version */
      UWord16           fpgaVersion[2];        /* 2 Words */

      /* FPA Dimensions */
      IMAGE_SIZE        fpaSize;               /* 2 Words */

      /* Global AGC Parameters */
      UWord16           agcLowIntensity;       /* 1 Word */
      UWord16           agcHighIntensity;      /* 1 Word */

      /* Op Mode Name */
      UWord16           actOpName[4];          /* 4 Words */

      /* Nuc Mode Name */
      UWord16           actNucName[4];         /* 4 Words */

      /* Misc Index Values */
      UWord16           bitFieldIndex;         /* 1 Word */
      // fanSpeedIndex:3;     /* Fan Speed Index, (Bits 0 - 2) */
      // baseNuc:6;           /* Base NUC table, (Bits 3 - 8) */
      // limitNuc:6;          /* Limit NUC table, (Bits 9 - 14) */
      // noRefresh:1;         /* Lock out refresh, (Bit 15) */

      /* Radiometric Softwre Data */
      UWord16           radSWInfo;             /* 1 Word */

      /* Camera alarm state info data */
      UWord16           alarm;                 /* 1 Word */
```

```
	/* Cal-flag Settings data */
	UWord16           calFlagRefs;           /* 1 Word */
	// coldRef:8;           /* Cold Ref Setting, (Bits 0 - 7) */
	// hotRef:8;            /* Hot Ref Setting, (Bits 8 - 15) */

	/* FPA Type Information */
	UWord16           fpaInfo;               /* 1 Word */
	// fpaType:4;           /* FPA Type Identifier (Bits 0 - 3) */
	// fpaSubType:4;        /* FPA Sub-Type Identifier (Bits 4 - 7) */
	// retRangeEn:1;        /* Ranging Reticle Enable (Bit 8) */
	// notUsed:6;           /* Currently not used (Bits 9 - 14) */
	// lockMemMirror:1;     /* Set if memory unlocked (Bit 15) */

	/* Filtered ADC Chnl A Result Values (A0, A1, A2, A3) */
	UWord16           adcAFiltered[4];       // 4 Words

	/* Filtered ADC Chnl B Result Values (B0, B1, B2, B3) */
	UWord16           adcBFiltered[4];       // 4 Words

	/* Current Button Panel Status */
	BTN_PANEL         btnPanel;              // 2 Words

};
typedef struct _CAMERA_CONFIG CAMERA_CONFIG, *PTR_CAMERA_CONFIG;
```

```
/*

Partial Camera Dynamic Configuration Info Structure – matches what is stored in global memory.

*/
struct _NVM_GLOBAL_CFG
{
      UWord16     CamMode;          /* Cast to/from NVM_CAM_MODE */
      UWord16     FpaMode;          /* Cast to/from NVM_FPA_PROC_MODE */

      UWord16     ReserveA;         /* Reserved */

      UWord16     ActMode;          /* Cast to/from NVM_AUTO_NUC_MODE */

      UWord16     ReserveB;         /* Reserved */

      NVM_AUTO_NUC_MODE AutoNucData;      /* See Struct Above */

      UWord16     AutoRfshTime;     /* Automatic calibration time period */
      UWord16     AutoRfshTemp;     /* Automatic calibration temp delta (counts) */

      UWord16     ReserveC[2];      /* Reserved */

      UWord16     ActPal;           /* Cast to/from NVM_ACT_PAL */
      UWord16     OvlMode;          /* Cast to/from NVM_OVL_MODE */

      NVM_RETICLE_POS   RtclAPos;   /* See Struct Above */
      NVM_RETICLE_POS   RtclBPos;   /* See Struct Above */

      UWord16     RadMode;          /* Cast to/from RAD_MODE */

      UWord16     AgcMode;          /* Cast to/from NVM_AGC_MODE */

      UWord16     ReserveE;         /* Reserved */

      NVM_MANUAL_ITT    ManualITT;  /* See Struct Above */
      NVM_AGC_LIMITS    AgcLimits;  /* See Struct Above */

      UWord16     LinearMap;        /* Cast to/from LINEAR_MAP */

      UWord16     AgcBinLimit;      /* AGC Bin Limit Value */

      UWord16     ReserveF[4];      /* Reserved */

      UWord16     ActZoneStat;      /* Cast to/from NVM_ACT_ZONE_STATS */

      NVM_VID_SCALE_TEMPS     VidScaleTemps;    /* See Struct Above */

      UWord16     ImageParams;      /* Cast to/from NVM_IMG_PARAMS */

      UWord16     LensID;           /* Cast to/from NVM_LENS_ID */

      UWord16     ReserveG[3];      /* Reserved */

};
typedef struct _NVM_GLOBAL_CFG NVM_GLOBAL_CFG, *PTR_NVM_GLOBAL_CFG;
```

Appendix B - Xilinx Register/Data Structure

```
/* Xilinx FPGA Register Map Structure */
typedef struct
{
      UWord16           OpCtrlRegLo;           /* Map to 0x4X00 */
      UWord16           OpCtrlRegHi;           /* Map to 0x4X01 */
      UWord16           UserCtrlReg;           /* Map to 0x4X02 */
      UWord16           StaticCtrlReg;         /* Map to 0x4X03 */
      UWord16           ImgHistoGrabReg;       /* Map to 0x4X04 */
      UWord16           InterruptReg;          /* Map to 0x4X05 */
      UWord16           Spare0Reg;             /* Map to 0x4X06 */
      UWord16           AtcOffsetCoefReg;      /* Map to 0x4X07 */
      UWord16           Reserved0[8];          /* Map to 0x4X08 - 0x4X0F */

      UWord16           Reserved1[16];         /* Map to 0x4X10 - 0x4X1F */

      UWord16           ProgSyncRegLo;         /* Map to 0x4X20 */
      UWord16           ProgSyncRegHi;         /* Map to 0x4X21 */
      UWord16           DspHorCntPreReg;       /* Map to 0x4X22 */
      UWord16           DspHorImgStartReg;     /* Map to 0x4X23 */
      UWord16           DspHorImgStopReg;      /* Map to 0x4X24 */
      UWord16           DspVerCntPreReg;       /* Map to 0x4X25 */
      UWord16           DspVerImgStartReg;     /* Map to 0x4X26 */
      UWord16           DspVerImgStopReg;      /* Map to 0x4X27 */
      UWord16           FpaSyncDlyRegLo;       /* Map to 0x4X28 */
      UWord16           FpaSyncDlyRegHi;       /* Map to 0x4X29 */
      UWord16           FpaHorStartReg;        /* Map to 0x4X2A */
      UWord16           FpaHorStopReg;         /* Map to 0x4X2B */
      UWord16           FpaHorTermCntReg;      /* Map to 0x4X2C */
      UWord16           FpaVerStartReg;        /* Map to 0x4X2D */
      UWord16           FpaVerStopReg;         /* Map to 0x4X2E */
      UWord16           FpaVerTermCntReg;      /* Map to 0x4X2F */

      UWord16           FpaHorRoiStartReg;     /* Map to 0x4X30 */
      UWord16           FpaHorRoiStopReg;      /* Map to 0x4X31 */
      UWord16           FpaVerRoiStartReg;     /* Map to 0x4X32 */
      UWord16           FpaVerRoiStopReg;      /* Map to 0x4X33 */
      UWord16           Reserved2[12];         /* Map to 0x4X34 - 0x4X3F */

      UWord16           Reserved3[16];         /* Map to 0x4X40 - 0x4X4F */

      UWord16           Reserved4[16];         /* Map to 0x4X50 - 0x4X5F */

      UWord16           Reserved5[16];         /* Map to 0x4X60 - 0x4X6F */

      UWord16           FpgaProgReg;           /* Map to 0x4X70 */
      UWord16           Reserved6[15];         /* Map to 0x4X71 - 0x4X7F */

      UWord16           NucMemDatAcc;          /* Map to 0x4X80 */
      UWord16           NucMemDatAccInc;       /* Map to 0x4X81 */
      UWord16           MarNucMem;             /* Map to 0x4X82 */
      UWord16           NucTableBaseReg;       /* Map to 0x4X83 */
      UWord16           FpaIntegRegLo;         /* Map to 0x4X84 */
      UWord16           FpaIntegRegHi;         /* Map to 0x4X85 */
      UWord16           FpaSerCtrlWord0Reg;    /* Map to 0x4X86 */
      UWord16           FpaSerCtrlWord1Reg;    /* Map to 0x4X87 */
      UWord16           FpaSerCtrlWord2Reg;    /* Map to 0x4X88 */
      UWord16           FpaSerCtrlWord3Reg;    /* Map to 0x4X89 */
      UWord16           FpaSptDac0Reg;         /* Map to 0x4X8A */
      UWord16           FpaSptDac1Reg;         /* Map to 0x4X8B */
      UWord16           FpaSptDac2Reg;         /* Map to 0x4X8C */
      UWord16           FpaSptDac3Reg;         /* Map to 0x4X8D */
      UWord16           Reserved7[2];          /* Map to 0x4X8E - 0x4X8F */

      UWord16           Reserved8[16];         /* Map to 0x4X90 - 0x4X9F */

      /* Utility Memory Access via MAR A */
```

```
        UWord16           MarAMemAcc;            /* Map to 0x4XA0 */
        UWord16           MarAMemAccInc;         /* Map to 0x4XA1 */
        UWord16           MarAMemAccClr;         /* Map to 0x4XA2 */
        UWord16           MarAMemAccClrInc;      /* Map to 0x4XA3 */
        UWord16           MarAMemAccMsb;         /* Map to 0x4XA4 */
        UWord16           MarAMemAccMsbInc;      /* Map to 0x4XA5 */
        UWord16           MarAMemAccMsbClr;      /* Map to 0x4XA6 */
        UWord16           MarAMemAccMsbClrInc;   /* Map to 0x4XA7 */
        /* Utility Memory Access via MAR B */
        UWord16           MarBMemAcc;            /* Map to 0x4XA8 */
        UWord16           MarBMemAccInc;         /* Map to 0x4XA9 */
        UWord16           MarBMemAccClr;         /* Map to 0x4XAA */
        UWord16           MarBMemAccClrInc;      /* Map to 0x4XAB */
        UWord16           MarBMemAccMsb;         /* Map to 0x4XAC */
        UWord16           MarBMemAccMsbInc;      /* Map to 0x4XAD */
        UWord16           MarBMemAccMsbClr;      /* Map to 0x4XAE */
        UWord16           MarBMemAccMsbClrInc;   /* Map to 0x4XAF */
        /* Utility Memory MAR A */
        UWord16           MarAMem_LSW;           /* Map to 0x4XB0 */
        UWord16           Spare1Reg[3];          /* Map to 0x4XB1 - 0x4XB3 */
        UWord16           MarAMem_MSW;           /* Map to 0x4XB4 */
        UWord16           MarAMem_MSWInc;        /* Map to 0x4XB5 */
        UWord16           MarAMem_MSWClr;        /* Map to 0x4XB6 */
        UWord16           MarAMem_MSWClrInc;     /* Map to 0x4XB7 */
        /* Utility Memory MAR B */
        UWord16           MarBMem_LSW;           /* Map to 0x4XB8 */
        UWord16           Spare2Reg[3];          /* Map to 0x4XB9 - 0x4XBB */
        UWord16           MarBMem_MSW;           /* Map to 0x4XBC */
        UWord16           MarBMem_MSWInc;        /* Map to 0x4XBD */
        UWord16           MarBMem_MSWClr;        /* Map to 0x4XBE */
        UWord16           MarBMem_MSWClrInc;     /* Map to 0x4XBF */

} fpga_IO;
```

Appendix C - Camera Command Enumerations

```
/* PC Master Command Code Enumerations */
enum
{
    CMD_FPGA_UPDATE_AVAILABLE = 1,
    CMD_COPY_SFLASH_PAGE,          /* Copy a page of serial flash memory to DSP */
    CMD_PROG_SFLASH_FULL,          /* Program a full page to serial flash */
    CMD_PROG_SFLASH_PARTIAL,       /* Program a partial page to serial flash */
    CMD_PROG_PRODUCT_ID,           /* Program DSP Data flash with Product ID */
    CMD_READ_PRODUCT_ID,           /* Read Product ID from DSP Data flash */
    CMD_PROG_STATIC_CFG,           /* Program DSP Data flash with Static Config */
    CMD_READ_STATIC_CFG,           /* Read Static Config from DSP Data flash */
    CMD_GET_CAMERA_TIME,           /* Read the RTC time and store to scratch pad */
    CMD_SET_CAMERA_TIME,           /* Set the RTC time from scratch pad */
    CMD_GET_NVM_DATA,              /* Read NVM data block to scratch pad */
    CMD_SET_NVM_DATA,              /* Set NVM data block from scratch pad */
    CMD_FOCUS_MOTOR_FAR,           /* Call focus-out routine */
    CMD_FOCUS_MOTOR_NEAR,          /* Call focus-in routine */
    CMD_IMAGE_GRAB,                /* Grab image into utility memory (Buffer Supplied) */
    CMD_READ_UTILITY_MEMORY,       /* Read a block of utility memory (Address Supplied) */
    CMD_NUC_FLASH_RAMP,            /* Debug Command to write ramp to parameter blocks */
    CMD_NUC_FLASH_MEMORY,          /* Read a block of NUC flash memory (Address Supplied) */
    CMD_NUC_FLASH_TEST_PATTERN,    /* Debug Command to write test pattern to NUC blocks */
    CMD_TEC_DRV_ENABLE,            /* Enable/Disable TEC Drive */
    CMD_TEC_TEMP_SELECT,           /* Select TEC Temperature (High/Low) */
    CMD_CAL_FLAG_SERVO,            /* Call either the Open or Close Routine */
    CMD_CAL_FLAG_REFERENCE,        /* Call either the Heated or Ambient Routine */
    CMD_ONE_PT_REFRESH,            /* Perform a directed 1-point refresh calibration */
    CMD_TWO_PT_NUC,                /* Perform a directed 2-point calibration */
    CMD_LOAD_COLOR_PAL,            /* Load user selected color palette */
    CMD_LOAD_OVLY_PAL,             /* Load user selected overlay palette */
    CMD_PIN_CHECK,                 /* Verify Memory Unlock PIN */
    CMD_FAN_SPEED_OPERATION,       /* Command to Test Fan Operation */
    CMD_GET_ADC_VALUES,            /* DEBUG ADC RETRIEVAL FUNCTION */
    CMD_ENABLE_RETICLE,            /* Turn Reticle on/off */
    CMD_RETICLE_POSITION,          /* Move Reticle to desired location */
    CMD_ONE_PT_UPDATE,             /* Perform a directed 1-point update calibration */
    CMD_WRITE_VID_ENC_REG,         /* DEBUG VIDEO ENCODER WRITE REGISTER */
    CMD_PERFORM_TEST,              /* Perform  user supplied test */
    CMD_OVERLAY_REFRESH,           /* Refresh symbology overlay */
    CMD_FREEZE_IMAGE,              /* Freeze/Unfreeze Display Image */
    CMD_DETECT_BAD_PIXELS,         /* NO LONGER USED */
    CMD_IRCON_LOAD_LUT,            /* Custom LUT table creation for radiometrics */
    CMD_LOAD_RAD_PARAMS,           /* Load Rad Params for compile time software */
    CMD_RESET_PFV_COUNT,           /* Toggles PFV Frame count bit */
    CMD_CLEAR_CONTINUE_FLAG,       /* Clears the idle while fault continue flag */
    CMD_UNIFORMITY_TEST,           /* Command to initiate a uniformity (flat field noise) test*/
    CMD_WRITE_UTILITY_MEMORY,      /* Write a block of utility memory (Address Supplied) */
    CMD_ADV_DETECT_BAD_PIXELS,     /* Command to initiate flash update of advanced detection of
                                        pixel defects */
    CMD_UPLOAD_NUC,                /* Command to intiate flash write of uploaded NUC terms */
    CMD_DOWNLOAD_NUC,              /* Command to intiate flash read of active NUC terms */
    CMD_COMPILE_DEFECT_LISTS,      /* Command to initiate a compilation of NUC defects to FLASH */
    CMD_RESTORE_FACTORY_DEFECTS,   /* Command to NUC to factory defect state */
    CMD_ENABLE_RANGE_RETICLE,      /* Turn Ranging Reticle On/Off */
    CMD_INIT_NUC_TABLE,            /* Initialize NUC table (erase to all 0xFF's) */
    CMD_CAMERA_RECOVER,            /* Attempt to recover camera from internal errors */
    CMD_PLACEHOLDER,
    CMD_UPDATE_EXP_PORT,           /* Updates Expansion Port Register from CfgData Value */

    CMD_UNDEFINED
};
```

Appendix D - Mapping of Serial Non-Volatile Memory

```
/* Complete Camera Dynamic Configuration Info Structure */
struct _NVM_DYN_CFG
{
      UWord16     CamMode;          /* Cast to/from NVM_CAM_MODE */
      UWord16     FpaMode;          /* Cast to/from NVM_FPA_PROC_MODE */

      UWord16     ReserveA;         /* Reserved */

      UWord16     ActMode;          /* Cast to/from NVM_AUTO_NUC_MODE */

      UWord16     ReserveB;         /* Reserved */

      NVM_AUTO_NUC_MODE AutoNucData;      /* See Struct Above */

      UWord16     AutoRfshTime;     /* Automatic calibration time period */
      UWord16     AutoRfshTemp;     /* Automatic calibration temp delta (counts) */

      UWord16     ReserveC[2];      /* Reserved */

      UWord16     ActPal;           /* Cast to/from NVM_ACT_PAL */
      UWord16     OvlMode;          /* Cast to/from NVM_OVL_MODE */

      NVM_RETICLE_POS   RtclAPos;   /* See Struct Above */
      NVM_RETICLE_POS   RtclBPos;   /* See Struct Above */

      UWord16     RadMode;          /* Cast to/from RAD_MODE */

      UWord16     AgcMode;          /* Cast to/from NVM_AGC_MODE */

      UWord16     ReserveE;         /* Reserved */

      NVM_MANUAL_ITT    ManualITT;  /* See Struct Above */
      NVM_AGC_LIMITS    AgcLimits;  /* See Struct Above */

      UWord16     LinearMap;        /* Cast to/from LINEAR_MAP */

      UWord16     AgcBinLimit;      /* AGC Bin Limit Value */

      UWord16     ReserveF[4];      /* Reserved */

      UWord16     ActZoneStat;      /* Cast to/from NVM_ACT_ZONE_STATS */

      NVM_VID_SCALE_TEMPS     VidScaleTemps;    /* See Struct Above */

      UWord16     ImageParams;      /* Cast to/from NVM_IMG_PARAMS */

      UWord16     LensID;           /* Cast to/from NVM_LENS_ID */

      UWord16     ReserveG[3];      /* Reserved */

      UWord32     OpTimeMin;        /* Elapsed operational time minutes */

      UWord16     CalFlashCount;    /* Calibration flash cycle count */
      UWord16     CalRfshCount;     /* Calibration refresh cycle count */
      UWord16     OverTempCount;    /* Over temperature alarm count */
      UWord16     PwrOnResetCnt;    /* Power on/reset cycle count */

      UWord16     ReserveH[2];      /* Reserved */

};
typedef struct _NVM_DYN_CFG   NVM_DYN_CFG, *PTR_NVM_DYN_CFG;
```

Appendix E - Dynamic Memory Definitions

Note: The address (Adr) listed in the tables that follow refer to the RAM offset address inside the Real Time Clock/NVM RAM part.

Adr	Entry Name	Mode	Factory Initialize	Bit Fields 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
32	Camera Mode Control	Short R/W	0x0009	Rtcl B En	Rtcl A En	Res-er-ved	Zn Stat En	Rfsh On Sw	Rfsh On Boot	Auto Rfsh Tmp	Auto Rfsh Tim	Auto Nuc Sw	Atc En	Reserved		Pfv Out En	SVid Out En	CVid 1Out En	CVid 0Out En
34	FPA Processor User Mode Control	Short R/W	0x0000	Reserved				PFV Src Sel	Res-er-ved	Clr Plt YSel	Clr Bar En	Dspl Vid Pol	Reserved					Mstr Sync Mod	
36	Reserved	Short R/W	0x0000	Reserved															
38	Active Operational Mode	Byte R/W	0x00									Reserved				Act Op Mod			
39	Active NUC Index	Byte R/W	0x00									Reserved		Act Nuc Indx					
40	Reserved	Short R/W	0x0000	Reserved															
42	Auto NUC Switch Low Saturation Intensity Threshold	Short R/W	0x0200	Reserved				Auto NUC Switch Low Saturation Intensity Threshold											
44	Auto NUC Switch Low Saturation Count Threshold	Short R/W	0x03E8	Auto NUC Switch Low Saturation Count Threshold															
46	Auto NUC Switch High Saturation Intensity Threshold	Short R/W	0x3E00	Reserved				Auto NUC Switch High Saturation Intensity Threshold											
48	Auto NUC Switch High Saturation Count Threshold	Short R/W	0x03E8	Auto NUC Switch High Saturation Count Threshold															

Camera Mode Control:

```
CVid0 Out En      0 – Composite Video 0 Output Disabled (Hi-Z);
                  1 – Composite Video 0 Output Enabled (Low Impedance Drive)
CVid1 Out En      0 – Composite Video 1 Output Disabled (Hi-Z);
                  1 – Composite Video 1 Output Enabled (Low Impedance Drive)
SVid Out En       0 – SVideo Output Disabled (Hi-Z);
                  1 – SVideo Output Enabled (Low Impedance Drive)
Pfv Out En        0 – Processed FPA Video Output Port Disabled (Hi-Z);
                  1 – Processed FPA Video Port Enabled (Low Impedance Drive)
Atc En            0 – Disable Ambient Temperature Compensation;
                  1 – Enable Ambient Temperature Compensation
Auto Nuc Sw       0 – Disable Auto NUC Switch;
                  1 – Enable Auto NUC Switch
Auto Rfsh Tim     0 – Disable Elapsed Time Based 1-Pt Refresh;
                  1 – Enable Elapsed Time Based 1-Pt Refresh
Auto Rfsh Tmp     0 – Disable Temperature Excursion Based 1-Pt Refresh;
```

	1 – Enable Temperature Excursion Based 1-Pt Refresh
Rfsh On Boot	0 – Disable 1-Pt Refresh on Bootup;
	1 – Enable 1-Pt Refresh on Bootup (Camera waits until FPA Temperature Stable)
Rfsh On Sw	0 – Disable 1-Pt Refresh on NUC Switch;
	1 – Enable 1-Pt Refresh on NUC Switch
Zn Stat En	0 – Disable Zone Statistics Computation;
	1 – Enable Zone Statistics Computation
Rtcl A En	0 – Disable Reticle A Overlay;
	1 – Enable Reticle A Overlay
Rtcl B En	0 – Disable Reticle B Overlay;
	1 – Enable Reticle B Overlay

FPA Processor User Mode Control:
Non-Volatile Copy of FPA Processor User Mode Control Register.
Used to Restore FPA Processor Register on Power On Reset.

Active Operational Mode:
Selects the Currently Active Op Mode (0 <= Act Op Mod <= 15).
Use the Appropriate Op Mode Descriptor Table Values to Load the FPA Processor Registers.

Active NUC Index:
Selects the Currently Active NUC Index (0 <= Act Nuc Indx <= 63).
Use the Appropriate NUC Mode Descriptor Table Values to Load the FPA Processor Registers.
Limit the NUC Index to the Range Defined by the Current Active Operational Mode.

Auto NUC Switch Parameters:
If Auto Nuc Sw = 1, then Decrement the NUC Index by 1 if the Number of FPA Pixels that are Less than the Auto NUC Switch Low Saturation Intensity Threshold is Greater than the Auto NUC Switch Low Saturation Count Threshold. Likewise, Increment the NUC Index by 1 if the Number of FPA Pixels that are Greater than the Auto NUC Switch High Saturation Intensity Threshold is Greater than the Auto NUC Switch High Saturation Count Threshold. In Any Case, Limit the NUC Index to the Range Defined by the Current Active Operational Mode.

Adr	Entry Name	Mode	Factory Initialize	Bit Fields 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
50	Auto Refresh Time Period	Short R/W	0x0258	Auto Refresh Time Period (Seconds)
52	Auto Refresh Temperature Delta Threshold	Short R/W	0x00EA	Auto Refresh Temperature Delta Threshold (Temperature ADC Counts)
54 - 56	Reserved	Short R/W	0x0000	Reserved
58	Active Color Palette	Byte R/W	0x00	Reserved | Act Clr Plt
59	Active Overlay Palette	Byte R/W	0x00	Reserved | Act Ovl Plt
60	Overlay Control	Byte R/W	0xA1	Reticle Size | Ovl Mod
61	Overlay Color	Byte R/W	0xD0	Ovl Foreground Color | Ovl Background Color
62	Reticle A Horizontal Position	Short R/W	0x0064	Reserved | Reticle A Horizontal Position
64	Reticle A Vertical Position	Byte R/W	0x78 NTSC 0x80 PAL	Reticle A Vertical Position
65	Reticle A Emissivity	Byte R/W	0x00	Reticle A Emissivity (0 to $1 - 2^{-8}$)

Auto Refresh Calibrate Parameters:
If Auto Cal Tim = 1, then Perform 1 Point Offset Refresh NUC when the Auto Calibrate Time Period Expires.
If Auto Cal Tmp = 1, then Perform 1 Point Offset Refresh NUC when the FPA Temperature Deviates by + /- the Auto Calibrate Temperature Delta Threshold from the Temperature at which the Previous 1 Point Offset Refresh was Performed.

Active Color Palette:
Selects the Currently Active Color Palette (0 <= Act Clr Plt <= 15).
Use the Appropriate Color Palette Table Values to Load the Color Palette Memory.

Active Overlay Palette:
Selects the Currently Active Overlay Palette (0 <= Act Ovl Plt <= 7).
Use the Appropriate Overlay Palette Table Values to Load the Overlay Palette Memory.

Overlay Control:
Ovl Mod 0 - Overlay Off
1 – Overlay Date/Time
2 – Overlay Date/Time/Contrast
3 – Overlay Full Status
4 to 15 - Reserved

Reticle Size Radius of Overlay Reticules

Overlay Color:
Ovl Background Color Color Index into Active Overlay Palette for Overlay Background Fields.
Ovl Foreground Color Color Index into Active Overlay Palette for Overlay Foreground Fields.

Adr	Entry Name	Mode	Factory Initialize	Bit Fields 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
66	Reticle B Horizontal Position	Short R/W	0x00DC	Reserved							Reticle B Horizontal Position								
68	Reticle B Vertical Position	Byte R/W	0x78 NTSC 0x80 PAL									Reticle B Vertical Position							
69	Reticle B Emissivity	Byte R/W	0x00									Reticle B Emissivity (0 to 1 - 2^{-8})							
70	Reticle Radiometric Control	Short R/W	0x0000	Reserved															Temp Unit
72	AGC Mode	Byte R/W	0x02									Reserved				Agc Mod			
73	Active ROI Index	Byte R/W	0x00									Reserved						Act Roi Indx	
74	Reserved	Short R/W	0x0000	Reserved															
76	Display Video Manual Low Intensity	Short R/W	0x0000	Reserved							Display Video Manual Low Intensity (Input Intensity Mapped to 0) (0 to 16,383)								
78	Display Video Manual High Intensity	Short R/W	0x3FFF	Reserved							Display Video Manual High Intensity (Input Intensity Mapped to 255) (0 to 16,383)								
80	AGC High Limit Intensity	Short R/W	0x3FFF	Reserved							AGC Low Limit Intensity								

Reticle Parameters:
If Rtcl X En = 1, then Draw Reticle X at the Reticle X Horizontal / Vertical Position.
If Reticle Emissivity is 0, Report Intensity on Overlay.
If Reticle Emissivity is not 0 Report Temperature on Overlay.

Reticle Radiometric Control:
Temp Unit 0 – Display Temperatures in Fahrenheit;
           1 – Display Temperatures in Celsius

AGC Mode:
Agc Mod   0 – AGC/ALC Off (Hold Present State);
           1 – Manual Contrast Adjust;
           2 – Linear AGC/ALC;
           3 – Histogram AGC/ALC;
           4 to 15 – Reserved

Active ROI Index:
Selects the Currently Active ROI Index (0 <= Act ROI Index <= 7).
Use the Appropriate Op Mode Descriptor Table Values to Load the FPA Processor Registers.

Display Video Manual Low / High Intensity Parameters:
If Agc Mod = 1, then:
    Intensity Transform = (Input – Manual Low Intensity) * (255/(Manual High Intensity – Manual Low Intensity)).

AGC Limit Intensity Parameters:
See below.

Adr	Entry Name	Mode	Factory Initialize	Bit Fields 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
82	AGC High Limit Intensity	Short R/W	0x3FFF	Reserved; AGC High Limit Intensity
84	AGC Ancillary Control	Short R/W	0x0000	Reserved; Lag Filter Rate; Linear Hi Map Indx; Linear Low Map Indx
86	AGC Bin Limit	Short R/W	0x001E	AGC Bin Limit
88 - 94	Reserved	Short R/W	0x0000	Reserved
96	Active Zone Stats Table	Byte R/W	0x00	Reserved; Act Zn Stats Tbl
97	Reserved	Byte R/W	0x00	Reserved
98	Display Video Zero Scale Temperature	Short R/W	0x0000	Display Video Zero Scale Temperature (ºK – LSB is TBD)
100	Display Video Full Scale Temperature	Short R/W	0x0000	Display Video Full Scale Temperature (ºK – LSB is TBD)
102	Camera Background Temperature	Byte R/W	0x32	Sign; Camera Background Temperature (ºC – LSB is .5); Frct
103	Display Video Emissivity	Byte R/W	0x00	Display Video Emissivity (0 to 1 - 2^{-8})

AGC Limit Intensity Parameters:
If Agc Mod >= 2, then Ignore Histogram Intensities Below AGC Low Limit Intensity and Above AGC High Limit Intensity when Computing Intensity Transform Table.

AGC Ancillary Control:
Linear Low Map Index 0 – Linear AGC Low Point at 1.25%;
1 – 0.25%;
2 – 0.05%;
3 – 0.01%;
Linear Hi Map Index 0 – Linear AGC High Point at 98.75%;
1 – 99.75%;
2 – 99.95%;
3 – 99.99%;
Lag Filter Rate (AGC Filtering Speed) 0 – Slow (Gradual Change);
1 – Medium Slow;
2 – Medium Fast;
3 – Fast (Rapid Change)

Active Zone Statistics Table:
Selects the Currently Active Zone Statistics Table (0 <= Act Zn Stats Table <= 31). Use the Appropriate Zone Statistics Descriptor Table Values to Compute and Process the Required Zones. Currently not implemented.

Radiometric Display Video Parameters:
Display Video Zero Scale Temperature, Display Video Full Scale Temperature, Camera Background Temperature and Display Video Emissivity are Used with Customer Supplied Software to Map Video Display to Absolute Temperature Scale.

Adr	Entry Name	Mode	Factory Initialize	Bit Fields 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
104	Lens ID Control	Byte R/W	0x05									Lens Auto Det	Res-erved	Lens ID					
105	Reserved	Byte R/W	0x00									Reserved							
106 - 110	Reserved	Short R/W	0x0000	Reserved															
112	Operational Elapsed Time (LSBs)	Short R/W	0x0000	Operational Elapsed Time LSBs (Minutes)															
114	Operational Elapsed Time (MSBs)	Short R/W	0x0000	Operational Elapsed Time MSBs (Minutes)															
116	Cal Flash Cycle Count	Short R/W	0x0000	Cal Flash Cycle Count															
118	Cal Refresh Cycle Count	Short R/W	0x0000	Cal Refresh Cycle Count															
120	Over-Temperature Alarm Count	Short R/W	0x0000	Over-Temperature Alarm Count															
122	Power On / Reset Cycle Count	Short R/W	0x0000	Power On / Reset Cycle Count															
124- 126	Reserved	Short R/W	0x0000	Reserved															

Lens ID Control:
Lens ID 0 – No Lens;

Fixed Focus / No Zoom (IDs 1 through 15):
1 – Reserved; 2 – Reserved; 3 – 13mm; 4 – Reserved; 5 – 25mm; 6 – Reserved;
7 – 50mm; 8 – Reserved; 9 – Reserved; 10 – 100mm; 11 to 15 – Reserved;

Manual Focus / No Zoom (IDs 16 through 31):
16 – Reserved; 19 – 13mm; 20 – Reserved; 21 – 25mm; 22 – Reserved;
23 – 50mm; 24 – Reserved; 25 – Reserved; 26 – 100mm; 27 to 31 – Reserved;
32 to 63 – Reserved for Advanced Lenses

Lens Auto Det 0 – Use Dynamic Configuration Table Lens ID;
1 – Automatically Detect Lens ID Using ADC Channel ANA4 (MS6 Bits)

Nonvolatile Diagnostic Parameters:
Operational Elapsed Time – Incremented Once per Minute

Cal Flash Cycle Count – Incremented Each Time a One-Point, Two-Point or One Point Update NUC Operation is Performed.

Cal Refresh Cycle Count – Incremented Each Time a One Point Refresh NUC Operation is performed.

Over-Temperature Alarm Count – Incremented Each Time the Camera Internal Temperature Exceeds 55ºC.

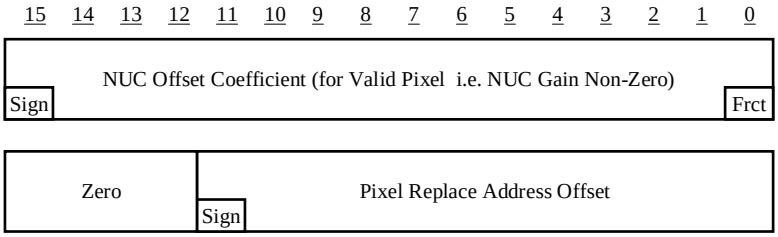
Power On/Reset Cycle Count – Incremented Each Time the Camera is Booted.

Appendix F – NUC Coefficient Format

NUC Offset Coefficient (Odd Addresses)

Range: -16,384 to +16,383.5 Resolution: 0.5
NUC Offset Coefficient Becomes Pixel Replace Address Offset
if NUC Gain Coefficient Equals Zero

Bit Fields



Pixel Replace Address Offset

Range: -643 to +643 Resolution: 1
Pixel Replace Address Offset = (Replace Relative Row Index * Pixels / Line) +
Replace Relative Col Index
where: –2 <= Replace Relative Row Index <= +2 and
-3 <= Replace Relative Col Index <= +3 and
Pixels / Line <= 320

NUC Gain Coefficient (Even Addresses)

Range: 0 to ~2 Resolution: 2^{-15}
Set NUC Gain Coefficient to Zero if Defective Pixel

Bit Fields