



**Serial Interface Developers Kit Reference Manual
for
Camera Link Based Cameras**

(Document Number 700-0000061-R01)

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

Serial Interface Developers Kit
Document Number 700-0000061
April 2006

Copyright Lumitron 2006
All Rights Reserved.

DISCLAIMER

All copyrights in this manual, and the hardware and software described in it, are the exclusive property of Lumitron, Inc. and its licensors. Claim of copyright does not imply waiver of Lumitron's or its licensor's other rights in the work. See the following Notice of Proprietary Rights.

NOTICE OF PROPRIETARY RIGHTS

This manual and the related hardware and software are confidential trade secrets and the property of Lumitron and its licensors. Use, examination, reproduction, copying, transfer and/or disclosure to others of all or any part of this manual and the related documentation is prohibited except with the express written consent of Lumitron.

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	1
2 GENERAL REQUIREMENTS	2
3 CAMERA BOOT SEQUENCE	3
3.1 Communication Configuration	3
3.2 Boot Messages.....	3
3.3 Software Upload.....	4
4 INTERFACE PROTOCOL	5
4.1 PC Master Information.....	5
4.2 Communications Configuration	5
5 HOST SIDE INTERFACE	6
5.1 Interface Information	6
5.2 Miscellaneous Serial Port Exports	6
5.2.1 csi_PortDetect	6
5.2.2 csi_GetNumPorts.....	6
5.2.3 csi_GetPortInfo	7
5.2.4 csi_GetActivePort	7
5.2.5 csi_OpenPort	7
5.2.6 csi_ConfigurePort	8
5.2.7 csi_ClosePort.....	8
5.2.8 csi_GetLastError	8
5.2.9 csi_EnterTerminalMode.....	8
5.2.10 csi_PauseTerminalMode	9
5.2.11 csi_ResumeTerminalMode.....	9
5.3 Camera PC Master Abstracted Function Call Exports	10
5.3.1 csi_PCMDetect	10
5.3.2 csi_PCMSendAppCmd	10
5.3.3 csi_PCMReadDataMem	11
5.3.4 csi_PCMWriteDataMem	12
5.3.5 csi_PCMWriteDataMemMask.....	12
5.4 S-Record Pointer and Size Exports.....	13
5.4.1 csi_SetSRecordPtr	13
5.4.2 csi_SetSRecordSize	15
5.4.3 csi_ResetSRecordInfo	15
5.4.4 csi_SetTerminalEvent.....	15
5.5 Camera Status and Status Code Exports	16

5.5.1 csi_Connected	16
5.5.2 csi_GetPCMStatus	16
5.5.3 csi_GetProcessCode	16
5.5.4 csi_GetProgressCode.....	17
5.5.5 csi_GetCamErrorCode	17
5.5.6 csi_GetCalStatusCode	18
5.6 Serial Flash Data Access Exports	19
5.6.1 csi_ReadSerialFlashPage	19
5.6.2 csi_ReadSerialFlashBlock.....	20
5.6.3 csi_WriteSerialFlashPage.....	20
5.6.4 csi_WritePartialSerialFlashPage	21
5.7 NUC Flash Data Access Exports	22
5.7.1 csi_ReadNucFlashMemory.....	22
5.8 Camera Product ID Access Exports.....	22
5.8.1 csi_ReadProductID.....	22
5.8.2 csi_WriteProductID	23
5.9 Camera Static Configuration Access Exports	23
5.9.1 csi_ReadStaticCfg	23
5.9.2 csi_WriteStaticCfg	24
5.10 Camera Utility Memory Access Exports	24
5.10.1 csi_ReadUtilityMemory	24
5.10.2 csi_WriteUtilityMemory	25
5.11 Camera Time Exports	26
5.11.1 csi_ReadCameraTime	26
5.11.2 csi_WriteCameraTime	26
5.12 Camera NVM Memory Access Exports.....	27
5.12.1 csi_ReadNVMBlock.....	27
5.12.2 csi_WriteNVMBlock	28
5.13 Camera Scaled Emittance Table Exports	28
5.13.1 csi_WriteAtcLUT	28
5.13.2 csi_ReadAtcLUT	29
5.14 Camera DAC 0 Table Exports	29
5.14.1 csi_ReadDac0LUT.....	29
5.14.2 csi_WriteDac0LUT.....	29
5.15 Camera Focus Exports.....	29
5.15.1 csi_FocusFar	29
5.15.2 csi_FocusNear	30
5.16 Miscellaneous Camera Command Exports	30
5.16.1 csi_ForceNVMUpdate.....	30
5.16.2 csi_GrabSingleFrameImage.....	31
5.16.3 csi_GetLensDescriptor	32
5.16.4 csi_TempRefreshDisable.....	32
5.16.5 csi_CallInProgress.....	32
5.16.6 csi_FrameInterruptDisable.....	33

5.16.7 csi_SetServoMode.....	33
5.16.8 csi_UpdateModelInfo.....	33

6 CUSTOM COMMANDS 35

6.1 Operational Software Defined Commands 35

6.1.1 CMD_COPY_SFLASH_PAGE	36
6.1.2 CMD_PROG_SFLASH_FULL	36
6.1.3 CMD_PROG_SFLASH_PARTIAL.....	36
6.1.4 CMD_PROG_PRODUCT_ID.....	36
6.1.5 CMD_READ_PRODUCT_ID	37
6.1.6 CMD_PROG_STATIC_CFG.....	37
6.1.7 CMD_READ_STATIC_CFG.....	38
6.1.8 CMD_GET_CAMERA_TIME	38
6.1.9 CMD_SET_CAMERA_TIME.....	38
6.1.10 CMD_GET_NVM_DATA.....	39
6.1.11 CMD_SET_NVM_DATA	39
6.1.12 CMD_FOCUS_MOTOR_FAR	39
6.1.13 CMD_FOCUS_MOTOR_NEAR	40
6.1.14 CMD_IMAGE_GRAB.....	40
6.1.15 CMD_READ_UTILITY_MEMORY.....	40
6.1.16 CMD_NUC_FLASH_RAMP.....	40
6.1.17 CMD_NUC_FLASH_MEMORY.....	41
6.1.18 CMD_NUC_FLASH_TEST_PATTERN.....	41
6.1.19 CMD_TEC_DRV_ENABLE	41
6.1.20 CMD_TEC_TEMP_SELECT	41
6.1.21 CMD_CAL_FLAG_SERVO.....	42
6.1.22 CMD_CAL_FLAG_REFERENCE.....	42
6.1.23 CMD_ONE_PT_REFRESH.....	43
6.1.24 CMD_TWO_PT_NUC.....	Error! Bookmark not defined.
6.1.25 CMD_LOAD_COLOR_PAL	43
6.1.26 CMD_LOAD_OVLY_PAL	43
6.1.27 CMD_PIN_CHECK.....	43
6.1.28 CMD_FAN_SPEED_OPERATION.....	43
6.1.29 CMD_GET_ADC_VALUES	44
6.1.30 CMD_ENABLE_RETICLE	44
6.1.31 CMD_RETICLE_POSITION.....	44
6.1.32 CMD_ONE_PT_UPDATE.....	44
6.1.33 CMD_WRITE_VID_ENC_REG	45
6.1.34 CMD_PERFORM_TEST	45
6.1.35 CMD_OVERLAY_REFRESH	45
6.1.36 CMD_FREEZE_IMAGE.....	45
6.1.37 CMD_DETECT_BAD_PIXELS	46
6.1.38 CMD_IRCON_LOAD_LUT	46
6.1.39 CMD_LOAD_RAD_PARAMS.....	46
6.1.40 CMD_RESET_PFV_COUNT.....	46
6.1.41 CMD_CLEAR_CONTINUE_FLAG	46
6.1.42 CMD_UNIFORMITY_TEST.....	46
6.1.43 CMD_WRITE_UTILITY_MEMORY	46
6.1.44 CMD_ADV_DETECT_BAD_PIXELS.....	47
6.1.45 CMD_UPLOAD_NUC.....	47
6.1.46 CMD_DOWNLOAD_NUC.....	47
6.1.47 CMD_ENABLE_RANGE_RETICLE	47
6.1.48 CMD_CAMERA_RECOVER	48
6.1.49 CMD_PROG_DSP_DATA_FLASH	48

6.1.50	CMD_UPDATE_EXP_PORT	48
6.1.51	CMD_UPDATE_MODE_INFO	48
6.1.52	CMD_RESTORE_SPI_NVM	49
7	CAMERA ELECTRONICS SIDE INTERFACE	50
7.1	DSP Data Memory	50
7.2	Global Configuration Structure (CAMERA_CONFIG).....	50
7.2.1	CameraConfig.nvmData	51
7.2.2	CameraConfig.updateNVM.....	51
7.2.3	CameraConfig.continueFlag.....	51
7.2.4	CameraConfig.CmdsReceived	51
7.2.5	CameraConfig.camStats.....	51
7.2.6	CameraConfig.camErrors	51
7.2.7	CameraConfig.camTime.....	52
7.2.8	CameraConfig.expPort	52
7.2.9	CameraConfig.swVersion	52
7.2.10	CameraConfig.swBuild	52
7.2.11	CameraConfig.fpgaVersion[2]	52
7.2.12	CameraConfig.fpaSize.....	53
7.2.13	CameraConfig.agcLowIntensity.....	53
7.2.14	CameraConfig.agcHighIntensity.....	53
7.2.15	CameraConfig.actOpName[4]	53
7.2.16	CameraConfig.actNucName[4].....	53
7.2.17	CameraConfig.modeRange	53
7.2.18	CameraConfig.radSWInfo.....	54
7.2.19	CameraConfig.alarm.....	54
7.2.20	CameraConfig.calFlagRefs	54
7.2.21	CameraConfig.fpaLens.....	54
7.2.22	CameraConfig.adcAFiltered[4]	56
7.2.23	CameraConfig.adcBFiltered[4]	56
7.2.24	CameraConfig.btnPanel	56
7.2.25	CameraConfig.miscState.....	57
7.3	Dynamic Configuration Structure (NVM_GLOBAL_CFG).....	57
7.3.1	nvmData.CamMode.....	58
7.3.2	nvmData.FpaMode	58
7.3.3	nvmData.ActMode	58
7.3.4	nvmData.AutoNucData	58
7.3.5	nvmData.AutoRfshTime.....	59
7.3.6	nvmData.AutoRfshTemp	59
7.3.7	nvmData.ActPal	59
7.3.8	nvmData.OvlMode	59
7.3.9	nvmData.RtclXPos.....	59
7.3.10	nvmData.RadMode.....	59
7.3.11	nvmData.AgcMode	59
7.3.12	nvmData.ManualTT	60
7.3.13	nvmData.AgcLimits.....	60
7.3.14	nvmData.LinearMap	60
7.3.15	nvmData.AgcBinLimit	60
7.3.16	nvmData.ActZoneStat.....	Error! Bookmark not defined.
7.3.17	nvmData.VidScaleTemps	60
7.3.18	nvmData.ImageParams	61
7.3.19	CameraConfig.currAtclIndex	57

7.4 Process Code Detection (CAMERA_STATUS)	61
7.5 Progress Code Detection (CAMERA_STATUS)	61
7.6 Error Code Detection (CAMERA_ERRORS)	62
7.6.1 Configuration ID Error.....	63
7.6.2 FPGA Test.....	63
7.6.3 Memory Test.....	64
7.6.4 Force '0' Test.....	64
7.6.5 Force '1' Test.....	65
7.6.6 Force Count Test.....	65
7.6.7 Force Count Coadd Test	65
7.6.8 Histogram Data Grab Test.....	66
7.6.9 NUC Gain and Offset Test.....	66
7.6.10 Video Encoder Test.....	66
7.7 Command Polling	67
7.8 Access to DSP Peripheral Registers (ArchIO)	67
7.9 Access to Xilinx FPGA Registers (FpgalO)	68
7.9.1 FPA Processor Operational Control Register Low	68
7.9.2 FPA Processor Operational Control Register High	68
7.9.3 FPA Processor User Mode Control Register.....	69
7.10 Access to Serial Data Flash	69
8 REMOTE CALIBRATION PROCESS	71
8.1 One Point Refresh Calibration (Internal Flag Only):.....	71
8.2 One Point Update Calibration (Internal Flag Only):.....	71
8.3 Upload NUC Table from Host.....	72
8.4 Download NUC Table to Host	72
APPENDIX A - CAMERA CONFIGURATION DATA STRUCTURES	73
APPENDIX B – FPA PROCESSOR FPGA REGISTER STRUCTURE	83
APPENDIX C - CAMERA COMMAND ENUMERATIONS	85
APPENDIX D - MAPPING OF SERIAL NON-VOLATILE MEMORY	87
APPENDIX E - DYNAMIC MEMORY DEFINITIONS	88
APPENDIX F – NUC COEFFICIENT FORMAT	94

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 Camera Control Interface (CCI) Camera Link Application software communicates with the ECS-320A based camera during system operation. This is done to provide a guideline of how to develop a host application to interact 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 will need to be custom developed incorporating libraries provided by Lumitron and the acquisition board manufacturer. This document encompasses only the serial interface with the camera electronics and not digital data acquisition.

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 v100 b50 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 emulate the Motorola communications library version 1.2. This can be accomplished using the libraries provided by Lumitron. The diagrams below show the two typical methods for connecting to a camera.

The complete list of interface files required at runtime and/or for application development are as follows:

CamSerInt.dll	Dynamic Link Library
CamSerInt.lib	Library File
clallserial.dll	Dynamic Link Library
clallserial.lib	Library File
clallserial.h	Header File
csi_dll_defines.h	Header File
csi_dll_structs.h	Header File
csi_dll_macros.h	Header File
csi_dll_exports.h	Header File

The diagram to the right shows a typical configuration for connecting a camera to a host PC via the serial port interface kit.

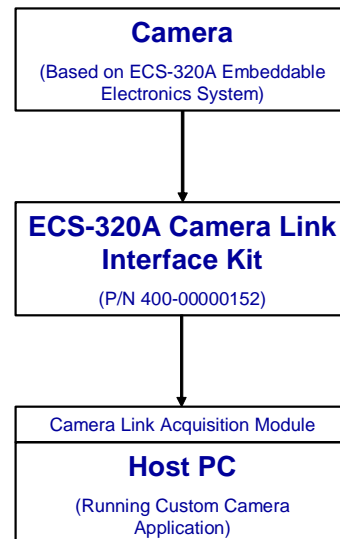
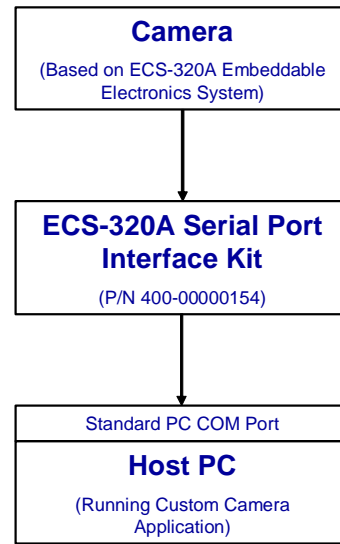
In this configuration the CamSerInt library would provide a serial interface with the camera via a standard PC COM port (currently only COM0 or COM1 are available for use).

In the next diagram, the camera is connected to a Camera Link acquisition module, which is installed in the PC, using the camera link interface kit.

In this configuration the CamSerInt library would make calls into the clAllSerial library, which in turn would enumerate available Camera Link serial ports back to the host application. The calling application would then select an available camera link serial port for communications and all read/write commands would be handed off to the clAllSerial interface library.

Note that one further interface library needs to exist for this configuration to function. This library is a subset of the clAllSerial library and is typically named 'clserxxx.dll' (where xxx is a manufacturer specified value – for example the NI-1426 equivalent is clsernat.dll). This library should be part of the acquisition module installation.

Along with the exposed interface provided by Lumitron 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 (P/N 600-0000026) and portions of it may be listed in this document.



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 operational software 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 progress or accepted packets. Note that the message contains the Lumitron bootloader version, the controller board configuration oscillator rate, and the Motorola S-Record loader version. Note these camera bootloader strings are located in the CamSerInt interface receive buffer. It will be necessary for the calling application to read the terminal mode receive buffer for display purposes.

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.

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 (Motorola Document DSP56F801-7UM/D – DSP56F80X User's Manual), FPA Processor FPGA registers (Appendix B), and global variables (Appendix A and Appendix F). Using Lumitron defined commands the capability expands to allow access to the real time clock, non-volatile RAM (Appendix E), serial data flash (paragraph 7.10), and coefficient data flash (Appendix G) 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 requires only two settings: port and baud rate. The host application will need to configure which communications port it will use and its associated baud rate. It is possible to cycle through the possible baud rates (9600, 38400, 115200) and verifying communication status to auto detect the cameras baud rate. The start/stop bit, parity, and data size settings are automatically configured by the interface library.

5 Host Side Interface

5.1 Interface Information

The exported library functions listed in this paragraph provide a wide-ranging means of controlling the camera electronics. Several of the functions are intended for use only by customers that are experienced with the Lumitron camera product line. Executing commands that write data to NUC coefficient flash, DSP data flash, or portions of the SPI serial flash can put the camera in a state where it cannot “see” or is unbootable. Please contact Lumitron for further information.

5.2 Miscellaneous Serial Port Exports

5.2.1 csi_PortDetect

Description: Routine will make call to 'cIAllSerial' library to check for available camera link module serial ports. Library will also attempt to detect the standard PC COM port(s) and if present will return the enumerated value. Only the first two available PC COM ports will be stored.

Function Declaration: bool CAMSERINTCC csi_PortDetect(void)

Parameters: None

Returns: True if successful, false if unsuccessful.

Example:

```
// Example use of port detect routine
if (csi_PortDetect())
{
    ports = csi_GetNumPorts();

    // Check selected port index is in range
    if (m_SerCfg.pcPort >= ports)
        m_SerCfg.pcPort = 0;
}
else
{
    OutputErrorStatus(_T("Error detecting CameraLink ports.\n"));
}
```

5.2.2 csi_GetNumPorts

Description: Routine to return the number of available serial ports.

Function Declaration: UINT32 CAMSERINTCC csi_GetNumPorts(void)

Parameters: None

Returns: Number of ports

Example:

```
CString          strPort = _T("");
int              i,
                ports = 0;

// Check for number of ports
ports = csi_GetNumPorts();

if (ports > 0)
{
    m_cbxCLPortIndex.ResetContent();
}
```

Serial Interface Developer's Reference Manual

```

csi_PortInfo* pPortInfo = new csi_PortInfo[ports];

for (i = 0; i < ports; i++)
{
    csi_GetPortInfo(i, &pPortInfo[i]);
}

// Try building up manufacturer and port ID into strings
for (i = 0; i < ports; i++)
{
    strPort.Format(
        "%s - %s",
        pPortInfo[i].manfName,
        pPortInfo[i].portID
    );

    m_cbxCLPortIndex.AddString(strPort);
    m_cbxCLPortIndex.SetCurSel(m_nCLPortIndex);
}

SAFE_DELETE_AR(pPortInfo);
}

```

5.2.3 csi_GetPortInfo

Description: Routine to retrieve the port information structure.

Function Declaration: void CAMSERINTCC csi_GetPortInfo(INT32 port, ptr_csi_PortInfo plnfo)

Parameters: port – serial port index; plnfo – pointer to port information structure

Returns: None

Example: See example in paragraph 5.2.2

5.2.4 csi_GetActivePort

Description: Routine to retrieve the active serial port index.

Function Declaration: INT32 CAMSERINTCC csi_GetActivePort(void)

Parameters: None

Returns: Active port index

Example: N/A

5.2.5 csi_OpenPort

Description: Routine to set the desired port index and then attempt to open that port for use.

Function Declaration: bool CAMSERINTCC csi_OpenPort(INT32 desiredPortIndex)

Parameters: desiredPortIndex – user supplied index

Returns: True if successful, false if unsuccessful

Example:

```

// Open the desired port by index
if (csi_OpenPort(m_SerCfg.pcPort))
{
    // Configure the baud rate
    if (csi_ConfigurePort(CL_BAUDRATE_115200))
    {
        m_SerCfg.baudRate = CL_BAUDRATE_115200;
    }
}

```

```

        // Clear out the CamCfg structure
        ZeroMemory((void*)&m_CamCfg, sizeof(cam_Config));

        m_CamCfg.fpgaVersion[0] = 0x5858;
        m_CamCfg.fpgaVersion[1] = 0x5858;

        bResult = true;

        csi_EnterTerminalMode();
    }
}
else
{
    // Error opening port
    OutputGeneralMessage("Could not open COM port.\n");
}
}

```

5.2.6 csi_ConfigurePort

Description: Routine to set the baud rate of the active port.

Function Declaration: bool CAMSERINTCC csi_ConfigurePort(UINT32 baudRate)

Parameters: baudRate – baud rate define (CL_BAUDRATE_115200, CL_BAUDRATE_38400, CL_BAUDRATE_9600)

Returns: True if successful, false if unsuccessful

Example: See example in paragraph 5.2.5

5.2.7 csi_ClosePort

Description: Routine to close active port.

Function Declaration: void CAMSERINTCC csi_ClosePort(void)

Parameters: None

Returns: None

Example: N/A

5.2.8 csi_GetLastError

Description: Routine to retrieve the last recorded error code.

Function Declaration: INT32 CAMSERINTCC csi_GetLastError(void)

Parameters: None

Returns: Error code (see header file for values)

Example:

```

if (!csi_PCMWriteDataMem(&manual, (DWORD)memberAddr, sizeof(nvm_ManualITT)))
{
    theApp.OutputErrorStatus(csi_GetLastError());
}

```

5.2.9 csi_EnterTerminalMode

Description: Routine to enter the terminal mode, this starts the thread (internal to library) that will monitor the port during camera boot/application download before the PC Master protocol is active on the camera end.

Function Declaration: void CAMSERINTCC csi_EnterTerminalMode(void)

Parameters: None

Returns: None

Example: See example in paragraph 5.2.5

5.2.10 csi_PauseTerminalMode

Description: Routine to pauses the terminal mode thread (internal to library), this should be done before attempting to connect to camera via PC Master protocol.

Function Declaration: void CAMSERINTCC csi_PauseTerminalMode(void)

Parameters: None

Returns: None

Example:

```
void CCCIApp::RestartPcmConnection()
{
    // First pause terminal mode thread
    csi_PauseTerminalMode();

    // Sleep a bit
    Sleep(250);

    // Detect PCM by command
    if (!csi_PCMDetect(1))
    {
        // Place (force) back in terminal mode
        StartPortModeTerminal();
    }
    else
    {
        gsTMStatus.appStarted = SET_BIT;
    }
}
```

5.2.11 csi_ResumeTerminalMode

Description: Routine to resume the terminal mode (internal to library), this might be done in situations when a PC Master connection can not be detected, camera has been powered off, or some other protocol issue has occurred.

Function Declaration: void CAMSERINTCC csi_ResumeTerminalMode(void)

Parameters: None

Returns: None

Example:

```
// If supposed to be in PCM mode
if (csi_Connected())
{
    // Detect PCM by command
    if (!csi_PCMDetect(0))
    {
        // Place (force) back in terminal mode
        csi_ResumeTerminalMode ();

        // Went into or already in terminal mode
        ClearInitialConnect();
    }
}
```

```

else
{
    gsTMStatus.appStarted = SET_BIT;

    SetInitialConnect();
}
}

```

5.3 PC Master Abstracted Function Call Exports

The following functions make up the baseline from which the remainder of the exported functions will be built upon. They also closely resemble the Motorola library interface functions that were used in previous versions of the serial interface library.

5.3.1 csi_PCMDetect

Description: Routine to check PCMaster connection status.

Function Declaration: bool csi_PCMDetect(UINT16 nRetry)

Parameters: nRetry – number of attempts to connect

Returns: True if connected via PC Master protocol, false if error or terminal mode

Example: See example in paragraph 5.2.10

5.3.2 csi_PCMSendAppCmd

Description: Routine to send a user defined application command to the camera.

Function Declaration: bool csi_PCMSendAppCmd(BYTE code, UINT32 argSize, LPCVOID argBuff)

Parameters: code – application defined command (see Appendix X); argSize – size of any associated data required by command; argBuffer – pointer to buffer containing command data

Returns: True if successful, false if unsuccessful

Example:

```

bool        bSuccess;
UWord16     outVal;

if (m_bFreeze)
{
    outVal = 1;
}
else
{
    outVal = 0;
}

if (csi_Connected())
{
    // (Un)Freeze current image
    bSuccess = csi_PCMSendAppCmd(
        (BYTE)CMD_FREEZE_IMAGE,
        sizeof(UWord16),
        &outVal
    );

    // If failed we need to break
    if (!bSuccess)
    {
        theApp.OutputErrorStatus(csi_GetLastError());
    }
}

```

```

}
else
{
    theApp.OutputConnectionStatus();
}

```

5.3.3 csi_PCMReadDataMem

Description: Routine to read a block of memory from the DSP and store it in the user supplied buffer. This routine can be used to read from any valid DSP read/write memory mapped address.

Function Declaration: bool csi_PCMReadDataMem(LPVOID dest, UINT32 addr, UINT16 size)

Parameters: dest – pointer to caller's destination buffer; addr – address in DSP memory space; size of data in bytes to read

Returns: True if successful, false if unsuccessful

Example:

```

void*      pData;
UWord16*  pWords;
bool      bSuccess;
CString   tStr = _T("");

if (!csi_Connected())
{
    theApp.OutputConnectionStatus();
    return;
}

pData = new BYTE[32];

pWords = (UWord16*)pData;

ZeroMemory(pData, 32);

// Send retrieve ADC data command
bSuccess = csi_PCMSendAppCmd(
    (BYTE)CMD_GET_ADC_VALUES,
    0,
    NULL
);

// If failed we need to break
if (!bSuccess)
{
    theApp.OutputErrorStatus(csi_GetLastError());
}
else
{
    // Wait until DSP is ready for next command
    csi_GetPCMStatus();

    // Read scratch pad data
    bSuccess = csi_PCMReadDataMem(
        pData,
        (UWord32)BASE_ADDR_SCRATCH_AREA,
        sizeof(BYTE) * 32
    );

    if (!bSuccess)
    {
        theApp.OutputErrorStatus(csi_GetLastError());
    }
    else
    {
        m_lbxReadout.ResetContent();
    }
}

```

Serial Interface Developer's Reference Manual

```

        tStr.Format(_T("ADC Readback Values:"));
        m_lbxReadout.AddString(tStr);

        tStr.Format(_T("AdcaAn0FD:\t%d"), *pWords++ >> 3);
        m_lbxReadout.AddString(tStr);

        ...

        tStr.Format(_T("AdcbAn7FD:\t%d"), *pWords >> 3);
        m_lbxReadout.AddString(tStr);
    }
}

```

5.3.4 csi_PCMWriteDataMem

Description: Routine to write a block of memory to the DSP (data memory) from the supplied buffer. This routine can be used to write to any valid DSP read/write memory mapped address.

Function Declaration: bool csi_PCMWriteDataMem(LPVOID src, UINT32 addr, UINT16 size)

Parameters: src – pointer to caller supplied buffer; addr – address in DSP memory space; size of data in bytes to read

Returns: True if successful, false if unsuccessful

Example:

```

nvm_AgcLimits    limits;
DWORD           memberAddr = theApp.GetCfgDataCameraAddr(&pNvm->AgcLimits);

ZeroMemory(&limits, sizeof(limits));

// If connected then output data to camera
if (csi_Connected())
{
    limits.AgcLowLimit = (UWord16)m_nLowLimit;

    limits.AgcHighLimit = (UWord16)m_nHighLimit;

    // Send data to camera
    if (!csi_PCMWriteDataMem(&limits, (DWORD)memberAddr, sizeof(nvm_AgcLimits)))
    {
        theApp.OutputErrorStatus(csi_GetLastError());
    }
    else
    {
        csi_ForceNVMUpdate();
    }
}

```

5.3.5 csi_PCMWriteDataMemMask

Description: Routine to write a block of memory to the DSP (data memory) from the supplied data buffer and supplied mask buffer. This routine can be used to write to any valid DSP read/write memory mapped address.

Function Declaration: bool csi_PCMWriteDataMemMask(LPVOID src, LPVOID mask, UINT32 addr, UINT16 size)

Parameters: src – pointer to caller supplied buffer; mask – pointer to caller supplied mask buffer; addr – address in DSP memory space; size of data in bytes to read

Returns: True if successful, false if unsuccessful

Example:

Serial Interface Developer's Reference Manual

```

bool          bSuccess;
UWord16      outVal;
UWord16      mask = BIT_7;
DWORD        memberAddr = theApp.GetCfgDataCameraAddr (&pNvm->FpaMode);

if (csi_Connected())
{
    UpdateData();

    // Set output value (same for both register and structure member)
    if (m_bColorBar)
        outVal = BIT_7;
    else
        outVal = 0;

    // Output value to register
    bSuccess = csi_PCMWriteDataMemMask(
        &outVal,
        &mask,
        (UWord32)FpgaAddr_UserCtrlReg,
        sizeof(outVal)
    );

    if (!bSuccess)
    {
        theApp.OutputErrorStatus(csi_GetLastError());
    }

    // Update the global NVM data structure
    bSuccess = csi_PCMWriteDataMemMask(
        &outVal,
        &mask,
        (DWORD)memberAddr,
        sizeof(outVal)
    );

    if (!bSuccess)
    {
        theApp.OutputErrorStatus(csi_GetLastError());
    }
    else
    {
        csi_ForceNVMUpdate();
    }
}
else
{
    theApp.OutputConnectionStatus();
}

```

5.4 S-Record Pointer and Size Exports

5.4.1 csi_SetSRecordPtr

Description: Sets the pointer to the base of the Motorola S-Record file in the interface library for uploading camera application software.

Function Declaration: void CAMSERINTCC csi_SetSRecordPtr(LPBYTE pBuf)

Parameters: pBuf – pointer to S-Record buffer

Returns: None

Example:

```

CFile*        pFile = new CFile;
CFileException e;

```

Serial Interface Developer's Reference Manual

```

BOOL          ret;
CString      tStr = _T("");
CString      filePath = _T("");
UWord32      bytes = 0;
UWord32      nSize;

if (m_bForceUpdate)
{
    // Now see if we can open and store the file
    // Get file path from control
    filePath = m_utedCameraApp.GetPath();

    // Open the file
    ret = pFile->Open(filePath, CFile::modeRead, &e);

    // If file open
    if (ret)
    {
        // Get the file size
        nSize = (DWORD)pFile->GetLength();

        // Now try to create the buffer
        m_pSRecData = new BYTE[nSize];

        // If we are successful read the file
        if (m_pSRecData != NULL)
        {
            bytes = pFile->Read(m_pSRecData, nSize);

            // If we got all the data
            if (bytes == nSize)
            {
                csi_SetSRecordPtr((LPBYTE)m_pSRecData);
                csi_SetSRecordSize(nSize);
            }
            else
            {
                // Inform user of file open error
                tStr.Format(_T("Error Reading File: %s"), filePath);
                theApp.DoMessageBox(tStr, MB_OK, 0);

                // Zero out terms
                csi_ResetSRecordInfo();
            }
        }
        else
        {
            // Inform user of file open error
            tStr.Format(_T("Error allocating file buffer!"));
            theApp.DoMessageBox(tStr, MB_OK, 0);
        }

        // Close the file
        pFile->Close();
    }
    else
    {
        // Inform user of file open error
        tStr.Format(IDS_FILE_ERROR_OPEN, filePath);
        theApp.DoMessageBox(tStr, MB_OK, 0);
    }
}
else
{
    // Zero out terms
    csi_ResetSRecordInfo();
}

```

5.4.2 csi_SetSRecordSize

Description: Routine to set the size of the Motorola S-Record file (in bytes) in the interface library.

Function Declaration: void CAMSERINTCC csi_SetSRecordSize(UINT32 size)

Parameters: size – size of file in bytes

Returns: None

Example: See example in paragraph 5.4.1

5.4.3 csi_ResetSRecordInfo

Description: Clears reference to the S-Record file in the interface library.

Function Declaration: void CAMSERINTCC csi_ResetSRecordInfo(void)

Parameters: None

Returns: None

Example: See example in 5.4.1

5.4.4 csi_SetTerminalEvent

Description: Routine for the calling application to provide an event handle so that it may be notified of data in the terminal mode buffer. This is useful if the calling application wants to display the initial bootloader output strings (see paragraph 3.2).

Function Declaration: void CAMSERINTCC csi_SetTerminalEvent(HANDLE hEvent)

Parameters: hEvent – handle to the caller created event handle for the terminal mode thread

Returns: None

Example:

```

UWord32          threadID;
CString          eventName;

// Create unique event name
eventName.Format(_T("TerminalModeThread"));

m_hTerminalEvent = CreateEvent(
    NULL,
    FALSE,          /* auto-reset */
    FALSE,          /* init. state */
    eventName.operator LPCTSTR()
);

// Local Terminal Thread (for bootloader & upload display)
m_hTerminalThread = CreateThread(
    NULL,
    0x1000,
    ::TerminalModeThread,
    this,
    NULL,
    &threadID
);

if (m_hTerminalThread)
{
    TRACE0("Terminal Mode Thread Created\n");
}
else
{

```

```

        CloseHandle(m_hTerminalEvent);
        m_hTerminalEvent = NULL;
    }

    csi_SetTerminalEvent(m_hTerminalEvent);

```

5.5 Camera Status and Status Code Exports

5.5.1 csi_Connected

Description: Routine to check if the port is in PC Master mode.

Function Declaration: bool CAMSERINTCC csi_Connected(void)

Parameters: None

Returns: True if connected in PC Master mode, false if in terminal mode or not connected

Example:

```

// If connected
if (csi_Connected())
{
    // Refresh the overlay
    if (!csi_PCMSendAppCmd((BYTE)CMD_OVERLAY_REFRESH, 0, NULL))
    {
        theApp.OutputErrorStatus(csi_GetLastError());
    }
}
else
{
    theApp.OutputConnectionStatus();
}

```

5.5.2 csi_GetPCMStatus

Description: Routine to delay until the camera is ready for another application command.

Function Declaration: BYTE CAMSERINTCC csi_GetPCMStatus(void)

Parameters: None

Returns: The enumerated command status response value (see header file)

Example:

```

// Wait until DSP is ready for next command
csi_GetPCMStatus();

// Send 1 pt update command
bSuccess = csi_PCMSendAppCmd(
    (BYTE)CMD_ONE_PT_UPDATE,
    sizeof(outVal),
    &outVal
);

```

5.5.3 csi_GetProcessCode

Description: Routine to retrieve the camera's current process code.

Function Declaration: UINT16 CAMSERINTCC csi_GetProcessCode(void)

Parameters: None

Returns: Current enumerated process code (see header file)

Example:

```

// Wait here until stable
while (procCode == PRC_TEC_STABILIZING)
{
    Sleep(100);

    procCode = csi_GetProcessCode();

    if (!flag)
    {
        theApp.OutputProcessStatus(procCode);
        flag = true;
    }
}

```

5.5.4 csi_GetProgressCode

Description: Routine to retrieve the camera's current progress code. Typically used for debugging boot progress.

Function Declaration: UINT16 CAMSERINTCC csi_GetProgressCode(void)

Parameters: None

Returns: Camera enumerated progress code (see header file)

Example:

```

// See if board is not configured
nCode = csi_GetProgressCode();

if (nCode == UNCONFIGURED_CONTROLLER)
    return;

// Get values from register settings
if (csi_Connected())
{
    // Loop until camera is far enough along
    tick = GetTickCount();

    while (FOREVER)
    {
        // Check for timeout
        if (GetTickCount() - tick > 125000)
            break;

        nCode = csi_GetProgressCode();

        if ((nCode == CAMERA_READY) && (csi_GetCalStatusCode() == HOST_READY))
        {
            break;
        }

        // Sleep a bit
        Sleep(10);
    }

    ...
}

```

5.5.5 csi_GetCamErrorCode

Description: Routine to retrieve the camera's error code data structure.

Function Declaration: void CAMSERINTCC csi_GetCamErrorCode(ptr_cam_Errors pErr)

Parameters: pErr – pointer to the user supplied destination buffer

Returns: None

Example:

```

// We need to get the error data
csi_GetCamErrorCode(&err);

...

// Update specific error
switch (err.ErrorCode)
{
default:
    // Format and output Specific Error string
    strErrSpecific.AppendFormat(IDS_NOT_APPLICABLE);
    break;

case (ERR_VID_ENCODER):
case (ERR_FORCE_ZERO):
case (ERR_FORCE_ONE):
case (ERR_FORCE_COUNT):
case (ERR_HISTO_GRAB):
case (ERR_GAIN_OFFSET):
case (ERR_FORCE_COUNT_COADD):
    // Set test fail flag
    SetPcmBootTestFail();
    // Format and output Specific Error string
    strErrSpecific.AppendFormat(IDS_NOT_APPLICABLE);
    break;

case (ERR_FPGA_LOAD):
    // Format and output Specific Error string
    strErrSpecific += _T("FPGA Done Bit not Set");
    break;

case (ERR_FPGA_TEST):
    // Set test fail flag
    SetPcmBootTestFail();
    // Format and output Specific Error string
    if (err.ErrorSubCode == ERR_WALKING_0_1)
    {
        strErrSpecific += _T("FPGA Walking 1's and 0's Test");
    }
    else if (err.ErrorSubCode == ERR_CROSSTALK)
    {
        strErrSpecific += _T("FPGA Crosstalk Test");
    }
    else if (err.ErrorSubCode == ERR_FPGA_MEMORY)
    {
        strErrSpecific += _T("FPGA Memory Test");
    }
    else
    {
        strErrSpecific += _T("Undetermined");
    }
    break;

...
}

```

5.5.6 csi_GetCalStatusCode

Description: Routine to retrieve the calibration status code. Allows host to check if camera is performing an internal calibration or is ready for a new process. This function should be replaced with

the 'csi_PCMReadDataMem' function to read the host status values directly. This method is more efficient.

Function Declaration: UINT16 CAMSERINTCC csi_GetCalStatusCode(void)

Parameters: None

Returns: Current calibration status code

Example: N/A

5.6 Serial Flash Data Access Exports

5.6.1 csi_ReadSerialFlashPage

Description: Routine to read a page of serial flash data. The routine first gets the data from flash and places it in a set location on the DSP data memory block (scratch pad buffer @ 0x00C0). Then it reads the data from there and puts it in the user supplied buffer. Typical uses would be to read the camera lens descriptor table, operational mode data structure or NUC mode structure.

Function Declaration: bool CAMSERINTCC csi_ReadSerialFlashPage(void* dataBuf, UINT16 page)

Parameters: dataBuf – pointer to user supplied buffer; page – page number in serial data flash

Returns: True if data is read, false if an error occurred

Example:

```
pData = new BYTE[264];

pBytes = (BYTE*)pData;

pWords = (UWord16*)pData;

// Make sure data is up to date
UpdateData();

ZeroMemory(pBytes, 264);

status = csi_ReadSerialFlashPage(
    pData,
    (UWord16)m_nSpiFlashPage
);

// Check the status from the camera
if (!status)
{
    theApp.OutputErrorStatus(_T("Error Reading Page\n"));
}
else
{
    m_lbxReadout.ResetContent();

    tStr.Format(_T("SPI Flash Page: %4d"), m_nSpiFlashPage);
    m_lbxReadout.AddString(tStr);

    if (m_bByteFormat)
    {
        for (x = 0; x < 264; x += 8, pBytes += 8)
        {
            tStr.Format(
                _T("0x%08X: %02X %02X %02X %02X %02X %02X %02X %02X"),
                x + (264 * m_nSpiFlashPage),
                pBytes[0],
                pBytes[1],
                pBytes[2],
                pBytes[3],
                pBytes[4],
                pBytes[5],
                pBytes[6],
                pBytes[7]
            );
        }
    }
}
```

Serial Interface Developer's Reference Manual

```

        pBytes[3],
        pBytes[4],
        pBytes[5],
        pBytes[6],
        pBytes[7]
    );

    m_lbxReadout.AddString(tStr);
}
else
{
    for (x = 0; x < 132; x += 4, pWords += 4)
    {
        tStr.Format(
            _T("0x%08X:\t%04X\t%04X\t%04X\t%04X"),
            (x << 1) + (264 * m_nSpiFlashPage),
            pWords[0],
            pWords[1],
            pWords[2],
            pWords[3]
        );

        m_lbxReadout.AddString(tStr);
    }
}
}

```

5.6.2 csi_ReadSerialFlashBlock

Description: Routine to read a block (8 pages per block) of serial flash data. The routine first gets the data from flash and places it in a set location on the DSP data memory block (scratch pad buffer @ 0x00C0). Then it reads the data from there and puts it in the user supplied buffer. The user is encouraged to use the 'csi_ReadSerialFlashPage' routine in a loop to read consecutive pages.

Function Declaration: bool CAMSERINTCC csi_ReadSerialFlashBlock(void* pDataBuf, UINT16 page)

Parameters: dataBuf – pointer to user supplied buffer; page – base page number in serial data flash (see header file for flash page listing)

Returns: True if data is read, false if an error occurred

Example: N/A

5.6.3 csi_WriteSerialFlashPage

Description: Routine to write a page of serial flash data. The routine first places the data into a set location on the DSP data memory block (scratch pad buffer @ 0x00C0). From there a command is sent to copy the data from the scratch pad buffer to a serial flash page. The command may not be accepted if the memory lock setting on the camera has not been cleared (depends on flash address).

Function Declaration: bool CAMSERINTCC csi_WriteSerialFlashPage(void* pDataBuf, ptr_pcm_FlashXfer pXferInfo)

Parameters: pDataBuf – pointer to user supplied buffer of data to be copied to serial flash; pXferInfo – pointer to transfer structure that contains data destination information (see structure header file)

Returns: True if data is written, false if an error occurred

Example:

```

BYTE testBuffer[264];
int x;

```

Serial Interface Developer's Reference Manual

```

if (csi_Connected())
{
    for (x = 0; x < PAGE_SIZE; x++)
        testBuffer[x] = x;

    // Setup XFER data structure
    xferdata.eraseFlag = BYPASS_CHECK;
    xferdata.offset = 0;
    xferdata.page = COMP_IMAGE_BASE_PAGE;
    xferdata.size = PAGE_SIZE;

    // Write a page of data to the camera
    status = csi_WriteSerialFlashPage(
        (void*)&testBuffer,
        &xferdata
    );

    // Check the status from the camera
    if (!status)
    {
        theApp.OutputErrorStatus(_T("Error Writing to SPI Flash\n"));
    }
}
else
{
    theApp.OutputConnectionStatus();
}

```

5.6.4 csi_WritePartialSerialFlashPage

Description: Routine to write a portion of a full page of serial flash data. The routine first places the data into a set location on the DSP data memory block (scratch pad buffer @ 0x00C0). From there a command is sent to copy the data from the scratch pad buffer to a serial flash page.

Function Declaration: bool CAMSERINTCC csi_WritePartialSerialFlashPage(void* pDataBuf, ptr_pcm_FlashXfer pXferInfo)

Parameters: pDataBuf – pointer to user supplied buffer of data to be copied to serial flash; pXferInfo – pointer to transfer structure that contains data destination information (see structure header file)

Returns: True if data is written, false if an error occurred

Example:

```

xferData.eraseFlag = 0;
xferData.offset = OFFSET_OPMODE_ROI;
xferData.page = OP_MODE_TABLE_BASE_PAGE + m_nOpMode;
xferData.size = sizeof(cam_RoiMode) * 8;

// Write to the page that contains the radiometric data
status = csi_WritePartialSerialFlashPage(
    &m_LocalData[m_nOpMode].modeRoi[0],
    &xferData
);

// If we get the data then check fo filename
if (!status)
{
    tStr.Format(_T("Error Writing ROI Parameters"));
    theApp.DoMessageBox(tStr, MB_OK, 0);
}

```

5.7 NUC Flash Data Access Exports

5.7.1 csi_ReadNucFlashMemory

Description: Routine to read a block of NUC flash data which is accessed through the camera FPGA. Uses the DSP scratch pad area for intermediate storage.

Function Declaration: bool CAMSERINTCC csi_ReadNucFlashMemory(void* dataBuf, ptr_pcm_MemXfer xferInfo)

Parameters: dataBuf – pointer to user's destination buffer; xferInfo – pointer to transfer structure that contains data source information (see structure header file)

Returns: True if data is read, false if an error occurred

Example:

```
pData = new UWord16[PC_MSTR_XFER_BUF_SIZE];

pWords = (UWord16*)pData;

// Setup Transfer Data Structure
xferInfo.addr = m_dwFlashAddr;
xferInfo.size = PC_MSTR_XFER_BUF_SIZE;

// Now go and get data for debug viewing
status = csi_ReadNucFlashMemory(pData, &xferInfo);

if (!status)
{
    theApp.OutputErrorStatus(_T("Error Reading NUC Flash Memory\n"));
}
else
{
    m_lbxReadout.ResetContent();

    tStr.Format(_T("NUC Flash Base Address: 0x%08X:"), m_dwFlashAddr);
    m_lbxReadout.AddString(tStr);

    for (y = 0, x = 0; x < PC_MSTR_XFER_BUF_SIZE; x += 4, y += 4, pWords += 4)
    {
        tStr.Format(
            _T("0x%04X:\t%04X\t%04X\t%04X"),
            y,
            pWords[0],
            pWords[1],
            pWords[2],
            pWords[3]
        );

        m_lbxReadout.AddString(tStr);
    }
}
```

5.8 Camera Product ID Access Exports

5.8.1 csi_ReadProductID

Description: Routine to read product ID structure data from the data ('X') memory area on the camera DSP. Uses the DSP scratch pad area for intermediate storage.

Function Declaration: bool CAMSERINTCC csi_ReadProductID(ptr_cam_ProdID pID)

Parameters: pID – pointer to user supplied destination structure buffer

Returns: True if data is read, false if an error occurred

Example:

```

cam_ProdID  id;
bool        ret;

// Product Serial Number Data is located at 0x2000 offset 0x0000
// in the embedded computer data memory

ret = csi_ReadProductID(&id);

if (ret)
{
    CCCIDoc*   pDoc = theApp.GetActiveDocument();

    // Copy original file data back to local values
    if (pDoc)
    {
        memcpy(&pDoc->m_ProdID, &id, sizeof(cam_ProdID));
    }

    // Set controls with data retrieved from camera
    SetDialogControlData(&id);
}

```

5.8.2 csi_WriteProductID

Description: Routine to write product ID structure data to the data ('X') memory area on the camera DSP. Uses the DSP scratch pad area for intermediate storage. The command may not be accepted if the memory lock setting on the camera has not been cleared. Since the operational software uses information in the product ID settings, changes may alter the camera useability.

Function Declaration: bool CAMSERINTCC csi_WriteProductID(ptr_cam_ProdID pID)

Parameters: pID – pointer to user supplied source structure buffer

Returns: True if data is written, false if error occurred

Example: N/A

5.9 Camera Static Configuration Access Exports

5.9.1 csi_ReadStaticCfg

Description: Routine to read static configuration data from the data ('X') memory area on the camera DSP. Uses the DSP scratch pad area for intermediate storage.

Function Declaration: bool CAMSERINTCC csi_ReadStaticCfg(ptr_cam_StaticCfg pCfg)

Parameters: pCfg – pointer to user's supplied destination structure buffer

Returns: True if data is read, false if error occurred

Example:

```

cam_StaticCfg  statCfg;

if (csi_ReadStaticCfg(&statCfg))
{
    if (statCfg.dspVideo & BIT_15)
        m_Monochrome = true;
    else
        m_Monochrome = false;
}

```

5.9.2 csi_WriteStaticCfg

Description: Routine to write static configuration data to the data ('X') memory area on the camera DSP. Uses the DSP scratch pad area for intermediate storage.

Function Declaration: bool CAMSERINTCC csi_WriteStaticCfg(ptr_cam_StaticCfg pCfg)

Parameters: pCfg – pointer to user's supplied source structure buffer

Returns: True if data is written, false if error occurred

Example: N/A

5.10 Camera Utility Memory Access Exports

5.10.1 csi_ReadUtilityMemory

Description: Routine to read a block of utility memory which is accessed through the camera FPGA. Uses the DSP scratch pad area for intermediate storage.

Function Declaration: bool CAMSERINTCC csi_ReadUtilityMemory(void* dataBuf, ptr_pcm_MemXfer xferInfo)

Parameters: dataBuf – pointer to user's destination buffer; xferInfo – pointer to transfer structure that contains data source information (see structure header file)

Returns: True if data is read, false if error occurred

Example:

```
pData = new UWord16[PC_MSTR_XFER_BUF_SIZE];

pWords = (UWord16*)pData;

// First Send Image Grab Command (default is buffer B)
csi_GrabSingleFrameImage();

// Setup Transfer Data Structure
xferInfo.addr = MAR_IMAGE_GRAB_B;
xferInfo.size = PC_MSTR_XFER_BUF_SIZE;

for (y = 0, j = 0;
     j < LOOPS_FOR_IMAGE_GRAB;
     j++, xferInfo.addr += PC_MSTR_XFER_BUF_SIZE)
{
    // Now go and get 1st four lines of image for debug viewing
    status = csi_ReadUtilityMemory(pData, &xferInfo);

    if (!status)
    {
        theApp.OutputErrorStatus(_T("Error Reading Utility Memory\n"));
        break;
    }
    else
    {
        if (j == 0)
        {
            m_lbxReadout.ResetContent();

            tStr.Format(
                _T("Image Grab: First %d Pixels"),
                PC_MSTR_XFER_BUF_SIZE * LOOPS_FOR_IMAGE_GRAB
            );
            m_lbxReadout.AddString(tStr);
        }
    }
}
```

Serial Interface Developer's Reference Manual

```

pWords = (UWord16*)pData;
for (x = 0; x < PC_MSTR_XFER_BUF_SIZE; x += 4, y += 4, pWords += 4)
{
    tStr.Format(
        _T("0x%04X:\t%04X\t%04X\t%04X\t%04X"),
        y,
        pWords[0],
        pWords[1],
        pWords[2],
        pWords[3]
    );

    m_lbxReadout.AddString(tStr);
}
}
}

```

5.10.2 csi_WriteUtilityMemory

Description: Routine to write a block of data to utility memory which is accessed through the camera FPGA. Uses the DSP scratch pad area for intermediate storage. The command may not be accepted if the memory lock setting on the camera has not been cleared. Since the operational software uses information in the static configuration settings, changes may alter the camera useability

Function Declaration: bool CAMSERINTCC csi_WriteUtilityMemory(void* pDataBuf, ptr_pcm_MemXfer xferInfo)

Parameters: dataBuf – pointer to user's source buffer; xferInfo – pointer to transfer structure that contains data destination information (see structure header file)

Returns: True if data is read, false if error occurred

Example:

```

// Loop on all pixel samples
while (count > 0)
{
    // See how much we can send
    if (count >= PC_MSTR_XFER_BUF_SIZE)
        xferMem.size = PC_MSTR_XFER_BUF_SIZE;
    else
        xferMem.size = (UWord16)count;

    // Loop to build up the transfer buffer values
    for (c = 0; c < xferMem.size; c++)
    {
        pBase[c] = *pBuf;

        // Increment term pointer
        pBuf++;
    }

    // Write the data to utility memory
    bCmdStatus = csi_WriteUtilityMemory((void*)pBase, &xferMem);

    // Increment address and decrement count
    xferMem.addr += PC_MSTR_XFER_BUF_SIZE;
    count -= xferMem.size;

    // Verify command status
    if (bCmdStatus)
    {
        // Update on screen progress
        tStrA.Format(tStrB, 100.f * (1.0f - (float)count/(float)m_nFpaPixelCount));
        m_stcLoadStatus.SetWindowText(tStrA);
    }
    else

```

```

    {
        // We have a failure
        tStrA = _T("Error Writing to Utility Memory, Process Terminated");
        theApp.DoMessageBox(tStrA, MB_OK, 0);
        bContinue = false;
        break;
    }
}

```

5.11 Camera Time Exports

5.11.1 csi_ReadCameraTime

Description: Routine to read time from the camera real time clock chip. Uses the DSP scratch pad area for intermediate storage.

Function Declaration: bool CAMSERINTCC csi_ReadCameraTime(ptr_cam_RtcData pTime)

Parameters: pTime – pointer to user's destination structure buffer

Returns: True if data is read, false if error occurred

Example:

```

cam_RtcData timeData = {0};

if (csi_ReadCameraTime(&timeData))
{
    /* CTime( int nYear, int nMonth, int nDay, int nHour,
             int nMin, int nSec, int nDST = -1 ); */
    CTime tTime(
        timeData.year + 2000,
        timeData.month,
        timeData.date,
        timeData.hours,
        timeData.minutes,
        timeData.seconds
    );

    m_dtTime = tTime;
    m_dtDate = tTime;

    UpdateData(FALSE);
}

```

5.11.2 csi_WriteCameraTime

Description: Routine to write time to the camera real time clock chip. Uses the DSP scratch pad area for intermediate storage.

Function Declaration: bool CAMSERINTCC csi_WriteCameraTime(ptr_cam_RtcData pTime)

Parameters: pTime – pointer to user's source structure buffer

Returns: True if data is written, false if error occurred

Example:

```

cam_RtcData timeData;
CTime      m_dtTime;

UpdateData();

timeData.hours = m_dtTime.GetHour();
timeData.minutes = m_dtTime.GetMinute();
timeData.seconds = m_dtTime.GetSecond();

```

```

timeData.year = m_dtDate.GetYear() - 2000;
timeData.month = m_dtDate.GetMonth();
timeData.date = m_dtDate.GetDay();
timeData.day = m_dtDate.GetDayOfWeek();

csi_WriteCameraTime(&timeData);

```

5.12 Camera NVM Memory Access Exports

5.12.1 csi_ReadNVMBlock

Description: Routine to read a block of non-volatile RAM from the real time clock chip. Uses the DSP scratch pad area for intermediate storage.

Function Declaration: bool CAMSERINTCC csi_ReadNVMBlock(void* pDataBuf, ptr_pcm_NvmXfer pXferInfo)

Parameters: pDataBuf – pointer to user's destination buffer; pXferInfo – pointer to transfer structure that contains data source information (see structure header file)

Returns: True if data is read, false if error occurred

Example:

```

xferData.offset = 0;
xferData.size = 96;

// Total number of available bytes
pData = new BYTE[96];

pBytes = (BYTE*)pData;

pWords = (UWord16*)pData;

ZeroMemory(pBytes, 96);

status = csi_ReadNVMBlock((void*)pData, &xferData);

// Check the status from the camera
if (status)
{
    m_lbxReadout.ResetContent();

    tStr.Format(_T("NVM Data Block:"));
    m_lbxReadout.AddString(tStr);

    if (m_bByteFormat)
    {
        for (x = 0; x < 96; x += 8, pBytes += 8)
        {
            tStr.Format(
                _T("0x%04X:\t%02X %02X %02X %02X %02X %02X %02X %02X"),
                x + 32,
                pBytes[0],
                pBytes[1],
                pBytes[2],
                pBytes[3],
                pBytes[4],
                pBytes[5],
                pBytes[6],
                pBytes[7]
            );

            m_lbxReadout.AddString(tStr);
        }
    }
}

```

```

    }
    else
    {
        for (x = 0; x < 48; x += 4, pWords += 4)
        {
            tStr.Format(
                _T("0x%04X:\t%04X\t%04X\t%04X\t%04X"),
                (x << 1) + 32,
                pWords[0],
                pWords[1],
                pWords[2],
                pWords[3]
            );

            m_lbxReadout.AddString(tStr);
        }
    }
}

```

5.12.2 csi_WriteNVMBlock

Description: Routine to write a block of non-volatile RAM to the real time clock chip. Uses the DSP scratch pad area for intermediate storage.

Function Declaration: bool CAMSERINTCC csi_WriteNVMBlock(void* pDataBuf, ptr_pcm_NvmXfer pXferInfo)

Parameters: pDataBuf – pointer to user's source buffer; pXferInfo – pointer to transfer structure that contains data destination information (see structure header file)

Returns: True if data is read, false if error occurred

Example:

```

pcm_NvmXfer xferData;
int x;
BYTE* pBytes;

xferData.offset = 0;
xferData.size = NUM_TEST_BYTES;

pBytes = new BYTE[NUM_TEST_BYTES];

if (pBytes)
{
    for (x = 0; x < NUM_TEST_BYTES; x++)
        pBytes[x] = (BYTE)(NUM_TEST_BYTES - x);

    csi_WriteNVMBlock((void*)pBytes, &xferData);
}

```

5.13 Camera Scaled Emittance Table Exports

5.13.1 csi_WriteAtcLUT

Description: Routine to write the Ambient Temperature Compensation (ATC) look-up table (used for adjusting the ATC NUC) to SPI flash. Uses the DSP scratch pad area for intermediate storage. This command should only be used by customers that completely understand the calibration process and the implications of modifying this table.

Function Declaration: bool CAMSERINTCC csi_WriteAtcLUT(UINT16* pTable, INT16 nActNucMode)

Parameters: pTable – pointer to the user supplied ATC look up table source buffer; nActNucMode – linear index value of the NUC table to be written

Returns: True if data is written, false if error occurred

Example: N/A

5.13.2 csi_ReadAtcLUT

Description: Routine to read the Ambient Temperature Compensation (ATC) look-up table (used for adjusting the ATC NUC) from SPI flash. Uses the DSP scratch pad area for intermediate storage.

Function Declaration: bool CAMSERINTCC csi_ReadAtcLUT(UINT16* pTable, INT16 nActNucMode)

Parameters: pTable – pointer to the user supplied ATC look up table destination buffer; nActNucMode – linear index value of the NUC table to be read

Returns: True if data is read, false if error occurred

Example: N/A

5.14 Camera DAC 0 Table Exports

5.14.1 csi_ReadDac0LUT

Description: Routine to read the DAC 0 look-up table (used for adjusting the ATC NUC) from SPI flash. Uses the DSP scratch pad area for intermediate storage.

Function Declaration: bool CAMSERINTCC csi_ReadDac0LUT(BYTE* pLUT, UINT16 nMode)

Parameters: pLUT – pointer to user supplied destination buffer; nMode – linear index of NUC mode

Returns: True if data is read, false if error occurred

Example: N/A

5.14.2 csi_WriteDac0LUT

Description: Routine to read the DAC 0 look-up table (used for adjusting the ATC NUC) from SPI flash. Uses the DSP scratch pad area for intermediate storage. This command should only be used by customers that completely understand the calibration process and the implications of modifying this table.

Function Declaration: bool CAMSERINTCC csi_WriteDac0LUT(BYTE* pLUT, UINT16 nMode)

Parameters: pLUT – pointer to user supplied source buffer; nMode – linear index of NUC mode

Returns: True if data is read, false if error occurred

Example: N/A

5.15 Camera Focus Exports

5.15.1 csi_FocusFar

Description: Routine executes a focus far command.

Function Declaration: void CAMSERINTCC csi_FocusFar(void)

Parameters: None

Returns: None

Example: N/A

5.15.2 csi_FocusNear

Description: Routine executes a focus near command.

Function Declaration: void CAMSERINTCC csi_FocusNear(void)

Parameters: None

Returns: None

Example: N/A

5.16 Miscellaneous Camera Command Exports

5.16.1 csi_ForceNVMUpdate

Description: Routine to set a flag on the camera that when polled will instruct the software to save the current global NVM data structure (mirrored) to the real time clock chip NVM. Typically a host application will change an operational parameter for the camera (such as the video palette index) via the global data structure (see paragraph 7.3) – to make sure that the setting is preserved on subsequent boots it is necessary to force the camera to store the value with this command.

Function Declaration: void CAMSERINTCC csi_ForceNVMUpdate(void)

Parameters: None

Returns: None

Example:

```

if (csi_Connected())
{
    BeginWaitCursor();

    UpdateData();

    m_nVideoPal = m_spnVideoPal.GetPos();
    palSel = (UWord16)m_nVideoPal;

    // Load video palette at supplied index
    bSuccess = csi_PCMSendAppCmd(
        (BYTE)CMD_LOAD_COLOR_PAL,
        sizeof(UWord16),
        &palSel
    );

    /* Display message if failed */
    if (!bSuccess)
    {
        theApp.OutputErrorStatus(csi_GetLastError());
    }
    else
    {
        csi_ForceNVMUpdate();
        UpdateVideoPaletteName();
    }

    EndWaitCursor();
}

```

5.16.2 csi_GrabSingleFrameImage

Description: Routine to initiate a frame grab and store image into the selected camera buffer. It can then be read using the 'csi_ReadUtilityMemory' routine. This routine is typically used only for debugging purposes.

Function Declaration: bool CAMSERINTCC csi_GrabSingleFrameImage(UINT16 bufSelect = IMG_GRAB_BUFFER_B)

Parameters: bufSelect – image grab buffer index on camera (Buffer B is default)

Returns: True if successful, false if error occurred

Example:

```
pData = new UWord16[PC_MSTR_XFER_BUF_SIZE];

pWords = (UWord16*)pData;

// First Send Image Grab Command (default is buffer B)
csi_GrabSingleFrameImage();

// Setup Transfer Data Structure
xferInfo.addr = MAR_IMAGE_GRAB_B;
xferInfo.size = PC_MSTR_XFER_BUF_SIZE;

for (y = 0, j = 0;
     j < LOOPS_FOR_IMAGE_GRAB;
     j++, xferInfo.addr += PC_MSTR_XFER_BUF_SIZE)
{
    // Now go and get 1st four lines of image for debug viewing
    status = csi_ReadUtilityMemory(pData, &xferInfo);

    if (!status)
    {
        theApp.OutputErrorStatus(_T("Error Reading Utility Memory\n"));
        break;
    }
    else
    {
        if (j == 0)
        {
            m_lbxReadout.ResetContent();

            tStr.Format(
                _T("Image Grab: First %d Pixels"),
                PC_MSTR_XFER_BUF_SIZE * LOOPS_FOR_IMAGE_GRAB
            );
            m_lbxReadout.AddString(tStr);
        }

        pWords = (UWord16*)pData;
        for (x = 0; x < PC_MSTR_XFER_BUF_SIZE; x += 4, y += 4, pWords += 4)
        {
            tStr.Format(
                _T("0x%04X:\t%04X\t%04X\t%04X\t%04X"),
                y,
                pWords[0],
                pWords[1],
                pWords[2],
                pWords[3]
            );

            m_lbxReadout.AddString(tStr);
        }
    }
}
```

5.16.3 csi_GetLensDescriptor

Description: Routine to read the lens descriptor table from serial

Function Declaration: bool CAMSERINTCC csi_GetLensDescriptor(ptr_cam_LensDescriptor pLD)

Parameters: pLD – pointer to structure of user supplied destination buffer

Returns: True if data is read, false if error occurred

Example:

```
cam_LensDescriptor    m_LensDesc[4];

// Get possible lens configurations
if (csi_GetLensDescriptor(m_LensDesc))
{
    // Set the Lens ID possible choices
    FillLensIDComboBox();

    // Set the OpMode Spinner Limits
    AdjustOpModeSpinnerLimits();
}
```

5.16.4 csi_TempRefreshDisable

Description: Routine to set flag on camera to temporarily disable any automatic refresh calibrations. Typically this would be used in a situation where an automatic refresh might interfere with data acquisition or to speed up file uploads/downloads.

Function Declaration: bool CAMSERINTCC csi_TempRefreshDisable(bool disable)

Parameters: disable – true to disable automatic refreshes, false to re-enable

Returns: True if successful, false if error occurred

Example:

```
// Temporarily disable NUC refresh
csi_TempRefreshDisable(true);

// Perform some task

...

// Re-enable NUC refresh
csi_TempRefreshDisable(false);
```

5.16.5 csi_CallInProgress

Description: Routine to set flag on camera to temporarily disable automatic update of analog offset and DAC 0 based on camera ambient temperature.

Function Declaration: bool CAMSERINTCC csi_CallInProgress(bool disable)

Parameters: disable – true to disable automatic updates, false to re-enable

Returns: True if successful, false if error occurred

Example:

```
// Set the Calibration In Progress flag
csi_CallInProgress(true);

// Perform some task

...

// Clear the Calibration In Progress flag
csi_CallInProgress(false);
```

5.16.6 csi_FrameInterruptDisable

Description: Routine to set flag on camera to temporarily disable the frame interrupt routine on the camera. This interrupt is currently used to read/track the ADC (power and temperature) values. Typically used in conjunction with 'csi_TempRefreshDisable'.

Function Declaration: void CAMSERINTCC csi_FrameInterruptDisable(bool disable)

Parameters: disable – true to disable the interrupt, false to re-enable

Returns: None

Example:

```
// Temporarily Disable the frame interrupt
csi_FrameInterruptDisable(true);

// Perform som task
...

// Re-enable the frame interrupt
csi_FrameInterruptDisable(false);
```

5.16.7 csi_SetServoMode

Description: Routine send a control command to the calibration flag servo.

Function Declaration: bool CAMSERINTCC csi_SetServoMode(UINT16 state, UINT16 factor)

Parameters: state – enumeration index for command (see header file); factor – value if command state is 'factor' else ignored

Returns: True if successful, false if error occurred

Example:

```
if (csi_Connected())
{
    UpdateData();

    // Set state mode
    if (m_bCloseCalFlag)
    {
        state = SERVO_CLOSE;
    }
    else
    {
        state = SERVO_OPEN;
    }

    // Send cal flag servo command
    if (!csi_SetServoMode(state, 0))
    {
        theApp.OutputErrorStatus(csi_GetLastError());
    }
}
```

5.16.8 csi_UpdateModelInfo

Description: Routine to tell the camera that base/limit NUC table mode information (in flash) has been updated. Typically called after updating lens descriptor or operational mode tables .

Function Declaration: bool CAMSERINTCC csi_UpdateModelInfo(void)

Parameters: None

Returns: None

Example:

```
bool          status;
pcm_FlashXfer xferData;
CString      tStr = _T("");

// Copy data to local object structure
GetDialogControlData(false);

xferData.eraseFlag = 0;
xferData.offset = 0;
xferData.page = LENS_DESCRIPTOR_BASE_PAGE;
xferData.size = sizeof(cam_LensDescriptor) * 4;

// Write to the page that contains the radiometric data
status = csi_WritePartialSerialFlashPage(
    &m_LocalLensDesc[0],
    &xferData
);

// If we get the data then check fo filename
if (!status)
{
    tStr.Format(_T("Error Writing Lens Descriptor Table"));
    theApp.DoMessageBox(tStr, MB_OK, 0);
}
else
{
    csi_UpdateModeInfo();
}
```

6 Custom Commands

This section covers a list of custom commands that can be sent to the camera to perform a task that can not be accomplished via direct memory or register writes. The commands will be sent using the “csi_PCMSendAppCmd” function (see paragraph 5.3.2).

6.1 Operational Software Defined Commands

Once the serial connection has been verified a command can be sent to the camera electronics. A full list of available commands for the electronics is enumerated in Appendix C, and 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. Many of these commands have already been incorporated into exported functions listed in paragraph 5.

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.

```
...
// Wait until camera DSP is ready for next command
CsiCmdStatusReady();

// Send focus motor 'far' command
csi_PCMSendAppCmd(CMD_FOCUS_MOTOR_FAR, 0, NULL);
...
```

Notes:

- The second argument of the call ‘csi_PCMSendAppCmd’ is set to zero and the third 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 ‘csi_PCMReadDataMem’ to retrieve the data from the ‘scratch pad’ buffer into a local host buffer. This is equivalent to calling the ‘csi_ReadSerialFlashPage’ function call.
- 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.).

6.1.1 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 `csi_PCMReadDataMem` function to read data from scratch pad.

6.1.2 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 `_pcm_FlashXfer`

Argument: See below.

```

/* SPI Flash Transfer Data Structure */
struct _pcm_FlashXfer
{
    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 `csi_PCMWriteDataMem` function to move data into scratch pad.

6.1.3 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 `_pcm_FlashXfer`

Argument: See below.

```

/* SPI Flash Transfer Data Structure */
struct _pcm_FlashXfer
{
    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 `csi_PCMWriteDataMem` function to move data into scratch pad.

6.1.4 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 csi_PCMWriteDataMem function to move data into scratch pad. Associated data structures listed below:

```

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

/* Top level structure to hold info on each defined component */
struct _cam_ProdID
{
    cam_SerNum  camera;
    cam_SerNum  controllerBD;
    cam_SerNum  camSupportBD;
    cam_SerNum  fpaSupportBD;
    cam_SerNum  calFlagAssy;
    cam_SerNum  peripheral[4];
    cam_SerNum  fpa;
    UWord16     ReserveBlk_A[2];
};

```

6.1.5 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 csi_PCMReadDataMem function to read data from scratch pad. See data structures above.

6.1.6 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 csi_PCMWriteDataMem function to move data into scratch pad. Associated data structures listed below:

```

/* Cal Flag Servo Configuration Structure */
struct _cam_CalFlgCfg
{
    UWord16     PwmPeriodFctr;
    UWord16     PwmCloseFctr;
    UWord16     PwmOpenFctr;
};

/* Static Configuration Structure */
struct _cam_StaticCfg

```

Serial Interface Developer's Reference Manual

```

{
    UWord16      RefClkRate;
    UWord16      comCfg;           /* Truncate from COM_PORT */
    UWord16      dspVideo;        /* Truncate from DISP_VIDEO_CFG */
    UWord16      procVideo;       /* Truncate from PROC_VIDEO_CFG */
    cam_CalFlgCfg calFlag;
    UWord16      fpaTempAdcAdj;    /* Fpa Temp ADC Offset Adjust Value */
    UWord16      ReserveBlk_B[24];
};

```

6.1.7 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 csi_PCMReadDataMem function to read data from scratch pad. See data structures above.

6.1.8 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 csi_PCMReadDataMem function to read data from scratch pad. Associated data structures listed below:

```

/* Structure to hold time info from Real Time Clock */
struct _cam_RtcData
{
    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) */
};

```

6.1.9 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 csi_PCMWriteDataMem function to move data into scratch pad. See data structures above.

6.1.10 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 _pcm_NvmXfer.

Argument: See below.

```

/* NVM Transfer Data Structure */
struct _pcm_NvmXfer
{
    UWord16    offset;           // Offset from byte 0
    UWord16    size;           // Size in bytes
};

```

Note: Use the csi_PCMReadDataMem 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 E for information on how the memory is mapped on the serial NVM part.

6.1.11 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 6.1.10

Note: Use the csi_PCMWriteDataMem 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 E for information on how the memory is mapped on the serial NVM part.

6.1.12 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.

6.1.13 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.

6.1.14 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.

6.1.15 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 _pcm_MemXfer.

Argument: See below.

```
/* Utility Memory Transfer Data Structure */
struct _pcm_MemXfer
{
    UWord32    addr; // Addr in utility/NUC flash/DSP data flash memory to read/write
    UWord16    size; // Size in WORDS (Limit to 160)
};
```

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 0x00026FFF */
#define MAR_IMAGE_GRAB_B   0x00027000 /* Thru 0x00041FFF */
#define MAR_SYMBOLGY_OVLY  0x00042000 /* Thru 0x0005CFFF */
#define MAR_NUC_REFRESH     0x0005D000 /* Thru 0x00077FFF */
#define MAR_EXT_DATA        0x00078000 /* Thru 0x0007FFFF */
```

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

6.1.16 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.

6.1.17 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 _pcm_MemXfer.

Argument: See below.

```
/* Utility Memory Transfer Data Structure */
struct _pcm_MemXfer
{
    UWord32    addr; // Addr in utility/NUC flash/DSP data flash memory to read/write
    UWord16    size; // Size in WORDS (Limit to 160)
};
```

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

6.1.18 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.

6.1.19 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.

6.1.20 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.

6.1.21 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 _misc_ServoMode.

Argument: See below.

```

/* Commanded Servo Setting Structure */
struct _misc_ServoMode
{
    UWord16    mode;        /* Mode of Servo: Open, Closed, Factor */
    UWord16    factor;      /* PWM Factor for user specified setting */
};

/* Servo Mode Enumeration */
enum eServoState
{
    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

6.1.22 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 _misc_CalFlagState.

Argument: See below.

```

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

/* Calibration Reference Flag Enumeration */
enum eCFState
{
    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.

6.1.23 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 8.1 for details.

6.1.24 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.

6.1.25 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.

6.1.26 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.

6.1.27 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.

6.1.28 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. This is a debug function for more accurate results read the filtered ADC values from the global data structure.

Command Code: Enumeration for CMD_GET_ADC_VALUES

Argument Size: 0

Argument: Null

Note: Use the `csi_PCMReadDataMem` 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.

6.1.29 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 `_pcm_ReticleXfer`.

Argument: See below.

```

/* Reticle Command Info Structure */
struct _pcm_ReticleXfer
{
    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
};

```

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

6.1.30 CMD_RETICLE_POSITION

Description: Move the selected reticle to desired location.

Command Code: Enumeration for CMD_RETICLE_POSITION

Argument Size: size of `_pcm_ReticleXfer`.

Argument: See paragraph 6.1.29.

Note: This command and CMD_ENABLE_RETICLE are functionally identical.

6.1.31 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: 0

Argument: NULL

Note: None.

6.1.32 CMD_WRITE_VID_ENC_REG

Description: Command to write value to video encoder register. This is a debug command function and should not be used.

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.

6.1.33 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.

```

/* Test selection index enum */
enum eCamTestSelect
{
    TEST_REGISTERS = 0,
    TEST_OPERATIONAL,
    TEST_MEMORY,
    TEST_UNIFORMITY,
    TEST_SENSITIVITY,
    TEST_NOISE,

    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).

6.1.34 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.

6.1.35 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.

6.1.36 CMD_DETECT_BAD_PIXELS

Description: Command no longer used.

6.1.37 CMD_IRCON_LOAD_LUT

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

6.1.38 CMD_LOAD_RAD_PARAMS

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

6.1.39 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.

6.1.40 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.

6.1.41 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.

6.1.42 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.

```

/* Utility Memory Transfer Data Structure */
struct _pcm_MemXfer
{
    UWord32    addr; // Addr in utility/NUC flash/DSP data flash memory to read/write
    UWord16    size; // Size in WORDS (Limit to 160)
};

```

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

6.1.43 CMD_ADV_DETECT_BAD_PIXELS

Description: Command no longer used.

6.1.44 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 multiple times to upload all the coefficient terms.

Command Code: Enumeration for CMD_UPLOAD_NUC

Argument Size: UWord16 parameter[2]

Argument:

parameter [0] – 0 for ATC gain terms, 1 for FPA gain terms, 2 – for FPA offset terms, and 3 – for Calibration Flag factor terms.

parameter [1] – The active NUC table (linear) 0 – 3 typical.

Note: See paragraph 8.3 for details on how to complete this process.

6.1.45 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 multiple times to download all the coefficient terms.

Command Code: Enumeration for CMD_DOWNLOAD_NUC

Argument Size: UWord16 parameter[2]

Argument:

parameter [0] – 0 for ATC gain terms, 1 for FPA gain terms, 2 – for FPA offset terms, and 3 – for Calibration Flag factor terms.

parameter [1] – The active NUC table (linear) 0 – 3 typical.

Note: See paragraph 8.4 for details on how to complete this process.

6.1.46 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.

6.1.47 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.

6.1.48 CMD_PROG_DSP_DATA_FLASH

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_PROG_DSP_DATA_FLASH

Argument Size: size of _pcm_MemXfer.

Argument: See below.

```

/* Utility Memory Transfer Data Structure */
struct _pcm_MemXfer
{
    UWord32    addr; // Addr in utility/NUC flash/DSP data flash memory to read/write
    UWord16    size; // Size in WORDS (Limit to 160)
};

```

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

6.1.49 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.

6.1.50 CMD_UPDATE_MODE_INFO

Description: Informs the camera software that a value associated with the base and limit mode index has been modified in the serial flash (typically after modifying the lens descriptor table). The camera will then read the data structure and make the necessary adjustments.

Command Code: Enumeration for CMD_UPDATE_MODE_INFO

Argument Size: 0

Argument: None

Note: None.

6.1.51 CMD_RESTORE_SPI_NVM

Description: Reads stored factory/user default settings from the SPI flash and writes those settings to the NVM memory.

Command Code: Enumeration for CMD_RESTORE_SPI_NVM

Argument Size: 0

Argument: None

Note: The current NVM information will be lost.

7 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 6.1) 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.

7.1 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 6.1.4 - 6.1.7. 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.

7.2 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 'csi_PCMReadDataMem' 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.

7.2.1 CameraConfig.nvmData

Type: NVM_GLOBAL_CFG

Size: 40 Words

Description: See paragraph 7.3.

7.2.2 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.

7.2.3 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.

7.2.4 CameraConfig.CmdsReceived

Type: UWord16

Size: 1 Word

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

7.2.5 CameraConfig.camStats

Type: _cam_Status

```

/* Camera Status Code Structure */
struct _cam_Status
{
    UWord16    ProcessCode;    /* Process Code Value */
    UWord16    ProgressCode;   /* Progress Code Value */
    UWord16    HostStatusCode; /* Host Status Code */
    UWord16    HostData;      /* Misc Data for status code */
};

```

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 7.5 below for information on the progress code detection.

7.2.6 CameraConfig.camErrors

Type: _cam_Errors

```

/* Camera Error Code Structure */
struct _cam_Errors

```

Serial Interface Developer's Reference Manual

```

{
    UWord16    ErrorCode;        /* Error Code */
    UWord16    ErrorSubCode;    /* Error Code */
    UWord16    ErrorCount;      /* Additional errors after 1st */
    UWord16    ErrorData[5];    /* Error Data Array */
};

```

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 7.6 for information on the error code detection.

7.2.7 CameraConfig.camTime

Type: _cam_RtcData

```

/* Structure to hold time info from Real Time Clock */
struct _cam_RtcData
{
    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

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

7.2.8 CameraConfig.expPort

Type: UWord16

Size: 1 Word

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

7.2.9 CameraConfig.swVersion

Type: UWord16

Size: 1 Word

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

7.2.10 CameraConfig.swBuild

Type: UWord16

Size: 1 Word

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

7.2.11 CameraConfig.fpgaVersion[2]

Type: UWord16

Size: 2 Words

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

7.2.12 CameraConfig.fpaSize

Type: _cam_ImageSize

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

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.

7.2.13 CameraConfig.agcLowIntensity

Type: UWord16

Size: 1 Word

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

7.2.14 CameraConfig.agcHighIntensity

Type: UWord16

Size: 1 Word

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

7.2.15 CameraConfig.actOpName[4]

Type: UWord16

Size: 4 Words (8 Bytes)

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

7.2.16 CameraConfig.actNucName[4]

Type: UWord16

Size: 4 Words (8 Bytes)

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

7.2.17 CameraConfig.modeRange

Type: UWord16

Size: 1 Word

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

Base Operational Mode: (Bits 0 - 2) Base OpMode Index. Read only data member.

Limit Operational Mode: (Bits 3 - 5) Limit OpMode index. Read only data member.

Base NUC table: (Bits 6 - 9) Base NUC index. Read only data member.

Limit NUC table: (Bits 10 - 13) Limit NUC index. Read only data member.

Empty: (Bits 14, 15) Not used.

7.2.18 CameraConfig.radSWInfo

Type: UWord16

Size: 1 Word

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

Customer ID: (Bits 0 - 7). Read only data member.

Radiometric Software Version: (Bits 8 - 15). Read only data member.

7.2.19 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.

Reserved Bits: (Bits 4 – 13).

Cooler/TEC Standby Mode (Temp): (Bit 14) Standby mode flag based on internal temperature.

Cooler/TEC Standby Mode (User): (Bit 15) Standby mode flag set by host.

7.2.20 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)

7.2.21 CameraConfig.fpaLens

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.

Serial Interface Developer's Reference Manual

FPA Type Identifier: (Bits 0 – 3)

FPA Sub-type Identifier: (Bits 4 – 7)

Lens ID: (Bits 8 - 13)

Reserved: (Bits 14 – 15)

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

```

/* Fpa Type Enumerations */
enum eFpaType
{
    FPA_NONE = 0,                /* No FPA */
    FPA_DRS_U3000A,
    FPA_ULIS_UL0308,
    FPA_RESERVED,
    FPA_INDIGO_9705,
    FPA_INDIGO_9809,
    FPA_INDIGO_0202,

    FPA_UNDEFINED                /* The rest are Reserved */
};

/* Fpa Sub-Type Enumerations */
enum eFpaSubTypeUlis
{
    FPA_ULIS_UL0308_IMAGER = 0,

    FPA_ULIS_UL0308_UNDEFINED    /* The rest are Reserved */
};

/* Fpa Sub-Type Enumerations */
enum eFpaSubType9705
{
    FPA_9705_INSB = 0,

    FPA_9705_UNDEFINED          /* The rest are Reserved */
};

/* Fpa Sub-Type Enumerations */
enum eFpaSubType9809
{
    FPA_9809_INSB = 0,
    FPA_9809_INGAAS,

    FPA_9809_UNDEFINED          /* The rest are Reserved */
};

/* Lens ID */
enum eLensIDIndex
{
    LENS_NONE = 0,

    /* ID's 1 - 16: Fixed Focus/No Zoom */
    LENS_RSVD_1,
    LENS_RSVD_2,
    LENS_13MM_FF_NZ, /* 13mm, Fixed Focus, No Zoom */
    LENS_RSVD_4,
    LENS_25MM_FF_NZ, /* 25mm, Fixed Focus, No Zoom */
    LENS_RSVD_6,
    LENS_50MM_FF_NZ, /* 50mm, Fixed Focus, No Zoom */
    LENS_RSVD_8,
    LENS_RSVD_9,
    LENS_100MM_FF_NZ, /* 100mm, Fixed Focus, No Zoom */
    LENS_RSVD_11,
    LENS_RSVD_12,
    LENS_RSVD_13,
    LENS_RSVD_14,

```

Serial Interface Developer's Reference Manual

```

LENS_RSVD_15,
LENS_RSVD_16,

/* ID's 17 - 31: Manual Focus/No Zoom */
LENS_RSVD_17,
LENS_RSVD_18,
LENS_13MM_MF_NZ, /* 13mm, Manual Focus, No Zoom */
LENS_RSVD_20,
LENS_25MM_MF_NZ, /* 25mm, Manual Focus, No Zoom */
LENS_RSVD_22,
LENS_50MM_MF_NZ, /* 50mm, Manual Focus, No Zoom */
LENS_RSVD_24,
LENS_RSVD_25,
LENS_100MM_MF_NZ, /* 100mm, Manual Focus, No Zoom */
LENS_RSVD_27,
LENS_RSVD_28,
LENS_RSVD_29,
LENS_RSVD_30,
LENS_RSVD_31,

/* ID's 32 - 63: Reserved for Advanced Lenses */
LENS_RSVD_32_TO_63 = 32,

LENS_UNDEFINED = 64,
};

```

7.2.22 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 values have had a filtering algorithm applied for stability and accuracy.

Index [0]: DSP Channel A0

Index [1]: DSP Channel A1

Index [2]: DSP Channel A2

Index [3]: DSP Channel A3

7.2.23 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 values have had a filtering algorithm applied for stability and accuracy.

Index [0]: DSP Channel B0

Index [1]: DSP Channel B1

Index [2]: DSP Channel B2

Index [3]: DSP Channel B3

7.2.24 CameraConfig.btnPanel

Type: _cam_BtnPanel

```

/* Button Panel State Structure */
struct _cam_BtnPanel
{

```

Serial Interface Developer's Reference Manual

```

    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;
};

```

Size: 2 Words

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

7.2.25 CameraConfig.miscState

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.

- Fan Enable: (Bit 0)
- Calibration in progress flag: (Bit 1)
- Reserved: (Bit 2)
- Range reticle enable flag: (Bit 3)
- Lockout 1-Pt refresh flag: (Bit 4)
- Lockout memory mirror flag: (Bit 5)
- Previous DAC 0 setting: (Bits 6 - 13)
- Empty: (Bits 14 - 15)

7.2.26 CameraConfig.currAtcIndex

Type: UWord16

Size: 1 Word

Description: The current index of the Ambient Temperature Coefficient look up table. The index is computed from the front end sensor temperatures.

7.3 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 F.

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 'csi_PCMWriteDataMem' routine to change a setting.

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

7.3.1 nvmData.CamMode

Type: UWord16

Size: 1 Word

Description: See Appendix F for specifics on this value.

7.3.2 nvmData.FpaMode

Type: UWord16

Size: 1 Word

Description: See Appendix F for specifics on this value.

7.3.3 nvmData.LensMode

Type: UWord16

Size: 1 Word

Description: See Appendix F for specifics on this value.

7.3.4 nvmData.ActMode

Type: UWord16

Size: 1 Word

Description: See Appendix F for specifics on this value.

7.3.5 nvmData.AutoNucData

Type: _nvm_AutoNucMode

```

/* Auto NUC Switch Parameters (Addr: 42) */
struct _nvm_AutoNucMode
{
    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 */
};

```

Size: 4 Words

Description: Currently not implemented. See Appendix F for specifics on this value.

7.3.6 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.

7.3.7 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.

7.3.8 nvmData.ActPal

Type: UWord16

Size: 1 Word

Description: See Appendix F for specifics on this value.

7.3.9 nvmData.OvlMode

Type: UWord16

Size: 1 Word

Description: See Appendix F for specifics on this value.

7.3.10 nvmData.RtclXPos

Type: _nvm_ReticlePos

```

/* Reticle Position & Emissivity (Addr: 62 & 66) */
struct _nvm_ReticlePos
{
    unsigned   RtclHorPos:9;   /* Reticle Horizontal Position */
    unsigned   Reserved:7;    /* Reserved */
    unsigned   RtclVerPos:8;   /* Reticle Vertical Position */
    unsigned   RtclEmiss:8;    /* Reticle Emissivity */
};

```

Size: 2 Words

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

7.3.11 nvmData.RadMode

Type: UWord16

Size: 1 Word

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

7.3.12 nvmData.AgcMode

Type: UWord16

Size: 1 Word

Description: See Appendix F for specifics on this value.

7.3.13 nvmData.ManualITT

Type: _nvm_ManualITT

```
/* Display Video Brightness & Contrast Data (Addr: 76) */
struct _nvm_ManualITT
{
    unsigned LowInt:14;      /* Manual Low Intensity */
    unsigned ReservedA:2;    /* Reserved */

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

Size: 2 Words

Description: See Appendix F for specifics on this value.

7.3.14 nvmData.AgcLimits

Type: _nvm_AgcLimits

```
/* AGC Intensity Limits (Addr: 80) */
struct _nvm_AgcLimits
{
    unsigned AgcLowLimit:14; /* AGC Low Limit Intensity */
    unsigned ReservedA:2;    /* Reserved */

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

Size: 2 Words

Description: See Appendix F for specifics on this value.

7.3.15 nvmData.LinearMap

Type: UWord16

Size: 1 Word

Description: See Appendix F for specifics on this value.

7.3.16 nvmData.AgcBinLimit

Type: UWord16

Size: 1 Word

Description: See Appendix F for specifics on this value.

7.3.17 nvmData.VidScaleTemps

Type: _nvm_VidScaleTemps

```
/* Display Video Zero & Full Scale Temperatures (Addr: 98) */
struct _nvm_VidScaleTemps
{
    Word16 ZeroScaleTemp; /* Display Video Zero Scale Temp */
    Word16 FullScaleTemp; /* Display Video Full Scale Temp */
};
```

Size: 2 Words

Description: See Appendix F for specifics on this value.

7.3.18 nvmData.ImageParams

Type: UWord16

Size: 1 Word

Description: See Appendix F for specifics on this value.

7.4 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 eCamProcessIndex
{
    PRC_UNDETERMINED = 0,
    PRC_FPGA_TESTS,
    PRC_OP_TESTS,
    PRC_MEM_TESTS,
    PRC_VIDENC_TESTS,
    PRC_TEC_STABILIZING,
    PRC_FPA_TEMP_OUT_OF_RANGE,

    PRC_CAM_READY
};

```

7.5 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 eCamProgressIndex
{
    UNDETERMINED = 0,           /* 0x0000 */
    UNCONFIGURED_CONTROLLER,  /* 0x0001 */
    FAULT_DETECTED,           /* 0x0002 */
    MAIN_START,                /* 0x0003 */
    DRIVERS_INITIALIZED,      /* 0x0004 */
    GLOBALS_INITIALIZED,      /* 0x0005 */
    SYSTEM_RESET,              /* 0x0006 */
    BOARD_ID_CONFIRM,          /* 0x0007 */
    FPGA_PROGRAMMED,           /* 0x0008 */
    FPGA_REGISTER_TEST,        /* 0x0009 */
    CAMERA_OP_TEST_SETUP,     /* 0x000A */
    INIT_FPA_SUPPORT_PWR,      /* 0x000B */
    TEC_DRIVE_INITIALIZED,     /* 0x000C */
    SYSTEM_TESTS_COMPLETE,     /* 0x000D */
    CAL_FLAG_INITIALIZED,      /* 0x000E */
    ENTERING_MAIN_LOOP,        /* 0x000F */
};

```

Serial Interface Developer's Reference Manual

```

PALETTES_LOADED,          /* 0x0010 */
OP_MODE_LOADED,          /* 0x0011 */
OVERLAY_INITIALIZED,     /* 0x0012 */
TEC_STABILIZED,         /* 0x0013 */

CAMERA_READY             /* 0x0014 */
};

```

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 7.6) and then set the continue flag (structure member 'continueFlag', paragraph 7.2.3). 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.

7.6 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 7.2.5). 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 7.2.5) 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 7.2.3). 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_RESULTS_UNKNOWN     0x8FFE
#define ERR_TEST_SKIPPED        0x8FFF

```

7.6.1 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 eCamTestErrConfigSubCode
{
    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.

7.6.2 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 eCamTestErrFpgaSubCode
{
    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.

7.6.3 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 eCamTestErrMemSubCode
{
    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.

7.6.4 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.

7.6.5 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.

7.6.6 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.

7.6.7 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.

7.6.8 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.

7.6.9 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.

7.6.10 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.

7.7 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 eCamCmdStatus
{
    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 csi_PCMReadDataMem command to retrieve that data.

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

7.8 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 'csi_PCMReadDataMem/csi_PCMWriteDataMem' 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.

7.9 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 'csi_PCMReadDataMem/csi_PCMWriteDataMem' 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.

7.9.1 FPA Processor Operational Control Register Low

Address: 0x4000

Bit 0 (Zero ATC Gain): 0 – Use NUC Coefficient Memory ATC Gain; 1 – Force NUC ATC Gain to Zero

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

Bit 2 (Zero FPA Offset): 0 – Use NUC Coefficient Memory FPA Offset; 1 – Force NUC FPA 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 5 (Atc Byte Sel): 0 – Select Low ATC Gain Coefficient (low temp); 1 - Select High ATC Gain Coefficient (high temp);

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

Bit 7 (Reserved)

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

Bit 11 (Dspl Byte Sel): 0 – ITT/FB Bits 7:0 Displayed; 1 – ITT/FB Bits 15:8 Displayed

Bit 12 (IH Grab Sel): 0 – Grab ITT byte selected by Dspl Byte Sel; 1 – Grab complete 16-bit value

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

Bit 14 (Reserved)

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

7.9.2 FPA Processor Operational Control Register High

Address: 0x4001

Bit 0 (DFId 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

7.9.3 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

7.10 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.

<u>Page(s)</u>	<u>Allocated Use</u>
0	Lens Descriptor Table
1	Factory NVM Settings Page
2 - 7	Reserved
8 - 15	Op Mode Descriptor Tables (8x)
16 - 27	NUC Mode Descriptor Tables (12x)
32 - 43	FPA Support DAC 0 LUT Tables (12x)
48 - 71	Analog Offset Adjust LUT Tables (12x)
72 - 190	Reserved
191	Overlay Palettes (8x)
192 - 223	Color Palette Y Tables (16x)
224 - 255	Color Palette Cr/Cb Tables (16x)
256 - 511	Reserved
512 - 1145	FPA Processor FPGA Configuration File
1146 - 1535	Expansion FPGA Configuration File
1536 - 1919	ATC Look Up Tables (12x)
2048 - 4095	Reserved

Serial Interface Developer's Reference Manual

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.

8 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 status enumerations */
enum eCamCalStatusIndex
{
    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,

    COMPILING_DEFECT_LISTS,

    CAL_FPA_OFFSETS_COMPUTED,
    CAL_STORE_FPA_OFFSETS,

    HOST_UNDEFINED
};

```

8.1 One Point Refresh Calibration (Internal Flag Only):

- (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

8.2 One Point Update Calibration (Internal Flag Only):

- (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.

Calibration is complete.

8.3 Upload NUC Table from Host

It is not recommended that the user proceed with an upload process without contacting Lumitron for specifics. Improper data coefficients or upload error can render the camera useless. Please refer to source code example (provided by Lumitron if purchased) for an example of the upload process.

8.4 Download NUC Table to Host

It is not recommended that the user proceed with a download process without contacting Lumitron for specifics. Please refer to source code example (provided by Lumitron if purchased) for an example of the download process.

Appendix A - Camera Configuration and Mode Data Structures

The following data structures have been extracted from the camera operational software project. It is important to note that the software project is targeted for a 16-bit processor; therefore care should be taken when copying/casting data to and from a host (32-bit) system.

```

/*****
/***** LENS DESCRIPTOR STRUCTURES *****/
/*****
/* Lens Descriptor Table Structure */
typedef struct
{
    unsigned    lensID:6;          /* Lens ID (Bits 0 - 5) */
    unsigned    rsvd1:2;          /* Reserved (Bits 6 - 7) */
    unsigned    baseMode:3;       /* Base OpMode Index (Bits 8 - 10) */
    unsigned    rsvd2:1;          /* Reserved (Bit 11) */
    unsigned    limitMode:3;      /* Base OpMode Index (Bits 12 - 14) */
    unsigned    rsvd3:1;          /* Reserved (Bit 15) */

    unsigned    pulseWidth:8;     /* Pulse Width Value (ms) (Bits 16 - 23) */
    unsigned    rsvd4:8;          /* Reserved (Bits 24 - 31) */

} sLensDescriptor;      /* Size = 2 Words */

/*****
/***** OP MODE STRUCTURES *****/
/*****
/* Camera Acquisition ROI Structure - Actual Register Values to be Stored */
typedef struct
{
    UWord16    HorROIStart;       /* Acq Horizontal ROI Start Factor */
    UWord16    HorROIStop;        /* Acq Horizontal ROI Stop Factor */
    UWord16    VerROIStart;       /* Acq Vertical ROI Start Factor */
    UWord16    VerROIStop;        /* Acq Vertical ROI Stop Factor */

} sModeRoi;      /* Size = 4 Words */

/* NUC Mode Register Structure */
typedef struct
{
    unsigned    baseNuc:4;        /* Base NUC Mode (Bits 0 - 3) */
    unsigned    limitNuc:4;       /* Limit NUC Mode (Bits 4 - 7) */
    unsigned    reserved:6;       /* Reserved (Bits 8 - 13) */
    unsigned    asyncMod:1;       /* Asynch Mode (Bit 14) */
    unsigned    nonImgMod:1;      /* Non-Imaging Mode (Bit 15) */

} sNucRange;      /* Size = 1 Word */

/* FPA Size Index Register Structure */
typedef struct
{
    unsigned    verFctr:2;        /* Vertical FPA Size Factor (Bits 0 - 1) */
    unsigned    rsvd1:6;          /* Reserved (Bits 2 - 7) */
    unsigned    horFctr:2;        /* Horizontal FPA Size Factor (Bits 8 - 9) */
    unsigned    rsvd2:6;          /* Reserved (Bits 10 - 15) */

} sFpaSize; /* Size = 1 Word */

/* Camera Operational Mode Descriptor Table Structure - Actual Register Values to be Stored */
typedef struct
{
    sNucRange    NucRange;        /* NUC Limit/Base Index, TEC Mode, Imaging Mode */

    UWord16    Reserve_A;        /* Reserved */

    UWord16    ProgSyncLSB;      /* Programmable Sync Factor LSB's */
    UWord16    ProgSyncMSB;      /* Programmable Sync Factor MSB's */

```

Serial Interface Developer's Reference Manual

```

UWord16  HorCountPreload; /* Display Horizontal Counter Preload */
UWord16  HorImgStartFctr; /* Display Horizontal Image Start Factor */
UWord16  HorImgStopFctr; /* Display Horizontal Image Stop Factor */

UWord16  VerCountPreload; /* Display Vertical Counter Preload */
UWord16  VerImgStartFctr; /* Display Vertical Image Start Factor */
UWord16  VerImgStopFctr; /* Display Vertical Image Stop Factor */

UWord16  SyncDelayLSB; /* Sync Delay Factor LSB's */
UWord16  SyncDelayMSB; /* Sync Delay Factor MSB's */

UWord16  FpaHorStartFctr; /* FPA Horizontal Start Factor */
UWord16  FpaHorStopFctr; /* FPA Horizontal Stop Factor */
UWord16  FpaHorCountFctr; /* FPA Horizontal Terminal Count Factor */

UWord16  FpaVerStartFctr; /* FPA Vertical Start Factor */
UWord16  FpaVerStopFctr; /* FPA Vertical Stop Factor */
UWord16  FpaVerCountFctr; /* FPA Vertical Terminal Count Factor */

sFpaSize  FpaSize; /* FPA Spatial Size Factors */

UWord16  ReserveBlk_A[45]; /* Reserved */

sModeRoi  modeRoi[8]; /* Array of ROI settings */

UWord16  ReserveBlk_B[32]; /* Reserved */

UWord16  OpName[4]; /* Op Mode Name */

} sOpMode; /* Size = 132 Words */

```

Serial Interface Developer's Reference Manual

```

/*****
/***** NUC MODE STRUCTURES *****/
/*****
/* Nuc Address Structure */
typedef struct
{
    unsigned    NucAddress:8;    /* NUC Coefficient Table Base Address */
    unsigned    ReservedBits_A:8; /* Reserved Bits */
} sNucBase; /* Size = 1 Word */

/* Camera NUC Mode Descriptor Table Structure */
typedef struct
{
    sNucBase    nucBase;        /* Nuc Base Address data */

    UWord16    FpaIntegFctrLSB; /* FPA Integration Factor LSB (15 - 0) */
    UWord16    FpaIntegFctrMSB; /* FPA Integration Factor MSB (19 - 16) */

    UWord16    SerialCtlWord[4]; /* FPA Serial Control Word Array */

    UWord16    DacSetPoint[4];   /* FPA Support DAC Set Point Array */

    UWord16    ReserveBlk_A[19]; /* Reserved */

    UWord16    TempSensorWF;     /* Temperature Sensor Weighting Factor */

    UWord16    EffBaseAdcTemp;   /* Steady State Base Effective Temperature
                                in ADC counts */

    UWord16    CalFlagCold;     /* Cal Flag Ref Cold Set Point */
    UWord16    CalFlagHot;      /* Cal Flag Ref Hot Set Point */

    UWord16    DacXferGain[4];   /* DAC Transfer Function Gain Value
                                (stored as double for host) */

    UWord16    DacXferOffset[4]; /* DAC Transfer Function Offset Value
                                (stored as double for host) */

    UWord16    CF1CoeffScale;    /* Cal Flag Factor First Order Coeff Scale */
    UWord16    CF2CoeffScale;    /* Cal Flag Factor Second Order Coeff Scale */

    UWord16    ReserveBlk_B[20]; /* Reserved */

    UWord16    radData[64];     /* Customer Defined Radiometric Data (Placeholder) */

    UWord16    NucName[4];      /* Nuc Mode Name */
} sNucMode; /* Size = 132 Words */

```

Serial Interface Developer's Reference Manual

```

/*****
/***** DYNAMIC CONFIGURATION STRUCTURES *****/
/*****/

/* Camera Mode Control */
typedef struct
{
    unsigned    CVid0OutEn:1;    /* Composite Video 0 Output Enable */
    unsigned    CVid1OutEn:1;    /* Composite Video 1 Output Enable */
    unsigned    SVidOutEn:1;     /* SVideo Output Enable */
    unsigned    PfvOutEn:1;      /* Processed Video Out Enable */
    unsigned    ReservedA:3;     /* Reserved */
    unsigned    AutoNucSw:1;     /* Auto NUC Switch Enable */
    unsigned    AutoRfshTim:1;   /* Elapsed Time Refresh Calibration Enable */
    unsigned    AutoRfshTmp:1;  /* Temp Excursion Refresh Calibration Enable */
    unsigned    RfshOnBoot:1;   /* Refresh Calibration on Boot Enable */
    unsigned    RfshOnSwitch:1; /* Refresh Calibration on Switch Enable */
    unsigned    ZnStatEn:1;     /* Zone Statistics Computation Enable */
    unsigned    ReservedC:1;    /* Reserved */
    unsigned    RtclAEn:1;      /* Reticle A Enable */
    unsigned    RtclBEn:1;      /* Reticle B Enable */
} sCamMode;

/* FPA Processor User Mode Control */
typedef struct
{
    unsigned    MstrSyncMod:2;   /* Master Sync Mode */
    unsigned    ReservedA:4;     /* Reserved */
    unsigned    DsplVidPol:1;    /* Display Video Polarity */
    unsigned    ClrBarEn:1;      /* Color Bar Enable */
    unsigned    ClrPltYSel:1;    /* Color Palette Y Select */
    unsigned    ReservedB:1;     /* Reserved */
    unsigned    PfvSrcSel:2;     /* PFV Source Select */
    unsigned    ReservedC:4;     /* Reserved */
} sFpaProcMode;

/* Lens ID Control */
typedef struct
{
    unsigned    LensIndex:2;     /* Lens Index into Descriptor Table */
    unsigned    ReservedA:5;     /* Reserved */
    unsigned    LensAutoDet:1;   /* Lens AutoDetect Bit */
    unsigned    ReservedB:8;     /* Reserved */
} sLensMode;

/* Active Operational Mode & NUC Index */
typedef struct
{
    unsigned    ActOpMod:3;      /* Active Operational Mode */
    unsigned    ReservedA:5;     /* Reserved */
    unsigned    ActNucIdx:4;     /* Active NUC Index (relative) */
    unsigned    ReservedB:4;     /* Reserved */
} sActMode;

/* Auto NUC Switch Parameters */
typedef struct
{
    unsigned    LowSatInt:14;    /* Auto NUC switch low saturation intensity */
    unsigned    ReservedA:2;     /* Reserved */
    UWord16    LowSatCount;     /* Auto NUC switch low saturation count */

    unsigned    HighSatInt:14;   /* Auto NUC switch high saturation intensity */
    unsigned    ReservedB:2;     /* Reserved */
    UWord16    HighSatCount;    /* Auto NUC switch high saturation count */
}

```

Serial Interface Developer's Reference Manual

```

} sAutoNucData;

/* Active Color & Overlay */
typedef struct
{
    unsigned    ActClrPal:4;        /* Active Color Palette */
    unsigned    ReservedA:4;       /* Reserved */
    unsigned    ActOvlPal:3;      /* Active Overlay Palette */
    unsigned    ReservedB:5;      /* Reserved */
} sActPal;

/* Active Overlay Mode & Reticle Size */
typedef struct
{
    unsigned    OvlMod:4;         /* Overlay Mode */
    unsigned    RtclSize:4;       /* Reticle Size */
    unsigned    BgndColor:4;      /* Text Background Color */
    unsigned    FgndColor:4;     /* Text Foreground Color */
} sOvlMode;

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

/* Radiometric Mode */
typedef struct
{
    unsigned    TempUnits:1;      /* Display Temperature Units */
    unsigned    Reserved:15;     /* Reserved */
} sRadMode;

/* AGC Mode */
typedef struct
{
    unsigned    AgcMod:4;         /* AGC Mode */
    unsigned    ReservedA:4;       /* Reserved */
    unsigned    ActRoiIndx:3;     /* Active ROI Index */
    unsigned    ReservedB:5;      /* Reserved */
} sAgcMode;

/* Display Video Manual Intensity Data */
typedef struct
{
    unsigned    LowInt:14;        /* Manual Low Intensity */
    unsigned    ReservedA:2;      /* Reserved */
    unsigned    HighInt:14;      /* Manual High Intensity */
    unsigned    ReservedB:2;      /* Reserved */
} sManualITT;

/* AGC Intensity Limits */
typedef struct
{
    unsigned    AgcLowLimit:14;   /* AGC Low Limit Intensity */
    unsigned    ReservedA:2;      /* Reserved */
    unsigned    AgcHighLimit:14; /* AGC High Limit Intensity */
    unsigned    ReservedB:2;      /* Reserved */
}

```

Serial Interface Developer's Reference Manual

```

} sAgcLimits;

/* AGC Intensity Limits */
typedef struct
{
    unsigned    LinearLowMap:2;    /* AGC Linear Low Map Index */
    unsigned    LinearHighMap:2;   /* AGC Linear High Map Index */
    unsigned    FilterRate:2;      /* AGC Filter Rate */
    unsigned    ReservedA:10;     /* Reserved */
} sLinearMap;

/* Display Video Zero & Full Scale Temperatures */
typedef struct
{
    UWord16     ZeroScaleTemp;     /* Display Video Zero Scale Temp */
    UWord16     FullScaleTemp;     /* Display Video Full Scale Temp */
} sVidScaleTemps;

/* Camera Background Temp & Display Video Emissivity */
typedef struct
{
    unsigned    BckGndTemp:8;      /* Camera Background Temperature (°C) */
    unsigned    VidEmiss:8;        /* Display Video Emissivity */
} sImageParams;

/* Global Exposed Camera Dynamic Configuration Info Structure */
/* See Appendix F below for further info */
typedef struct
{
    /* NVM Addr (Bytes) : Size (Words) : Subtotal Size (Words) */
    sCamMode     CamMode;          /* 032 : 01 : 01 */
    sFpaProcMode FpaMode;          /* 034 : 01 : 02 */

    sLensMode     LensMode;        /* 036 : 01 : 03 */

    sActMode     ActMode;          /* 038 : 01 : 04 */

    UWord16      ReserveB;         /* 040 : 01 : 05 */

    sAutoNucData AutoNucData;      /* 042 : 04 : 09 */

    UWord16      AutoRfshTime;     /* 050 : 01 : 10 */
    UWord16      AutoRfshTemp;     /* 052 : 01 : 11 */

    UWord16      ReserveC[2];      /* 054 : 02 : 13 */

    sActPal      ActPal;           /* 058 : 01 : 14 */
    sOvlMode     OvlMode;          /* 060 : 01 : 15 */
    sRtclPos     RtclAPos;         /* 062 : 02 : 17 */
    sRtclPos     RtclBPos;         /* 066 : 02 : 19 */

    sRadMode     RadMode;          /* 070 : 01 : 20 */

    sAgcMode     AgcMode;          /* 072 : 01 : 21 */

    UWord16      ReserveE;         /* 074 : 01 : 22 */

    sManualITT   ManualITT;        /* 076 : 02 : 24 */
    sAgcLimits   AgcLimits;        /* 080 : 02 : 26 */
    sLinearMap   LinearMap;        /* 084 : 01 : 27 */

    UWord16      AgcBinLimit;      /* 086 : 01 : 28 */

    UWord16      ReserveF[5];      /* 088 : 04 : 32 */

```

Serial Interface Developer's Reference Manual

```
sVidScaleTemps  VidScaleTemps; /* 098 : 02 : 35 */
sImageParams    ImageParams; /* 102 : 01 : 36 */

UWord16         ReserveG[4]; /* 104 : 04 : 40 */
} sGlobalCfgData;
```

Serial Interface Developer's Reference Manual

```

/*****
/***** GLOBAL ACCESS STRUCTURES *****/
/*****
/* Camera Status Code Structure */
typedef struct
{
    UWord16      ProcessCode;      /* Process Code Value */
    UWord16      ProgressCode;     /* Progress Code Value */
    UWord16      HostStatusCode;   /* Calibration Status Value */
    UWord16      HostData;        /* Host Data Value */
} sCameraStatus; /* Size = 4 Words */

/* Camera Error Code Structure */
typedef struct
{
    UWord16      ErrorCode;        /* Error Code */
    UWord16      ErrorSubCode;     /* Error Code */
    UWord16      ErrorCount;      /* Additional errors after 1st */
    UWord16      ErrorData[5];    /* Error Data Array */
} sCameraErrors; /* Size = 8 Words */

/* Camera Alarm State Structure */
typedef struct
{
    unsigned     otWarning:1;     /* Overtemp Warning Alarm State */
    unsigned     otWarningAck:1;  /* Overtemp Warning Alarm Acknowledge */

    unsigned     battLevel:1;     /* Battery Level (Power) Alarm State */
    unsigned     battLevelAck:1;  /* Battery Level (Power) Alarm Acknowledge */

    unsigned     Reserved:10;     /* Reserved */

    unsigned     standByTemp:1;   /* Cooler/TEC Standby mode (due to internal temp) */
    unsigned     standByUser:1;   /* Cooler/TEC Standby mode (user enabled) */
} sAlarmState;

/* Operational Mode and NUC Mode Settings Structure */
typedef struct
{
    unsigned     baseOpMode:3;     /* Base OpMode (Bits 0 - 2) */
    unsigned     limitOpMode:3;   /* Limit OpMode (Bits 3 - 5) */

    unsigned     baseNuc:4;       /* Base NUC table (Bits 6 - 9) */
    unsigned     limitNuc:4;      /* Limit NUC table (Bits 10 - 13) */

    unsigned     empty:2;        /* (Bits 14 - 15) */
} sModeRanges;

/* FPA & Lens Type Settings Structure */
typedef struct
{
    unsigned     type:4;          /* FPA Type Identifier (Bits 0 - 3) */
    unsigned     subType:4;       /* FPA Sub-Type Identifier (Bits 4 - 7) */

    unsigned     lensID:6;        /* Lens ID from Descriptor Table (Bits 8 - 13) */
    unsigned     empty:2;        /* (Bits 14 - 15) */
} sFpaLens;

/* Miscellaneous Camera State Settings Structure */
typedef struct
{
    unsigned     fanEnable:1;     /* Fan Speed Index (Bit 0) */

```

Serial Interface Developer's Reference Manual

```

    unsigned    calInProgress:1;    /* Front End ADC Index Update Disable (Bit 1) */
    unsigned    cipDacUpdate:1;    /* Set if DAC0 updated from SPI by CalInProgress (cip) */

    unsigned    retRangeEn:1;      /* Ranging Reticle Enable (Bit 3) */
    unsigned    noRefresh:1;       /* Lockout NUC Refresh (Bit 4) */
    unsigned    lockMemMirror:1;   /* Lockout Memory Mirror (Bit 5) */

    unsigned    prevDAC0:8;        /* Previous DAC 0 setting (Bits 6 - 13) */
    unsigned    empty:2;           /* (Bits 14 - 15) */
} sMiscState;

/* Cal-flag Settings Structure */
typedef struct
{
    unsigned    coldRef:8;         /* Cold Ref Setting */
    unsigned    hotRef:8;         /* Hot Ref Setting */
} sCalFlagRefs;

/* Structure to hold time info from Real Time Clock */
typedef struct
{
    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 */
} sRTCData; /* Size = 7 Words */

typedef struct
{
    UWord16     rows;            /* Rows (Vertical) */
    UWord16     cols;           /* Columns (Horizontal) */
} sImage;

/*
    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 (@ 0x0040) so that the info
    is always available via PCMaster.
*/

typedef struct
{
    /* NVM Config Data for Global Access */
    sGlobalCfgData    nvMData;    /* 40 Words */

    /* NVM Data Update Address */
    bool              updateNVM;  /* 1 Word */

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

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

    /* Camera Error Codes */

```

Serial Interface Developer's Reference Manual

```

sCameraErrors    camErrors;                /* 8 Words */

/* Current camera time */
sRTCData         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 */
sImage          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 */

/* Current Mode Range Info */
sModeRanges     modeRange;                /* 1 Word */

/* Radiometric Software Data */
sRadSWInfo      radSWInfo;               /* 1 Word */

/* Camera alarm state info data */
sAlarmState     alarm;                   /* 1 Word */

/* Cal-flag Settings data */
sCalFlagRefs    calFlagRefs;            /* 1 Word */

/* FPA Type Settings Structure */
sFpaLens        fpaLens;                 /* 1 Word */

/* ADC Channel A Register Values (A0, A1, A2, A3) */
UWord16         adcAFiltered[4];        /* 4 Words */

/* ADC Channel B Register Values (B0, B1, B2, B3) */
UWord16         adcBFiltered[4];        /* 4 Words */

/* Button Panel Status */
sButtonPanel    btnPanel;                /* 2 Words */

/* Miscellaneous Camera State Settings Structure */
sMiscState      miscState;               /* 1 Word */

/* Current ATC LUT Index based on Front-end ADC Temp */
UWord16         currAtcIndex;            /* 1 Word */
} sConfigData; /* Size = 96 Words, 128 Available in Linker File */

```

Appendix B – FPA Processor FPGA Register Structure

```

/* Xilinx FPGA Register Map Structure */
struct _cam_Fpga_IO
{
    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      AnalogOffAdjReg;      /* Map to 0x4X06 */
    UWord16      AmbientTCompReg;      /* 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      FpaSizeFctrReg;       /* Map to 0x4X30 */
    UWord16      Reserved2[15];        /* Map to 0x4X31 - 0x4X3F */

    UWord16      FpaHorRoiStartReg;    /* Map to 0x4X40 */
    UWord16      FpaHorRoiStopReg;     /* Map to 0x4X41 */
    UWord16      FpaVerRoiStartReg;    /* Map to 0x4X42 */
    UWord16      FpaVerRoiStopReg;     /* Map to 0x4X43 */
    UWord16      Reserved3[12];        /* Map to 0x4X44 - 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 */
}

```

Serial Interface Developer's Reference Manual

```

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 */
};

```

Appendix C - Camera Command Enumerations

```

/* PC Master Command Code Enumerations */
enum eCCICCommandCodes
{
    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_UTILTY_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,                 /* CURRENTLY DISABLED */
    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,          /* Detect defective pixels */
    CMD_IRCON_LOAD_LUT,             /* CURRENTLY DISABLED */
    CMD_LOAD_RAD_PARAMS,            /* CURRENTLY DISABLED */
    CMD_RESET_PFV_COUNT,            /* Toggles PFV Frame count bit */
    CMD_CLEAR_CONTINUE_FLAG,        /* Clears the idle while fault continue flag */
    CMD_UNIFORMITY_TEST,            /* Cmd to initiate a uniformity (flat field noise) test */
    CMD_WRITE_UTILTY_MEMORY,        /* Write a block of utility memory (Address Supplied) */
    CMD_ADV_DETECT_BAD_PIXELS,      /* CURRENTLY DISABLED */
    CMD_UPLOAD_NUC,                 /* Command to initiate flash write of uploaded NUC terms */
    CMD_DOWNLOAD_NUC,               /* Command to initiate flash read of active NUC terms */
    CMD_COMPILE_DEFECT_LISTS,       /* CURRENTLY DISABLED */
    CMD_RESTORE_FACTORY_DEFECTS,    /* CURRENTLY DISABLED */
    CMD_ENABLE_RANGE_RETICLE,       /* Turn Ranging Reticle On/Off */
    CMD_INIT_NUC_TABLE,             /* CURRENTLY DISABLED */
    CMD_CAMERA_RECOVER,             /* Attempt to recover camera from internal errors */
    CMD_PROG_DSP_DATA_FLASH,        /* Program DSP Data FLASH at provided address */
    CMD_UPDATE_EXP_PORT,            /* Updates Expansion Port Register from CfgData Value */
    CMD_UPDATE_MODE_INFO,           /* Mode info in flash has changed, re-init operating data */
    CMD_RESTORE_SPI_NVM,            /* Restore NVM from stored table in SPI Flash */

    CMD_UNDEFINED
};

```

Appendix D – Serial Data FLASH Page Defines

```
#define LENS_DESCRIPTOR_BASE_PAGE      0      /* 1 Table - 1 Page */
#define FACTORY_NVM_BASE_PAGE          1      /* 1 Table - 1 Page */
                                           /* Pages 2 - 7 Reserved */
#define OP_MODE_TABLE_BASE_PAGE       8      /* 8 Tables - 8 Pages */
#define NUC_MODE_TABLE_BASE_PAGE      16     /* 12 Tables - 12 Pages */
                                           /* Pages 28 - 31 Reserved */
#define DAC0_LUT_BASE_PAGE            32     /* 12 Tables - 12 Pages */
                                           /* Pages 44 - 47 Reserved */
#define ANALOG_OFF_ADJ_BASE_PAGE      48     /* 12 Tables - 24 Pages */
                                           /* Pages 72 - 190 Reserved */
#define OVLY_PAL_TABLE_BASE_PAGE      191   /* 8 Tables - 1 Page */
#define CLR_PAL_Y_TABLE_BASE_PAGE     192   /* 16 Tables - 32 Pages */
#define CLR_PAL_CX_TABLE_BASE_PAGE    224   /* 16 Tables - 32 Pages */
                                           /* Pages 256 - 511 Reserved */
#define FPA_PROC_FPGA_CFG_BASE_PAGE   512
#define EXP_FPGA_CFG_BASE_PAGE        1146
#define ATC_LUT_TABLE_BASE_PAGE       1536 /* 12 Tables - 384 Pages */
                                           /* Pages 1920 - 4095 Reserved */
```

Appendix E - Mapping of Serial Non-Volatile Memory

```

/* Complete Camera Dynamic Configuration Info Structure */
struct _nvm_DynCfg
{
    UWord16      CamMode;          /* Cast to/from nvm_CamMode */
    UWord16      FpaMode;          /* Cast to/from nvm_FpaProcMode */

    UWord16      LensMode;         /* Cast to/from nvm_LensMode */

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

    UWord16      ReserveB;         /* Reserved */

    nvm_AutoNucMode 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_ActPal */
    UWord16      OvlMode;          /* Cast to/from nvm_OvlMode */

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

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

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

    UWord16      ReserveE;         /* Reserved */

    nvm_ManualITT ManualITT;       /* See Struct Above */
    nvm_AgcLimits AgcLimits;       /* See Struct Above */

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

    UWord16      AgcBinLimit;      /* AGC Bin Limit Value */

    UWord16      ReserveF[5];      /* Reserved */

    nvm_VidScaleTemps VidScaleTemps; /* See Struct Above */

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

    UWord16      ReserveG[4];      /* 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 */
};

```

Appendix F - 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-erved	Zn Stat En	Rfsh On Sw	Rfsh On Boot	Auto Rfsh Tmp	Auto Rfsh Tim	Auto Nuc Sw	Reserved			Pfv Out En	SVid Out En	CVid1 Out En	CVid0 Out En
34	FPA Processor User Mode Control	Short R/W	0x0000	Reserved				PFV Src Sel	Res-erved	Clr Pt YSel	Clr Bar En	Dspl Vid Pol	Reserved			Mstr Sync Mod			
36	Lens Index Control	Short R/W	0x0000	Reserved								Lens Auto Det	Res-erved			Lens Indx			
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)
- 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;

Serial Interface Developer's Reference Manual

	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.

Lens Index Control:

Lens Index - Selects the active lens descriptor table (0 <= LensIndx <= 3).

Lens Auto Detect	0 - Use dynamic configuration table LensIndx;
	1 - Automatically detect lens ID using ADCA channel 4 to determine LensIndx via lens descriptor table

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.

Serial Interface Developer's Reference Manual

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.

Serial Interface Developer's Reference Manual

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.

Serial Interface Developer's Reference Manual

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 - 96	Reserved	Short R/W	0x0000	Reserved															
98	Display Video Zero Scale Temperature	Short R/W	0x0000	Display Video Zero Scale Temperature (°K - LSB is TBD)															
100	Display Video Full Scale Temperature	Short R/W	0x0000	Display Video Full Scale Temperature (°K - LSB is TBD)															
102	Camera Background Temperature	Byte R/W	0x32													Camera Background Temperature (°C - 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)

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.

Serial Interface Developer's Reference Manual

Adr	Entry Name	Mode	Factory Initialize	Bit Fields															
				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
104-110	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 55°C.

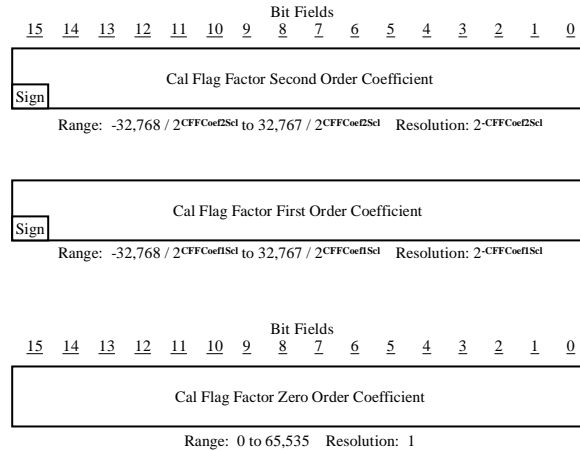
Power On/Reset Cycle Count - Incremented Each Time the Camera is Booted.

Appendix G – NUC Coefficient Format

NUC Table Base

+20
↑
+14
+13
↑
+10.5

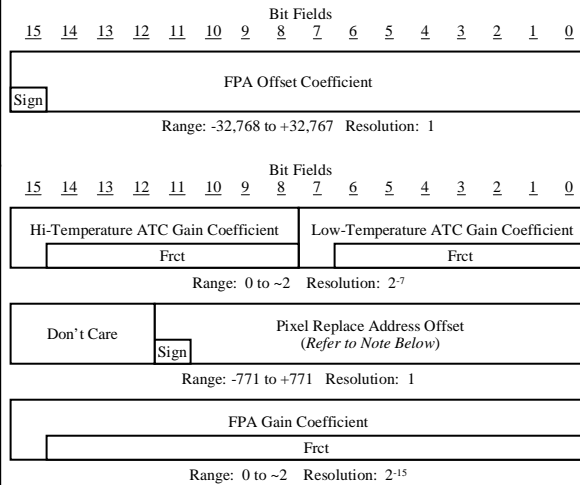
<p><u>Cal Flag Factor First and Second Order Coefficients</u></p> <p>Memory Required = $384 * 288 * 2 = 221,184$ Words = 7 Blocks</p> <p>Flash Addr = (Pixel Number * 2) + NUC Starting Addr + 458,752 First Order Term on Even Addr Second Order Term on Subsequent Odd Addr</p> <p>On Defective Pixel, Set Cal Flag Factor First and Second Order Coefficients to 0</p>
<p><u>Cal Flag Factor Zero Order Coefficients</u></p> <p>Memory Required = $384 * 288 = 110,592$ Words = 3.5 Blocks</p> <p>Flash Addr = Pixel Number + NUC Starting Addr + 344,064</p> <p>On Defective Pixel, Set Cal Flag Factor Zero Order Coefficient to 0</p>



NUC Table Base

+10.5
↑
+7
+6
↑
Start

<p><u>FPA Offset Coefficients</u></p> <p>Memory Required = $384 * 288 = 110,592$ Words = 3.5 Blocks</p> <p>Flash Addr = FPA Pixel Number + NUC Starting Addr + 229,376</p> <p>On Defective Pixel, Set FPA Offset to 0</p>
<p><u>ATC Gain / FPA Gain Coefficients</u></p> <p>Memory Required = $384 * 288 * 2 = 221,184$ Words = 7 Blocks</p> <p>Flash Addr = (Pixel Number * 2) + NUC Starting Addr ATC Gain on Even Addr / FPA Gain on Subsequent Odd Addr</p> <p>On Defective Pixel, Set FPA Gain to 0 and Set ATC Gain to Pixel Replace Address Offset</p>



Note: Pixel Replace Address Offset = (Replace Relative Row Index * Pixels / Line) + Replace Relative Col Index
where: $-2 \leq$ Replace Relative Row Index \leq +2 and $-3 \leq$ Replace Relative Col Index \leq +3 and Pixels / Line \leq 384