LLNL Nanosecond Gated Camera 2.1.1

Generated by Doxygen 1.8.17

1 CODE_OF_CONDUCT	1
2 CONTRIBUTING	3
3 Namespace Index	5
3.1 Packages	5
4 Hierarchical Index	7
4.1 Class Hierarchy	7
5 Class Index	9
5.1 Class List	9
6 File Index	11
6.1 File List	11
7 Namespace Documentation	13
7.1 nsCamera Namespace Reference	13
7.1.1 Variable Documentation	13
7.1.1.1all	13
7.2 nsCamera.boards Namespace Reference	13
7.2.1 Variable Documentation	14
7.2.1.1all	14
7.3 nsCamera.boards.LLNL_v1 Namespace Reference	14
7.4 nsCamera.boards.LLNL_v4 Namespace Reference	14
7.5 nsCamera.CameraAssembler Namespace Reference	14
7.6 nsCamera.comms Namespace Reference	14
7.6.1 Variable Documentation	15
7.6.1.1all	15
7.7 nsCamera.comms.GigE Namespace Reference	
7.8 nsCamera.comms.RS422 Namespace Reference	15
7.9 nsCamera.sensors Namespace Reference	15
7.9.1 Variable Documentation	16
7.9.1.1all	16
7.10 nsCamera.sensors.daedalus Namespace Reference	16
7.11 nsCamera.sensors.icarus Namespace Reference	16
7.12 nsCamera.sensors.icarus2 Namespace Reference	16
7.13 nsCamera.utils Namespace Reference	16
7.13.1 Variable Documentation	17
7.13.1.1all	17
7.14 nsCamera.utils.crc16pure Namespace Reference	17

7.14.1 Detailed Description
7.14.2 Function Documentation
7.14.2.1 _crc16()
7.14.2.2 crc16xmodem()
7.14.3 Variable Documentation
7.14.3.1 CRC16_XMODEM_TABLE
7.15 nsCamera.utils.FlatField Namespace Reference
7.15.1 Function Documentation
7.15.1.1 generateFF()
7.15.1.2 getFilenames()
7.15.1.3 getROlvector()
7.15.1.4 removeFF()
7.15.1.5 removeFFall()
7.15.1.6 tslopes()
7.15.2 Variable Documentation
7.15.2.1 action
7.15.2.2 args
7.15.2.3 default
7.15.2.4 dest
7.15.2.5 directory
7.15.2.6 framelist
7.15.2.7 help
7.15.2.8 nargs
7.15.2.9 parser
7.16 nsCamera.utils.GenTec Namespace Reference
7.16.1 Variable Documentation
7.16.1.1 gt
7.17 nsCamera.utils.Ophir Namespace Reference
7.18 nsCamera.utils.Packet Namespace Reference
7.19 nsCamera.utils.Subregister Namespace Reference
7.20 nsCameraExample Namespace Reference
7.20.1 Variable Documentation
7.20.1.1 BOARD
7.20.1.2 ca
7.20.1.3 COMM
7.20.1.4 SENSOR
7.20.1.5 timing
7.20.1.6 tune
7.21 testSuite Namespace Reference

7.21.1 Function Documentation	 27
7.21.1.1 testSuite()	 27
7.21.2 Variable Documentation	 36
7.21.2.1 action	 36
7.21.2.2 args	 36
7.21.2.3 board	 36
7.21.2.4 comm	 37
7.21.2.5 default	 37
7.21.2.6 dest	 37
7.21.2.7 help	 37
7.21.2.8 interactive	 37
7.21.2.9 ipAdd	 37
7.21.2.10 None	 38
7.21.2.11 parser	 38
7.21.2.12 portNum	 38
7.21.2.13 sensor	 38
7.21.2.14 swtrigger	 38
8 Class Documentation	39
8.1 nsCameraAssembler.CameraAssembler Class Reference	
8.1.1 Detailed Description	
8.1.2 Constructor & Destructor Documentation	
8.1.2.1init()	
8.1.3 Member Function Documentation	
8.1.3.1 abortReadoff()	
8.1.3.2 arm()	
8.1.3.3 batchAcquire()	
8.1.3.4 bytes2str()	
8.1.3.5 checkCRC()	
8.1.3.6 checkRegSet()	
8.1.3.7 checkSensorVoltStat()	
8.1.3.8 checkStatus()	 51
8.1.3.9 checkStatus2()	
8.1.3.10 clearStatus()	 51
8.1.3.10 clearStatus()	 51 51
8.1.3.10 clearStatus()	 51 51 52
8.1.3.10 clearStatus()	 51 51 52 52
8.1.3.10 clearStatus()	 51 51 52 52

8.1.3.16 dumpNumpy()
8.1.3.17 dumpRegisters()
8.1.3.18 dumpStatus()
8.1.3.19 dumpSubregisters()
8.1.3.20 enableLED()
8.1.3.21 flatten()
8.1.3.22 generateFrames()
8.1.3.23 getBoardInfo()
8.1.3.24 getEnter()
8.1.3.25 getManualTiming()
8.1.3.26 getMonV()
8.1.3.27 getPot()
8.1.3.28 getPotV()
8.1.3.29 getPressure()
8.1.3.30 getRegister()
8.1.3.31 getSubregister()
8.1.3.32 getTemp()
8.1.3.33 getTimer()
8.1.3.34 getTiming()
8.1.3.35 initBoard()
8.1.3.36 initialize()
8.1.3.37 initPots()
8.1.3.38 initPowerCheck()
8.1.3.39 initSensor()
8.1.3.40 latchPots()
8.1.3.41 loadTextFrames()
8.1.3.42 mmReadoff()
8.1.3.43 parseReadoff()
8.1.3.44 plotFrames()
8.1.3.45 powerCheck()
8.1.3.46 printBoardInfo()
8.1.3.47 readImgs()
8.1.3.48 readoff()
8.1.3.49 readSerial()
8.1.3.50 readSRAM()
8.1.3.51 reboot()
8.1.3.52 reinitialize()
8.1.3.53 reportEdgeDetects()
8.1.3.54 reportStatus()

	8.1.3.55 resetTimer()	75
	8.1.3.56 resolveSubreg()	75
	8.1.3.57 saveFrames()	76
	8.1.3.58 saveNumpys()	77
	8.1.3.59 saveTiffs()	78
	8.1.3.60 sendCMD()	79
	8.1.3.61 sensorSpecific()	79
	8.1.3.62 setArbTiming()	80
	8.1.3.63 setFrames()	80
	8.1.3.64 setHighFullWell()	81
	8.1.3.65 setInterlacing()	81
	8.1.3.66 setLED()	82
	8.1.3.67 setManualShutters()	82
	8.1.3.68 setPot()	82
	8.1.3.69 setPotV()	83
	8.1.3.70 setPowerSave()	86
	8.1.3.71 setPPER()	86
	8.1.3.72 setRegister()	87
	8.1.3.73 setRows()	87
	8.1.3.74 setSubregister()	89
	8.1.3.75 setTiming()	90
	8.1.3.76 setTriggerDelay()	90
	8.1.3.77 setZeroDeadTime()	90
	8.1.3.78 startCapture()	91
	8.1.3.79 str2bytes()	91
	8.1.3.80 str2nparray()	91
	8.1.3.81 submitMessages()	92
	8.1.3.82 waitForSRAM()	93
	8.1.3.83 writeSerial()	93
8.1.4 N	lember Data Documentation	93
	8.1.4.1 abort	93
	8.1.4.2 armed	93
	8.1.4.3 board	94
	8.1.4.4 boardname	94
	8.1.4.5 commname	94
	8.1.4.6 comms	94
	8.1.4.7 currtime	94
	8.1.4.8 cycle	
	8.1.4.9 FPGAboardtype	95

8.1.4.10 FPGAinterfaces)5
8.1.4.11 FPGAinvalid)5
8.1.4.12 FPGANum)5
8.1.4.13 FPGArad)5
8.1.4.14 FPGAsensor)5
8.1.4.15 FPGAVersion	}6
8.1.4.16 inittime	}6
8.1.4.17 iplist	}6
8.1.4.18 logcrit	
8.1.4.19 logcritbase	}6
8.1.4.20 logdebug	}6
8.1.4.21 logdebugbase	}7
8.1.4.22 logerr	}7
8.1.4.23 logerrbase	}7
8.1.4.24 loginfo)7
8.1.4.25 loginfobase	}7
8.1.4.26 logtag	
8.1.4.27 logwarn)8
8.1.4.28 logwarnbase)8
8.1.4.29 oldtime)8
8.1.4.30 packageroot)8
8.1.4.31 padToFull)8
8.1.4.32 parsedtime	
8.1.4.33 payloaderror)9
8.1.4.34 platform)9
8.1.4.35 port)9
8.1.4.36 PY3)9
8.1.4.37 read)9
8.1.4.38 savetime)9
8.1.4.39 sensmanual)0
8.1.4.40 sensor)0
8.1.4.41 sensorname)0
8.1.4.42 senstiming)0
8.1.4.43 trigtime)0
8.1.4.44 unstringed)1
8.1.4.45 verblevel	
8.1.4.46 verbmap)1
8.1.4.47 verbose)1
8.1.4.48 version)1

8.1.4.49 waited	01
8.2 nsCamera.sensors.daedalus.daedalus Class Reference	02
8.2.1 Detailed Description	02
8.2.2 Constructor & Destructor Documentation	03
8.2.2.1init()	03
8.2.3 Member Function Documentation	05
8.2.3.1 checkSensorVoltStat()	05
8.2.3.2 getManualTiming()	06
8.2.3.3 getTiming()	06
8.2.3.4 parseReadoff()	08
8.2.3.5 reportStatusSensor()	09
8.2.3.6 sensorSpecific()	09
8.2.3.7 setArbTiming()	11
8.2.3.8 setHighFullWell()	12
8.2.3.9 setInterlacing()	13
8.2.3.10 setManualShutters()	14
8.2.3.11 setTiming()	15
8.2.3.12 setTriggerDelay()	17
8.2.3.13 setZeroDeadTime()	17
8.2.4 Member Data Documentation	18
8.2.4.1 bytesperpixel	18
8.2.4.2 ca	18
8.2.4.3 firstframe	19
8.2.4.4 firstrow	19
8.2.4.5 fpganumID	19
8.2.4.6 height	19
8.2.4.7 HFW	19
8.2.4.8 interlacing	19
8.2.4.9 lastframe	20
8.2.4.10 lastrow	20
8.2.4.11 logcrit	20
8.2.4.12 logdebug	20
8.2.4.13 logerr	20
8.2.4.14 loginfo	20
8.2.4.15 logwarn	21
8.2.4.16 maxframe	21
8.2.4.17 maxheight	21
8.2.4.18 maxwidth	21
8.2.4.19 minframe	21

8.2.4.20 nframes
8.2.4.21 sens_registers
8.2.4.22 sens_subregisters
8.2.4.23 width
8.2.4.24 ZDT
8.3 nsCamera.utils.GenTec.GenTec Class Reference
8.3.1 Detailed Description
8.3.2 Constructor & Destructor Documentation
8.3.2.1init()
8.3.3 Member Function Documentation
8.3.3.1 closeDevice()
8.3.3.2 GenTecReadTest()
8.3.3.3 ready()
8.3.3.4 sendSerial()
8.3.4 Member Data Documentation
8.3.4.1 serial
8.4 nsCamera.comms.GigE.GigE Class Reference
8.4.1 Detailed Description
8.4.2 Constructor & Destructor Documentation
8.4.2.1init()
8.4.3 Member Function Documentation
8.4.3.1 arm()
8.4.3.2 closeDevice()
8.4.3.3 getCardInfo()
8.4.3.4 getCardIP()
8.4.3.5 openDevice()
8.4.3.6 readoff()
8.4.3.7 readSerial()
8.4.3.8 sendCMD()
8.4.3.9 writeSerial()
8.4.4 Member Data Documentation
8.4.4.1 _zest
8.4.4.2 ca
8.4.4.3 CardInfo
8.4.4.4 CardInfoP
8.4.4.5 closecard
8.4.4.6 Connection
8.4.4.7 dport
8.4.4.8 logcrit

8.4.4.9 logdebug	. 136
8.4.4.10 logerr	. 136
8.4.4.11 loginfo	. 137
8.4.4.12 logwarn	
8.4.4.13 mode	. 137
8.4.4.14 payloadsize	. 137
8.4.4.15 readTimeout	. 137
8.4.4.16 skipError	. 137
8.4.4.17 writeTimeout	. 138
8.4.4.18 ZCountCards	. 138
8.4.4.19 ZOpenConnection	. 138
8.4.4.20 ZReadData	. 138
8.4.4.21 ZWriteData	. 138
8.5 nsCamera.sensors.icarus.icarus Class Reference	. 139
8.5.1 Detailed Description	. 139
8.5.2 Constructor & Destructor Documentation	. 140
8.5.2.1init()	. 140
8.5.3 Member Function Documentation	. 141
8.5.3.1 checkSensorVoltStat()	. 141
8.5.3.2 getManualTiming()	. 142
8.5.3.3 getTiming()	. 142
8.5.3.4 parseReadoff()	. 143
8.5.3.5 reportStatusSensor()	. 144
8.5.3.6 sensorSpecific()	. 144
8.5.3.7 setArbTiming()	. 145
8.5.3.8 setHighFullWell()	. 147
8.5.3.9 setInterlacing()	. 147
8.5.3.10 setManualShutters()	. 148
8.5.3.11 setTiming()	. 149
8.5.3.12 setTriggerDelay()	. 151
8.5.3.13 setZeroDeadTime()	. 151
8.5.4 Member Data Documentation	. 152
8.5.4.1 bytesperpixel	. 152
8.5.4.2 ca	. 152
8.5.4.3 firstframe	. 152
8.5.4.4 firstrow	
8.5.4.5 fpganumID	
8.5.4.6 height	
8.5.4.7 icarustype	. 153

8.5.4.8 interlacing	153
8.5.4.9 lastframe	153
8.5.4.10 lastrow	153
8.5.4.11 logcrit	153
8.5.4.12 logdebug	154
8.5.4.13 logerr	154
8.5.4.14 loginfo	154
8.5.4.15 logwarn	154
8.5.4.16 maxframe	154
8.5.4.17 maxheight	154
8.5.4.18 maxwidth	155
8.5.4.19 minframe	155
8.5.4.20 nframes	155
8.5.4.21 sens_registers	155
8.5.4.22 sens_subregisters	155
8.5.4.23 width	155
8.6 nsCamera.sensors.icarus2.icarus2 Class Reference	156
8.6.1 Detailed Description	156
8.6.2 Constructor & Destructor Documentation	157
8.6.2.1init()	157
8.6.3 Member Function Documentation	158
8.6.3.1 checkSensorVoltStat()	158
8.6.3.2 getManualTiming()	158
8.6.3.3 getTiming()	159
8.6.3.4 parseReadoff()	160
8.6.3.5 reportStatusSensor()	161
8.6.3.6 sensorSpecific()	161
8.6.3.7 setArbTiming()	162
8.6.3.8 setHighFullWell()	163
8.6.3.9 setInterlacing()	164
8.6.3.10 setManualShutters()	164
8.6.3.11 setTiming()	165
8.6.3.12 setTriggerDelay()	167
8.6.3.13 setZeroDeadTime()	168
8.6.4 Member Data Documentation	168
8.6.4.1 bytesperpixel	168
8.6.4.2 ca	168
8.6.4.3 firstframe	168
8.6.4.4 firstrow	169

	8.6.4.5 fpganumID
	8.6.4.6 height
	8.6.4.7 icarustype
	8.6.4.8 interlacing
	8.6.4.9 lastframe
	8.6.4.10 lastrow
	8.6.4.11 logcrit
	8.6.4.12 logdebug
	8.6.4.13 logerr
	8.6.4.14 loginfo
	8.6.4.15 logwarn
	8.6.4.16 maxframe
	8.6.4.17 maxheight
	8.6.4.18 maxwidth
	8.6.4.19 minframe
	8.6.4.20 nframes
	8.6.4.21 sens_registers
	8.6.4.22 sens_subregisters
	8.6.4.23 width
	a.boards.LLNL_v1.llnl_v1 Class Reference
8.7.1 D	etailed Description
8.7.2 C	onstructor & Destructor Documentation
	8.7.2.1init()
8.7.3 M	ember Function Documentation
	8.7.3.1 checkStatus()
	8.7.3.2 checkStatus2()
	8.7.3.3 clearStatus()
	8.7.3.4 configADCs()
	8.7.3.5 disarm()
	8.7.3.6 dumpStatus()
	8.7.3.7 enableLED()
	8.7.3.8 getPressure()
	8.7.3.9 getTemp()
	8.7.3.10 getTimer()
	8.7.3.11 initBoard()
	8.7.3.12 initPots()
	8.7.3.13 initSensor()
	8.7.3.14 latchPots()
	8.7.3.15 readSRAM()

8.7.3.16 reportEdgeDetects()
8.7.3.17 reportStatus()
8.7.3.18 resetTimer()
8.7.3.19 setLED()
8.7.3.20 setPowerSave()
8.7.3.21 setPPER()
8.7.3.22 softReboot()
8.7.3.23 startCapture()
8.7.3.24 waitForSRAM()
8.7.4 Member Data Documentation
8.7.4.1 ADC5_bipolar
8.7.4.2 ADC5_mult
8.7.4.3 ca
8.7.4.4 dummySensorVals
8.7.4.5 logcrit
8.7.4.6 logdebug
8.7.4.7 logerr
8.7.4.8 loginfo
8.7.4.9 logwarn
8.7.4.10 monitor_controls
8.7.4.11 registers
8.7.4.12 rs422_baud
8.7.4.13 rs422_cmd_wait
8.7.4.14 subreg_aliases
8.7.4.15 subregisters
8.7.4.16 subreglist
8.7.4.17 VREF
8.8 nsCamera.boards.LLNL_v4.llnl_v4 Class Reference
8.8.1 Detailed Description
8.8.2 Constructor & Destructor Documentation
8.8.2.1init()
8.8.3 Member Function Documentation
8.8.3.1 checkStatus()
8.8.3.2 checkStatus2()
8.8.3.3 clearStatus()
8.8.3.4 configADCs()
8.8.3.5 disarm()
8.8.3.6 dumpStatus()
8.8.3.7 enableLED()
8.8.3.6 dumpStatus()

8.8.3.8 getPressure()	201
8.8.3.9 getTemp()	202
8.8.3.10 getTimer()	202
8.8.3.11 initBoard()	203
8.8.3.12 initPots()	203
8.8.3.13 initSensor()	204
8.8.3.14 latchPots()	204
8.8.3.15 readSRAM()	205
8.8.3.16 reportEdgeDetects()	205
8.8.3.17 reportStatus()	206
8.8.3.18 resetTimer()	207
8.8.3.19 setLED()	207
8.8.3.20 setPowerSave()	208
8.8.3.21 setPPER()	208
8.8.3.22 softReboot()	209
8.8.3.23 startCapture()	209
8.8.3.24 waitForSRAM()	210
8.8.4 Member Data Documentation	211
8.8.4.1 ADC5_bipolar	211
8.8.4.2 ADC5_mult	211
8.8.4.3 ca	212
8.8.4.4 defoff	212
8.8.4.5 defsens	212
8.8.4.6 dummySensorVals	212
8.8.4.7 logcrit	212
8.8.4.8 logdebug	212
8.8.4.9 logerr	213
8.8.4.10 loginfo	213
8.8.4.11 logwarn	213
8.8.4.12 monitor_controls	213
8.8.4.13 registers	213
8.8.4.14 rs422_baud	213
8.8.4.15 rs422_cmd_wait	214
8.8.4.16 subreg_aliases	214
8.8.4.17 subregisters	214
8.8.4.18 subreglist	214
8.8.4.19 VREF	214
nsCamera.utils.Ophir.Ophir Class Reference	215
8.9.1 Detailed Description	215

8.9

	3.9.2 Constructor & Destructor Documentation
	8.9.2.1init()
	3.9.3 Member Function Documentation
	8.9.3.1 closeOphir()
	8.9.3.2 OphirTest()
	3.9.4 Member Data Documentation
	8.9.4.1 COM
	8.9.4.2 DevHan
	8.9.4.3 DHandle
8.10 r	sCamera.utils.Packet.Packet Class Reference
	3.10.1 Detailed Description
	3.10.2 Constructor & Destructor Documentation
	8.10.2.1init()
	3.10.3 Member Function Documentation
	8.10.3.1 bytes2str()
	8.10.3.2 calculateCRC()
	8.10.3.3 checkCRC()
	8.10.3.4 checkReadPacket()
	8.10.3.5 checkResponsePacket()
	8.10.3.6 checkResponseString()
	8.10.3.7 pktStr()
	8.10.3.8 str2bytes()
	3.10.4 Member Data Documentation
	8.10.4.1 _addr
	8.10.4.2 _cmd
	8.10.4.3 _crc
	8.10.4.4 _data
	8.10.4.5 _payload
	8.10.4.6 _payload_length
	8.10.4.7 _preamble
	8.10.4.8 _seqID
	8.10.4.9 _type
	8.10.4.10 PY3
8.11 r	sCamera.comms.RS422.RS422 Class Reference
	3.11.1 Detailed Description
	3.11.2 Constructor & Destructor Documentation
	8.11.2.1init()
	3.11.3 Member Function Documentation
	8.11.3.1 arm()

	8.11.3.2 closeDevice()	0
	8.11.3.3 readoff()	1
	8.11.3.4 readSerial()	2
	8.11.3.5 sendCMD()	2
	8.11.3.6 serialClose()	4
	8.11.3.7 writeSerial()	5
8	11.4 Member Data Documentation	5
	8.11.4.1 _baud	5
	8.11.4.2 _datatimeout	6
	8.11.4.3 _par	6
	8.11.4.4 _port	6
	8.11.4.5 _read_timeout	6
	8.11.4.6 _ser	6
	8.11.4.7 _stop	6
	8.11.4.8 _write_timeout	7
	8.11.4.9 ca	7
	8.11.4.10 logcrit	7
	8.11.4.11 logdebug	7
	8.11.4.12 logerr	7
	8.11.4.13 loginfo	7
	8.11.4.14 logwarn	8
	8.11.4.15 mode	8
	8.11.4.16 payloadsize	8
	8.11.4.17 PY3	8
	8.11.4.18 skipError	8
8.12 ns	Camera.utils.Subregister.SubRegister Class Reference	9
8	12.1 Detailed Description	9
8	12.2 Constructor & Destructor Documentation	9
	8.12.2.1init()	0
8	12.3 Member Data Documentation	0
	8.12.3.1 addr	0
	8.12.3.2 max	1
	8.12.3.3 max_value	1
	8.12.3.4 maxV	1
	8.12.3.5 min	1
	8.12.3.6 minV	1
	8.12.3.7 name	1
	8.12.3.8 register	2
	8.12.3.9 resolution	2

Index

8.12.3.10 start_bit	242
8.12.3.11 value	242
8.12.3.12 width	242
8.12.3.13 writable	242
8.13 nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO Class Reference	243
8.13.1 Detailed Description	243
8.13.2 Member Data Documentation	243
8.13.2.1 _fields	243
8.13.2.2 ubyte4	244
8.13.2.3 ubyte6	244
9 File Documentation	245
9.1 nsCamera/initpy File Reference	245
9.2 nsCamera/boards/initpy File Reference	245
9.3 nsCamera/comms/initpy File Reference	245
9.4 nsCamera/sensors/initpy File Reference	246
9.5 nsCamera/utils/initpy File Reference	246
9.6 nsCamera/boards/LLNL_v1.py File Reference	246
9.7 nsCamera/boards/LLNL_v4.py File Reference	246
9.8 nsCamera/CameraAssembler.py File Reference	247
9.9 nsCamera/CODE_OF_CONDUCT.md File Reference	247
9.10 nsCamera/comms/GigE.py File Reference	247
9.11 nsCamera/comms/RS422.py File Reference	247
9.12 nsCamera/CONTRIBUTING.md File Reference	248
9.13 nsCamera/docs/nsCameraExample.py File Reference	248
9.14 nsCamera/docs/testSuite.py File Reference	248
9.15 nsCamera/sensors/daedalus.py File Reference	249
9.16 nsCamera/sensors/icarus.py File Reference	249
9.17 nsCamera/sensors/icarus2.py File Reference	249
9.18 nsCamera/utils/crc16pure.py File Reference	249
9.19 nsCamera/utils/FlatField.py File Reference	250
9.20 nsCamera/utils/GenTec.py File Reference	250
9.21 nsCamera/utils/Ophir.py File Reference	251
9.22 nsCamera/utils/Packet.py File Reference	251
9.23 nsCamera/utils/Subregister.py File Reference	251

253

CODE_OF_CONDUCT

title: "Code of Conduct"

A code of conduct defines standards for how to engage in a community. Such a file signals an inclusive environment that respects all contributions. It also outlines procedures for addressing problems between members of your project's community. The file is typically named in all-caps with underscores: CODE_OF_CONDUCT.md.

Our example here is adapted from the website in the Attribution section below. It contains a few blanks _____ where you fill in your project/repo/team's name and/or email address. For more information, see GitHub's advice for adding a code of conduct to your project.

Example Community Code of Conduct

Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

Our Standards

Examples of behavior that contributes to creating a positive environment include:

- · Using welcoming and inclusive language
- · Being respectful of differing viewpoints and experiences
- · Gracefully accepting constructive criticism

2 CODE OF CONDUCT

- · Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- · The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- · Public or private harassment
- · Publishing others' private information, such as a physical or electronic address, without explicit permission
- · Other conduct which could reasonably be considered inappropriate in a professional setting

Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the NSGCC project or its community. Examples of representing the project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of the project may be further defined and clarified by NSGCC maintainers.

Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at funsten1@llnl.gov or the LLNL GitHub Admins at github-admin@llnl.gov. The project team will review and investigate all complaints, and will respond in a way that it deems appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project or organization's leadership.

Attribution

This Code of Conduct is adapted from the Contributor Covenant (version 1.4).

CONTRIBUTING

Contributing Guidelines

nsCamera is an open source project. We welcome questions, feature requests, or bug reports at <code>jerhill@llnl.</code> \leftarrow gov. We do not yet have a system in place for external contribution, but please contact us if you are interested in contributing. Please also refer to our code of conduct.

4 CONTRIBUTING

Namespace Index

3.1 Packages

Here are the packages with brief descriptions (if available):

nsCamera
nsCamera.boards
nsCamera.boards.LLNL_v1
nsCamera.boards.LLNL_v4
nsCamera.CameraAssembler
nsCamera.comms
nsCamera.comms.GigE
nsCamera.comms.RS422
nsCamera.sensors
nsCamera.sensors.daedalus
nsCamera.sensors.icarus
nsCamera.sensors.icarus2
nsCamera.utils
nsCamera.utils.crc16pure
nsCamera.utils.FlatField
nsCamera.utils.GenTec
nsCamera.utils.Ophir
nsCamera.utils.Packet
nsCamera.utils.Subregister
nsCameraExample
testSuite

6 Namespace Index

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

nsCamera.CameraAssembler.CameraAssembler
nsCamera.sensors.daedalus.daedalus
nsCamera.utils.GenTec.GenTec
nsCamera.comms.GigE.GigE
nsCamera.sensors.icarus.icarus
nsCamera.sensors.icarus2.icarus2
nsCamera.boards.LLNL_v1.llnl_v1
nsCamera.boards.LLNL_v4.llnl_v4
nsCamera.utils.Ophir.Ophir
nsCamera.utils.Packet.Packet
nsCamera.comms.RS422.RS422
Structure
nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO
nsCamera.utils.Subregister.SubRegister

8 Hierarchical Index

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

nsCamera.CameraAssembler.CameraAssembler
nsCamera.sensors.daedalus.daedalus
nsCamera.utils.GenTec.GenTec
nsCamera.comms.GigE.GigE
nsCamera.sensors.icarus.icarus
nsCamera.sensors.icarus2.icarus2
nsCamera.boards.LLNL_v1.llnl_v1
nsCamera.boards.LLNL_v4.llnl_v4
nsCamera.utils.Ophir.Ophir
nsCamera.utils.Packet.Packet
nsCamera.comms.RS422.RS422
nsCamera.utils.Subregister.SubRegister
nsCamera.comms.GigE.GigE.ZESTETM1 CARD INFO

10 Class Index

File Index

6.1 File List

Here is a list of all files with brief descriptions:

nsCamera/initpy
nsCamera/CameraAssembler.py
nsCamera/boards/initpy
nsCamera/boards/LLNL_v1.py
nsCamera/boards/LLNL_v4.py
nsCamera/comms/initpy
nsCamera/comms/GigE.py
nsCamera/comms/RS422.py
nsCamera/docs/nsCameraExample.py
nsCamera/docs/testSuite.py
nsCamera/sensors/initpy
nsCamera/sensors/daedalus.py
nsCamera/sensors/icarus.py
nsCamera/sensors/icarus2.py
nsCamera/utils/initpy
nsCamera/utils/crc16pure.py
nsCamera/utils/FlatField.py
nsCamera/utils/GenTec.py
nsCamera/utils/Ophir.py
nsCamera/utils/Packet.py
nsCamera/utils/Subregister.py

12 File Index

Namespace Documentation

7.1 nsCamera Namespace Reference

Namespaces

- boards
- CameraAssembler
- comms
- sensors
- utils

Variables

• list __all__ = ["CameraAssembler.py"]

7.1.1 Variable Documentation

```
7.1.1.1 __all__
list nsCamera.__all__ = ["CameraAssembler.py"] [private]
Definition at line 26 of file __init__.py.
```

7.2 nsCamera.boards Namespace Reference

Namespaces

- LLNL_v1
- LLNL_v4

Variables

```
• list __all__ = ["LLNL_v1", "LLNL_v4"]
```

7.2.1 Variable Documentation

```
7.2.1.1 __all__
list nsCamera.boards.__all__ = ["LLNL_v1", "LLNL_v4"] [private]
Definition at line 23 of file init .py.
```

7.3 nsCamera.boards.LLNL_v1 Namespace Reference

Classes

class llnl_v1

7.4 nsCamera.boards.LLNL_v4 Namespace Reference

Classes

class llnl_v4

7.5 nsCamera.CameraAssembler Namespace Reference

Classes

· class CameraAssembler

7.6 nsCamera.comms Namespace Reference

Namespaces

- GigE
- RS422

Variables

```
• list __all__ = ["RS422", "GigE"]
```

7.6.1 Variable Documentation

```
7.6.1.1 __all__
list nsCamera.comms.__all__ = ["RS422", "GigE"] [private]
Definition at line 25 of file __init__.py.
```

7.7 nsCamera.comms.GigE Namespace Reference

Classes

• class GigE

7.8 nsCamera.comms.RS422 Namespace Reference

Classes

• class RS422

7.9 nsCamera.sensors Namespace Reference

Namespaces

- daedalus
- · icarus
- icarus2

Variables

```
• list __all__ = ["icarus", "icarus2", "daedalus"]
```

7.9.1 Variable Documentation

```
7.9.1.1 __all__
list nsCamera.sensors.__all__ = ["icarus", "icarus2", "daedalus"] [private]
Definition at line 23 of file __init__.py.
```

7.10 nsCamera.sensors.daedalus Namespace Reference

Classes

· class daedalus

7.11 nsCamera.sensors.icarus Namespace Reference

Classes

· class icarus

7.12 nsCamera.sensors.icarus2 Namespace Reference

Classes

• class icarus2

7.13 nsCamera.utils Namespace Reference

Namespaces

- crc16pure
- FlatField
- GenTec
- Ophir
- Packet
- Subregister

Variables

```
• list __all__ = ["SubRegister", "Packet", "GenTec", "Ophir", "FlatField"]
```

7.13.1 Variable Documentation

```
7.13.1.1 __all__
list nsCamera.utils.__all__ = ["SubRegister", "Packet", "GenTec", "Ophir", "FlatField"] [private]
Definition at line 30 of file __init__.py.
```

7.14 nsCamera.utils.crc16pure Namespace Reference

Functions

- def _crc16 (data, crc, table)
- def crc16xmodem (data, crc=0)

Variables

• list CRC16 XMODEM TABLE

7.14.1 Detailed Description

```
Pure python library for calculating CRC16 NOTE: modified slightly to combine Python 2 and Python 3 versions in single file
```

7.14.2 Function Documentation

7.14.2.1 _crc16()

```
def nsCamera.utils.crc16pure._crc16 (
                data,
                crc,
                table ) [private]
Calculate CRC16 using the given table.
'data'
             - data for calculating CRC, must be a string
'crc'
             - initial value
'table'
             - table for caclulating CRC (list of 256 integers)
Return calculated value of CRC
Definition at line 299 of file crc16pure.py.
        _crc16(data, crc, table):
"""Calculate CRC16 using the given table.
299 def
300
        'data'
301
                   - data for calculating CRC, must be a string
                    - initial value
302
        'crc'
                 - table for caclulating CRC (list of 256 integers)
        'table'
303
304
        Return calculated value of CRC
305
306
        for byte in data:
307
            if sys.version_info > (3,):
308
                crc = ((crc \ll 8) \& 0xFF00) ^ table[((crc <math> > 8) \& 0xFF) ^ byte]
309
            else:
310
                crc = ((crc « 8) & 0xFF00) ^ table[((crc » 8) & 0xFF) ^ ord(byte)]
311
312
       return crc & 0xFFFF
313
314
```

7.14.2.2 crc16xmodem()

Definition at line 315 of file crc16pure.py.

```
315 def crc16xmodem (data, crc=0):
316 """Calculate CRC-CCITT (XModem) variant of CRC16.
317 'data' - data for calculating CRC, must be a string
318 'crc' - initial value
319 Return calculated value of CRC
320 """
321 return _crc16(data, crc, CRC16_XMODEM_TABLE)
```

7.14.3 Variable Documentation

7.14.3.1 CRC16_XMODEM_TABLE

```
list nsCamera.utils.crc16pure.CRC16_XMODEM_TABLE
```

Definition at line 39 of file crc16pure.py.

7.15 nsCamera.utils.FlatField Namespace Reference

Functions

- def getFilenames (frame="Frame 1")
- def getROlvector (imgfilename, roi)
- def tslopes (x, y)
- def generateFF (FRAMES=["Frame_0", "Frame_1", "Frame_2", "Frame_3"], roi=[0, 0, 512, 1024], directory="", ncores=-1)
- def removeFF (filename, directory="", roi=[0, 0, 512, 1024])
- def removeFFall (directory="", FRAMES=["Frame_0", "Frame_1", "Frame_2", "Frame_3"], roi=[0, 0, 512, 1024])

Variables

- parser = argparse.ArgumentParser()
- action
- dest
- default
- help
- nargs
- args = parser.parse_args()
- list framelist = ["Frame_" + str(frame) for frame in args.frames]
- · directory

7.15.1 Function Documentation

7.15.1.1 generateFF()

```
def nsCamera.utils.FlatField.generateFF (
                 FRAMES = ["Frame_0", "Frame_1", "Frame_2", "Frame_3"],
                 roi = [0, 0, 512, 1024],
                 directory = "",
                 ncores = -1 )
Definition at line 58 of file FlatField.py.
       FRAMES=["Frame_0", "Frame_1", "Frame_2", "Frame_3"],
59
       roi=[0, 0, 512, 1024], directory="",
60
61
62
       ncores=-1,
63):
        # TODO: documentation
       # use of ROI here not compatible with use of ROI in removeFF
65
       if directory:
67
           cwd = os.getcwd()
           newpath = os.path.join(cwd, directory)
           os.chdir(newpath)
71
       if not FRAMES:
           print("No framelist provided, defaulting to four frames")
FRAMES = ["Frame_0", "Frame_1", "Frame_2", "Frame_3"]
72
73
       for f in FRAMES:
           files = getFilenames(frame=f)
76
            imgslist = [getROIvector(fn, roi) for fn in files] # a list of flattened images
           imgsarray = np.vstack(imgslist) # turn the list into an array
npix = np.shape(imgsarray)[1] # total number of pixels
77
78
79
           x = np.median(imgsarray, axis=1) # median of each image used for flat fielding
            y = []
80
81
            for i in range(npix):
                # each member of y represents a pixel, as a list of magnitudes over all the
82
83
                    images
84
                y.append(imgsarray[:, i])
           # get pixel gain and offset for flatfield ff using Thiel-Sen slopes
85
86
            ff = []
87
            ff = parallel.Parallel(n_jobs=ncores, verbose=5, pre_dispatch="2 * n_jobs")(
88
                delayed(tslopes)(x, pixel) for pixel in y
89
90
           \# x is the dependent variable; here uses median of image as characteristic of
91
           # noise level
92
            m, c = zip(*ff)
                              # separate into gain and offset
9.3
            m = np.array(m)
           m[m < 0.1] = 0.1 \# handle outliers
94
            m[m > 1000] = 1000 # handle outliers
95
96
            m = 1.0 / m
            m = m.reshape(roi[3] - roi[1], roi[2] - roi[0]) # turn into matrix
c = np.array(c).reshape(roi[3] - roi[1], roi[2] - roi[0]) # turn into matrix
97
98
99
            with open("px_gain_%s.txt" % f.replace("Frame_", "f"), "w+") as file:
100
1 0 1
                 np.savetxt(file, m)
             with open("px_off_%s.txt" % f.replace("Frame_", "f"), "w+") as file:
102
103
                 np.savetxt(file, c)
104
105
```

7.15.1.2 getFilenames()

Definition at line 32 of file FlatField.py.

7.15.1.3 getROlvector()

```
def nsCamera.utils.FlatField.getROIvector (
               imgfilename,
               roi )
return a numpy row vector of version of the image
Definition at line 40 of file FlatField.py.
40 def getROIvector(imgfilename, roi):
41
42
       return a numpy row vector of version of the image
4.3
      img = imread(imgfilename)
44
      vroi = img[(roi[1]) : (roi[3]), (roi[0]) : (roi[2])].flatten()
45
46
      return vroi
47
```

7.15.1.4 removeFF()

def nsCamera.utils.FlatField.removeFF (

48

```
filename.
                  directory = "",
                  roi = [0, 0, 512, 1024] )
Definition at line 106 of file FlatField.py.
106 def removeFF(filename, directory="", roi=[0, 0, 512, 1024]):
         if directory:
108
             cwd = os.getcwd()
             newpath = os.path.join(cwd, directory)
109
110
             os.chdir(newpath)
         framenum = re.search("Frame_(\d)", filename).group(1)
gainFilename = "px_gain_f" + framenum + ".txt"
111
112
         gainall = np.loadtxt(gainFilename)
113
         gain = gainall[(roi[1]) : (roi[3]), (roi[0]) : (roi[2])]
offFilename = "px_off_f" + framenum + ".txt"
114
115
116
         offsetall = np.loadtxt(offFilename, dtype="uint32")
          \texttt{offset = offsetall[(roi[1]) : (roi[3]), (roi[0]) : (roi[2])] } 
117
118
119
         beforeImageall = imread(filename)
         beforeImage = beforeImageall[(roi[1]) : (roi[3]), (roi[0]) : (roi[2])]
120
121
         imageMed = np.median(beforeImage)
122
         flat = imageMed * gain + offset
123
         flat = flat.clip(0)
124
         fix = beforeImage - flat
125
126
         clipped = fix.clip(0)
         fixinit = clipped.astype("uint16")
127
         fiximg = Image.fromarray(fixinit)
128
129
130
         fixFilename = filename[:-4] + "ff" + filename[-4:]
         fiximg.save(fixFilename)
131
132
```

7.15.1.5 removeFFall()

```
def nsCamera.utils.FlatField.removeFFall (
                directory = "",
                FRAMES = ["Frame_0", "Frame_1", "Frame_2", "Frame_3"],
                roi = [0, 0, 512, 1024] )
Definition at line 133 of file FlatField.py.
133 def removeFFall(
        directory="",
134
        FRAMES=["Frame_0", "Frame_1", "Frame_2", "Frame_3"],
135
        roi=[0, 0, 512, 1024],
136
137):
        cwd = os.getcwd()
138
        if directory:
   newpath = os.path.join(cwd, directory)
139
140
        else:
141
           newpath = cwd
142
143
        os.chdir(newpath)
        files = next(os.walk("./"))[2]
144
        filelist = []
145
        for frame in FRAMES:
146
            file list.extend([k \ for \ k \ in \ files \ if \ frame \ in \ k \ and \ "tif" \ in \ k])
147
148
        for fname in filelist:
149
            removeFF(fname, directory, roi)
150
151
```

7.15.1.6 tslopes()

```
def nsCamera.utils.FlatField.tslopes ( x, \\ y \ ) theilslopes implements a method for robust linear regression. It computes the slope as the median of all slopes between paired values.
```

Definition at line 49 of file FlatField.py.

7.15.2 Variable Documentation

7.15.2.1 action

nsCamera.utils.FlatField.action

Definition at line 157 of file FlatField.py.

7.15.2.2 args

```
nsCamera.utils.FlatField.args = parser.parse_args()
```

Definition at line 167 of file FlatField.py.

7.15.2.3 default

```
nsCamera.utils.FlatField.default
```

Definition at line 157 of file FlatField.py.

7.15.2.4 dest

```
nsCamera.utils.FlatField.dest
```

Definition at line 157 of file FlatField.py.

7.15.2.5 directory

```
nsCamera.utils.FlatField.directory
```

Definition at line 169 of file FlatField.py.

7.15.2.6 framelist

```
nsCamera.utils.FlatField.framelist = ["Frame_" + str(frame) for frame in args.frames]
```

Definition at line 168 of file FlatField.py.

7.15.2.7 help

 ${\tt nsCamera.utils.FlatField.help}$

Definition at line 157 of file FlatField.py.

7.15.2.8 nargs

nsCamera.utils.FlatField.nargs

Definition at line 161 of file FlatField.py.

7.15.2.9 parser

```
nsCamera.utils.FlatField.parser = argparse.ArgumentParser()
```

Definition at line 155 of file FlatField.py.

7.16 nsCamera.utils.GenTec Namespace Reference

Classes

class GenTec

Variables

• gt = GenTec()

7.16.1 Variable Documentation

7.16.1.1 gt

```
nsCamera.utils.GenTec.gt = GenTec()
```

Definition at line 92 of file GenTec.py.

7.17 nsCamera.utils.Ophir Namespace Reference

Classes

• class Ophir

7.18 nsCamera.utils.Packet Namespace Reference

Classes

class Packet

7.19 nsCamera.utils.Subregister Namespace Reference

Classes

class SubRegister

7.20 nsCameraExample Namespace Reference

Variables

7.20.1 Variable Documentation

7.20.1.1 BOARD

```
string nsCameraExample.BOARD = "LLNL_v4"
```

The CameraAssembler code initializes and manages objects corresponding to the three components that comprise a particular nsCamera system. The 'verbose' flag controls the output of status messages as the code executes.

Definition at line 31 of file nsCameraExample.py.

7.20.1.2 ca

nsCameraExample.ca = CameraAssembler(commname=COMM, boardname=BOARD, sensorname=SENSOR, verbose=4)

Definition at line 42 of file nsCameraExample.py.

7.20.1.3 COMM

```
string nsCameraExample.COMM = "GigE"
```

Definition at line 34 of file nsCameraExample.py.

7.20.1.4 SENSOR

string nsCameraExample.SENSOR = "icarus2"

Definition at line 38 of file nsCameraExample.py.

7.20.1.5 timing

nsCameraExample.timing

The initialization phase sets the default high-speed timing parameters. Override these settings here; alternatively, switch to manual shutter control

Definition at line 63 of file nsCameraExample.py.

7.20.1.6 tune

nsCameraExample.tune

The initialization phase sets default operating parameters. These parameters may be overridden by explicit directives as shown here. setPotV (potname, voltage) - Voltage is float setPotV sets contents of pot 'potname' to the value corresponding to 'voltage' based on board.monVmin and board.monVmax Valid 'name' entries are listed as keys in the 'channel_lookups' dictionary in the board code

Definition at line 82 of file nsCameraExample.py.

7.21 testSuite Namespace Reference

Functions

• def testSuite (board, comm, sensor, portNum, ipAdd, interactive=True, swtrigger=True)

Variables

- parser = argparse.ArgumentParser()
- action
- dest
- default
- help
- None
- args = parser.parse_args()
- · interactive
- swtrigger
- board
- comm
- sensor
- portNum
- ipAdd

7.21.1 Function Documentation

7.21.1.1 testSuite()

```
def testSuite.testSuite (
              board,
              comm,
              sensor,
              portNum,
              ipAdd,
              interactive = True,
              swtrigger = True )
Regression testing script to exercise cameraAssembler functions and camera features.
Comment out entries in 'tests' list to skip component tests
   board: board name for cameraAssembler
    comm: comm name for cameraAssembler
    \verb"sensor: sensor name for cameraAssembler"
    portNum: (optional) port number
    ipAdd: (optional) ip address (e.g., ^\prime192.168.1.100^\prime)
    interactive: if False, does not wait for user input, skips some tests
    swtrigger: if True, uses software triggering, does not wait for external
      triggers
```

Definition at line 39 of file testSuite.py.

```
39 def testSuite(board, comm, sensor, portNum, ipAdd, interactive=True, swtrigger=True):
40
41
       Regression testing script to exercise cameraAssembler functions and camera features.
       Comment out entries in 'tests' list to skip component tests
43
44
45
          board: board name for cameraAssembler
           comm: comm name for cameraAssembler
           sensor: sensor name for cameraAssembler
           portNum: (optional) port number
           ipAdd: (optional) ip address (e.g., '192.168.1.100')
50
           interactive: if False, does not wait for user input, skips some tests
51
           swtrigger: if True, uses software triggering, does not wait for external
52
             triggers
53
54
       # Comment out any tests you wish to skip. Irrelevant tests (e.g., Manual Timing if
55
          Daedalus sensor is attached) will be ignored
56
57
       tests = [
           "HST acquisition",
58
           "SaveTiffs",
59
60
           "PlotFrames",
61
           "POT/DAC set & read",
           "SW Trigger", # perform SW trigger test when HW triggering selected
62
63
           "Arm/Disarm",
           "HST setting",
64
           "Reinitialization",
65
           "PowerSave",
66
           "Timer",
67
           "Register R/W",
"Register self-clear",
68
69
           "Register dump",
70
           "LED",
71
           "Manual Timing",
72
73
       ]
74
75
       def statusVerify(caObject, checklist):
76
          errs = 0
77
           for stat, flag in checklist:
               _, check = caObject.getSubregister(stat)
78
79
                if bool(int(check)) is not bool(flag):
80
                   errs += 1
                   print("+Error: " + stat + " is not " + str(bool(flag)))
81
82
           if not errs:
83
               print("+Status verify passed")
84
       errtemp = 0
85
86
       sensorregs = {}
87
       boardregs = {}
88
       boardselfclear = {}
89
90
       def test_v1(ca):
91
           print("\n# LLNL_v1 board-specific checks")
92
           if "LED" in tests:
93
               \verb|print("\nRolling LEDs")| \\
94
               ca.enableLED(0)
               ca.enableLED(1)
95
96
               for i in range(1, 5):
97
                   for j in range(1, 9):
                       ca.setLED(j, 1)
99
                        time.sleep(0.05 * i)
100
                         ca.setLED(j, 0)
101
            if "POT/DAC set & read" in tests:
102
103
                time.sleep(1)
104
                print("\n-Pot check-")
                 for i in [2, 3, 4, 6, 8]:
potname = "POT" + str(i)
105
106
                     monname = "MON_CH" + str(i)
107
108
                     print("Testing " + potname)
                     temperr = 0
109
                     for j in range(7):
110
                         desired = j * 0.5
111
112
                         potobj = getattr(ca.board, potname)
113
                         minvolt = potobj.resolution
                         ca.setPotV(potname, desired, tune=True)
114
                         actual = ca.getMonV(monname)
115
116
                         \# skip v=0, we expect it to be off
                         if abs(desired - actual) > minvolt and bool(desired):
117
118
                             print (
```

```
119
                                    "\{0:.2f\}: actual = \{1:.5f\}; delta = \{2:.2f\} mV".format(
120
                                        (1.0 * desired), actual, 1000 * abs(actual - desired)
121
122
123
                               temperr = 1
                      if not temperr:
124
125
                          print("+" + potname + " tunes properly")
126
127
             v1regs = OrderedDict(
128
                      "ADC_RESET": "0000001F",
129
                      "ADC5_CONFIG_DATA": "FFFFFFFF",
130
                      "POT_REG4_TO_1": "FFFFFFFF",
"POT_REG8_TO_5": "FFFFFFFFF",
131
132
                      "POT_REG12_TO_9": "FFFFFFFF",
"POT_REG13": "000000FF",
133
134
                      "ADC5_PPER": "OFFFFFFF",
135
                      "LED_GP": "000000FF",
136
                      "ADC_STANDBY": "0000001F",
137
                      "TEMP_SENSE_PPER": "OFFFFFFF",
138
                      "SENSOR_VOLT_CTL": "00000001",
139
140
141
142
             v1selfclear = OrderedDict({"POT_CTL": "00000001",})
143
144
             v1restore = [
                  ("ADC5_PPER", "001E8480"),
145
                  ("ADC_RESET", "00000000"),
146
                  ("ADC5_CONFIG_DATA", "81A883FF"), ("ADC_CTL", "00000010"),
147
148
149
150
151
             return v1regs, v1selfclear, v1restore
152
153
        def test v4(ca):
             print("\n# LLNL_v4 board-specific checks")
154
             if "POT/DAC set & read" in tests:
    print("\n-DAC check-")
155
156
                  for i in ["A", "B", "C", "D", "E", "F", "G", "H"]:
   dacname = "DAC" + i
157
158
                      monname = "MON_CH" + i
159
160
                      print("Testing " + dacname)
161
                      temperr = 0
162
                      for j in range(7):
163
                           desired = j * 0.5
164
                           # semi-arbitrary, need to adjust to minimize search time
165
                          minvolt = 0.005
166
                           ca.setPotV(dacname, desired, tune=True)
167
                           actual = ca.getMonV(monname)
168
                           \# skip v=0, we expect it to be off
169
                           if abs(desired - actual) > minvolt and bool(desired):
170
                               print(
171
                                   "\{0:.2f\}: actual = \{1:.5f\}; delta = \{2:.2f\} mV".format(
172
                                        (1.0 * desired), actual, 1000 * abs(actual - desired)
173
174
175
                               temperr = 1
176
                      if not temperr:
177
                          print("+" + dacname + " tunes properly")
178
179
             v4regs = OrderedDict(
180
181
                      "ADC_RESET": "0000000F",
                      "DAC_REG_A_AND_B": "FFFFFFF",
182
                      "DAC_REG_C_AND_D": "FFFFFFFF",
183
                      "DAC_REG_E_AND_F": "FFFFFFFF",
184
                      "DAC_REG_G_AND_H": "FFFFFFFF",
185
186
187
             )
188
189
             v4selfclear = OrderedDict(
                 {"DAC_CTL": "00000001", "SW_COARSE_CONTROL": "FFFFFFFF",}
190
191
192
193
             v4restore = [
                 ("ADC_PPER", "001E8480"),
("ADC_RESET", "00000000"),
194
195
196
197
198
             return v4regs, v4selfclear, v4restore
199
```

```
200
        def test_icarus(ca, interactive, swtrigger):
201
            print("\n# Icarus sensor-specific checks")
202
203
            if "Manual Timing" in tests:
204
                # MANUAL TIMING & ACQUISITION
205
                print("\n-Testing manual shutter control-")
206
                ca.setManualShutters(
207
                    timing=[
208
                         (100, 100, 100, 100, 100, 100, 100),
                         (100, 100, 100, 100, 100, 100, 100),
209
210
211
212
                statusVerify(ca, [("MANSHUT_MODE", 1), ("STAT_HSTCONFIGURED", 0)])
213
                print (
214
                    "The next few messages should include two error messages about "
215
                    "invalid timing sequences "
216
217
                time.sleep(1)
                ca.setManualShutters(timing=[(100, 100, 100, 100, 100, 100, 100)])
218
219
                ca.setManualShutters(
220
                    timing=[
221
                         (10.5, 100, 100, 100, 100, 100, 100),
                         (100, 100, 100, 100, 100, 100, 100),
222
223
224
225
                time.sleep(1)
226
                print("\n-Testing manual shutter acquisition-")
227
228
                if swtrigger:
                    print("Using software trigger")
229
                    statusVerify(
230
231
                        ca,
232
                             ("STAT_ADCSCONFIGURED", 1),
2.33
                             (ca.potsdacsconfigured, 1),
234
                             ("STAT_HSTCONFIGURED", 0),
2.35
2.36
                             ("MANSHUT_MODE", 1),
237
                        ],
2.38
                    ca.arm("Software")
239
                    statusVerify(ca, [("STAT_COARSE", 1), ("STAT_FINE", 1)])
240
2.41
                else:
                    ca.arm()
242
243
                    if interactive:
244
                        ca.getEnter(
245
                             "> Please initiate hardware trigger, then press ENTER to "
246
                             "continue. <\n"
247
248
249
                frames, datalen, data_err = ca.readoff(waitOnSRAM=True)
250
                print("Data length: " + str(datalen))
251
                if data_err:
252
                    print("+Error in acquisition!")
253
                else:
254
                    print("+No error in acquisition reported")
255
256
                if interactive:
257
                    if "PlotFrames" in tests:
258
259
                             "Plots of the acquired frames are being displayed. Please "
                             "inspect to verify proper acquisition."
260
261
                             "\n> Close plots to continue <"
262
263
                         ca.plotFrames(frames)
                     if "SaveTiffs" in tests:
264
                        ca.saveTiffs(frames, filename="msc_test")
265
266
                         ca.getEnter(
267
                             "Tiff files of the acquired frames have been saved. Please "
268
                             "inspect to verify correct saves.
269
                             "\n> Press ENTER to continue <\n"
270
271
272
                else:
273
                    if "SaveTiffs" in tests:
274
                        ca.saveTiffs(frames, filename="msc_test")
275
                        print("Tiffs from manual shutter control test have been saved")
276
277
                # REINITIALIZATION WITH MANUAL SHUTTERS
                if "Reinitialization" in tests and interactive:
278
279
                    ca.setManualShutters(
280
                        timing=[
```

```
281
                                  (25, 50, 75, 100, 125, 150, 175),
282
                                  (175, 150, 125, 100, 75, 50, 25),
283
284
285
                        ca.getEnter(
286
                             "\n-Testing reinitialization with manual shutter control-\n>"
287
                             "Please power-cycle the board, then press ENTER to continue <"
288
289
                        time.sleep(1)
290
                        if ca.powerCheck():
291
                            print("\n+Loss of power WAS NOT detected")
                        else:
292
293
                            print("\n+Loss of power WAS detected")
                        time.sleep(1)
295
                        ca.reinitialize()
296
                        statusVerify(ca, [("STAT_TIMERCOUNTERRESET", 1)])
297
                        ca.sensor.getManualTiming()
                        if ca.sensor.getManualTiming() != [
[25, 50, 75, 100, 125, 150, 175],
[175, 150, 125, 100, 75, 50, 25],
298
299
300
301
                        1:
302
                             print (
303
                                  "+Manual timing WAS NOT restored properly after "
304
                                  "reinitialization "
305
                             )
306
                        else:
307
                             print(
                                  "+Manual timing WAS restored properly after "
308
                                  "reinitialization "
309
310
311
              icarusregs = OrderedDict(
312
313
                        "VRESET_WAIT_TIME": "7FFFFFFF",
"ICARUS_VER_SEL": "00000001",
"MANUAL_SHUTTERS_MODE": "00000001",
314
315
316
                        "W0_INTEGRATION": "03FFFFFF",
"W0_INTERFRAME": "03FFFFFF",
317
318
                        "W1_INTEGRATION": "03FFFFFF",
"W1_INTERFRAME": "03FFFFFF",
319
320
                        "W2_INTEGRATION": "03FFFFFF",
"W2_INTEGRATION": "03FFFFFF",
"W3_INTEGRATION": "03FFFFFF",
321
322
323
                        "WO_INTEGRATION_B": "03FFFFFF",
"WO_INTERFRAME_B": "03FFFFFF",
"W1_INTEGRATION_B": "03FFFFFF",
324
325
326
                        "W1_INTERFRAME_B": "03FFFFFF",
327
                        "W2_INTEGRATION_B": "03FFFFFF",
328
                        "W2_INTERFRAME_B": "03FFFFFF",
329
330
                        "W3_INTEGRATION_B": "03FFFFFF",
331
332
              )
333
334
              if ca.sensorname == "icarus":
335
                   icarusregs["VRESET_HIGH_VALUE"] = "000000FF"
336
337
              return icarusregs
338
339
         def test_daedalus(ca, interactive, swtrigger):
340
              print("\n# Daedalus sensor-specific checks")
              daedalusregs = {} # TODO: add daedalus registers when available
341
342
              return daedalusregs
343
344
         print("-Initial setup-")
345
         ca = CameraAssembler(
346
              commname=comm,
347
              boardname=board,
348
              sensorname=sensor,
349
              verbose=5,
350
              port=portNum,
351
              ip=ipAdd,
352
353
         ca.potsdacsconfigured = "STAT_DACSCONFIGURED"
354
355
         statusVerify(ca, [("STAT_TIMERCOUNTERRESET", 1)])
356
         if "PowerSave" in tests:
357
              print("\n-Testing PowerSave mode-")
358
359
              ca.setPowerSave(1)
              statusVerify(ca, [("POWERSAVE", 1)])
ca.setPowerSave(0)
360
361
```

```
362
             statusVerify(ca, [("POWERSAVE", 0)])
363
364
         # ARM/DISARM
365
         if "Arm/Disarm" in tests:
366
             print("\n-Testing Arm-")
367
              # HST has to be set for arming to complete; tested separately later
             ca.setTiming("A", (2, 2))
ca.setTiming("B", (2, 2))
368
369
370
             ca.arm()
371
             time.sleep(1)
372
             statusVerify(
373
                 ca,
374
                  ſ
375
                       ("STAT_ADCSCONFIGURED", 1),
376
                       (ca.potsdacsconfigured, 1),
377
                       ("STAT_COARSE", 0),
                       ("STAT_FINE", 0),
("STAT_ARMED", 1),
378
379
380
                  ],
381
382
             time.sleep(1)
             print("\n-Testing Disarm-")
383
384
             ca.disarm()
385
             statusVerify(ca, [("STAT_ARMED", 0)])
386
             time.sleep(1)
387
         # HIGH SPEED TIMING & ACQUISITION
388
         if "HST setting" in tests:
389
             \label{eq:print("n-Testing high speed timing control-")} print("\n-Testing high speed timing control-")
390
             print("\n-resting high speed timing control-
ca.setTiming("A", (39, 1), 0)
ca.setTiming("B", (1, 1), 31)
if not ("A", 39, 1, 0) == ca.getTiming("A"):
391
392
393
             errtemp = 1
if not ("B", 1, 1, 37) == ca.getTiming("B"):
394
395
396
                  errtemp = 1
397
             if errtemp:
398
                  print("Error in setting high speed timing")
399
                  errtemp = 0
400
             ca.setTiming("A", (5, 2), 3)
401
             ca.setTiming("B", (3, 4), 1)
402
403
             if not ("A", 5, 2, 3) == ca.getTiming("A"):
404
                  errtemp = 1
405
             if not ("B", 3, 4, 1) == ca.getTiming("B"):
406
                  errtemp = 1
407
             if errtemp:
408
                  print("Error in setting high speed timing")
409
                  errtemp = 0
410
411
             ca.setTiming("B", (10, 10), 0)
412
413
             time.sleep(1)
414
             print(
415
                  "The next few messages should include a warning about inter-frame timing:"
416
417
             time.sleep(1)
418
             ca.setTiming("A", (9, 8), 1)
419
420
                  "The next few messages should include a error message regarding timing "
421
                  "sequence: "
422
423
             time.sleep(1)
424
             ca.setTiming("A", (15, 15), 15)
425
             statusVerify(ca, [("STAT_HSTCONFIGURED", 1), ("MANSHUT_MODE", 0)])
426
427
             time.sleep(1)
428
429
         # ca.setInterlacing(2) # TODO: sensor-specific testing?
430
         if "HST acquisition" in tests:
431
432
             print("\n-Testing HST acquisition-")
             ca.arm()
433
434
             time.sleep(1)
435
436
             ca.reportStatus()
             statusVerify(
437
438
                 ca,
439
                       ("STAT_ADCSCONFIGURED", 1),
440
                       (ca.potsdacsconfigured, 1),
441
                       ("STAT_HSTCONFIGURED", 1),
442
```

```
443
                      ("MANSHUT_MODE", 0),
444
                      ("STAT_COARSE", 0),
445
                      ("STAT_FINE", 0),
446
                 1,
447
            if swtrigger:
448
449
                 print("Using software trigger")
450
                 ca.arm("Software")
451
                 statusVerify(ca, [("STAT_COARSE", 1), ("STAT_FINE", 1)])
452
453
            else:
454
                 if interactive:
455
                     ca.getEnter(
456
                          "> Please initiate hardware trigger, then press ENTER to "
457
                          "continue. <\n "
458
459
460
            frames, datalen, data_err = ca.readoff(waitOnSRAM=True)
461
462
            margin = 1600
463
            print(
                 "Check of dummy sensor; number of pixels exceeding margin of "
464
465
                 + str(margin)
466
                 + " from dummy sensor expected value:"
467
468
            for frame in frames:
                 bads, diff = ca.dummyCheck(frame, margin)
469
470
                 print(bads)
471
            print("Data length: " + str(datalen) + " bytes")
472
473
            if data err:
474
                 print("+Error in acquisition!")
475
            else:
                 print("+No error in acquisition reported")
476
477
478
            if interactive:
                 if "PlotFrames" in tests:
479
480
                     print(
                          "\nPlots of the acquired frames are being displayed. Please " "inspect to verify proper acquisition. " \,
481
482
                          "\n> Close plots to continue <"
483
484
                 ca.plotFrames(frames)
if "SaveTiffs" in tests:
485
486
                     ca.saveTiffs(frames, filename="hst_test")
487
488
                     ca.getEnter(
489
                          "Tiff files of the acquired frames have been saved. Please " \,\,
490
                          "inspect to verify correct saves. '
491
                          "\n> Press ENTER to continue. <\n"
492
493
            else:
                 if "SaveTiffs" in tests:
494
495
                     ca.saveTiffs(frames, filename="hst_test")
496
                     print("Tiffs from HST test have been saved")
497
498
            if not swtrigger and "SW Trigger" in tests:
499
                print("\n-Testing HST acquisition with software trigger-")
500
                 ca.arm()
                 time.sleep(1)
501
502
503
                 ca.reportStatus()
504
                 statusVerify(
505
                     ca,
506
                     Γ
                          ("STAT_ADCSCONFIGURED", 1),
507
508
                          (ca.potsdacsconfigured, 1),
                          ("STAT_HSTCONFIGURED", 1),
509
510
                          ("MANSHUT_MODE", 0),
511
                          ("STAT_COARSE", 0),
                          ("STAT_FINE", 0),
512
513
                     1,
                 )
514
515
                 ca.arm("Software")
516
                 statusVerify(ca, [("STAT_COARSE", 1), ("STAT_FINE", 1)])
517
518
519
                 frames, datalen, data err = ca.readoff(waitOnSRAM=True)
520
521
                 margin = 1600
522
                 print(
523
                     "Check of dummy sensor; number of pixels exceeding margin of "
```

```
524
                     + str(margin)
525
                     + " from dummy sensor expected value:"
526
527
                 for frame in frames:
                     bads, diff = ca.dummyCheck(frame, margin)
528
529
                     print(bads)
530
531
                print("Data length: " + str(datalen) + " bytes")
532
                 if data err:
533
                     print("+Error in acquisition!")
534
                    print("+No error in acquisition reported")
535
536
537
                 if interactive:
538
                     if "PlotFrames" in tests:
539
                         print(
540
                              "\nPlots of the acquired frames are being displayed. Please "
                              "inspect to verify proper acquisition. '
541
542
                              "\n> Close plots to continue <"
543
544
                          ca.plotFrames(frames)
545
                     if "SaveTiffs" in tests:
546
                         ca.saveTiffs(frames, filename="SWtrig_test")
547
                         ca.getEnter(
548
                              "Tiff files of the acquired frames have been saved. Please "
                              "inspect to verify correct saves.
549
                              "\n> Press ENTER to continue. <\n"
550
551
552
                else:
                     if "SaveTiffs" in tests:
553
                         ca.saveTiffs (frames, filename="SWtrig_test")
print("Tiffs from HST test have been saved")
554
555
556
        # REINITIALIZATION
557
        if "Reinitialization" in tests and interactive:
558
559
            time.sleep(1)
            ca.setTiming("A", (2, 3), 4) ca.setTiming("B", (5, 3), 1)
560
561
562
            time.sleep(1)
563
            ca.getEnter(
564
                 "\n-Testing reinitialization with high speed timing-\n>Please power-cycle "
                 "the board, then press ENTER to continue <"
565
566
567
            if ca.powerCheck():
568
                print("\n+Loss of power WAS NOT detected")
569
            else:
570
                print("\n+Loss of power WAS detected")
571
            time.sleep(1)
572
            ca.reinitialize()
573
            statusVerify(ca, [("STAT_TIMERCOUNTERRESET", 1)])
            if ("A", 2, 3, 4) != ca.getTiming("A") or ("B", 5, 3, 1,) != ca.getTiming("B"):
574
                print("+High speed timing WAS NOT restored properly after reinitialization")
575
576
            else:
577
                print("+High speed timing WAS restored properly after reinitialization")
578
579
        if ca.sensorname == "icarus" or ca.sensorname == "icarus2":
580
            sensorregs = test_icarus(ca, interactive, swtrigger)
581
        elif ca.sensorname == "daedalus":
582
            sensorregs = test_daedalus(ca, interactive, swtrigger)
583
584
585
        print("\n-Testing miscellaneous board features-")
586
587
        if "Timer" in tests:
            print("Checking on-board timer reset")
588
589
            ca.resetTimer()
590
            ztime = ca.getTimer()
591
            if not ztime:
592
                print("+Timer reset check successful")
593
            else:
594
                print("+Timer reset failed, timer reads " + str(ztime))
595
            statusVerify(ca, [("STAT_TIMERCOUNTERRESET", 1)])
596
597
        print("Temperature sensor reading: " + str(ca.getTemp()))
598
        time.sleep(1)
599
        if ca.boardname == "llnl v1":
600
            ca.potsdacsconfigured = "STAT_POTSCONFIGURED"
601
            boardregs, boardselfclear, boardrestore = test_v1(ca)
602
        elif ca.boardname == "llnl_v4":
        ca.potsdacsconfigured = "STAT_DACSCONFIGURED"
603
604
```

```
605
             boardregs, boardselfclear, boardrestore = test_v4(ca)
606
607
         if "POT/DAC set & read" in tests:
608
             print("\n-VRST check-")
609
             for a in (0, 0.05, 0.15, 0.25, 0.5, 0.75, 1, 3, 3.5):
610
                  ca.setPotV("VRST", voltage=a, tune=True)
                  actual = ca.getMonV("VRST")
612
                      "\{0:.2f\} : actual = \{1:.5f\} ; delta = \{2:.2f\} mV".format(
613
                           (1.0 * a), actual, 1000 * abs(actual - a)
614
616
617
         if "Register R/W" in tests:
618
619
             print("\n\n-Verifying register read/writes-\n")
620
             regchecklist = OrderedDict(
621
                  { # register name: writable bits
                      # read-only, write-only, and self-clearing registers are skipped
"HS_TIMING_DATA_ALO": "FFFFFFFF",
622
623
                       "HS_TIMING_DATA_AHI": "000000FF",
624
                      "HS_TIMING_DATA_BLO": "FFFFFFFF",
"HS_TIMING_DATA_BHI": "000000FF",
625
626
627
                       "CTRL_REG": "0000000F",
628
                      "HST_SETTINGS": "00000003",
                      "DIAG_MAX_CNT_0": "FFFFF00FF",
"DIAG_MAX_CNT_1": "FFFFFFFF",
"TRIGGER_CTL": "00000003",
629
630
631
                       "FPA_ROW_INITIAL": "000003FF",
632
                       "FPA_ROW_FINAL": "000003FF"
633
                       "FPA_FRAME_INITIAL": "00000003",
634
                       "FPA_FRAME_FINAL": "00000003",
635
                       "FPA_DIVCLK_EN_ADDR": "00000001",
636
                       "FPA_OSCILLATOR_SEL_ADDR": "00000003",
"ADC1_CONFIG_DATA": "FFFFFFFF",
"ADC2_CONFIG_DATA": "FFFFFFFF",
637
638
639
                       "ADC3_CONFIG_DATA": "FFFFFFFF",
640
                       "ADC4_CONFIG_DATA": "FFFFFFFF",
641
                       "ADC_RESET": "0000001F",
642
643
644
             )
645
646
             regchecklist.update(sensorregs)
647
             regchecklist.update(boardregs)
648
649
             checkvals = ["00000000", "FFFFFFFF"]
650
651
             for reg, mask in regchecklist.items():
652
                  temperr = 0
653
                  for val in checkvals:
654
                       valmasked = "{0:0=8x}".format(int(val, 16) & int(mask, 16))
655
                      if temperr:
656
657
                       if not ca.checkRegSet(reg, valmasked):
658
                           temperr = 1
659
660
                  if not temperr:
661
                      print("+ {: <24} - R/W OK".format(reg))</pre>
             ca.submitMessages(boardrestore)
663
         if "Register self-clear" in tests:
665
             time.sleep(1)
667
             print("\n^{n-Verifying} self-clearing registers-\n^{"})
668
669
             selfclear = OrderedDict(
                  { # register name: writable bits
670
671
                      "HS_TIMING_CTL": "FFFFFFFF", # Read-write registers
                       "TIMER_CTL": "FFFFFFF",
672
                       "ADC_CTL": "FFFFFFF",
673
                      "STAT_REG_SRC": "00004FFF", # Read-only registers "STAT_REG2_SRC": "FFFFFFFF",
675
676
677
             )
678
679
             selfclear.update(boardselfclear)
680
             for reg, mask in selfclear.items():
681
682
                  ca.setRegister(reg, "FFFFFFFF")
683
                  ca.getRegister(reg)
684
                 time.sleep(0.1)
685
                  _, resp = ca.getRegister(reg)
```

```
686
                   masked = int(resp, 16) & int(mask, 16)
687
688
                   if not masked:
689
690
                        print("+ {: <17} - self-clear OK".format(reg))</pre>
691
                        print(
    "+ {: <17} - self-clear FAIL: ".format(reg)
    + "0x"
    "'^^-°" format(masked)</pre>
692
693
694
697
698
              ca.submitMessages(boardrestore)
699
700
         if "Register dump" in tests:
              time.sleep(1)
701
             print("\n\n-Register dump-\n")
print("\n".join(ca.dumpRegisters()))
702
703
704
705
        ca.closeDevice()
706
         time.sleep(1)
         logging.info("Done")
707
708
709
```

7.21.2 Variable Documentation

7.21.2.1 action

testSuite.action

Definition at line 717 of file testSuite.py.

7.21.2.2 args

```
testSuite.args = parser.parse_args()
```

Definition at line 749 of file testSuite.py.

7.21.2.3 board

testSuite.board

Definition at line 754 of file testSuite.py.

7.21.2.4 comm

testSuite.comm

Definition at line 755 of file testSuite.py.

7.21.2.5 default

testSuite.default

Definition at line 719 of file testSuite.py.

7.21.2.6 dest

testSuite.dest

Definition at line 718 of file testSuite.py.

7.21.2.7 help

testSuite.help

Definition at line 720 of file testSuite.py.

7.21.2.8 interactive

testSuite.interactive

Definition at line 752 of file testSuite.py.

7.21.2.9 ipAdd

testSuite.ipAdd

Definition at line 758 of file testSuite.py.

7.21.2.10 None

testSuite.None

Definition at line 737 of file testSuite.py.

7.21.2.11 parser

testSuite.parser = argparse.ArgumentParser()

Definition at line 713 of file testSuite.py.

7.21.2.12 portNum

testSuite.portNum

Definition at line 757 of file testSuite.py.

7.21.2.13 sensor

testSuite.sensor

Definition at line 756 of file testSuite.py.

7.21.2.14 swtrigger

testSuite.swtrigger

Definition at line 753 of file testSuite.py.

Chapter 8

Class Documentation

8.1 nsCamera.CameraAssembler.CameraAssembler Class Reference

Public Member Functions

- def __init__ (self, boardname="Ilnl_v4", commname="GigE", sensorname="icarus2", verbose=4, port=None, ip=None, logfile=None, logfag=None)
- def initBoard (self)

Aliases to other objects' methods.

- def initPots (self)
- def latchPots (self)
- def initSensor (self)
- def configADCs (self)
- def disarm (self)
- def startCapture (self, mode)
- def readSRAM (self)
- def waitForSRAM (self, timeout=None)
- def getTimer (self)
- def resetTimer (self)
- def enableLED (self, status=1)
- def setLED (self, LED=1, status=1)
- def setPowerSave (self, status=1)
- def setPPER (self, time=None)
- def getTemp (self, scale=None)
- def getPressure (self, offset=None, sensitivity=None, units=None)
- · def clearStatus (self)
- def checkStatus (self)
- def checkStatus2 (self)
- def reportStatus (self)
- def reportEdgeDetects (self)
- def dumpStatus (self)
- def checkSensorVoltStat (self)
- def setTiming (self, side=None, sequence=None, delay=None)
- def setArbTiming (self, side=None, sequence=None)

- def getTiming (self, side=None, actual=None)
- def setManualShutters (self, timing=None)
- · def getManualTiming (self)
- def sensorSpecific (self)
- def setInterlacing (self, ifactor=None)
- def setHighFullWell (self, flag=True)
- def setZeroDeadTime (self, flag=True)
- def setTriggerDelay (self, delayblocks=0)
- def parseReadoff (self, frames)
- def sendCMD (self, pkt)
- def arm (self, mode=None)
- def readoff (self, waitOnSRAM=None, timeout=0, fast=None)
- def writeSerial (self, cmd, timeout=None)
- def readSerial (self, size, timeout=None)
- def closeDevice (self)
- · def initialize (self)

End aliases.

- def reinitialize (self)
- · def reboot (self)
- def getBoardInfo (self)
- def getRegister (self, regname)
- def setRegister (self, regname, regval)
- def resolveSubreg (self, srname)
- def getSubregister (self, subregname)
- def setSubregister (self, subregname, valstring)
- def submitMessages (self, messages, errorstring="Error")
- def getPot (self, potname, errflag=False)
- def setPot (self, potname, value=1.0, errflag=False)
- def getPotV (self, potname, errflag=False)
- def setPotV (self, potname, voltage, tune=False, accuracy=0.01, iterations=20, approach=0.75, errflag=False)
- def getMonV (self, monname, errflag=False)
- def readImgs (self, waitOnSRAM=True, mode="Hardware")
- def deInterlace (self, frames, ifactor=1)
- def saveFrames (self, frames, path=None, filename="frames", prefix=None)
- def saveTiffs (self, frames, path=None, filename="Frame", prefix=None, index=None)
- def saveNumpys (self, frames, path=None, filename="Frame", prefix=None, index=None)
- def dumpNumpy (self, datastream, path=None, filename="Dump", prefix=None)
- def plotFrames (self, frames, index=None)
- def checkCRC (self, rval)
- def checkRegSet (self, regname, teststring)
- def initPowerCheck (self)
- def powerCheck (self, delta=10)
- def dummyCheck (self, image, margin, dummyVals=None)
- def printBoardInfo (self)
- def dumpRegisters (self)
- def dumpSubregisters (self)
- def str2bytes (self, astring)
- def bytes2str (self, bytesequence)
- def str2nparray (self, valstring)
- def flatten (self, x)

- def getEnter (self, text)
- def mmReadoff (self, waitOnSRAM, variation=None)
- def setFrames (self, minframe=None, maxframe=None)
- def setRows (self, minrow=0, maxrow=None, fullsize=False)
- def generateFrames (self, data)
- def abortReadoff (self, flag=True)
- def batchAcquire (self, sets=1, trig="Hardware", path=None, filename="Frame", prefix=None, showProgress=0)
- def loadTextFrames (self, filename='frames.txt', path=None)

Public Attributes

- version
- currtime
- oldtime
- · trigtime
- waited
- read
- unstringed
- · parsedtime
- · savetime
- cycle
- boardname
- commname
- sensorname

For regular version.

- verbose
- port
- PY3
- platform
- FPGAVersion
- FPGANum
- FPGAboardtype
- FPGArad
- FPGAsensor
- FPGAinterfaces
- FPGAinvalid
- · iplist
- packageroot
- armed
- · senstiming
- sensmanual
- inittime
- padToFull
- abort
- verbmap
- logtag
- logcritbase
- logerrbase
- logwarnbase

- · loginfobase
- logdebugbase
- · logcrit
- logerr
- logwarn
- loginfo
- logdebug
- · verblevel
- payloaderror
- sensor
- comms
- board

8.1.1 Detailed Description

```
Code to assemble correct code to manage FPGA, frame grabber, and sensor
Exposed methods:
    initialize() - initializes board registers and pots, sets up sensor
    reinitialize() - initialize board and sensors, restore last known timer settings
    reboot() - perform software reset of board and reinitialize
    getBoardInfo() - parses FPGA_NUM register to retrieve board description
    getRegister(regname) - retrieves contents of named register
    setRegister(regname, string) - sets named register to given value
    resolveSubreq(srname) - resolves alias and retrieves object associated with
    getSubregister(subregname) - return substring of register identified in board
      attribute 'subregname'
    setSubregister(subregname, valstring) - replace substring of register identified
     in board attribute 'subregname' with 'valstring'
    submitMessages(messages) - set registers or subregisters based on list of
     destination/payload tuples
    getPot(potname) - returns float (0 < value < 1) corresponding to integer stored
     in pot or monitor 'potname'
    setPot(potname, value) - 0 < value < 1; sets named pot to fixed-point number =
      'value' * (maximum pot value)
    getPotV(potname) - returns voltage setting of 'potname'
    setPotV(potname, voltage) - sets named pot to voltage
    getMonV(monname) - returns voltage read by monitor 'monname' (or monitor
     associated with given potname)
    readImgs() - calls arm() and readoff() functions
    deInterlace(frames, interlacing) - extract interlaced frames
    saveFrames(frames) - save image object as one file
    saveTiffs(frames) - save individual frames as tiffs
    saveNumpys(frames) - save individual frames as numpy data files
    dumpNumpy(datastream) - save datastream string to numpy file
    plotFrames(frames) - plot individual frames as tiffs
    checkCRC(string) - checks last four characters of string is valid CRC for rest
      of string
    checkRegSet(register, string) - test set and get register functions for named
     register
    initPowerCheck() - start timers for power continutity check
    powerCheck(delta) - check that board power has not failed
    dummyCheck(image, margin) - counts how many pixels differ from expected dummy
     sensor values by more than margin
    printBoardInfo() - print board information derived from FPGA_NUM register
    dumpRegisters() - return contents of all board registers
    dumpSubregisters() - return contents of all named board subregisters
    str2bytes(string) - convert hexadecimal string to byte string
    bytes2str(sequence) - convert byte sequence to hexadecimal string
    str2nparray(string) - convert string of hexadecimal values into uint16 array
```

```
flatten(llist) - flattens list of lists into single list
   getEnter(text) - print text, then wait for Enter keypress
    mmReadoff(waitflag, variation) - convenience function for MicroManager plugin
    setFrames(min, max) - select subset of frames for readoff
    setRows(min, max, fullsize) - select subset of rows for readoff
    generateFrames(data) - processes data stream from board into frames
   abortReadoff() - cancel readoff in wait-for-SRAM loop
   batchAquire() - fast acquire a finite series of images
    loadTextFrames() - load data sets previously saved as text and convert to frames
Includes aliases to board- and sensor- specific functions:
   Board functions
        initBoard() - initialize default board register settings and configures ADCs
       initPots() - configure default pot settings before image acquisition
       latchPots() - latch all pot settings into sensor
       initSensor() - register sensor, set default timing settings
       configADCs() - set default ADC configuration
       startCapture() - reads ADC data into SRAM
       disarm() - take camera out of waiting-for-trigger state
        readSRAM() - trigger read from SRAM
       waitForSRAM() - puts board in wait state until data are ready in SRAM
       clearStatus() - clear contents of status registers
       checkStatus() - print contents of status register as reversed bit string
       \verb|checkStatus2|() - print contents of status register 2 as reversed bit string|\\
        reportStatus() - print report on contents of status registers
       resetTimer() - reset on-board timer
       getTimer() - read on-board timer
       enableLED(status) - enable (default) or disable (status = 0) on-board LEDs
       setLED(LED#, status) - turn LED on (default) or off (status = 0)
        setPowerSave(status) - turn powersave functionality on (default) or off
          (status = 0)
        getPressure() - read on-board pressure sensor
       getTemp() - read on-board temperature sensor
        checkStatus() - read and return status bits in status register 1
       checkStatus2() - read and return status bits in status register 2
       clearStatus() - clear status registers 1 and 2
       reportStatus() - print out human-readable board status report based on
         status registers
    Sensor functions
       checkSensorVoltStat() - checks that jumper settings match sensor selection
       setTiming(side, sequencetuple, delay) - configure high-speed timing
       setArbTiming(side, sequencelist) - configure arbitrary high-speed timing
          sequence
        getTiming(side) - returns high speed timing settings from registers
       setManualShutters() - configures manual shutter timing
       getManualTiming() - returns manual shutter settings from registers
       sensorSpecific() - returns register settings specific to implemented sensor
       setInterlacing(ifactor) - sets interlacing factor
        setHighFullWell(flag) - controls High Full Well mode
       setZeroDeadTime(flag) - controls Zero Dead Time mode
       setTriggerDelay(delayblocks) - sets trigger delay
       parseReadoff(frames) - performs sensor-specific parsing and separation of
          images
    Comms functions
       sendCMD(pkt) - sends packet object via serial port
       arm() - configures software buffers & arms camera
       {\tt readoff()} - waits for data ready flag, then downloads image data
       writeSerial(cmdString) - submits string 'cmdstring' (usually string is
          preformed packet)
       readSerial(stringlength) - reads string of length 'stringlength' from serial
       closeDevice() - disconnect interface and release resources
Informational class variables:
   version - nsCamera software version
   FPGAVersion - firmware version (date)
   FPGANum - firmware implementation identifier
   FPGAboardtype - FPGA self-identified board type (should match 'boardname')
   FPGArad = Boolean indicating radiation-tolerant FPGA build
```

```
FPGAsensor = FPGA self-identified sensor family (should correspond to
    'sensorname')
FPGAinterfaces = FPGA self-identified interfaces (list should include
    'commname')
FPGAinvalid = invalid FPGA information in register
```

Definition at line 47 of file CameraAssembler.py.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 __init__()

```
def nsCameraAssembler.CameraAssembler.__init__ (
              self.
              boardname = "llnl_v4",
              commname = "GigE",
              sensorname = "icarus2",
              verbose = 4,
              port = None,
              ip = None,
              logfile = None,
              logtag = None )
Args:
   boardname: name of FPGA board: llnl_v1, llnl_v4
    commname: name of communication interface: rs422, gige
    sensorname: name of sensor: icarus, icarus2, daedalus
   verbose: optional, sets logging level
0: print no logging messages
1: print CRITICAL logging messages (camera will not operate, e.g.,
  unable to connect to board)
2: print ERROR logging messages (camera will not operate as directed,
  e.g., an attempt to set the timing mode has failed, but the camera
  is still operational)
3: print WARNING logging messages (camera will operate as directed, but
  perhaps not as expected, e.g., ca.setTiming('A', (9, 8), 1) may be
  programmed correctly, but the actual timing generated by the board
  will be {1} [9, 8, 9, 14, 9, 8, 9]
4: print INFO logging messages (operational messages from ordinary
  camera operation)
   port: optional integer
RS422: preselects comport for RS422, bypasses port search
GigE: preselect OrangeTree control port for GigE (ignored if ip option
  not also given)
    ip: optional string (e.g., '192.168.1.100')
GigE: bypasses network search and selects particular OrangeTree board -
  required for some operating systems
    logfile: optional string, name of file to divert console output
   errtag: suffix to add to logging labels
```

Definition at line 170 of file CameraAssembler.py.

```
170
                    def __init__(
171
                               self,
172
                              boardname="llnl_v4",
                              commname="GigE"
173
                              sensorname="icarus2",
174
175
                              verbose=4,
176
                              port=None,
177
                              ip=None,
178
                              logfile=None,
179
                              logtag=None,
180
                   ):
181
182
                              Args:
183
                                        boardname: name of FPGA board: llnl_v1, llnl_v4
184
                                        commname: name of communication interface: rs422, gige
185
                                        sensorname: name of sensor: icarus, icarus2, daedalus
                                        verbose: optional, sets logging level
186
187
                                                  0: print no logging messages
188
                                                   1: print CRITICAL logging messages (camera will not operate, e.g.,
189
                                                       unable to connect to board)
                                                   2: print ERROR logging messages (camera will not operate as directed,
190
191
                                                       e.g., an attempt to set the timing mode has failed, but the camera
192
                                                       is still operational)
                                                  3: print WARNING logging messages (camera will operate as directed, but
193
                                                        perhaps not as expected, e.g., ca.setTiming(^{\prime}A^{\prime}, (9, 8), 1) may be
194
195
                                                        programmed correctly, but the actual timing generated by the board
196
                                                        will be {1} [9, 8, 9, 14, 9, 8, 9]
                                                   4: print INFO logging messages (operational messages from ordinary
197
198
                                                       camera operation)
199
                                        port: optional integer
                                                  RS422: preselects comport for RS422, bypasses port search
200
201
                                                   {\tt GigE: preselect \ OrangeTree \ control \ port \ for \ GigE \ (ignored \ if \ ip \ option}
202
                                                       not also given)
                                        ip: optional string (e.g., '192.168.1.100')
    GigE: bypasses network search and selects particular OrangeTree board -
203
2.04
2.05
                                                        required for some operating systems % \left( \frac{1}{2}\right) =\left( \frac{1}{2}\right) \left( 
206
                                        logfile: optional string, name of file to divert console output
2.07
                                        errtag: suffix to add to logging labels
208
209
                              self.version = "2.1.1"
210
                              self.currtime = 0
211
                              self.oldtime = 0
                              self.trigtime = []
212
213
                              self.waited = []
214
                              self.read = []
215
                              self.unstringed = []
216
                              self.parsedtime = []
217
                              self.savetime = []
218
                              self.cycle = []
219
                              self.boardname = boardname.lower()
220
                             if self.boardname == "llnlv1":
221
                                        self.boardname = "llnl_v1"
222
                              if self.boardname == "llnlv4":
223
                                        self.boardname = "llnl_v4"
224
                              self.commname = commname.lower()
225
                              self.sensorname = sensorname.lower()
                              self.verbose = verbose
226
                              self.port = port
227
228
                              self.python, self.pyth1, self.pyth2, _, _ = sys.version_info
                              self.PY3 = self.python >= 3
229
230
                              self.platform = platform.system()
231
                              self.arch, _ = platform.architecture()
232
233
                              self.FPGAVersion = ""
                              self.FPGANum = ""
235
                              # FPGA information here and below populated during initialization using
236
                                        getBoardInfo
237
                              self.FPGAboardtype = ""
238
                              self.FPGArad = False
239
                              self.FPGAsensor = ""
240
                              self.FPGAinterfaces = []
241
                              # indicates invalid FPGA information in register# (0x80000001 accepted as valid)
242
243
                              self.FPGAinvalid = False
244
245
                              self.iplist = None
246
                              self.packageroot = os.path.dirname(inspect.getfile(CameraAssembler))
247
                              self.armed = False
2.48
249
                               # only one of these collections (senstiming, sensmanual) should be nonempty at
```

```
250
             # any given time
             self.senstiming = {} # preserve HST setting against possible power failure
self.sensmanual = [] # preserve manual timing
251
252
253
             self.inittime = 0
254
             self.padToFull = False
255
             self.abort = False
256
257
             self.verbmap = {
258
                  0: 99,
                  1: logging.CRITICAL,
259
                  2: logging.ERROR,
260
                  3: logging.WARNING,
261
262
                  4: logging.INFO,
                  5: logging.DEBUG,
263
264
265
             if logtag is None:
266
                  logtag = ""
             self.logtag = logtag
self.logcritbase = "CRITICAL" + self.logtag + ": "
267
268
             self.logerrbase = "ERROR" + self.logtag + ": "
269
             self.logwarnbase = "WARNING" + self.logtag + ": "
self.loginfobase = "INFO" + self.logtag + ": "
270
271
             self.logdebugbase = "DEBUG" + self.logtag + ": "
272
273
274
             self.logcrit = self.logcritbase + "[CA] "
             self.logerr = self.logerrbase + "[CA] "
self.logwarn = self.logwarnbase + "[CA] "
275
276
             self.loginfo = self.loginfobase + "[CA] "
277
2.78
             self.logdebug = self.logdebugbase + "[CA] "
279
280
             self.verblevel = self.verbmap.get(verbose, 5) # defaults to 5 for invalid entry
281
282
             if loafile:
                  logging.basicConfig(format="%(message)s", filename=logfile)
2.83
284
             else:
                 logging.basicConfig(format="%(message)s")
285
             logging.getLogger().setLevel(self.verblevel)
logging.getLogger("matplotlib.font_manager").disabled = True
286
287
288
             if ip:
289
290
                      iphex = socket.inet_aton(ip)
291
                  except socket.error:
292
293
                      logging.critical(self.logcrit + "CameraAssembler: invalid IP provided")
294
                      sys.exit(1)
                  ipnum = [0, 0, 0, 0]
295
296
                  for i in range(4):
297
                       if self.PY3:
298
                           ipnum[i] = iphex[i]
299
300
                           ipnum[i] = int(iphex[i].encode("hex"), 16)
301
                  self.iplist = ipnum
302
303
             self.payloaderror = False
304
             self.initialize()
305
```

8.1.3 Member Function Documentation

8.1.3.1 abortReadoff()

```
Simple abort command for readoff in waiting mode--does not interrupt download in progress. Requires external threading to function. WARNING: if not intercepted by active readoff command, will terminate next readoff command immediately at inception.

Args:

flag: Sets passive abort flag read by readoff command
Returns:

boolean: updated setting of flag
```

Definition at line 1937 of file CameraAssembler.py.

```
1937
         def abortReadoff(self, flag=True):
1938
1939
             Simple abort command for readoff in waiting mode--does not interrupt download in
               progress. Requires external threading to function. WARNING: if not
1940
1941
                intercepted by active readoff command, will terminate next readoff command
1942
                immediately at inception.
1943
             Args:
                flag: Sets passive abort flag read by readoff command
1944
1945
             Returns:
            boolean: updated setting of flag
1946
1947
            self.abort = flag
1948
1949
            return flag
1950
```

8.1.3.2 arm()

Definition at line 416 of file CameraAssembler.py.

```
416 def arm(self, mode=None):
417 return self.comms.arm(mode)
418
```

8.1.3.3 batchAcquire()

```
DO NOT USE unless providing a varying value (a fixed prefix will cause
       overwriting)
    showProgress: if non-zero, show notice every 'showProgress' acquisitions and
      print total acquisition time
Returns:
    Time taken for acquisition (seconds)
Definition at line 1951 of file CameraAssembler.py.
1951
         def batchAcquire(
1952
             self,
1953
             sets=1.
1954
             trig="Hardware".
             path=None,
filename="Frame",
1955
1956
             prefix=None,
1957
1958
             showProgress=0.
1959
         ):
1960
             Acquire a series of images as fast as possible, then process and save to disk.
1961
1962
1963
             Args:
1964
                 sets: Number of acquisitions to perform
1965
                 path: save path, defaults to './output'
1966
                 filename: defaults to 'frames.bin'
                 prefix: prepended to filename, defaults to time/date (e.g. '160830-124704_')
1967
1968
                   DO NOT USE unless providing a varying value (a fixed prefix will cause
1969
1970
                 \verb|showProgress| if non-zero|, show notice every 'showProgress' acquisitions and
1971
                   print total acquisition time
1972
1973
             Time taken for acquisition (seconds)
             Returns:
1974
1975
1976
             datalist = ["0"] * sets
1977
             timelist = [datetime.now()] * sets
1978
             logging.info(
1979
                 self.loginfo
1980
                 + "batchAcquire: temporarily disabling warning and information "
1981
1982
1983
             logging.getLogger().setLevel(self.verbmap.get(2))
1984
             beforeread = time.time()
1985
             for i in range(sets):
1986
                 if showProgress and not (i + 1) % showProgress:
                     print(self.loginfo + "batchAcquire: Acquiring set " + str(i + 1))
1987
1988
                 self.arm(trig)
1989
                 data, datalen, data_err = self.readoff(fast=True)
1990
                 datalist[i] = data
1991
                 timelist[i] = datetime.now()
             afterread = time.time()
1992
1993
             if showProgress:
1994
                 print(
1995
                     self.loginfo
1996
                     + "batchAcquire: "
                     + str(afterread - beforeread)
+ " seconds for "
1997
1998
1999
                     + str(sets)
                     + " sets"
2000
2001
                 )
2002
             setnum = 0
2003
             if path is None:
                 path = os.path.join(os.getcwd(), "output")
2004
2005
             for (imset, imtime) in zip(datalist, timelist):
                 setnum = setnum + 1
2006
2007
                 if showProgress and not setnum % showProgress:
                     print(self.loginfo + "batchAcquire: Saving set " + str(setnum))
2008
                 parsed = self.generateFrames(imset)
2009
                 if prefix is None:
2010
                     setprefix = imtime.strftime("%y%m%d-%H%M%S%f")[:-2] + "_"
2011
2012
                 else:
2013
                     setprefix = prefix
                 self.saveTiffs(parsed, path, filename, prefix=setprefix)
2014
2015
             logging.getLogger().setLevel(self.verblevel)
2016
             logging.info(self.loginfo + "batchAcquire: re-enabling logging")
2017
             return afterread - beforeread
```

2018

8.1.3.4 bytes2str()

```
def nsCamera.CameraAssembler.CameraAssembler.bytes2str (
               bytesequence )
Python-version-agnostic converter of bytes to hexadecimal strings
Args:
    bytesequence: sequence of bytes as string (Py2) or bytes (Py3)
    hexadecimal string representation of 'bytes' without '0x'
Definition at line 1686 of file CameraAssembler.py.
1686
        def bytes2str(self, bytesequence):
1687
1688
            Python-version-agnostic converter of bytes to hexadecimal strings
1689
1690
                bytesequence: sequence of bytes as string (Py2) or bytes (Py3)
1691
1692
            Returns:
1693
            hexadecimal string representation of 'bytes' without '0x' """
1694
1695
1696
            estring = binascii.b2a_hex(bytesequence)
1697
           if self.PY3:
                estring = str(estring)[2:-1]
1698
1699
           return estring
1700
```

8.1.3.5 checkCRC()

Definition at line 1472 of file CameraAssembler.py.

```
def checkCRC(self, rval):
1473
1474
            Calculate CRC for rval[:-4] and compare with expected CRC in rval[-4:]
1475
1476
            Args:
1477
                rval: hexadecimal string
1478
1479
            Returns:
            boolean, True if CRCs match
1480
1481
1482
            data_crc = int(rval[-4:], base=16)
1483
            CRC_calc = crc16pure.crc16xmodem(self.str2bytes(rval[:-4]))
            return CRC_calc == data_crc
1484
1485
```

8.1.3.6 checkRegSet()

```
def nsCamera.CameraAssembler.CameraAssembler.checkRegSet (
                self,
                regname,
                teststring )
Quick check to confirm that data read from register matches data write
    regname: register to test
    teststring: value to assign to register, as hexadecimal string without '0x'
    boolean, True if read and write values match
Definition at line 1486 of file CameraAssembler.py.
1486
         def checkRegSet(self, regname, teststring):
1487
1488
             Quick check to confirm that data read from register matches data write
1489
1490
1491
                 regname: register to test
1492
                 teststring: value to assign to register, as hexadecimal string without {}^{\prime} Ox {}^{\prime}
1493
1494
1495
                 boolean, True if read and write values match
1496
1497
             self.setRegister(regname, teststring)
             # tell board to send data; wait to clear before interrogating register contents
if regname == "SRAM_CTL":
1498
1499
1500
                 time.sleep(2)
1501
                 if self.commname == "rs422":
1502
                     logging.info(
1503
                         self.loginfo + "skipping 'SRAM_CTL' register check for RS422"
1504
1505
                     return True
1506
             else:
1507
                 time.sleep(0.1)
1508
             temp = self.getRegister(regname)
1509
             resp = temp[1].upper()
1510
             if resp != teststring.upper():
                 logging.error(
1511
1512
                     self.logerr
1513
                     + "checkRegSet failure: "
1514
                     + regname
1515
                     + " ; set: "
1516
                     + teststring
                     + " ; read:
1517
1518
                     + resp
1519
1520
                 return False
1521
             return True
1522
```

8.1.3.7 checkSensorVoltStat()

```
def nsCamera.CameraAssembler.CameraAssembler.checkSensorVoltStat ( self \ )
```

Definition at line 377 of file CameraAssembler.py.

```
377     def checkSensorVoltStat(self):
378         return self.sensor.checkSensorVoltStat()
379
```

8.1.3.8 checkStatus()

```
\label{lem:cameraAssembler.CameraAssembler.checkStatus (} self \ )
```

Definition at line 362 of file CameraAssembler.py.

```
362     def checkStatus(self):
363         return self.board.checkStatus()
364
```

8.1.3.9 checkStatus2()

```
\label{eq:cameraAssembler.CameraAssembler.checkStatus2} \mbox{ (} \\ self \mbox{ )}
```

Definition at line 365 of file CameraAssembler.py.

```
365          def checkStatus2(self):
366          return self.board.checkStatus2()
367
```

8.1.3.10 clearStatus()

```
def nsCameraAssembler.CameraAssembler.clearStatus ( self \ )
```

Definition at line 359 of file CameraAssembler.py.

```
359     def clearStatus(self):
360         return self.board.clearStatus()
361
```

8.1.3.11 closeDevice()

```
def nsCamera.CameraAssembler.CameraAssembler.closeDevice ( self \ )
```

Definition at line 428 of file CameraAssembler.py.

```
428 def closeDevice(self):
429 return self.comms.closeDevice()
430
```

8.1.3.12 configADCs()

8.1.3.13 deInterlace()

1219

```
def nsCamera.CameraAssembler.CameraAssembler.deInterlace (
               self,
               frames.
               ifactor = 1)
Extracts interlaced frames. If interlacing does not evenly divide the height,
  remainder lines will be dropped
Args:
    frames: list of full-sized frames
    ifactor: interlacing factor; number of interlaced lines (generates
      ifactor + 1 images per frame)
Returns: list of deinterlaced frames
Definition at line 1183 of file CameraAssembler.py.
        def deInterlace(self, frames, ifactor=1):
1184
1185
            Extracts interlaced frames. If interlacing does not evenly divide the height,
1186
              remainder lines will be dropped
1187
            Args:
1188
                frames: list of full-sized frames
1189
                ifactor: interlacing factor; number of interlaced lines (generates
1190
                  ifactor + 1 images per frame)
1191
1192
            Returns: list of deinterlaced frames
            if ifactor == 0: # don't do anything
1194
1195
                return frames
1196
            warntrimmed = False
1197
            if self.padToFull:
1198
                newheight = self.sensor.maxheight // (ifactor + 1)
1199
                if newheight != (self.sensor.maxheight / (ifactor + 1)):
1200
                    warntrimmed = True
1201
            else:
                newheight = self.sensor.height // (ifactor + 1)
1202
1203
                if newheight != (self.sensor.height / (ifactor + 1)):
1204
                    warntrimmed = True
1205
1206
            if warntrimmed:
1207
                logging.warning(
1208
                    self.logwarn + "deInterlace: interlacing setting requires dropping of "
1209
                     "lines to maintain consistent frame sizes
1210
                )
            delaced = []
1211
1212
            for frame in frames:
                for sub in range(ifactor + 1):
1213
                    current = np.zeros((newheight, self.sensor.width), dtype=int)
1214
                    for line in range(newheight):
1215
                        current[line] = frame[(ifactor + 1) * line + sub]
1216
                    delaced.append(current)
1217
1218
            return delaced
```

8.1.3.14 disarm()

8.1.3.15 dummyCheck()

Definition at line 1553 of file CameraAssembler.py.

```
def dummyCheck(self, image, margin, dummyVals=None):
1554
1555
             Compare image with 'canonical' dummy sensor image (actual values estimated)
1556
1557
             Args:
1558
                 image: numpy array containing frame image
1559
                 margin: maxmimum allowed error for sensor
1560
                 dummyVals: condensed array of expected dummy sensor image values
1561
1562
             Returns:
1563
                 tuple, (number of pixels exceeding difference margin, numpy array containing
                   image subtracted from expected dummy image)
1564
1565
             if dummyVals is None:
1566
                 dummyVals = self.board.dummySensorVals
1567
             stripe0 = []
1568
             stripe1 = []
1569
1570
             for i in range(16):
1571
                 stripe0.append([dummyVals[0][i]] * 32)
1572
                 {\tt stripe1.append([dummyVals[1][i]] * 32)}
1573
            stripet = [val for sublist in stripe0 for val in sublist]
             striped = [val for sublist in stripel for val in sublist]
testVals = [stripet] * 512 + [stripeb] * 512
1574
1575
             testimage = np.array(testVals)
1576
1577
             if image.size == testimage.size:
1578
                 image.shape = (self.sensor.height, self.sensor.width)
1579
                 diff = testimage - image
                 diffabs = [abs(i) for sublist in diff for i in sublist]
1580
1581
                 bads = sum(1 for i in diffabs if i > margin)
1582
                 return bads, diff
1583
             else:
1584
                 logging.error(
                      self.logerr + "dummyCheck: Image size does not match dummy image; "
1585
1586
                      "returning zero, actual testimage "
1587
1588
                 return 0, testimage
1589
```

8.1.3.16 dumpNumpy()

```
def nsCamera.CameraAssembler.CameraAssembler.dumpNumpy (
               self,
               datastream,
               path = None,
               filename = "Dump",
               prefix = None )
Datastream is converted directly to numpy array and saved to disk. No attempt
  to parse headers or separate into individual frames is made.
Args:
    datastream: string to be saved
    path: save path, defaults to './output'
    filename: defaults to 'Dump'
    prefix: prepended to 'filename', defaults to time/date
       (e.g. '160830-124704_')
Returns:
    Error string
Definition at line 1389 of file CameraAssembler.py.
        def dumpNumpy(
1390
            self, datastream, path=None, filename="Dump", prefix=None,
1391
1392
1393
            Datastream is converted directly to numpy array and saved to disk. No attempt
              to parse headers or separate into individual frames is made.
1394
1395
1396
            Args:
1397
                datastream: string to be saved
1398
                path: save path, defaults to './output'
                filename: defaults to 'Dump'
prefix: prepended to 'filename', defaults to time/date
1399
1400
1401
                  (e.g. '160830-124704_')
1402
1403
            Returns:
            Error string
1404
1405
1406
            logging.info(self.loginfo + "dumpNumpy")
1407
            err =
1408
            if path is None:
1409
                path = os.path.join(os.getcwd(), "output")
1410
            if prefix is None:
                prefix = time.strftime("%y%m%d-%H%M%S_", time.localtime())
1411
1412
            if not os.path.exists(path):
1413
                os.makedirs(path)
1414
            npdata = self.str2nparray(datastream)
1415
                nppath = os.path.join(path, prefix + filename + ".npy")
1416
1417
                np.save(nppath, npdata)
1418
            except:
                err = self.logerr + "dumpNumpy: unable to save data stream"
1419
1420
                logging.error(err)
            return err
1421
1422
```

8.1.3.17 dumpRegisters()

```
def nsCameraAssembler.CameraAssembler.dumpRegisters ( self )
```

```
List contents of all registers in board.registers. WARNING: some status flags will reset when read.

DEPRECATED: use dumpStatus() instead

Returns:

Sorted list: [register name (register address) : register contents as hexadecimal string without '0x']
```

Definition at line 1621 of file CameraAssembler.py.

```
def dumpRegisters(self):
1623
              List contents of all registers in board.registers. WARNING: some status flags
1624
                will reset when read.
1625
             DEPRECATED: use dumpStatus() instead
1626
1627
             Returns:
1628
                 Sorted list: [register name (register address) : register contents as
1629
                   hexadecimal string without '0x']
1630
1631
             dump = \{\}
1632
             for key in self.board.registers.keys():
                 err, rval = self.getRegister(key)
dump[key] = rval
1633
1634
1635
             reglistmax = int(max(self.board.registers.values()), 16)
             dumplist = [0] * (reglistmax + 1)
1636
             for k, v in dump.items():
1637
                  regnum = self.board.registers[k]
1638
                  dumplist[int(regnum, 16)] = (
   "(" + regnum + ") {0:<24} {1}".format(k, v.upper())</pre>
1639
1640
1641
             reglist = [a for a in dumplist if a]
1642
1643
             return reglist
1644
```

8.1.3.18 dumpStatus()

```
\begin{tabular}{ll} def nsCamera.CameraAssembler.CameraAssembler.dumpStatus ( \\ self ) \end{tabular}
```

Definition at line 374 of file CameraAssembler.py.

```
374     def dumpStatus(self):
375         return self.board.dumpStatus()
376
```

8.1.3.19 dumpSubregisters()

Definition at line 1645 of file CameraAssembler.py.

```
def dumpSubregisters(self):
1646
             List contents of all subregisters in board.channel_lookups and
1647
1648
               board.monitor_lookups.
1649
             WARNING: some registers will reset when read- only the first subregister from
1650
               such a register will return the correct value, the remainder will return zeros
1651
1652
             DEPRECATED: use dumpStatus() instead
1653
1654
             Returns:
1655
                dictionary {subregister name : subregister contents as binary string
1656
                  without initial '0b'}
1657
1658
             dump = {}
1659
            for sub in self.board.subreglist:
1660
                 key = sub.name
1661
                 err, resp = self.getSubregister(key)
1662
                 if err:
1663
                    logging.warning(
                         self.logwarn + "dumpSubregisters: unable to read subregister " + key
1664
1665
1666
                val = hex(int(resp, 2))
                dump[key] = val
1667
            return dump
1668
1669
```

8.1.3.20 enableLED()

```
def nsCamera.CameraAssembler.CameraAssembler.enableLED ( self, \\ status = 1 \ )
```

Definition at line 341 of file CameraAssembler.py.

```
341     def enableLED(self, status=1):
342         return self.board.enableLED(status)
343
```

8.1.3.21 flatten()

```
def nsCamera.CameraAssembler.CameraAssembler.flatten ( self, \\ x \ )
```

Flatten list of lists into single list

Definition at line 1719 of file CameraAssembler.py.

```
1719 def flatten(self, x):
1720 """
1721 Flatten list of lists into single list
1722 """
1723 if isinstance(x, collections.Iterable):
1724 return [a for i in x for a in self.flatten(i)]
1725 else:
1726 return [x]
```

8.1.3.22 generateFrames()

```
def nsCamera.CameraAssembler.CameraAssembler.generateFrames (
                self,
                data )
Processes data stream from board into frames and applies sensor-specific
  parsing. Generates padded data for fullsize option of setRows.
Aras:
    data: stream from board.
Returns: list of parsed frames
Definition at line 1901 of file CameraAssembler.py.
         def generateFrames(self, data):
             Processes data stream from board into frames and applies sensor-specific
1903
1904
               parsing. Generates padded data for fullsize option of setRows.
1905
1906
             Args:
1907
                 data: stream from board.
1908
1909
             Returns: list of parsed frames
1910
1911
             allframes = self.str2nparray(data)
1912
             # self.oldtime = self.currtime
1913
             # self.currtime = time.time()
1914
             # self.unstringed.append(self.currtime - self.oldtime)
1915
             frames = [0] * self.sensor.nframes
            framesize = self.sensor.width * self.sensor.height
1916
1917
            if self.padToFull:
1918
                 toprows = self.sensor.firstrow
1919
                 botrows = (self.sensor.maxheight - 1) - self.sensor.lastrow
1920
                 for n in range(self.sensor.nframes):
1921
                     padtop = np.zeros(toprows * self.sensor.maxwidth, dtype=int)
padbot = np.zeros(botrows * self.sensor.maxwidth, dtype=int)
1922
1923
                     thisframe = np.concatenate(
                          (padtop, allframes[n \star framesize : (n + 1) \star framesize], padbot)
1924
1925
1926
                     frames[n] = thisframe
1927
             else:
1928
                 for n in range(self.sensor.nframes):
1929
                     frames[n] = allframes[n * framesize : (n + 1) * framesize]
1930
             self.clearStatus()
1931
            parsed = self.parseReadoff(frames)
1932
             # self.oldtime = self.currtime
1933
            # self.currtime = time.time()
1934
             # self.parsedtime.append(self.currtime - self.oldtime)
1935
            return parsed
1936
```

8.1.3.23 getBoardInfo()

Definition at line 570 of file CameraAssembler.py.

```
570
        def getBoardInfo(self):
571
572
            Get board info from FPGA_NUM register. Returns error flag if register contents
573
              are invalid and tuple (board version number, rad tolerance flag, sensor name)
574
575
            tuple (errorFlag, (board version, rad tolerance flag, sensor name))
576
577
578
            invalidFPGANum = False
579
            interfaces = []
580
581
            if int(self.FPGANum[0], 16) & 8:
                if self.FPGANum[1] == "1":
582
583
                    boardtype = "LLNLv1"
                elif self.FPGANum[1] == "4":
584
                    boardtype = "LLNLv4"
585
586
                else:
587
                    boardtype = "LLNLv?"
                    invalidFPGANum = True
588
589
            else:
                boardtype = "SNLrevC"
590
591
                logging.warning(
592
                    self.logwarn + "FPGA self-identifies as SNLrevC, which is not "
593
                     "supported by this software '
594
595
                invalidFPGANum = True
            self.FPGAboardtype = boardtype
596
597
598
            if int(self.FPGANum[6], 16) & 1:
599
                rad = True
600
            else:
                rad = False
601
602
            self.FPGArad = rad
603
            if self.FPGANum[7] == "1":
604
605
                sensor = "Icarus"
            elif self.FPGANum[7] == "2":
    sensor = "Daedalus"
606
607
            elif self.FPGANum[7] == "3":
608
609
                sensor = "Horus"
610
            else:
                sensor = "Undefined"
611
612
                invalidFPGANum = True
613
            self.FPGAsensor = sensor
614
615
            if int(self.FPGANum[5], 16) & 1:
616
                interfaces.append("RS422")
617
            if int(self.FPGANum[5], 16) & 2:
618
                interfaces.append("GigE")
619
            self.FPGAinterfaces = interfaces
620
621
            if invalidFPGANum:
                if self.FPGANum == "80000001":
622
623
                     invalidFPGANum = False
624
625
                    logging.warning(self.logwarn + "FPGA self-identification is invalid")
626
            self.FPGAinvalid = invalidFPGANum
627
            return invalidFPGANum, (boardtype, rad, sensor)
628
```

8.1.3.24 getEnter()

```
def nsCamera.CameraAssembler.CameraAssembler.getEnter ( self, text )

Wait for enter key to be pressed.

Args: text: message asking for keypress
```

Definition at line 1728 of file CameraAssembler.py.

```
def getEnter(self, text):
1729
1730
            Wait for enter key to be pressed.
1731
1732
            text: message asking for keypress
1733
1734
1735
            if self.PY3:
1736
                input(text)
1737
            else:
1738
                raw_input(text)
1739
```

8.1.3.25 getManualTiming()

```
\label{lem:cameraAssembler.cameraAssembler.getManualTiming (} self \ )
```

Definition at line 392 of file CameraAssembler.py.

```
def getManualTiming(self):
    return self.sensor.getManualTiming()
394
```

8.1.3.26 getMonV()

Definition at line 1122 of file CameraAssembler.py.

```
def getMonV(self, monname, errflag=False):
1122
1123
1124
             Reads voltage from monitor named or that associated with the pot named 'monname'
1125
1126
            Aras:
1127
                monname: name of pot or monitor, e.g., VRST or MON_CH2 found in
                  board.subreg_aliases or defined in board.subregisters
1128
1129
                errflag: if True, return tuple with error string
1130
            Returns:
1131
1132
                 if errflag:
                     tuple: (error string, float value of voltage measured by monitor)
1133
```

```
1134
1135
                     float value of voltage measured by monitor
1136
1137
            monname = monname.upper()
             if monname in self.board.subreg_aliases:
1138
1139
                 monname = self.board.subreg_aliases[monname].upper()
1141
            for key, value in self.board.monitor_controls.items():
1142
                 if value == monname:
1143
                    monname = key
             if monname not in self.board.monitor_controls:
                 if monname in self.board.subreglist:
1145
1146
                     pass # no change necessary
                 else:
1147
1148
                     err = (
1149
                         self.logerr + "getMonV: invalid lookup " + monname + ", returning 0"
1150
1151
                     logging.error(err)
                     if errflag:
1152
1153
                         return err, 0
                     return 0
1154
1155
             err, monval = self.getPot(monname, errflag=True)
             if err:
1156
1157
                 logging.error(
                     self.logerr + "getMonV: unable to read monitor value for " + monname
1158
1159
1160
             if self.board.ADC5 bipolar:
                 if monval >= 0.5:
1161
                     monval -= 1 # handle negative measurements (two's complement)
1162
1163
                 if errflag:
                     return err, 2 * self.board.ADC5_mult * monval * self.board.VREF
1164
                 return 2 * self.board.ADC5_mult * monval * self.board.VREF
1165
1166
             else:
1167
                 if errflag:
                 return err, self.board.ADC5_mult * monval * self.board.VREF
return self.board.ADC5_mult * monval * self.board.VREF
1168
1169
1170
```

8.1.3.27 getPot()

```
def nsCamera.CameraAssembler.CameraAssembler.getPot (
              self.
              potname,
              errflag = False )
Retrieves value of pot or ADC monitor subregister, scaled to [0,1).
Args:
    potname: name of pot or monitor, e.g., VRST or MON_CH2 found in
      board.subreq_aliases or defined in board.subregisters
    errflag: if True, return tuple with error string
Returns:
    if errflag:
tuple: (error string, float value of subregister, scaled to [0,1) )
float value of subregister, scaled to [0,1)
Definition at line 816 of file CameraAssembler.py.
       def getPot(self, potname, errflag=False):
816
817
818
           Retrieves value of pot or ADC monitor subregister, scaled to [0,1).
819
820
           Args:
               potname: name of pot or monitor, e.g., VRST or MON_CH2 found in
821
                board.subreg_aliases or defined in board.subregisters
822
```

```
823
                errflag: if True, return tuple with error string
824
825
            Returns:
               if errflag:
826
827
                    tuple: (error string, float value of subregister, scaled to [0,1) )
828
829
                   float value of subregister, scaled to [0,1)
830
831
            potname, potobj, _ = self.resolveSubreg(potname)
832
            if not potobj:
833
               err = (
                   self.logerr + "getPot: invalid lookup: " + potname + ' , returning "0" '
834
835
836
                logging.error(err)
837
                if errflag:
838
                   return err, "0"
                return "0"
839
840
            err, b_pot_value = self.getSubregister(potname)
841
            if err:
842
               logging.warning(
                    self.logerr + "getPot: unable to read subregister " + potname
843
844
845
            # convert binary string back to decimal
            f_reg_value = 1.0 * int(b_pot_value, 2)
846
847
            value = (f_reg_value - potobj.min) / (potobj.max - potobj.min)
            if errflag:
848
849
               return err, value
850
            return value
8.5.1
```

8.1.3.28 getPotV()

```
def nsCamera.CameraAssembler.CameraAssembler.getPotV (
               self,
              potname,
               errflag = False )
Reads voltage _setting_ (not actual voltage) of specified pot
Args:
    potname: name of pot or monitor, e.g., VRST or MON_CH2 found in
      board.subreg_aliases or defined in board.subregisters
    errflag: if True, return tuple with error string
Returns:
    if errflag:
tuple: (error string, float value of pot voltage)
    else:
float value of pot voltage
Definition at line 921 of file CameraAssembler.py.
921
       def getPotV(self, potname, errflag=False):
922
923
           Reads voltage _setting_ (not actual voltage) of specified pot
924
           Args:
925
               potname: name of pot or monitor, e.g., VRST or MON_CH2 found in
926
927
                board.subreg_aliases or defined in board.subregisters
928
               errflag: if True, return tuple with error string
929
930
           Returns:
              if errflag:
931
                  tuple: (error string, float value of pot voltage)
932
933
               else:
934
                  float value of pot voltage
935
```

```
936
            potname, potobj, _ = self.resolveSubreg(potname)
937
            if not potobj:
938
                err = (
939
                   self.logerr
940
                    + "getPotV: invalid lookup: "
                    + potname
+ ' , returning "0" '
941
942
943
944
                logging.error(err)
                if errflag:
                return err, "0"
return "0"
946
947
948
            err, val = self.getPot(potname, errflag=True)
            if err:
950
                logging.error(self.logerr + "getPotV: unable to read pot " + potname)
951
           minV = potobj.minV
            maxV = potobj.maxV
952
953
            if errflag:
                return err, val * (maxV - minV)
954
            return val * (maxV - minV)
955
956
```

8.1.3.29 getPressure()

return self.board.getPressure(offset, sensitivity, units)

8.1.3.30 getRegister()

357

358

635

636

```
def nsCamera.CameraAssembler.CameraAssembler.getRegister (
              self,
               regname )
Retrieves contents of named register as hexadecimal string without '0x'
Aras:
    regname: name of register as given in ICD
Returns:
    tuple: (error string, register contents as hexadecimal string without '0x')
Definition at line 630 of file CameraAssembler.py.
630
       def getRegister(self, regname):
631
           Retrieves contents of named register as hexadecimal string without '0x'
632
633
634
           Args:
```

regname: name of register as given in ICD

def nsCamera.CameraAssembler.CameraAssembler.getSubregister (

```
637
            tuple: (error string, register contents as hexadecimal string without '0x')
638
639
            regname = regname.upper()
640
641
            if regname not in self.board.registers:
642
643
                   self.logerr + "invalid register name: " + regname + "; returning zeros"
644
645
               logging.error(err)
                return err, "00000000"
            sendpkt = Packet(cmd="1", addr=self.board.registers[regname])
647
            err, rval = self.comms.sendCMD(sendpkt)
649
            if err:
650
               logging.error(self.logerr + "getRegister " + regname + " " + err)
651
            return err, rval[8:16]
```

8.1.3.31 getSubregister()

```
self.
                subregname )
Returns substring of register identified in board attribute 'subregname'
Args:
    subregname: listed in board.subreg_aliases or defined in board.subregisters
Returns:
    tuple: (error string, contents of subregister as binary string without '0b')
Definition at line 697 of file CameraAssembler.py.
        def getSubregister(self, subregname):
697
698
699
            Returns substring of register identified in board attribute 'subregname'
700
701
702
                subreqname: listed in board.subreq_aliases or defined in board.subreqisters
703
704
            Returns:
            tuple: (error string, contents of subregister as binary string without '0b')
705
706
707
            subregname, subregobj, _ = self.resolveSubreg(subregname)
            if not subregobj:
708
709
                err = (
710
                    self.logerr
711
                    + "getSubregister: invalid lookup: "
712
                    + subregname
                    + ' , returning "0" string '
713
714
                logging.error(err)
return err, "".zfill(8)
715
716
717
            err, resp = self.getRegister(subregobj.register)
718
            if err:
719
                logging.error(
720
                     self.logerr
721
                     + "getSubregister: unable to retrieve register setting: "
722
                     + subregname
723
                    + ' , returning "0" string'
724
                return err, "".zfill(8)
725
            hex_str = "0x" + resp # this should be a hexadecimalstring
b_reg_value = "{0:0=32b}".format(int(hex_str, 16)) # convert to binary string
726
727
728
            # list indexing is reversed from bit string; the last bit of the string is at
729
                index 0 in the list (thus bit 0 is at index 0)
730
            startindex = 31 - subregobj.start_bit
731
            return "", b_reg_value[startindex : startindex + subregobj.width]
732
```

8.1.3.32 getTemp()

return self.board.getTemp(scale)

8.1.3.33 getTimer()

354

355

```
def nsCameraAssembler.CameraAssembler.getTimer ( self )
```

Definition at line 335 of file CameraAssembler.py.

```
335     def getTimer(self):
336         return self.board.getTimer()
337
```

8.1.3.34 getTiming()

Definition at line 386 of file CameraAssembler.py.

```
def getTiming(self, side=None, actual=None):
return self.sensor.getTiming(side, actual)
388
```

8.1.3.35 initBoard()

```
\label{lem:cameraAssembler.CameraAssembler.initBoard (} self \ )
```

Aliases to other objects' methods.

Definition at line 308 of file CameraAssembler.py.

```
308     def initBoard(self):
309         return self.board.initBoard()
310
```

8.1.3.36 initialize()

```
def nsCamera.CameraAssembler.CameraAssembler.initialize ( self )
```

End aliases.

Initialize board registers and set pots

Definition at line 433 of file CameraAssembler.py.

```
433
        def initialize (self):
434
435
            Initialize board registers and set pots
436
437
438
440
441
            # get sensor
            if self.sensorname == "icarus":
442
443
                import nsCamera.sensors.icarus as snsr
            elif self.sensorname == "icarus2":
444
                 import nsCamera.sensors.icarus2 as snsr
445
            elif self.sensorname == "daedalus":
446
                import nsCamera.sensors.daedalus as snsr
447
448
            else: # catch-all for added sensors to attempt object encapsulation
                sensormodname = ".sensors." + self.sensorname
449
450
                    sensormod = importlib.import_module(sensormodname, "nsCamera")
451
452
                except ImportError:
                    logging.critical(self.logcrit + "invalid sensor name")
453
454
                     sys.exit(1)
455
                snsr = getattr(sensormod, self.sensorname)
456
            self.sensor = snsr(self)
457
            # kill existing connections (for reinitialize)
if hasattr(self, "comms"):
458
459
460
                self.closeDevice()
461
            # get communications interface
if self.commname == "rs422":
462
463
464
                import nsCamera.comms.RS422 as comms
465
            elif self.commname == "gige":
466
                import nsCamera.comms.GigE as comms
467
            else:
468
                commsmodname = ".comms." + self.commname
469
                 try:
470
                    commsmod = importlib.import_module(commsmodname, "nsCamera")
                 except ImportError:
471
472
                     logging.critical(self.logcrit + "invalid comms name")
473
                     sys.exit(1)
474
                 comms = getattr(commsmod, self.commname)
475
            self.comms = comms(self)
476
477
            # get board
478
            if self.boardname == "llnl_v1":
479
                 import nsCamera.boards.LLNL_v1 as brd
480
481
                self.board = brd.llnl_v1(self)
482
            elif self.boardname == "llnl_v4":
                import nsCamera.boards.LLNL_v4 as brd
483
484
485
                self.board = brd.llnl_v4(self)
486
            else:
                boardmodname = ".board." + self.boardname
487
488
489
                    boardmod = importlib.import module(boardmodname, "nsCamera")
490
                 except ImportError:
491
                     logging.critical(self.logcrit + "invalid board name")
492
                     svs.exit(1)
                boardobj = getattr(boardmod, self.boardname)
self.board = boardobj(self)
493
494
495
496
            # ###############
497
```

```
498
            # # For cython version
499
500
              if self.sensorname == "icarus":
501
502
                  import nsCamera.sensors.icarus as snsr
503
                  self.sensor = snsr.icarus(self)
504
             # elif self.sensorname == "icarus2":
505
                 import nsCamera.sensors.icarus2 as snsr
506
                  self.sensor = snsr.icarus2(self)
507
            # elif self.sensorname == "daedalus":
508
                  import nsCamera.sensors.daedalus as snsr
509
                  self.sensor = snsr.daedalus(self)
510
511
            # # kill existing connections (for reinitialize)
512
            # if hasattr(self, "comms"):
513
                  self.closeDevice()
514
            # # get communications interface
515
            # if self.commname == "rs422":
516
                  import nsCamera.comms.RS422 as comms
517
                  self.comms = comms.RS422(self)
518
519
            # elif self.commname == "gige":
520
                  import nsCamera.comms.GigE as comms
521
                  self.comms = comms.GigE(self)
522
            # # get board
523
            # if self.boardname == "llnl_v1":
524
525
                  import nsCamera.boards.LLNL_v1 as brd
526
                  self.board = brd.llnl v1(self)
            # elif self.boardname == "llnl_v4":
527
528
                  import nsCamera.boards.LLNL_v4 as brd
                  self.board = brd.llnl_v4(self)
529
            # ##############
530
531
            err, rval = self.getRegister("FPGA_NUM")
if err or rval == "":
532
533
                err, rval = self.getRegister("FPGA_NUM")
if err or rval == "":
534
535
                    logging.critical(
536
                         self.logcrit + "Initialization failed: unable to communicate with "
537
538
                         "board. "
539
                    )
                sys.exit(1)
540
541
542
            self.initBoard()
543
            self.initPots()
544
            self.initSensor()
545
            self.initPowerCheck()
546
            self.getBoardInfo()
547
            self.printBoardInfo()
548
```

8.1.3.37 initPots()

```
def nsCamera.CameraAssembler.CameraAssembler.initPots ( self )
```

Definition at line 311 of file CameraAssembler.py.

```
311     def initPots(self):
312     return self.board.initPots()
313
```

8.1.3.38 initPowerCheck()

```
\label{eq:cameraAssembler.CameraAssembler.initPowerCheck (} self \; )
```

Reset software and board timers for monitoring power status

Definition at line 1523 of file CameraAssembler.py.

```
def initPowerCheck(self):

1524 """

1525 Reset software and board timers for monitoring power status

1526 """

1527 self.inittime = time.time()

1528 logging.info(self.loginfo + "resetting timer for power check function")

1529 self.resetTimer()
```

8.1.3.39 initSensor()

```
\label{eq:cameraAssembler.CameraAssembler.initSensor} \mbox{ (} \\ self \mbox{ )}
```

Definition at line 317 of file CameraAssembler.py.

```
317     def initSensor(self):
318         return self.board.initSensor()
```

8.1.3.40 latchPots()

```
\label{lem:cameraAssembler.CameraAssembler.latchPots (} self \ )
```

Definition at line 314 of file CameraAssembler.py.

```
314          def latchPots(self):
315          return self.board.latchPots()
316
```

8.1.3.41 loadTextFrames()

```
def nsCamera.CameraAssembler.CameraAssembler.loadTextFrames (
               filename = 'frames.txt',
               path = None)
Load a image set previously saved as text and convert to frames. NOTE: to work
  properly, the cameraAssembler object must have the same geometry and sensor
  tyoe that was used to create the text file
Args:
    filename: name of textfile to load
    path: path to file, if not the current working directory
       Returns: list of parsed frames
Definition at line 2019 of file CameraAssembler.py.
2019
        def loadTextFrames(self, filename='frames.txt', path=None):
2020
2021
            Load a image set previously saved as text and convert to frames. NOTE: to work
2022
              properly, the camera\mbox{\sc Assembler} object must have the same geometry and sensor
2023
              tyoe that was used to create the text file
2024
2025
2026
                filename: name of textfile to load
2027
                path: path to file, if not the current working directory
2028
2029
           Returns: list of parsed frames
2030
2031
            if path is None:
2032
                path = os.path.join(os.getcwd())
2033
            textfile = os.path.join(path, filename)
2034
2035
                f = open(textfile, "r")
2036
                s = f.read()
2037
2038
                frames = self.generateFrames(s)
                return frames
2039
2040
            except OSError as err:
2041
               print("OS error: {0}".format(err))
2042
            except ValueError:
2043
               print ("Could not convert data to an integer.")
2044
            except:
2045
               print("Unexpected error:", sys.exc_info()[0])
2046
2047
2048 """
2049 Copyright (c) 2022, Lawrence Livermore National Security, LLC. All rights reserved.
2050 LLNL-CODE-838080
2051
2052 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
2053 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy
2054 (DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
2055 See license for disclaimers, notice of U.S. Government Rights and license terms and
2056 conditions.
2057 """
```

8.1.3.42 mmReadoff()

```
Convenience function for parsing frames for use by MicroManager plugin
Args:
     waitOnSRAM: readoff wait flag
     variation: format of frames generated from readoff
default - return first frame only
"LastFrame" - return last frame only
"Average" - provide average of frames as single frame
"Landscape" - stitch frames together horizontally into single wide frame
Returns:
    ndarray - single image frame
Definition at line 1740 of file CameraAssembler.py.
         def mmReadoff(self, waitOnSRAM, variation=None):
1740
1741
1742
              Convenience function for parsing frames for use by MicroManager plugin
1743
              Args:
1744
                  waitOnSRAM: readoff wait flag
1745
                  variation: format of frames generated from readoff
                      default - return first frame only
"LastFrame" - return last frame only
1746
1747
                      "Average" - provide average of frames as single frame
"Landscape" - stitch frames together horizontally into single wide frame
1748
1749
1750
1751
             Returns:
             ndarray - single image frame
1752
1753
             frames, datalen, data_err = self.readoff(waitOnSRAM)
if variation == "LastFrame":
1754
1755
1756
                  return frames[self.sensor.nframes - 1]
```

return np.sum(frames, axis=0) // self.sensor.nframes

shaped = [np.reshape(frame, (1024, 512)) for frame in frames]

8.1.3.43 parseReadoff()

else:

1757

1758

1759

1760

1761

1762

1763

1764

```
def nsCamera.CameraAssembler.CameraAssembler.parseReadoff ( self, \\ frames \ )
```

return np.concatenate(shaped, axis=1)

Definition at line 410 of file CameraAssembler.py.

```
410 def parseReadoff(self, frames):
411 return self.sensor.parseReadoff(frames)
412
```

elif variation == "Average":

return frames[0]

elif variation == "Landscape":

8.1.3.44 plotFrames()

```
Plot frame or list of frames as individual graphs.

Args:
frames: numpy array or list of numpy arrays index: number to start frame numbering

Returns:
Error string
```

Definition at line 1423 of file CameraAssembler.py.

```
1423
         def plotFrames(self, frames, index=None):
1424
1425
             Plot frame or list of frames as individual graphs.
1426
1427
1428
                 frames: numpy array or list of numpy arrays
1429
                  index: number to start frame numbering
1430
1431
             Returns:
             Error string
1432
1433
1434
             logging.info(self.loginfo + "plotFrames")
             err = "
1435
1436
             if index is None:
1437
                 nframe = self.sensor.firstframe
1438
             else:
1439
                  nframe = index
1440
1441
             if type(frames) is not list:
                  frames = [frames]
1442
1443
1444
              \mbox{\tt\#} if this is a text string from fast readoff, do the numpy conversion now
             if isinstance(frames[0], str):
1445
                  frames = self.generateFrames(frames)
1446
1447
             framestemp = np.copy(frames)
for frame in framestemp:
1448
1449
1450
                  try:
1451
                      if self.padToFull:
1452
                          frame.shape = (
                               self.sensor.maxheight // (self.sensor.interlacing + 1),
1453
1454
                               self.sensor.maxwidth,
1455
                          )
1456
                      else:
1457
                          frame.shape = (
                               self.sensor.height // (self.sensor.interlacing + 1),
1458
1459
                               self.sensor.width,
1460
1461
                  except:
                      err = self.logerr + "plotFrames: unable to plot frame"
1462
1463
                      logging.error(err)
1464
                 plt.imshow(frame, cmap="gray")
name = "Frame %d" % nframe
1465
1466
                  plt.title(name)
1467
1468
                  plt.show()
1469
                  nframe += 1
1470
             return err
1471
```

8.1.3.45 powerCheck()

```
Check to see if board power has persisted since powerCheck was last initialized.
  Compares time elapsed since initialization against board's timer. If
  difference is greater than 'delta,' flag as False (power has likely failed)
Args:
    delta: difference in seconds permitted between software and board timers
Returns:
    boolean, 'True' means timer difference is less than 'delta' parameter;
     'False' indicates power failure
Definition at line 1531 of file CameraAssembler.py.
        def powerCheck(self, delta=10):
1532
1533
            Check to see if board power has persisted since powerCheck was last initialized.
1534
              Compares time elapsed since initialization against board's timer. If
              difference is greater than 'delta,' flag as False (power has likely failed)
1535
1536
1537
            Args:
1538
                delta: difference in seconds permitted between software and board timers
1539
1540
            Returns:
                boolean, 'True' means timer difference is less than 'delta' parameter;
1541
                         'False' indicates power failure
1542
1543
1544
           elapsed = time.time() - self.inittime
1545
            difference = abs(elapsed - self.getTimer())
1546
            if difference > delta:
1547
                logging.warning(
1548
                    self.logwarn + "powerCheck function has failed; may indicate current "
1549
                    "or recent power failure '
1550
                )
1551
            return difference < delta
```

8.1.3.46 printBoardInfo()

1552

```
def nsCamera.CameraAssembler.CameraAssembler.printBoardInfo ( self \ )
```

Definition at line 1590 of file CameraAssembler.py.

```
1590
        def printBoardInfo(self):
1591
             logging.info(
1592
                 self.loginfo
1593
                 + "Python version: "
1594
                 + str(self.python)
1595
1596
                 + str(self.pyth1)
1597
1598
                 + str(self.pyth2)
1599
1600
             logging.info(self.loginfo + "nsCamera software version: " + self.version)
1601
             logging.info(self.loginfo + "FPGA firmware version: " + self.FPGAVersion)
             logging.info(self.loginfo + "FPGA implementation: " + self.FPGANum)
1602
1603
            if self.FPGAinvalid:
1604
                 logging.info(self.loginfo + "FPGA information unavailable")
1605
                 logging.info(self.loginfo + "Board type: " + self.FPGAboardtype)
1606
                 logging.info(self.loginfo + "Rad-Tolerant: " + str(self.FPGArad))
1607
                 logging.info(self.loginfo + "Sensor family: " + self.FPGAsensor)
1608
1609
                 logging.info(
                     self.loginfo + "Available interfaces: " + ", ".join(self.FPGAinterfaces)
1610
1611
             if self.commname == "gige":
1612
                ci = self.comms.CardInfoP.contents
1613
                 ip = ".".join(str(e) for e in [b for b in ci.IPAddr])
1614
                 logging.info(
1615
                     self.loginfo + "GigE connected to " + ip + ":" + str(self.port)
1616
```

```
1617 )
1618 elif self.commname == "rs422":
1619 logging.info(self.loginfo + "RS422 connected to " + self.comms._port)
1620
```

8.1.3.47 readImgs()

Definition at line 1171 of file CameraAssembler.py.

```
def readImgs(self, waitOnSRAM=True, mode="Hardware"):
1172
1173
             Combines arm() and readoff() functions
1174
1175
1176
                tuple (list of numpy arrays, length of downloaded payload, payload error
1177
                  flag) returned by readoff
1178
1179
            logging.info(self.loginfo + "readImgs")
1180
            self.arm(mode)
            return self.readoff(waitOnSRAM)
1182
```

8.1.3.48 readoff()

Definition at line 419 of file CameraAssembler.py.

```
419     def readoff(self, waitOnSRAM=None, timeout=0, fast=None):
420     return self.comms.readoff(waitOnSRAM, timeout, fast)
421
```

8.1.3.49 readSerial()

Definition at line 425 of file CameraAssembler.py.

```
425 def readSerial(self, size, timeout=None):
426 return self.comms.readSerial(size, timeout)
427
```

8.1.3.50 readSRAM()

```
def nsCameraAssembler.CameraAssembler.readSRAM ( self )
```

Definition at line 329 of file CameraAssembler.py.

```
329     def readSRAM(self):
330     return self.board.readSRAM()
331
```

8.1.3.51 reboot()

```
def nsCamera.CameraAssembler.CameraAssembler.reboot ( self \ ) Perform soft reboot on board and reinitialize
```

Definition at line 563 of file CameraAssembler.py.

```
563 def reboot(self):
564 """
565 Perform soft reboot on board and reinitialize
566 """
567 self.board.softReboot()
568 self.reinitialize()
```

8.1.3.52 reinitialize()

553

554 555

556

```
def nsCamera.CameraAssembler.CameraAssembler.reinitialize (
               self )
Reinitialize board registers and pots, reinitialize sensor timing (if
  previously set)
Definition at line 549 of file CameraAssembler.py.
549
       def reinitialize(self):
550
           Reinitialize board registers and pots, reinitialize sensor timing (if
551
552
           previously set)
```

logging.info(self.loginfo + "reinitializing")

for side in self.senstiming: 557 self.setTiming(side, self.senstiming[side][0], self.senstiming[side][1]) 558 559 560 if self.sensmanual: # should be mutually exclusive with anything in senstiming 561 self.setManualShutters(self.sensmanual) 562

8.1.3.53 reportEdgeDetects()

self.initialize()

```
def nsCamera.CameraAssembler.CameraAssembler.reportEdgeDetects (
              self )
```

Definition at line 371 of file CameraAssembler.py.

```
def reportEdgeDetects(self):
372
            return self.board.reportEdgeDetects()
373
```

8.1.3.54 reportStatus()

```
def nsCamera.CameraAssembler.CameraAssembler.reportStatus (
              self )
```

Definition at line 368 of file CameraAssembler.py.

```
368
       def reportStatus(self):
369
            return self.board.reportStatus()
```

8.1.3.55 resetTimer()

```
\label{lem:cameraAssembler.CameraAssembler.resetTimer (} self \ )
```

Definition at line 338 of file CameraAssembler.py.

```
338     def resetTimer(self):
339         return self.board.resetTimer()
340
```

8.1.3.56 resolveSubreg()

Definition at line 675 of file CameraAssembler.py.

```
675
       def resolveSubreg(self, srname):
676
677
            Resolves subregister name or alias, returns object associated with subregister
678
             and flag indicating writability
679
680
681
               srname: name or alias of subregister
683
           tuple(subregister name string, associated object, writable flag)
684
685
            writable = False
686
            srname = srname.upper()
688
            if srname in self.board.subreg_aliases:
689
               srname = self.board.subreg_aliases[srname].upper()
690
            if srname in self.board.subreglist:
691
               srobj = getattr(self.board, srname)
               writable = getattr(self.board, srname).writable
692
693
           else:
694
               srobj = None
695
           return srname, srobj, writable
696
```

8.1.3.57 saveFrames()

```
def nsCamera.CameraAssembler.CameraAssembler.saveFrames (
               self,
               frames.
               path = None,
               filename = "frames",
               prefix = None)
Save list of numpy arrays to disk. If passed an unprocessed text string, convert
  to numpy before saving. Use 'prefix=""' for no prefix
Args:
    frames: numpy array or list of numpy arrays OR text string
    path: save path, defaults to './output'
    filename: defaults to 'frames.bin'
    prefix: prepended to filename, defaults to time/date (e.g. '160830-124704_')
Returns:
    Error string
Definition at line 1220 of file CameraAssembler.py.
1220
        def saveFrames(
            self, frames, path=None, filename="frames", prefix=None,
1221
1222
1223
1224
            Save list of numpy arrays to disk. If passed an unprocessed text string, convert
              to numpy before saving. Use 'prefix=""' for no prefix
1225
1226
1227
            Args:
1228
                frames: numpy array or list of numpy arrays OR text string
                path: save path, defaults to './output'
1229
1230
                filename: defaults to 'frames.bin'
1231
                prefix: prepended to filename, defaults to time/date (e.g. '160830-124704_')
1232
1233
            Returns:
            Error string
1234
1235
            logging.info(self.loginfo + "saveFrames")
1236
1237
1238
            if path is None:
1239
                path = os.path.join(os.getcwd(), "output")
1240
            if prefix is None:
                prefix = datetime.now().strftime("%y%m%d-%H%M%S%f")[:-5] + "_"
            if not os.path.exists(path):
1242
1243
                os.makedirs(path)
1244
1245
            if isinstance(frames[0], str):
1246
                filename = filename + ".txt"
                savefile = open(os.path.join(path, prefix + filename), "w+")
1247
1248
                savefile.write(frames)
1249
1250
1251
                filename = filename + ".bin"
1252
                stacked = np.stack(frames)
1253
                try:
1254
                    stacked = stacked.reshape(
1255
1256
                            self.sensor.nframes,
                            self.sensor.height // (self.sensor.interlacing + 1),
1257
1258
                            self.sensor.width.
1259
                        )
1260
                    )
                except Exception as e:
1261
                    err = self.logerr + "saveFrames: unable to save frames: " + str(e)
1262
                    logging.error(err)
1263
1264
                stacked.tofile(os.path.join(path, prefix + filename))
1265
1266
            return err
1267
```

8.1.3.58 saveNumpys()

```
def nsCamera.CameraAssembler.CameraAssembler.saveNumpys (
               self,
               frames,
               path = None,
               filename = "Frame",
               prefix = None,
               index = None)
Save numpy array or list of numpy arrays to disk as individual numpy data files,
  with frame number appended to filename.
Args:
    frames: numpy array or list of numpy arrays or single numpy array
    path: save path, defaults to './output'
    filename: defaults to 'Frame' followed by frame number
    prefix: prepended to 'filename', defaults to time/date
       (e.g. '160830-124704 ')
    index: number to start frame numbering
Returns:
    Error string
Definition at line 1329 of file CameraAssembler.py.
1329
        def saveNumpys(
            self, frames, path=None, filename="Frame", prefix=None, index=None,
1330
1331
1332
1333
            Save numpy array or list of numpy arrays to disk as individual numpy data files,
1334
              with frame number appended to filename.
1335
1336
1337
                frames: numpy array or list of numpy arrays or single numpy array
                path: save path, defaults to './output' filename: defaults to 'Frame' followed by frame number
1338
1339
                prefix: prepended to 'filename', defaults to time/date
1340
1341
                   (e.g. '160830-124704_')
1342
                index: number to start frame numbering
1343
1344
            Returns:
            Error string
1345
1346
1347
            logging.info(self.loginfo + "saveNumpys")
1348
            if path is None:
1349
1350
                path = os.path.join(os.getcwd(), "output")
1351
            if prefix is None:
                prefix = datetime.now().strftime("%y%m%d-%H%M%S%f")[:-5] + "_"
1352
1353
            if not os.path.exists(path):
1354
                os.makedirs(path)
1355
            if index is None:
1356
                nframe = self.sensor.firstframe
1357
            else:
1358
                nframe = index
1359
            if type(frames) is not list:
1360
                frames = [frames]
1361
1362
             # if this is a text string from fast readoff, do the numpy conversion now
1363
            if isinstance(frames[0], str):
1364
                 frames = self.generateFrames(frames)
1365
1366
            framestemp = np.copy(frames)
             for frame in framestemp:
1367
1368
                 try:
                     if self.padToFull:
1369
1370
                        frame.shape = (
                             self.sensor.maxheight // (self.sensor.interlacing + 1),
1371
1372
                             self.sensor.maxwidth,
1373
                        )
```

```
1374
                     else:
1375
                         frame.shape = (
1376
                             self.sensor.height // (self.sensor.interlacing + 1),
                             self.sensor.width,
1377
1378
1379
                     namenum = filename + "_%d" % nframe
1380
                     nppath = os.path.join(path, prefix + namenum + ".npy")
1381
                     np.save(nppath, frame)
1382
                     nframe += 1
1383
                 except:
1384
                     err = self.logerr + "saveNumpys: unable to save arrays"
1385
                     logging.error(err)
1386
1387
            return err
1388
```

8.1.3.59 saveTiffs()

```
def nsCamera.CameraAssembler.CameraAssembler.saveTiffs (
              self,
              frames.
              path = None,
              filename = "Frame",
              prefix = None,
              index = None)
Save numpy array or list of numpy arrays or single array to disk as individual
  tiffs, with frame number appended to filename.
   frames: numpy array or list of numpy arrays
   path: save path, defaults to './output'
   filename: defaults to 'Frame' followed by frame number
   prefix: prepended to 'filename', defaults to time/date
      (e.g. '160830-124704_')
    index: number to start frame numbering
Returns:
   Error string
```

Definition at line 1268 of file CameraAssembler.py.

```
1268
         def saveTiffs(
1269
             self, frames, path=None, filename="Frame", prefix=None, index=None,
1270
1271
1272
             Save numpy array or list of numpy arrays or single array to disk as individual
1273
               tiffs, with frame number appended to filename.
1274
1275
1276
                 frames: numpy array or list of numpy arrays
                  path: save path, defaults to './output'
1277
                  filename: defaults to 'Frame' followed by frame number
1278
                 prefix: prepended to 'filename', defaults to time/date (e.g. '160830-124704_')
1279
1280
1281
                 index: number to start frame numbering
1282
1283
             Returns:
             Error string
1284
1285
             logging.info(self.loginfo + "saveTiffs")
err = ""
1286
1287
             if path is None:
1288
1289
                 path = os.path.join(os.getcwd(), "output")
1290
             if prefix is None:
```

```
1291
                  prefix = datetime.now().strftime("%y%m%d-%H%M%S%f")[:-5] + "_"
1292
             if not os.path.exists(path):
1293
                  os.makedirs(path)
             if index is None:
1294
1295
                 nframe = self.sensor.firstframe
1296
1297
                 nframe = index
1298
1299
             if type(frames) is not list:
1300
                  frames = [frames]
1301
             # if this is a text string from fast readoff, do the numpy conversion now
1302
             if isinstance(frames[0], str):
1303
                  frames = self.generateFrames(frames)
1304
             framestemp = np.copy(frames)
for frame in framestemp:
1305
1306
1307
                  try:
1308
                      if self.padToFull:
1309
                          frame.shape = (
1310
                              self.sensor.maxheight // (self.sensor.interlacing + 1),
1311
                               self.sensor.maxwidth,
1312
                          )
1313
                      else:
1314
                          frame.shape = (
                               self.sensor.height // (self.sensor.interlacing + 1),
1315
1316
                               self.sensor.width,
                          )
1317
1318
                      frameimg = Image.fromarray(frame)
                      namenum = filename + "_%d" % nframe
tifpath = os.path.join(path, prefix + namenum + ".tif")
1319
1320
1321
                      frameimg.save(tifpath)
1322
                      nframe += 1
1323
                 except:
                      err = self.logerr + "saveTiffs: unable to save images"
1324
                      logging.error(err)
1325
1326
1327
            return err
1328
```

8.1.3.60 sendCMD()

```
def nsCamera.CameraAssembler.CameraAssembler.sendCMD ( self, \\ pkt \ )
```

Definition at line 413 of file CameraAssembler.py.

```
413 def sendCMD(self, pkt):
414 return self.comms.sendCMD(pkt)
415
```

8.1.3.61 sensorSpecific()

```
def nsCamera.CameraAssembler.CameraAssembler.sensorSpecific ( self \ )
```

Definition at line 395 of file CameraAssembler.py.

```
395    def sensorSpecific(self):
396        return self.sensor.sensorSpecific()
397
```

8.1.3.62 setArbTiming()

8.1.3.63 setFrames()

Definition at line 1765 of file CameraAssembler.py.

```
1765
         def setFrames(self, minframe=None, maxframe=None):
1766
1767
             Sets bounds on frames returned by board, inclusive (e.g., 0,3 returns four
1768
             frames). If called without parameters, resets to full set of frames.
1769
1770
1771
                 minframe: first frame to read from board
1772
                 maxframe: last frame to read from board
1773
1774
             Error string
1775
1776
1777
             if minframe is None:
1778
                 minframe = self.sensor.minframe
1779
             if maxframe is None:
1780
                 maxframe = self.sensor.maxframe
1781
1782
                 not isinstance(minframe, int)
1783
                 or minframe < self.sensor.minframe</pre>
1784
                 or minframe > maxframe
1785
                 or not isinstance (maxframe, int)
1786
                 or maxframe > self.sensor.maxframe
1787
             ):
1788
                 err = (
1789
                     self.logerr + "setFrames: invalid frame limits submitted. Frame "
1790
                     "selection remains unchanged. "
1791
1792
                 logging.error(err)
1793
                 return err
1794
1795
             initframe = hex(minframe)[2:].zfill(8)
1796
             finframe = hex(maxframe)[2:].zfill(8)
```

```
err1, _ = self.setRegister("FPA_FRAME_INITIAL", initframe)
1798
            err2, _ = self.setRegister("FPA_FRAME_FINAL", finframe)
1799
            self.sensor.firstframe = minframe
1800
            self.sensor.lastframe = maxframe
1801
            self.sensor.nframes = maxframe - minframe + 1
1802
            self.comms.payloadsize = (
1803
                self.sensor.width
1804
                * self.sensor.height
1805
                * self.sensor.nframes
1806
                * self.sensor.bytesperpixel
1807
1808
            plural = ""
1809
            if self.sensor.nframes > 1:
1810
                plural = "s"
1811
            logging.info(
1812
                self.loginfo
                + "Readoff set to "
1813
                + str(self.sensor.nframes)
1814
                + " frame"
1815
1816
                + plural
1817
1818
                + str(minframe)
1819
1820
                + str(maxframe)
1821
                 + ")"
1822
1823
            err = err1 + err2
1824
            if err:
                logging.error(
1825
                    self.logerr + "setFrames may not have functioned properly: " + err
1826
1827
                )
1828
            return err
1829
```

8.1.3.64 setHighFullWell()

```
def nsCamera.CameraAssembler.CameraAssembler.setHighFullWell ( self, \\ flag = True \ )
```

Definition at line 401 of file CameraAssembler.py.

8.1.3.65 setInterlacing()

Definition at line 398 of file CameraAssembler.py.

```
398     def setInterlacing(self, ifactor=None):
399     return self.sensor.setInterlacing(ifactor)
400
```

8.1.3.66 setLED()

346

8.1.3.67 setManualShutters()

```
def nsCameraAssembler.CameraAssembler.setManualShutters ( self, \\ timing = None \ )
```

Definition at line 389 of file CameraAssembler.py.

```
389 def setManualShutters(self, timing=None):
390 return self.sensor.setManualShutters(timing)
391
```

8.1.3.68 setPot()

```
def nsCamera.CameraAssembler.CameraAssembler.setPot (
              self,
              potname,
              value = 1.0,
              errflag = False )
Sets value of pot to value, normalized so that '1.0' corresponds with the
  fixed point maximum value of pot.
Args:
   potname: common name of pot, e.g., VRST found in board.subreg_aliases or
      defined in board.subregisters
   value: float between 0 and 1
   errflag: if True, return tuple with error string
Returns:
   if errflag:
tuple: (error string, response packet as string)
   else:
response packet as string
```

```
Definition at line 852 of file CameraAssembler.py.
        def setPot(self, potname, value=1.0, errflag=False):
853
854
            Sets value of pot to value, normalized so that ^{\prime}1.0^{\prime} corresponds with the
855
              fixed point maximum value of pot.
856
857
            Args:
858
               potname: common name of pot, e.g., VRST found in board.subreg_aliases or
859
                  defined in board.subregisters
                value: float between 0 and 1
861
                errflag: if True, return tuple with error string
862
863
            Returns:
864
                if errflag:
865
                    tuple: (error string, response packet as string)
866
867
                    response packet as string
868
869
            if value < 0:</pre>
870
                value = 0.0
871
            if value > 1:
872
                value = 1.0
873
874
            potname, potobj, writable = self.resolveSubreg(potname)
875
            if not potobj:
876
                err = (
                    self.logerr + "setPot: invalid lookup: " + potname + ' , returning "0" '
877
878
879
                logging.error(err)
880
                if errflag:
                return err, "0"
return "0"
881
882
883
            if not writable:
                err = self.logerr + "setPot: not a writable subregister: " + potname
884
885
                logging.error(err)
886
                if errflag:
                    return err, "0"
887
888
                return 0
889
            setpoint = int(round(value * potobj.max_value))
            setpointpadded = "{num:{fill}{width}b}".format(
890
891
                num=setpoint, fill="0", width=potobj.width
892
893
            err, rval = self.setSubregister(potname, setpointpadded)
894
            if err:
895
                logging.error(
896
                     self.logerr
897
                     + "setPot: unable to confirm setting of subregister: "
898
                     + potname
899
900
            ident = potname[3:]
901
            if ident[0].isdigit(): # numbered pot scheme
902
                potnumlatch = int(ident) * 2 + 1
903
                potnumlatchstring = "{num:{fill}{width}x}".format(
904
                     num=potnumlatch, fill="0", width=8
905
906
                err1, resp = self.setRegister("POT_CTL", potnumlatchstring)
907
            else: # alphabetical DAC scheme
                ident = ident.upper() # expects single character, e.g. 'A' from 'DACA'
identnum = ord(ident) - ord("A") # DACA -> 0
908
909
910
                potnumlatch = int(identnum) * 2 + 1
                potnumlatchstring = "{num:{fill}{width}x}".format(
911
912
                     num=potnumlatch, fill="0", width=8
913
914
                err1, resp = self.setRegister("DAC_CTL", potnumlatchstring)
915
            if err1:
916
                logging.error(self.logerr + "setPot: unable to latch register")
917
            if errflag:
918
                return err + err1, rval
919
            return rval
```

8.1.3.69 setPotV()

```
\label{eq:cameraAssembler.CameraAssembler.setPotV} \mbox{ (} \\ self. \mbox{ }
```

```
potname,
               voltage,
               tune = False,
               accuracy = 0.01,
               iterations = 20,
               approach = 0.75,
               errflag = False )
Sets pot to specified voltage. If tune=True, uses monitor to adjust pot to
  correct voltage. Tuning will attempt to tune to closest LSB on pot; if
  'accuracy' > LSB resolution, will only complain if tuning is unable to get
  the voltage within 'accuracy'
Args:
    potname: common name of pot, e.g., VRST found in board.subreg_aliases or
      defined in board.subregisters
    voltage: voltage bound by pot max and min (set in board object)
    tune: if True, iterate with monitor to correct voltage
    accuracy: acceptable error in volts (if None, attempts to find closest
      possible pot setting and warns if last iteration does not reduce error
      below the resolution of the pot)
    iterations: number of iteration attempts
    approach: approximation parameter (>1 may cause overshoot)
    errflag: if True, return tuple with error string
Returns:
    if errflag:
tuple: (error string, response string)
    else:
response string
Definition at line 957 of file CameraAssembler.py.
957
       def setPotV(
958
           self.
959
           potname,
960
           voltage,
961
           tune=False,
962
           accuracy=0.01,
963
           iterations=20,
964
           approach=0.75,
965
           errflag=False,
966
967
968
           Sets pot to specified voltage. If tune=True, uses monitor to adjust pot to
969
             correct voltage. Tuning will attempt to tune to closest LSB on pot; if
             'accuracy' > LSB resolution, will only complain if tuning is unable to get the voltage within 'accuracy'
970
971
972
973
           Args:
974
               potname: common name of pot, e.g., VRST found in board.subreg_aliases or
975
                 defined in board.subregisters
976
               voltage: voltage bound by pot max and min (set in board object)
               tune: if True, iterate with monitor to correct voltage
977
978
               accuracy: acceptable error in volts (if None, attempts to find closest
979
                 possible pot setting and warns if last iteration does not reduce error
980
                 below the resolution of the pot)
981
               iterations: number of iteration attempts
982
               approach: approximation parameter (>1 may cause overshoot)
983
               errflag: if True, return tuple with error string
984
985
           Returns:
986
               if errflag:
987
                   tuple: (error string, response string)
988
               else:
                   response string
989
990
           potname, potobj, writable = self.resolveSubreg(potname)
991
992
           if not potobj:
993
               err = (
                   self.logerr
994
```

```
995
                    + "setPotV: invalid lookup: "
996
                    + potname
997
                       , returning "0" '
998
999
                logging.error(err)
1000
                 if errflag:
1001
                     return err, "0"
1002
                 return "0"
1003
             if not writable:
                 err = self.logerr + "setPotV: not a writable subregister: " + potname
1004
1005
                 logging.error(err)
1006
                 if errflag:
                 return err, "0"
return "0"
1007
1008
1009
             if voltage < potobj.minV:</pre>
1010
                 voltage = potobj.minV
1011
             if voltage > potobj.maxV:
                 voltage = potobj.maxV
1012
1013
             setting = (voltage - potobj.minV) / (potobj.maxV - potobj.minV)
             err, rval = self.setPot(potname, setting, errflag=True)
1014
             time.sleep(0.1)
1015
1016
             if tune:
1017
                 if potname not in self.board.monitor_controls.values():
1018
                     err = (
1019
                         self.logerr
                         + "setPotV: pot '"
1020
1021
                         + potname
                         + "' does not have a corresponding monitor"
1022
1023
1024
                     logging.error(err)
1025
                     if errflag:
1026
                         return err, rval
1027
                     return rval
1028
                 self.setPot(potname, 0.65)
1029
                 time.sleep(0.2)
                 err1, mon65 = self.getMonV(potname, errflag=True)
1030
1031
                 self.setPot(potname, 0.35)
1032
                 time.sleep(0.2)
1033
                 err2, mon35 = self.getMonV(potname, errflag=True)
1034
                 # theoretical voltage range assuming linearity
                 potrange = (mon65 - mon35) / 0.3
1035
                 stepsize = potrange / (potobj.max_value + 1)
1036
1037
                 err += err1 + err2
1038
                 if err or potrange < 1:
                     err += " ERROR: [CA] setPotV: unable to tune pot " + potname
1039
1040
                     if potrange < 1: # potrange should be on the order of 3.3 or 5 volts
1041
                         err += "; monitor shows insufficient change with pot variation"
1042
                     logging.error(err)
                     if errflag:
1043
1044
                         return err, rval
1045
                     return rval
1046
                 potzero = 0.35 - (mon35 / potrange)
1047
                 potone = 1.65 - (mon65 / potrange)
                 if potzero < 0:</pre>
1048
1049
                     potzero = 0
1050
                 if potone > 1:
                     potone = 1
1051
1052
1053
                 if accuracy > stepsize:
1054
                     mindiff = accuracy
1055
                 else:
1056
                     mindiff = stepsize
1057
                 setting = potzero + (voltage / potone)
1058
                 self.setPot(potname, setting)
1059
                 lastdiff = 0
1060
                 smalladjust = 0
                 err3 = ""
                 for _ in range(iterations):
1062
1063
                     err3i, measured = self.getMonV(potname, errflag=True)
1064
                     if err3i:
1065
                         err3 = err3 + err3i +
                     diff = voltage - measured
1066
                     if abs(diff - lastdiff) < stepsize / 2:
1067
1068
                         if (
                             smalladjust > 12
1069
1070
                         ): # magic number for now; if it doesn't converge after several
1071
                                 tries, it never will, usually because the setting is pinned
1072
                                  to 0 or 1 and adjust can't change it
1073
                              logging.warning(
1074
                                  self.logwarn
1075
                                  + "setPotV: Tuning converged too slowly: pot "
```

```
1076
                                   + potname
1077
                                   + " set to "
1078
                                   + str(voltage)
1079
                                   + "V, monitor returns "
1080
                                   + str(measured)
1081
                                   + "V"
1082
1083
                              if errflag:
1084
                                  return "", rval
1085
                              return rval
1086
                          smalladjust += 1
1087
                      if not int(2 * diff / stepsize):
                         if errflag:
    return "", rval
1088
1089
1090
                          return rval
1091
                     adjust = approach * (diff / potrange)
                      setting += adjust
1092
                      if setting > 1:
1093
1094
                          setting = 1
                      elif setting < 0:</pre>
1095
1096
                          setting = 0
                      err1, rval = self.setPot(potname, setting, True)
1097
                      lastdiff = diff
1098
1099
                      time.sleep(0.2)
1100
                  err4, measured = self.getMonV(potname, errflag=True)
                 diff = voltage - measured
1101
                 # code will try to get to within one stepsize, but will only complain if it
1102
                 # doesn't get within mindiff
if int(diff / mindiff):
1103
1104
1105
                      logging.warning(
1106
                          self.logwarn
                          + "setPotV: pot "
1107
                          + potname
+ " set to "
1108
1109
                          + str(voltage)
1110
                          + "V, monitor returns "
1111
                          + str(measured)
1112
1113
1114
                 err += err1 + err2 + err3 + err4
1115
1116
             if err:
                 logging.error(self.logerr + "setPotV: errors occurred: " + err)
1117
             if errflag:
1118
1119
                 return err, rval
1120
             return rval
1121
```

8.1.3.70 setPowerSave()

```
def nsCamera.CameraAssembler.CameraAssembler.setPowerSave ( self, \\ status = 1 \ )
```

Definition at line 347 of file CameraAssembler.py.

```
347     def setPowerSave(self, status=1):
348         return self.board.setPowerSave(status)
349
```

8.1.3.71 setPPER()

Definition at line 350 of file CameraAssembler.py.

```
350     def setPPER(self, time=None):
351     return self.board.setPPER(time)
352
```

8.1.3.72 setRegister()

```
def nsCamera.CameraAssembler.CameraAssembler.setRegister (
               self,
               regname,
               regval )
Sets named register to given value as hexadecimal string without '0x'
Aras:
    regname: name of register as given in ICD
    regval: value to assign to register, as hexadecimal string without '0x'
Returns:
    tuple: (error string, response string)
Definition at line 653 of file CameraAssembler.py.
        def setRegister(self, regname, regval):
655
            Sets named register to given value as hexadecimal string without {}^{\prime}\text{Ox'}
656
657
658
               regname: name of register as given in ICD
659
               regval: value to assign to register, as hexadecimal string without '0x'
660
661
           tuple: (error string, response string)
662
663
664
           regname = regname.upper()
665
            if regname not in self.board.registers:
               err = self.logerr + "Invalid register name: " + regname
666
667
               logging.error(err)
               return err, "00000000"
668
           pkt = Packet(addr=self.board.registers[regname], data=regval)
669
670
            err, rval = self.comms.sendCMD(pkt)
671
           if err:
               logging.error(self.logerr + "setRegister " + regname + ": " + err)
672
673
            return err, rval
674
```

8.1.3.73 setRows()

```
Sets bounds on rows returned by board, inclusive (e.g., 0,1023 returns all 1024
  rows). If called without parameters, resets to full image size.
Args:
    minrow: first row to return from board
    maxrow: last row to return from board
    fullsize: if True, generate full size frames, padding collected rows with
    zeroes as necessary
Definition at line 1830 of file CameraAssembler.py.
         def setRows(self, minrow=0, maxrow=None, fullsize=False):
1831
1832
             Sets bounds on rows returned by board, inclusive (e.g., 0,1023 returns all 1024
1833
              rows). If called without parameters, resets to full image size.
1834
1835
1836
                minrow: first row to return from board
1837
                 maxrow: last row to return from board
1838
                 fullsize: if True, generate full size frames, padding collected rows with
1839
                zeroes as necessary
1840
            err = ""
1841
1842
            if maxrow is None:
                maxrow = self.sensor.maxheight - 1
1843
1844
            if (
1845
                not isinstance(minrow, int)
1846
                or minrow < 0
                or minrow > maxrow
1847
1848
                or not isinstance (maxrow, int)
1849
                or maxrow >= self.sensor.maxheight
1850
            ):
1851
                 err = (
                     self.logerr + "setRows: invalid row arguments submitted. Frame size "
1852
1853
                     "remains unchanged. '
1854
1855
                 logging.error(err)
1856
                 return err
1857
1858
            initrow = hex(minrow)[2:].zfill(8)
1859
            finrow = hex(maxrow)[2:].zfill(8)
1860
            err1, _ = self.setRegister("FPA_ROW_INITIAL", initrow)
1861
             err2, _ = self.setRegister("FPA_ROW_FINAL", finrow)
1862
            self.sensor.firstrow = minrow
            self.sensor.lastrow = maxrow
1863
1864
             self.sensor.height = maxrow - minrow + 1
1865
            self.comms.payloadsize = (
1866
                self.sensor.width
1867
                * self.sensor.height
1868
                 * self.sensor.nframes
1869
                 * self.sensor.bytesperpixel
1870
1871
1872
            if self.commname == "rs422":
1873
                self.comms._datatimeout = (
1874
                    (1.0 * self.sensor.height / self.sensor.maxheight)
1875
                     * 5e7
1876
                     * self.sensor.nframes
1877
                     / self.comms._baud
1878
1879
            if fullsize:
1880
1881
                self.padToFull = True
1882
             else:
1883
                self.padToFull = False
1884
             logging.info(
1885
                self.loginfo
1886
                 + "Readoff set to "
1887
                 + str(self.sensor.height)
1888
                + " rows ("
1889
                + str(minrow)
1890
1891
                 + str(maxrow)
                 + ")"
1892
1893
            )
            err = err1 + err2
1894
1895
            if err:
1896
                 logging.error(
1897
                     self.logerr + "setRows may not have functioned properly: " + err
```

```
1898 )
1899 return err
1900
```

8.1.3.74 setSubregister()

```
def nsCamera.CameraAssembler.CameraAssembler.setSubreqister (
                 self.
                 subregname,
                 valstring )
Sets substring of register identified in board attribute 'subregname' to
  valstring if subregister is writable
Aras:
     subregname: listed in board.subreg_aliases or defined in board.subregisters
     valstring: binary string without '0b'
Returns:
    tuple: (error, packet response string) from setRegister
Definition at line 733 of file CameraAssembler.py.
733
        def setSubregister(self, subregname, valstring):
734
735
             Sets substring of register identified in board attribute 'subregname' to
736
              valstring if subregister is writable
737
738
739
                 subregname: listed in board.subreg_aliases or defined in board.subregisters
740
                 valstring: binary string without '0b'
741
742
             tuple: (error, packet response string) from setRegister
743
744
745
             subregname, subregobj, writable = self.resolveSubreg(subregname)
746
             if not subregobj:
747
                 err = self.logerr + "setSubregister: invalid lookup: " + subregname
748
                 logging.error(err)
                 return err, "0"
749
750
             if not writable:
751
                 err = (
752
                     self.logerr
753
                      + "setSubregister: not a writable subregister: "
754
                     + subregname
755
756
                 logging.error(err)
                 return err, "0"
757
758
             if len(str(valstring)) > subregobj.width:
                 err = self.logerr + "setSubregister: replacement string is too long"
759
760
                 logging.error(err)
                 return err, "0"
761
             # read current value of register data
762
             err, resp = self.getRegister(subregobj.register)
763
764
             if err:
765
                 logging.error(
766
                     self.logerr + "setSubregister: unable to retrieve register setting; "
                     "setting of " + subregname + " likely failed)"
767
768
                 )
769
                 return err, "0"
            hex_str = "0x" + resp
b_reg_value = "{0:0=32b}".format(int(hex_str, 16)) # convert to binary
# list indexing is reversed from bit string; the last bit of the string is at
# index 0 in the list (thus bit 0 is at index 0)
770
771
772
773
774
             startindex = 31 - subregobj.start\_bit
             valstringpadded = str(valstring).zfill(subregobj.width)
775
776
             fullreg = list(b_reg_value)
```

```
fullreg[startindex : startindex + subregobj.width] = valstringpadded
# convert binary string back to hexadecimal string for writing
new_reg_value = "".join(fullreg)
h_reg_value = "{num:{fill}{width}x}".format(
num=int(new_reg_value, 2), fill="0", width=8
}
return self.setRegister(subregobj.register, h_reg_value)
```

8.1.3.75 setTiming()

```
def nsCamera.CameraAssembler.CameraAssembler.setTiming (
    self,
    side = None,
    sequence = None,
    delay = None )
```

Definition at line 380 of file CameraAssembler.py.

```
def setTiming(self, side=None, sequence=None, delay=None):
return self.sensor.setTiming(side, sequence, delay)
382
```

8.1.3.76 setTriggerDelay()

```
def nsCameraAssembler.CameraAssembler.setTriggerDelay ( self, \\ delayblocks = 0 \ )
```

Definition at line 407 of file CameraAssembler.py.

```
407 def setTriggerDelay(self, delayblocks=0):
408 return self.sensor.setTriggerDelay(delayblocks)
409
```

8.1.3.77 setZeroDeadTime()

Definition at line 404 of file CameraAssembler.py.

```
404     def setZeroDeadTime(self, flag=True):
405         return self.sensor.setZeroDeadTime(flag)
406
```

8.1.3.78 startCapture()

8.1.3.79 str2bytes()

328

Definition at line 1670 of file CameraAssembler.py.

```
def str2bytes(self, astring):
1672
            Python-version-agnostic converter of hexadecimal strings to bytes
1673
1674
1675
               astring: hexadecimal string without '0x'
1676
1677
            Returns:
            byte string equivalent to input string
1678
1679
           if self.PY3:
1680
               dbytes = binascii.a2b_hex(astring)
1681
1682
           else:
1683
              dbytes = astring.decode("hex")
1684
            return dbytes
1685
```

8.1.3.80 str2nparray()

```
Definition at line 1701 of file CameraAssembler.py.
```

```
def str2nparray(self, valstring):
1702
1703
             Convert string into array of uint16s
1704
1705
1706
                 valstring: string of hexadecimal characters
1707
1708
            numpy array of uint16
1709
1710
1711
            stringlen = len(valstring)
1712
            arraylen = int(stringlen / 4)
1713
            outarray = np.empty(int(arraylen), dtype="uint16")
1714
1715
            for i in range(0, arraylen):
                outarray[i] = int(valstring[4 * i : 4 * i + 4], 16)
1716
1717
             return outarray
1718
```

def nsCamera.CameraAssembler.CameraAssembler.submitMessages (

8.1.3.81 submitMessages()

808

810

811

813

814

815

```
messages,
               errorstring = "Error" )
Serially set multiple register / subregister values
Aras:
    messages: list of tuples (register name, hexadecimal string without '0x')
      and/or (subregister name, binary string without '0b')
    errorstring: error message to print in case of failure
Returns:
    tuple (accumulated error string, response string of final message)
Definition at line 785 of file CameraAssembler.py.
        def submitMessages(self, messages, errorstring="Error"):
786
787
           Serially set multiple register / subregister values
788
789
           Args:
790
               messages: list of tuples (register name, hexadecimal string without '0x')
791
                and/or (subregister name, binary string without '0b')
               errorstring: error message to print in case of failure
792
793
794
           Returns:
           tuple (accumulated error string, response string of final message)
795
796
           errs = ""
797
           err = ""
798
799
           rval = ""
           for m in messages:
800
               if m[0].upper() in self.board.registers:
801
                   err, rval = self.setRegister(m[0].upper(), m[1])
802
               elif m[0].upper() in self.board.subreglist:
803
804
                   err, rval = self.setSubregister(m[0].upper(), m[1])
               else:
805
                   err = (
806
                       self.logerr
807
```

+ "submitMessages: Invalid register/subregister: "

+ errorstring

+ m[0]

errs = errs + err

return err, rval

logging.error(err)

8.1.3.82 waitForSRAM()

Definition at line 332 of file CameraAssembler.py.

```
332 def waitForSRAM(self, timeout=None):
333 return self.board.waitForSRAM(timeout)
334
```

8.1.3.83 writeSerial()

Definition at line 422 of file CameraAssembler.py.

```
422 def writeSerial(self, cmd, timeout=None):
423 return self.comms.writeSerial(cmd, timeout)
424
```

8.1.4 Member Data Documentation

8.1.4.1 abort

nsCamera.CameraAssembler.CameraAssembler.abort

Definition at line 245 of file CameraAssembler.py.

8.1.4.2 armed

nsCamera.CameraAssembler.CameraAssembler.armed

Definition at line 237 of file CameraAssembler.py.

8.1.4.3 board

nsCamera.CameraAssembler.CameraAssembler.board

Definition at line 481 of file CameraAssembler.py.

8.1.4.4 boardname

nsCamera.CameraAssembler.CameraAssembler.boardname

Definition at line 209 of file CameraAssembler.py.

8.1.4.5 commname

nsCamera.CameraAssembler.CameraAssembler.commname

Definition at line 214 of file CameraAssembler.py.

8.1.4.6 comms

nsCamera.CameraAssembler.CameraAssembler.comms

Definition at line 475 of file CameraAssembler.py.

8.1.4.7 currtime

nsCamera.CameraAssembler.CameraAssembler.currtime

Definition at line 200 of file CameraAssembler.py.

8.1.4.8 cycle

 $\verb|nsCameraAssembler.CameraAssembler.cycle| \\$

Definition at line 208 of file CameraAssembler.py.

8.1.4.9 FPGAboardtype

 $\verb|nsCameraAssembler.CameraAssembler.FPGAboardtype| \\$

Definition at line 227 of file CameraAssembler.py.

8.1.4.10 FPGAinterfaces

 $\verb|nsCamera Assembler.Camera Assembler.FPGA interfaces|$

Definition at line 230 of file CameraAssembler.py.

8.1.4.11 FPGAinvalid

nsCamera.CameraAssembler.CameraAssembler.FPGAinvalid

Definition at line 233 of file CameraAssembler.py.

8.1.4.12 FPGANum

nsCamera.CameraAssembler.CameraAssembler.FPGANum

Definition at line 224 of file CameraAssembler.py.

8.1.4.13 FPGArad

nsCamera.CameraAssembler.CameraAssembler.FPGArad

Definition at line 228 of file CameraAssembler.py.

8.1.4.14 FPGAsensor

nsCamera.CameraAssembler.CameraAssembler.FPGAsensor

Definition at line 229 of file CameraAssembler.py.

8.1.4.15 FPGAVersion

nsCamera.CameraAssembler.CameraAssembler.FPGAVersion

Definition at line 223 of file CameraAssembler.py.

8.1.4.16 inittime

nsCamera.CameraAssembler.CameraAssembler.inittime

Definition at line 243 of file CameraAssembler.py.

8.1.4.17 iplist

 $\verb|nsCamera.CameraAssembler.CameraAssembler.iplist|$

Definition at line 235 of file CameraAssembler.py.

8.1.4.18 logcrit

 $\verb|nsCamera.CameraAssembler.CameraAssembler.logcrit|\\$

Definition at line 264 of file CameraAssembler.py.

8.1.4.19 logcritbase

nsCamera.CameraAssembler.CameraAssembler.logcritbase

Definition at line 258 of file CameraAssembler.py.

8.1.4.20 logdebug

 $\verb|nsCameraAssembler.CameraAssembler.logdebug|$

Definition at line 268 of file CameraAssembler.py.

8.1.4.21 logdebugbase

 $\verb|nsCamera.CameraAssembler.CameraAssembler.logdebugbase|$

Definition at line 262 of file CameraAssembler.py.

8.1.4.22 logerr

nsCamera.CameraAssembler.CameraAssembler.logerr

Definition at line 265 of file CameraAssembler.py.

8.1.4.23 logerrbase

 $\verb|nsCameraAssembler.CameraAssembler.logerrbase| \\$

Definition at line 259 of file CameraAssembler.py.

8.1.4.24 loginfo

 $\verb|nsCamera.CameraAssembler.CameraAssembler.loginfo|\\$

Definition at line 267 of file CameraAssembler.py.

8.1.4.25 loginfobase

nsCamera.CameraAssembler.CameraAssembler.loginfobase

Definition at line 261 of file CameraAssembler.py.

8.1.4.26 logtag

 $\verb|nsCameraAssembler.CameraAssembler.logtag| \\$

Definition at line 257 of file CameraAssembler.py.

8.1.4.27 logwarn

nsCamera.CameraAssembler.CameraAssembler.logwarn

Definition at line 266 of file CameraAssembler.py.

8.1.4.28 logwarnbase

nsCamera.CameraAssembler.CameraAssembler.logwarnbase

Definition at line 260 of file CameraAssembler.py.

8.1.4.29 oldtime

 $\verb|nsCamera.CameraAssembler.CameraAssembler.oldtime| \\$

Definition at line 201 of file CameraAssembler.py.

8.1.4.30 packageroot

 $\verb|nsCamera.CameraAssembler.CameraAssembler.packageroot|\\$

Definition at line 236 of file CameraAssembler.py.

8.1.4.31 padToFull

 $\verb|nsCamera.CameraAssembler.CameraAssembler.padToFull|$

Definition at line 244 of file CameraAssembler.py.

8.1.4.32 parsedtime

nsCamera.CameraAssembler.CameraAssembler.parsedtime

Definition at line 206 of file CameraAssembler.py.

8.1.4.33 payloaderror

nsCamera.CameraAssembler.CameraAssembler.payloaderror

Definition at line 293 of file CameraAssembler.py.

8.1.4.34 platform

nsCamera.CameraAssembler.CameraAssembler.platform

Definition at line 220 of file CameraAssembler.py.

8.1.4.35 port

 $\verb|nsCamera.CameraAssembler.CameraAssembler.port|\\$

Definition at line 217 of file CameraAssembler.py.

8.1.4.36 PY3

nsCamera.CameraAssembler.CameraAssembler.PY3

Definition at line 219 of file CameraAssembler.py.

8.1.4.37 read

nsCamera.CameraAssembler.CameraAssembler.read

Definition at line 204 of file CameraAssembler.py.

8.1.4.38 savetime

 $\verb|nsCamera.CameraAssembler.CameraAssembler.savetime|\\$

Definition at line 207 of file CameraAssembler.py.

8.1.4.39 sensmanual

nsCamera.CameraAssembler.CameraAssembler.sensmanual

Definition at line 242 of file CameraAssembler.py.

8.1.4.40 sensor

nsCamera.CameraAssembler.CameraAssembler.sensor

Definition at line 456 of file CameraAssembler.py.

8.1.4.41 sensorname

nsCamera.CameraAssembler.CameraAssembler.sensorname

For regular version.

Definition at line 215 of file CameraAssembler.py.

8.1.4.42 senstiming

 $\verb|nsCamera.CameraAssembler.CameraAssembler.senstiming| \\$

Definition at line 241 of file CameraAssembler.py.

8.1.4.43 trigtime

 $\verb|nsCamera Assembler.Camera Assembler.trigtime| \\$

Definition at line 202 of file CameraAssembler.py.

8.1.4.44 unstringed

nsCamera.CameraAssembler.CameraAssembler.unstringed

Definition at line 205 of file CameraAssembler.py.

8.1.4.45 verblevel

nsCamera.CameraAssembler.CameraAssembler.verblevel

Definition at line 270 of file CameraAssembler.py.

8.1.4.46 verbmap

nsCamera.CameraAssembler.CameraAssembler.verbmap

Definition at line 247 of file CameraAssembler.py.

8.1.4.47 verbose

nsCamera.CameraAssembler.CameraAssembler.verbose

Definition at line 216 of file CameraAssembler.py.

8.1.4.48 version

nsCamera.CameraAssembler.CameraAssembler.version

Definition at line 199 of file CameraAssembler.py.

8.1.4.49 waited

nsCamera.CameraAssembler.CameraAssembler.waited

Definition at line 203 of file CameraAssembler.py.

The documentation for this class was generated from the following file:

nsCamera/CameraAssembler.py

8.2 nsCamera.sensors.daedalus.daedalus Class Reference

Public Member Functions

- def init (self, camassem)
- def checkSensorVoltStat (self)
- def sensorSpecific (self)
- def setInterlacing (self, ifactor)
- def setHighFullWell (self, flag)
- def setZeroDeadTime (self, flag)
- def setTriggerDelay (self, delayblocks)
- def setTiming (self, side, sequence, delay)
- def setArbTiming (self, side, sequence)
- def getTiming (self, side, actual)
- def setManualShutters (self, timing)
- def getManualTiming (self)
- def parseReadoff (self, frames)
- def reportStatusSensor (self, statusbits)

Public Attributes

- ca
- logcrit
- logerr
- logwarn
- loginfo
- logdebug
- minframe
- maxframe
- firstframe
- lastframe
- nframes
- maxwidth
- maxheight
- firstrow
- lastrow
- width
- height
- bytesperpixel
- fpganumID
- interlacing
- ZDT
- HFW
- sens_registers
- · sens_subregisters

8.2.1 Detailed Description

Definition at line 26 of file daedalus.py.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 init () def nsCamera.sensors.daedalus.daedalus.__init__ (camassem) Definition at line 27 of file daedalus.py. def __init__(self, camassem): self.ca = camassem 28 29 self.logcrit = self.ca.logcritbase + "[Daedalus] " self.logerr = self.ca.logerrbase + "[Daedalus] 30 self.logwarn = self.ca.logwarnbase + "[Daedalus] " 31 self.loginfo = self.ca.loginfobase + "[Daedalus] 32 self.logdebug = self.ca.logdebugbase + "[Daedalus] " 33 logging.info(self.loginfo + "initializing sensor object") 34 35 self.minframe = 0 36 self.maxframe = 237 38 self.firstframe = self.minframe self.lastframe = self.maxframe 39 self.nframes = self.maxframe - self.minframe + 1 40 self.maxwidth = 512 41 42 self.maxheight = 1024self.firstrow = 0 self.lastrow = self.maxheight - 1 43 44 self.width = self.maxwidth 45 46 self.height = self.maxheight self.bytesperpixel = 2 self.fpganumID = "2" # last nybble of FPGA_NUM 47 48 49 self.interlacing = 050 self.ZDT = False self.HFW = False 51 52 53 self.sens_registers = OrderedDict(54 5.5 "HST_READBACK_A_LO": "018", "HST_READBACK_A_HI": "019", 56 "HST_READBACK_B_LO": "01A" 57 "HST_READBACK_B_HI": "01B" 59 "HSTALLWEN_WAIT_TIME": "03F", 60 "FRAME_ORDER_SEL": "04B", "HST_TRIGGER_DELAY_DATA_LO": "120", 61 62 "HST_TRIGGER_DELAY_DATA_HI": "121", "HST_PHI_DELAY_DATA_LO": "122", "HST_PHI_DELAY_DATA_HI": "123", "HST_TRIG_DELAY_READBACK_LO": "125", "HST_TRIG_DELAY_READBACK_HI": "126", "HST_PHI_DELAY_READBACK_LO": "127", "HST_PHI_DELAY_READBACK_HI": "128", "HST_COUNT_TRIG": "130", "HST_DELAY_EN": "131", 70 "HST_TEST_PHI_EN": "132", "RSL_HFW_MODE_EN": "133", 71 "RSL_ZDT_MODE_R_EN": "135", 73 "RSL_ZDT_MODE_L_EN": "136", 74 "BGTRIMA": "137", 75 76 "BGTRIMB": "138", "COLUMN_TEST_EN": "139", "RSL_CONFIG_DATA_RO": "140", 78 "RSL_CONFIG_DATA_R1": "141", 79 80 "RSL_CONFIG_DATA_R2": "142", "RSL_CONFIG_DATA_R3": "143", 81 "RSL CONFIG DATA R4": "144", 82 "RSL_CONFIG_DATA_R5": "145" 83 "RSL_CONFIG_DATA_R6": "146", 84 "RSL_CONFIG_DATA_R7": "147", "RSL_CONFIG_DATA_R8": "148", 8.5

"RSL_CONFIG_DATA_R9": "149",

86

87

```
88
                     "RSL_CONFIG_DATA_R10": "14A",
                    "RSL_CONFIG_DATA_R11": "14B",
89
90
                     "RSL_CONFIG_DATA_R12": "14C"
                    "RSL_CONFIG_DATA_R13": "14D",
91
                     "RSL_CONFIG_DATA_R14": "14E",
                    "RSL_CONFIG_DATA_R15": "14F"
93
94
                     "RSL_CONFIG_DATA_R16": "150"
95
                    "RSL_CONFIG_DATA_R17": "151"
                     "RSL_CONFIG_DATA_R18": "152",
                    "RSL_CONFIG_DATA_R19": "153",
                     "RSL_CONFIG_DATA_R20": "154",
98
                     "RSL_CONFIG_DATA_R21": "155",
100
                      "RSL_CONFIG_DATA_R22": "156",
                      "RSL_CONFIG_DATA_R23": "157"
101
102
                      "RSL_CONFIG_DATA_R24": "158",
103
                      "RSL_CONFIG_DATA_R25": "159",
104
                      "RSL_CONFIG_DATA_R26": "15A",
                      "RSL_CONFIG_DATA_R27": "15B",
105
106
                      "RSL_CONFIG_DATA_R28": "15C",
                      "RSL_CONFIG_DATA_R29": "15D",
107
                      "RSL_CONFIG_DATA_R30": "15E",
"RSL_CONFIG_DATA_R31": "15F",
108
109
110
                      "RSL_CONFIG_DATA_LO": "160",
111
                      "RSL_CONFIG_DATA_L1": "161",
                      "RSL_CONFIG_DATA_L2": "162",
112
                      "RSL CONFIG DATA L3": "163"
113
                      "RSL_CONFIG_DATA_L4": "164",
114
                      "RSL_CONFIG_DATA_L5": "165"
115
                      "RSL_CONFIG_DATA_L6": "166"
116
                      "RSL_CONFIG_DATA_L7": "167"
117
                      "RSL_CONFIG_DATA_L8": "168",
118
                      "RSL_CONFIG_DATA_L9": "169"
119
                      "RSL_CONFIG_DATA_L10": "16A"
120
                      "RSL_CONFIG_DATA_L11": "16B"
121
                      "RSL_CONFIG_DATA_L12": "16C"
122
                      "RSL_CONFIG_DATA_L13": "16D"
123
                      "RSL_CONFIG_DATA_L14": "16E",
124
                      "RSL_CONFIG_DATA_L15": "16F"
125
                      "RSL_CONFIG_DATA_L16": "170"
126
                      "RSL_CONFIG_DATA_L17": "171"
127
                      "RSL_CONFIG_DATA_L18": "172",
128
                      "RSL_CONFIG_DATA_L19": "173"
129
                      "RSL_CONFIG_DATA_L20": "174"
130
                      "RSL_CONFIG_DATA_L21": "175"
131
                      "RSL_CONFIG_DATA_L22": "176",
132
                      "RSL_CONFIG_DATA_L23": "177"
133
                      "RSL_CONFIG_DATA_L24": "178"
134
                      "RSL_CONFIG_DATA_L25": "179"
135
                      "RSL_CONFIG_DATA_L26": "17A",
136
                      "RSL_CONFIG_DATA_L27": "17B",
"RSL_CONFIG_DATA_L28": "17C",
137
138
                      "RSL_CONFIG_DATA_L29": "17D",
139
140
                      "RSL_CONFIG_DATA_L30": "17E",
                      "RSL_CONFIG_DATA_L31": "17F"
141
142
                      "RSL_READ_BACK_RO": "180",
143
                      "RSL_READ_BACK_R1": "181",
                      "RSL_READ_BACK_R2": "182",
144
145
                      "RSL_READ_BACK_R3": "183",
                      "RSL_READ_BACK_R4": "184",
146
147
                      "RSL_READ_BACK_R5": "185",
                      "RSL_READ_BACK_R6": "186",
148
149
                      "RSL_READ_BACK_R7": "187",
150
                      "RSL_READ_BACK_R8": "188",
                      "RSL_READ_BACK_R9": "189",
151
                      "RSL_READ_BACK_R10": "18A"
152
                      "RSL_READ_BACK_R11": "18B",
153
                      "RSL_READ_BACK_R12": "18C"
155
                      "RSL_READ_BACK_R13": "18D",
                      "RSL_READ_BACK_R14": "18E",
156
                      "RSL_READ_BACK_R15": "18F",
157
158
                      "RSL_READ_BACK_R16": "190",
                      "RSL_READ_BACK_R17": "191",
159
                      "RSL_READ_BACK_R18": "192"
160
                      "RSL_READ_BACK_R19": "193",
161
                      "RSL_READ_BACK_R20": "194",
162
                      "RSL_READ_BACK_R21": "195",
163
                      "RSL READ BACK R22": "196"
164
                      "RSL_READ_BACK_R23": "197",
165
                      "RSL_READ_BACK_R24": "198",
166
                      "RSL_READ_BACK_R25": "199",
"RSL_READ_BACK_R26": "19A",
167
168
```

```
169
                         "RSL_READ_BACK_R27": "19B",
170
                         "RSL_READ_BACK_R28": "19C",
171
                         "RSL_READ_BACK_R29": "19D"
172
                         "RSL_READ_BACK_R30": "19E",
173
                         "RSL_READ_BACK_R31": "19F",
                         "RSL_READ_BACK_LO": "1AO",
174
175
                         "RSL_READ_BACK_L1": "1A1",
176
                         "RSL_READ_BACK_L2": "1A2",
177
                         "RSL_READ_BACK_L3": "1A3",
                         "RSL_READ_BACK_L4": "1A4",
178
179
                         "RSL_READ_BACK_L5": "1A5",
                         "RSL_READ_BACK_L6": "1A6",
180
181
                         "RSL_READ_BACK_L7": "1A7",
                         "RSL_READ_BACK_L8": "1A8",
182
183
                         "RSL_READ_BACK_L9": "1A9",
184
                         "RSL_READ_BACK_L10": "1AA"
                         "RSL_READ_BACK_L11": "1AB",
185
                         "RSL_READ_BACK_L12": "1AC"
186
                         "RSL_READ_BACK_L13": "1AD"
187
                         "RSL_READ_BACK_L14": "1AE",
188
                         "RSL_READ_BACK_L15": "1AF",
189
                         "RSL_READ_BACK_L16": "1B0",
190
191
                         "RSL_READ_BACK_L17": "1B1",
192
                         "RSL_READ_BACK_L18": "1B2",
                         "RSL_READ_BACK_L19": "1B3",
193
                         "RSL_READ_BACK_L20": "1B4"
194
                         "RSL_READ_BACK_L21": "1B5"
195
                         "RSL_READ_BACK_L22": "1B6",
196
                         "RSL_READ_BACK_L23": "1B7",
"RSL_READ_BACK_L24": "1B8",
197
198
                         "RSL_READ_BACK_L25": "1B9"
199
                         "RSL_READ_BACK_L26": "1BA"
200
                         "RSL_READ_BACK_L27": "1BB"
201
                          "RSL_READ_BACK_L28": "1BC"
202
                         "RSL_READ_BACK_L29": "1BD"
203
                          "RSL_READ_BACK_L30": "1BE"
2.04
                         "RSL_READ_BACK_L31": "1BF",
2.05
206
                   }
207
              )
208
             self.sens_subregisters = [
   ("STAT_RSLROWOUTL", "STAT_REG", 3, 1, False),
   ("STAT_RSLROWOUTR", "STAT_REG", 4, 1, False),
   ("STAT_RSLNALLWENR", "STAT_REG", 12, 1, False),
   ("STAT_RSLNALLWENL", "STAT_REG", 15, 1, False),
209
210
211
212
213
                   ("STAT_CONFIGHSTDONE", "STAT_REG", 16, 1, False),
214
                  ("SLOWREADOFF_0", "CTRL_REG", 4, 1, True), ("SLOWREADOFF_1", "CTRL_REG", 5, 1, True),
215
216
                    ("HFW", "RSL_HFW_MODE_EN", 0, 1, True),
("ZDT_R", "RSL_ZDT_MODE_R_EN", 0, 1, True),
("ZDT_L", "RSL_ZDT_MODE_L_EN", 0, 1, True),
217
218
219
220
221
```

8.2.3 Member Function Documentation

8.2.3.1 checkSensorVoltStat()

```
def nsCamera.sensors.daedalus.daedalus.checkSensorVoltStat ( self \ ) Checks register tied to sensor select jumpers to confirm match with sensor object  
Returns: boolean, True if jumpers select for Daedalus sensor
```

Definition at line 222 of file daedalus.py.

```
def checkSensorVoltStat(self):
223
224
            Checks register tied to sensor select jumpers to confirm match with sensor
225
226
227
            Returns:
            boolean, True if jumpers select for Daedalus sensor
228
229
230
            err, status = self.ca.getSubregister("DAEDALUS_DET")
231
232
               logging.error(self.logerr + "unable to confirm sensor status")
233
                return False
234
            if not int(status):
235
               logging.error(self.logerr + "Daedalus sensor not detected")
                return False
237
            return True
238
```

8.2.3.2 getManualTiming()

Definition at line 766 of file daedalus.py.

```
def getManualTiming(self):
766
767
768
            Dummy function; feature is not implemented on Daedalus
769
770
            Returns:
           list of 2 dummy lists
771
772
773
            logging.warning(
774
               self.logwarn + "manual shutter control is not implemented in the "
775
                "Daedalus sensor "
776
            return [[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0]]
778
```

8.2.3.3 getTiming()

```
actual = True: returns actual high speed intervals that will be generated by the
  FPGA as list [delay, open0, closed0, open1, closed1, open2, closed2, open3]
actual = False: Returns high speed timing settings as set by setTiming. Assumes
  that timing was set via the setTiming method--it will not accurately report
  arbitrary timings set by direct register sets or manual shutter control
Aras:
    side: Hemisphere 'A' or 'B'
    actual: False: return HST settings
    True: calculate and return actual HST behavior
Returns:
    actual= False: tuple
                               (hemisphere label,
                       'open shutter' in ns,
                        'closed shutter' in ns,
                       initial delay in ns)
    True: list of times [delay, open0, closed0, open1, closed1, open2,
       closed2, open3]
Definition at line 685 of file daedalus.pv.
        def getTiming(self, side, actual):
685
686
687
            actual = True: returns actual high speed intervals that will be generated by the
688
              FPGA as list [delay, open0, closed0, open1, closed1, open2, closed2, open3]
            actual = False: Returns high speed timing settings as set by setTiming. Assumes
689
690
              that timing was set via the setTiming method--it will not accurately report
691
              arbitrary timings set by direct register sets or manual shutter control
692
693
            Args:
                side: Hemisphere 'A' or 'B'
694
695
                actual: False: return HST settings
696
                        True: calculate and return actual HST behavior
697
698
            Returns:
699
                actual= False: tuple
                                        (hemisphere label,
                                         'open shutter' in ns,
'closed shutter' in ns,
700
701
702
                                         initial delay in ns)
703
                         True: list of times [delay, open0, closed0, open1, closed1, open2,
704
                           closed2, open3]
705
706
            if side is None:
707
                side = "A"
708
709
            logging.info(self.loginfo + "get timing, side " + side.upper())
710
            if side.upper() == "A":
                lowreg = "HS_TIMING_DATA_ALO"
711
712
                highreg = "HS_TIMING_DATA_AHI"
713
            elif side.upper() == "B":
714
                lowreg = "HS_TIMING_DATA_BLO"
715
                highreg = "HS_TIMING_DATA_BHI"
716
            else:
717
                logging.error(
718
                    self.logerr
                     + "Invalid sensor side: "
719
720
                     + side
721
                    + "; timing settings unchanged"
722
            return "", 0, 0, 0
err, lowpart = self.ca.getRegister(lowreg)
723
724
725
            errl, highpart = self.ca.getRegister(highreg)
726
            if err or err1:
727
                logging.error(
728
                    self.logerr + "Unable to retrieve timing setting (getTiming), "
729
                    "returning zeroes "
730
            return side.upper(), 0, 0, 0
full40hex = highpart[-2:] + lowpart.zfill(8)
full40bin = "{0:0=40b}".format(int(full40hex, 16))
731
732
733
734
            if actual:
                full160 = 4 * full40bin
735
                gblist = [[k, len(list(g))] for k, g in itertools.groupby(full160)]
736
                times = [int(x[1]) \text{ for } x \text{ in } gblist[:-7:-1]]
737
                times[0] = times[0] - 1
738
739
                return times
740
            else:
```

```
741
                gblist = [[k, len(list(g))] for k, g in itertools.groupby(full40bin)]
742
                delay = gblist[-1][1] - 1
743
                timeon = gblist[-2][1]
744
                if len(gblist) == 2: # 39,1 corner case
745
                    timeoff = 1
746
                elif len(gblist) == 3: # sequence fits only once
747
                   timeoff = 40 - timeon
748
749
                   timeoff = gblist[-3][1]
                return side.upper(), timeon, timeoff, delay
750
751
```

8.2.3.4 parseReadoff()

Definition at line 779 of file daedalus.py.

```
779
        def parseReadoff(self, frames):
780
781
             Parses frames from board into images
782
783
                 frames: data sets returned from board
784
             Returns:
             list of frames reordered and deinterlaced
785
786
787
             w = self.width
788
             if self.ca.padToFull:
789
                 rows = self.maxheight
790
791
                 rows = self.lastrow - self.firstrow + 1
792
             parsed = []
793
             for frame in frames:
794
                 current = np.zeros((rows, w), dtype=int)
795
                 mapped = np.zeros((rows, w), dtype=int)
796
                 frame = frame.reshape(rows, w)
797
798
                 for entry in range(int(w / 2)):
799
                     col = 32 * (entry % 8) + entry // 8 # lookup from daedlookup.xls
                      for row in range(rows):
801
                          current[row][col] = frame[row][2 * entry]
                          current[row][col + 256] = frame[row][2 * entry + 1]
802
803
804
                 for row in range(rows):
                     mapped[row][0:32] = current[row][320:352]
mapped[row][32:64] = current[row][352:384]
805
                     mapped[row][64:96] = current[row][192:224]
807
                     mapped[row][96:128] = current[row][160:192]
808
                     mapped[row][128:160] = current[row][256:288]
809
810
                     mapped[row][160:192] = current[row][288:320]
811
                     mapped[row][192:224] = current[row][416:448]
                     mapped[row][224:256] = current[row][32:64]
812
813
                     mapped[row][256:288] = current[row][128:160]
                     mapped[row] [288:320] = current[row] [224:256]
814
                     mapped[row][320:352] = current[row][384:416]
815
                     mapped[row][352:384] = current[row][448:480]
816
                     mapped[row][384:416] = current[row][480:512]
817
                     mapped[row][416:448] = current[row][0:32]
818
                     mapped[row][448:480] = current[row][64:96]
mapped[row][480:512] = current[row][96:128]
819
820
                 parsed.append(mapped)
821
```

```
822
823    images = self.ca.deInterlace(parsed, self.interlacing)
824    flatimages = [x.flatten() for x in images]
825    return flatimages
826
```

8.2.3.5 reportStatusSensor()

```
def nsCamera.sensors.daedalus.daedalus.reportStatusSensor (
                self.
                statusbits )
Print status messages from sensor-specific bits of status register or object
  status flags
Args:
    statusbits: result of checkStatus()
Definition at line 827 of file daedalus.py.
827
        def reportStatusSensor(self, statusbits):
828
829
            Print status messages from sensor-specific bits of status register or object
830
              status flags
831
832
            Args:
            statusbits: result of checkStatus()
833
834
            if int(statusbits[3]):
835
                logging.info(self.loginfo + "RSLROWINL detected")
836
837
           if int(statusbits[4]):
                logging.info(self.loginfo + "RSLROWINR detected")
838
839
            if int(statusbits[12]):
                logging.info(self.loginfo + "RSLNALLWENR detected")
840
841
           if int(statusbits[15]):
                logging.info(self.loginfo + "RSLNALLWENL detected")
842
843
            if int(statusbits[16]):
                logging.info(self.loginfo + "CONFIGHSTDONE detected")
844
845
            if self.HFW:
846
                logging.info(self.loginfo + "High Full Well mode active")
847
            if self.ZDT:
                logging.info(self.loginfo + "Zero Dead Time mode active")
848
849
850
851 """
852 Copyright (c) 2022, Lawrence Livermore National Security, LLC. All rights reserved.
853 LLNL-CODE-838080
854
855 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
856 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy
857 (DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
858 See license for disclaimers, notice of U.S. Government Rights and license terms and
859 conditions.
860 """
```

8.2.3.6 sensorSpecific()

Returns:

list of tuples, (Sensor-specific register, default setting)

Definition at line 239 of file daedalus.py.

```
def sensorSpecific(self):
240
241
              Returns:
242
                  list of tuples, (Sensor-specific register, default setting)
243
2.44
              return
                  ("FPA_FRAME_INITIAL", "00000000"),
245
                   ("FPA_FRAME_FINAL", "00000002"),
("FPA_ROW_INITIAL", "00000000"),
("FPA_ROW_FINAL", "000003FF"),
246
247
248
                   ("HS_TIMING_DATA_ALO", "00006666"), ("HS_TIMING_DATA_AHI", "00000000"), ("HS_TIMING_DATA_BLO", "00006666"),
249
                                                               \# 0db6 = 2-1; 6666 = 2-2
250
251
                   ("HS_TIMING_DATA_BHI", "00000000"),
252
                   ("FRAME_ORDER_SEL", "00000000"),
("RSL_HFW_MODE_EN", "00000000"),
("RSL_ZDT_MODE_R_EN", "00000000"),
253
254
255
                   ("RSL_ZDT_MODE_L_EN", "00000000"),
("RSL_CONFIG_DATA_RO", "00000000"),
256
2.57
                   ("RSL_CONFIG_DATA_R1", "00000000"),
258
                   ("RSL_CONFIG_DATA_R2", "00000000"),
259
                   ("RSL_CONFIG_DATA_R3",
                                              "00000000"),
2.60
                   ("RSL_CONFIG_DATA_R4", "00000000"),
261
                   ("RSL_CONFIG_DATA_R5", "00000000"), ("RSL_CONFIG_DATA_R6", "00000000"),
2.62
263
                   ("RSL_CONFIG_DATA_R7",
                                              "00000000"),
264
                   ("RSL_CONFIG_DATA_R8", "00000000"),
2.65
                   ("RSL_CONFIG_DATA_R9", "00000000"),
("RSL_CONFIG_DATA_R10", "00000000"),
266
2.67
                   ("RSL_CONFIG_DATA_R11", "00000000"),
2.68
                   ("RSL_CONFIG_DATA_R12", "00000000"),
269
                   ("RSL_CONFIG_DATA_R13", "00000000"),
270
                   ("RSL_CONFIG_DATA_R14", "00000000"),
271
                   ("RSL_CONFIG_DATA_R15", "00000000"),
272
273
                   ("RSL_CONFIG_DATA_R16", "00000000"),
                   ("RSL_CONFIG_DATA_R17", "00000000"),
274
275
                   ("RSL_CONFIG_DATA_R18", "00000000"),
                   ("RSL_CONFIG_DATA_R19", "00000000"),
276
277
                   ("RSL_CONFIG_DATA_R20", "00000000"),
                   ("RSL_CONFIG_DATA_R21", "00000000"),
278
                   ("RSL_CONFIG_DATA_R22", "00000000"),
279
                   ("RSL_CONFIG_DATA_R23", "00000000"),
280
281
                   ("RSL_CONFIG_DATA_R24", "00000000"),
282
                   ("RSL_CONFIG_DATA_R25", "00000000"),
283
                   ("RSL_CONFIG_DATA_R26", "00000000"),
284
                   ("RSL_CONFIG_DATA_R27", "00000000"),
285
                   ("RSL_CONFIG_DATA_R28", "00000000"),
286
                   ("RSL_CONFIG_DATA_R29", "00000000"),
287
                   ("RSL_CONFIG_DATA_R30", "00000000"),
288
                   ("RSL_CONFIG_DATA_R31", "00000000"),
289
                   ("RSL_CONFIG_DATA_LO",
                                              "00000000"),
290
                   ("RSL_CONFIG_DATA_L1", "00000000"),
291
                   ("RSL_CONFIG_DATA_L2",
                                              "00000000"),
                   ("RSL_CONFIG_DATA_L3",
                                              "00000000"),
292
                   ("RSL_CONFIG_DATA_L4",
                                              "00000000"),
293
                   ("RSL_CONFIG_DATA_L5",
                                              "00000000"),
294
295
                   ("RSL_CONFIG_DATA_L6",
                                              "00000000"),
                   ("RSL_CONFIG_DATA_L7",
296
297
                   ("RSL_CONFIG_DATA_L8",
                                              "00000000"),
298
                   ("RSL_CONFIG_DATA_L9", "00000000"),
                   ("RSL_CONFIG_DATA_L10", "00000000"),
299
300
                   ("RSL_CONFIG_DATA_L11", "00000000"),
301
                   ("RSL_CONFIG_DATA_L12", "00000000"),
302
                   ("RSL_CONFIG_DATA_L13", "00000000"),
303
                   ("RSL_CONFIG_DATA_L14", "00000000"),
                   ("RSL_CONFIG_DATA_L15", "00000000"),
304
305
                   ("RSL_CONFIG_DATA_L16", "00000000"),
306
                   ("RSL_CONFIG_DATA_L17", "00000000"),
                   ("RSL_CONFIG_DATA_L18", "00000000"),
307
                   ("RSL CONFIG DATA L19", "00000000"),
308
                   ("RSL_CONFIG_DATA_L20", "00000000"),
309
                   ("RSL_CONFIG_DATA_L21", "00000000"),
310
                   ("RSL_CONFIG_DATA_L22", "00000000"), ("RSL_CONFIG_DATA_L23", "00000000"),
311
312
                   ("RSL_CONFIG_DATA_L24", "00000000"),
313
```

```
314
                              ("RSL_CONFIG_DATA_L25", "00000000"),
                             ("RSL_CONFIG_DATA_L26", "00000000"), ("RSL_CONFIG_DATA_L27", "00000000"), ("RSL_CONFIG_DATA_L28", "00000000"),
315
316
317
                             ("RSL_CONFIG_DATA_L29", "00000000"), ("RSL_CONFIG_DATA_L30", "00000000"), ("RSL_CONFIG_DATA_L31", "00000000"),
318
319
320
                             ("HST_TRIGGER_DELAY_DATA_LO", "00000000"), ("HST_TRIGGER_DELAY_DATA_HI", "00000000"),
321
322
                             ("HST_PHI_DELAY_DATA_LO", "00000000"),
("HST_PHI_DELAY_DATA_HI", "00000000"),
323
324
                              ("SLOWREADOFF_0", "0"),
325
326
                             ("SLOWREADOFF_1", "0"),
327
                     ]
328
```

8.2.3.7 setArbTiming()

Definition at line 612 of file daedalus.py.

```
612
        def setArbTiming(self, side, sequence):
613
614
            Args:
615
                side: Hemisphere 'A' or 'B'
616
                sequence: list of arbitrary timing intervals, beginning with initial delay.
617
                   The conventional timing (5,2) with delay = 3 would be represented by
                   [3,5,2,5,2,5,2,5].
619
                *WARNING* arbitrary timings will not be restored after a board power cycle
620
621
                tuple (error string, 10-character hexadecimal representation of timing
622
623
                  sequence)
624
625
            if side is None:
626
                side = "A"
627
            if sequence is None:
628
                sequence = [0, 3, 2, 3, 2, 3, 2, 3]
629
630
            logging.info(
631
                self.loginfo + "HST side " + side.upper() + " (arbitrary): " + str(sequence)
632
            if side.upper() == "A":
633
                lowreg = "HS_TIMING_DATA_ALO"
634
635
                highreq = "HS_TIMING_DATA_AHI"
            elif side.upper() == "B":
    lowreg = "HS_TIMING_DATA_BLO"
636
637
                highreg = "HS_TIMING_DATA_BHI"
638
639
            else:
640
                err = (
                    self.logerr
641
                     + "Invalid sensor side: "
642
```

```
643
                       + side
644
                       + "; timing settings unchanged"
645
                  logging.error(err)
646
647
                  return err, "0000000000"
648
             full40 = [0] * 40
             bitlist = []
flag = 0 # similar to setTiming, but starts with delay
649
650
651
             sequence = sequence[: (2 * self.nframes)]
652
             for a in sequence:
653
                  add = [flag] * a
                  bitlist += add
655
                  if flag:
656
                      flag = 0
657
                  else:
658
                      flag = 1
             reversedlist = bitlist[39::-1]
659
             full40[-(len(reversedlist) + 1) : -1] = reversedlist
660
             full40bin = "".join(str(x) for x in full40) full40hex = "%x" % int(full40bin, 2)
661
662
             highpart = full40hex[-10:-8].zfill(8)
lowpart = full40hex[-8:].zfill(8)
663
664
             self.ca.setRegister(lowreg, lowpart)
self.ca.setRegister(highreg, highpart)
665
666
             # deactivates manual shutter mode if previously engaged
667
668
             self.ca.setRegister("MANUAL_SHUTTERS_MODE", "00000000")
669
             actual = self.getTiming(side, actual=True)
670
671
             if actual != sequence:
672
                  logging.warning(
673
                       self.logwarn + "Due to sequence length, actual timing sequence "
674
                       "for side "
675
                       + side
                       + " will be "
676
                       + "{"
677
678
                       + str(actual[0])
679
680
                       + str(actual[1 : 2 * self.nframes])
681
682
                 )
683
             return actual
684
```

8.2.3.8 setHighFullWell()

Args:

```
def nsCamera.sensors.daedalus.daedalus.setHighFullWell (
               self.
               flag )
Activates High Full Well mode. All frames are acquired simultaneously. Zero Dead
  Time mode and interlacing will be automatically deactivated. NOTE: after \ensuremath{\mathsf{T}}
  deactivating HFW, the board remains in uninterlaced mode (interlacing = 0)
Args:
    flag: True to activate HFW mode, False to deactivate
Returns:
    Error message
Definition at line 393 of file daedalus.py.
       def setHighFullWell(self, flag):
393
394
395
           Activates High Full Well mode. All frames are acquired simultaneously. Zero Dead
396
             Time mode and interlacing will be automatically deactivated. NOTE: after
397
             deactivating HFW, the board remains in uninterlaced mode (interlacing = 0)
398
399
```

def nsCamera.sensors.daedalus.daedalus.setInterlacing (

```
400
                flag: True to activate HFW mode, False to deactivate
401
402
            Returns:
           Error message
403
404
405
            err0 = ""
406
            if flag:
407
               if self.ZDT:
408
                    logging.warning(
                        self.logwarn + "ZDT mode will be disengaged because of HFW "
409
410
                        "setting "
411
412
                    err0 = self.setZeroDeadTime(False)
413
                err1, _ = self.ca.setSubregister("HFW", "1")
414
                self.HFW = True
415
               logging.info(self.loginfo + "High Full Well mode active")
416
           else:
417
               self.HFW = False
               err1, _ = self.ca.setSubregister("HFW", "0")
418
419
               self.setInterlacing(0)
               logging.info(self.loginfo + "High Full Well mode inactivate")
420
421
            err = err0 + err1
422
           if err:
423
               logging.error(self.logerr + "HFW option may not be set correctly ")
424
           return err
425
```

8.2.3.9 setInterlacing()

```
self.
               ifactor )
Sets interlacing factor. NOTE: if called directly when HFW or ZDT mode is
active, this will disengage those modes automatically.
Aras:
    ifactor: number of interlaced lines (generates ifactor + 1 images per frame)
      defaults to 0 (no interlacing)
Returns:
    integer: active interlacing factor (unchanged if error)
Definition at line 329 of file daedalus.py.
329
        def setInterlacing(self, ifactor):
330
331
            Sets interlacing factor. NOTE: if called directly when HFW or ZDT mode is
332
            active, this will disengage those modes automatically.
333
334
335
               ifactor: number of interlaced lines (generates ifactor + 1 images per frame)
336
                 defaults to 0 (no interlacing)
337
338
            Returns:
           integer: active interlacing factor (unchanged if error) _{\tt nnn}
339
340
341
            if ifactor is None:
342
               ifactor = 0
343
            if (
344
               not isinstance(ifactor, int)
345
               or ifactor < 0
               or ifactor > (self.maxheight - 1)
346
347
            ):
348
                err = (
                   self.logerr + "invalid interlacing factor submitted. "
349
350
                    "Interlacing remains unchanged.
351
```

```
352
                 logging.error(err)
353
                 return self.interlacing
354
             if self.HFW:
                 logging.warning(
355
356
                     self.logwarn + "HFW mode will be disengaged because of new "
357
                     "interlacing setting "
358
359
                 self.setHighFullWell(False)
360
             if self.ZDT:
361
                 logging.warning(
                     self.logwarn + "ZDT mode will be disengaged because of new "
362
                     "interlacing setting "
363
364
365
                 self.setZeroDeadTime(False)
366
             if ifactor == 0:
367
                 bitscheme = self.maxheight * [0]
368
             else:
369
                 pattern = [0] + ifactor * [1]
                 reps = 1 + self.maxheight // (ifactor + 1)
370
                 bitscheme = (reps * pattern)[0 : self.maxheight]
371
            err = ""
372
373
             for a in range(32):
374
                 rname = "RSL_CONFIG_DATA_R" + str(a)
375
                 lname = "RSL_CONFIG_DATA_L" + str(a)
376
                 regbits = bitscheme[32 * a : 32 * (a + 1)]
377
                 \# generated pattern is reverse order from placement in register (element 0
                    of the list is the LSB of the register)
378
379
                 bitsrev = regbits[::-1]
                 b = "".join(s) # assemble as binary number for processing
380
381
                 hexval = "%x" % int(b, 2)
382
                 val = hexval.zfill(8)
383
384
                 err0, _ = self.ca.setRegister(rname, val)
                 err1, _ = self.ca.setRegister(lname, val)
385
386
                 err = err + err0 + err1
387
             if err:
            logging.error(self.logerr + "interlacing may not be set correctly: " + err) logging.info(self.loginfo + "Interlacing factor set to " + str(ifactor))
388
389
390
             self.interlacing = ifactor
391
             return self.interlacing
392
```

8.2.3.10 setManualShutters()

```
def nsCamera.sensors.daedalus.daedalus.setManualShutters (
                self.
                timing )
Dummy function; feature is not implemented on Daedalus
Returns:
    tuple (error string, dummy response string from final message)
Definition at line 752 of file daedalus.pv.
752
        def setManualShutters(self, timing):
753
754
            Dummy function; feature is not implemented on Daedalus
755
756
            Returns:
            tuple (error string, dummy response string from final message)
757
758
759
            err = (
                self.logerr + "manual shutter control is not implemented in the "
760
761
                "Daedalus sensor "
762
            logging.error(err)
return err, "00000000"
763
764
765
```

8.2.3.11 setTiming()

```
def nsCamera.sensors.daedalus.daedalus.setTiming (
               self,
               side.
               sequence,
               delay )
Sets timing registers based on 'sequence.' WARNING: if entire sequence does not
  fit into the 40-bit register space, then the actual timings may differ from
  those requested. If the timing sequence fits only once into register space
  (i.e., for a single frame, open + closed > 19 ns ), then actual timing will be
  (n, 40-n) irrespective of setting of second parameter, e.g. (35,1) will
  actually result in (35,5) timing
Args:
    side: Hemisphere 'A' or 'B'
    sequence: two-element tuple of timing durations in ns, e.g., '(5,2)'
    delay: initial delay in ns
Returns:
    tuple (error string, 10-character hexadecimal representation of timing
      sequence)
Definition at line 493 of file daedalus.py.
493
        def setTiming(self, side, sequence, delay):
494
           Sets timing registers based on 'sequence.' WARNING: if entire sequence does not
495
496
             fit into the 40-bit register space, then the actual timings may differ from
497
             those requested. If the timing sequence fits only once into register space
498
              (i.e., for a single frame, open + closed > 19 ns ), then actual timing will be
499
              (n, 40-n) irrespective of setting of second parameter, e.g. (35,1) will
500
             actually result in (35,5) timing
501
502
           Args:
503
               side: Hemisphere 'A' or 'B'
504
                sequence: two-element tuple of timing durations in ns, e.g., '(5,2)'
505
               delay: initial delay in ns
506
507
           Returns:
508
               tuple (error string, 10-character hexadecimal representation of timing
509
510
511
           if side is None:
512
               side = "A"
513
           if sequence is None:
               sequence = (3, 2)
515
           if delay is None:
516
               delay = 0
517
           if len(sequence) != 2:
519
               err = (
                   self.logerr
520
                    + "Invalid sequence setting for side: "
521
522
                    + side
                    + "; timing settings are unchanged"
524
525
               logging.error(err)
                return err, "0000000000"
526
527
           logging.info(
528
               self.loginfo
529
               + "HST side "
530
               + side.upper()
531
532
               + str(sequence)
               + "; delay =
533
534
               + str(delay)
535
           if side.upper() == "A":
    lowreg = "HS_TIMING_DATA_ALO"
536
537
               highreg = "HS_TIMING_DATA_AHI"
538
```

```
539
            elif side.upper() == "B":
540
                 lowreg = "HS_TIMING_DATA_BLO"
541
                 highreg = "HS_TIMING_DATA_BHI"
542
            else:
543
                 err = (
                     self.logerr
544
545
                     + "Invalid sensor side: "
546
                      + side
547
                     + "; timing settings unchanged"
548
549
                 logging.error(err)
                 return err, "0000000000"
550
551
            if (sequence[0] + sequence[1]) + delay > 40:
552
                 err = (
553
                     self.logerr + "Timing sequence is too long to be implemented; "
554
                     "timing settings unchanged "
555
556
                 logging.error(err)
557
                 return err, "0000000000"
558
559
            self.ca.senstiming[side.upper()] = (sequence, delay)
            self.ca.sensmanual = [] # clear manual settings from ca
560
561
562
            full40 = [0] * 40
563
            bitlist = []
564
            flag = 1
565
            sequence = sequence[:2]
566
            for a in sequence:
                 add = [flag] * a
567
                 bitlist += add
568
569
                 if flag:
570
                    flag = 0
571
                 else:
572
                    flag = 1
573
            # automatically truncates sequence to 39 characters
574
            reversedlist = bitlist[39::-1]
            repeats = (40 - delay) // len(reversedlist)
575
576
            if repeats > self.nframes:
577
                 repeats = self.nframes
578
             \# Pattern from sequence repeated to fit inside 40 bits up to a maximum of
579
                'nframes' times
            repeated = reversedlist * repeats
580
            if (len(repeated) + delay + 1) < 40 and repeats == self.nframes:
# add 'stop' bit for ZDT mode if full sequence is less than the full 40 bits
581
582
                 repeated = [1] + repeated
583
            584
            full40bin = "".join(str(x) for x in full40)
full40hex = "%x" % int(full40bin, 2)
585
586
            highpart = full40hex[-10:-8].zfill(8)
587
588
            lowpart = full40hex[-8:].zfill(8)
             err0, _ = self.ca.setRegister(lowreg, lowpart)
589
            err1, _ = self.ca.setRegister(highreq, highpart)
err2, _ = self.ca.setRegister("HS_TIMING_CTL", "00000001")
590
591
592
            err = err0 + err1 + err2
593
            if err:
594
                 logging.error(self.logerr + "Timing may not have been set correctly")
595
             if repeats < self.nframes:</pre>
596
                 actual = self.getTiming(side, actual=True)
597
                 expected = [delay] + 3 * list(sequence) + [sequence[0]]
598
                 if actual != expected:
                     logging.warning(
599
600
                          self.logwarn + "Warning: Due to sequence length, actual "
601
                          "timing sequence for side "
602
                          + side
                          + " will be "
603
                          + " { "
604
605
                          + str(actual[0])
606
                          + " "
607
608
                          + str(actual[1 : 2 * self.nframes])
609
610
            return err, full40hex
611
```

8.2.3.12 setTriggerDelay()

```
def nsCamera.sensors.daedalus.daedalus.setTriggerDelay (
                 self.
                 delayblocks )
NOTE: THIS IS BASED ON AN UNCERTAIN INTERPRETATION OF THE HDD
Args:
     delayblocks: number of 150 ps blocks to delay trigger (maximum of 38?)
Definition at line 468 of file daedalus.py.
468
        def setTriggerDelay(self, delayblocks):
470
             NOTE: THIS IS BASED ON AN UNCERTAIN INTERPRETATION OF THE HDD
471
472
             delayblocks: number of 150 ps blocks to delay trigger (maximum of 38?)
473
474
475
             if not isinstance(delayblocks, int) or delayblocks < 0 or delayblocks > 38:
476
                 err = 0
477
                     self.logerr + "invalid trigger delay submitted. Delay remains "
478
                     "unchanged. "
479
480
                 logging.error(err)
481
                 return err
             delayseq = (38 - delayblocks) * [0] + delayblocks * [1] + [0, 1]
482
             seqstr = "".join(str(x) for x in delayseq)
seqhex = "%x" % int(seqstr, 2)
483
484
485
             highpart = seqhex[-10:-8].zfill(8)
             lowpart = seqhex[-8:].zfill(8)
486
             err0, _ = self.ca.setRegister("HST_TRIGGER_DELAY_DATA_LO", lowpart) err1, _ = self.ca.setRegister("HST_TRIGGER_DELAY_DATA_HI", highpart)
487
488
                    _ = self.ca.setRegister("HS_TIMING_CTL", "00000001")
             err2,
489
             delayed = delayblocks * 0.15
490
             logging.info(self.loginfo + "Trigger delay = " + str(delayed) + " ns")
491
492
```

8.2.3.13 setZeroDeadTime()

Definition at line 426 of file daedalus.py.

```
def setZeroDeadTime(self, flag):
427
428
             Activates Zero Dead Time mode. Even rows follow the assigned HST schedule; odd
429
               rows are acquired while the 'shutter' for the even rows are closed. High Full
430
               Well mode and interlacing will be automatically deactivated.
431
            NOTE: after deactivating ZDT, the board reverts to uninterlaced mode
432
              (interlacing = 0)
433
434
435
                 flag: True to activate ZDT mode, False to deactivate
436
437
             Returns:
            Error message
438
439
440
             err0 = ""
441
             if flag:
                 if self.HFW:
442
443
                     logging.warning(
444
                         self.logwarn + "HFW mode will be disengaged because of ZDT "
445
                          "setting "
446
447
                     err0 = self.setHighFullWell(False)
448
                 err1, _ = self.ca.setSubregister("ZDT_R", "1")
                err2, _ = self.ca.setSubregister("ZDT_L", "1")
self.ZDT = False # preclude ZDT deactivation message
449
450
                 self.setInterlacing(0)
451
452
                 self.interlacing = 1
453
                 self.ZDT = True
454
                 logging.info(
455
                     self.loginfo + "Zero Dead Time mode active; actual interlacing = 1"
456
457
            else:
                self.ZDT = False
458
                 err1, _ = self.ca.setSubregister("ZDT_R", "0")
err2, _ = self.ca.setSubregister("ZDT_L", "0")
459
460
461
                 self.setInterlacing(0)
                 logging.info(self.loginfo + "Zero Dead Time mode inactivate")
462
463
             err = err0 + err1 + err2
464
            if err:
                 logging.error(self.logerr + "ZDT option may not be set correctly ")
465
466
             return err
467
```

8.2.4 Member Data Documentation

8.2.4.1 bytesperpixel

nsCamera.sensors.daedalus.daedalus.bytesperpixel

Definition at line 47 of file daedalus.py.

8.2.4.2 ca

nsCamera.sensors.daedalus.daedalus.ca

Definition at line 28 of file daedalus.py.

8.2.4.3 firstframe

nsCamera.sensors.daedalus.daedalus.firstframe

Definition at line 38 of file daedalus.py.

8.2.4.4 firstrow

nsCamera.sensors.daedalus.daedalus.firstrow

Definition at line 43 of file daedalus.py.

8.2.4.5 fpganumID

nsCamera.sensors.daedalus.daedalus.fpganumID

Definition at line 48 of file daedalus.py.

8.2.4.6 height

nsCamera.sensors.daedalus.daedalus.height

Definition at line 46 of file daedalus.py.

8.2.4.7 HFW

nsCamera.sensors.daedalus.daedalus.HFW

Definition at line 51 of file daedalus.py.

8.2.4.8 interlacing

nsCamera.sensors.daedalus.daedalus.interlacing

Definition at line 49 of file daedalus.py.

8.2.4.9 lastframe

nsCamera.sensors.daedalus.daedalus.lastframe

Definition at line 39 of file daedalus.py.

8.2.4.10 lastrow

nsCamera.sensors.daedalus.daedalus.lastrow

Definition at line 44 of file daedalus.py.

8.2.4.11 logcrit

nsCamera.sensors.daedalus.daedalus.logcrit

Definition at line 29 of file daedalus.py.

8.2.4.12 logdebug

nsCamera.sensors.daedalus.daedalus.logdebug

Definition at line 33 of file daedalus.py.

8.2.4.13 logerr

nsCamera.sensors.daedalus.daedalus.logerr

Definition at line 30 of file daedalus.py.

8.2.4.14 loginfo

 $\verb|nsCamera.sensors.daedalus.daedalus.loginfo|\\$

Definition at line 32 of file daedalus.py.

8.2.4.15 logwarn

nsCamera.sensors.daedalus.daedalus.logwarn

Definition at line 31 of file daedalus.py.

8.2.4.16 maxframe

nsCamera.sensors.daedalus.daedalus.maxframe

Definition at line 37 of file daedalus.py.

8.2.4.17 maxheight

nsCamera.sensors.daedalus.daedalus.maxheight

Definition at line 42 of file daedalus.py.

8.2.4.18 maxwidth

nsCamera.sensors.daedalus.daedalus.maxwidth

Definition at line 41 of file daedalus.py.

8.2.4.19 minframe

nsCamera.sensors.daedalus.daedalus.minframe

Definition at line 36 of file daedalus.py.

8.2.4.20 nframes

nsCamera.sensors.daedalus.daedalus.nframes

Definition at line 40 of file daedalus.py.

8.2.4.21 sens_registers

nsCamera.sensors.daedalus.daedalus.sens_registers

Definition at line 53 of file daedalus.py.

8.2.4.22 sens_subregisters

nsCamera.sensors.daedalus.daedalus.sens_subregisters

Definition at line 209 of file daedalus.py.

8.2.4.23 width

nsCamera.sensors.daedalus.daedalus.width

Definition at line 45 of file daedalus.py.

8.2.4.24 ZDT

nsCamera.sensors.daedalus.daedalus.ZDT

Definition at line 50 of file daedalus.py.

The documentation for this class was generated from the following file:

nsCamera/sensors/daedalus.py

8.3 nsCamera.utils.GenTec.GenTec Class Reference

Public Member Functions

- def __init__ (self)
- def closeDevice (self)
- def sendSerial (self, ser, message, sleep=0.3)
- def ready (self)
- def GenTecReadTest (self)

Public Attributes

serial

8.3.1 Detailed Description

Definition at line 25 of file GenTec.py.

8.3.2 Constructor & Destructor Documentation

```
8.3.2.1 __init__()
```

```
\begin{tabular}{ll} \tt def nsCamera.utils.GenTec.GenTec.\_init\_\_ ( \\ & self \end{tabular} \label{table}
```

Definition at line 26 of file GenTec.py.

```
def __init__(self):
           self.serial = None
           ports = list(serial.tools.list_ports.comports())
28
            for p, desc, add in ports:
29
30
                try:
                    ser = serial.Serial(
32
                        p,
115200,
33
                        parity=serial.PARITY_NONE,
34
35
                        timeout=0.01,
                        write_timeout=0.01,
36
37
                    resp = self.sendSerial(ser, "*VER")
if "Maestro" in resp:
38
39
                        self.serial = ser
40
41
                   # print (desc, add, resp) # uncomment to see available ports
43
               except Exception as e:
                  print(e)
           if not self.serial:
45
               print("Unable to contact a GenTec Device")
46
47
```

8.3.3 Member Function Documentation

8.3.3.1 closeDevice()

```
\label{lem:control} \mbox{def nsCamera.utils.GenTec.GenTec.closeDevice (} \\ self \mbox{)}
```

Definition at line 48 of file GenTec.py.

```
48 def closeDevice(self):
49 self.serial.close()
50
```

8.3.3.2 GenTecReadTest()

```
def nsCamera.utils.GenTec.GenTec.GenTecReadTest (
                 self )
Definition at line 60 of file GenTec.py.
        def GenTecReadTest(self):
            print(self.sendSerial(self.serial, "*VER"))
61
            print("Press ctrl-c to stop read")
62
63
            try:
                while 1:
64
                     time.sleep(1)
65
                     if not "Not" in self.sendSerial(
    self.serial, "*NVU"
66
                     ): # skip when response is 'Not Available'
68
                         print(self.sendSerial(self.serial, "*CVU"))
69
            except KeyboardInterrupt:
70
                print("\n --GenTecTest terminated--")
71
72
                 # self.serial.close()
73
74
75 """Command list, with response in []
75 "--Command list, with response in []
76 "*SCS03" - set display range to index 03 (see manual p61 for indices) []
77 "*STL18.0" - set internal trigger level to 18% []
78 "*GTL" - get internal trigger level [2.0\r\n]
79 "*GMD" - get index of current display mode (see manual p65) [0\r\n]
80 "*CVU" - get current device reading [0.012\r\n]
81 "*NVU" - check if new data available [text response]
82 "*PWC01550" - set wavelength (interpolate for non-standard) to 1550nm (five digits) []
83 "*GWL" - get wavelength setting [1064\r\n]
84 "*VER" - get device info [MAESTRO Version 1.00.18\r\n]
85 "*STS" - query status [extended list, see p72]
86 "*ST2" - extended query status [extended list, see p74]
88 see p58 for parsing joulemeters in binary
89 """
90
8.3.3.3 ready()
def nsCamera.utils.GenTec.GenTec.ready (
                 self )
Definition at line 57 of file GenTec.py.
        def ready(self):
            self.sendSerial(self.serial, "*CVU")  # should clear NVU in prep for new data
5.8
59
8.3.3.4 sendSerial()
def nsCamera.utils.GenTec.GenTec.sendSerial (
                 self,
                  ser,
                 message,
                 sleep = 0.3)
Definition at line 51 of file GenTec.py.
        def sendSerial(self, ser, message, sleep=0.3):
51
52
            ser.write(message)
53
            time.sleep(sleep)
54
            avail = ser.in_waiting
55
            return ser.read(avail)
56
```

8.3.4 Member Data Documentation

8.3.4.1 serial

nsCamera.utils.GenTec.GenTec.serial

Definition at line 27 of file GenTec.py.

The documentation for this class was generated from the following file:

nsCamera/utils/GenTec.py

8.4 nsCamera.comms.GigE.GigE Class Reference

Classes

class ZESTETM1_CARD_INFO

Public Member Functions

- def __init__ (self, camassem)
- def sendCMD (self, pkt)
- def arm (self, mode)
- def readoff (self, waitOnSRAM, timeout, fast)
- def writeSerial (self, outstring, timeout=None)
- def readSerial (self, size, timeout=None)
- def openDevice (self)
- def closeDevice (self)
- def getCardIP (self)
- def getCardInfo (self)

Public Attributes

- ca
- logcrit
- logerr
- logwarn
- loginfo
- logdebug
- mode
- writeTimeout
- readTimeout
- payloadsize
- skipError
- dport
- · closecard
- CardInfo
- CardInfoP
- ZCountCards
- ZOpenConnection
- ZWriteData
- ZReadData
- Connection

Private Attributes

zest

8.4.1 Detailed Description

```
Code to manage Gigabit Ethernet connection to board. Each GigE object manages a single OT card; to use multiple cards, instantiate multiple cameraAssembler objects, each specifying the unique IPs of the corresponding OT card.

Note: Orange Tree card must be configured before use. See the README for details

Exposed methods:

arm() - puts camera into wait state for external trigger readoff() - waits for data ready register flag, then copies camera image data into numpy arrays

sendCMD(pkt) - sends packet object via serial port readSerial(size, timeout) - read 'size' bytes from connection writeSerial(outstring) - submits string 'outstring' over connection closeDevice() - close connections and free resources getCardIP() - returns IP address of OT card getCardInfo() - prints report of details of OT card and connection
```

Definition at line 26 of file GigE.py.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 __init__()

```
def nsCamera.comms.GigE.GigE.__init__ (
                self,
                camassem )
Args:
    camassem: parent cameraAssembler object
Definition at line 46 of file GigE.py.
       def __init__(self, camassem):
46
47
48
           Args:
           camassem: parent cameraAssembler object
49
50
51
           self.ca = camassem
52
           self.logcrit = self.ca.logcritbase + "[GigE] "
           self.logerr = self.ca.logerrbase + "[GigE]
5.3
           self.logwarn = self.ca.logwarnbase + "[GigE] "
54
           self.loginfo = self.ca.loginfobase + "[GigE] "
55
           self.logdebug = self.ca.logdebugbase + "[GigE] "
56
           logging.info(self.loginfo + "initializing comms object")
57
5.8
           self.mode = 1
           self.writeTimeout = 10000
59
60
           self.readTimeout = 10000
           self.payloadsize = (
61
               self.ca.sensor.width
62
6.3
               * self.ca.sensor.height
64
               * self.ca.sensor.nframes
65
               \star self.ca.sensor.bytesperpixel
66
           self.skipError = False
67
68
69
           if self.ca.port:
70
               if isinstance(self.ca.port, int) and 0 < self.ca.port < 65536:</pre>
71
                   self.dport = self.ca.port
72
               else:
73
                   logging.error(
                       self.logerr + "GigE: invalid port number supplied, defaulting to "
"20482 "
74
75
76
77
                   self.dport = 20482
78
           else:
79
               self.dport = 20482 # default
80
81
           self.ca.port = self.dport
82
83
           if self.ca.arch == "64bit":
84
               arch = "64"
85
           else:
86
               arch = "32"
87
           if self.ca.platform == "Windows":
               lib_name = "ZestETM1.dll"
89
           elif self.ca.platform == "Linux" or self.ca.platform == "Darwin":
91
               lib_name = "libZestETM1.so"
92
           else:
93
               logging.warning(
                   self.logwarn + "System does not self-identify as Linux, Windows, "
95
                    "or Mac. Assuming posix-style libraries "
96
               lib_name = "libZestETM1.so"
98
99
           self.closecard = False
100
101
            libpath = os.path.join(self.ca.packageroot, "comms", "ZestETM1", arch, lib_name)
102
            self._zest = C.CDLL(libpath)
103
            self.CardInfo = self.ZESTETM1 CARD INFO()
104
105
            self.CardInfoP = C.pointer(self.CardInfo)
106
107
            # functions
            self.ZCountCards = self._zest.ZestETM1CountCards
108
            self.ZCountCards.argtypes = [
109
```

```
110
                C.POINTER(C.c_ulong),
111
                C.POINTER(C.POINTER(self.ZESTETM1_CARD_INFO)),
                C.c_int,
112
113
115
            self.ZOpenConnection = self._zest.ZestETM1OpenConnection
            self.ZOpenConnection.argtypes = [
117
              C.POINTER(self.ZESTETM1_CARD_INFO),
118
                C.c_int,
119
                C.c_ushort,
                C.c_ushort,
121
                C.POINTER(C.c_void_p),
122
            ]
123
124
            self.ZWriteData = self._zest.ZestETM1WriteData
125
            self.ZWriteData.argtypes = [
126
                C.c_void_p,
                C.c_void_p,
127
128
                C.c_ulong,
                C.POINTER(C.c_ulong),
129
130
                C.c_ulong,
131
132
133
            self.ZReadData = self._zest.ZestETM1ReadData
            self.ZReadData.argtypes = [
134
135
                C.c void p,
136
                C.c_void_p,
137
                C.c_ulong,
                C.POINTER(C.c_ulong),
138
139
                C.c_ulong,
140
            ]
141
            self.Connection = C.c_void_p()
142
143
            self.openDevice()
144
```

8.4.3 Member Function Documentation

8.4.3.1 arm()

```
def nsCamera.comms.GigE.GigE.arm (
              self,
              mode )
Puts camera into wait state for trigger. Mode determines source; arm() in
  CameraAssembler defaults to 'Hardware'
Args:
           'Software' activates software triggering, disables hardware trigger
    mode:
    'Hardware activates hardware triggering, disables software trigger
     Hardware is the default
    'Dual' activates dual edge hardware trigger mode and disables
      software trigger
Returns:
    tuple (error, response string)
Definition at line 184 of file GigE.py.
       def arm(self, mode):
184
185
           Puts camera into wait state for trigger. Mode determines source; arm() in
186
187
             CameraAssembler defaults to 'Hardware'
```

```
188
189
            Args:
               mode:
190
                       'Software' activates software triggering, disables hardware trigger
191
                        'Hardware activates hardware triggering, disables software trigger
192
                          Hardware is the default
193
                        'Dual' activates dual edge hardware trigger mode and disables
194
                          software trigger
195
196
            Returns:
            tuple (error, response string)
197
198
199
            if not mode:
200
               mode = "Hardware"
            logging.info(self.loginfo + "arm")
201
202
            self.ca.clearStatus()
203
            self.ca.latchPots()
            err, resp = self.ca.startCapture(mode)
204
205
           if err:
206
                logging.error(self.logerr + "unable to arm camera")
207
            else:
208
                self.ca.armed = True
209
                self.skipError = True
210
            return err, resp
211
```

8.4.3.2 closeDevice()

```
\label{eq:comms_gige} \mbox{\tt def nsCamera.comms.GigE.GigE.closeDevice (} \\ self \mbox{\tt )}
```

Close connection to Orange Tree card and free resources

Definition at line 372 of file GigE.py.

```
def closeDevice(self):
372
373
374
            Close connection to Orange Tree card and free resources
375
376
            self._zest.ZestETM1CloseConnection(self.Connection)
377
            if self.closecard:
378
                    self._zest.ZestETM1FreeCards(self.CardInfoP)
379
380
                except:
381
                   logging.error(self.logerr + "Error reported in OT card closure")
382
            self._zest.ZestETM1Close()
383
```

8.4.3.3 getCardInfo()

Prints status message with information returned by OT card

Definition at line 392 of file GigE.py.

```
def getCardInfo(self):
393
                    Prints status message with information returned by OT card
394
395
396
                    ci = self.CardInfoP.contents
397
                    print("GigE Card Status:")
398
                    print("--
399
                    print("IP: " + ".".join(str(e) for e in [b for b in ci.IPAddr]))
                    print("ControlPort: " + str(ci.ControlPort))
print("Timeout: " + str(ci.Timeout))
print("HTTPPort: " + str(ci.HTTPPort))
401
402
                   print("HIPPORT: " + $LF(cI.HIPPORT))
print("MACAddr: " + ".".join(format(e, "02X") for e in [b for b in ci.MACAddr]))
print("SubNet: " + ".".join(str(e) for e in [b for b in ci.SubNet]))
print("Gateway: " + ".".join(str(e) for e in [b for b in ci.Gateway]))
print("SerialNumber: " + str(ci.SerialNumber))
403
404
405
406
                    print("FirmwareVersion: " + str(ci.FirmwareVersion))
print("HardwareVersion: " + str(ci.HardwareVersion))
407
408
                    print("----")
409
410
```

8.4.3.4 getCardIP()

```
def nsCamera.comms.GigE.GigE.getCardIP ( self \ ) Query IP address of OT card Returns: address of OT card as list of bytes
```

Definition at line 384 of file GigE.py.

```
384 def getCardIP(self):
385 """
386 Query IP address of OT card
387
388 Returns: address of OT card as list of bytes
389 """
390 return self.CardInfo.IPAddr
391
```

8.4.3.5 openDevice()

Find Orange Tree card and open a connection; if ip is supplied as parameter for the CameraAssembler, bypass network search and connect directly to indicated IP address

```
Definition at line 307 of file GigE.py.
```

```
def openDevice(self):
308
309
            Find Orange Tree card and open a connection; if ip is supplied as parameter for
310
              the CameraAssembler, bypass network search and connect directly to indicated
311
312
313
            err = self._zest.ZestETM1Init()
314
            if err:
                logging.critical(self.logcrit + "ZestETM1Init failure")
315
                sys.exit(1)
            logging.info(self.loginfo + "searching for Orange Tree cards")
317
318
            NumCards = C.c_ulong(0)
319
320
            if self.ca.iplist:
321
                ubyte4 = C.c_ubyte * 4
322
                self.CardInfo.IPAddr = ubyte4(*self.ca.iplist)
323
                self.CardInfo.ControlPort = C.c_ushort(self.dport)
324
                self.CardInfo.Timeout = C.c_ulong(self.writeTimeout)
325
                self.closecard = False
326
            else:
327
                err = self.ZCountCards(C.byref(NumCards), C.byref(self.CardInfoP), 2000)
328
                self.closecard = True
329
                if err:
                    logging.critical(self.logcrit + "CountCards failure")
330
331
                    sys.exit(1)
                if NumCards.value == 0:
332
                    self.ZCountCards(C.byref(NumCards), C.byref(self.CardInfoP), 3000)
333
334
                    # try again with longer wait (e.g., after powerup)
335
                    if NumCards.value == 0:
                        logging.info(self.loginfo + "trying to connect again, please wait")
336
                         self.ZCountCards(C.byref(NumCards), C.byref(self.CardInfoP), 5000)
337
338
                         if NumCards.value == 0:
                            logging.info(self.loginfo + "still trying to connect...")
339
340
                             self.ZCountCards(
                                 C.byref(NumCards), C.byref(self.CardInfoP), 6000
341
342
343
                             if NumCards.value == 0:
344
                                 self.ZCountCards(
                                     C.byref(NumCards), C.byref(self.CardInfoP), 7000
345
346
347
                                 if NumCards.value == 0:
348
                                     self.ZCountCards(
349
                                         C.byref(NumCards), C.byref(self.CardInfoP), 7000
350
351
                                     if NumCards.value == 0:
352
                                         logging.critical(
353
                                             self.logcrit + "no Orange Tree cards found"
354
355
                                         sys.exit(1)
356
                else:
357
                    logging.info(
358
                        self.loginfo
359
360
                         + str(NumCards.value)
361
                         + " Orange Tree card(s) found"
362
                       # TODO: add check for GigE bit in board description
363
            err = self.ZOpenConnection(
364
                self.CardInfoP, 0, self.dport, 0, C.byref(self.Connection)
365
366
            if err:
367
                logging.critical(
368
                    self.logcrit + "OpenConnection failure, error #" + str(err)
369
370
                sys.exit(1)
371
```

8.4.3.6 readoff()

timeout,
fast)

Copies image data from board into numpy arrays. The FPGA returns a packet without the CRC suffix

Args:

waitOnSRAM: if True, wait until SRAM_READY flag is asserted to begin copying
data

timeout: passed to waitForSRAM; after this many seconds begin copying data irrespective of SRAM_READY status; 'zero' means wait indefinitely WARNING: If acquisition fails, the SRAM will not contain a current image, but the code will copy the data anyway

fast: if False, parse and convert frames to numpy arrays; if True, return
unprocessed text stream

Returns:

tuple (list of numpy arrays OR raw text stream, length of downloaded payload in bytes, payload error flag) since CRC check is handled by TCP/IP, payload error flag is always False for GigE

Definition at line 212 of file GigE.pv.

```
212
        def readoff(self, waitOnSRAM, timeout, fast):
213
            Copies image data from board into numpy arrays. The FPGA returns a packet
214
              without the CRC suffix
215
216
2.17
                waitOnSRAM: if True, wait until SRAM_READY flag is asserted to begin copying
218
219
                  data
220
                timeout: passed to waitForSRAM; after this many seconds begin copying data
221
                  irrespective of SRAM_READY status; 'zero' means wait indefinitely
2.2.2
                  WARNING: If acquisition fails, the SRAM will not contain a current image,
223
                    but the code will copy the data anyway
                fast: if False, parse and convert frames to numpy arrays; if True, return
224
225
                  unprocessed text stream
226
227
            Returns:
228
                tuple (list of numpy arrays OR raw text stream, length of downloaded payload
229
                  in bytes, payload error flag) since CRC check is handled by TCP/IP,
230
                  payload error flag is always False for GigE
231
232
            logging.info(self.loginfo + "readoff")
233
234
            \ensuremath{\text{\#}} Wait for data to be ready on board
235
            # Skip wait only if explicitly tagged 'False' ('None' defaults to True)
            if not waitOnSRAM==False:
236
237
                self.ca.waitForSRAM(timeout)
238
            self.skipError = False
            self.ca.oldtime = self.ca.currtime
239
            self.ca.currtime = time.time()
240
            self.ca.waited.append(self.ca.currtime - self.ca.oldtime)
241
242
            err, rval = self.ca.readSRAM()
243
            if err:
244
                logging.error(self.logerr + "Error detected in readSRAM")
245
            self.ca.oldtime = self.ca.currtime
            self.ca.currtime = time.time()
246
247
            self.ca.read.append(self.ca.currtime - self.ca.oldtime)
248
            # extract the data. Remove header; the FPGA returns a packet without the CRC
249
              suffix
250
            data = rval[32:]
251
            if fast:
252
                return data, len(data) // 2, bool(err)
253
            else:
254
                parsed = self.ca.generateFrames(data)
255
                return parsed, len(data) // 2, bool(err)
256
```

8.4.3.7 readSerial()

```
def nsCamera.comms.GigE.GigE.readSerial (
               self,
               size,
               timeout = None )
Read bytes from the serial port. Does not verify packets.
Args:
   size: number of bytes to read
   timeout: serial timeout in sec (defaults to self.readTimeout)
Returns:
   tuple (error string, string read from serial port)
Definition at line 281 of file GigE.py.
281
        def readSerial(self, size, timeout=None):
282
2.83
            Read bytes from the serial port. Does not verify packets.
2.84
285
286
              size: number of bytes to read
287
              timeout: serial timeout in sec (defaults to self.readTimeout)
288
289
           tuple (error string, string read from serial port)
290
2.91
292
           if not timeout:
293
               timeout = self.readTimeout
294
            inbuff = C.create_string_buffer(size + 1)
295
            inbuffp = C.pointer(inbuff)
296
            readlen = C.c_ulong(0)
2.97
            err = self.ZReadData(self.Connection, inbuffp, size, C.byref(readlen), timeout)
           if err:
298
299
               if self.skipError:
300
                   self.skipError = False
301
               else:
302
                   logging.error(self.logerr + "readSerial error #" + str(err))
303
                # 32768 = socket error, 32776 = timeout, see comms/ZestETM1/ZestETM1.h line
                # 77 et seq.
305
           return self.ca.bytes2str(inbuff.raw)[:-2]
```

8.4.3.8 sendCMD()

Definition at line 145 of file GigE.py.

```
def sendCMD(self, pkt):
146
147
             Submit packet and verify response packet
148
             Packet communications with FPGA omit CRC suffix, so adds fake CRC bytes to
149
150
151
             Args:
152
                pkt: Packet object
153
154
             Returns:
             tuple (error, response string)
155
156
157
             pktStr = pkt.pktStr()[0:16]
             err = ""
158
159
             self.ca.writeSerial(pktStr)
160
             if (
                 hasattr(self.ca, "board")
and pktStr[4] == "0"
161
162
                 and pktStr[5:8] == self.ca.board.registers["SRAM_CTL"]
163
164
             ):
165
                 bufsize = self.payloadsize + 16
166
                 resptext = self.readSerial(bufsize)
167
                 if len(resptext) < bufsize + 16:</pre>
168
                      err += (
                          self.logerr + "sendCMD- packet too small, payload may be incomplete"
169
170
171
                      logging.error(err)
172
             else:
                 # add fake CRC to maintain consistency with other comms
173
174
                 resp = self.readSerial(8)
175
                 if len(resp) < 8:</pre>
                     err += self.logerr + "sendCMD- response too small, returning zeros" resptext = "000000000000000000000"
176
177
                     logging.error(err)
178
                 else:
179
                      resptext = resp + "0000"
180
181
182
             return err, resptext
183
```

8.4.3.9 writeSerial()

```
def nsCamera.comms.GigE.GigE.writeSerial (
               self.
               outstring,
               timeout = None)
Transmit string to board
Aras:
    outstring: string to write
    timeout: serial timeout in sec (defaults to self.writeTimeout)
Returns:
    integer number of bytes written
Definition at line 257 of file GigE.py.
257
        def writeSerial(self, outstring, timeout=None):
258
259
           Transmit string to board
260
           Args:
261
               outstring: string to write
               timeout: serial timeout in sec (defaults to self.writeTimeout)
262
263
2.64
           Returns:
265
               integer number of bytes written
2.66
```

```
267
            if not timeout:
268
               timeout = self.writeTimeout
269
            outstring = self.ca.str2bytes(outstring)
           outbuff = C.create_string_buffer(outstring)
270
271
           outbuffp = C.pointer(outbuff)
272
           outbufflen = len(outstring)
273
            writelen = C.c_ulong(0)
274
           err = self.ZWriteData(
275
               self.Connection, outbuffp, outbufflen, C.byref(writelen), timeout
               logging.error(self.logerr + "writeSerial error #" + str(err))
279
            return writelen
```

8.4.4 Member Data Documentation

8.4.4.1 _zest

```
nsCamera.comms.GigE.GigE._zest [private]
```

Definition at line 102 of file GigE.py.

8.4.4.2 ca

```
nsCamera.comms.GigE.GigE.ca
```

Definition at line 51 of file GigE.py.

8.4.4.3 CardInfo

```
nsCamera.comms.GigE.GigE.CardInfo
```

Definition at line 104 of file GigE.py.

8.4.4.4 CardInfoP

```
\verb|nsCamera.comms.GigE.GigE.CardInfoP| \\
```

Definition at line 105 of file GigE.py.

8.4.4.5 closecard

nsCamera.comms.GigE.GigE.closecard

Definition at line 99 of file GigE.py.

8.4.4.6 Connection

nsCamera.comms.GigE.GigE.Connection

Definition at line 142 of file GigE.py.

8.4.4.7 dport

nsCamera.comms.GigE.GigE.dport

Definition at line 71 of file GigE.py.

8.4.4.8 logcrit

nsCamera.comms.GigE.GigE.logcrit

Definition at line 52 of file GigE.py.

8.4.4.9 logdebug

nsCamera.comms.GigE.GigE.logdebug

Definition at line 56 of file GigE.py.

8.4.4.10 logerr

nsCamera.comms.GigE.GigE.logerr

Definition at line 53 of file GigE.py.

8.4.4.11 loginfo

nsCamera.comms.GigE.GigE.loginfo

Definition at line 55 of file GigE.py.

8.4.4.12 logwarn

nsCamera.comms.GigE.GigE.logwarn

Definition at line 54 of file GigE.py.

8.4.4.13 mode

nsCamera.comms.GigE.GigE.mode

Definition at line 58 of file GigE.py.

8.4.4.14 payloadsize

 ${\tt nsCamera.comms.GigE.GigE.payloadsize}$

Definition at line 61 of file GigE.py.

8.4.4.15 readTimeout

nsCamera.comms.GigE.GigE.readTimeout

Definition at line 60 of file GigE.py.

8.4.4.16 skipError

 $\verb|nsCamera.comms.GigE.GigE.skipError|\\$

Definition at line 67 of file GigE.py.

8.4.4.17 writeTimeout

nsCamera.comms.GigE.GigE.writeTimeout

Definition at line 59 of file GigE.py.

8.4.4.18 ZCountCards

 $\verb|nsCamera.comms.GigE.GigE.ZCountCards| \\$

Definition at line 108 of file GigE.py.

8.4.4.19 ZOpenConnection

nsCamera.comms.GigE.GigE.ZOpenConnection

Definition at line 115 of file GigE.py.

8.4.4.20 ZReadData

nsCamera.comms.GigE.GigE.ZReadData

Definition at line 133 of file GigE.py.

8.4.4.21 ZWriteData

nsCamera.comms.GigE.GigE.ZWriteData

Definition at line 124 of file GigE.py.

The documentation for this class was generated from the following file:

nsCamera/comms/GigE.py

8.5 nsCamera.sensors.icarus.icarus Class Reference

Public Member Functions

- def __init__ (self, camassem)
- def checkSensorVoltStat (self)
- def sensorSpecific (self)
- def setInterlacing (self, ifactor)
- def setHighFullWell (self, flag)
- def setZeroDeadTime (self, flag)
- def setTriggerDelay (self, delayblocks)
- def setTiming (self, side, sequence, delay)
- def setArbTiming (self, side, sequence)
- def getTiming (self, side, actual)
- def setManualShutters (self, timing)
- def getManualTiming (self)
- def parseReadoff (self, frames)
- def reportStatusSensor (self, statusbits)

Public Attributes

- ca
- logcrit
- logerr
- logwarn
- loginfo
- logdebug
- · minframe
- maxframe
- firstframe
- lastframe
- nframes
- · maxwidth
- maxheight
- · firstrow
- lastrow
- width
- height
- bytesperpixel
- icarustype
- fpganumID
- interlacing
- sens_registers
- sens_subregisters

8.5.1 Detailed Description

Definition at line 26 of file icarus.py.

8.5.2 Constructor & Destructor Documentation

```
8.5.2.1 init ()
def nsCamera.sensors.icarus.icarus.__init__ (
                   camassem )
Definition at line 27 of file icarus.py.
        def __init__(self, camassem):
    self.ca = camassem
28
29
             self.logcrit = self.ca.logcritbase + "[Icarus] "
             self.logerr = self.ca.logerrbase + "[Icarus]
30
31
             self.logwarn = self.ca.logwarnbase + "[Icarus] "
             self.loginfo = self.ca.loginfobase + "[Icarus] "
32
             self.logdebug = self.ca.logdebugbase + "[Icarus] "
33
             logging.info(self.loginfo + "initializing sensor object")
34
35
36
             self.minframe = 1
             self.maxframe = 2
37
38
             self.firstframe = self.minframe
             self.lastframe = self.maxframe
39
             # WARNING: the camera will always 'acquire' four frames, but will only generate
40
                 images for the middle two; HST and manual shutters will manage all four
41
42
                 frames
             self.nframes = self.maxframe - self.minframe + 1
43
             self.maxwidth = 512
44
             self.maxheight = 1024
45
46
             self.firstrow = 0
             self.lastrow = self.maxheight - 1
47
             self.width = self.maxwidth
self.height = self.maxheight
48
49
             self.bytesperpixel = 2
self.icarustype = 1  # 2-frame version
self.fpganumID = "1"  # last nybble of FPGA_NUM
50
51
52
53
             self.interlacing = 0
54
5.5
             self.sens_registers = OrderedDict(
56
57
                       "VRESET_WAIT_TIME": "03E",
                      "ICARUS_VER_SEL": "041",
59
                       "VRESET_HIGH_VALUE": "04A"
                      "MISC_SENSOR_CTL": "04C",
"MANUAL_SHUTTERS_MODE": "050",
60
61
62
                      "W0_INTEGRATION": "051",
                       "WO_INTERFRAME": "052",
64
                       "W1_INTEGRATION": "053",
                       "W1_INTERFRAME": "054",
65
                      "W2_INTEGRATION": "055",
                       "W2_INTERFRAME": "056",
                       "W3_INTEGRATION": "057"
69
                       "WO_INTEGRATION_B": "058",
                      "WO_INTERFRAME_B": "059",
71
                       "W1_INTEGRATION_B": "05A",
                       "W1_INTERFRAME_B": "05B",
                       "W2_INTEGRATION_B": "05C",
73
                      "W2_INTERFRAME_B": "05D",
74
75
                       "W3_INTEGRATION_B": "05E",
76
                       "TIME_ROW_DCD": "05F",
                  }
78
79
             self.sens_subregisters = [
                  ("MANSHUT_MODE", "MANUAL_SHUTTERS_MODE", 0, 1, True),
81
                  ("STAT_W3TOPLEDGE1", "STAT_REG", 3, 1, False),
("STAT_W3TOPREDGE1", "STAT_REG", 4, 1, False),
("STAT_HST_ALL_W_EN_DETECTED", "STAT_REG", 12, 1, False),
82
83
84
                  ("REVREAD", "CTRL_REG", 4, 1, True), ("PDBIAS_LOW", "CTRL_REG", 6, 1, True), ("ROWDCD_CTL", "CTRL_REG", 7, 1, True),
8.5
86
87
```

```
("PDBIAS_UNREADY", "STAT_REG2", 5, 1, False),
("ACCUMULATION_CTL", "MISC_SENSOR_CTL", 0, 1, True),
("HST_TST_ANRST_EN", "MISC_SENSOR_CTL", 1, 1, True),
("HST_TST_BNRST_EN", "MISC_SENSOR_CTL", 2, 1, True),
("HST_TST_ANRST_IN", "MISC_SENSOR_CTL", 3, 1, True),
("HST_TST_BNRST_IN", "MISC_SENSOR_CTL", 4, 1, True),
("HST_PXL_RST_EN", "MISC_SENSOR_CTL", 5, 1, True),
("HST_CONT_MODE", "MISC_SENSOR_CTL", 6, 1, True),
("COLDED_EN" "MISC_SENSOR_CTL", 7, 1, True),
88
89
90
91
93
94
95
                                   ("COL_DCD_EN", "MISC_SENSOR_CTL", 7, 1, True), ("COL_READOUT_EN", "MISC_SENSOR_CTL", 8, 1, True),
96
98
                        ]
100
                           if self.ca.boardname == "llnl_v1":
101
                                    self.sens_subregisters.append(
102
                                              ("VRESET_HIGH", "VRESET_HIGH_VALUE", 7, 8, True)
103
104
                           else:
105
                                  self.sens_subregisters.append(
                                              ("VRESET_HIGH", "VRESET_HIGH_VALUE", 15, 16, True)
106
107
108
```

8.5.3 Member Function Documentation

def nsCamera.sensors.icarus.icarus.checkSensorVoltStat (

8.5.3.1 checkSensorVoltStat()

```
self )
Checks register tied to sensor select jumpers to confirm match with sensor
  object
Returns:
    boolean, True if jumpers select for Icarus sensor
Definition at line 109 of file icarus.py.
109
       def checkSensorVoltStat(self):
110
            Checks register tied to sensor select jumpers to confirm match with sensor
111
112
             object
113
           Returns:
114
           boolean, True if jumpers select for Icarus sensor
115
116
117
            err, status = self.ca.getSubregister("ICARUS_DET")
118
           if err:
               logging.error(self.logerr + "unable to confirm sensor status")
119
120
               return False
121
            if not int(status):
               logging.error(self.logerr + "Icarus sensor not detected")
122
123
               return False
124
           return True
125
```

8.5.3.2 getManualTiming()

```
def nsCamera.sensors.icarus.icarus.getManualTiming (
                self )
Read off manual shutter settings
Returns:
    list of 2 lists of timing from A and B sides, respectively
Definition at line 538 of file icarus.py.
        def getManualTiming(self):
538
539
540
            Read off manual shutter settings
541
            Returns:
            list of 2 lists of timing from A and B sides, respectively \ensuremath{\text{\mbox{\tiny NUN}}}
542
543
544
            aside = []
            bside = []
545
546
            for reg in [
                 "WO_INTEGRATION",
547
                 "W0_INTERFRAME",
548
549
                 "W1_INTEGRATION",
550
                 "W1_INTERFRAME",
                 "W2_INTEGRATION",
5.51
                 "W2 INTERFRAME",
552
                "W3_INTEGRATION",
553
554
            ]:
                 _, reghex = self.ca.getRegister(reg)
555
556
                aside.append(25 * int(reghex, 16))
557
            for reg in [
                 "WO_INTEGRATION_B",
558
559
                 "W0_INTERFRAME_B",
                 "W1_INTEGRATION_B",
560
561
                "W1_INTERFRAME_B",
562
                 "W2_INTEGRATION_B"
563
                "W2_INTERFRAME_B"
564
                 "W3_INTEGRATION_B",
565
            ]:
566
                   reghex = self.ca.getRegister(reg)
567
                bside.append(25 * int(reghex, 16))
568
            return [aside, bside]
569
```

8.5.3.3 getTiming()

```
def nsCamera.sensors.icarus.icarus.getTiming (
              self,
              side,
              actual )
actual = True: returns actual high speed intervals that will be generated by the
 FPGA as list [delay, open0, closed0, open1, closed1, open2, closed2, open3]
actual = False: Returns high speed timing settings as set by setTiming. Assumes
  that timing was set via the setTiming method--it will not accurately report
 arbitrary timings set by direct register sets or manual shutter control.
Args:
    side: Hemisphere 'A' or 'B'
    actual: False: return HST settings
   True: calculate and return actual HST behavior
Returns:
    actual= False: tuple
                           (hemisphere label,
                    'open shutter' in ns,
                    'closed shutter' in ns,
                    initial delay in ns)
   True: list of relevant times [delay, open1, closed1, open2]
```

```
Definition at line 414 of file icarus.py.
        def getTiming(self, side, actual):
415
416
            actual = True: returns actual high speed intervals that will be generated by the
              FPGA as list [delay, open0, closed0, open1, closed1, open2, closed2, open3]
417
418
            actual = False: Returns high speed timing settings as set by setTiming. Assumes
419
              that timing was set via the setTiming method--it will not accurately report
420
              arbitrary timings set by direct register sets or manual shutter control.
421
422
            Args:
                side: Hemisphere 'A' or 'B'
423
                actual: False: return HST settings
424
425
                         True: calculate and return actual HST behavior
426
427
            Returns:
428
                actual= False: tuple
                                        (hemisphere label,
429
                                          open shutter' in ns,
                                          'closed shutter' in ns,
430
431
                                          initial delay in ns)
                         True: list of relevant times [delay, open1, closed1, open2]
432
433
            if side is None:
434
435
                side = "A"
436
437
            logging.info(self.loginfo + "get timing, side " + side.upper())
            if side.upper() == "A":
438
                lowreg = "HS_TIMING_DATA_ALO"
439
                highreg = "HS_TIMING_DATA_AHI"
440
            elif side.upper() == "B":
    lowreg = "HS_TIMING_DATA_BLO"
441
442
                highreg = "HS_TIMING_DATA_BHI"
443
444
            else:
                logging.error(
445
446
                    self.logerr
                    + "Invalid sensor side: "
447
448
                     + side
449
                    + "; timing settings unchanged"
450
            return "", 0, 0, 0
err, lowpart = self.ca.getRegister(lowreg)
451
452
453
            err1, highpart = self.ca.getRegister(highreg)
454
            if err or err1:
455
                logging.error(
                     self.logerr + "Unable to retrieve timing setting (getTiming), "
456
457
                     "returning zeroes "
458
459
                 return side.upper(), 0, 0, 0
            full40hex = highpart[-2:] + lowpart.zfill(8)
460
461
            full40bin = "{0:0=40b}".format(int(full40hex, 16))
462
            if actual:
463
                 full160 = 4 * full40bin
464
                 gblist = [[k, len(list(g))] for k, g in itertools.groupby(full160)]
                 times = [int(x[1]) for x in gblist[:-9:-1]]
465
466
                 times[0] = times[0] - 1
467
                 \# get timing for frames 1 and 2, keep delay as offset
468
                times12 = [times[0]] + times[3:6]
469
                 return times12
470
            else:
471
                gblist = [[k, len(list(g))] for k, g in itertools.groupby(full40bin)]
                delay = gblist[-1][1] - 1
472
473
                timeon = gblist[-2][1]
474
                if len(gblist) == 2: # 39,1 corner case
475
                    timeoff = 1
476
                elif len(gblist) == 3: # sequence fits only once
477
                    timeoff = 40 - timeon
478
                else:
479
                    timeoff = gblist[-3][1]
480
                return side.upper(), timeon, timeoff, delay
```

8.5.3.4 parseReadoff()

481

Dummy function; unnecessary for Icarus sensor

```
Definition at line 570 of file icarus.py.
```

```
570 def parseReadoff(self, frames):
571 """
572 Dummy function; unnecessary for Icarus sensor
573 """
574 return frames
575
```

8.5.3.5 reportStatusSensor()

Definition at line 576 of file icarus.py.

```
def reportStatusSensor(self, statusbits):
577
578
            Print status messages from sensor-specific bits of status register
579
580
            Args:
            statusbits: result of checkStatus()
581
582
            if int(statusbits[3]):
583
                logging.info(self.loginfo + "W3_Top_L_Edge1 detected")
584
585
            if int(statusbits[4]):
586
                logging.info(self.loginfo + "W3_Top_R_Edge1 detected")
587
            if int(statusbits[12]):
                logging.info(self.loginfo + "HST_All_W_En detected")
588
589
590
591 """
592 Copyright (c) 2022, Lawrence Livermore National Security, LLC. All rights reserved.
593 LLNL-CODE-838080
594
595 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
596 contract no. \overline{\text{DE}}-\text{AC52}-07\text{NA}27344 (Contract 44) between the U.S. Department of Energy
597 (DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
598 See license for disclaimers, notice of U.S. Government Rights and license terms and
599 conditions. 600 """
```

8.5.3.6 sensorSpecific()

```
Definition at line 126 of file icarus.py.
```

```
def sensorSpecific(self):
126
127
128
                list of tuples, (Sensor-specific register, default setting)
129
130
131
                icarussettings = [
132
                     ("ICARUS_VER_SEL", "00000001"),
                      ("FPA_FRAME_INITIAL", "00000001"),
("FPA_FRAME_FINAL", "00000002"),
("FPA_ROW_INITIAL", "00000000"),
("FPA_ROW_FINAL", "000003FF"),
133
134
135
136
                      ("VRESET_WAIT_TIME", "0000927CO"),
("HS_TIMING_DATA_BHI", "00000000"),
("HS_TIMING_DATA_BLO", "00006666"),
("HS_TIMING_DATA_AHI", "00000000"),
137
138
139
                                                                          \# 0db6 = 2-1; 6666 = 2-2
140
                      ("HS_TIMING_DATA_ALO", "00006666"),
141
142
143
                if self.ca.boardname == "llnl_v1":
144
                      icarussettings.append(
                            ("VRESET_HIGH_VALUE", "000000D5") # 3.3 V (FF = 3.96)
145
146
147
                else:
148
                     icarussettings.append(("VRESET_HIGH_VALUE", "0000FFFF"))
149
150
                return icarussettings
151
```

8.5.3.7 setArbTiming()

```
def nsCamera.sensors.icarus.icarus.setArbTiming (
                self,
                side.
                sequence )
Set arbitrary high-speed timing sequence. NOTE: Icarus sensors generally cannot
  use 1 ns timing, so should be at least 2 ns for frame 2 wnd 3 open and their
  interframe
Aras:
     side: Hemisphere 'A' or 'B'
     sequence: list of arbitrary timing intervals, beginning with initial delay.
       The conventional timing (5,2) with delay = 3 would be represented by
       [3,5,2,5,2,5,2,5]. NOTE: although Icarus only images the middle two
       frames, timing must be provided for all four frames; to implement frame 1
       open for X, shutter closed for Y, and frame 2 open for Z, use the
       seequence [0,1,1,X,Y,Z,1,1]
     *WARNING* arbitrary timings will not be restored after a board power cycle
Returns:
     list: Actual timing results
Definition at line 323 of file icarus.py.
323
        def setArbTiming(self, side, sequence):
324
325
            Set arbitrary high-speed timing sequence. NOTE: Icarus sensors generally cannot
              use 1 ns timing, so should be at least 2 ns for frame 2 wnd 3 open and their
326
327
              interframe
328
            Args:
329
                side: Hemisphere 'A' or 'B'
                sequence: list of arbitrary timing intervals, beginning with initial delay. The conventional timing (5,2) with delay = 3 would be represented by [3,5,2,5,2,5,2,5]. NOTE: although Icarus only images the middle two
330
331
332
                   frames, timing must be provided for all four frames; to implement frame 1
333
                  open for X, shutter closed for Y, and frame 2 open for Z, use the
334
                  seequence [0,1,1,X,Y,Z,1,1]
335
```

```
336
                *WARNING* arbitrary timings will not be restored after a board power cycle
337
338
            Returns:
            list: Actual timing results
339
340
            if side is None:
341
342
                side = "A"
343
            if sequence is None:
344
                sequence = [0, 3, 2, 3, 2, 3, 2, 3]
345
346
            logging.info(
347
                self.loginfo + "HST side " + side.upper() + " (arbitrary): " + str(sequence)
348
            if side.upper() == "A":
349
350
                lowreg = "HS_TIMING_DATA_ALO"
351
                highreg = "HS_TIMING_DATA_AHI"
352
            elif side.upper() == "B":
                lowreg = "HS_TIMING_DATA_BLO"
353
                highreg = "HS_TIMING_DATA_BHI"
354
355
            else:
356
                err = (
357
                    self.logerr
358
                    + "Invalid sensor side: "
359
                    + side
                    + "; timing settings unchanged"
360
361
362
                logging.error(err)
                return err, "0000000000"
363
            full40 = [0] * 40
364
            bitlist = []
365
            flag = 0 # similar to setTiming, but starts with delay
366
            sequence = sequence[:8] # need all 4 frames to work properly
367
            for a in sequence:
    add = [flag] * a
368
369
                bitlist += add
370
371
                if flag:
372
                    flag = 0
373
                else:
                   flag = 1
374
            reversedlist = bitlist[39::-1]
375
            376
377
378
            highpart = full40hex[-10:-8].zfill(8)
379
            lowpart = full40hex[-8:].zfill(8)
380
381
            self.ca.setRegister(lowreg, lowpart)
382
            self.ca.setRegister(highreg, highpart)
            self.ca.setRegister("HS_TIMING_CTL", "00000001")
383
384
            # deactivates manual shutter mode if previously engaged
385
            self.ca.setRegister("MANUAL_SHUTTERS_MODE", "00000000")
386
            actual = self.getTiming(side, actual=True)
387
            f0delay = sequence[1] + sequence[2]
            if actual != sequence[:1] + sequence[3:6]:
388
389
                logging.warning(
390
                    self.logwarn + "Due to sequence length and use of the Icarus model "
391
                    "1 sensor, the actual timing sequence for side '
392
                    + side
393
                    + " will be "
                    + "{"
394
395
                    + str(actual[0] + f0delay)
396
397
398
                    + str(actual[1 : 2 * self.nframes])
399
                )
            else:
400
401
                logging.warning(
                    self.logwarn + "Due to use of the Icarus model 1 sensor, the actual"
402
403
                    " timing sequence for side "
404
                    + side
405
                    + " will be "
                    + "{"
406
407
                    + str(actual[0] + f0delay)
408
409
410
                    + str(actual[1 : 2 * self.nframes])
411
412
            return actual
413
```

8.5.3.8 setHighFullWell()

self.logwarn + "HighFullWell mode is not supported by the Icarus "

8.5.3.9 setInterlacing()

if flag:

logging.warning(

"sensor. "

168 169

170

171 172

173 174

Definition at line 152 of file icarus.py.

```
152
        def setInterlacing(self, ifactor):
153
154
           Dummy function; feature is not implemented on Icarus2
155
156
           Returns:
           integer 1
157
158
           if ifactor:
159
160
161
                  self.logwarn + "Interlacing is not supported by the Icarus sensor."
162
163
           return 1
164
```

8.5.3.10 setManualShutters()

```
def nsCamera.sensors.icarus.icarus.setManualShutters (
                                timing )
Manual shutter timing, seven intervals for each side of the imager given in
    nanoseconds, e.g., [(100,50,100,50,100,50,100),(100,50,100,50,100,50,100)]
The timing list is flattened before processing; the suggested tuple structure is
     just for clarity (first tuple is A, second is B) and is optional.
The actual timing is rounded down to nearest multiple of 25 ns. (Each
    count = 25 ns. e.g., 140 ns rounds down to a count of '5' which corresponds
    to 125 ns))
        timing: 14-element list (substructure optional) in nanoseconds
Returns:
         tuple (error string, response string from final message)
Definition at line 482 of file icarus.py.
482
                def setManualShutters(self, timing):
483
                        Manual shutter timing, seven intervals for each side of the imager given in
484
485
                            nanoseconds, e.g., [(100,50,100,50,100,50,100),(100,50,100,50,100,50,100)]
486
487
                        The timing list is flattened before processing; the suggested tuple structure is
488
                            just for clarity (first tuple is A, second is B) and is optional.
489
490
                        The actual timing is rounded down to nearest multiple of 25 ns. (Each
                            count = 25 ns. e.g., 140 ns rounds down to a count of '5' which corresponds
491
492
                            to 125 ns))
493
494
                        Args:
495
                                timing: 14-element list (substructure optional) in nanoseconds
496
497
498
                                tuple (error string, response string from final message)
499
500
                        if timing is None:
501
502
                                timing = [
503
                                         (100, 50, 100, 50, 100, 50, 100),
504
                                         (100, 50, 100, 50, 100, 50, 100),
505
506
507
                        logging.info(self.loginfo + "Manual shutter sequence: " + str(timing))
                        flattened = self.ca.flatten(timing)
508
509
                        if len(flattened) != 14 or not all(type(x) is int for x in flattened):
                                err = self.logerr + "Invalid manual shutter timing list: " + str(timing)
510
                                logging.error(err + "; timing settings unchanged")
return err, "00000000"
511
512
513
514
                        timecounts = [a // 25 for a in flattened]
515
                        self.ca.sensmanual = timing
516
                        self.ca.senstiming = {} # clear HST settings from ca object
517
518
                        control_messages = [
                               trol_messages = [
  ("W0_INTEGRATION", "{0:#0{1}x}".format(timecounts[0], 10)[2:10]),
  ("W0_INTERFRAME", "{0:#0{1}x}".format(timecounts[1], 10)[2:10]),
  ("W1_INTEGRATION", "{0:#0{1}x}".format(timecounts[2], 10)[2:10]),
  ("W1_INTEGRATION", "{0:#0{1}x}".format(timecounts[3], 10)[2:10]),
  ("W2_INTEGRATION", "{0:#0{1}x}".format(timecounts[4], 10)[2:10]),
  ("W2_INTEGRATION", "{0:#0{1}x}".format(timecounts[5], 10)[2:10]),
  ("W3_INTEGRATION", "{0:#0{1}x}".format(timecounts[6], 10)[2:10]),
  ("W3_INTEGRATION", "{0:#0{1}x}".format(timecounts[6], 10)[2:10]),
  ("W0_INTEGRATION", "{0:#0{1}x}".format(timecounts[
519
520
521
522
523
524
525
                                ("W0_INTEGRATION", "{0:#0{1}x}".format(timecounts[7], 10)[2:10]), 
("W0_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[7], 10)[2:10]), 
("W0_INTERFRAME_B", "{0:#0{1}x}".format(timecounts[8], 10)[2:10]), 
("W1_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[9], 10)[2:10]), 
("W1_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[10], 10)[2:10]), 
("W2_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[11], 10)[2:10]),
526
527
528
529
530
```

def nsCamera.sensors.icarus.icarus.setTiming (

```
531 ("W2_INTERFRAME_B", "{0:#0{1}x}".format (timecounts[12], 10) [2:10]),
532 ("W3_INTEGRATION_B", "{0:#0{1}x}".format (timecounts[13], 10) [2:10]),
533 ("H5_TIMING_CTL", "00000000"),
534 ("MANUAL_SHUTTERS_MODE", "00000001"),
535 ]
536 return self.ca.submitMessages(control_messages, " setManualShutters: ")
537
```

8.5.3.11 setTiming()

```
self.
               side.
               sequence,
               delay )
Sets timing registers based on 'sequence.' WARNING: if entire sequence does not
  fit into the 40-bit register space, then the actual timings may differ from
  those requested. If the timing sequence fits only once into register space
  (i.e., for a single frame, open + closed > 19 ns ), then actual timing will be
  (n, 40-n) irrespective of setting of second parameter, e.g. (35,1) will
  actually result in (35,5) timing
NOTE: Icarus sensors generally cannot use 1 ns timing, so all values (besides the
  delay) should be at least 2 ns
Aras:
    side: Hemisphere 'A' or 'B'
    sequence: two-element tuple of timing durations in ns, e.g., '(5,2)'
    delay: initial delay in ns
Returns:
    tuple (error string, 10-character hexadecimal representation of timing
sequence)
Definition at line 195 of file icarus.py.
195
       def setTiming(self, side, sequence, delay):
196
197
           Sets timing registers based on 'sequence.' WARNING: if entire sequence does not
198
             fit into the 40-bit register space, then the actual timings may differ from
199
             those requested. If the timing sequence fits only once into register space
200
              (i.e., for a single frame, open + closed > 19 ns ), then actual timing will be
201
             (n, 40-n) irrespective of setting of second parameter, e.g. (35,1) will
202
             actually result in (35,5) timing
203
           NOTE: Icarus sensors generally cannot use 1 ns timing, so all values (besides the
             delay) should be at least 2 ns
205
           Args:
               side: Hemisphere 'A' or 'B'
206
207
               sequence: two-element tuple of timing durations in ns, e.g., '(5,2)'
208
               delay: initial delay in ns
209
210
           Returns:
211
               tuple (error string, 10-character hexadecimal representation of timing
212
                  sequence)
213
214
           if side is None:
               side = "A"
215
216
           if sequence is None:
217
               sequence = (3, 2)
218
           if delay is None:
219
               delay = 0
220
221
           if len(sequence) != 2:
222
               err = (
                   self.logerr
223
                   + "Invalid sequence setting for side: "
224
                   + side
225
```

```
+ "; timing settings are unchanged"
226
227
228
                logging.error(err)
                 return err, "0000000000"
229
230
            logging.info(
231
                self.loginfo
232
                 + "HST side "
233
                 + side.upper()
234
                + ": "
235
                + str(sequence)
                + "; delay =
236
237
                 + str(delay)
238
239
            if side.upper() == "A":
240
                 lowreg = "HS_TIMING_DATA_ALO"
241
                highreg = "HS_TIMING_DATA_AHI"
242
            elif side.upper() == "B":
                lowreg = "HS_TIMING_DATA_BLO"
243
                highreg = "HS_TIMING_DATA_BHI"
244
245
            else:
246
                err = (
247
                     self.logerr
248
                     + "Invalid sensor side: "
249
                     + side
                     + "; timing settings unchanged"
250
251
252
                logging.error(err)
                 return err, "0000000000"
253
            if (sequence[0] + sequence[1]) + delay > 40:
2.54
255
                err = (
256
                     self.logerr + "Timing sequence is too long to be implemented; "
257
                     "timing settings unchanged "
258
                logging.error(err)
return err, "0000000000"
259
260
2.61
2.62
            self.ca.senstiming[side.upper()] = (sequence, delay)
263
            self.ca.sensmanual = [] # clear manual settings from ca
2.64
            full40 = [0] * 40
265
266
            bitlist = []
267
            flag = 1
268
            sequence = sequence[:2]
269
            for a in sequence:
270
                add = [flag] * a
271
                bitlist += add
272
                if flag:
273
                    flag = 0
274
2.75
                     flag = 1
276
                 # automatically truncates sequence to 39 characters
2.77
                 reversedlist = bitlist[39::-1]
2.78
            repeats = (40 - delay) // len(reversedlist)
279
            # all four frames must be managed, even though only two are acquired
280
            if repeats > 4:
281
                repeats = 4
282
             # Pattern from sequence repeated to fit inside 40 bits up to a maximum of
283
                'nframes' times
284
            repeated = reversedlist * repeats
285
             full40[-(len(repeated) + delay + 1) : -(delay + 1)] = repeated
            full40bin = "".join(str(x) for x in full40)
full40hex = "%x" % int(full40bin, 2)
286
287
288
            highpart = full40hex[-10:-8].zfill(8)
            lowpart = full40hex[-8:].zfill(8)
289
            self.ca.setRegister(lowreg, lowpart)
290
291
            self.ca.setRegister(highreg, highpart)
            self.ca.setRegister("HS_TIMING_CTL", "00000001")
292
293
            # deactivates manual shutter mode if previously engaged
            self.ca.setRegister("MANUAL_SHUTTERS_MODE", "00000000")
294
295
            f0delay = sequence[0] + sequence[1]
296
            if repeats < 4:
297
                actual = self.getTiming(side, actual=True)
298
                expected = [delay] + list(sequence) + [sequence[0]]
                 if actual != expected:
299
300
                     logging.warning(
                         self.logwarn + "Due to sequence length and use of the Icarus "
301
302
                          "model 1 sensor, the actual timing sequence for side
303
                         + side
                         + " will be "
304
                         + "{"
305
                         + str(actual[0] + f0delay)
306
```

```
307
308
309
                         + str(actual[1 : 2 * self.nframes])
310
311
            else:
312
               logging.warning(
313
                    self.logwarn + "Due to use of the Icarus model 1 sensor, the"
314
                    " initial delay for side "
315
316
                    + " will actually be "
                    + str(delay + f0delay)
318
                     + " nanoseconds"
319
320
321
           return "", full40hex
```

8.5.3.12 setTriggerDelay()

```
def nsCamera.sensors.icarus.icarus.setTriggerDelay (
               delayblocks )
Dummy function; feature is not implemented on Icarus
Definition at line 185 of file icarus.py.
        def setTriggerDelay(self, delayblocks):
185
186
187
            Dummy function; feature is not implemented on Icarus
188
189
            if delayblocks:
190
               logging.warning(
                    self.logwarn + "Trigger Delay mode is not supported by the Icarus "
191
192
                    "sensor. '
193
194
```

8.5.3.13 setZeroDeadTime()

```
def nsCamera.sensors.icarus.icarus.setZeroDeadTime (
               self,
               flag )
Dummy function; feature is not implemented on Icarus
Definition at line 175 of file icarus.py.
175
        def setZeroDeadTime(self, flag):
176
177
            Dummy function; feature is not implemented on Icarus
178
179
            if flag:
               logging.warning(
180
                   self.logwarn + "ZeroDeadTime mode is not supported by the Icarus "
181
182
                    "sensor. "
183
```

184

8.5.4 Member Data Documentation

8.5.4.1 bytesperpixel

nsCamera.sensors.icarus.icarus.bytesperpixel

Definition at line 50 of file icarus.py.

8.5.4.2 ca

nsCamera.sensors.icarus.icarus.ca

Definition at line 28 of file icarus.py.

8.5.4.3 firstframe

nsCamera.sensors.icarus.icarus.firstframe

Definition at line 38 of file icarus.py.

8.5.4.4 firstrow

nsCamera.sensors.icarus.icarus.firstrow

Definition at line 46 of file icarus.py.

8.5.4.5 fpganumID

nsCamera.sensors.icarus.icarus.fpganumID

Definition at line 52 of file icarus.py.

8.5.4.6 height

nsCamera.sensors.icarus.icarus.height

Definition at line 49 of file icarus.py.

8.5.4.7 icarustype

nsCamera.sensors.icarus.icarus.icarustype

Definition at line 51 of file icarus.py.

8.5.4.8 interlacing

nsCamera.sensors.icarus.icarus.interlacing

Definition at line 53 of file icarus.py.

8.5.4.9 lastframe

nsCamera.sensors.icarus.icarus.lastframe

Definition at line 39 of file icarus.py.

8.5.4.10 lastrow

nsCamera.sensors.icarus.icarus.lastrow

Definition at line 47 of file icarus.py.

8.5.4.11 logcrit

nsCamera.sensors.icarus.icarus.logcrit

Definition at line 29 of file icarus.py.

8.5.4.12 logdebug

nsCamera.sensors.icarus.icarus.logdebug

Definition at line 33 of file icarus.py.

8.5.4.13 logerr

nsCamera.sensors.icarus.icarus.logerr

Definition at line 30 of file icarus.py.

8.5.4.14 loginfo

nsCamera.sensors.icarus.icarus.loginfo

Definition at line 32 of file icarus.py.

8.5.4.15 logwarn

nsCamera.sensors.icarus.icarus.logwarn

Definition at line 31 of file icarus.py.

8.5.4.16 maxframe

nsCamera.sensors.icarus.icarus.maxframe

Definition at line 37 of file icarus.py.

8.5.4.17 maxheight

nsCamera.sensors.icarus.icarus.maxheight

Definition at line 45 of file icarus.py.

8.5.4.18 maxwidth

nsCamera.sensors.icarus.icarus.maxwidth

Definition at line 44 of file icarus.py.

8.5.4.19 minframe

nsCamera.sensors.icarus.icarus.minframe

Definition at line 36 of file icarus.py.

8.5.4.20 nframes

nsCamera.sensors.icarus.icarus.nframes

Definition at line 43 of file icarus.py.

8.5.4.21 sens_registers

nsCamera.sensors.icarus.icarus.sens_registers

Definition at line 55 of file icarus.py.

8.5.4.22 sens_subregisters

 $\verb|nsCamera.sensors.icarus.icarus.sens_subregisters|\\$

Definition at line 80 of file icarus.py.

8.5.4.23 width

nsCamera.sensors.icarus.icarus.width

Definition at line 48 of file icarus.py.

The documentation for this class was generated from the following file:

nsCamera/sensors/icarus.py

8.6 nsCamera.sensors.icarus2.icarus2 Class Reference

Public Member Functions

- def __init__ (self, camassem)
- def checkSensorVoltStat (self)
- def sensorSpecific (self)
- def setInterlacing (self, ifactor)
- def setHighFullWell (self, flag)
- def setZeroDeadTime (self, flag)
- def setTriggerDelay (self, delayblocks)
- def setTiming (self, side, sequence, delay)
- def setArbTiming (self, side, sequence)
- def getTiming (self, side, actual)
- def setManualShutters (self, timing)
- def getManualTiming (self)
- def parseReadoff (self, frames)
- def reportStatusSensor (self, statusbits)

Public Attributes

- ca
- logcrit
- logerr
- logwarn
- loginfo
- logdebug
- · minframe
- maxframe
- firstframe
- lastframe
- nframes
- · maxwidth
- maxheight
- firstrow
- lastrow
- width
- height
- bytesperpixel
- icarustype
- fpganumID
- interlacing
- sens_registers
- sens_subregisters

8.6.1 Detailed Description

Definition at line 24 of file icarus2.py.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 __init__()

```
def nsCamera.sensors.icarus2.icarus2.__init__ (
                    camassem )
Definition at line 25 of file icarus2.py.
        def __init__(self, camassem):
    self.ca = camassem
26
27
              self.logcrit = self.ca.logcritbase + "[Icarus2] "
              self.logerr = self.ca.logerrbase + "[Icarus2]
28
2.9
              self.logwarn = self.ca.logwarnbase + "[Icarus2] "
              self.loginfo = self.ca.loginfobase + "[Icarus2] "
30
              self.logdebug = self.ca.logdebugbase + "[Icarus2] "
31
              logging.info(self.loginfo + "initializing sensor object")
32
33
              self.minframe = 0
              self.maxframe = 3
34
              self.firstframe = self.minframe
35
              self.lastframe = self.maxframe
36
              self.nframes = self.maxframe - self.minframe + 1
37
              self.maxwidth = 512
38
             self.maxheight = 1024
39
             self.firstrow = 0
40
              self.lastrow = self.maxheight - 1
41
              self.width = self.maxwidth
self.height = self.maxheight
42
43
44
              self.bytesperpixel = 2
              self.icarustype = 0  # 4-frame version
self.fpganumID = "1"  # last nybble of FPGA_NUM
4.5
46
47
              self.interlacing = 0
48
49
              self.sens_registers = OrderedDict(
50
                         "VRESET_WAIT_TIME": "03E",
51
52
                         "ICARUS_VER_SEL": "041",
53
                         "MISC_SENSOR_CTL": "04C"
54
                         "MANUAL_SHUTTERS_MODE": "050",
55
                         "W0_INTEGRATION": "051",
                        "WO_INTERFRAME": "052"
57
                         "W1_INTEGRATION": "053",
58
                         "W1_INTERFRAME": "054",
                         "W2_INTEGRATION": "055",
59
60
                        "W2_INTERFRAME": "056",
                         "W3_INTEGRATION": "057"
                         "WO_INTEGRATION_B": "058",
                         "WO_INTERFRAME_B": "059",
                        "W1_INTEGRATION_B": "05A",
                        "W1_INTERFRAME_B": "05B",
                        "W2_INTEGRATION_B": "05C",
                        "W2_INTERFRAME_B": "05D",
                        "W3_INTEGRATION_B": "05E",
                         "TIME_ROW_DCD": "05F",
                   }
71
72
73
              self.sens_subregisters = [
                 ("MANSHUT_MODE", "MANUAL_SHUTTERS_MODE", 0, 1, True),
                   ("STAT_W3TOPLEDGE1", "STAT_REG", 3, 1, False), ("STAT_W3TOPREDGE1", "STAT_REG", 4, 1, False),
76
                   ("STAT_HST_ALL_W_EN_DETECTED", "STAT_REG", 12, 1, False),
78
                   ("REVREAD", "CTRL_REG", 4, 1, True),
                   ("REVREAD", "CTRL_REG", 4, 1, True),

("PDBIAS_UNREADY", "STAT_REG2", 5, 1, False),

("PDBIAS_LOW", "CTRL_REG", 6, 1, True),

("ROWDCD_CTL", "CTRL_REG", 7, 1, True),

("ACCUMULATION_CTL", "MISC_SENSOR_CTL", 0, 1, True),

("HST_TST_ANRST_EN", "MISC_SENSOR_CTL", 1, 1, True),

("HST_TST_BNRST_EN", "MISC_SENSOR_CTL", 2, 1, True),

("HST_TST_ANRST_IN", "MISC_SENSOR_CTL", 3, 1, True),
79
80
81
82
83
84
8.5
```

```
86 ("HST_TST_BNRST_IN", "MISC_SENSOR_CTL", 4, 1, True),
87 ("HST_PXL_RST_EN", "MISC_SENSOR_CTL", 5, 1, True),
88 ("HST_CONT_MODE", "MISC_SENSOR_CTL", 6, 1, True),
89 ("COL_DCD_EN", "MISC_SENSOR_CTL", 7, 1, True),
90 ("COL_READOUT_EN", "MISC_SENSOR_CTL", 8, 1, True),
91 ]
92
```

8.6.3 Member Function Documentation

8.6.3.1 checkSensorVoltStat()

```
def nsCamera.sensors.icarus2.icarus2.checkSensorVoltStat (
               self )
Checks register tied to sensor select jumpers to confirm match with sensor
object
Returns:
    boolean, True if jumpers select for Icarus sensor
Definition at line 93 of file icarus2.py.
      def checkSensorVoltStat(self):
9.3
94
          Checks register tied to sensor select jumpers to confirm match with sensor
95
96
          object
97
98
          Returns:
           boolean, True if jumpers select for Icarus sensor
99
100
           err, status = self.ca.getSubregister("ICARUS_DET")
101
102
103
               logging.error(self.logerr + "unable to confirm sensor status")
104
105
           if not int(status):
106
               logging.error(self.logerr + "Icarus sensor not detected")
               return False
107
108
           return True
109
```

8.6.3.2 getManualTiming()

Definition at line 485 of file icarus2.py.

```
def getManualTiming(self):
486
487
            Read off manual shutter settings
488
            list of 2 lists of timing from A and B sides, respectively
489
490
491
            aside = []
492
            bside = []
            for reg in [
494
                "WO_INTEGRATION",
495
                "WO_INTERFRAME",
496
                "W1_INTEGRATION",
497
                "W1_INTERFRAME",
498
                "W2_INTEGRATION",
499
                "W2_INTERFRAME",
500
                "W3_INTEGRATION",
501
            1:
502
                 _, reghex = self.ca.getRegister(reg)
503
                aside.append(25 * int(reghex, 16))
504
            for reg in [
505
                "WO_INTEGRATION_B",
506
                "WO_INTERFRAME_B",
507
                "W1_INTEGRATION_B",
508
                "W1_INTERFRAME_B",
509
                "W2 INTEGRATION B",
510
                "W2_INTERFRAME_B",
                "W3_INTEGRATION_B",
511
512
           ]:
513
                  , reghex = self.ca.getRegister(reg)
                bside.append(25 * int(reghex, 16))
514
515
            return [aside, bside]
516
```

8.6.3.3 getTiming()

```
def nsCamera.sensors.icarus2.icarus2.getTiming (
              self,
              side.
              actual )
actual = True: returns actual high speed intervals that will be generated by the
   FPGA as list [delay, open0, closed0, open1, closed1, open2, closed2,
False: Returns high speed timing settings as set by setTiming. Assumes
   that timing was set via the setTiming method--it will not accurately
   report arbitrary timings set by direct register sets or manual
    shutter control
Args:
    side: Hemisphere 'A' or 'B'
    actual: False: return HST settings
    True: calculate and return actual HST behavior
Returns:
   actual= False: tuple (hemisphere label,
                    'open shutter' in ns,
                    'closed shutter' in ns,
                    initial delay in ns)
    True: list of times [delay, open0, closed0, open1, closed1, open2,
     closed2, open3]
```

```
Definition at line 363 of file icarus2.py.
```

```
def getTiming(self, side, actual):
364
365
            actual = True: returns actual high speed intervals that will be generated by the
366
                         FPGA as list [delay, open0, closed0, open1, closed1, open2, closed2,
367
                         open3]
368
                      False: Returns high speed timing settings as set by setTiming. Assumes
369
                         that timing was set via the setTiming method--it will not accurately
370
                         report arbitrary timings set by direct register sets or manual
371
                         shutter control
372
373
374
            Args:
375
                side: Hemisphere 'A' or 'B'
376
                actual: False: return HST settings
377
                         True: calculate and return actual HST behavior
378
379
            Returns:
380
                actual= False: tuple
                                         (hemisphere label,
381
                                          'open shutter' in ns,
                                          'closed shutter' in ns,
382
                                          initial delay in ns)
383
384
                         True: list of times [delay, open0, closed0, open1, closed1, open2,
385
                           closed2, open31
386
387
            if side is None:
                side = "A"
388
389
            logging.info(self.loginfo + "get timing, side " + side.upper())
if side.upper() == "A":
390
391
                lowreg = "HS_TIMING_DATA_ALO"
392
                highreg = "HS_TIMING_DATA_AHI"
393
            elif side.upper() == "B":
    lowreg = "HS_TIMING_DATA_BLO"
394
395
                highreg = "HS_TIMING_DATA_BHI"
396
397
            else:
398
                logging.error(
399
                    self.logerr
                     + "Invalid sensor side: "
400
401
                     + side
402
                     + "; timing settings unchanged"
403
            return "", 0, 0, 0
err, lowpart = self.ca.getRegister(lowreg)
404
405
406
            err1, highpart = self.ca.getRegister(highreg)
407
            if err or err1:
408
                logging.error(
                     self.logerr + "Unable to retrieve timing setting (getTiming), "
409
410
                     "returning zeroes "
411
412
                 return side.upper(), 0, 0, 0
413
            full40hex = highpart[-2:] + lowpart.zfill(8)
            full40bin = "{\tilde{0:0}=40b}".format(int(full40hex, 16))
414
415
             if actual:
416
                 full160 = 4 * full40bin
417
                gblist = [[k, len(list(g))] for k, g in itertools.groupby(full160)]
418
                 times = [int(x[1]) for x in gblist[:-9:-1]]
                times[0] = times[0] - 1
419
420
                return times
421
            else:
                gblist = [[k, len(list(g))] for k, g in itertools.groupby(full40bin)]
422
423
                delay = gblist[-1][1]
424
                timeon = gblist[-2][1]
                if len(gblist) < 4: # sequence fits only once</pre>
425
                    timeoff = 40 - timeon
426
427
                else:
                    timeoff = gblist[-3][1]
428
429
                return side.upper(), timeon, timeoff, delay
430
```

8.6.3.4 parseReadoff()

Dummy function; unnecessary for Icarus2 sensor

Definition at line 517 of file icarus2.py.

```
517 def parseReadoff(self, frames):
518 """
519 Dummy function; unnecessary for Icarus2 sensor
520 """
521 return frames
522
```

8.6.3.5 reportStatusSensor()

Definition at line 523 of file icarus2.py.

```
523
        def reportStatusSensor(self, statusbits):
524
525
            Print status messages from sensor-specific bits of status register
526
527
            Args:
            statusbits: result of checkStatus()
528
529
            if int(statusbits[3]):
530
531
                logging.info(self.loginfo + "W3_Top_L_Edge1 detected")
532
            if int(statusbits[4]):
533
                logging.info(self.loginfo + "W3_Top_R_Edge1 detected")
534
            if int(statusbits[12]):
                logging.info(self.loginfo + "HST_All_W_En detected")
535
536
537
538 """
539 Copyright (c) 2022, Lawrence Livermore National Security, LLC. All rights reserved.
540 LLNL-CODE-838080
541
542 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
543 contract no. \overline{\text{DE}}-\text{AC52}-07\text{NA}27344 (Contract 44) between the U.S. Department of Energy
544 (DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
545 See license for disclaimers, notice of U.S. Government Rights and license terms and
546 conditions. 547 """
```

8.6.3.6 sensorSpecific()

Definition at line 110 of file icarus2.py.

```
def sensorSpecific(self):
110
111
                  Returns:
112
                  list of tuples, (Sensor-specific register, default setting)
113
114
115
116
                        ("ICARUS_VER_SEL", "00000000"),
                         ("FPA_FRAME_INITIAL", "00000000"),
("FPA_FRAME_FINAL", "00000003"),
("FPA_ROW_INITIAL", "00000000"),
("FPA_ROW_FINAL", "000003FF"),
117
118
120
                         ("HS_TIMING_DATA_BHI", "00000000"),
("HS_TIMING_DATA_BLO", "00006666"),
121
122
                                                                                   \# 0db6 = 2-1; 6666 = 2-2
                         ("HS_TIMING_DATA_AHI", "00000000"), ("HS_TIMING_DATA_ALO", "00006666"),
123
125
126
```

8.6.3.7 setArbTiming()

Definition at line 287 of file icarus2.py.

```
def setArbTiming(self, side, sequence):
287
288
289
            Set arbitrary high-speed timing sequence. NOTE: Icarus sensors generally cannot
290
              use 1 ns timing, so all values (besides the delay) should be at least 2 ns
291
            Args:
                side: Hemisphere 'A' or 'B'
292
                 sequence: list of arbitrary timing intervals, beginning with initial delay.
293
294
                   The conventional timing (5,2) with delay = 3 would be represented by
295
                   [3,5,2,5,2,5,2,5].
296
                *WARNING* arbitrary timings will not be restored after a board power cycle
297
298
            Returns:
299
                tuple (error string, 10-character hexadecimal representation of timing
300
                  sequence)
301
302
            if side is None:
303
                side = "A"
304
            if sequence is None:
305
                sequence = [0, 3, 2, 3, 2, 3, 2, 3]
306
307
            logging.info(
                self.loginfo + "HST side " + side.upper() + " (arbitrary): " + str(sequence)
308
309
            if side.upper() == "A":
    lowreg = "HS_TIMING_DATA_ALO"
310
311
                highreg = "HS_TIMING_DATA_AHI"
312
```

```
313
             elif side.upper() == "B":
314
                 lowreg = "HS_TIMING_DATA_BLO"
315
                 highreg = "HS_TIMING_DATA_BHI"
316
317
                 err = (
318
                      self.logerr
319
                      + "Invalid sensor side: "
320
321
                      + "; timing settings unchanged"
322
323
                 logging.error(err)
                 return err, "0000000000"
324
             # TODO; restore arbitrary timing after power cycle?
325
             full40 = [0] * 40
326
             bitlist = []
flag = 0 # similar to setTiming, but starts with delay
327
328
             sequence = sequence[: (2 * self.nframes)]
329
330
             for a in sequence:
331
                 add = [flaq] * a
                 bitlist += add
332
                 if flag:
333
                     flag = 0
334
335
                 else:
336
                     flag = 1
             reversedlist = bitlist[39::-1]
337
             full40[-(len(reversedlist) + 1) : -1] = reversedlist
338
             full40bin = "".join(str(x) for x in full40) full40hex = "%x" % int(full40bin, 2)
339
340
             highpart = full40hex[-10:-8].zfill(8)
lowpart = full40hex[-8:].zfill(8)
341
342
343
             self.ca.setRegister(lowreg, lowpart)
             self.ca.setRegister(highreg, highpart)
344
             # deactivates manual shutter mode if previously engaged
345
346
             self.ca.setRegister("MANUAL_SHUTTERS_MODE", "00000000")
347
348
             actual = self.getTiming(side, actual=True)
349
             if actual != sequence:
350
                  logging.warning(
                      self.logwarn + "Due to sequence length, actual timing sequence " "for side "
351
352
353
                      + side
                      + " will be " + "{"
354
355
356
                      + str(actual[0])
                      + "}"
+ ""
357
358
359
                      + str(actual[1 : 2 * self.nframes])
360
                 )
361
             return actual
362
```

8.6.3.8 setHighFullWell()

```
self,
                flag )
Dummy function; feature is not implemented on Icarus2
Definition at line 140 of file icarus2.py.
140
        def setHighFullWell(self, flag):
141
142
            Dummy function; feature is not implemented on Icarus2
143
            if flag:
145
                logging.warning(
                    self.logwarn + "HighFullWell mode is not supported by the Icarus2 "
146
147
                    "sensor.
148
                )
149
```

def nsCamera.sensors.icarus2.icarus2.setHighFullWell (

8.6.3.9 setInterlacing()

```
def nsCamera.sensors.icarus2.icarus2.setInterlacing (
               self,
               ifactor )
Dummy function; feature is not implemented on Icarus2
Returns:
    integer 1
Definition at line 127 of file icarus2.py.
       def setInterlacing(self, ifactor):
128
129
           Dummy function; feature is not implemented on Icarus2
130
131
           Returns:
           integer 1
132
134
           if ifactor:
135
               logging.warning(
                  self.logwarn + "Interlacing is not supported by the Icarus2 sensor."
136
137
138
           return 1
139
```

8.6.3.10 setManualShutters()

440

```
def nsCamera.sensors.icarus2.icarus2.setManualShutters (
               self.
               timing )
Manual shutter timing, seven intervals for each side of the imager given in
  nanoseconds, e.g., [(100,50,100,50,100,50,100),(100,50,100,50,100,50,100)]
The timing list is flattened before processing; the suggested tuple structure is
  just for clarity (first tuple is A, second is B) and is optional.
The actual timing is rounded down to nearest multiple of 25 ns. (Each
  count = 25 ns. e.g., 140 ns rounds down to a count of '5' which corresponds
  to 125 ns))
    timing: 14-element list (substructure optional) in nanoseconds
Returns:
    tuple (error string, response string from final message)
Definition at line 431 of file icarus2.py.
431
       def setManualShutters(self, timing):
432
433
           Manual shutter timing, seven intervals for each side of the imager given in
             nanoseconds, e.g., [(100,50,100,50,100,50,100),(100,50,100,50,100,50,100)]
434
435
           The timing list is flattened before processing; the suggested tuple structure is
436
             just for clarity (first tuple is A, second is B) and is optional.
437
438
           The actual timing is rounded down to nearest multiple of 25 ns. (Each
439
             count = 25 ns. e.g., 140 ns rounds down to a count of '5' which corresponds
```

```
441
                             to 125 ns))
442
443
444
                                timing: 14-element list (substructure optional) in nanoseconds
445
446
                         tuple (error string, response string from final message)
447
448
449
                         if timing is None:
450
                                 timing = [
                                          (100, 50, 100, 50, 100, 50, 100),
451
                                          (100, 50, 100, 50, 100, 50, 100),
452
453
454
                         logging.info(self.loginfo + "Manual shutter sequence: " + str(timing))
455
                         flattened = self.ca.flatten(timing)
                         if len(flattened) != 14 or not all(type(x) is int for x in flattened):
                                 err = self.logerr + "Invalid manual shutter timing list: " + str(timing)
457
                                 logging.error(err + "; timing settings unchanged")
return err, "00000000"
458
459
460
461
                         timecounts = [a // 25 \text{ for a in flattened}]
462
                         self.ca.sensmanual = timing
463
                         self.ca.senstiming = {} # clear HST settings from ca object
464
465
                         control messages = [
                                 [rol_messages = [
("W0_INTEGRATION", "{0:#0{1}x}".format(timecounts[0], 10)[2:10]),
("W0_INTERFAME", "{0:#0{1}x}".format(timecounts[1], 10)[2:10]),
("W1_INTEGRATION", "{0:#0{1}x}".format(timecounts[2], 10)[2:10]),
("W1_INTERFRAME", "{0:#0{1}x}".format(timecounts[3], 10)[2:10]),
("W2_INTEGRATION", "{0:#0{1}x}".format(timecounts[4], 10)[2:10]),
("W2_INTEGRATION", "{0:#0{1}x}".format(timecounts[5], 10)[2:10]),
("W3_INTEGRATION", "{0:#0{1}x}".format(timecounts[6], 10)[2:10]),
("W3_INTEGRATION", "{0:#0{1}x}".format(timecounts[6], 10)[2:10]),
("W3_INTEGRATION P) " "{0:#0{1}x}" format(timecounts[7], 10)[2:10]),
466
467
468
469
470
471
472
                                 ("W3_INTEGRATION", "{0:#0{1}x}".format(timecounts[0], 10)[2:10]),
("W0_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[7], 10)[2:10]),
("W0_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[8], 10)[2:10]),
("W1_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[9], 10)[2:10]),
("W1_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[10], 10)[2:10]),
("W2_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[11], 10)[2:10]),
("W2_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[12], 10)[2:10]),
("W3_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[12], 10)[2:10]),
("W3_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[13], 10)[2:10]),
("W3_INTEGRATION_B", "{0:#0{1}x}".format(timecounts[13], 10)[2:10]),
473
474
475
476
477
478
479
                                  ("HS_TIMING_CTL", "00000000"),
480
481
                                  ("MANUAL_SHUTTERS_MODE", "00000001"),
482
483
                         return self.ca.submitMessages(control_messages, " setManualShutters: ")
484
```

8.6.3.11 setTiming()

```
def nsCamera.sensors.icarus2.icarus2.setTiming (
              self.
              side,
              sequence,
              delay )
Sets timing registers based on 'sequence.' WARNING: if entire sequence does not
  fit into the 40-bit register space, then the actual timings may differ from
  those requested. If the timing sequence fits only once into register space
  (i.e., for a single frame, open + closed > 19 ns), then actual timing will be
  (n, 40-n) irrespective of setting of second parameter, e.g. (35,1) will
  actually result in (35,5) timing.
NOTE: Icarus sensors generally cannot use 1 ns timing, so all values (besides
 the delay) should be at least 2 ns
Aras:
    side: Hemisphere 'A' or 'B'
    sequence: two-element tuple of timing durations in ns, e.g., '(5,2)'
    delay: initial delay in ns
Returns:
    tuple (error string, 10-character hexadecimal representation of timing
     sequence)
```

Definition at line 170 of file icarus2.py.

```
170
        def setTiming(self, side, sequence, delay):
171
172
            Sets timing registers based on 'sequence.' WARNING: if entire sequence does not
173
              fit into the 40-bit register space, then the actual timings may differ from
174
              those requested. If the timing sequence fits only once into register space
175
               (i.e., for a single frame, open + closed > 19 ns), then actual timing will be
176
               (n, 40-n) irrespective of setting of second parameter, e.g. (35,1) will
177
              actually result in (35,5) timing.
178
            NOTE: Icarus sensors generally cannot use 1 ns timing, so all values (besides
179
              the delay) should be at least 2 ns
180
181
            Args:
                side: Hemisphere 'A' or 'B'
182
183
                sequence: two-element tuple of timing durations in ns, e.g., '(5,2)'
184
                delay: initial delay in ns
185
186
187
                tuple (error string, 10-character hexadecimal representation of timing
188
                  sequence)
189
190
            if side is None:
191
                side = "A"
192
            if sequence is None:
193
                sequence = (3, 2)
194
            if delay is None:
195
                delay = 0
196
197
            if len(sequence) != 2:
198
                err = (
199
                    self.logerr
                     + "Invalid sequence setting for side: "
200
201
                    + side
                    + "; timing settings are unchanged"
2.02
203
2.04
                logging.error(err)
                return err, "0000000000"
2.05
206
            logging.info(
2.07
                self.loginfo
                + "HST side "
208
209
                + side.upper()
210
                + ": "
211
                + str(sequence)
212
                + "; delay =
213
                + str(delay)
214
215
            if side.upper() == "A":
                lowreg = "HS_TIMING_DATA_ALO"
highreg = "HS_TIMING_DATA_AHI"
216
217
218
            elif side.upper() == "B":
                lowreg = "HS_TIMING_DATA_BLO"
219
                highreg = "HS_TIMING_DATA_BHI"
220
221
            else:
222
223
                     self.logerr
224
                     + "Invalid sensor side: "
225
                     + side
226
                     + "; timing settings unchanged"
227
228
                logging.error(err)
                 return err, "0000000000"
229
230
            if (sequence[0] + sequence[1]) + delay > 40:
231
                err = (
232
                    self.logerr + "Timing sequence is too long to be implemented; "
                     "timing settings unchanged "
233
235
                logging.error(err)
236
                return err, "0000000000"
237
238
            self.ca.senstiming[side.upper()] = (sequence, delay)
239
            self.ca.sensmanual = [] # clear manual settings from ca
240
241
            full40 = [0] * 40
            bitlist = []
242
243
            flag = 1
244
            sequence = sequence[:2]
245
            for a in sequence:
                add = [flaq] * a
246
247
                bitlist += add
                if flag:
2.48
                    flag = 0
249
```

```
250
251
                   flag = 1
252
            # automatically truncates sequence to 39 characters
253
            reversedlist = bitlist[39::-1]
254
            repeats = (40 - delay) // len(reversedlist)
255
            if repeats > self.nframes:
256
                repeats = self.nframes
257
            # Pattern from sequence repeated to fit inside 40 bits up to a maximum of
258
            # 'nframes' times
            repeated = reversedlist * repeats
            full40[-(len(repeated) + delay + 1) : -(delay + 1)] = repeated
260
            full40bin = "".join(str(x) for x in full40)
261
262
            full40hex = "%x" % int(full40bin, 2)
            highpart = full40hex[-10:-8].zfill(8)
263
264
            lowpart = full40hex[-8:].zfill(8)
265
            self.ca.setRegister(lowreg, lowpart)
266
            self.ca.setRegister(highreg, highpart)
            self.ca.setRegister("HS_TIMING_CTL", "00000001")
267
268
            # deactivates manual shutter mode if previously engaged
            self.ca.setRegister("MANUAL_SHUTTERS_MODE", "00000000")
269
270
            if repeats < self.nframes:</pre>
                actual = self.getTiming(side, actual=True)
271
                expected = [delay] + 3 * list(sequence) + [sequence[0]]
272
                if actual != expected:
273
274
                    logging.warning(
                        self.logwarn + "Due to sequence length, actual timing "
275
276
                         "sequence for side
277
                         + side
                        + " will be "
278
                        + "{"
279
280
                         + str(actual[0])
281
                        + " "
282
                         + str(actual[1 : 2 * self.nframes])
2.83
284
            return "", full40hex
285
286
```

8.6.3.12 setTriggerDelay()

```
def nsCamera.sensors.icarus2.icarus2.setTriggerDelay (
               self,
               delayblocks )
Dummy function; feature is not implemented on Icarus2
Definition at line 160 of file icarus2.py.
160
        def setTriggerDelay(self, delayblocks):
161
162
            Dummy function; feature is not implemented on Icarus2
163
164
            if delayblocks:
165
               logging.warning(
166
                   self.logwarn + "Trigger Delay is not supported by the Icarus2 "
167
                    "sensor.
168
169
```

8.6.3.13 setZeroDeadTime()

```
def nsCamera.sensors.icarus2.icarus2.setZeroDeadTime (
              self,
              flag )
Dummy function; feature is not implemented on Icarus2
```

Definition at line 150 of file icarus2.py.

```
def setZeroDeadTime(self, flag):
150
151
152
            Dummy function; feature is not implemented on Icarus2
153
           if flag:
154
155
                logging.warning(
                    self.logwarn + "ZeroDeadTime mode is not supported by the Icarus2 "
156
157
                    "sensor. "
158
159
```

8.6.4 Member Data Documentation

8.6.4.1 bytesperpixel

nsCamera.sensors.icarus2.icarus2.bytesperpixel

Definition at line 44 of file icarus2.py.

8.6.4.2 ca

nsCamera.sensors.icarus2.icarus2.ca

Definition at line 26 of file icarus2.py.

8.6.4.3 firstframe

nsCamera.sensors.icarus2.icarus2.firstframe

Definition at line 35 of file icarus2.py.

8.6.4.4 firstrow

nsCamera.sensors.icarus2.icarus2.firstrow

Definition at line 40 of file icarus2.py.

8.6.4.5 fpganumID

nsCamera.sensors.icarus2.icarus2.fpganumID

Definition at line 46 of file icarus2.py.

8.6.4.6 height

nsCamera.sensors.icarus2.icarus2.height

Definition at line 43 of file icarus2.py.

8.6.4.7 icarustype

nsCamera.sensors.icarus2.icarus2.icarustype

Definition at line 45 of file icarus2.py.

8.6.4.8 interlacing

nsCamera.sensors.icarus2.icarus2.interlacing

Definition at line 47 of file icarus2.py.

8.6.4.9 lastframe

 $\verb|nsCamera.sensors.icarus2.icarus2.lastframe| \\$

Definition at line 36 of file icarus2.py.

8.6.4.10 lastrow

nsCamera.sensors.icarus2.icarus2.lastrow

Definition at line 41 of file icarus2.py.

8.6.4.11 logcrit

nsCamera.sensors.icarus2.icarus2.logcrit

Definition at line 27 of file icarus2.py.

8.6.4.12 logdebug

nsCamera.sensors.icarus2.icarus2.logdebug

Definition at line 31 of file icarus2.py.

8.6.4.13 logerr

nsCamera.sensors.icarus2.icarus2.logerr

Definition at line 28 of file icarus2.py.

8.6.4.14 loginfo

nsCamera.sensors.icarus2.icarus2.loginfo

Definition at line 30 of file icarus2.py.

8.6.4.15 logwarn

nsCamera.sensors.icarus2.icarus2.logwarn

Definition at line 29 of file icarus2.py.

8.6.4.16 maxframe

nsCamera.sensors.icarus2.icarus2.maxframe

Definition at line 34 of file icarus2.py.

8.6.4.17 maxheight

nsCamera.sensors.icarus2.icarus2.maxheight

Definition at line 39 of file icarus2.py.

8.6.4.18 maxwidth

nsCamera.sensors.icarus2.icarus2.maxwidth

Definition at line 38 of file icarus2.py.

8.6.4.19 minframe

nsCamera.sensors.icarus2.icarus2.minframe

Definition at line 33 of file icarus2.py.

8.6.4.20 nframes

nsCamera.sensors.icarus2.icarus2.nframes

Definition at line 37 of file icarus2.py.

8.6.4.21 sens_registers

nsCamera.sensors.icarus2.icarus2.sens_registers

Definition at line 49 of file icarus2.py.

8.6.4.22 sens_subregisters

```
nsCamera.sensors.icarus2.icarus2.sens_subregisters
```

Definition at line 73 of file icarus2.py.

8.6.4.23 width

```
nsCamera.sensors.icarus2.icarus2.width
```

Definition at line 42 of file icarus2.py.

The documentation for this class was generated from the following file:

nsCamera/sensors/icarus2.py

8.7 nsCamera.boards.LLNL_v1.llnl_v1 Class Reference

Public Member Functions

- def __init__ (self, camassem)
- def initBoard (self)
- def initPots (self)
- def latchPots (self)
- def initSensor (self)
- · def configADCs (self)
- def softReboot (self)
- def disarm (self)
- def startCapture (self, mode="Hardware")
- def readSRAM (self)
- def waitForSRAM (self, timeout)
- def getTimer (self)
- def resetTimer (self)
- def enableLED (self, status)
- def setLED (self, LED, status)
- def setPowerSave (self, status)
- def setPPER (self, time)
- def getTemp (self, scale)
- def getPressure (self, offset, sensitivity, units)
- def clearStatus (self)
- · def checkStatus (self)
- def checkStatus2 (self)
- def reportStatus (self)
- def reportEdgeDetects (self)
- def dumpStatus (self)

Public Attributes

- ca
- logcrit
- logerr
- logwarn
- loginfo
- logdebug
- VREF
- ADC5 mult
- ADC5_bipolar
- rs422_baud
- rs422_cmd_wait
- subreg_aliases
- monitor_controls
- · subreglist

Static Public Attributes

- · registers
- list subregisters
- list dummySensorVals

8.7.1 Detailed Description

```
Livermore LLNL v1.0 board

Compatible communication protocols: RS422, GigE

Compatible sensors: icarus, icarus2, daedalus
```

Definition at line 29 of file LLNL_v1.py.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 __init__()

```
def nsCamera.boards.LLNL_v1.llnl_v1.__init___ (
                   self,
                   camassem )
Definition at line 233 of file LLNL_v1.py.
233
         def __init__(self, camassem):
              self.ca = camassem
              self.logcrit = self.ca.logcritbase + "[LLNL_v1] "
235
              self.logerr = self.ca.logerrbase + "[LLNL_v1] "
236
              self.logwarn = self.ca.logwarnbase + "[LLNL_v1] "
237
              self.loginfo = self.ca.loginfobase + "[LLNL_v1] "
238
239
              self.logdebug = self.ca.logdebugbase + "[LLNL_v1]
240
              logging.info(self.loginfo + "initializing board object")
              self.VREF = 2.5 # default
self.ADC5_mult = 2 # monmax = 2 * VREF
241
242
              # False => monitor range runs 0 to monmax, True => +/- monmax
243
              self.ADC5_bipolar = True
244
245
246
              self.rs422 baud = 921600
              self.rs422_cmd_wait = 0.3
247
248
              fpgaNum_pkt = Packet(cmd="1", addr=self.registers["FPGA_NUM"])
fpgaRev_pkt = Packet(cmd="1", addr=self.registers["FPGA_REV"])
249
250
2.51
              _, _ = self.ca.sendCMD(fpgaNum_pkt) # dummy duplicate call
252
253
              err, rval = self.ca.sendCMD(fpgaNum_pkt)
2.54
              self.ca.FPGANum = rval[8:16]
255
              err, rval = self.ca.sendCMD(fpgaRev_pkt)
2.56
2.57
              self.ca.FPGAVersion = rval[8:16]
258
              # map channels to signal names for abstraction at the camera assembler level;
# each requires a corresponding entry in 'subregisters'
2.59
260
              if self.ca.sensorname == "icarus" or self.ca.sensorname == "icarus2":
    self.subreg_aliases = OrderedDict(
261
2.62
263
                             "COL_BOT_IBIAS_IN": "POT1",
2.64
                             "HST_A_PDELAY": "POT2", "HST_B_NDELAY": "POT3",
2.65
266
                             "HST_RO_IBIAS": "POT4"
2.67
268
                             "HST_OSC_VREF_IN": "POT5",
                             "HST_B_PDELAY": "POT6",
269
                             "HST_OSC_CTL": "POT7",
"HST_A_NDELAY": "POT8",
270
271
272
                             "COL_TOP_IBIAS_IN": "POT9",
                             "HST_OSC_R_BIAS": "POT10",
"VAB": "POT11",
273
274
2.75
                             "HST_RO_NC_IBIAS": "POT12",
276
                             "VRST": "POT13",
277
                             "MON_HST_A_PDELAY": "MON_CH2",
                             "MON_HST_B_NDELAY": "MON_CH3",
"MON_HST_RO_IBIAS": "MON_CH4",
278
279
280
                             "MON_HST_OSC_VREF_IN": "MON_CH5",
                             "MON_HST_B_PDELAY": "MON_CH6",
"MON_HST_OSC_CTL": "MON_CH7",
281
282
                             "MON_HST_A_NDELAY": "MON_CH8",
283
284
285
286
                   # Read-only; identifies controls corresponding to monitors
287
                   self.monitor_controls = OrderedDict(
288
289
                             "MON_CH2": "POT2",
                             "MON_CH3": "POT3",
290
291
                             "MON_CH4": "POT4",
                             "MON_CH5": "POT5",
292
                             "MON_CH6": "POT6",
293
                             "MON_CH7": "POT7",
"MON_CH8": "POT8",
294
295
                             # Note: VRST is not measured across the pot; it will read a voltage
296
297
                                approximately 1 Volt lower than pot13's actual output
                             "MON_VRST": "POT13",
298
299
300
301
              else: # Daedalus
                   self.subreg_aliases = OrderedDict(
302
```

```
303
304
                          "HST_OSC_CTL": "POT4",
305
                          "HST_RO_NC_IBIAS": "POT5",
                          "HST_OSC_VREF_IN": "POT6",
306
                          "VAB": "POT11",
"MON_TSENSEOUT": "MON_CH2",
307
308
309
                          "MON_BGREF": "MON_CH3",
310
                          "MON_HST_OSC_CTL": "MON_CH4",
311
                          "MON_HST_RO_NC_IBIAS": "MON_CH5",
                          "MON_HST_OSC_VREF_IN": "MON_CH6",
312
                          "MON_COL_TST_IN": "MON_CH7",
313
314
                          "MON_HST_OSC_PBIAS_PAD": "MON_CH8",
315
                     }
316
317
                 # Read-only; identifies controls corresponding to monitors
318
                 self.monitor_controls = OrderedDict(
319
320
                          "MON_CH4": "POT4",
                          "MON_CH5": "POT5",
321
                          "MON_CH6": "POT6",
322
                          \ensuremath{\sharp} Note: VRST is not measured across the pot; it will read a voltage
323
324
                           lower than pot13's actual output
325
                          "MON_VRST": "POT13",
326
                     }
327
                 )
328
329
             self.subreglist = []
330
             for s in self.subregisters:
331
                 self.subreglist.append(s[0].upper())
332
                 sr = SubRegister(
333
                     self,
                     name=s[0].upper(),
334
335
                     register=s[1].upper(),
                     start_bit=s[2],
336
337
                     width=s[3].
338
                     writable=s[4],
339
340
                 setattr(self, s[0].upper(), sr)
341
            # set voltage ranges for all pots
342
            for n in range(1, 13):
    potname = "POT" + str(n)
343
344
345
                 potobj = getattr(self, potname)
346
                 potobj.minV = 0
                 potobj.maxV = 3.3
347
348
                 \mbox{\#} resolution is approximately .0129 V / LSB
349
                 potobj.resolution = (1.0 * potobj.maxV - potobj.minV) / potobj.max_value
350
            self.POT13.minV = 0
351
            self.POT13.maxV = 3.96
352
             \# POT13 resolution is approximately .0155 V / LSB
353
            self.POT13.resolution = (
354
                 1.0 * self.POT13.maxV - self.POT13.minV
355
             ) / self.POT13.max_value
356
```

8.7.3 Member Function Documentation

8.7.3.1 checkStatus()

```
Definition at line 773 of file LLNL_v1.py.
```

```
773
        def checkStatus(self):
774
775
            Check status register, convert to reverse-order bit stream (i.e., bit 0 is
776
             statusbits[0])
777
778
            bit string (no '0b') in reversed order
779
780
781
            err, rval = self.ca.getRegister("STAT_REG")
782
            if not rval:
783
                logging.error(
784
                   self.logerr + "Unable to check status register (zeroes returned)"
785
786
                rval = "0"
            rvalbits = bin(int(rval, 16))[2:].zfill(32)
788
            statusbits = rvalbits[::-1]
789
            return statusbits # TODO: add error handling
790
```

8.7.3.2 checkStatus2()

```
def nsCamera.boards.LLNL_v1.llnl_v1.checkStatus2 (
               self )
Check second status register, convert to reverse-order bit stream (i.e., bit 0
  is statusbits[0])
Returns: bit string (no '0b') in reversed order
Definition at line 791 of file LLNL_v1.py.
791
       def checkStatus2(self):
792
793
            Check second status register, convert to reverse-order bit stream (i.e., bit 0
794
             is statusbits[0])
795
796
            Returns: bit string (no '0b') in reversed order
797
798
            err, rval = self.ca.getRegister("STAT_REG2")
799
            if not rval:
               logging.error(
800
801
                   self.logerr + "Unable to check status register 2 (zeroes returned)"
802
               rval = "0"
            rvalbits = bin(int(rval, 16))[2:].zfill(5)
804
805
            statusbits = rvalbits[::-1]
806
            return statusbits # TODO: add error handling
807
```

8.7.3.3 clearStatus()

```
Definition at line 759 of file LLNL_v1.py.
```

```
759
        def clearStatus(self):
760
            Check status registers to clear them
761
762
763
            error string
764
765
766
            err1, rval = self.ca.getRegister("STAT_REG_SRC")
767
            err2, rval = self.ca.getRegister("STAT_REG2_SRC")
768
            err = err1 + err2
769
770
                logging.error(self.logerr + "clearStatus failed")
771
            return err
772
```

8.7.3.4 configADCs()

Definition at line 479 of file LLNL_v1.py.

```
479
          def configADCs(self):
480
481
               Sets default ADC configuration (does not latch settings)
482
483
               tuple (error string, response string) from final control message
484
485
486
               logging.info(self.loginfo + "configADCs")
487
488
               control messages = [
489
                    # just in case ADC_RESET was set (pull all ADCs out # of reset)
490
                          "ADC RESET",
491
                         "00000000",
492
493
                    # workaround for uncertain behavior after previous readoff
494
495
                         "ADC1_CONFIG_DATA",
496
497
                         "FFFFFFFF",
498
                    ("ADC2_CONFIG_DATA", "FFFFFFFF"),
499
                    ("ADC3_CONFIG_DATA", "FFFFFFFF"),
("ADC4_CONFIG_DATA", "FFFFFFFF"),
500
501
                    ("ADC_CTL", "FFFFFFFF"),
502
                    ("ADC1_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
("ADC2_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
("ADC3_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
503
504
505
                    ("ADC4_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
("ADC5_CONFIG_DATA", "81A883FF"), # int Vref 2.50V
506
507
508
509
               return self.ca.submitMessages(control_messages, " configADCs: ")
510
```

8.7.3.5 disarm()

```
def nsCamera.boards.LLNL_v1.llnl_v1.disarm (
                self )
Takes camera out of trigger wait state. Has no effect if camera is not already
Returns:
    tuple (error string, response string) from final control message
Definition at line 523 of file LLNL v1.py.
        def disarm(self):
523
524
525
            Takes camera out of trigger wait state. Has no effect if camera is not already
526
              in wait state.
527
528
            Returns:
            tuple (error string, response string) from final control message
529
530
            logging.info(self.loginfo + "disarm")
531
532
            self.ca.clearStatus()
            self.ca.armed = False
533
534
            control_messages = [
    ("HW_TRIG_EN", "0"),
535
                ("DUAL_EDGE_TRIG_EN", "0"),
536
                ("SW_TRIG_EN", "0"),
537
538
            return self.ca.submitMessages(control_messages, " disarm: ")
539
540
```

8.7.3.6 dumpStatus()

Definition at line 866 of file LLNL_v1.py.

```
def dumpStatus(self):
866
867
868
             Create dictionary of status values, DAC settings, monitor values, and register
869
870
871
             WARNING: the behavior of self-resetting subregisters may be difficult to predict
872
              and may generate contradictory results
873
874
             Returns:
            dictionary of system diagnostic values _{\mbox{\scriptsize min}}
875
876
877
             statusbits = self.checkStatus()
             statusbits2 = self.checkStatus2()
878
879
             temp = self.ca.getTemp()
880
```

```
881
            statDict = OrderedDict(
882
883
                    "Temperature reading": "\{0:1.2f\}".format(temp) + " C",
884
                    "Sensor read complete": str(statusbits[0]),
885
                    "Coarse trigger detected": str(statusbits[1]),
886
                    "Fine trigger detected": str(statusbits[2]),
887
                    "Sensor readout in progress": str(statusbits[5]),
888
                    "Sensor readout complete": str(statusbits[6]),
889
                    "SRAM readout started": str(statusbits[7]),
                    "SRAM readout complete": str(statusbits[8]),
                    "High-speed timing configured": str(statusbits[9]),
891
                    "All ADCs configured": str(statusbits[10]),
893
                    "All pots configured": str(statusbits[11])
                    "HST_All_W_En detected": str(statusbits[12]),
895
                    "Timer has reset": str(statusbits[13]),
                    "Camera is Armed": str(statusbits[14]),
897
                    "FPA_IF_TO": str(statusbits2[0]),
                    "SRAM_RO_TO": str(statusbits2[1]),
898
899
                    "PixelRd Timeout Error": str(statusbits2[2]),
                    "UART_TX_TO_RST": str(statusbits2[3]),
900
901
                    "UART_RX_TO_RST": str(statusbits2[4]),
902
                }
903
            )
904
905
            DACDict = OrderedDict()
            MonDict = OrderedDict()
906
907
            for entry in self.subreq_aliases:
                if self.subreg_aliases[entry][0] == "P":
908
                    val = str(round(self.ca.getPotV(entry), 3)) + " V"
909
                    DACDict["POT_" + entry] = val
910
911
                else:
                    val = str(round(self.ca.getMonV(entry), 3)) + " V"
912
                    MonDict[entry] = val
913
914
915
            regDict = OrderedDict()
916
            for key in self.registers.keys():
917
                err, rval = self.ca.getRegister(key)
918
                regDict[key] = rval
919
            dumpDict = OrderedDict()
920
921
            for x in [statDict, MonDict, POTDict, regDict]:
922
                dumpDict.update(x)
923
            return dumpDict
924
925
926 """
927 Copyright (c) 2022, Lawrence Livermore National Security, LLC. All rights reserved.
928 LLNL-CODE-838080
929
930 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
931 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy
932 (DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
933 See license for disclaimers, notice of U.S. Government Rights and license terms and
934 conditions.
935 """
```

8.7.3.7 enableLED()

```
Definition at line 655 of file LLNL_v1.py.
```

```
def enableLED(self, status):
656
657
            Enable/disable on-board LEDs
658
659
660
               status: 0 for disabled, 1 for enabled
661
662
            tuple: (error string, response string from subregister set)
663
664
665
           if status:
666
               status = 1
667
            return self.ca.setSubregister("LED_EN", str(status))
668
```

8.7.3.8 getPressure()

```
def nsCamera.boards.LLNL_v1.llnl_v1.getPressure (
                 self,
                 offset,
                 sensitivity,
                 units )
Read pressure sensor
Currently unimplemented
Returns:
    0 as float
Definition at line 745 of file LLNL_v1.py.

745 def getPressure(self, offset, sensitivity, units):
746 """
746
747
             Read pressure sensor
748
749
            Currently unimplemented
750
751
             Returns:
             0 as float
752
753
754
             logging.warning(
                 "WARNING: [LLNL_v1] 'getPressure' is not implemented on the LLNLv1 board"
755
756
757
             return 0.0
```

8.7.3.9 getTemp()

758

```
Definition at line 720 of file LLNL_v1.py.
```

```
720
        def getTemp(self, scale):
721
722
            Read temperature sensor
723
724
                scale: temperature scale to report (defaults to C, options are F and K)
725
            temperature as float on given scale
726
727
728
            err, rval = self.ca.getRegister("TEMP_SENSE_DATA")
729
            if err:
730
                logging.error(
                    self.logerr + "unable to retrieve temperature information ("
'getTemp), returning "0" '
731
732
733
                return 0.0
735
            ctemp = int(rval[-3:], 16) / 16.0
736
            if scale == "K":
737
738
                temp = ctemp + 273.15
            elif scale == "F":
739
                temp = 1.8 * ctemp + 32
740
741
            else:
742
               temp = ctemp
743
            return temp
744
```

8.7.3.10 getTimer()

Definition at line 628 of file LLNL v1.py.

```
def getTimer(self):
628
629
            Read value of on-board timer
630
631
632
            Returns:
            timer value as integer
633
634
            err, rval = self.ca.getRegister("TIMER_VALUE")
635
636
            if err:
                    self.logerr + "unable to retrieve timer information (getTimer), " 'returning "0" '
637
                logging.error(
638
639
640
641
                return 0
642
            return int(rval, 16)
643
```

8.7.3.11 initBoard()

```
def nsCamera.boards.LLNL_v1.llnl_v1.initBoard (
               self )
Register and reset board, set up firmware for sensor
Returns:
    tuple (error string, response string) from final control message
Definition at line 357 of file LLNL v1.py.
        def initBoard(self):
357
358
359
            Register and reset board, set up firmware for sensor
360
361
            Returns:
            tuple (error string, response string) from final control message
362
363
            logging.info(self.loginfo + "initBoard")
364
            control_messages = [("LED_EN", "1")]
365
366
367
            self.clearStatus()
368
            self.configADCs()
369
            err, resp = self.ca.getSubregister("ADC5_VREF3")
370
371
            if err:
                logging.error(self.logerr + "unable to read 'ADC5_VREF3'")
372
            if int(resp, 2): # check to see if Vref is 3 or 2.5 volts
    vrefmax = 3.0
373
374
375
            else:
376
               vrefmax = 2.5
            err, resp = self.ca.getSubregister("ADC5_VREF")
377
378
            if err:
                logging.error(self.logerr + "unable to read 'ADC5_VREF'")
379
            self.VREF = vrefmax \star int(resp, 2) / 1024.0
380
            err, multmask = self.ca.getSubregister("ADC5_MULT")
381
382
383
                logging.error(self.logerr + "unable to read 'ADC5_MULT'")
384
            if multmask[0] and multmask[1] and multmask[3] and multmask[5]:
385
                self.ADC5_mult = 2
386
            elif not (multmask[0] or multmask[1] or multmask[3] or multmask[5]):
387
                self.ADC5_mult = 4
388
389
               logging.error(self.logerr + "inconsistent mode settings on ADC5")
390
            return self.ca.submitMessages(control_messages, " initBoard: ")
```

8.7.3.12 initPots()

391

def nsCamera.boards.LLNL_v1.llnl_v1.initSensor (

self)

```
Definition at line 392 of file LLNL_v1.py.
```

```
def initPots(self):
393
394
             Configure default pot settings before image acquisition
395
396
             tuple (error string, response string) from final control message
397
398
399
             logging.info(self.loginfo + "initPots")
             if self.ca.sensorname == "icarus" or self.ca.sensorname == "icarus2":
                  err0, _ = self.ca.setPot("HST_A_PDELAY", 0, errflag=True)
err1, _ = self.ca.setPotV("HST_B_NDELAY", 3.3, errflag=True)
401
402
                  err2, _ = self.ca.setPotV("HST_RO_IBIAS", 2.5, tune=True, errflag=True)
403
                  err3, _ = self.ca.setPotV("HST_OSC_VREF_IN", 2.9, tune=True, errflag=True)
404
                  err4, _ = self.ca.setPot("HST_B_PDELAY", 0, errflag=True)
err5, _ = self.ca.setPotV("HST_OSC_CTL", 1.45, tune=True, errflag=True)
405
406
                  err6, _ = self.ca.setPotV("HST_A_NDELAY", 3.3, errflag=True)
407
                  err7, _ = self.ca.setPotV("VAB", 0.5, errflag=True)
408
                  err8, _ = self.ca.setPotV("HST_RO_NC_IBIAS", 2.5, errflag=True)
409
                  err9, _ = self.ca.setPotV("VRST", 0.3, tune=True, errflag=True)
410
411
                  err = err0 + err1 + err2 + err3 + err4 + err5 + err6 + err7 + err8 + err9
             else: # Daedalus
412
                  err0, _ = self.ca.setPotV("HST_OSC_CTL", 1.0, tune=True, errflag=True)
413
                  err1, _ = self.ca.setPotV("HST_RO_NC_IBIAS", 1.0, errflag=True)
err2, _ = self.ca.setPotV("HST_OSC_VREF_IN", 1.0, tune=True, errflag=True)
414
415
                  err3, _ = self.ca.setPotV("VAB", 0.5, errflag=True)
416
                  err = err0 + err1 + err2 + err3
417
             return err, ""
418
419
```

8.7.3.13 initSensor()

```
Register sensor, set default timing settings
Returns:
    tuple (error string, response string) from final control message
Definition at line 446 of file LLNL_v1.py.
446
        def initSensor(self):
447
448
            Register sensor, set default timing settings
449
450
            tuple (error string, response string) from final control message
451
452
            logging.info(self.loginfo + "initSensor")
453
454
            if self.ca.FPGANum[7] is not self.ca.sensor.fpganumID:
455
                logging.error(
456
                    self.logerr + "unable to confirm sensor compatibility with FPGA"
457
458
            self.registers.update(self.ca.sensor.sens_registers)
            self.subregisters.extend(self.ca.sensor.sens_subregisters)
459
460
            for s in self.ca.sensor.sens_subregisters:
461
                sr = SubRegister(
462
                    self,
                    name=s[0].upper(),
463
                    register=s[1].upper(),
464
                    start_bit=s[2],
465
466
                    width=s[3],
467
                    writable=s[4],
468
                setattr(self, s[0].upper(), sr)
469
470
                self.subreglist.append(s[0])
471
            self.ca.checkSensorVoltStat()
472
            control_messages = self.ca.sensorSpecific() + [
```

8.7.3.14 latchPots()

```
def nsCamera.boards.LLNL_v1.llnl_v1.latchPots (
                                                                 self )
Latch pot settings into sensor
Returns:
                 tuple (error string, response string) from final control message
Definition at line 420 of file LLNL v1.py.
                                 def latchPots(self):
 420
 421
 422
                                                  Latch pot settings into sensor
 423
424
                                                  tuple (error string, response string) from final control message
 425
426
                                                 logging.info(self.loginfo + "latchPots")
427
 428
                                                 control_messages = [
    ("POT_CTL", "00000003"),  # latches register settings for pot 1
    ("POT_CTL", "00000005"),
    ("POT_CTL", "00000007"),
429
 430
 431
 432
                                                                 ("POT_CTL", "0000000"),
("POT_CTL", "0000000B"),
 433
 434
                                                               ("POT_CTL", "0000000B"), ("POT_CTL", "0000000D"), ("POT_CTL", "0000000F"), ("POT_CTL", "00000011"), ("POT_CTL", "00000013"), ("POT_CTL", "00000015"), ("POT_CTL", "00000017"), ("POT_CTL", "000000017"), ("POT_CTL", "00000017"), ("POT_CTL", "DOT_CTL", "DOT
435
 436
 437
 438
 439
 440
                                                                   ("POT_CTL", "00000019"),
("POT_CTL", "0000001B"),
 441
 442
 443
 444
                                                  return self.ca.submitMessages(control_messages, " latchPots: ")
 445
```

8.7.3.15 readSRAM()

Definition at line 582 of file LLNL_v1.py.

8.7.3.16 reportEdgeDetects()

Unimplemented

```
def nsCamera.boards.LLNL_v1.llnl_v1.reportEdgeDetects ( self \ )
```

Definition at line 857 of file LLNL v1.py.

```
857 def reportEdgeDetects(self):
858 """
859 Unimplemented
860 """
861 logging.warning(
862 self.logwarn + "'reportEdgeDetects' is not implemented on the LLNLv1 "
863 "board"
864 )
```

8.7.3.17 reportStatus()

```
def nsCamera.boards.LLNL_v1.llnl_v1.reportStatus ( self \ )
```

Check contents of status register, print relevant messages

Definition at line 808 of file LLNL_v1.py.

```
def reportStatus(self):
809
810
            Check contents of status register, print relevant messages
811
812
            statusbits = self.checkStatus()
            statusbits2 = self.checkStatus2()
813
814
            logging.info(self.loginfo + "Status report:")
815
            if int(statusbits[0]):
816
                logging.info(self.loginfo + "Sensor read complete")
            if int(statusbits[1]):
817
818
                logging.info(self.loginfo + "Coarse trigger detected")
819
            if int(statusbits[2]):
                logging.info(self.loginfo + "Fine trigger detected")
820
821
            if int(statusbits[5]):
                logging.info(self.loginfo + "Sensor readout in progress")
822
823
            if int(statusbits[6]):
                logging.info(self.loginfo + "Sensor readout complete")
824
825
            if int(statusbits[7]):
```

```
826
                logging.info(self.loginfo + "SRAM readout started")
827
            if int(statusbits[8]):
828
                logging.info(self.loginfo + "SRAM readout complete")
829
            if int(statusbits[9]):
830
                logging.info(self.loginfo + "High-speed timing configured")
831
            if int(statusbits[10]):
                logging.info(self.loginfo + "All ADCs configured")
833
            if int(statusbits[11]):
834
                logging.info(self.loginfo + "All pots configured")
            if int(statusbits[13]):
                logging.info(self.loginfo + "Timer has reset")
837
            if int(statusbits[14]):
838
                logging.info(self.loginfo + "Camera is Armed")
839
            self.ca.sensor.reportStatusSensor(statusbits)
840
            temp = int(statusbits[27:15:-1], 2) / 16.0
841
            logging.info(
842
                self.loginfo + "Temperature reading: " + "{0:1.2f}".format(temp) + " C"
843
844
            press = int(statusbits[:27:-1], 2)
            logging.info(self.loginfo + "Pressure reading: " + "{0:1.2f}".format(press))
845
            if int(statusbits2[0]):
846
847
                logging.info(self.loginfo + "FPA_IF_TO")
848
            if int(statusbits2[1]):
849
                logging.info(self.loginfo + "INFO: [LLNL_v1] SRAM_RO_TO")
850
            if int(statusbits2[2]):
                logging.info(self.loginfo + "PixelRd Timeout Error")
851
            if int(statusbits2[3]):
852
                logging.info(self.loginfo + "UART_TX_TO_RST")
853
854
            if int(statusbits2[4]):
                logging.info(self.loginfo + "UART_RX_TO_RST")
855
856
```

8.7.3.18 resetTimer()

Definition at line 644 of file LLNL v1.py.

```
644
         def resetTimer(self):
645
646
               Reset on-board timer
647
648
               Returns:
               tuple (error string, response string from register set)
649
650
               logging.info(self.loginfo + "resetTimer")
651
               control_messages = [("RESET_TIMER", "1"), ("RESET_TIMER", "0")]
return self.ca.submitMessages(control_messages, " resetTimer: ")
652
653
654
```

8.7.3.19 setLED()

```
def nsCamera.boards.LLNL_v1.llnl_v1.setLED (
               self,
               LED,
               status )
Illuminate on-board LED
Aras:
    LED: LED number (1-8)
    status: 0 is off, 1 is on
Returns:
    tuple: (error string, response string from subregister set)
Definition at line 669 of file LLNL_v1.py.
669
       def setLED(self, LED, status):
670
671
           Illuminate on-board LED
672
673
674
               LED: LED number (1-8)
675
               status: 0 is off, 1 is on
676
677
           tuple: (error string, response string from subregister set)
678
679
680
           key = "LED" + str(LED)
681
           return self.ca.setSubregister(key, str(status))
682
```

8.7.3.20 setPowerSave()

```
def nsCamera.boards.LLNL_v1.llnl_v1.setPowerSave (
               self,
               status )
Select powersave option
Args:
    status: setting for powersave option (1 is enabled)
Returns:
    tuple (error string, response string from subregister set)
Definition at line 683 of file LLNL_v1.py.
683
       def setPowerSave(self, status):
684
685
           Select powersave option
686
687
               status: setting for powersave option (1 is enabled)
688
689
690
           Returns:
           tuple (error string, response string from subregister set)
691
```

return self.ca.setSubregister("POWERSAVE", str(status))

if status:

status = 1

692 693

694

695

696

8.7.3.21 setPPER()

516

517 518 519

520 521

522

```
def nsCamera.boards.LLNL_v1.llnl_v1.setPPER (
               self,
               time )
Set polling period for ADCs.
    time: milliseconds, between 1 and 255, defaults to 50
Returns:
    tuple (error string, response string from subregister set OR invalid time
      setting string)
Definition at line 697 of file LLNL_v1.py.
       def setPPER(self, time):
698
699
           Set polling period for ADCs.
700
701
               time: milliseconds, between 1 and 255, defaults to 50
702
703
704
               tuple (error string, response string from subregister set OR invalid time
705
                 setting string)
706
707
           if not time:
708
               time = 50
709
           if not isinstance(time, int) or time < 1 or time > 255:
710
               err = (
711
                   self.logerr + "invalid poll period submitted. Setting remains "
                   "unchanged. "
712
713
               logging.error(err)
714
715
               return err, str(time)
716
           else:
717
               binset = bin(time)[2:].zfill(8)
               return self.ca.setSubregister("PPER", binset)
718
719
8.7.3.22 softReboot()
def nsCamera.boards.LLNL_v1.llnl_v1.softReboot (
               self )
Perform software reboot of board. WARNING: board reboot will likely prevent
  correct response and therefore will generate an error message
Returns:
    tuple (error string, response string) from final control message
Definition at line 511 of file LLNL v1.py.
511
       def softReboot(self):
512
513
           Perform software reboot of board. WARNING: board reboot will likely prevent
514
             correct response and therefore will generate an error message
515
```

tuple (error string, response string) from final control message

return self.ca.submitMessages(control_messages, " disarm: ")

logging.info(self.loginfo + "reboot")
control_messages = [("RESET", "1")]

8.7.3.23 startCapture()

```
def nsCamera.boards.LLNL_v1.llnl_v1.startCapture (
                self,
                mode = "Hardware" )
Reads ADC data into SRAM
Returns:
    tuple (error string, response string) from final control message
Definition at line 541 of file LLNL v1.py.
        def startCapture(self, mode="Hardware"):
541
542
543
            Reads ADC data into SRAM
544
545
            Returns:
            tuple (error string, response string) from final control message _{\tt mum}
546
547
            logging.info(self.loginfo + "startCapture")
548
549
            if self.ca.sensmanual:
                timingReg = "MANSHUT_MODE"
550
5.51
            else:
                timingReg = "HST_MODE"
552
553
            if mode.upper() == "SOFTWARE":
554
555
                trigmess = [
                    ("HW_TRIG_EN", "0"),
556
                    ("DUAL_EDGE_TRIG_EN", "0"), ("SW_TRIG_EN", "1"),
557
558
                     ("SW_TRIG_START", "1"),
559
560
                ]
561
            elif mode.upper() == "DUAL":
562
                trigmess = [
                     ("SW_TRIG_EN", "0"),
("HW_TRIG_EN", "1"),
563
564
                     ("DUAL_EDGE_TRIG_EN", "1"),
565
566
                ]
            else: # HARDWARE
567
568
                trigmess = [
                    ("DUAL_EDGE_TRIG_EN", "0"),
569
                     ("SW_TRIG_EN", "0"),
("HW_TRIG_EN", "1"),
570
571
572
573
574
            control_messages = [
                 ("ADC_CTL", "0000001F"), # configure all ADCs
575
576
                 (timingReg, "1"),
577
578
579
            control_messages.extend(trigmess)
580
            return self.ca.submitMessages(control_messages, " startCapture: ")
```

8.7.3.24 waitForSRAM()

```
Wait until subreg 'SRAM_READY' flag is true or timeout is exceeded;
  timeout = None or zero means wait indefinitely
Args:
    timeout - time in seconds before readoff proceeds automatically without
      waiting for SRAM_READY flag
Returns:
    error string
Definition at line 593 of file LLNL v1.py.
593
        def waitForSRAM(self, timeout):
594
            Wait until subreg 'SRAM_READY' flag is true or timeout is exceeded;
595
596
             timeout = None or zero means wait indefinitely
597
598
               timeout - time in seconds before readoff proceeds automatically without
599
600
                 waiting for SRAM_READY flag
601
602
            Returns:
            error string
603
604
605
           logging.info(self.loginfo + "waitForSRAM")
606
            waiting = True
607
            starttime = time.time()
608
            err = ""
609
            while waiting:
610
               err, status = self.ca.getSubregister("SRAM_READY")
611
612
                    logging.error(
613
                       self.logerr + "error in register read: " + err + " (waitForSRAM)"
614
615
                if int(status):
616
                    waiting = False
617
                    logging.info(self.loginfo + "SRAM ready")
618
               if self.ca.abort:
619
                    waiting = False
620
                    logging.info(self.loginfo + "readoff aborted by user")
621
                    self.ca.abort = False
622
                if timeout and time.time() - starttime > timeout:
623
                    err += self.logerr + "SRAM timeout; proceeding with download attempt"
                    logging.error(err)
625
           return err
```

8.7.4 Member Data Documentation

8.7.4.1 ADC5 bipolar

nsCamera.boards.LLNL_v1.llnl_v1.ADC5_bipolar

Definition at line 244 of file LLNL_v1.py.

8.7.4.2 ADC5_mult

nsCamera.boards.LLNL_v1.llnl_v1.ADC5_mult

Definition at line 242 of file LLNL_v1.py.

8.7.4.3 ca

nsCamera.boards.LLNL_v1.llnl_v1.ca

Definition at line 234 of file LLNL_v1.py.

8.7.4.4 dummySensorVals

```
list nsCamera.boards.LLNL_v1.llnl_v1.dummySensorVals [static]
```

Definition at line 194 of file LLNL_v1.py.

8.7.4.5 logcrit

nsCamera.boards.LLNL_v1.llnl_v1.logcrit

Definition at line 235 of file LLNL_v1.py.

8.7.4.6 logdebug

nsCamera.boards.LLNL_v1.llnl_v1.logdebug

Definition at line 239 of file LLNL_v1.py.

8.7.4.7 logerr

nsCamera.boards.LLNL_v1.llnl_v1.logerr

Definition at line 236 of file LLNL_v1.py.

8.7.4.8 loginfo

nsCamera.boards.LLNL_v1.llnl_v1.loginfo

Definition at line 238 of file LLNL_v1.py.

8.7.4.9 logwarn

```
nsCamera.boards.LLNL_v1.llnl_v1.logwarn
```

Definition at line 237 of file LLNL_v1.py.

8.7.4.10 monitor_controls

```
nsCamera.boards.LLNL_v1.llnl_v1.monitor_controls
```

Definition at line 287 of file LLNL_v1.py.

8.7.4.11 registers

```
nsCamera.boards.LLNL_v1.llnl_v1.registers [static]
```

Definition at line 38 of file LLNL_v1.py.

8.7.4.12 rs422_baud

```
nsCamera.boards.LLNL_v1.llnl_v1.rs422_baud
```

Definition at line 246 of file LLNL_v1.py.

8.7.4.13 rs422 cmd wait

```
nsCamera.boards.LLNL_v1.llnl_v1.rs422_cmd_wait
```

Definition at line 247 of file LLNL_v1.py.

8.7.4.14 subreg_aliases

nsCamera.boards.LLNL_v1.llnl_v1.subreg_aliases

Definition at line 262 of file LLNL_v1.py.

8.7.4.15 subregisters

list nsCamera.boards.LLNL_v1.llnl_v1.subregisters [static]

Definition at line 99 of file LLNL_v1.py.

8.7.4.16 subreglist

nsCamera.boards.LLNL_v1.llnl_v1.subreglist

Definition at line 329 of file LLNL v1.py.

8.7.4.17 VREF

nsCamera.boards.LLNL_v1.llnl_v1.VREF

Definition at line 241 of file LLNL_v1.py.

The documentation for this class was generated from the following file:

nsCamera/boards/LLNL_v1.py

8.8 nsCamera.boards.LLNL_v4.llnl_v4 Class Reference

Public Member Functions

- def __init__ (self, camassem)
- def initBoard (self)
- def initPots (self)
- def latchPots (self)
- def initSensor (self)
- · def configADCs (self)
- def softReboot (self)
- def disarm (self)
- def startCapture (self, mode="Hardware")
- def readSRAM (self)
- def waitForSRAM (self, timeout)
- def getTimer (self)
- def resetTimer (self)
- def enableLED (self, status)
- def setLED (self, LED, status)
- def setPowerSave (self, status)
- def setPPER (self, time)
- def getTemp (self, scale)
- def getPressure (self, offset, sensitivity, units)
- def clearStatus (self)
- · def checkStatus (self)
- def checkStatus2 (self)
- def reportStatus (self)
- def reportEdgeDetects (self)
- def dumpStatus (self)

Public Attributes

- ca
- logcrit
- logerr
- logwarn
- loginfo
- logdebug
- VREF
- ADC5 mult
- ADC5_bipolar
- rs422_baud
- · rs422_cmd_wait
- defoff
- defsens
- subreg_aliases
- · monitor_controls
- · subreglist

Static Public Attributes

- · registers
- list subregisters
- · list dummySensorVals

8.8.1 Detailed Description

Livermore LLNL v4.0 board

Compatible communication protocols: RS422, GigE Compatible sensors: icarus, icarus2, daedalus

Definition at line 29 of file LLNL_v4.py.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 __init__()

```
def nsCamera.boards.LLNL_v4.llnl_v4.__init__ (
                  self,
                  camassem )
Definition at line 218 of file LLNL_v4.py.
         def __init__(self, camassem):
218
              self.ca = camassem
220
              self.logcrit = self.ca.logcritbase + "[LLNL_v4] "
              self.logerr = self.ca.logerrbase + "[LLNL_v4]
221
              self.logwarn = self.ca.logwarnbase + "[LLNL_v4]
222
              self.loginfo = self.ca.loginfobase + "[LLNL_v4] "
223
224
              self.logdebug = self.ca.logdebugbase + "[LLNL_v4]
225
              logging.info(self.loginfo + "Iinitializing board object")
              self.VREF = 3.3 # must be supplied externally for ADC128S102
226
227
              self.ADC5_mult = 1
228
229
              # ADC128S102; False => monitor range runs 0 to monmax, True => +/- monmax
230
              self.ADC5 bipolar = False
2.31
              self.rs422 baud = 921600
232
              self.rs422\_cmd\_wait = 0.3
233
              fpgaNum_pkt = Packet(cmd="1", addr=self.registers["FPGA_NUM"])
fpgaRev_pkt = Packet(cmd="1", addr=self.registers["FPGA_REV"])
2.34
235
236
              _, _ = self.ca.sendCMD(fpgaNum_pkt) # dummy duplicate call
237
238
              err, rval = self.ca.sendCMD(fpgaNum_pkt)
239
              self.ca.FPGANum = rval[8:16]
240
              err, rval = self.ca.sendCMD(fpgaRev_pkt)
2.41
2.42
              self.ca.FPGAVersion = rval[8:16]
243
              self.defoff = 34.5  # default pressure sensor offset
self.defsens = 92.5  # default pressure sensor sensitivity
2.44
245
246
247
              # map channels to signal names for abstraction at the camera assembler level;
              # each requires a corresponding entry in 'subregisters'
if self.ca.sensorname == "icarus" or self.ca.sensorname == "icarus2":
248
249
250
                  self.subreg_aliases = OrderedDict(
251
                            "HST_A_PDELAY": "DACA"
"HST_A_NDELAY": "DACB"
252
253
                            "HST_B_PDELAY": "DACC"
254
                            "HST_B_NDELAY": "DACD"
"HST_RO_IBIAS": "DACE"
255
256
257
                            "HST_RO_NC_IBIAS": "DACE",
258
                            "HST_OSC_CTL": "DACF",
                            "VAB": "DACG",
259
                            "VRST": "DACH",
260
261
                            "MON_PRES_MINUS": "MON_CH1",
262
                            "MON_PRES_PLUS": "MON_CH2",
                            "MON_TEMP": "MON_CH3",
"MON_COL_TOP_IBIAS_IN": "MON_CH4",
263
264
265
                            "MON_HST_OSC_R_BIAS": "MON_CH5",
                            "MON_VAB": "MON_CH6",
266
                            "MON_HST_RO_IBIAS": "MON_CH7"
267
                            "MON_HST_RO_NC_IBIAS": "MON_CH7",
268
269
                            "MON_VRST": "MON_CH8",
                            "MON_COL_BOT_IBIAS_IN": "MON_CH9",
270
                            "MON_HST_A_PDELAY": "MON_CH10",
"MON_HST_B_NDELAY": "MON_CH11",
271
272
273
                            "DOSIMETER": "MON_CH12",
                            "MON_HST_OSC_VREF_IN": "MON_CH13",
274
275
                            "MON_HST_B_PDELAY": "MON_CH14",
276
                            "MON_HST_OSC_CTL": "MON_CH15",
                            "MON_HST_A_NDELAY": "MON_CH16",
277
                            "MON_CHA": "MON_CH10",
278
                            "MON_CHB": "MON_CH16",
279
280
                            "MON_CHC": "MON_CH14",
                            "MON_CHD": "MON_CH11",
281
                            "MON_CHE": "MON_CH7",
282
                            "MON_CHF": "MON_CH15",
283
                            "MON_CHG": "MON_CH6",
284
                            "MON_CHH": "MON_CH8",
285
286
287
```

```
288
                     # Read-only; identifies controls corresponding to monitors
289
                     self.monitor_controls = OrderedDict(
290
291
                               "MON_CH10": "DACA",
                               "MON_CH16": "DACB",
"MON_CH14": "DACC",
292
293
                               "MON_CH11": "DACD",
"MON_CH7": "DACE",
294
295
296
                               "MON_CH15": "DACF",
297
                               "MON_CH6": "DACG",
298
                               "MON_CH8": "DACH",
299
300
                    )
               else: # Daedalus
301
302
                    self.subreg_aliases = OrderedDict(
303
304
                               "HST_OSC_VREF_IN": "DACC",
                               "HST_OSC_CTL": "DACE",
"COL_TST_IN": "DACF",
305
306
                               "VAB": "DACG",
307
                               "MON_PRES_MINUS": "MON_CH1",
"MON_PRES_PLUS": "MON_CH2",
308
309
                               "MON_TEMP": "MON_CH3",
"MON_VAB": "MON_CH6",
310
311
                               "MON_HST_OSC_CTL": "MON_CH7",
312
                               MON_TSENSE_OUT": "MON_CH10",
"MON_BGREF": "MON_CH11",
"DOSIMETER": "MON_CH12",
313
314
315
                               "MON_HST_RO_NC_IBIAS": "MON_CH13",
"MON_HST_OSC_VREF_IN": "MON_CH14",
"MON_COL_TST_IN": "MON_CH15",
"MON_HST_OSC_PBIAS_PAD": "MON_CH16",
316
317
318
319
                               "MON_CHC": "MON_CH14",
320
                               "MON_CHE": "MON_CH7",
"MON_CHF": "MON_CH5",
"MON_CHG": "MON_CH6",
321
322
323
324
                          }
325
                    # Read-only; identifies controls corresponding to monitors
self.monitor_controls = OrderedDict(
326
327
328
                               "MON_CH14": "DACC",
"MON_CH7": "DACE",
"MON_CH15": "DACF",
329
330
331
                               "MON_CH6": "DACG",
332
333
                          }
334
335
               self.subreglist = []
336
               for s in self.subregisters:
337
                    self.subreglist.append(s[0].upper())
338
                    sr = SubRegister(
339
                          self,
340
                          name=s[0].upper(),
341
                          register=s[1].upper(),
342
                          start_bit=s[2],
343
                          width=s[3],
344
                          writable=s[4],
345
346
                    setattr(self, s[0].upper(), sr)
347
               # set voltage ranges for all DACs - WARNING: actual output voltage limited to
                    external supply (3.3 V)
349
350
               # setpot('potx', n) will generate 3.3 V for all n > .66
               for n in range(0, 8):
351
                   potname = "DAC" + string.ascii_uppercase[n]
352
                    potobj = getattr(self, potname)
353
                   potobj.minV = 0
354
355
                   potobj.maxV = 5 #
                    potobj.resolution = (
356
357
                          1.0 * potobj.maxV - potobj.minV
358
                    ) / potobj.max_value # 76 uV / LSB
359
```

8.8.3 Member Function Documentation

8.8.3.1 checkStatus()

```
def nsCamera.boards.LLNL_v4.llnl_v4.checkStatus (
               self )
Check status register, convert to reverse-order bit stream (i.e., bit 0 is
  statusbits[0])
Returns:
    bit string (no '0b') in reversed order
Definition at line 741 of file LLNL_v4.py.
741
        def checkStatus(self):
742
743
            Check status register, convert to reverse-order bit stream (i.e., bit {\tt 0} is
744
             statusbits[0])
745
746
           Returns:
            bit string (no '0b') in reversed order """
747
748
            err, rval = self.ca.getRegister("STAT_REG")
749
750
            rvalbits = bin(int(rval, 16))[2:].zfill(32)
751
            statusbits = rvalbits[::-1]
752
            return statusbits
753
```

8.8.3.2 checkStatus2()

Definition at line 754 of file LLNL_v4.py.

```
754
        def checkStatus2(self):
755
756
            Check second status register, convert to reverse-order bit stream (i.e., bit {\tt 0}
757
              is statusbits[0])
758
759
            Returns: bit string (no '0b') in reversed order
760
761
            err, rval = self.ca.getRegister("STAT_REG2")
762
            rvalbits = bin(int(rval, 16))[2:].zfill(6)
763
            statusbits = rvalbits[::-1]
764
            return statusbits
765
```

8.8.3.3 clearStatus()

def nsCamera.boards.LLNL_v4.llnl_v4.clearStatus (

```
self )
Check status registers to clear them
Returns:
    error string
Definition at line 727 of file LLNL v4.py.
727
         def clearStatus(self):
728
729
             Check status registers to clear them
730
731
             Returns:
             error string
732
733
             err1, rval = self.ca.getRegister("STAT_REG_SRC")
734
             err2, rval = self.ca.getRegister("STAT_REG2_SRC")
735
             err = err1 + err2
736
737
             if err:
                 logging.error(self.logerr + "clearStatus failed")
738
739
             return err
740
8.8.3.4 configADCs()
def nsCamera.boards.LLNL_v4.llnl_v4.configADCs (
                 self )
Sets default ADC configuration (does not latch settings)
     tuple (error string, response string) from final control message
Definition at line 436 of file LLNL v4.py.
436
         def configADCs(self):
437
438
             Sets default ADC configuration (does not latch settings)
439
440
             tuple (error string, response string) from final control message
441
442
443
             logging.info(self.loginfo + "configADCs")
444
445
             control_messages = [
446
                  # just in case ADC_RESET was set on any of the ADCs (pull all ADCs out of
447
                     reset)
                  ("ADC_RESET", "00000000"),
448
                  ("ADC1_CONFIG_DATA", "FFFFFFFF"),
("ADC2_CONFIG_DATA", "FFFFFFFF"),
449
450
                  ("ADC3_CONFIG_DATA", "FFFFFFFF"),
("ADC4_CONFIG_DATA", "FFFFFFFF"),
451
452
453
                  ("ADC_CTL", "FFFFFFFF"),
                  ("ADC1_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
("ADC2_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
454
455
                  ("ADC3_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
("ADC4_CONFIG_DATA", "81A801FF"), # ext Vref 1.25V
456
457
458
459
             return self.ca.submitMessages(control_messages, " configADCs: ")
460
```

8.8.3.5 disarm()

```
def nsCamera.boards.LLNL_v4.llnl_v4.disarm (
               self )
Takes camera out of trigger wait state. Has no effect if camera is not in wait
  state.
Returns:
    tuple (error string, response string) from final control message
Definition at line 473 of file LLNL v4.py.
        def disarm(self):
474
475
            Takes camera out of trigger wait state. Has no effect if camera is not in wait
476
477
478
           tuple (error string, response string) from final control message _{\tt nnn}
479
480
481
            logging.info(self.loginfo + "disarm")
            self.ca.clearStatus()
            self.ca.armed = False
483
           control_messages = [
484
               ("HW_TRIG_EN", "0"),
485
486
                ("DUAL_EDGE_TRIG_EN", "0"),
487
                ("SW_TRIG_EN", "0"),
488
489
            self.ca.comms.skipError = False
490
            return self.ca.submitMessages(control_messages, " disarm: ")
8.8.3.6 dumpStatus()
def nsCamera.boards.LLNL_v4.llnl_v4.dumpStatus (
               self )
```

Definition at line 857 of file LLNL_v4.py.

```
857
        def dumpStatus(self):
             Create dictionary of status values, DAC settings, monitor values, and register
859
860
               values
861
862
             Returns:
             dictionary of system diagnostic values
863
864
             statusbits = self.checkStatus()
statusbits2 = self.checkStatus2()
865
866
867
             temp = int(statusbits[23:16:-1], 2) * 3.3 * 1000 / 4096
868
             press = int(statusbits[:23:-1], 2) * 3.3 * 1000 / 4096
869
870
871
             statDict = OrderedDict(
872
873
                      "Temperature sensor reading": "\{0:1.2f\}".format(temp) + " C",
```

```
874
                     "Pressure reading": str(round(self.ca.getPressure(), 3)) + " Torr",
875
                     "Pressure sensor reading": "{0:1.2f}".format(press) + " mV",
876
                     "Sensor read complete": str(statusbits[0]),
877
                    "Coarse trigger detected": str(statusbits[1]),
878
                     "Fine trigger detected": str(statusbits[2]),
879
                    "W3_Top_L_Edge1 detected": str(statusbits[3]),
880
                     "W3_Top_R_Edge1 detected": str(statusbits[4]),
881
                     "Sensor readout in progress": str(statusbits[5]),
                     "Sensor readout complete": str(statusbits[6]),
882
                    "SRAM readout started": str(statusbits[7]),
883
                    "SRAM readout complete": str(statusbits[8]),
884
                    "High-speed timing configured": str(statusbits[9]),
886
                     "All ADCs configured": str(statusbits[10]),
                    "All DACs configured": str(statusbits[11]),
887
888
                    "HST_All_W_En detected": str(statusbits[12]),
889
                    "Timer has reset": str(statusbits[13]),
890
                     "Camera is Armed": str(statusbits[14]),
                    "FPA_IF_TO": str(statusbits2[0]),
891
                    "SRAM_RO_TO": str(statusbits2[1]),
892
893
                    "PixelRd Timeout Error": str(statusbits2[2]),
                     "UART_TX_TO_RST": str(statusbits2[3]),
894
895
                     "UART_RX_TO_RST": str(statusbits2[4]),
                     "PDBIAS Unready": str(statusbits2[5]),
896
897
                }
898
            )
899
900
            DACDict = OrderedDict()
            MonDict = OrderedDict()
901
902
            for entry in self.subreg_aliases:
                if self.subreg_aliases[entry][0] == "D":
903
                    val = str(round(self.ca.getPotV(entry), 3)) + " V"
DACDict["DAC_" + entry] = val
904
905
906
                    val = str(round(self.ca.getMonV(entry), 3)) + " V"
907
908
                    MonDict[entry] = val
909
910
            regDict = OrderedDict()
911
            for key in self.registers.keys():
912
                err, rval = self.ca.getRegister(key)
913
                regDict[key] = rval
914
915
            dumpDict = OrderedDict()
916
            for x in [statDict, MonDict, DACDict, regDict]:
917
                dumpDict.update(x)
918
            return dumpDict
919
920
921 """
922 Copyright (c) 2022, Lawrence Livermore National Security, LLC. All rights reserved.
923 LLNL-CODE-838080
924
925 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
926 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy
927 (DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
928 See license for disclaimers, notice of U.S. Government Rights and license terms and
929 conditions.
930 """
```

8.8.3.7 enableLED()

```
Definition at line 606 of file LLNL_v4.py.
```

```
def enableLED(self, status):

"""

Dummy function; feature is not implemented on Icarus

Dummy function; feature is not implemented on Icarus

Returns:

tuple: dummy of (error string, response string from subregister set)

"""

return "", "0"
```

```
8.8.3.8 getPressure()
def nsCamera.boards.LLNL_v4.llnl_v4.getPressure (
                self.
                offset,
                sensitivity,
                units )
Read pressure sensor. Uses default offset and sensitivity defined in init
  function unless alternatives are specified. NOTE: to reset defaults, reassign
  board.defoff and board.defsens explicitly
Args:
    offset: non-default offset in mv/V
    sensitivity: non-default sensitivity in mV/V/span
    units: units to report pressure (defaults to Torr, options are psi, bar,
      inHg, atm)
Returns:
    Pressure as float in chosen units, defaults to torr
Definition at line 686 of file LLNL v4.py.
686
        def getPressure(self, offset, sensitivity, units):
687
688
            Read pressure sensor. Uses default offset and sensitivity defined in init
689
              function unless alternatives are specified. NOTE: to reset defaults, reassign
690
              board.defoff and board.defsens explicitly
691
692
693
                offset: non-default offset in mv/V
694
                sensitivity: non-default sensitivity in mV/V/span
695
                units: units to report pressure (defaults to Torr, options are psi, bar,
                 inHg, atm)
697
698
699
                Pressure as float in chosen units, defaults to torr
700
701
702
            if offset is None:
703
                offset = self.defoff
704
            if sensitivity is None:
705
                sensitivity = self.defsens
706
            if units is None:
707
                units = "torr"
708
709
            pplus = self.ca.getMonV("MON_PRES_PLUS")
710
            pminus = self.ca.getMonV("MON_PRES_MINUS")
            delta = 1000 * (pplus - pminus)
ratio = sensitivity / 30 # nominal = 21 / 30
711
712
            psi = (delta - offset) / ratio
713
            if units.lower() == "psi":
714
715
            press = psi
elif units.lower() == "bar":
716
717
                press = psi / 14.504
```

```
718
           elif units.lower() == "atm":
719
               press = psi / 14.695
720
            elif units.lower() == "inHg":
721
               press = psi * 2.036
722
            else:
723
               press = 51.715 * psi # default to Torr
724
725
           return press
726
```

8.8.3.9 getTemp()

```
{\tt def nsCamera.boards.LLNL\_v4.llnl\_v4.getTemp} \ \ (
                self,
                scale )
Read temperature sensor
Args:
    scale: temperature scale to report (defaults to C, options are {\tt F} and {\tt K})
Returns:
    temperature as float on given scale
Definition at line 661 of file LLNL v4.py.
        def getTemp(self, scale):
662
663
            Read temperature sensor
664
            Args:
665
                scale: temperature scale to report (defaults to C, options are F and K)
666
667
            Returns:
            temperature as float on given scale
668
669
            err, rval = self.ca.getMonV("MON_TEMP", errflag=True)
670
671
            if err:
672
                logging.error(
                    self.logerr + "unable to retrieve temperature information (" 'getTemp), returning "0" '
673
674
675
                return 0.0
676
            ctemp = rval * 1000 - 273.15
677
            if scale == "K":
678
            temp = ctemp + 273.15
elif scale == "F":
679
680
681
                temp = 1.8 * ctemp + 32
682
            else:
683
                temp = ctemp
684
            return temp
685
```

8.8.3.10 getTimer()

```
Definition at line 579 of file LLNL_v4.py.
579
        def getTimer(self):
580
581
            Read value of on-board timer
582
583
           timer value as integer
584
585
586
            err, rval = self.ca.getRegister("TIMER_VALUE")
587
           if err:
588
               logging.error(
589
                   self.logerr + "unable to retrieve timer information (getTimer), "
590
                    returning "0" '
591
592
               return 0
           return int (rval, 16)
8.8.3.11 initBoard()
def nsCamera.boards.LLNL_v4.llnl_v4.initBoard (
               self )
Register and reset board, set up firmware for sensor
Returns:
    tuple (error string, response string) from final control message
Definition at line 360 of file LLNL_v4.py.
360
        def initBoard(self):
361
362
            Register and reset board, set up firmware for sensor
363
364
           tuple (error string, response string) from final control message _{\tt mum}
365
366
367
           logging.info(self.loginfo + "initBoard")
368
            control_messages = []
369
            self.clearStatus()
370
            self.configADCs()
371
            return self.ca.submitMessages(control_messages, " initBoard: ")
372
8.8.3.12 initPots()
def nsCamera.boards.LLNL_v4.llnl_v4.initPots (
               self )
Dummy function; initial DAC values are set by firmware at startup
Returns:
    tuple (empty string, empty string)
Definition at line 373 of file LLNL v4.py.
        def initPots(self):
373
```

Dummy function; initial DAC values are set by firmware at startup

tuple (empty string, empty string)

Generated by Doxygen

return "", ""

374

375 376 377

378 379

380 381

8.8.3.13 initSensor()

```
def nsCamera.boards.LLNL_v4.llnl_v4.initSensor (
                self )
Register sensor, set default timing settings
Returns:
    tuple (error string, response string) from final control message
Definition at line 402 of file LLNL_v4.py.
        def initSensor(self):
402
403
            Register sensor, set default timing settings
404
405
406
            Returns:
            tuple (error string, response string) from final control message
407
408
409
            logging.info(self.loginfo + "initSensor")
410
            if self.ca.FPGANum[7] is not self.ca.sensor.fpganumID:
411
                logging.warning(
                    self.logwarn + "unable to confirm sensor compatibility with FPGA"
412
413
414
            self.registers.update(self.ca.sensor.sens_registers)
415
            self.subregisters.extend(self.ca.sensor.sens_subregisters)
416
            for s in self.ca.sensor.sens_subregisters:
417
                sr = SubRegister(
418
                    self,
419
                    name=s[0].upper(),
420
                     register=s[1].upper(),
421
                    start_bit=s[2],
422
                    width=s[3],
423
                    writable=s[4],
424
425
                setattr(self, s[0].upper(), sr)
426
                self.subreglist.append(s[0])
            # self.ca.checkSensorVoltStat() # SENSOR_VOLT_STAT and SENSOR_VOLT_CTL are
427
428
                deactivated for v4 icarus and daedalus firmware for now.
429
            control_messages = self.ca.sensorSpecific() + [
                # ring w/caps=01, relax=00, ring w/o caps = 02
("FPA_OSCILLATOR_SEL_ADDR", "00000000"),
430
431
                 ("FPA_DIVCLK_EN_ADDR", "00000001"),
432
433
434
            return self.ca.submitMessages(control_messages, " initSensor: ")
435
```

8.8.3.14 latchPots()

383 384

385 386

Latch DAC settings into sensor

Returns:

```
387
                      tuple (error string, response string) from final control message
388
389
                 logging.info(self.loginfo + "latchPots")
390
                 control_messages = [
391
                       ("DAC_CTL", "00000001"), # latches register settings for DACA
                      ("DAC_CTL", "00000001"),
("DAC_CTL", "00000003"),
("DAC_CTL", "00000005"),
("DAC_CTL", "00000000"),
("DAC_CTL", "00000009"),
("DAC_CTL", "00000000"),
("DAC_CTL", "00000000"),
392
393
394
395
396
397
398
                      ("DAC_CTL", "0000000F"),
399
400
                return self.ca.submitMessages(control_messages, " latchPots: ")
401
```

8.8.3.15 readSRAM()

Definition at line 533 of file LLNL v4.py.

```
def readSRAM(self):
533
534
             Start readoff of SRAM
535
536
537
             Returns:
             tuple (error string, response string from register set) \ensuremath{\text{\tiny """}}
538
539
             logging.info(self.loginfo + "readSRAM")
540
541
             control_messages = [("READ_SRAM", "1")]
             return self.ca.submitMessages(control_messages, " readSRAM: ")
542
543
```

8.8.3.16 reportEdgeDetects()

```
def nsCamera.boards.LLNL_v4.llnl_v4.reportEdgeDetects ( self\ ) Report edge detects
```

Definition at line 822 of file LLNL v4.py.

```
822 def reportEdgeDetects(self):
823 """
824 Report edge detects
825 """
826 err, rval = self.ca.getRegister("STAT_EDGE_DETECTS")
827 # shift to left to fake missing edge detect
828 edgebits = bin(int(rval, 16) « 1)[2:].zfill(32)
829 # reverse to get order matching assignment
830 bitsrev = edgebits[::-1]
```

```
831
            detdict = {}
832
            bitidx = 0
833
            for frame in range(4):
834
                for vert in ("TOP", "BOT"):
                    for edge in range(1, 3):
835
836
                        for hor in ("L", "R"):
837
                            detname = (
838
                                "W"
839
                                + str(frame)
                                + "_"
840
841
                                + vert
842
843
                                + hor
                                 + "_EDGE"
844
845
                                + str(edge)
846
847
                            detdict[detname] = bitsrev[bitidx]
848
                            bitidx += 1
849
            # remove faked detect
            del detdict["W0_TOP_L_EDGE1"]
850
            print("Edge detect report:")
851
            print("----")
852
            for key, val in detdict.items():
853
               print(key + ": " + val)
854
            print("----")
855
856
```

8.8.3.17 reportStatus()

Check contents of status register, print relevant messages

Definition at line 766 of file LLNL_v4.py.

```
766
        def reportStatus(self):
767
768
            Check contents of status register, print relevant messages
769
770
            statusbits = self.checkStatus()
771
            statusbits2 = self.checkStatus2()
772
            print("Status report:")
           print("----")
773
774
           if int(statusbits[0]):
775
                print("Sensor read complete")
776
            if int(statusbits[1]):
777
                print("Coarse trigger detected")
778
            if int(statusbits[2]):
779
                print("Fine trigger detected")
780
            if int(statusbits[3]):
781
                print("W3_Top_L_Edge1 detected")
782
            if int(statusbits[4]):
783
                print("W3_Top_R_Edge1 detected")
            if int(statusbits[5]):
785
                print("Sensor readout in progress")
786
            if int(statusbits[6]):
787
                print("Sensor readout complete")
788
            if int(statusbits[7]):
789
                print("SRAM readout started")
790
            if int(statusbits[8]):
791
                print("SRAM readout complete")
792
            if int(statusbits[9]):
793
                print("High-speed timing configured")
            if int(statusbits[10]):
794
795
                print("All ADCs configured")
796
            if int(statusbits[11]):
797
                print("All DACs configured")
798
            if int(statusbits[12]):
799
                print("HST_All_W_En detected")
```

```
800
             if int(statusbits[13]):
801
                  print("Timer has reset")
802
              if int(statusbits[14]):
803
                  print("Camera is Armed")
             temp = int(statusbits[23:16:-1], 2) * 3.3 * 1000 / 4096 print("Temperature reading: " + "{0:1.2f}".format(temp) + " C")
804
805
             press = int(statusbits[:23:-1], 2) * 3.3 * 1000 / 4096
print("Pressure sensor reading: " + "{0:1.2f}".format(press) + " mV")
806
807
808
             if int(statusbits2[0]):
                  print("FPA_IF_TO")
810
            if int(statusbits2[1]):
                  print("SRAM_RO_TO")
811
812
             if int(statusbits2[2]):
                  print("PixelRd Timeout Error")
813
814
             if int(statusbits2[3]):
                  print("UART_TX_TO_RST")
816
             if int(statusbits2[4]):
817
                  print("UART_RX_TO_RST")
             if int(statusbits2[5]):
818
819
                  print("PDBIAS Unready")
             print("----")
820
821
```

8.8.3.18 resetTimer()

```
def nsCamera.boards.LLNL_v4.llnl_v4.resetTimer (
               self )
Reset on-board timer
Returns:
   tuple (error string, response string from register set)
Definition at line 595 of file LLNL_v4.py.
       def resetTimer(self):
596
597
           Reset on-board timer
598
599
           tuple (error string, response string from register set)
600
601
602
           logging.info(self.loginfo + "resetTimer")
           control_messages = [("RESET_TIMER", "1"), ("RESET_TIMER", "0")]
603
           return self.ca.submitMessages(control_messages, "
```

8.8.3.19 setLED()

Definition at line 615 of file LLNL_v4.py.

8.8.3.20 setPowerSave()

```
def nsCamera.boards.LLNL_v4.llnl_v4.setPowerSave (
               self,
               status )
Select powersave option
Aras:
    status: setting for powersave option (1 is enabled)
    tuple (error string, response string from subregister set)
Definition at line 624 of file LLNL_v4.py.
624
       def setPowerSave(self, status):
625
626
           Select powersave option
627
628
629
               status: setting for powersave option (1 is enabled)
630
631
           Returns:
           tuple (error string, response string from subregister set) """
632
633
           if status:
634
               status = 1
635
           return self.ca.setSubregister("POWERSAVE", str(status))
636
637
```

8.8.3.21 setPPER()

```
Definition at line 638 of file LLNL_v4.py.
```

```
def setPPER(self, time):
639
640
            Set polling period for ADCs.
641
642
               time: milliseconds, between 1 and 255; defaults to 50
643
644
645
               tuple (error string, response string from subregister set OR invalid time
646
                 setting string)
647
            if not time:
648
649
               time = 50
            if not isinstance(time, int) or time < 1 or time > 255:
650
651
                err = (
                   self.logerr + "invalid poll period submitted. Setting remains "
653
                    "unchanged. "
654
655
                logging.error(err)
656
                return err, str(time)
657
           else:
               binset = bin(time)[2:].zfill(8)
658
659
                return self.ca.setSubregister("PPER", binset)
660
```

8.8.3.22 softReboot()

```
def nsCamera.boards.LLNL_v4.llnl_v4.softReboot (
               self )
Perform software reboot of board. WARNING: board reboot will likely prevent
  correct response and therefore will generate an error message
Returns:
    tuple (error string, response string) from final control message
Definition at line 461 of file LLNL_v4.py.
461
       def softReboot(self):
462
463
           Perform software reboot of board. WARNING: board reboot will likely prevent
464
             correct response and therefore will generate an error message
465
466
           tuple (error string, response string) from final control message
467
468
469
           logging.info(self.loginfo + "reboot")
           control_messages = [("RESET", "0")]
470
471
           return self.ca.submitMessages(control_messages, " disarm: ")
472
```

8.8.3.23 startCapture()

```
Reads ADC data into SRAM
Returns:
    tuple (error string, response string) from final control message
Definition at line 492 of file LLNL v4.py.
        def startCapture(self, mode="Hardware"):
492
493
            Reads ADC data into SRAM
494
495
496
             Returns:
            tuple (error string, response string) from final control message
497
498
499
            logging.info(self.loginfo + "startCapture")
500
             if self.ca.sensmanual:
501
                 timingReg = "MANSHUT_MODE"
502
             else:
503
                timingReg = "HST_MODE"
504
505
             if mode.upper() == "SOFTWARE":
                 trigmess = [
506
                    ("HW_TRIG_EN", "0"),
507
508
                     ("DUAL_EDGE_TRIG_EN", "0"),
509
                     ("SW_TRIG_EN", "1"),
                     ("SW_TRIG_START", "1"),
510
511
512
             elif mode.upper() == "DUAL":
513
                trigmess = [
                     ("SW_TRIG_EN", "0"),
("HW_TRIG_EN", "1"),
514
515
516
                     ("DUAL_EDGE_TRIG_EN", "1"),
517
                 ]
518
            else: # HARDWARE
519
                trigmess = [
520
                    ("DUAL_EDGE_TRIG_EN", "0"),
                     ("SW_TRIG_EN", "0"),
("HW_TRIG_EN", "1"),
521
522
523
524
            control_messages = [
   ("ADC_CTL", "0000000F"), # configure all ADCs
525
526
                 (timingReg, "1"),
527
528
529
530
             control_messages.extend(trigmess)
531
             return self.ca.submitMessages(control_messages, " startCapture: ")
532
```

8.8.3.24 waitForSRAM()

```
Definition at line 544 of file LLNL_v4.py.
```

```
def waitForSRAM(self, timeout):
545
546
            Wait until subreg 'SRAM_READY' flag is true or timeout is exceeded;
547
             timeout = None or zero means wait indefinitely
548
549
550
               timeout - time in seconds before readoff proceeds automatically without
551
                 waiting for SRAM_READY flag
552
           Returns:
           error string
555
            logging.info(self.loginfo + "waitForSRAM")
556
557
            waiting = True
558
            starttime = time.time()
559
           err = ""
           while waiting:
560
561
                err, status = self.ca.getSubregister("SRAM_READY")
562
                if err:
                    err = self.logerr + "error in register read: " + err + " (waitForSRAM)"
563
564
                    logging.error(err)
565
                if int(status):
566
                    waiting = False
                    logging.info(self.loginfo + "SRAM ready")
567
568
                if self.ca.abort:
569
                    waiting = False
                    logging.info(self.loginfo + "readoff aborted by user")
570
571
                    self.ca.abort = False
                if timeout and time.time() - starttime > timeout:
572
573
                    err += self.logerr + "SRAM timeout; proceeding with download attempt"
574
                    logging.error(err)
575
                    return err
576
577
           return err
578
```

8.8.4 Member Data Documentation

8.8.4.1 ADC5 bipolar

```
nsCamera.boards.LLNL_v4.llnl_v4.ADC5_bipolar
```

Definition at line 230 of file LLNL v4.py.

8.8.4.2 ADC5 mult

```
nsCamera.boards.LLNL_v4.llnl_v4.ADC5_mult
```

Definition at line 227 of file LLNL_v4.py.

8.8.4.3 ca

```
nsCamera.boards.LLNL_v4.llnl_v4.ca
```

Definition at line 219 of file LLNL_v4.py.

8.8.4.4 defoff

```
nsCamera.boards.LLNL_v4.llnl_v4.defoff
```

Definition at line 244 of file LLNL_v4.py.

8.8.4.5 defsens

```
nsCamera.boards.LLNL_v4.llnl_v4.defsens
```

Definition at line 245 of file LLNL_v4.py.

8.8.4.6 dummySensorVals

```
list nsCamera.boards.LLNL_v4.llnl_v4.dummySensorVals [static]
```

Definition at line 179 of file LLNL_v4.py.

8.8.4.7 logcrit

```
nsCamera.boards.LLNL_v4.llnl_v4.logcrit
```

Definition at line 220 of file LLNL_v4.py.

8.8.4.8 logdebug

nsCamera.boards.LLNL_v4.llnl_v4.logdebug

Definition at line 224 of file LLNL_v4.py.

8.8.4.9 logerr

nsCamera.boards.LLNL_v4.llnl_v4.logerr

Definition at line 221 of file LLNL_v4.py.

8.8.4.10 loginfo

nsCamera.boards.LLNL_v4.llnl_v4.loginfo

Definition at line 223 of file LLNL_v4.py.

8.8.4.11 logwarn

nsCamera.boards.LLNL_v4.llnl_v4.logwarn

Definition at line 222 of file LLNL_v4.py.

8.8.4.12 monitor_controls

nsCamera.boards.LLNL_v4.llnl_v4.monitor_controls

Definition at line 289 of file LLNL_v4.py.

8.8.4.13 registers

nsCamera.boards.LLNL_v4.llnl_v4.registers [static]

Definition at line 38 of file LLNL_v4.py.

8.8.4.14 rs422_baud

 ${\tt nsCamera.boards.LLNL_v4.llnl_v4.rs422_baud}$

Definition at line 231 of file LLNL_v4.py.

8.8.4.15 rs422_cmd_wait

```
nsCamera.boards.LLNL_v4.llnl_v4.rs422_cmd_wait
```

Definition at line 232 of file LLNL_v4.py.

8.8.4.16 subreg_aliases

```
\verb|nsCamera.boards.LLNL_v4.llnl_v4.subreg_aliases|\\
```

Definition at line 250 of file LLNL_v4.py.

8.8.4.17 subregisters

```
list nsCamera.boards.LLNL_v4.llnl_v4.subregisters [static]
```

Definition at line 97 of file LLNL_v4.py.

8.8.4.18 subreglist

```
nsCamera.boards.LLNL_v4.llnl_v4.subreglist
```

Definition at line 335 of file LLNL_v4.py.

8.8.4.19 VREF

```
{\tt nsCamera.boards.LLNL\_v4.llnl\_v4.VREF}
```

Definition at line 226 of file LLNL_v4.py.

The documentation for this class was generated from the following file:

nsCamera/boards/LLNL_v4.py

8.9 nsCamera.utils.Ophir.Ophir Class Reference

Public Member Functions

- def init (self)
- def closeOphir (self)
- def OphirTest (self)

Public Attributes

- COM
- DHandle
- DevHan

8.9.2.1 __init__()

8.9.1 Detailed Description

Definition at line 24 of file Ophir.py.

8.9.2 Constructor & Destructor Documentation

```
def nsCamera.utils.Ophir.Ophir.__init__ (
               self )
Definition at line 25 of file Ophir.py.
      def __init__(self): # TODO; add initialization parameters for wavelength, etc.
         self.COM = None
           self.DHandle = None
          OphirCOM = win32com.client.Dispatch("OphirLMMeasurement.CoLMMeasurement")
           # Stop & Close all devices
          OphirCOM.StopAllStreams()
          OphirCOM.CloseAll()
           # Scan for connected Devices
33
          DeviceList = OphirCOM.ScanUSB()
          for Device in DeviceList: # if any device is connected
              DeviceHandle = OphirCOM.OpenUSBDevice(Device) # open first device
              exists = OphirCOM.IsSensorExists(DeviceHandle, 0)
              if exists:
                  self.COM = OphirCOM
                  self.DevHan = DeviceHandle
         if not self.COM:
```

print("Unable to open an Ophir device")
self.COM.SetMeasurementMode(self.DevHan, 0, 1)

self.SetWavelength(self.DevHan, 0, 3)

self.SetThreshold(self.DevHan, 0, 0)

self.SetRange(self.DevHan, 0, 0)

41

43

4.5

46

47

8.9.3 Member Function Documentation

8.9.3.1 closeOphir()

8.9.3.2 OphirTest()

Definition at line 52 of file Ophir.py.

```
def OphirTest (self):
53
           # misc functions
           print("GetDeviceInfo") # (u'StarBright', u'SB1.37', u'795577')
55
           print(self.COM.GetDeviceInfo(self.DevHan))
           print("GetDriverVersion") # WinUSB
56
57
           print(self.COM.GetDriverVersion())
           print("GetSensorInfo") # (u'788341', u'Pyroelectric', u'PD10-pJ-C')
5.8
59
           print(self.COM.GetSensorInfo(self.DevHan, 0))
           print("GetVersion") # 901
60
61
           print(self.COM.GetVersion())
           print("GetDiffuser") # (0, (u'N/A',)) - no adjustable diffuser on sensor
63
           print(self.COM.GetDiffuser(self.DevHan, 0))
           print("GetFilter") # (-1, ()) - only applicable to photodiode sensors
           print(self.COM.GetFilter(self.DevHan, 0))
           print("GetMeasurementMode") # (1, (u'Power', u'Energy', u'Exposure'))
           print(self.COM.GetMeasurementMode(self.DevHan, 0))
68
           print("GetPulseLengths") # (0, (u'5.0us',))
           print(self.COM.GetPulseLengths(self.DevHan, 0))
70
           print("GetRanges") # (0, (u'200nJ', u'20.0nJ', u'2.00nJ', u'200pJ'))
71
           print(self.COM.GetRanges(self.DevHan, 0))
           print("GetThreshold")
           # (0, (u'Min', u'28', u'38', u'48', u'58', u'68', u'78', u'88', u'98', # u'108', u'118', u'128', u'138', u'148', u'158', u'168', u'178', u'188', # u'198', u'208', u'218', u'228', u'238', u'248', u'258'))
           print(self.COM.GetThreshold(self.DevHan, 0))
           print("GetWavelengths")
           # (0, (u'213', u'248', u'355', u'532', u'905', u'1064'))
           print(self.COM.GetWavelengths(self.DevHan, 0))
           print("GetWavelengthsExtra") # (True, 200, 1100) - true is modifiable
           print(self.COM.GetWavelengthsExtra(self.DevHan, 0))
81
83
            # try tweaking some settings
           # Not applicable for PD10: AddWavelength
85
           print("\nTrying some modifications\n")
86
           self.COM.SetMeasurementMode(self.DevHan, 0, 2)
           # self.COM.SaveSettings(self.DevHan, 0) # not sure what this actually does
88
           print("GetMeasurementMode") # (1, (u'Power', u'Energy', u'Exposure'))
89
           print(self.COM.GetMeasurementMode(self.DevHan, 0))
90
91
92
           self.COM.SetWavelength(self.DevHan, 0, 3)
93
           print ("GetWavelengths")
            # (0, (u'213', u'248', u'355', u'532', u'905', u'1064'))
94
```

```
print(self.COM.GetWavelengths(self.DevHan, 0))
96
97
            self.COM.SetRange(self.DevHan, 0, 2)
98
            print("GetRanges") # (0, (u'200nJ', u'20.0nJ', u'2.00nJ', u'200pJ'))
            print(self.COM.GetRanges(self.DevHan, 0))
99
100
101
            self.COM.SetThreshold(self.DevHan, 0, 9)
            print("GetThreshold")
102
             # (0, (u'Min', u'28', u'38', u'48', u'58', u'68', u'78', u'88', u'98',

# u'108', u'118', u'128', u'138', u'148', u'158', u'168', u'178', u'188',

# u'198', u'208', u'218', u'228', u'238', u'248', u'258'))
103
104
105
            print(self.COM.GetThreshold(self.DevHan, 0))
106
107
108
             # self.COM(self.DevHan, 0, 2, 1) #turns on immediate mode, need to watch for
109
                dataready event to use?
110
111
            # An Example for Range control. first get the ranges
             # ranges = self.COM.GetRanges(self.DevHan, 0) # returns outputs as tuple
112
113
            # print (ranges)
114
             # # change range at your will
             \# if ranges[0] > 0:
115
                  newRange = ranges[0] - 1
116
117
             # else:
118
                 newRange = ranges[0] + 1
            # # set new range
119
120
             # self.COM.SetRange(self.DevHan, 0, newRange)
121
            # An Example for data retrieving
122
123
             self.COM.StartStream(self.DevHan, 0) # start measuring
124
            for i in range(10):
                 time.sleep(0.2)
                                   # wait a little for data
125
                 data = self.COM.GetData(self.DevHan, 0)
126
127
                 \# data is length 3 tuple of length n tuples of measurements (double),
128
                 # timestamp (double), status (long)
                 if len(data[0]) > 0:
129
130
                     # if any data available, print the first one from the batch
131
                     print(
132
                          "Reading = \{0\}, TimeStamp = \{1\}, Status = \{2\} ".format(
                              data[0][0], data[1][0], data[2][0]
133
134
135
136
             self.COM.StopStream(self.DevHan, 0)
137
138
             # Restore defaults
139
             self.COM.SetMeasurementMode(self.DevHan, 0, 1)
140
             self.COM.SetWavelength(self.DevHan, 0, 3)
141
             self.COM.SetRange(self.DevHan, 0, 0)
142
             self.COM.SetThreshold(self.DevHan, 0, 0)
143
144
145 """
146 Copyright (c) 2022, Lawrence Livermore National Security, LLC. All rights reserved.
147 LLNL-CODE-838080
149 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
150 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy
151 (DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
152 See license for disclaimers, notice of U.S. Government Rights and license terms and
153 conditions.
154 """
```

8.9.4 Member Data Documentation

8.9.4.1 COM

nsCamera.utils.Ophir.Ophir.COM

Definition at line 26 of file Ophir.py.

8.9.4.2 DevHan

```
nsCamera.utils.Ophir.Ophir.DevHan
```

Definition at line 39 of file Ophir.py.

8.9.4.3 DHandle

```
nsCamera.utils.Ophir.Ophir.DHandle
```

Definition at line 27 of file Ophir.py.

The documentation for this class was generated from the following file:

• nsCamera/utils/Ophir.py

8.10 nsCamera.utils.Packet.Packet Class Reference

Public Member Functions

- def __init__ (self, preamble="aaaa", cmd="0", addr="", data="00000000", seqID="", payload_length="", payload="", crc="")
- def pktStr (self)
- def calculateCRC (self)
- def checkCRC (self)
- def checkReadPacket (self, resppkt)
- def checkResponsePacket (self, resppkt)
- def checkResponseString (self, respstr)
- def str2bytes (self, bstring)
- def bytes2str (self, bytesequence)

Public Attributes

• PY3

Private Attributes

- _preamble
- _cmd
- _addr
- _data
- _seqID
- · _payload_length
- _payload
- _crc
- _type

8.10.1 Detailed Description

Definition at line 28 of file Packet.py.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 __init__()

Definition at line 51 of file Packet.py.

```
def __init__(
     # NOTE: 'numerical' components are handled as hex strings
           self,
          preamble="aaaa",
cmd="0",
54
55
           addr=""
56
          data="00000000",
           seqID="",
58
           payload_length="",
59
         payload="",
60
           crc="",
61
62
           self.PY3 = sys.version_info > (3,)
63
           self._preamble = preamble # 16 bit packet preamble
64
           self._cmd = str(cmd) # 4 bit command packet
65
            self.addr = addr.zfill(3) # 12 bit address packet
self_data = data.zfill(8) # 32 bit data packet
66
67
68
            # 16 bit sequence ID packet (only Read Burst)
```

```
69
           self._seqID = seqID
70
           # 16 bit payload packet (only Read Burst)
71
           self._payload_length = payload_length
           # variable payload packet (only Read Burst) for now it's 16 bits
72
73
           self._payload = payload
74
           # 16 bit CRC-CCIT (XModem) packet
75
           self._crc = crc
76
           self._type = ""
77
           if self._crc == "": # check if packet to be sent needs crc appended
               self._crc = self.calculateCRC()
```

8.10.3 Member Function Documentation

8.10.3.1 bytes2str()

Definition at line 218 of file Packet.py.

```
218
        def bytes2str(self, bytesequence):
219
220
            Python-version-agnostic converter of bytes to hexadecimal strings
221
            Args:
223
                bytesequence: sequence of bytes as string (Py2) or bytes (Py3)
224
225
            Returns:
            hexadecimal string representation of 'bytes' without '0x' """ \ensuremath{\text{"N"}}
226
227
228
            estring = binascii.b2a_hex(bytesequence)
            if self.PY3:
229
230
                estring = str(estring)[2:-1]
231
            return estring
2.32
233
234 """
235 Copyright (c) 2022, Lawrence Livermore National Security, LLC. All rights reserved.
236 LLNL-CODE-838080
2.37
238 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
239 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy
240 (DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
241 See license for disclaimers, notice of U.S. Government Rights and license terms and
242 conditions.
243 """
```

8.10.3.2 calculateCRC()

```
def nsCamera.utils.Packet.Packet.calculateCRC (
                self )
Calculate CRC-CCIT (XModem) (2 bytes) from 8 byte packet for send and rcv
Returns:
    CRC as hexadecimal string without '0x'
Definition at line 107 of file Packet.py.
        def calculateCRC(self):
107
108
109
            Calculate CRC-CCIT (XModem) (2 bytes) from 8 byte packet for send and rcv
110
111
            CRC as hexadecimal string without '0x'
112
113
114
            preamble = self._preamble
115
            crc = self._crc
116
            self._crc = ""
            self._preamble = ""
117
118
119
            CRC_dec = crc16pure.crc16xmodem(self.str2bytes(self.pktStr()))
            # input = int type decimal, output = hex string with 0x at the beginning CRC_hex_0x = "0x%0.4X" % CRC_dec
120
121
122
            \ensuremath{\text{\#}} make all hex letters lower case for comparison
123
            CRC_hex = CRC_hex_0x.lower()
124
            \# input = hex string with 0x at the beginning, output = hex str with 0x removed
            CRC_hex = CRC_hex[2:]
125
126
            self._preamble = preamble
127
            self._crc = crc
128
            return CRC_hex
```

8.10.3.3 checkCRC()

129

```
def nsCamera.utils.Packet.Packet.checkCRC (
               self )
Returns: boolean, True if CRC check passes
Definition at line 130 of file Packet.pv.
130
        def checkCRC(self):
131
132
            Returns: boolean, True if CRC check passes
133
           if self.calculateCRC() == self._crc:
134
135
               return True
136
            else:
137
               return False
```

138

8.10.3.4 checkReadPacket()

```
def nsCamera.utils.Packet.Packet.checkReadPacket (
                  self,
                  resppkt )
Confirm that Read Single occurred without error
Args:
     resppkt: response packet
     tuple (error string, response packet as string)
Definition at line 139 of file Packet.py.
139
         def checkReadPacket(self, resppkt):
140
141
             Confirm that Read Single occurred without error
142
143
                 resppkt: response packet
144
145
             Returns:
             tuple (error string, response packet as string) _{\tt nnn}
146
147
148
             err = ""
             if int(resppkt._cmd.upper(), 16) - int(self._cmd.upper(), 16) != 0x8:
    err = "invalid command; "
149
150
             if resppkt._addr.upper() != self._addr.upper():
    err += "invalid address; "
151
152
             if resppkt._crc.upper() != resppkt.calculateCRC().upper():
    err += "invalid CRC; "
155
             return err, resppkt.pktStr()
```

8.10.3.5 checkResponsePacket()

```
def nsCamera.utils.Packet.Packet.checkResponsePacket (
                self.
                resppkt )
Confirm that Write Single occurred without error
Args:
    resppkt: response packet
Returns:
    tuple (error string, response packet as string)
Definition at line 157 of file Packet.py.
157
        def checkResponsePacket(self, resppkt):
158
159
            Confirm that Write Single occurred without error
160
            Args:
161
                resppkt: response packet
162
163
            Returns:
            tuple (error string, response packet as string)
164
165
            err = ""
166
            if int(resppkt._data, 16) & 1:
    err += "Checksum error; "
167
168
            if int(resppkt._data, 16) & 2:
169
                err += "Invalid command / command not executed; "
170
            err1, rval = self.checkReadPacket(resppkt)
171
172
            err += err1
            return err, rval
173
174
```

8.10.3.6 checkResponseString()

```
def nsCamera.utils.Packet.Packet.checkResponseString (
               respstr )
Checks response string for error indicators
Args:
    respstr: packet as hexadecimal string
Returns:
    tuple (error string, response packet string)
Definition at line 175 of file Packet.py.
        def checkResponseString(self, respstr):
175
176
177
            Checks response string for error indicators
178
179
               respstr: packet as hexadecimal string
180
            Returns:
181
           tuple (error string, response packet string)
182
183
            {\tt respstring = respstr.decode(encoding="UTF-8")}
184
185
            resppkt = Packet(
186
               preamble=respstring[0:4],
187
                cmd=respstring[4],
188
                addr=respstring[5:8],
189
               data=respstring[8:16],
190
           )
191
           if resppkt._cmd == "8":
192
193
                # verify response to write command
194
                err, rval = self.checkResponsePacket(resppkt)
195
           elif resppkt._cmd == "9":
196
               err, rval = self.checkReadPacket(resppkt) # verify response to read command
197
198
               err = "Packet command invalid; "
               rval = ""
199
200
           return err, rval
201
```

8.10.3.7 pktStr()

```
def nsCamera.utils.Packet.Packet.pktStr (
                self )
Generate hexadecimal string form of packet
Returns:
    packet as hexadecimal string without ' \, \text{Ox}'
Definition at line 80 of file Packet.py.
80
       def pktStr(self):
81
           Generate hexadecimal string form of packet
82
83
84
           packet as hexadecimal string without {\rm '0x'}
8.5
86
```

if self._seqID != "":

87

```
88
               # Read burst response
               packetparts = [
90
                   self._preamble,
91
                   self._cmd,
                   self._seqID,
93
                   self._payload_length,
94
                   self._payload,
95
                   self._crc,
           else:
               # Single Command/Response response
               packetparts = [self._preamble, self._cmd, self._addr, self._data, self._crc]
100
            stringparts = [
               part.decode("ascii") if isinstance(part, bytes) else part
101
102
                for part in packetparts
103
           out = "".join(stringparts)
104
105
            return out
106
```

8.10.3.8 str2bytes()

```
def nsCamera.utils.Packet.Packet.str2bytes (
               self,
               bstring )
Python-version-agnostic converter of hexadecimal strings to bytes
Aras:
    bstring: hexadecimal string without '0x'
    byte string equivalent to input string
Definition at line 202 of file Packet.py.
       def str2bytes(self, bstring):
2.02
203
2.04
           Python-version-agnostic converter of hexadecimal strings to bytes
2.05
206
           Args:
207
               bstring: hexadecimal string without '0x'
208
           Returns:
209
           byte string equivalent to input string
210
211
           if self.PY3:
212
213
               dbytes = binascii.a2b_hex(bstring)
214
215
               dbytes = bstring.decode("hex")
216
           return dbytes
217
```

8.10.4 Member Data Documentation

8.10.4.1 _addr

nsCamera.utils.Packet.Packet._addr [private]

Definition at line 55 of file Packet.py.

8.10.4.2 _cmd

```
nsCamera.utils.Packet.Packet._cmd [private]
```

Definition at line 54 of file Packet.py.

8.10.4.3 _crc

```
nsCamera.utils.Packet.Packet._crc [private]
```

Definition at line 64 of file Packet.py.

8.10.4.4 _data

```
nsCamera.utils.Packet.Packet._data [private]
```

Definition at line 56 of file Packet.py.

8.10.4.5 _payload

```
nsCamera.utils.Packet.Packet._payload [private]
```

Definition at line 62 of file Packet.py.

8.10.4.6 _payload_length

```
nsCamera.utils.Packet.Packet._payload_length [private]
```

Definition at line 60 of file Packet.py.

8.10.4.7 _preamble

```
nsCamera.utils.Packet.Packet._preamble [private]
```

Definition at line 53 of file Packet.py.

8.10.4.8 _seqID

```
nsCamera.utils.Packet.Packet._seqID [private]
```

Definition at line 58 of file Packet.py.

8.10.4.9 _type

```
nsCamera.utils.Packet.Packet._type [private]
```

Definition at line 65 of file Packet.py.

8.10.4.10 PY3

```
nsCamera.utils.Packet.Packet.PY3
```

Definition at line 52 of file Packet.py.

The documentation for this class was generated from the following file:

nsCamera/utils/Packet.py

8.11 nsCamera.comms.RS422.RS422 Class Reference

Public Member Functions

- def __init__ (self, camassem, baud=921600, par="O", stop=1)
- def serialClose (self)
- def sendCMD (self, pkt)
- def arm (self, mode)
- def readoff (self, waitOnSRAM, timeout, fast)
- def writeSerial (self, outstring, timeout)
- def readSerial (self, size, timeout=None)
- def closeDevice (self)

Public Attributes

- ca
- logcrit
- logerr
- logwarn
- loginfo
- logdebug
- mode
- PY3
- skipError
- · payloadsize

Private Attributes

- baud
- _par
- _stop
- · _read_timeout
- · _write_timeout
- _datatimeout
- _port
- _ser

8.11.1 Detailed Description

```
Code to manage RS422 connection. Will automatically query available COM interfaces until a board is found. Use the 'port=x' parameter in cameraAssembler call to specify a particular COM interface.

Exposed methods:

arm() - Puts camera into wait state for external trigger readoff() - Waits for data ready register flag, then copies camera image data into numpy arrays sendCMD(pkt) - sends packet object via serial port readSerial(size, timeout) - read 'size' bytes from serial port writeSerial(cmd) - submits string 'cmd' (assumes string is preformed packet) closeDevice() - close serial connections
```

Definition at line 28 of file RS422.py.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 __init__()

```
def nsCamera.comms.RS422.RS422.__init__ (
                 self,
                 camassem,
                 baud = 921600,
                 par = "0",
                 stop = 1)
Args:
     camassem: parent cameraAssembler object
    baud: bits per second
    par: parity type
    stop: number of stop bits
Definition at line 44 of file RS422.pv.
       def __init__(self, camassem, baud=921600, par="0", stop=1):
44
45
           Args:
46
                camassem: parent cameraAssembler object
47
48
                baud: bits per second
49
                par: parity type
50
               stop: number of stop bits
51
           self.ca = camassem
52
           self.logcrit = self.ca.logcritbase + "[RS422] "
5.3
            self.logerr = self.ca.logerrbase + "[RS422]
54
            self.logwarn = self.ca.logwarnbase + "[RS422] "
55
           self.loginfo = self.ca.loginfobase + "[RS422] "
self.logdebug = self.ca.logdebugbase + "[RS422] "
56
57
5.8
           logging.info(self.loginfo + "initializing comms object")
59
            self.mode = 0
            self._baud = baud # Baud rate (bits/second)
60
61
           self._par = par # Parity bit
            self._stop = stop # Number of stop bits
62
            self._read_timeout = 1 # default timeout for ordinary packets
63
64
            self.\_write\_timeout = 1
65
           self._datatimeout = 5e7 * self.ca.sensor.nframes / baud # timeout for data read
66
            self.PY3 = sys.version_info > (3,)
67
           self.skipError = False
68
           port = ""
           ports = list(serial.tools.list_ports.comports())
69
           for p, desc, add in ports:
    if self.ca.port is None or p == "COM" + str(self.ca.port):
70
71
                    logging.info(self.loginfo + "found comm port " + p)
72
73
                    try:
74
                        with serial. Serial (
75
                            p,
76
                             self._baud,
77
                             parity=self._par,
78
                             timeout=0.01,
79
                             write_timeout=0.01,
80
                         ) as ser:
81
                             ser.write(self.ca.str2bytes("aaaa1000000000001a84"))
82
                            time.sleep(1)
83
                             s = ser.read(10)
84
                             resp = self.ca.bytes2str(s)
85
                             if (
                                 resp[0:5].lower() == "aaaa9"
86
                             ): # TODO: add check for RS422 bit in board description
                                 boardid = resp[8:10]
if boardid == "00":
88
89
                                     logging.critical(
90
91
                                         self.logcrit + "SNLrevC board detected - not "
92
                                          "compatible with nsCamera >= 2.0"
93
                                     sys.exit(1)
94
                                 elif boardid == "81":
95
                                     logging.info(self.loginfo + "LLNLv1 board detected")
96
                                 elif boardid == "84":
97
                                     logging.info(self.loginfo + "LLNLv4 board detected")
98
99
                                 else:
```

```
100
                                    logging.info(
101
                                        self.loginfo + "unidentified board detected"
102
103
                                logging.info(self.loginfo + "connected to " + p)
104
105
                                ser.reset_input_buffer()
106
                                ser.reset_output_buffer()
107
108
                    except Exception as e:
                        logging.error(self.logerr + "port identification: " + str(e))
109
            if port == "":
                if self.ca.port:
111
                    logging.critical(
112
113
                        self.logcrit + "No usable board found at port " + str(self.ca.port)
114
115
                    sys.exit(1)
116
                else:
117
                   logging.critical(self.logcrit + "No usable board found")
118
                    sys.exit(1)
            self._port = port # COM port to use for RS422 link
119
            self.ca.port = port[3:] # re-extract port number from com name
120
121
            self. ser = serial.Serial( # Class RS422
122
123
                port=self._port,
124
                baudrate=self._baud,
125
                parity=self._par,
                stopbits=self._stop,
126
127
                timeout=self._read_timeout, # timeout for serial read
128
                bytesize=serial.EIGHTBITS,
129
            self.payloadsize = (
130
131
               self.ca.sensor.width
132
                * self.ca.sensor.height
133
                * self.ca.sensor.nframes
134
                * self.ca.sensor.bytesperpixel
135
136
            self._ser.flushInput()
137
            if not self._ser.is_open:
                logging.critical(self.logcrit + "Unable to open serial connection")
138
139
                sys.exit(1)
140
```

8.11.3 Member Function Documentation

8.11.3.1 arm()

Definition at line 288 of file RS422.py.

```
def arm(self, mode):
289
290
            Puts camera into wait state for trigger. Mode determines source; arm() in
291
              CameraAssembler defaults to 'Hardware'
292
293
            Args:
294
               mode:
                        'Software' activates software triggering, disables hardware trigger
295
                        'Hardware' activates hardware triggering, disables software trigger
296
                          Hardware is the default
297
                        'Dual' activates dual edge hardware trigger mode and disables
298
                          software trigger
299
300
            Returns:
            tuple (error, response string)
301
302
303
            if not mode:
               mode = "Hardware"
304
305
            logging.info(self.loginfo + "arm")
306
            self.ca.clearStatus()
307
            self.ca.latchPots()
308
            err, resp = self.ca.startCapture(mode)
309
            if err:
310
               logging.error(self.logerr + "unable to arm camera")
311
            else:
312
               self.ca.armed = True
313
                self.skipError = True
314
            return err, resp
315
```

8.11.3.2 closeDevice()

Close primary serial interface

Definition at line 402 of file RS422.pv.

```
def closeDevice(self):
402
403
            Close primary serial interface
404
405
406
            self._ser.close()
407
408
409 """
410 Copyright (c) 2022, Lawrence Livermore National Security, LLC. All rights reserved.
411 LLNL-CODE-838080
412
413 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
414 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy
415 (DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
416 See license for disclaimers, notice of U.S. Government Rights and license terms and
417 conditions.
418 """
```

8.11.3.3 readoff()

```
def nsCamera.comms.RS422.RS422.readoff (
               self.
               waitOnSRAM,
               timeout.
               fast )
Copies image data from board into numpy arrays
Args:
    waitOnSRAM: if True, wait until SRAM_READY flag is asserted to begin copying
    timeout: passed to waitForSRAM; after this many seconds begin copying data
      irrespective of SRAM_READY status; 'zero' means wait indefinitely
      WARNING: If acquisition fails, the SRAM will not contain a current image,
but the code will copy the data anyway
    fast: if False, parse and convert frames to numpy arrays; if True, return
      unprocessed text stream
Returns:
    tuple (list of numpy arrays OR raw text stream, length of downloaded payload
      in bytes, payload error flag)
    NOTE: This reduces readoff by <1 second, so will have no noticeable impact
      when using RS422
Definition at line 316 of file RS422.pv.
316
        def readoff(self, waitOnSRAM, timeout, fast):
317
318
           Copies image data from board into numpy arrays
319
               waitOnSRAM: if True, wait until SRAM_READY flag is asserted to begin copying
320
321
                 data
               timeout: passed to waitForSRAM; after this many seconds begin copying data
322
                 irrespective of SRAM_READY status; 'zero' means wait indefinitely
323
                 WARNING: If acquisition fails, the SRAM will not contain a current image,
324
325
                   but the code will copy the data anyway
326
               fast: if False, parse and convert frames to numpy arrays; if True, return
327
                 unprocessed text stream
328
329
           Returns:
330
               tuple (list of numpy arrays OR raw text stream, length of downloaded payload
331
                 in bytes, payload error flag)
332
               NOTE: This reduces readoff by <1 second, so will have no noticeable impact
333
                 when using RS422
334
335
           logging.info(self.loginfo + "readoff")
336
           errortemp = False
337
338
           # Wait for data to be ready on board, turns off error messaging
339
            # Skip wait only if explicitly tagged 'False' ('None' defaults to True)
340
           if not waitOnSRAM == False:
341
               logging.getLogger().setLevel(logging.CRITICAL)
342
                self.ca.waitForSRAM(timeout)
343
               logging.getLogger().setLevel(self.ca.verblevel)
344
           # Retrieve data
345
346
           err, rval = self.ca.readSRAM()
347
348
               logging.error(self.logerr + "Error detected in readSRAM")
349
           time.sleep(0.3)
350
            # extract only the read burst data. Remove header & CRC footer
351
           read_burst_data = rval[36:-4]
352
353
           if fast:
354
               return read_burst_data, len(read_burst_data) // 2, errortemp
355
356
               parsed = self.ca.generateFrames(read_burst_data)
357
               return parsed, len(read_burst_data) // 2, errortemp
358
```

8.11.3.4 readSerial()

```
def nsCamera.comms.RS422.RS422.readSerial (
                self,
                size,
                timeout = None )
Read bytes from the serial port. Does not verify packets.
   size: number of bytes to read
   timeout: serial timeout in sec
Returns:
   tuple (error string, string read from serial port)
Definition at line 375 of file RS422.py.
375
        def readSerial(self, size, timeout=None):
376
377
            Read bytes from the serial port. Does not verify packets.
378
379
380
              size: number of bytes to read
381
               timeout: serial timeout in sec
382
383
            tuple (error string, string read from serial port) _{\mbox{\scriptsize mum}}
384
385
            err = ""
386
387
            if timeout:
388
               self._ser.timeout = timeout
389
390
               self._ser.timeout = self._read_timeout
391
            resp = self._ser.read(size)
            if len(resp) < 10: # bytes
392
393
                err += (
394
                   self.logerr
395
                    + "readSerial : packet too small: '"
396
                    + self.ca.bytes2str(resp)
397
398
399
                logging.error(err)
400
            return err, self.ca.bytes2str(resp)
401
```

8.11.3.5 sendCMD()

```
Definition at line 147 of file RS422.py.
```

```
def sendCMD(self, pkt):
148
149
            Submit packet and verify response packet. Recognizes readoff packet and adjusts.
150
            Read size and timeout appropriately
151
152
153
               pkt: Packet object
154
155
            Returns:
            tuple (error, response string)
157
158
            pktStr = pkt.pktStr()
            self._ser.flushInput()
159
160
            time.sleep(0.01) # wait 10 ms in between flushing input and output buffers
161
            self._ser.flushOutput()
162
            self.ca.writeSerial(pktStr)
            err0 = ""
163
            err = ""
164
            resp = ""
165
            tries = 3 # make a function parameter?
166
167
168
169
                hasattr(self.ca, "board")
                and pktStr[4] == "0"
170
171
                and pktStr[5:8] == self.ca.board.registers["SRAM_CTL"]
172
            ):
173
                logging.info(
                    self.loginfo + "Payload size (bytes) = " + str(self.payloadsize)
174
175
176
                crcresp0 = ""
                crcresp1 = ""
177
                smallresp = ""
178
                emptyResponse = False
179
180
                wrongSize = False
181
                for i in range(tries):
182
                    err, resp = self.readSerial(
183
                        self.payloadsize + 20, timeout=self._datatimeout
184
                    if err:
185
186
                        logging.error(
                             self.logerr + "sendCMD: read payload failed " + pktStr + err
187
188
189
                        self.ca.payloaderror = True
190
191
                         if not len(resp):
192
                             err0 = self.logerr + "sendCMD: empty response from board"
193
                             logging.error(err0)
194
                             emptyResponse = True
195
                             self.ca.payloaderror = True
196
                         elif len(resp) != 2 * (self.payloadsize + 20):
197
                             err0 = (
198
                                self.logerr
199
                                 + "sendCMD: incorrect response; expected "
200
                                 + str(self.payloadsize + 20)
201
                                 + " bytes, received "
202
                                 + str(len(resp) // 2)
203
204
                             logging.error(err0)
205
                             wrongSize = True
                             smallresp = resp
206
207
                             self.ca.payloaderror = True
208
                         elif not self.ca.checkCRC(resp[4:20]):
209
                             err0 = (
210
                                self.logerr
                                 + "sendCMD:
211
212
                                 + pktStr
                                 + " - payload preface CRC fail"
213
214
215
                             logging.error(err0)
216
                             self.ca.payloaderror = True
217
                             crcresp1 = resp
218
                         elif not self.ca.checkCRC(resp[24:]):
                             err0 = (
219
                                self.logerr + "sendCMD: " + pktStr + " - payload CRC fail"
220
221
222
                             logging.error(err0)
223
                             self.ca.payloaderror = True
224
                             crcresp0 = resp
225
                        err += err0
                    time.sleep(5)
226
```

```
227
                     if self.ca.payloaderror:
228
                         if (
229
                             i == tries - 1
230
                         ): # keep best results over multiple tries; e.g., if first try is
231
                                 bad CRC and second try is an incomplete payload, use the
232
                                 first payload
233
                             if crcresp0:
234
                                 logging.error(
235
                                      self.logerr + "sendCMD: Unable to acquire "
236
                                      "CRC-confirmed payload after "
237
                                      + str(tries)
238
                                      + " attempts. Continuing with unconfirmed payload"
239
240
                                 resp = crcresp0
241
                             elif crcresp1:
242
                                 logging.error(
                                      self.logerr + "sendCMD: Unable to acquire "
243
244
                                      "CRC-confirmed readoff after "
245
                                      + str(tries)
246
                                      + " attempts. Continuing with unconfirmed payload"
247
248
                                 resp = crcresp1
249
                             elif wrongSize:
250
                                 logging.error(
251
                                      self.logerr + "sendCMD: Unable to acquire complete "
                                      "payload after "
252
253
                                      + str(tries)
                                      + " attempts. Dumping datastream to file."
254
2.5.5
                                 )
256
                                 resp = smallresp
2.57
                                 self.ca.dumpNumpy(resp)
258
                             elif emptyResponse:
259
                                 logging.error(
                                      self.logerr + "sendCMD: Unable to acquire any "
"payload after " + str(tries) + " attempts."
2.60
261
2.62
2.63
                         else:
264
                             logging.info(
                                 self.loginfo + "Retrying download, attempt #" + str(i + 1)
2.65
266
                             err = ""
267
                             err0 = ""
268
269
                             self.ca.payloaderror = False
270
                             self.ca.writeSerial(pktStr)
271
272
                         logging.info(self.loginfo + "Download successful")
273
274
275
276
                time.sleep(0.03)
277
                self._ser.timeout = 0.02
278
                err, resp = self.readSerial(10)
279
                if err:
280
                     logging.error(
281
                         self.logerr + "sendCMD: readSerial failed (regular packet) " + err
282
283
                 elif not self.ca.checkCRC(resp[4:20]):
284
                     err = self.logerr + "sendCMD- regular packet CRC fail: " + resp
285
                     logging.error(err)
286
            return err, resp
287
```

8.11.3.6 serialClose()

Close serial interface

Definition at line 141 of file RS422.py.

```
141 def serialClose(self):
142 """
143 Close serial interface
144 """
145 self._ser.close() # close serial interface COM port
146
```

8.11.3.7 writeSerial()

Definition at line 359 of file RS422.py.

```
def writeSerial(self, outstring, timeout):
    """
359
360
361
            Args:
362
                outstring: string to write
363
                timeout: serial timeout in sec
364
            Returns:
            . integer length of string written to serial port _{\tt mum}
365
366
367
            if timeout:
368
                self._ser.timeout = timeout
369
370
                self._ser.timeout = self._write_timeout
371
            lengthwritten = self._ser.write(self.ca.str2bytes(outstring))
372
            self._ser.timeout = self._read_timeout
373
            return lengthwritten
374
```

8.11.4 Member Data Documentation

8.11.4.1 _baud

```
nsCamera.comms.RS422.RS422._baud [private]
```

Definition at line 60 of file RS422.py.

236 Class Documentation

8.11.4.2 _datatimeout

```
nsCamera.comms.RS422.RS422._datatimeout [private]
```

Definition at line 65 of file RS422.py.

8.11.4.3 _par

```
nsCamera.comms.RS422.RS422._par [private]
```

Definition at line 61 of file RS422.py.

8.11.4.4 _port

```
nsCamera.comms.RS422.RS422._port [private]
```

Definition at line 119 of file RS422.py.

8.11.4.5 _read_timeout

```
nsCamera.comms.RS422.RS422._read_timeout [private]
```

Definition at line 63 of file RS422.py.

8.11.4.6 ser

```
nsCamera.comms.RS422.RS422._ser [private]
```

Definition at line 122 of file RS422.py.

8.11.4.7 _stop

```
nsCamera.comms.RS422.RS422._stop [private]
```

Definition at line 62 of file RS422.py.

8.11.4.8 _write_timeout

nsCamera.comms.RS422.RS422._write_timeout [private]

Definition at line 64 of file RS422.py.

8.11.4.9 ca

nsCamera.comms.RS422.RS422.ca

Definition at line 52 of file RS422.py.

8.11.4.10 logcrit

nsCamera.comms.RS422.RS422.logcrit

Definition at line 53 of file RS422.py.

8.11.4.11 logdebug

nsCamera.comms.RS422.RS422.logdebug

Definition at line 57 of file RS422.py.

8.11.4.12 logerr

nsCamera.comms.RS422.RS422.logerr

Definition at line 54 of file RS422.py.

8.11.4.13 loginfo

nsCamera.comms.RS422.RS422.loginfo

Definition at line 56 of file RS422.py.

238 Class Documentation

8.11.4.14 logwarn

nsCamera.comms.RS422.RS422.logwarn

Definition at line 55 of file RS422.py.

8.11.4.15 mode

nsCamera.comms.RS422.RS422.mode

Definition at line 59 of file RS422.py.

8.11.4.16 payloadsize

nsCamera.comms.RS422.RS422.payloadsize

Definition at line 130 of file RS422.py.

8.11.4.17 PY3

nsCamera.comms.RS422.RS422.PY3

Definition at line 66 of file RS422.py.

8.11.4.18 skipError

nsCamera.comms.RS422.RS422.skipError

Definition at line 67 of file RS422.py.

The documentation for this class was generated from the following file:

nsCamera/comms/RS422.py

8.12 nsCamera.utils.Subregister.SubRegister Class Reference

Public Member Functions

def __init__ (self, board, name, register, start_bit=31, width=8, writable=False, value=255, minV=0, maxV=5)

Public Attributes

- name
- · register
- addr
- start_bit
- width
- value
- max_value
- min
- max
- writable
- minV
- maxV
- resolution

8.12.1 Detailed Description

```
Represents a subset of a 32-bit register [31..0] starting at 'start_bit' consisting of 'width' bits. Consistent with the ICD usage, start_bit is MSB e.g., for [7..0], the start_bit is '7'.
```

Definition at line 21 of file Subregister.py.

8.12.2 Constructor & Destructor Documentation

240 **Class Documentation**

8.12.2.1 __init__()

```
def nsCamera.utils.Subregister.SubRegister.__init__ (
              self,
              board,
              name,
              register,
              start_bit = 31,
              width = 8,
              writable = False,
              value = 255,
              minV = 0,
              maxV = 5)
```

Definition at line 28 of file Subregister.py.

```
2.8
       def __init__(
29
           self.
30
           board,
31
           name,
32
           register,
33
           start bit=31,
34
           width=8,
3.5
           writable=False,
36
           value=255,
37
           minV=0,
           maxV=5,
38
39
      ):
40
           self.name = name
41
           self.register = register
42
           self.addr = board.registers[register]
43
           self.start_bit = start_bit
          self.width = width
self.value = value
45
46
           self.max\_value = 2 ** width - 1 # used to normalize the input values to 1
47
           self.min = 0
48
           self.max = self.max_value
49
           self.writable = writable
50
           self.minV = minV
51
           self.maxV = maxV
           # resolution should be reset after init if actual min and max are different
           self.resolution = (1.0 * maxV - minV) / self.max_value
55
56 """
57 Copyright (c) 2022, Lawrence Livermore National Security, LLC. All rights reserved.
60 This work was produced at the Lawrence Livermore National Laboratory (LLNL) under
61 contract no. DE-AC52-07NA27344 (Contract 44) between the U.S. Department of Energy
62 (DOE) and Lawrence Livermore National Security, LLC (LLNS) for the operation of LLNL.
63 See license for disclaimers, notice of U.S. Government Rights and license terms and
64 conditions.
65 ""
```

8.12.3 Member Data Documentation

8.12.3.1 addr

nsCamera.utils.Subregister.SubRegister.addr

Definition at line 31 of file Subregister.py.

8.12.3.2 max

nsCamera.utils.Subregister.SubRegister.max

Definition at line 37 of file Subregister.py.

8.12.3.3 max_value

nsCamera.utils.Subregister.SubRegister.max_value

Definition at line 35 of file Subregister.py.

8.12.3.4 maxV

 $\verb|nsCamera.utils.Subregister.SubRegister.maxV| \\$

Definition at line 40 of file Subregister.py.

8.12.3.5 min

nsCamera.utils.Subregister.SubRegister.min

Definition at line 36 of file Subregister.py.

8.12.3.6 minV

nsCamera.utils.Subregister.SubRegister.minV

Definition at line 39 of file Subregister.py.

8.12.3.7 name

 $\verb|nsCamera.utils.Subregister.SubRegister.name|\\$

Definition at line 29 of file Subregister.py.

242 Class Documentation

8.12.3.8 register

nsCamera.utils.Subregister.SubRegister.register

Definition at line 30 of file Subregister.py.

8.12.3.9 resolution

nsCamera.utils.Subregister.SubRegister.resolution

Definition at line 42 of file Subregister.py.

8.12.3.10 start_bit

nsCamera.utils.Subregister.SubRegister.start_bit

Definition at line 32 of file Subregister.py.

8.12.3.11 value

 $\verb|nsCamera.utils.Subregister.SubRegister.value|\\$

Definition at line 34 of file Subregister.py.

8.12.3.12 width

 $\verb|nsCamera.utils.Subregister.SubRegister.width|\\$

Definition at line 33 of file Subregister.py.

8.12.3.13 writable

 $\verb|nsCamera.utils.Subregister.SubRegister.writable|$

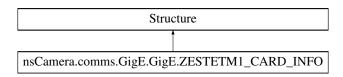
Definition at line 38 of file Subregister.py.

The documentation for this class was generated from the following file:

nsCamera/utils/Subregister.py

8.13 nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO Class Reference

Inheritance diagram for nsCamera.comms.GigE.GigE.ZESTETM1 CARD INFO:



Static Public Attributes

```
int ubyte4 = C.c_ubyte * 4int ubyte6 = C.c_ubyte * 6
```

Static Private Attributes

list _fields_

8.13.1 Detailed Description

Definition at line 411 of file GigE.py.

8.13.2 Member Data Documentation

8.13.2.1 fields

```
list nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO._fields_ [static], [private]
```

Initial value:

```
[
    ("IPAddr", ubyte4),
    ("ControlPort", C.c_ushort),
    ("Timeout", C.c_ulong),
    ("HTTPPort", C.c_ushort),
    ("MACAddr", ubyte6),
    ("SubNet", ubyte4),
    ("Gateway", ubyte4),
    ("SerialNumber", C.c_ulong),
    ("FirmwareVersion", C.c_ulong),
    ("HardwareVersion", C.c_ulong),
}
```

Definition at line 414 of file GigE.py.

244 Class Documentation

8.13.2.2 ubyte4

```
int nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO.ubyte4 = C.c_ubyte * 4 [static]
```

Definition at line 412 of file GigE.py.

8.13.2.3 ubyte6

```
int nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO.ubyte6 = C.c_ubyte * 6 [static]
```

Definition at line 413 of file GigE.py.

The documentation for this class was generated from the following file:

• nsCamera/comms/GigE.py

Chapter 9

File Documentation

9.1 nsCamera/__init__.py File Reference

Namespaces

nsCamera

Variables

• list nsCamera.__all__ = ["CameraAssembler.py"]

9.2 nsCamera/boards/__init__.py File Reference

Namespaces

nsCamera.boards

Variables

• list nsCamera.boards.__all__ = ["LLNL_v1", "LLNL_v4"]

9.3 nsCamera/comms/__init__.py File Reference

Namespaces

nsCamera.comms

246 File Documentation

Variables

• list nsCamera.comms.__all__ = ["RS422", "GigE"]

9.4 nsCamera/sensors/__init__.py File Reference

Namespaces

nsCamera.sensors

Variables

• list nsCamera.sensors.__all__ = ["icarus", "icarus2", "daedalus"]

9.5 nsCamera/utils/__init__.py File Reference

Namespaces

nsCamera.utils

Variables

• list nsCamera.utils.__all__ = ["SubRegister", "Packet", "GenTec", "Ophir", "FlatField"]

9.6 nsCamera/boards/LLNL_v1.py File Reference

Classes

• class nsCamera.boards.LLNL_v1.llnl_v1

Namespaces

• nsCamera.boards.LLNL v1

9.7 nsCamera/boards/LLNL_v4.py File Reference

Classes

class nsCamera.boards.LLNL v4.llnl v4

Namespaces

• nsCamera.boards.LLNL_v4

9.8 nsCamera/CameraAssembler.py File Reference

Classes

· class nsCamera.CameraAssembler.CameraAssembler

Namespaces

nsCamera.CameraAssembler

9.9 nsCamera/CODE_OF_CONDUCT.md File Reference

9.10 nsCamera/comms/GigE.py File Reference

Classes

- · class nsCamera.comms.GigE.GigE
- class nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO

Namespaces

• nsCamera.comms.GigE

9.11 nsCamera/comms/RS422.py File Reference

Classes

· class nsCamera.comms.RS422.RS422

Namespaces

nsCamera.comms.RS422

248 File Documentation

9.12 nsCamera/CONTRIBUTING.md File Reference

9.13 nsCamera/docs/nsCameraExample.py File Reference

Namespaces

nsCameraExample

Variables

- string nsCameraExample.BOARD = "LLNL_v4"
- string nsCameraExample.COMM = "GigE"
- string nsCameraExample.SENSOR = "icarus2"
- nsCameraExample.ca = CameraAssembler(commname=COMM, boardname=BOARD, sensorname=SENSOR, verbose=4)
- nsCameraExample.timing
- nsCameraExample.tune

9.14 nsCamera/docs/testSuite.py File Reference

Namespaces

· testSuite

Functions

def testSuite.testSuite (board, comm, sensor, portNum, ipAdd, interactive=True, swtrigger=True)

Variables

- testSuite.parser = argparse.ArgumentParser()
- · testSuite.action
- testSuite.dest
- testSuite.default
- · testSuite.help
- testSuite.None
- testSuite.args = parser.parse args()
- testSuite.interactive
- testSuite.swtrigger
- · testSuite.board
- testSuite.comm
- testSuite.sensor
- · testSuite.portNum
- testSuite.ipAdd

9.15 nsCamera/sensors/daedalus.py File Reference

Classes

· class nsCamera.sensors.daedalus.daedalus

Namespaces

· nsCamera.sensors.daedalus

9.16 nsCamera/sensors/icarus.py File Reference

Classes

· class nsCamera.sensors.icarus.icarus

Namespaces

• nsCamera.sensors.icarus

9.17 nsCamera/sensors/icarus2.py File Reference

Classes

· class nsCamera.sensors.icarus2.icarus2

Namespaces

• nsCamera.sensors.icarus2

9.18 nsCamera/utils/crc16pure.py File Reference

Namespaces

• nsCamera.utils.crc16pure

Functions

- def nsCamera.utils.crc16pure._crc16 (data, crc, table)
- def nsCamera.utils.crc16pure.crc16xmodem (data, crc=0)

250 File Documentation

Variables

list nsCamera.utils.crc16pure.CRC16 XMODEM TABLE

9.19 nsCamera/utils/FlatField.py File Reference

Namespaces

· nsCamera.utils.FlatField

Functions

- def nsCamera.utils.FlatField.getFilenames (frame="Frame 1")
- def nsCamera.utils.FlatField.getROlvector (imgfilename, roi)
- def nsCamera.utils.FlatField.tslopes (x, y)
- def nsCamera.utils.FlatField.generateFF (FRAMES=["Frame_0", "Frame_1", "Frame_2", "Frame_3"], roi=[0, 0, 512, 1024], directory="", ncores=-1)
- def nsCamera.utils.FlatField.removeFF (filename, directory="", roi=[0, 0, 512, 1024])
- def nsCamera.utils.FlatField.removeFFall (directory="", FRAMES=["Frame_0", "Frame_1", "Frame_2", "Frame _2", "Fr

Variables

- nsCamera.utils.FlatField.parser = argparse.ArgumentParser()
- · nsCamera.utils.FlatField.action
- nsCamera.utils.FlatField.dest
- nsCamera.utils.FlatField.default
- · nsCamera.utils.FlatField.help
- nsCamera.utils.FlatField.nargs
- nsCamera.utils.FlatField.args = parser.parse_args()
- list nsCamera.utils.FlatField.framelist = ["Frame_" + str(frame) for frame in args.frames]
- · nsCamera.utils.FlatField.directory

9.20 nsCamera/utils/GenTec.py File Reference

Classes

class nsCamera.utils.GenTec.GenTec

Namespaces

nsCamera.utils.GenTec

Variables

nsCamera.utils.GenTec.gt = GenTec()

9.21 nsCamera/utils/Ophir.py File Reference

Classes

• class nsCamera.utils.Ophir.Ophir

Namespaces

· nsCamera.utils.Ophir

9.22 nsCamera/utils/Packet.py File Reference

Classes

· class nsCamera.utils.Packet.Packet

Namespaces

· nsCamera.utils.Packet

9.23 nsCamera/utils/Subregister.py File Reference

Classes

· class nsCamera.utils.Subregister.SubRegister

Namespaces

• nsCamera.utils.Subregister

252 File Documentation

Index

all	nsCamera.utils.Packet.Packet, 225
nsCamera, 13	_read_timeout
nsCamera.boards, 14	nsCamera.comms.RS422.RS422, 236
nsCamera.comms, 15	_seqID
nsCamera.sensors, 16	nsCamera.utils.Packet.Packet, 225
nsCamera.utils, 17	_ser
init	nsCamera.comms.RS422.RS422, 236
nsCamera.boards.LLNL_v1.llnl_v1, 173	_stop
nsCamera.boards.LLNL_v4.llnl_v4, 194	nsCamera.comms.RS422.RS422, 236
nsCamera.CameraAssembler.CameraAssembler, 44	_type
nsCamera.comms.GigE.GigE, 126	nsCamera.utils.Packet.Packet, 226
nsCamera.comms.RS422.RS422, 227	_write_timeout
nsCamera.sensors.daedalus.daedalus, 103	nsCamera.comms.RS422.RS422, 236
nsCamera.sensors.icarus.icarus, 140	_zest
nsCamera.sensors.icarus2.icarus2, 157	nsCamera.comms.GigE.GigE, 135
nsCamera.utils.GenTec.GenTec, 123	
nsCamera.utils.Ophir.Ophir, 215	abort
nsCamera.utils.Packet.Packet, 219	nsCamera.CameraAssembler.CameraAssembler, 93
nsCamera.utils.Subregister.SubRegister, 239	abortReadoff
addr	nsCamera.CameraAssembler.CameraAssembler, 46
nsCamera.utils.Packet.Packet, 224	action
baud	nsCamera.utils.FlatField, 22
nsCamera.comms.RS422.RS422, 235	testSuite, 36
_cmd	ADC5_bipolar
nsCamera.utils.Packet.Packet, 224	nsCamera.boards.LLNL_v1.llnl_v1, 190
_crc	nsCamera.boards.LLNL_v4.llnl_v4, 211
nsCamera.utils.Packet, Packet, 225	ADC5_mult
_crc16	nsCamera.boards.LLNL_v1.llnl_v1, 190
nsCamera.utils.crc16pure, 17	nsCamera.boards.LLNL_v4.llnl_v4, 211
_data	addr
nsCamera.utils.Packet.Packet, 225	nsCamera.utils.Subregister.SubRegister, 240
_datatimeout	args
nsCamera.comms.RS422.RS422, 235	nsCamera.utils.FlatField, 22
fields	testSuite, 36
nsCamera.comms.GigE.GigE.ZESTETM1 CARD INF	arm O. naCamara Camara Assamblar Camara Assamblar 47
243	nsoamera.oameranssembler.oameranssembler, 47
_par	nsCamera.comms.GigE.GigE, 128
nsCamera.comms.RS422.RS422, 236	nsCamera.comms.RS422.RS422, 229
_payload	armed
nsCamera.utils.Packet.Packet, 225	nsCamera.CameraAssembler.CameraAssembler, 93
_payload_length	batchAcquire
nsCamera.utils.Packet.Packet, 225	nsCamera.CameraAssembler.CameraAssembler, 47
_port	BOARD
nsCamera.comms.RS422.RS422, 236	nsCameraExample, 25
nreamble	hoard

nsCamera.CameraAssembler.CameraAssembler, 93 testSuite, 36	$ns Camera. Camera Assembler. Camera Assembler, {\bf 51} \\ close card$
boardname	nsCamera.comms.GigE.GigE, 135 closeDevice
nsCamera.CameraAssembler.CameraAssembler, 94 bytes2str	nsCamera.CameraAssembler.CameraAssembler, 51
nsCamera.CameraAssembler.CameraAssembler, 48	nsCamera.comms.GigE.GigE, 129
nsCamera.utils.Packet.Packet, 220	nsCamera.comms.RS422.RS422, 230
bytesperpixel	nsCamera.utils.GenTec.GenTec, 123
nsCamera.sensors.daedalus.daedalus, 118	closeOphir
nsCamera.sensors.icarus.icarus, 152	nsCamera.utils.Ophir.Ophir, 216
nsCamera.sensors.icarus2.icarus2, 168	COM
modification of the contract o	nsCamera.utils.Ophir.Ophir, 217
ca	COMM
nsCamera.boards.LLNL_v1.llnl_v1, 190	nsCameraExample, 26
nsCamera.boards.LLNL_v4.llnl_v4, 211	comm
nsCamera.comms.GigE.GigE, 135	testSuite, 36
nsCamera.comms.RS422.RS422, 237	commname
nsCamera.sensors.daedalus.daedalus, 118	nsCamera.CameraAssembler.CameraAssembler, 94
nsCamera.sensors.icarus.icarus, 152	comms
nsCamera.sensors.icarus2.icarus2, 168	nsCamera.CameraAssembler.CameraAssembler, 94
nsCameraExample, 25	configADCs
calculateCRC	nsCamera.boards.LLNL_v1.llnl_v1, 177
nsCamera.utils.Packet.Packet, 220	nsCamera.boards.LLNL_v4.llnl_v4, 198
CardInfo	nsCamera.CameraAssembler.CameraAssembler, 51
nsCamera.comms.GigE.GigE, 135	Connection
CardInfoP	nsCamera.comms.GigE.GigE, 136
nsCamera.comms.GigE.GigE, 135	CRC16_XMODEM_TABLE
checkCRC	nsCamera.utils.crc16pure, 18
nsCamera.CameraAssembler.CameraAssembler, 49	crc16xmodem
nsCamera.utils.Packet.Packet, 221	nsCamera.utils.crc16pure, 18
checkReadPacket	currtime
nsCamera.utils.Packet.Packet, 221	nsCamera.CameraAssembler.CameraAssembler, 94
checkRegSet	cycle
nsCamera.CameraAssembler.CameraAssembler, 49	nsCamera.CameraAssembler.CameraAssembler, 94
checkResponsePacket	
nsCamera.utils.Packet.Packet, 222	default
checkResponseString	nsCamera.utils.FlatField, 23
nsCamera.utils.Packet.Packet, 222	testSuite, 37
checkSensorVoltStat	defoff
nsCamera.CameraAssembler.CameraAssembler, 50	nsCamera.boards.LLNL_v4.llnl_v4, 212
nsCamera.sensors.daedalus.daedalus, 105	defsens
nsCamera.sensors.icarus.icarus, 141	nsCamera.boards.LLNL_v4.llnl_v4, 212
nsCamera.sensors.icarus2.icarus2, 158	deInterlace
checkStatus	nsCamera.CameraAssembler.CameraAssembler, 52
nsCamera.boards.LLNL_v1.llnl_v1, 175	dest
nsCamera.boards.LLNL_v4.llnl_v4, 196	nsCamera.utils.FlatField, 23
nsCamera.CameraAssembler.CameraAssembler, 50	testSuite, 37
checkStatus2	DevHan
nsCamera.boards.LLNL_v1.llnl_v1, 176	nsCamera.utils.Ophir.Ophir, 217
nsCamera.boards.LLNL_v4.llnl_v4, 197	DHandle
nsCamera.CameraAssembler.CameraAssembler, 51	nsCamera.utils.Ophir.Ophir, 218
clearStatus	directory
nsCamera.boards.LLNL_v1.llnl_v1, 176	nsCamera.utils.FlatField, 23
nsCamera.boards.LLNL_v4.llnl_v4, 197	disarm

nsCamera.boards.LLNL_v1.llnl_v1, 177 nsCamera.boards.LLNL_v4.llnl_v4, 198	nsCamera.utils.FlatField, 23
nsCamera.CameraAssembler.CameraAssembler, 52	generateFF
dport	nsCamera.utils.FlatField, 19
nsCamera.comms.GigE.GigE, 136	generateFrames
dummyCheck	nsCamera.CameraAssembler.CameraAssembler, 56
nsCamera.CameraAssembler.CameraAssembler, 53	GenTecReadTest
dummySensorVals	nsCamera.utils.GenTec.GenTec, 123
nsCamera.boards.LLNL_v1.llnl_v1, 191	getBoardInfo
nsCamera.boards.LLNL_v4.llnl_v4, 212	nsCamera.CameraAssembler.CameraAssembler, 57
dumpNumpy	getCardInfo
nsCamera.CameraAssembler.CameraAssembler, 53	nsCamera.comms.GigE.GigE, 129
dumpRegisters	getCardIP
nsCamera.CameraAssembler.CameraAssembler, 54	nsCamera.comms.GigE.GigE, 130
dumpStatus	getEnter
nsCamera.boards.LLNL_v1.llnl_v1, 178	nsCamera.CameraAssembler.CameraAssembler, 58
nsCamera.boards.LLNL_v4.llnl_v4, 199	getFilenames
nsCamera.CameraAssembler.CameraAssembler, 55	nsCamera.utils.FlatField, 20
dumpSubregisters	getManualTiming
nsCamera.CameraAssembler.CameraAssembler, 55	nsCamera.CameraAssembler.CameraAssembler, 59
anablal ED	nsCamera.sensors.daedalus.daedalus, 106
enableLED	nsCamera.sensors.icarus.icarus, 141
nsCamera.boards.LLNL_v1.llnl_v1, 179	nsCamera.sensors.icarus2.icarus2, 158
nsCamera.boards.LLNL_v4.llnl_v4, 200 nsCamera.CameraAssembler.CameraAssembler, 56	getMonV
risGamera.GameraAssembler.GameraAssembler, 56	nsCamera.CameraAssembler.CameraAssembler, 59
firstframe	getPot
nsCamera.sensors.daedalus.daedalus, 118	nsCamera.CameraAssembler.CameraAssembler, 60
nsCamera.sensors.icarus.icarus, 152	getPotV
nsCamera.sensors.icarus2.icarus2, 168	nsCamera.CameraAssembler.CameraAssembler, 61
firstrow	getPressure
nsCamera.sensors.daedalus.daedalus, 119	nsCamera.boards.LLNL_v1.llnl_v1, 180
nsCamera.sensors.icarus.icarus, 152	nsCamera.boards.LLNL_v4.llnl_v4, 201
nsCamera.sensors.icarus2.icarus2, 168	nsCamera.CameraAssembler.CameraAssembler, 62
flatten	getRegister
nsCamera.CameraAssembler.CameraAssembler, 56	nsCamera.CameraAssembler.CameraAssembler, 62
FPGAboardtype	getROlvector
nsCamera.CameraAssembler.CameraAssembler, 94	nsCamera.utils.FlatField, 21
FPGAinterfaces	getSubregister
nsCamera.CameraAssembler.CameraAssembler, 95	nsCamera.CameraAssembler.CameraAssembler, 63 getTemp
FPGAinvalid	nsCamera.boards.LLNL_v1.llnl_v1, 180
nsCamera.CameraAssembler.CameraAssembler, 95	nsCamera.boards.LLNL_v4.llnl_v4, 202
FPGANum nsCamera.CameraAssembler.CameraAssembler, 95	nsCamera.CameraAssembler.CameraAssembler, 63
fpganumID	getTimer
nsCamera.sensors.daedalus.daedalus, 119	nsCamera.boards.LLNL_v1.llnl_v1, 181
nsCamera.sensors.icarus.icarus, 152	nsCamera.boards.LLNL_v4.llnl_v4, 202
nsCamera.sensors.icarus2.icarus2, 169	nsCamera.CameraAssembler.CameraAssembler, 64
FPGArad	getTiming
nsCamera.CameraAssembler.CameraAssembler, 95	nsCamera.CameraAssembler.CameraAssembler, 64
FPGAsensor	nsCamera.sensors.daedalus.daedalus, 106
nsCamera.CameraAssembler.CameraAssembler, 95	nsCamera.sensors.icarus.icarus, 142
FPGAVersion	nsCamera.sensors.icarus2.icarus2, 159
nsCamera.CameraAssembler.CameraAssembler, 95	gt
framelist	nsCamera.utils.GenTec, 24

height	nsCamera.CameraAssembler.CameraAssembler, 67
nsCamera.sensors.daedalus.daedalus, 119	logcrit
nsCamera.sensors.icarus.icarus, 152	nsCamera.boards.LLNL_v1.llnl_v1, 191
nsCamera.sensors.icarus2.icarus2, 169	nsCamera.boards.LLNL_v4.llnl_v4, 212
help	nsCamera.CameraAssembler.CameraAssembler, 96
nsCamera.utils.FlatField, 23	nsCamera.comms.GigE.GigE, 136
testSuite, 37	nsCamera.comms.RS422.RS422, 237
HFW	nsCamera.sensors.daedalus.daedalus, 120
nsCamera.sensors.daedalus.daedalus, 119	nsCamera.sensors.icarus.icarus, 153
icaructura	nsCamera.sensors.icarus2.icarus2, 170
icarustype	logcritbase
nsCamera.sensors.icarus.icarus, 153	nsCamera.CameraAssembler.CameraAssembler, 96
nsCamera.sensors.icarus2.icarus2, 169 initBoard	logdebug
	nsCamera.boards.LLNL_v1.llnl_v1, 191
nsCamera.boards.LLNL_v1.llnl_v1, 181 nsCamera.boards.LLNL_v4.llnl_v4, 203	nsCamera.boards.LLNL_v4.llnl_v4, 212
nsCamera.CameraAssembler.CameraAssembler, 64	nsCamera.CameraAssembler.CameraAssembler, 96
initialize	nsCamera.comms.GigE.GigE, 136
nsCamera.CameraAssembler.CameraAssembler, 64	nsCamera.comms.RS422.RS422, 237
initPots	nsCamera.sensors.daedalus.daedalus, 120
nsCamera.boards.LLNL v1.llnl v1, 182	nsCamera.sensors.icarus.icarus, 153
nsCamera.boards.LLNL_v4.llnl_v4, 203	nsCamera.sensors.icarus2.icarus2, 170
nsCamera.CameraAssembler.CameraAssembler, 66	logdebugbase
initPowerCheck	nsCamera.CameraAssembler.CameraAssembler, 96
nsCamera.CameraAssembler.CameraAssembler, 66	logerr
initSensor	nsCamera.boards.LLNL_v1.llnl_v1, 191
nsCamera.boards.LLNL_v1.llnl_v1, 183	nsCamera.boards.LLNL_v4.llnl_v4, 212
nsCamera.boards.LLNL_v4.llnl_v4, 203	nsCamera.CameraAssembler.CameraAssembler, 97
nsCamera.CameraAssembler.CameraAssembler, 67	nsCamera.comms.GigE.GigE, 136
inittime	nsCamera.comms.RS422.RS422, 237
nsCamera.CameraAssembler.CameraAssembler, 96	nsCamera.sensors.daedalus.daedalus, 120
interactive	nsCamera.sensors.icarus.icarus, 154
testSuite, 37	nsCamera.sensors.icarus2.icarus2, 170
interlacing	logerrbase
nsCamera.sensors.daedalus.daedalus, 119	nsCamera.CameraAssembler.CameraAssembler, 97
nsCamera.sensors.icarus.icarus, 153	loginfo
nsCamera.sensors.icarus2.icarus2, 169	nsCamera.boards.LLNL_v1.llnl_v1, 191
ipAdd	nsCamera.boards.LLNL_v4.llnl_v4, 213
testSuite, 37	nsCamera.CameraAssembler.CameraAssembler, 97
iplist	nsCamera.comms.GigE.GigE, 136
nsCamera.CameraAssembler.CameraAssembler, 96	nsCamera.comms.RS422.RS422, 237
	nsCamera.sensors.daedalus.daedalus, 120
lastframe	nsCamera.sensors.icarus.icarus, 154
nsCamera.sensors.daedalus.daedalus, 119	nsCamera.sensors.icarus2.icarus2, 170
nsCamera.sensors.icarus.icarus, 153	loginfobase
nsCamera.sensors.icarus2.icarus2, 169	nsCamera.CameraAssembler.CameraAssembler, 97
lastrow	logtag
nsCamera.sensors.daedalus.daedalus, 120	nsCamera.CameraAssembler.CameraAssembler, 97
nsCamera.sensors.icarus.icarus, 153	logwarn
nsCamera.sensors.icarus2.icarus2, 169	nsCamera.boards.LLNL_v1.llnl_v1, 191 nsCamera.boards.LLNL_v4.llnl_v4, 213
latchPots	nsCamera.CameraAssembler.CameraAssembler, 97
nsCamera.boards.LLNL_v1.llnl_v1, 184 nsCamera.boards.LLNL_v4.llnl_v4, 204	nsCamera.comms.GigE.GigE, 137
nsCamera.CameraAssembler.CameraAssembler, 67	nsCamera.comms.RS422.RS422, 237
loadTextFrames	
IVAU IEXILITATIIES	nsCamera.sensors.daedalus.daedalus, 120

nsCamera.sensors.icarus.icarus, 154 nsCamera.sensors.icarus2.icarus2, 170	nsCamera.boards.LLNL_v1.llnl_v1, 172 init, 173
logwarnbase	ADC5_bipolar, 190
nsCamera.CameraAssembler.CameraAssembler, 98	_ ·
nscamera.cameraAssembler.cameraAssembler, 90	ADC5_mult, 190
max	ca, 190
nsCamera.utils.Subregister.SubRegister, 240	checkStatus, 175
max value	checkStatus2, 176
nsCamera.utils.Subregister.SubRegister, 241	clearStatus, 176
maxframe	configADCs, 177
	disarm, 177
nsCamera.sensors.daedalus.daedalus, 121	dummySensorVals, 191
nsCamera.sensors.icarus.icarus, 154	dumpStatus, 178
nsCamera.sensors.icarus2.icarus2, 170	enableLED, 179
maxheight	getPressure, 180
nsCamera.sensors.daedalus.daedalus, 121	getTemp, 180
nsCamera.sensors.icarus.icarus, 154	getTimer, 181
nsCamera.sensors.icarus2.icarus2, 171	initBoard, 181
maxV	initPots, 182
nsCamera.utils.Subregister.SubRegister, 241	initSensor, 183
maxwidth	latchPots, 184
nsCamera.sensors.daedalus.daedalus, 121	logcrit, 191
nsCamera.sensors.icarus.icarus, 154	logdebug, 191
nsCamera.sensors.icarus2.icarus2, 171	logerr, 191
min	loginfo, 191
nsCamera.utils.Subregister.SubRegister, 241	logwarn, 191
minframe	monitor_controls, 192
nsCamera.sensors.daedalus.daedalus, 121	readSRAM, 184
nsCamera.sensors.icarus.icarus, 155	registers, 192
nsCamera.sensors.icarus2.icarus2, 171	_
minV	reportStatus, 185
nsCamera.utils.Subregister.SubRegister, 241	reportStatus, 185
mmReadoff	resetTimer, 186
nsCamera.CameraAssembler.CameraAssembler, 68	rs422_baud, 192
mode	rs422_cmd_wait, 192
nsCamera.comms.GigE.GigE, 137	setLED, 186
nsCamera.comms.RS422.RS422, 238	setPowerSave, 187
monitor_controls	setPPER, 187
nsCamera.boards.LLNL_v1.llnl_v1, 192	softReboot, 188
nsCamera.boards.LLNL_v4.llnl_v4, 213	startCapture, 188
	subreg_aliases, 192
name	subregisters, 192
nsCamera.utils.Subregister.SubRegister, 241	subreglist, 193
nargs	VREF, 193
nsCamera.utils.FlatField, 23	waitForSRAM, 189
nframes	nsCamera.boards.LLNL_v4, 14
nsCamera.sensors.daedalus.daedalus, 121	nsCamera.boards.LLNL_v4.llnl_v4, 193
nsCamera.sensors.icarus.icarus, 155	init, 194
nsCamera.sensors.icarus2.icarus2, 171	ADC5_bipolar, 211
None	ADC5_mult, 211
testSuite, 37	ca, 211
nsCamera, 13	checkStatus, 196
all, 13	checkStatus2, 197
nsCamera.boards, 13	clearStatus, 197
all, 14	configADCs, 198
nsCamera.boards.LLNL_v1, 14	defoff, 212

defsens, 212	comms, 94
disarm, 198	configADCs, 51
dummySensorVals, 212	currtime, 94
dumpStatus, 199	cycle, 94
enableLED, 200	deInterlace, 52
getPressure, 201	disarm, 52
getTemp, 202	dummyCheck, 53
getTimer, 202	dumpNumpy, 53
initBoard, 203	dumpRegisters, 54
initPots, 203	dumpStatus, 55
initSensor, 203	dumpSubregisters, 55
latchPots, 204	enableLED, 56
logcrit, 212	flatten, 56
logdebug, 212	FPGAboardtype, 94
logerr, 212	FPGAinterfaces, 95
loginfo, 213	FPGAinvalid, 95
logwarn, 213	FPGANum, 95
monitor_controls, 213	FPGArad, 95
readSRAM, 205	FPGAsensor, 95
registers, 213	FPGAVersion, 95
reportEdgeDetects, 205	generateFrames, 56
reportStatus, 206	getBoardInfo, 57
resetTimer, 207	getEnter, 58
rs422_baud, 213	getManualTiming, 59
rs422_cmd_wait, 213	getMonV, 59
setLED, 207	getPot, 60
setPowerSave, 208	getPotV, 61
setPPER, 208	getPressure, 62
softReboot, 209	getRegister, 62
startCapture, 209	getSubregister, 63
subreg_aliases, 214	getTemp, 63
subregisters, 214	getTimer, 64
subreglist, 214	getTiming, 64
VREF, 214	initBoard, 64
waitForSRAM, 210	initialize, 64
nsCamera.CameraAssembler, 14	initPots, 66
nsCamera.CameraAssembler.CameraAssembler, 39	initPowerCheck, 66
init , 44	initSensor, 67
abort, 93	inittime, 96
abortReadoff, 46	iplist, 96
arm, 47	latchPots, 67
armed, 93	loadTextFrames, 67
batchAcquire, 47	logcrit, 96
board, 93	logcritbase, 96
boardname, 94	logdebug, 96
bytes2str, 48	logdebugbase, 96
checkCRC, 49	logerr, 97
checkRegSet, 49	logerrbase, 97
checkSensorVoltStat, 50	loginfo, 97
checkStatus, 50	loginfobase, 97
checkStatus2, 51	logtag, 97
clearStatus, 51	logwarn, 97
closeDevice, 51	logwarnbase, 98 mmReadoff, 68
commname, 94	minineadoll, 00

oldtime, 98	unstringed, 100
packageroot, 98	verblevel, 101
padToFull, 98	verbmap, 101
parsedtime, 98	verbose, 101
parseReadoff, 69	version, 101
payloaderror, 98	waited, 101
platform, 99	waitForSRAM, 92
plotFrames, 69	writeSerial, 93
port, 99	nsCamera.comms, 14
powerCheck, 70	all, 15
printBoardInfo, 71	nsCamera.comms.GigE, 15
PY3, 99	nsCamera.comms.GigE.GigE, 125
read, 99	init, 126
readImgs, 72	_zest, 135
readoff, 72	arm, 128
readSerial, 72	ca, 135
readSRAM, 73	CardInfo, 135
reboot, 73	CardInfoP, 135
reinitialize, 73	closecard, 135
reportEdgeDetects, 74	closeDevice, 129
reportStatus, 74	Connection, 136
resetTimer, 74	dport, 136
resolveSubreg, 75	getCardInfo, 129
saveFrames, 75	getCardIP, 130
saveNumpys, 76	logcrit, 136
saveTiffs, 78	logdebug, 136
savetime, 99	logerr, 136
sendCMD, 79	loginfo, 136
sensmanual, 99	logwarn, 137
sensor, 100	mode, 137
sensorname, 100	openDevice, 130
sensorSpecific, 79	payloadsize, 137
senstiming, 100	readoff, 131
setArbTiming, 79	readSerial, 132
setFrames, 80	readTimeout, 137
setHighFullWell, 81	sendCMD, 133
setInterlacing, 81	skipError, 137
setLED, 81	writeSerial, 134
setManualShutters, 82	writeTimeout, 137
setPot, 82	ZCountCards, 138
setPotV, 83	ZOpenConnection, 138
setPowerSave, 86	ZReadData, 138
setPPER, 86	ZWriteData, 138
setRegister, 87	nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFO,
setRows, 87	243
setSubregister, 89	_fields_, 243
setTiming, 90	ubyte4, 243
setTriggerDelay, 90	ubyte4, 243 ubyte6, 244
setZeroDeadTime, 90	nsCamera.comms.RS422, 15
startCapture, 90	nsCamera.comms.RS422, 15
startCapture, 90 str2bytes, 91	init, 227
-	
str2nparray, 91	_baud, 235
submitMessages, 92	_datatimeout, 235
trigtime, 100	_par, 236

_port, 236	sensorSpecific, 109
_read_timeout, 236	setArbTiming, 111
_ser, 236	setHighFullWell, 112
_stop, 236	setInterlacing, 113
_write_timeout, 236	setManualShutters, 114
arm, 229	setTiming, 114
ca, 237	setTriggerDelay, 116
closeDevice, 230	setZeroDeadTime, 117
logcrit, 237	width, 122
logdebug, 237	ZDT, 122
logerr, 237	nsCamera.sensors.icarus, 16
loginfo, 237	nsCamera.sensors.icarus.icarus, 139
logwarn, 237	init, 140
mode, 238	bytesperpixel, 152
payloadsize, 238	ca, 152
PY3, 238	checkSensorVoltStat, 141
readoff, 230	firstframe, 152
readSerial, 231	firstrow, 152
sendCMD, 232	fpganumID, 152
serialClose, 234	getManualTiming, 141
skipError, 238	getTiming, 142
writeSerial, 235	height, 152
nsCamera.sensors, 15	icarustype, 153
all, 16	interlacing, 153
nsCamera.sensors.daedalus, 16	lastframe, 153
nsCamera.sensors.daedalus.daedalus, 102	lastrow, 153
init, 103	logcrit, 153
bytesperpixel, 118	logdebug, 153
ca, 118	logerr, 154
checkSensorVoltStat, 105	loginfo, 154
firstframe, 118	logwarn, 154
firstrow, 119	maxframe, 154
fpganumID, 119	maxheight, 154
getManualTiming, 106	maxwidth, 154
getTiming, 106	minframe, 155
height, 119	nframes, 155
HFW, 119	parseReadoff, 143
interlacing, 119	reportStatusSensor, 144
lastframe, 119	sens_registers, 155
lastrow, 120	sens_subregisters, 155
logcrit, 120	sensorSpecific, 144
logdebug, 120	setArbTiming, 145
logerr, 120	setHighFullWell, 146
loginfo, 120	setInterlacing, 147
logwarn, 120	setManualShutters, 147
maxframe, 121	setTiming, 149
maxheight, 121	setTriggerDelay, 151
maxwidth, 121	setZeroDeadTime, 151
minframe, 121	width, 155
nframes, 121	nsCamera.sensors.icarus2, 16
parseReadoff, 108	nsCamera.sensors.icarus2.icarus2, 156
reportStatusSensor, 109	init, 157
sens_registers, 121	bytesperpixel, 168
sens_subregisters, 122	ca, 168

checkSensorVoltStat, 158	removeFFall, 21
firstframe, 168	tslopes, 22
firstrow, 168	nsCamera.utils.GenTec, 24
fpganumID, 169	gt, 24
getManualTiming, 158	nsCamera.utils.GenTec.GenTec, 122
getTiming, 159	init, 123
height, 169	closeDevice, 123
icarustype, 169	GenTecReadTest, 123
interlacing, 169	ready, 124
lastframe, 169	sendSerial, 124
lastrow, 169	serial, 125
logcrit, 170	nsCamera.utils.Ophir, 24
logdebug, 170	nsCamera.utils.Ophir.Ophir, 215
logerr, 170	init, 215
loginfo, 170	closeOphir, 216
logwarn, 170	COM, 217
maxframe, 170	DevHan, 217
maxheight, 171	DHandle, 218
maxwidth, 171	OphirTest, 216
minframe, 171	nsCamera.utils.Packet, 25
nframes, 171	nsCamera.utils.Packet.Packet, 218
parseReadoff, 160	init, 219
reportStatusSensor, 161	_addr, 224
sens_registers, 171	_cmd, 224
sens_subregisters, 171	_crc, 225
sensorSpecific, 161	_data, 225
setArbTiming, 162	_payload, 225
setHighFullWell, 163	_payload_length, 225
setInterlacing, 163	_preamble, 225
setManualShutters, 164	_seqID, 225
setTiming, 165	_type, 226
setTriggerDelay, 167	bytes2str, 220
setZeroDeadTime, 167	calculateCRC, 220
width, 172	checkCRC, 221
nsCamera.utils, 16	checkReadPacket, 221
all, 17	checkResponsePacket, 222
nsCamera.utils.crc16pure, 17	checkResponseString, 222
_crc16, 17	pktStr, 223
CRC16_XMODEM_TABLE, 18	PY3, 226
crc16xmodem, 18	str2bytes, 224
nsCamera.utils.FlatField, 19	nsCamera.utils.Subregister, 25
action, 22	nsCamera.utils.Subregister.SubRegister, 239
args, 22	init , 239
default, 23	addr, 240
dest, 23	max, 240
directory, 23	max_value, 241
framelist, 23	maxV, 241
generateFF, 19	min, 241
getFilenames, 20	minV, 241
getROlvector, 21	name, 241
help, 23	register, 241
nargs, 23	resolution, 242
parser, 24	start_bit, 242
removeFF, 21	value, 242

width, 242	nsCamera.CameraAssembler.CameraAssembler, 98
writable, 242	payloadsize
nsCamera/initpy, 245	nsCamera.comms.GigE.GigE, 137
nsCamera/boards/initpy, 245	nsCamera.comms.RS422.RS422, 238
nsCamera/boards/LLNL_v1.py, 246	pktStr
nsCamera/boards/LLNL_v4.py, 246	nsCamera.utils.Packet.Packet, 223
nsCamera/CameraAssembler.py, 247	platform
nsCamera/CODE_OF_CONDUCT.md, 247	nsCamera.CameraAssembler.CameraAssembler, 99
nsCamera/comms/initpy, 245	plotFrames
nsCamera/comms/GigE.py, 247	nsCamera.CameraAssembler.CameraAssembler, 69
nsCamera/comms/RS422.py, 247	port
nsCamera/CONTRIBUTING.md, 248	nsCamera.CameraAssembler.CameraAssembler, 99
nsCamera/docs/nsCameraExample.py, 248	portNum
nsCamera/docs/testSuite.py, 248	testSuite, 38
nsCamera/sensors/initpy, 246	powerCheck
nsCamera/sensors/daedalus.py, 249	nsCamera.CameraAssembler.CameraAssembler, 70
nsCamera/sensors/icarus.py, 249	printBoardInfo
nsCamera/sensors/icarus2.py, 249	nsCamera.CameraAssembler.CameraAssembler, 71
nsCamera/utils/initpy, 246	PY3
nsCamera/utils/crc16pure.py, 249	nsCamera.CameraAssembler.CameraAssembler, 99
nsCamera/utils/FlatField.py, 250	nsCamera.comms.RS422.RS422, 238
nsCamera/utils/GenTec.py, 250	nsCamera.utils.Packet.Packet, 226
nsCamera/utils/Ophir.py, 251	
nsCamera/utils/Packet.py, 251	read
nsCamera/utils/Subregister.py, 251	nsCamera.CameraAssembler.CameraAssembler, 99
nsCameraExample, 25	readImgs
BOARD, 25	nsCamera.CameraAssembler.CameraAssembler, 72
ca, 25	readoff
COMM, 26	nsCamera.CameraAssembler.CameraAssembler, 72
SENSOR, 26	nsCamera.comms.GigE.GigE, 131
timing, 26	nsCamera.comms.RS422.RS422, 230
tune, 26	readSerial
oldtime	nsCamera.CameraAssembler.CameraAssembler, 72
nsCamera.CameraAssembler.CameraAssembler, 98	nsCamera.comms.GigE.GigE, 132
openDevice	nsCamera.comms.RS422.RS422, 231
nsCamera.comms.GigE.GigE, 130	readSRAM
OphirTest	nsCamera.boards.LLNL_v1.llnl_v1, 184
nsCamera.utils.Ophir.Ophir, 216	nsCamera.boards.LLNL_v4.llnl_v4, 205
,	nsCamera.CameraAssembler.CameraAssembler, 73
packageroot	readTimeout
nsCamera.CameraAssembler.CameraAssembler, 98	nsCamera.comms.GigE.GigE, 137
padToFull	ready
nsCamera.CameraAssembler.CameraAssembler, 98	nsCamera.utils.GenTec.GenTec, 124
parsedtime	reboot
nsCamera.CameraAssembler.CameraAssembler, 98	nsCamera.CameraAssembler.CameraAssembler, 73
parser	register
nsCamera.utils.FlatField, 24	nsCamera.utils.Subregister.SubRegister, 241
testSuite, 38	registers
parseReadoff	nsCamera.boards.LLNL_v1.llnl_v1, 192
nsCamera.CameraAssembler.CameraAssembler, 69	nsCamera.boards.LLNL_v4.llnl_v4, 213
nsCamera.sensors.daedalus.daedalus, 108	reinitialize
nsCamera.sensors.icarus.icarus, 143	nsCamera.CameraAssembler.CameraAssembler, 73
nsCamera.sensors.icarus2.icarus2, 160	removeFF
payloaderror	nsCamera.utils.FlatField, 21

removeFFall	nsCameraExample, 26
nsCamera.utils.FlatField, 21	sensor
reportEdgeDetects	nsCamera.CameraAssembler.CameraAssembler,
nsCamera.boards.LLNL_v1.llnl_v1, 185	100
nsCamera.boards.LLNL_v4.llnl_v4, 205	testSuite, 38
nsCamera.CameraAssembler.CameraAssembler, 74	sensorname
reportStatus	nsCamera.CameraAssembler.CameraAssembler,
nsCamera.boards.LLNL_v1.llnl_v1, 185	100
nsCamera.boards.LLNL_v4.llnl_v4, 206	sensorSpecific
nsCamera.CameraAssembler.CameraAssembler, 74	nsCamera.CameraAssembler.CameraAssembler, 79
reportStatusSensor	nsCamera.sensors.daedalus.daedalus, 109
nsCamera.sensors.daedalus.daedalus, 109	nsCamera.sensors.icarus.icarus, 144
nsCamera.sensors.icarus.icarus, 144	nsCamera.sensors.icarus2.icarus2, 161
nsCamera.sensors.icarus2.icarus2, 161	senstiming
resetTimer	nsCamera.CameraAssembler.CameraAssembler,
nsCamera.boards.LLNL_v1.llnl_v1, 186	100
nsCamera.boards.LLNL_v4.llnl_v4, 207	serial
nsCamera.CameraAssembler.CameraAssembler, 74	nsCamera.utils.GenTec.GenTec, 125
resolution	serialClose
nsCamera.utils.Subregister.SubRegister, 242	nsCamera.comms.RS422.RS422, 234
resolveSubreg	setArbTiming
nsCamera.CameraAssembler.CameraAssembler, 75	nsCamera.CameraAssembler.CameraAssembler, 79
rs422_baud	nsCamera.sensors.daedalus.daedalus, 111
nsCamera.boards.LLNL_v1.llnl_v1, 192	nsCamera.sensors.icarus.icarus, 145
nsCamera.boards.LLNL_v4.llnl_v4, 213	nsCamera.sensors.icarus2.icarus2, 162
rs422_cmd_wait	setFrames
nsCamera.boards.LLNL_v1.llnl_v1, 192	nsCamera.CameraAssembler.CameraAssembler, 80
nsCamera.boards.LLNL_v4.llnl_v4, 213	setHighFullWell
	nsCamera.CameraAssembler.CameraAssembler, 81
saveFrames	nsCamera.sensors.daedalus.daedalus, 112
nsCamera.CameraAssembler.CameraAssembler, 75	nsCamera.sensors.icarus.icarus, 146
saveNumpys	nsCamera.sensors.icarus2.icarus2, 163
nsCamera.CameraAssembler.CameraAssembler, 76	setInterlacing
saveTiffs	nsCamera.CameraAssembler.CameraAssembler, 81
nsCamera.CameraAssembler.CameraAssembler, 78	nsCamera.sensors.daedalus.daedalus, 113
savetime	nsCamera.sensors.icarus.icarus, 147
nsCamera.CameraAssembler.CameraAssembler, 99	nsCamera.sensors.icarus2.icarus2, 163
sendCMD	setLED
nsCamera.CameraAssembler.CameraAssembler, 79	nsCamera.boards.LLNL_v1.llnl_v1, 186
nsCamera.comms.GigE.GigE, 133	nsCamera.boards.LLNL_v4.llnl_v4, 207
nsCamera.comms.RS422.RS422, 232	nsCamera.CameraAssembler.CameraAssembler, 81
sendSerial	setManualShutters
nsCamera.utils.GenTec.GenTec, 124	nsCamera.CameraAssembler.CameraAssembler, 82
sens_registers	nsCamera.sensors.daedalus.daedalus, 114
nsCamera.sensors.daedalus.daedalus, 121	nsCamera.sensors.icarus.icarus, 147
nsCamera.sensors.icarus.icarus, 155	nsCamera.sensors.icarus2.icarus2, 164
nsCamera.sensors.icarus2.icarus2, 171	setPot
sens_subregisters	nsCamera.CameraAssembler.CameraAssembler, 82
nsCamera.sensors.daedalus.daedalus, 122	setPotV
nsCamera.sensors.icarus.icarus, 155	nsCamera.CameraAssembler.CameraAssembler, 83
nsCamera.sensors.icarus2.icarus2, 171	setPowerSave
sensmanual	nsCamera.boards.LLNL_v1.llnl_v1, 187
nsCamera.CameraAssembler.CameraAssembler, 99	nsCamera.boards.LLNL_v4.llnl_v4, 208
SENSOR	nsCamera.CameraAssembler.CameraAssembler, 86

setPPER	testSuite, 38
nsCamera.boards.LLNL_v1.llnl_v1, 187	
nsCamera.boards.LLNL_v4.llnl_v4, 208	testSuite, 27
nsCamera.CameraAssembler.CameraAssembler, 86	action, 36
setRegister	args, 36
nsCamera.CameraAssembler.CameraAssembler, 87	board, 36
setRows	comm, 36
nsCamera.CameraAssembler.CameraAssembler, 87	default, 37
setSubregister	dest, 37
nsCamera.CameraAssembler.CameraAssembler, 89	help, 37
setTiming	interactive, 37
nsCamera.CameraAssembler.CameraAssembler, 90	ipAdd, 37
nsCamera.sensors.daedalus.daedalus, 114	None, 37
nsCamera.sensors.icarus.icarus, 149	parser, 38
nsCamera.sensors.icarus2.icarus2, 165	portNum, 38
setTriggerDelay	sensor, 38
nsCamera.CameraAssembler.CameraAssembler, 90	swtrigger, 38
nsCamera.sensors.daedalus.daedalus, 116	testSuite, 27
nsCamera.sensors.icarus.icarus, 151	timing
nsCamera.sensors.icarus2.icarus2, 167	nsCameraExample, 26
setZeroDeadTime	trigtime
nsCamera.CameraAssembler.CameraAssembler, 90	nsCamera.CameraAssembler.CameraAssembler,
nsCamera.sensors.daedalus.daedalus, 117	100
nsCamera.sensors.icarus.icarus, 151	tslopes
nsCamera.sensors.icarus2.icarus2, 167	nsCamera.utils.FlatField, 22
skipError	tune
nsCamera.comms.GigE.GigE, 137	nsCameraExample, 26
nsCamera.comms.RS422.RS422, 238	
softReboot	ubyte4
nsCamera.boards.LLNL_v1.llnl_v1, 188	nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFC
nsCamera.boards.LLNL_v4.llnl_v4, 209	243
start_bit	ubyte6
nsCamera.utils.Subregister.SubRegister, 242	nsCamera.comms.GigE.GigE.ZESTETM1_CARD_INFC
startCapture	244
nsCamera.boards.LLNL_v1.llnl_v1, 188	unstringed
nsCamera.boards.LLNL v4.llnl v4, 209	nsCamera.CameraAssembler.CameraAssembler,
nsCamera.CameraAssembler.CameraAssembler, 90	100
str2bytes	l
nsCamera.CameraAssembler.CameraAssembler, 91	value
nsCamera.utils.Packet.Packet, 224	nsCamera.utils.Subregister.SubRegister, 242
	verblevel
str2nparray nsCamera.CameraAssembler.CameraAssembler, 91	nsCamera.CameraAssembler.CameraAssembler,
submitMessages	101
nsCamera.CameraAssembler.CameraAssembler, 92	verbmap
	nsCamera.CameraAssembler.CameraAssembler,
subreg_aliases nsCamera.boards.LLNL v1.llnl v1, 192	101
:	verbose
nsCamera.boards.LLNL_v4.llnl_v4, 214	nsCamera.CameraAssembler.CameraAssembler,
subregisters	101
nsCamera.boards.LLNL_v1.llnl_v1, 192	version
nsCamera.boards.LLNL_v4.llnl_v4, 214	nsCamera.CameraAssembler.CameraAssembler,
subreglist	101
nsCamera.boards.LLNL_v1.llnl_v1, 193	VREF
nsCamera.boards.LLNL_v4.llnl_v4, 214	nsCamera.boards.LLNL_v1.llnl_v1, 193
swtrigger	nsCamera.boards.LLNL_v4.llnl_v4, 214

```
waited
    nsCamera.CameraAssembler.CameraAssembler,
         101
waitForSRAM
    nsCamera.boards.LLNL_v1.llnl_v1, 189
    nsCamera.boards.LLNL_v4.llnl_v4, 210
    nsCamera.CameraAssembler.CameraAssembler, 92
width
    nsCamera.sensors.daedalus.daedalus, 122
    nsCamera.sensors.icarus.icarus, 155
    nsCamera.sensors.icarus2.icarus2, 172
    nsCamera.utils.Subregister.SubRegister, 242
writable
    nsCamera.utils.Subregister.SubRegister, 242
writeSerial
    nsCamera.CameraAssembler.CameraAssembler, 93
    nsCamera.comms.GigE.GigE, 134
    nsCamera.comms.RS422.RS422, 235
writeTimeout
    nsCamera.comms.GigE.GigE, 137
ZCountCards
    nsCamera.comms.GigE.GigE, 138
ZDT
    nsCamera.sensors.daedalus.daedalus, 122
ZOpenConnection
    nsCamera.comms.GigE.GigE, 138
ZReadData
    nsCamera.comms.GigE.GigE, 138
ZWriteData
    nsCamera.comms.GigE.GigE, 138
```