ADIS1700x Vision-Sensing Camera Module User Guide

(includes ADIS17001 and ADIS17002)

Revision 1.0, October 2017

Part Number
82-100136-01

Analog Devices, Inc.
One Technology Way
Norwood, MA 02062-9106

ANALOG DEVICES

Copyright Information

Disclaimer

Analog Devices, Inc. reserves the right to change this product without prior notice. Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use; nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under the patent rights of Analog Devices, Inc.

Trademark and Service Mark Notice

The Analog Devices logo, Blackfin, Blackfin+, SHARC, SHARC+, CrossCore, VisualDSP++, EZ-KIT, EZ-Extender, and EngineerZone are registered trademarks of Analog Devices, Inc.

All other brand and product names are trademarks or service marks of their respective owners.

Contents

ADIS1700x Library

Communication Protocol

1 Preface

Thank you for purchasing the Analog Devices, Inc. ADIS1700x Vision-Sensing Camera Module development kit.

The ADIS1700x is a low-power vision-sensing camera module in a small form factor for interfacing to USB 2.0 HOST-compliant devices including single board computers. It includes a tri-axial accelerometer for image stabilization, tilt and impact detection. The ADIS1700x combines industry-leading logarithmic sensitive video imaging technology with digital signal processing and an accelerometer that optimizes video performance.

The ADSP-BF707 processor is a member of the Blackfin® family of products. The Blackfin+ processor combines a dual-MAC 16-bit state-of-the-art signal processing engine, the advantages of a clean, orthogonal RISC-like microprocessor instruction set, and single-instruction, multiple-data (SIMD) multimedia capabilities into a single instruction-set architecture. New enhancements to the Blackfin+® core add 32-bit MAC and 16-bit complex MAC support, cache enhancements, branch prediction and other instruction set improvements—all while maintaining instruction set compatibility to previous Blackfin products.

The ADIS1700x Vision-Sensing Camera Module development kit is shipped with all of the necessary hardware - you can start the evaluation immediately. The package contains the standalone evaluation board, an interposer board to enable connection through a standard USB connector, USB cable, a 67 ° FOV and 110° HFOV lenses, a single-user license of CrossCore Embedded Studio, and an ICE-1000 emulator for debugging.

Product Overview

The ADIS1700x features:

- Analog Devices ADSP-BF707 processor
- USB 2.0 compliant interface
- 10kV ESD interface
- Analog Devices ADVS200x low-power monochrome QVGA imager with logarithmic sensitivity

Purpose of This Manual

This manual provides instructions for installing hardware and software associated with the ADIS1700x Vision-Sensing Camera Module. It describes the SNAP Framework used on the Blackfin low-power imaging platform and demonstrates the use of the camera applications.

Intended Audience

The primary audience for this manual is a programmer who is familiar with the Analog Devices Blackfin+ ADSP-BF70x processor and ADVS200x imager.

For additional information about the Analog Devices camera module, see the *ADIS17001/ADIS17002 Low Power, Small Form Factor Vision-Sensing Camera Module Data Sheet*.

Other applicable documentation for the ADIS17001/ADIS17002 includes:

- *ADSP-BF70x Blackfin+ Processor Hardware Reference*
- *ADSP-BF70x Blackfin+ Processor Programming Reference*
- *ADVS200x Demo User Guide*

Technical Support

You can reach Analog Devices processors and Embedded Vision Sensing technical support in the following ways:

- Post your questions in the Embedded Vision Sensing support community at EngineerZone ®:

  http://ez.analog.com/community/embedded-vision-sensing

- Submit your questions to technical support directly at:

  http://www.analog.com/support

- E-mail your questions about processors, DSPs, and tools development software from ***CrossCore Embedded Studio***® or ***VisualDSP++***®.

  If using CrossCore Embedded Studio or VisualDSP++ choose ***Help > Email Support***. This creates an e-mail to processor.tools.support@analog.com and automatically attaches your CrossCore Embedded Studio or VisualDSP++ version information and `license.dat` file.

- E-mail your questions about processors and processor applications to:

  processor.tools.support@analog.com

  processor.china@analog.com

- Contact your Analog Devices sales office or authorized distributor. Locate one at:

  http://www.analog.com/adi-sales

- Send questions by mail to:

  Analog Devices, Inc.

  One Technology Way

  P.O. Box 9106

Norwood, MA 02062-9106

USA

Product Information

Product information can be obtained from the Analog Devices website.

Analog Devices Website

The Analog Devices website, http://www.analog.com, provides information about a broad range of products - analog integrated circuits, amplifiers, converters, and digital signal processors.

To access a complete technical library for each processor family, go to http://www.analog.com/processors/technical_library. The manuals selection opens a list of current manuals related to the product as well as a link to the previous revisions of the manuals. When locating your manual title, note a possible errata check mark next to the title that leads to the current correction report against the manual.

Also note, MyAnalog.com is a free feature of the Analog Devices website that allows customization of a web page to display only the latest information about products you are interested in. You can choose to receive weekly e-mail notifications containing updates to the web pages that meet your interests, including documentation errata against all manuals. MyAnalog.com provides access to books, application notes, data sheets, code examples, and more.

Visit MyAnalog.com to sign up. If you are a registered user, just log on. Your user name is your e-mail address.

EngineerZone

EngineerZone is a technical support forum from Analog Devices, Inc. It allows you direct access to ADI technical support engineers. You can search FAQs and technical information to get quick answers to your embedded processing and DSP design questions.

Use EngineerZone to connect with other DSP developers who face similar design challenges. You can also use this open forum to share knowledge and collaborate with the ADI support team and your peers. Visit http://ez.analog.com to sign up.

Notation Conventions

Text conventions used in this manual are identified and described as follows. Additional conventions, which apply only to specific chapters, may appear throughout this document.

Example	*Description*
File > ***Close***	Titles in bold style indicate the location of an item within the CrossCore Embedded Studio IDE's menu system (for example, the ***Close*** command appears on the ***File*** menu).

Example	*Description*
`{this | that}`	Alternative required items in syntax descriptions appear within curly brackets and separated by vertical bars; read the example as `this` or `that`. One or the other is required.
`[this | that]`	Optional items in syntax descriptions appear within brackets and separated by vertical bars; read the example as an optional `this` or `that`.
`[this, …]`	Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipsis; read the example as an optional comma-separated list of `this`.
`.SECTION`	Commands, directives, keywords, and feature names are in text with `letter gothic` font.
`filename`	Non-keyword placeholders appear in text with `letter gothic` font and italic style format.
NOTE:	**NOTE:** For correct operation, .. A note provides supplementary information on a related topic.
CAUTION:	**CAUTION:** Incorrect device operation may result if ... **CAUTION:** Device damage may result if ... A caution identifies conditions or inappropriate usage of the product that could lead to undesirable results or product damage.
ATTENTION:	**ATTENTION:** Injury to device users may result if ... A warning identifies conditions or inappropriate usage of the product that could lead to conditions that are potentially hazardous for devices users.

2 ADIS1700x Functional Description

This chapter describes how the ADIS1700x interfaces with functional blocks and the SNAP Framework.

SNAP Framework Overview

The SNAP Framework is a set of libraries that enable the user to quickly develop applications for the Blackfin processor. It contains functionality for configuring and accessing lower level drivers (such as SPI, EPPI, TWI, etc.) as well as provides functionality for higher level services. It also contains functionality to communicate with the host using a defined protocol. The source code is available upon request. It is targeted to be compiled and linked with the SNAP framework and to be run on the ADIS1700x board. Features include continuous image capture from ADVS200x sensor and configuration of ADVS200x sensor.

The ADVS200x camera sensor is configured to produce the monochrome video at 10 frames per second. Frame rate can be changed using the ADIS1700x SNAP sensor library and application. The video captured from ADVS200x camera sensor on the ADIS1700x platform can be analysed by the suitable application. The SNAP framework connects to the GUI running on the host PC through a control interface and protocol on USB. Its streams the captured video data real time on the USB interface.

System Overview

The SNAP Framework block diagram is shown in ***ADIS1700x Software System Block Diagram***. The software package includes the SNAP Framework (drivers, services) library built in C/C++ and assembly sources. Sample applications and associated documents are provided with the package.

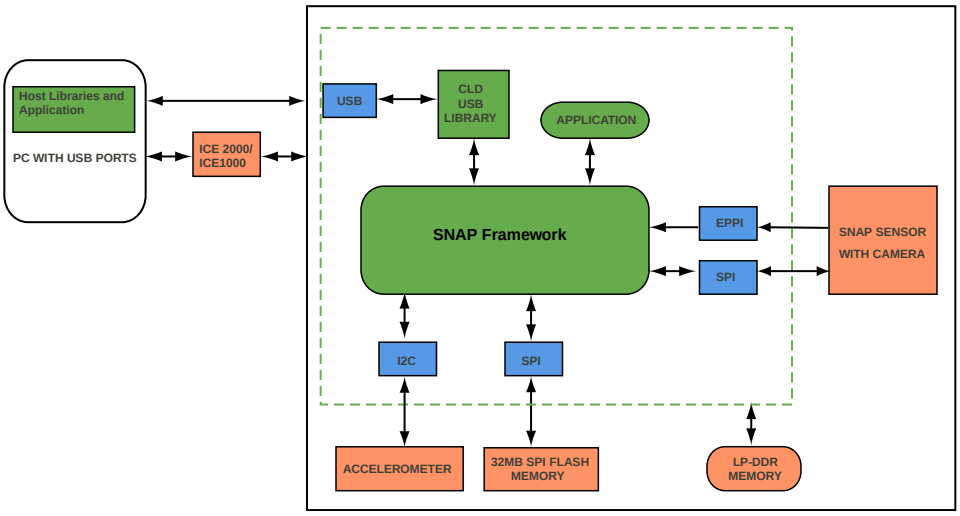


Figure 2-1: ADIS1700x Software System Block Diagram

Block Diagram

The ***ADIS1700x Block Diagram*** shows the functional blocks of the ADIS1700x Vision-Sensing Camera Module.

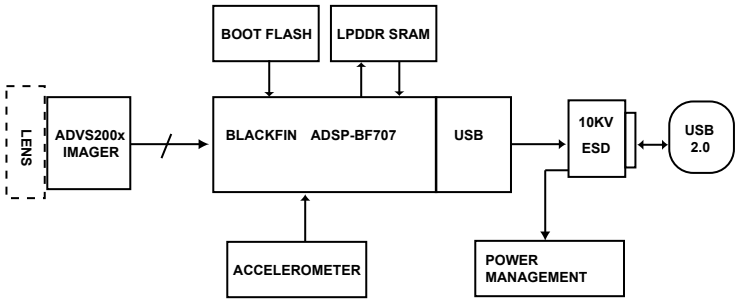


Figure 2-2: ADIS1700x Block Diagram

3 Using the ADIS1700x

This chapter provides information for getting starting with the ADIS1700x evaluation kit.

Evaluation System Contents

The ADIS1700x evaluation system consists of the following items:

- ADIS1700x Vision-Sensing Camera Module
- Micro USB Interposer board
- USB cable
- Demo applications for functional/performance evaluation. (Download from http://www.analog.com/adis17001).
- Vision Software Framework and host API libraries. (Download from http://www.analog.com/adis17001).
- ICE-1000 USB-based JTAG Emulator
- CrossCore Embedded Studio – Perpetual License (restricted use for ADIS1700x)
- 67° FOV and 110° HFOV Lenses

Contact the vendor where you purchased the evaluation board or contact Analog Devices, Inc if any item is missing.

Default Configuration

The ADIS1700x is designed to run as a standalone unit.

When removing the ADIS1700x board from the package, handle the board carefully to avoid the discharge of static electricity, which can damage some components.

CAUTION:  ***ESD (electrostatic discharge) sensitive device.*** Charged devices and circuit boards can discharge without detection. Although this product features patented or proprietary protection circuitry,

damage can occur on devices subjected to high energy ESD. Therefore, proper ESD precautions should be taken to avoid performance degradation or loss of functionality.

Board Installation

Complete the following steps to set up the hardware.

1. Connect the ADIS1700x board from the adapter shown in the *ADIS1700x Evaluation Board* to a USB port of a PC using the supplied USB cable.
2. Optionally, connect the ICE-1000 or ICE-2000 to the JTAG connection and the other end to a USB port of a PC. This step is required for debugging using CCES.

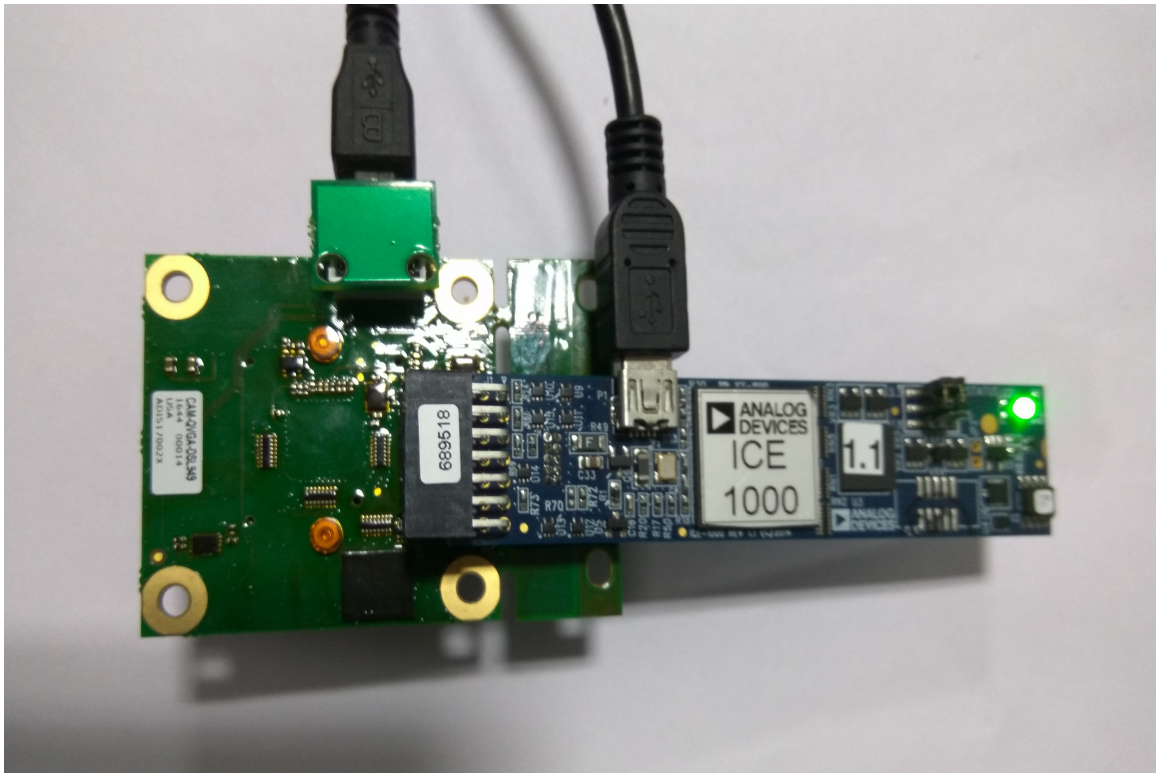


Figure 3-1: ADIS1700x Evaluation Board

4 Installation

This chapter describes how to the install the SNAP Sensor library and create a Win32 project.

Install the SNAP Sensor Library

Install the following components on the target PC:

- ***Visual C+ Redistributable for Visual Studio 2015***. (Download from https://www.microsoft.com.)
- ***Microsoft .NET framework 4.5.2.*** (Download from https://www.microsoft.com.)

Complete the following steps to install the `SnapSensorLibrary`.

1. Run the installer `SnapSensorLibrary_Demo.msi`.

   NOTE: If a previous version of the application already exists on the PC, it must be uninstalled first. See the Uninstall SNAP Sensor Library section.

2. Follow the instructions of the installer.

   The applications are installed under *Analog Devices >SnapSensorLibrary>*

Install the ADI WinUSB Driver

The ADI WinUSB driver must be installed so that the application can communicate with a device running the the SNAP Framework over the USB.

Complete the following steps to install the ADI WinUSB driver.

1. Connect the device to the PC through the USB port.

   Windows starts searching for the driver.

2. Stop the automatic search and select the option for manually locating the driver.
3. Point to the driver folder (***C:\Analog Devices\SnapSensorLibrary\Drivers\ADI WinUSB).***

Uninstall SNAP Sensor Library

Complete the following steps to uninstall the SNAP Sensor library application.

1. Run the Windows Control Panel.
2. Select *Uninstall or change a Program*.
3. Select the item (for example, *SnapSensorLibrary (Development)*) and click *Uninstall*
4. Follow the instructions of the uninstaller.

Create a Win32 Project

- Ensure that Visual Studio 2015 (or later) is installed.
- Ensure that the SnapSensorLibrary development edition is installed

Complete the following steps to create a Visual Studio project.

1. In Visual Studio create a new project (or add a new project to an existing solution) of type Visual C++ Win32 Console application.
2. Disable the precompiled header and ***Security Development Lifecycle (SDL)*** checks.
3. Click ***Finish***.
4. Delete the `stdafx.h` from the header files and `stdafx.cpp` from the source files.
5. Remove the default `.cpp` file created for the project.
6. Add the required `.cpp` files to the project being built. The files are available in ***Code Samples*** (***<<installed directory>>\SnapSensorLibrary\Lib\CodeSamples***). Create the corresponding console application `.exe` file (for example, `GetDeviceStatus.cpp`).
    a. Right click projects. Select *Add* > *Existing Item* and then select the required `.cpp` files from the code ***Samples***.
7. Add the include files
    a. Add the respective `.h` file (for example, `DeviceStatus.h`) to the ***Header Files*** option available in the project.
    b. Right click *Header Files*. Select *Add* > *Existing Item* and then select the required `.h` files from the code *Samples*.
8. Modify the C/C++ settings in the project properties. Copy and paste *$(SnapSensorLibrary_LibPath) Include* to *Additional Include Directories*.

9. Modify the project properties to link to the library files. In the Linker settings, copy and paste *$(SnapSensorLibrary_LibPath) Include* to *Additional Include Directories.* In the Linker settings, *Additional Dependencies* must have at least `SnapCoreLib.lib` and can also contain one or more of the following libraries depending upon the application to be built:
    a. `SnapSensorConnectionLib.lib` — if communication with a Blip-Mini or ADIS1700x is required .
    b. `SnapCoreImgProcLib.lib` — if image processing functionality is required
    c. `SnapADIS1700xConnectionLib.lib` — if communication with a ADIS1700x is required

NOTE: Use namespaces `SnapSensor` and `SnapSensor::SensorConnection` when communicating with a Blip-Mini or ADIS1700x

Build the project and confirm that the expected output appears on the console.

5 Demo Applications

The ADIS1700x Vision-Sensing Camera Module demo and development packages are supported by the following applications:

- ADVS200x Demo Application – an application that demonstrates the capability of the ADVS200x sensor. See the *ADVS200x Demo User Guide* for details.
- ADIS1700x Demo Application – an interface to interact with the ADIS1700x device.
- Device Re-programmer Application – an application that allows the user to reprogram a device through the USB.
- ImgSeqFile Viewer Application – an application that allows the user to open a video file (`ImgSeqFile`) and browse its contents. This application is available only with the development installer.
- NamedValueFile Viewer Application – application that allows the user to open a named value file (such as the parameters file) and browse its contents. This application is available only with the development installer.
- ADVS200x Parameters Editor Application – an application that allows the user to edit the ADVS200x configuration parameter and read and write the ADVS200x parameters to a file. This application is available only with the development installer.

SnapSensorLibrary

The `SnapSensorLibrary` enables the user to develop applications to communicate with the ADIS1700x unit over the USB from a Linux or Windows host. This includes functionality for updating firmware, creating and capturing video and downloading device info.

NOTE: The shortcuts to the installed console applications are available in the Windows Program Menu under *AnalogDevices* > *SnapSensorLibrary vx.x.x*.

Code Samples

The source code for all of the console applications is provided.

GetFirmwareVersions

`GetFirmwareVersions` is a console application that has the following features:

- Connects to a device running the SNAP Framework
- Downloads and displays the device bootloader firmware version
- Downloads and displays the device application firmware versions

GetDeviceInfo

`GetDeviceInfo` is a console application that has the following features:

- Connects to a device running the SNAP Framework
- Downloads and displays the device information

GetDeviceStatus

`GetDeviceStatus` is a console application that has the following features:

- Connects to a device running the SNAP Framework
- Downloads and displays the current mode of the firmware
- Downloads and displays the current version of the firmware
- Downloads and displays the device status

UpdateApplicationFirmware

`UpdateApplicationFirmware` is a console application that has the following features:

- Connects to a device running the SNAP Framework
- Reads a user-supplied application firmware file
- Updates the application firmware in the device and stores it to flash
- Downloads and displays the version of the application firmware

UpdateBootloaderFirmware

`UpdateBootloaderFirmware` is a console application that has the following features:

- Connects to a device running the SNAP Framework
- Reads a user-supplied bootloader firmware file
- Updates the bootloader firmware in the device and stores it to flash
- Downloads and displays the version of the bootloader firmware

CreateRestorePoint

`CreateRestorePoint` is a console application that has the following features:

- Connects to a device running the SNAP Framework
- Copies the current application firmware and data into the restore point section of the flash

RecoverRestorePoint

`RecoverRestorePoint` is a console application that has the following features:

- Connects to a device running the SNAP Framework
- Copies the current application firmware and data from the restore point section of the flash

ContinuousVideoRecorder

`ContinuousVideoRecorder` is a console application that has the following features:

- Connects to a device running the SNAP Framework
- Creates an image sequence (video) file with the user-supplied name
- Continuously captures image data from the ADVS200x sensor on the device
- Saves each captured frame into the image sequence file

RandomFrameVideoRecorder

`RandomFrameVideoRecorder` is a console application that has the following features:

- Connects to a device running the SNAP Framework
- Creates an image sequence (video) file with the user-supplied name
- Whenever the user hits a key, it captures an image from the ADVS200x sensor on the device and saves it to the image sequence file

CaptureLoop

`CaptureLoop` is a console application that has the following features:

- Connects to a device running the SNAP Framework
- Continuously downloads the image data from the ADVS200x sensor in a loop that runs in a separate thread

NOTE: This console application is only available with the development installer.

MultiSensorCaptureLoop

`MultiSensorCaptureLoop` is a console application that has the following features:

- Connects to up to three devices in parallel running the SNAP Framework
- Continuously downloads the image data from each of the ADVS200x sensors in loops that run in separate threads

NOTE: This console application is only available with the development installer.

UserCaptureLoop

`UserCaptureLoop` is a console application that has the following features:

- Connects to a device running the SNAP Framework
- Continuously downloads the image data from each of the ADVS200x sensors in a loop running in the main program thread

NOTE: This console application is only available with the development installer.

ADIS1700xAPIConsole

`ADIS1700xAPIConsole` is a console application that demonstrates the interface to an ADIS1700x device.

NOTE: This console application is only available with the development installer.

ADIS1700xGetConfig

`ADIS1700xGetConfig` is a console application that has the following features:

- Connects to a device running the ADIS1700x application firmware
- Downloads and displays the production parameters
- Downloads and displays the LED status
- Downloads and displays the IMU configuration
- Downloads and displays the ADC readings

NOTE: This console application is only available with the development installer.

ADIS1700xCaptureLoop

`ADIS1700xCaptureLoop` is a console application that has the following features:

- Connects to a device running the ADIS1700x application firmware
- Continuously downloads the image data and IMU data from the ADIS1700x in a loop that runs in a separate thread

NOTE: This console application is only available with the development installer.

ADIS1700xMultiSensorCaptureLoop

`ADIS1700xMultiSensorCaptureLoop` is a console application that has the following features:

- Connects to up to three devices in parallel running the ADIS1700x application firmware

- Continuously downloads the image data and IMU data from each of the ADIS1700x in loops that run in separate threads

NOTE: This console application is only available with the development installer.

ADIS1700xUserCaptureLoop

`ADIS1700xUserCaptureLoop` is a console application that has the following features:

- Connects to a device running the ADIS1700x application firmware
- Continuously downloads the image data and IMU data from the ADIS1700x in a loop running in the main program thread

NOTE: This console application is only available with the development installer.

SNAP Development Libraries

The SNAP development libraries are available for application development in the C++ (Windows or Linux) and in `.Net` (Windows). The following libraries are included:

- `SnapCoreLib`
- `SnapCoreImgProcLib`
- `SnapImageDisplayProcLib`
- `SnapSensorConnectionLib`
- `SnapADIS1700xConnectionLib`
- `Snap`
- `SnapADIS1700x`

SnapCoreLib

The `SnapCoreLib` library is used to manage memory, define images, strings and vectors, provide mathmatical operations and handle files. See the SnapSensorLibrary for details.

SnapCoreImgProcLib

The `SnapCoreImgProcLib` library is used to create image regions, calibrate the camera and process images. See the SnapSensorLibrary for details.

SnapImageDisplayLib

The `SnapImageDisplayLib` library is used to convert and save images.

The library includes the following functionality:

- Converts 10-bit images to 8-bit display images with or without image enhancement
- Saves images into bitmap files

SnapSensorConnectionLib

The `SnapSensorConnectionLib` library is used to connect and communicate with a device and read or write to the flash storage on the device.

The library includes the following functionality:

- Discovers and connects to devices that are running the SNAP Sensor Framework
- Communicates with the firmware running in a device that uses the SNAP Sensor Framework
- Communicate with the ADVS200x sensor service on the device to get images and adjust the ADVS200x parameters
- Reads from and writes to the flash storage on a device that uses the SNAP Sensor Framework

SnapADIS1700xConnectionLib

The `SnapADIS1700xConnectionLib` library is used to connect and communicate with a device.

The library includes the following functionality:

- Discovers and connects to devices that are running the SNAP Sensor Framework and ADIS1700x application firmware
- Communicates with the ADIS1700x firmware running in a device that uses the SNAP Sensor Framework
- Communicates with the IMU (accelerometer) in a device that uses the Snap Sensor Framework, allowing parameter adjustment and data reading

Snap

The `Snap` library provides various user controls and classes to build Windows UI applications.

SnapADIS1700x

The `SnapADIS1700x` library provides various user controls and classes to build Windows UI applications.

ADIS1700x Demo Application

The `ADIS1700x Demo` application is a UI application (part of the SNAP Framework) that provides a user-friendly interface to interact with the ADIS1700x device. The application has the following features:

- Configures the device, such as adjusting parameters
- Displays live view of device output such as the luminance image and IMU measurements
- Records videos

- Updates the firmware of the device

User Interface

The *User Interface* figure shows the different sections of the user interface.

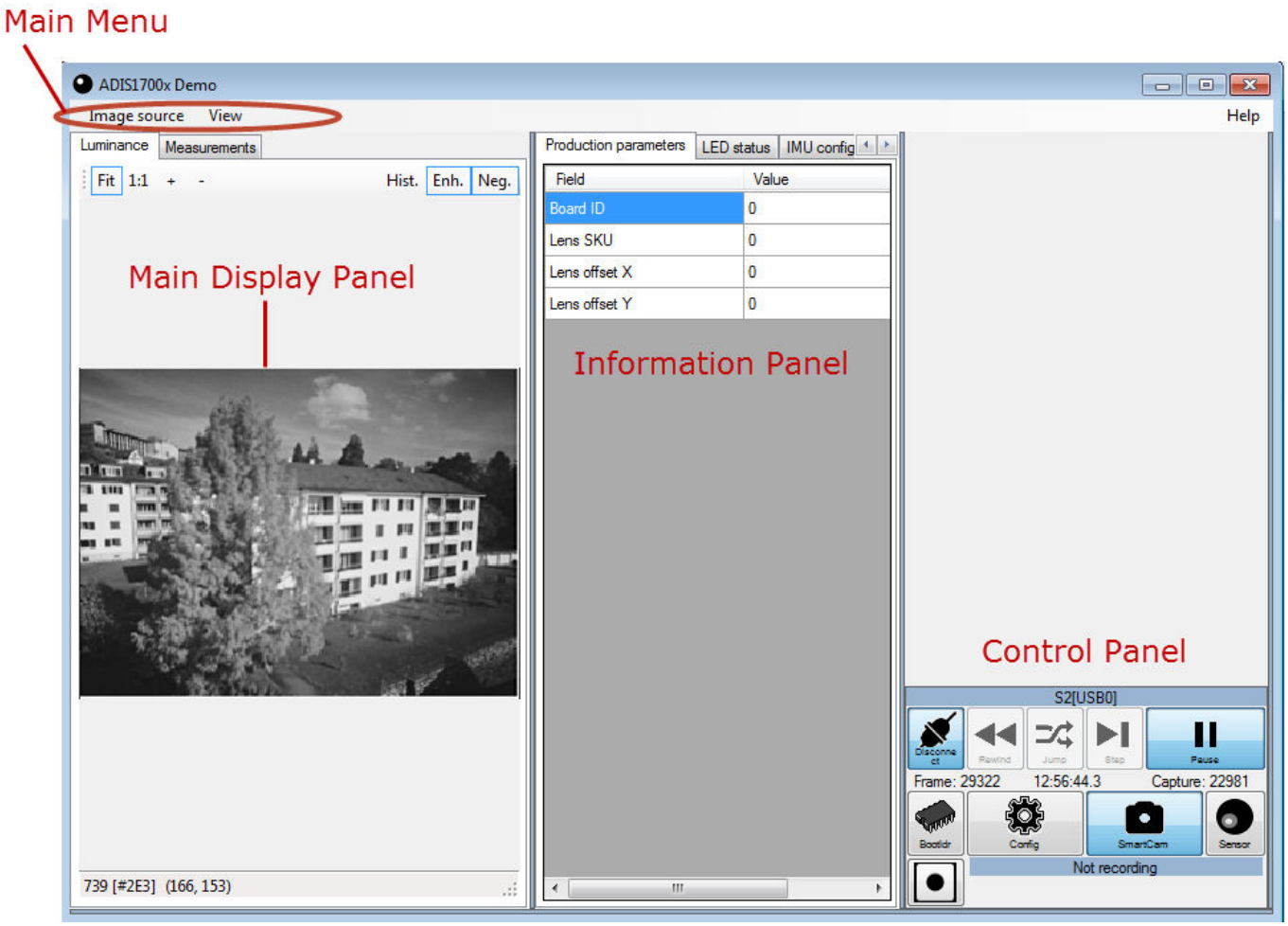


Figure 5-1: User Interface

Connecting to the Device

The following sections describe the steps to connect to the ADIS1700x.

- Open a Connection
- Close a Connection

Open a Connection

Complete the following steps to open a connection.

1. Connect the ADIS1700x unit to the PC through the USB port.

2. From the User Interface Main Menu select ***Image source>Open>ADIS1700x on USB....***

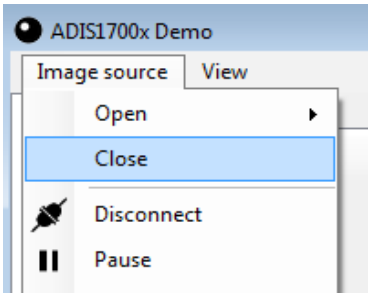
   The application starts a discovery process to find all the devices connected to the PC. A list of connected devices is displayed.

3. Select the device in the list and click ***Select***.

Close a Connection

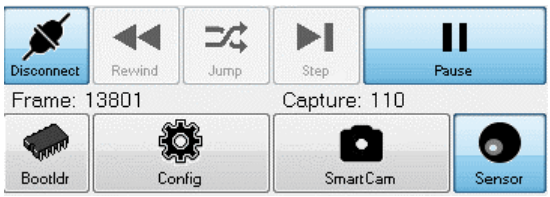
Complete the following step to close a connection.

1. From the User Interface Main Menu select ***Image source>Close***.



Operating Modes

The User Interface Control Panel allows the user to change modes. The ADIS1700x device operates in one of four modes: Bootloader (Bootldr), Configuration (Config), Smart Camera (Smart Cam) and Sensor.



Smart Camera Mode

Smart Camera mode is the default mode. In this mode, the ADIS1700x device is continuously acquiring luminance images from the ADVS200x sensor and measurements from the IMU accelerometer. The data is polled by the PC application and displayed.

Configuration Mode

In configuration mode, the ADIS1700x device does not collect any image from the ADVS200x or any measurements from the IMU. This mode is used by the PC application to allow the user to:

- Set up the ADVS200x parameters
- Set up the IMU parameters
- Set the time on the unit

Bootloader Mode

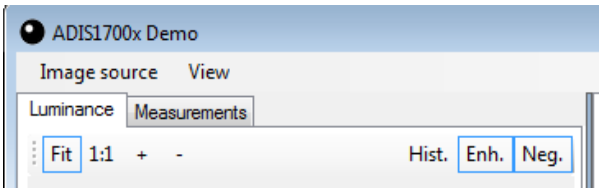
In bootloader mode, a “bootloader” firmware starts running on the ADIS1700x device. The ADVS200x sensor and IMU are not accessed. This mode is used by the PC application to allow the user to upload new firmware.

ADVS200x Luminance Image

The ADIS1700x delivers luminance images captured from the on-board ADVS200x sensor. The image has the following properties:

- Image size is QVGA (320 x 240 pixel)
- Each pixel is a 10-bit value (ranging from 0 to 1023), where 0 is the brightest pixel and 1023 is the darkest pixel
- The pixel value is proportional to the log of the light intensity. The value is an indicator of the time the pixel requires to reach a reference voltage

In Smart Camera mode, ADIS1700x Demo application continuously polls for the luminance image. The application displays the image in the User Interface Main Display Panel under the ***Luminance*** tab.



Displayed Image Enhancement

The ***Image Enhancement*** figures show the effect of enhancement on an image of a building.

Table 5-1: Image Enhancement

Image Without Enhancement	Image With Enhancement
	

Displayed Image Negation

The luminance image has a value of 0 for the brightest pixel and a value of 1023 for the darkest pixel. Luminance value is an indicator of the time for the pixel to reach a reference voltage. A pixel exposed to a bright light reaches the reference voltage in a shorter period of time and therefore its value is small. A pixel exposed to a dim light reaches the reference voltage in a longer period of time and therefore its value is large.

It is possible to negate the pixel values for displaying. So, a 0 value pixel is displayed bright and a 1023 value pixel is displayed dark.

The ***Image Negation*** figures show the effect of negation on an image of a building.

Table 5-2: Image Negation

Image Without Negation	Image With Negation
	

Image Histogram

In the User Interface Main Menu under the ***Luminance*** tab, select the ***Hist.*** option to display a histogram of the image.

A histogram is computed and updated for each new image displayed. The histogram is displayed below the image as shown in the ***Histogram Example*** figure. The horizontal axis shows the pixel values. The vertical axis shows the number of pixels with a value.

Figure 5-2: Histogram Example

Using Spread

It is possible to select a window in the histogram for displaying the image. Edit the *Spread* parameters to modify a window. The width of the window is defined by the *Spread Bits* value. The position of the window in the histogram is defined by the *Spread Min* value.

For example:

- If *Spread Bits* is set to 7, the window width is 2^7 (for example, 128)
- If the *Spread Min* is set to 154, so the window position is at 154

As a result, the histogram window starts at 154 and ends at 281. In the displayed image all pixels below 154 and above 281 are clipped. The ***Histogram Using Spread*** image shows the modified image and histogram.

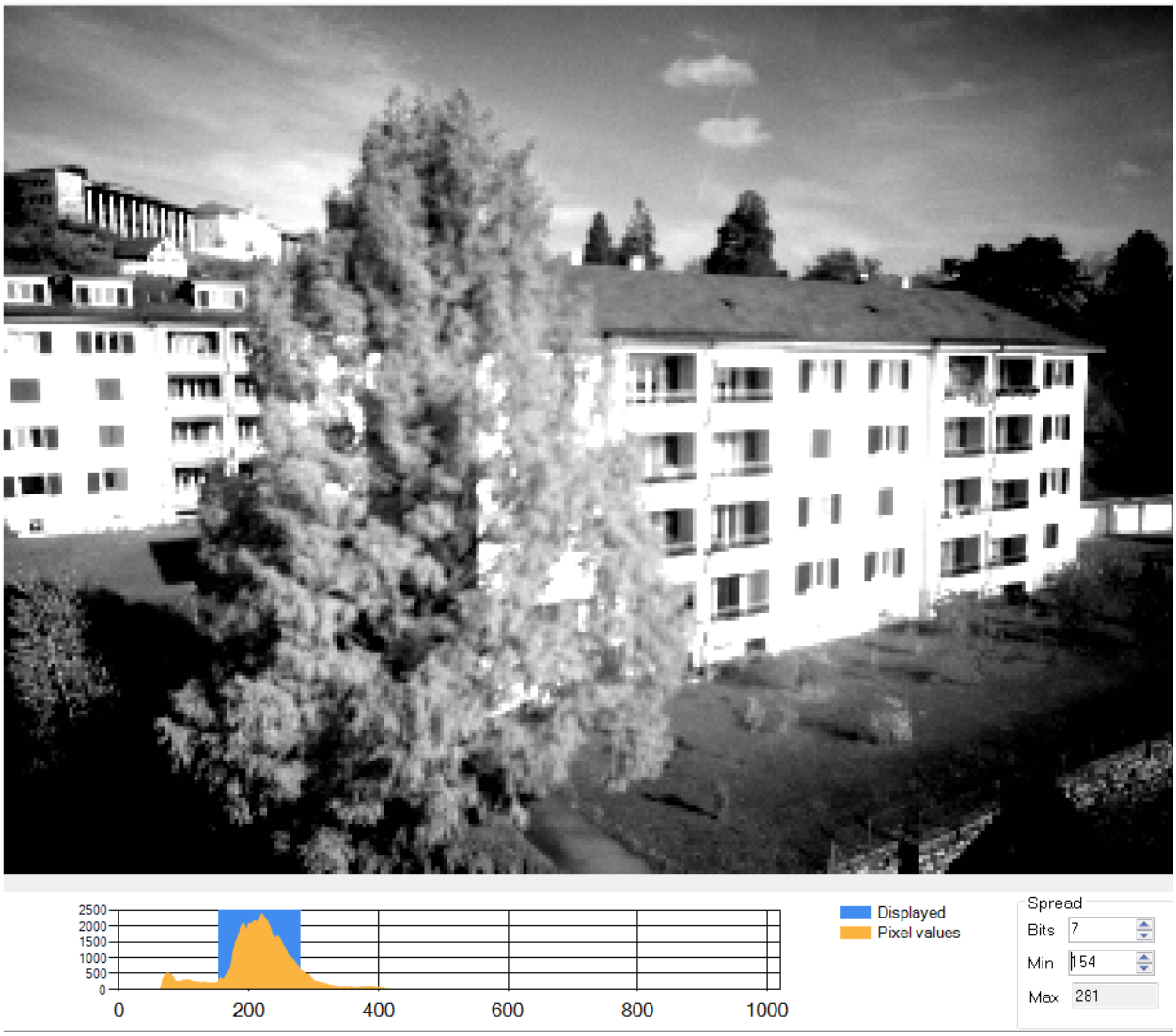


Figure 5-3: Histogram Using Spread

Configuring the ADVS200x

The following tasks are associated with configuring the ADVS200x:

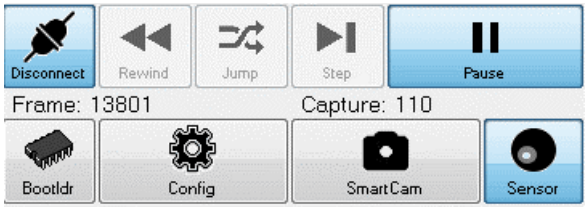
- Modifying ADVS200x Parameters
- Saving and Loading ADVS200x Parameters

Modifying ADVS200x Parameters

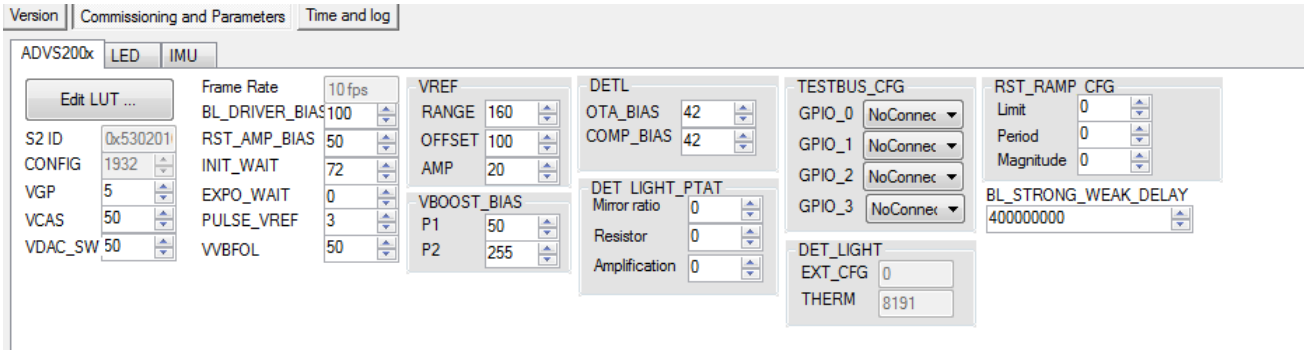
The following steps describe how to edit the ADVS200x parameters.

1. Select Configuration mode.

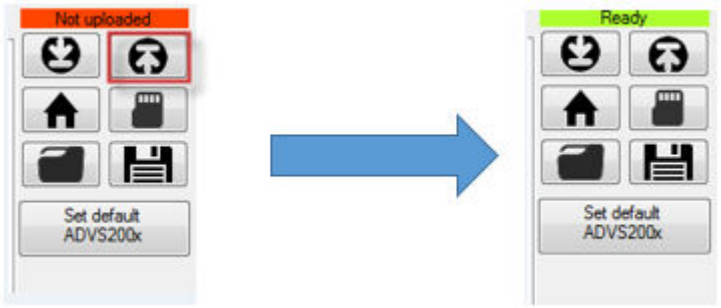
   NOTE: Switching modes can take a few seconds.



2. In the configuration panel select the ***Commisioning and Parameters*** tab.



3. Select the ***ADVS200x*** tab.
4. Modify the parameters, as required.
5. Click ***Upload*** icon to upload the ADVS200x parameters to the ADIS1700x device.



6. Click ***Yes*** when requested to save the data on the device flash.
7. Select Smart Camera mode

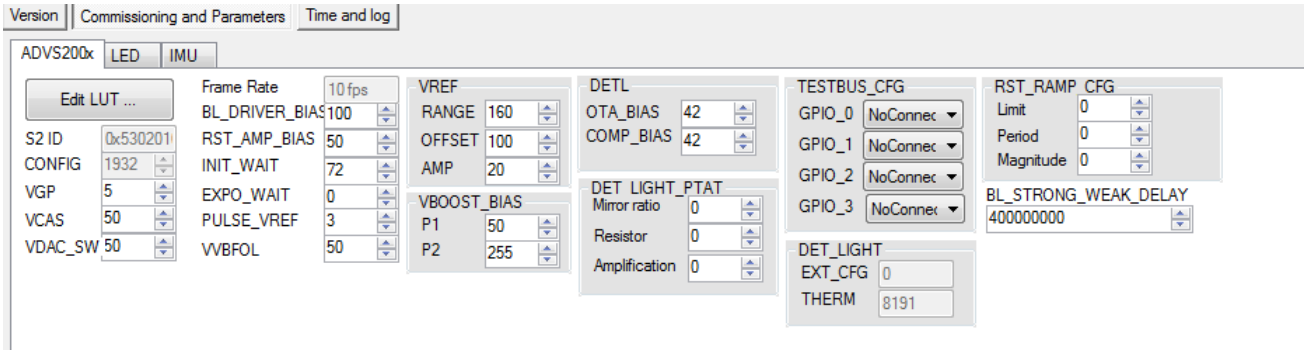
Setting the Defaut ADVS200x Parameters

The following steps describe how to set the default ADVS200x parameters.

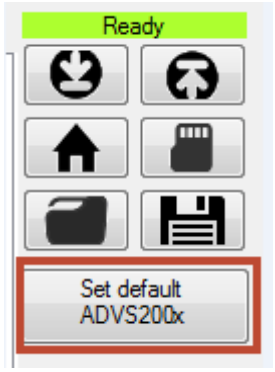
1. Select Configuration mode.

   NOTE: Switching modes can take a few seconds.

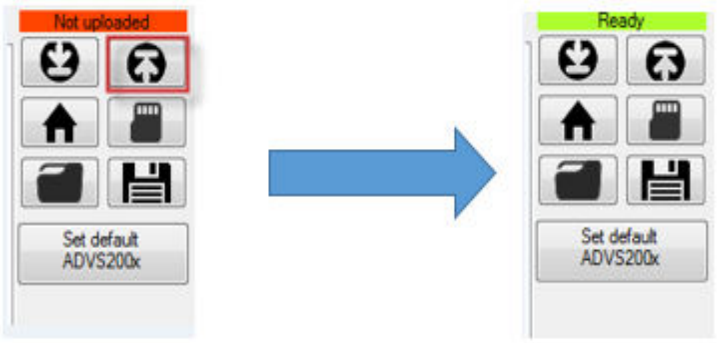
2. In the configuration panel select the ***Commissioning and Parameters*** tab.



3. Select the ***ADVS200x*** tab.
4. Click ***Set default ADVS200x***.



5. Click ***Upload*** to upload the ADVS200x parameters to the ADIS1700x device.



6. Click *Yes* when requested to save the data on the ADIS1700x device flash.
7. Select Smart Camera mode.

Saving and Loading ADVS200x Parameters

Complete the following steps to save the ADVS200x parameters to a file:

1. Load the ADVS200x Parameters from a file
2. Save the ADVS200x Parameters to a file

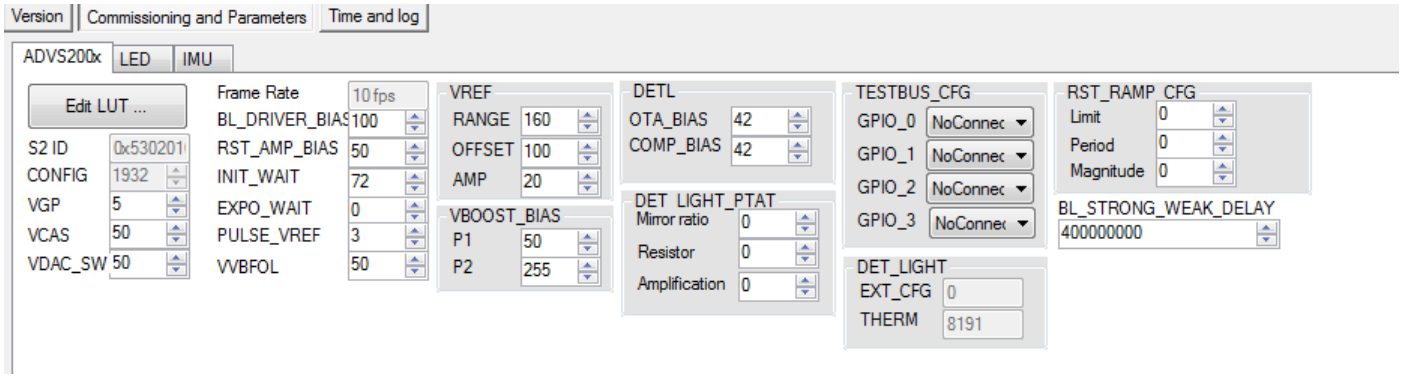
Load the ADVS200x Parameters

Complete the following steps to load the ADVS200x parameters to a file:

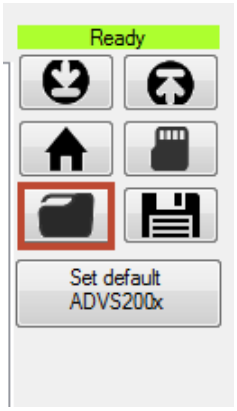
1. Select Configuration mode.

   NOTE: Switching modes can take a few seconds.

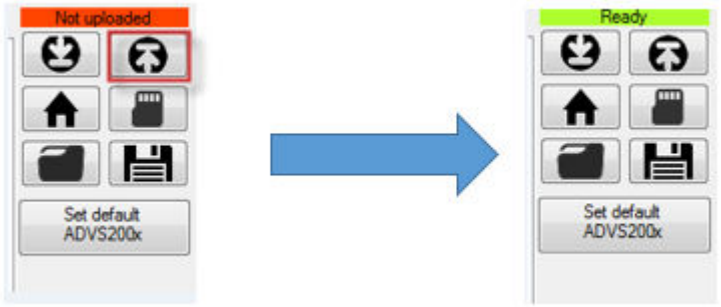
2. In the configuration panel select the *Commissioning and Parameters* tab.



3. Click the ***Load from file*** icon.



4. Click ***Upload*** to upload the ADVS200x parameters to the device.



5. Click *Yes* when requested to save the data on the device flash.
6. Select Smart Camera mode.

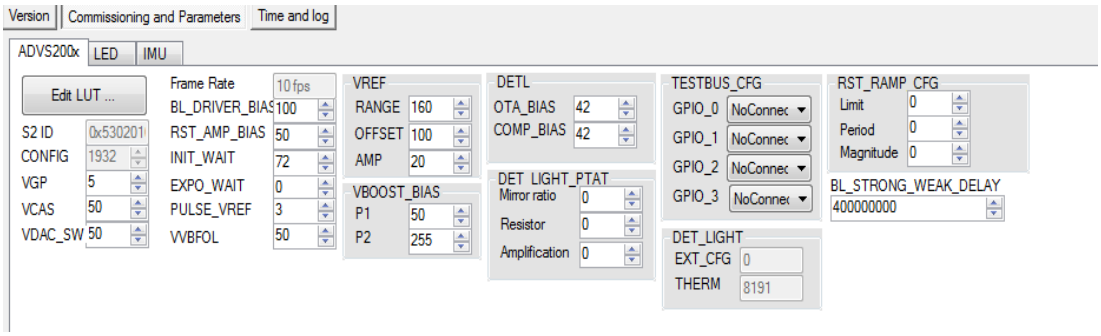
Save the ADVS200x Parameters

Complete the following steps to save the ADVS200x parameters to a file:

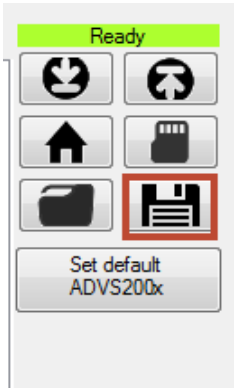
1. Select Configuration mode.

   NOTE: Switching modes can take a few seconds.

2. In the configuration panel select the ***Commissioning and Parameters*** tab.



3. Select the ***ADVS200x*** tab.
4. Click the ***Save to file*** icon.



5. Select Smart Camera mode.

Accelerometer Axis Measurement

The ADIS1700x delivers axis measurements from the on-board accelerometer. Measurements are collected at a pre-defined sample rate and then down-sampled by averaging before making them available to the host application.

In the Smart Camera mode, the ADIS1700x Demo application continuously polls for these measurements and displays them in the User Interface Main Menu under the ***Measurements*** tab.

Graphical Display

The measurements are displayed as a graph when the ***Graph*** option is selected. The graph is updated each time the measurements are polled by the ADIS1700x Demo application. The ***Graphical Display*** figure shows a graph of the accelerometer axis measurements. The vertical axis of the graph changes when the display is refreshed. To view a fixed range, select the ***Fixed range*** option.

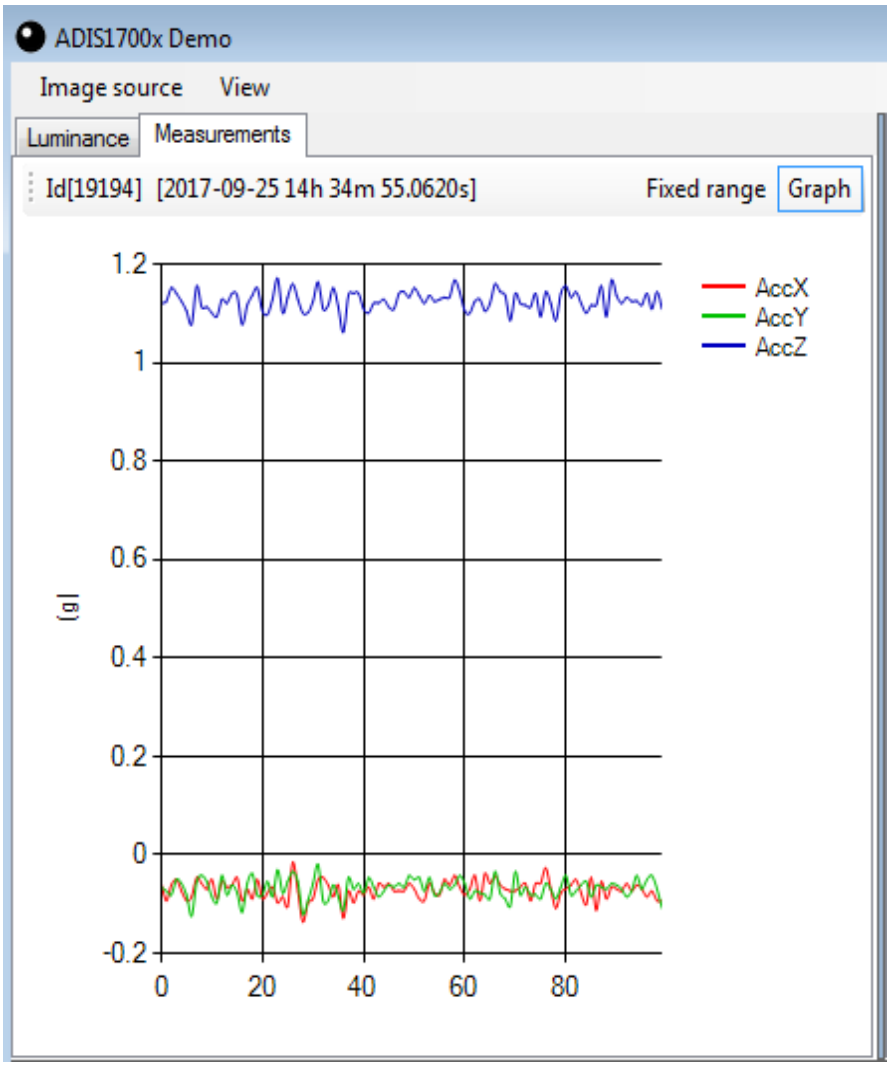
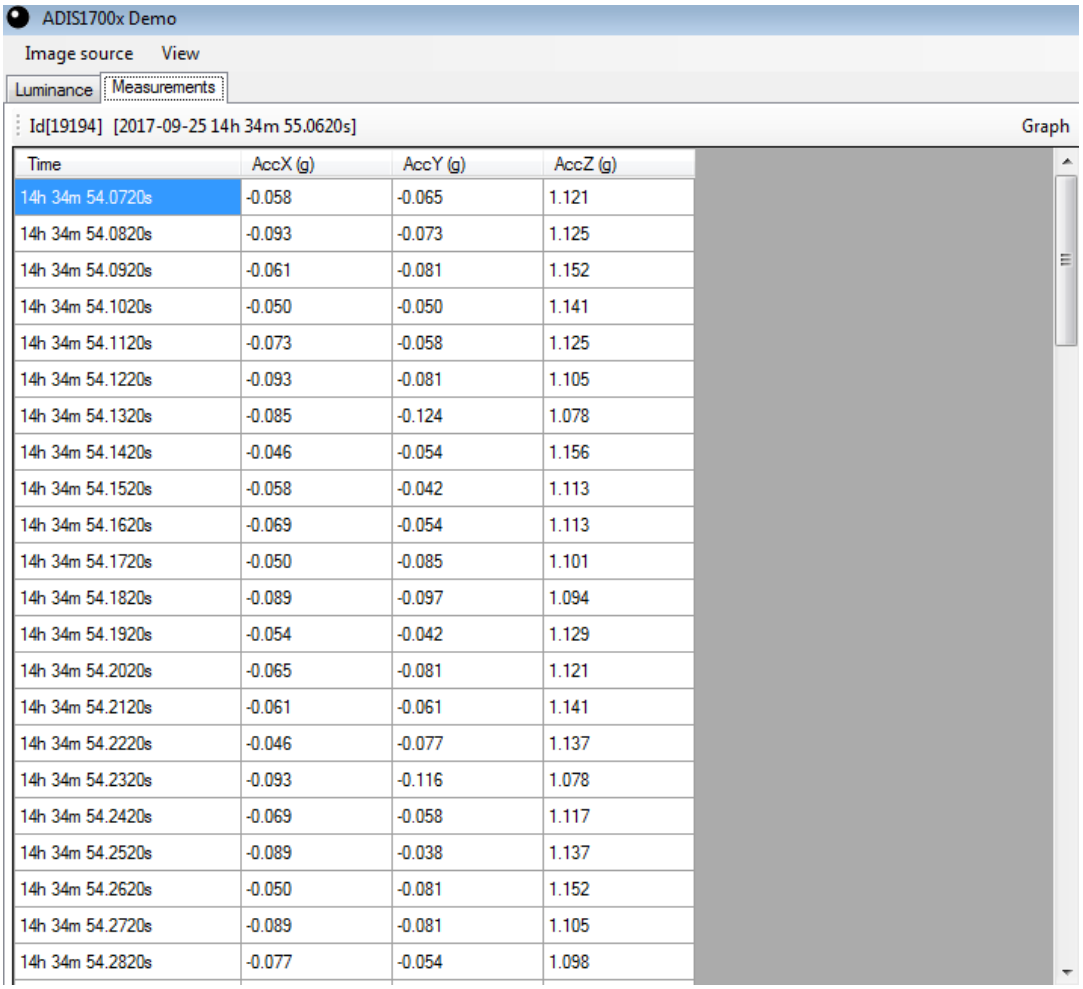


Figure 5-4: Graphical Display

Tab Display

The measurements are displayed as a table of values when the ***Graph*** option is unselected. The table is updated each time the measurements are polled by the ADIS1700x Demo application.



Time	AccX (g)	AccY (g)	AccZ (g)
14h 34m 54.0720s	-0.058	-0.065	1.121
14h 34m 54.0820s	-0.093	-0.073	1.125
14h 34m 54.0920s	-0.061	-0.081	1.152
14h 34m 54.1020s	-0.050	-0.050	1.141
14h 34m 54.1120s	-0.073	-0.058	1.125
14h 34m 54.1220s	-0.093	-0.081	1.105
14h 34m 54.1320s	-0.085	-0.124	1.078
14h 34m 54.1420s	-0.046	-0.054	1.156
14h 34m 54.1520s	-0.058	-0.042	1.113
14h 34m 54.1620s	-0.069	-0.054	1.113
14h 34m 54.1720s	-0.050	-0.085	1.101
14h 34m 54.1820s	-0.089	-0.097	1.094
14h 34m 54.1920s	-0.054	-0.042	1.129
14h 34m 54.2020s	-0.065	-0.081	1.121
14h 34m 54.2120s	-0.061	-0.061	1.141
14h 34m 54.2220s	-0.046	-0.077	1.137
14h 34m 54.2320s	-0.093	-0.116	1.078
14h 34m 54.2420s	-0.069	-0.058	1.117
14h 34m 54.2520s	-0.089	-0.038	1.137
14h 34m 54.2620s	-0.050	-0.081	1.152
14h 34m 54.2720s	-0.089	-0.081	1.105
14h 34m 54.2820s	-0.077	-0.054	1.098

Figure 5-5: Tabular Display

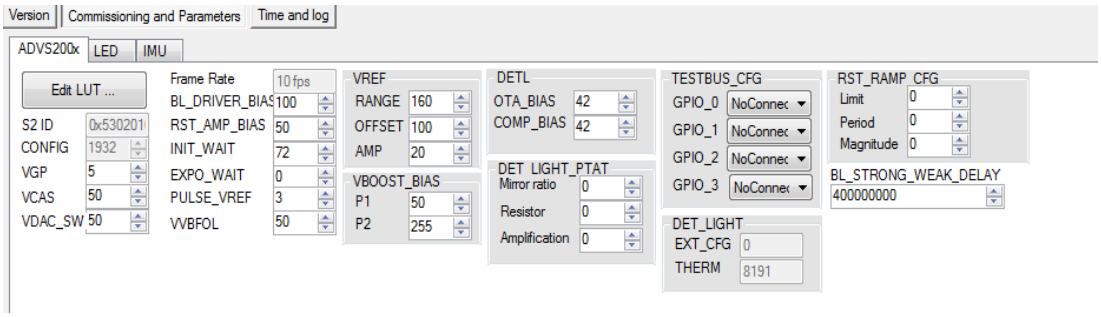
Modifying IMU Parameters

The following steps describe how to edit the IMU configuration parameters.

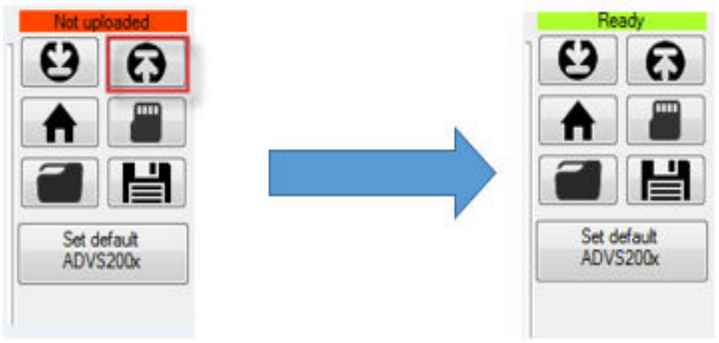
1. Select Configuration mode.

   NOTE: Switching modes can take a few seconds.

2. In the configuration panel select the ***Commissioning and Parameters*** tab.



3. Select the ***IMU*** tab.
4. Modify the parameters, as required.
    a. Choose the ***Accelerometer*** tab to edit the accelerometer offsets, ranges and filter values.
5. Click ***Upload*** to upload the IMU parameters to the ADIS1700x device.



6. Click ***Yes*** when requested to save the data on the ADIS1700x device flash.
7. Select Smart Camera mode

Updating Firmware

There are two tasks associated with updating the firmware stored in the device flash storage:

- Update Bootloader Firmware
- Update Application Firmware

When the device powers up, the bootloader firmware is loaded into memory and starts running. The bootloader firmware then looks for the application firmware in the flash storage. It loads it into memory and starts running it.

When the bootloader firmware is running, the application mode is Bootloader.

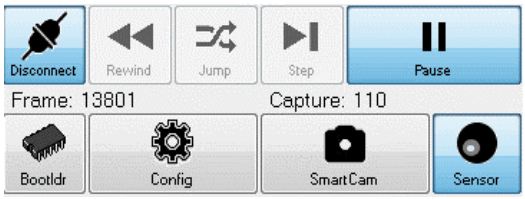
When the application firmware is running, the application mode is Configuration, Sensor or Smart Camera.

To update the bootloader or application firmware, switch to the Bootloader mode (the bootloader firmware is running).

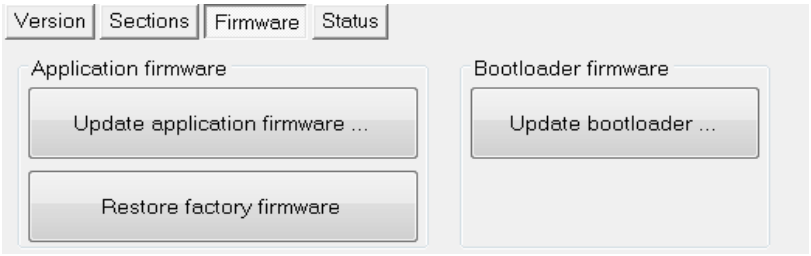
Update Bootloader Firmware

The following steps describe how to update the bootloader firmware.

1. Select Bootloader mode.



2. Select the ***Firmware*** tab.

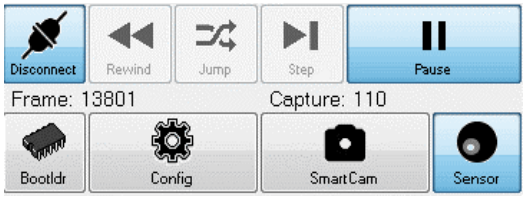


3. Click ***Update bootloader***.
4. Select the firmware file (for example, *ADIS1700x_Bootloader_vXX.ldr* for ADIS1700x).

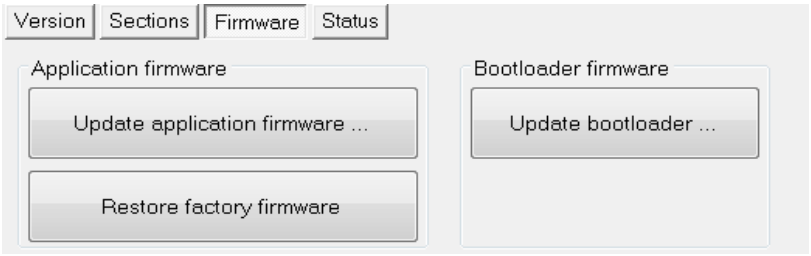
Update Application Firmware

The following steps describe how to update the application firmware.

1. Select Bootloader mode.



2. Select the ***Firmware*** tab.



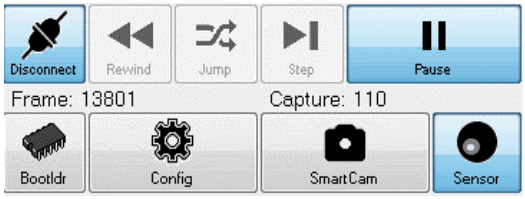
3. Click ***Update application firmware***.

4. Select the firmware file (for example, *ADIS1700x_application_vxx.ldr* for ADIS1700x).

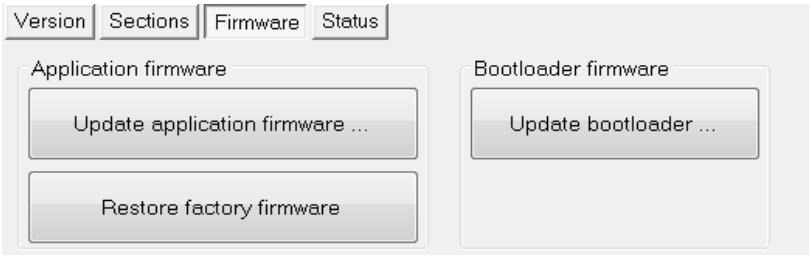
Recovering from a Restore Point

Complete the following steps to recover the original firmware application and its associated data.

1. Select Bootloader mode.



2. Select the ***Firmware*** tab.



3. Click ***Restore factory firmware***.

Device Re-programmer Application

The `Device Re-programmer` application is a UI application that allows the user to reprogram a device through the USB. This application works with the Blip-Mini and ADIS1700x devices. The application has the following features:

- Installs the bootloader firmware
- Installs the application firmware
- Installs the ADVS200x parameters
- Erases all data (logging, processing parameters, commissioning data, etc.)

Re-program the Device

Complete the following steps to reprogram the ADIS1700x camera module.

1. Disconnect all ADIS1700x or Blip-Mini devices from the PC.
2. Launch the Device Re-programmer demo.
3. Select the *Programmer* tab

4. Click ADIS1700x for the *Device type*.

   CAUTION: Select the correct device type. Otherwise, the device could be re-programmed incorrectly and may become unusable without re-programming through a JTAG connector.

5. Click *Continue*.

   Device discovery begins.

6. Plug in the device during the discover process.

   Once the device is plugged-in, the device is automatically discovered and re-programmed.

7. Wait for the device re-programming to be completed. This can take several minutes.

8. Remove the device.

ImgSeqFile Viewer Application

The `ImgSeqFile Viewer` application is a UI application that allows the user to open a video file (`ImgSeqFile`) and browse its contents. This application is available only with the development installer.

NamedValueFile Viewer Application

The `NamedViewerFile Viewer` application is a UI application that allows the user to open a named value file (such as the parameters file) and browse its contents. It also allows exporting of the content in various formats such as JSON and BSON. This application is available only with the development installer.

ADVS200x Demo Application

The `ADVS200x Demo` application is a UI application that demonstrates the capability of the ADVS200x sensor. It provides a user-friendly interface to interact with a device that has an on-board ADVS200x sensor and is running the SNAP Framework (for example, the ADIS1700x). The application has the following features:

- Configures the device, such as adjusting ADVS200x parameters
- Displays live device output such as the luminance image
- Computes and displays motion and occlusion
- Demonstrates conversion between image and world coordinates (camera calibration)
- Records and plays back videos
- Updates the firmware of the device

The `ADVS200x Demo` application communicates with the device using the SNAP Sensor library. A fully documented API of this library for .Net (Windows) and C++ (Windows and Linux) is available.

See the ***ADVS200x Demo User Guide*** for more information.

ADVS200x Parameters Editor Application

The `ADVS200x Parameters Editor` application is a UI application that allows the user to:

- Edit the ADVS200x configuration parameters
- Read and write the ADVS200x parameters to a file

This application does not require that the device be connected to run.

This application is available only with the development installer.

6 ADIS1700x Library

The ADIS1700x library is used to communicate between the host computer and the ADIS1700x. The library consists of APIs for the following functions:

- Device Discovery and Initialization — Discover and connect to devices that are running the SNAP Framework and ADIS1700x application firmware
- Application Level Data — Communicate with the ADIS1700x application firmware running in a device using the SNAP Framework
- IMU Service — Communicate with the IMU in a device that uses the SNAP Framework, adjust parameters and read data
- ADVS200x Camera Service — Communicate with the ADVS200x sensor

For details, see the Communication Protocol.

Device Discovery and Initialization

The APIs defined in this section are for device discovery and initialization.

DiscoverFrameworkDevices

Discovers devices connected to the SBC and returns an array discovered devices

Class

FrameworkConnectionFactory

Returns

An array of FrameworkDevice objects, each representing a unique ADIS1700x device connected to the SBC

CreateADIS1700xApplication

Creates an instance of ADIS1700x Application object that allows communication to the ADIS1700x device

Class

ADIS1700xConnectionFactory

Parameters

The FrameworkDevice objects found in discovery process

Connect

Establishes a communication channel to the ADIS1700x device

Class

ADIS1700xApplication

Disconnect

Closes the communication channel to the ADIS1700x device

Class

ADIS1700xApplication

Ping

The host sends a Ping message to the ADIS1700x device to check if the communication is alive. The device sends back an Acknowledge (ACK) message.

Class

ADIS1700xApplication

GetSoftwareVersion

Returns the firmware version of the ADIS1700x application

Class

ADIS1700xApplication

Returns

The firmware version: ReleaseType, MajorVersion, MinorVersion, Build

Application Level Data

The APIs defined in this section are used for application level data.

GetLedStatus

Returns the status of the on-board LEDs

Class

ADIS1700xApplication

Returns

Led status of LEDs present

Parameters

An array LedStatus structures. Each LedStatus object represents the status of one LED and consists of the:

- LED ID
- LED Mode (0 = OFF, 1 = ON)

SetLedStatus

Sets the status of the on-board LEDs

Class

ADIS1700xApplication

Parameters

An array LedStatus structures. Each LedStatus object represents the status of one LED and consists of the:

- LED ID
- LED Mode (0 = OFF, 1 = ON)

IMU Service

The APIs defined in this section are used for serving the IMU.

GetConfiguration

Returns the IMU configuration settings

Class

ADIS1700xImuModule

Returns

- Accelerometer sampling rate (32-bit integer representing milliHertz)

NOTE: Measurements are collected from the accelerometer at a higher rate (1KHz) and the data is downsampled to the configuration value. Currently, this value is 100Hz.

- Accelerometer X-axis range: (32-bit integer value representing mg)
- Accelerometer Y-axis range
- Accelerometer Z-axis range
- Accelerometer temperature range

GetMeasurements

Returns the accelerometer measurements

Class

ADIS1700xImuModule

Parameters

The number of measurements (last n) to return

NOTE: Maximum 1000

Flags

Flags indicating if the following should be returned:

- Timestamp for each measurement
- Accelerometer X-axis range: (32-bit integer value representing mg)
- Accelerometer Y-axis range
- Accelerometer Z-axis range

Returns

The number of measurements returned

- A 32-bit id of the last measurement.

  NOTE: The measurement is downsampled from the actual rate of reading the accelerometer. Each time a downsample measurement is computed, its ID is incremented and the measurement is tagged with a time.

- Time stamp of the last measurement:
  - A 16-bit days
  - A 32-bit time-of-day (in resolution of 0.1ms)
- An array of 32-bit time-of-day tags for each measurement returned (if requested)

- An array of 16-bit accelerometer X measurements (if requested)
- An array of 16-bit accelerometer Y measurements (if requested)
- An array of 16-bit accelerometer Z measurements (if requested)

ADVS200x Camera Service

The APIs defined in this section are used for servicing the camera.

GetADVS200xRegisters

Returns the values of the ADVS200x imager registers

Class

CameraModule

Parameters

The IADVS200x registers interface to an object containing ADVS200x imager register values

Returns

The register values are filled up in the object

SetADVS200xRegisters

Sets the values of the ADVS200x imager registers

Class

CameraModule

Parameters

The IADVS200x registers interface to an object containing ADVS200x imager register values

GetADVS200xLut

Returns the values of the ADVS200x imager look-up table

Class

CameraModule

Parameters

The IADVS200x LUT interface to an object containing ADVS200x imager LUT values

Returns

The LUT values are filled up in the object

SetADVS200xLut

Sets the values of the ADVS200x imager look-up table

Class

CameraModule

Parameters

The IADVS200x Lut interface to an object containing ADVS200x imager LUT values

GetLuminanceImage

Returns the last captured luminance image

Class

CameraModule

Returns

A 32-bit id (frame index) of the last captured image

NOTE: The frame index is incremented each time the processor receives a new image from the ADVS200x imager

- Time stamp of the last measurement:
  - A 16-bit days
  - A 32-bit time-of-day (in resolution of 0.1ms)

  The time stamp is computed as the halfway point during the integration period of the image.
- The image interface to an object containing a 320x240 10-bit luminance image

7 Communication Protocol

The ADIS1700x communication protocol, also known as the SNAP Protocol, provides the interface between the ADVS200x imager and the host PC. The protocol can be used to integrate other sensors and modules. It is part of the ecosystem available to develop embedded applications built around the ADVS200x sensor. The development system consists of two nested layers:

- The SNAP Framework - an embedded library (on the ADIS1700x)
- The SnapSensorLibrary and ADIS1700x library (on the host)

The SNAP Framework offers a set of services and drivers to quickly evaluate products and rapidly prototype their applications.

One of the key features is that the SNAP Framework library can grant to every high-level embedded module (service) a connection to a host PC. A host PC application can establish and manage this connection using the tools provided by the SNAP Sensor library and ADIS1700x library. This connection implements the SNAP protocol.

The SNAP Framework library can be used in all the stages of the development including:

- Product evaluation: sensor data quality can be evaluated and the customer can rapidly check the sensors fit their needs.
- Prototyping: the application can be developed on a host PC and demonstrated live using data from the device. Data can be captured for different scenarios and stored into scenario capture files that the developer can then playback on the host PC for non-regression, alternate algorithms comparisons, and performance evaluation.
- Product development: the code is ported to the device and it can be tested inverting the data flow (playing back the sensors data library on the device).

Firmware

The device firmware usually consists of two packages:

- Bootloader Firmware
- Application Firmware

Architecture Description

The SNAP protocol uses a master/slave architecture. Messages from the master to the slave are called *commands*; messages from the slave to the master are called *responses*.

The communication sequence is always initiated by the master. The master sends a command; the slave acknowledges the command and then sends the response. The sequence is terminated by the master which acknowledges the response. Each sequence must be completed before the next sequence can start. Commands and responses have a header and payload structure as shown in the ***Header + Payload Structure*** table .

Table 7-1: Header + Payload Structure

Header_L1	Payload_L1	
	Header_L2	Payload_L2

The payload can encapsulate another header + payload structure. In this case, there are two levels (layers). The first layer is always present and is known as the *transport layer*. A second layer, known as the *message layer*, is foreseen only for commands and responses whose content type is message. The second layer is defined as the message layer.

The following two requirements apply:

1. Fields shall be word aligned. Bytes can be packed. 16-bit fields shall be 16-bit word aligned and 32-bit fields shall be 32-bit word aligned.
2. Endianness is little endian (as for BF70x family and intel PCs)

These two requirements permit easier message handling when the client (generally, the embedded processor) and the host (generally, the PC) have the same endianness. In this case, the entire data structure can be copied to or from the message payload buffer when respectively writing or reading messages.

Transport Layer

The transport layer is the header of the first level. It is identical for commands and responses.

Commands and responses consist of four types:

- Ping
- ACK (acknowledge)
- NACK (not acknowledged)
- Message

Ping, ACK and NACK do not have a payload. The entire protocol is based on content type 5 (Type Message) which has a payload. The payload of a command or response of type message encapsulates a new message with its own header and payload.

The Ping, ACK and NACK messages are sent to the host once the transport layer has been decoded and verified by the device. The verification is based only on the consistency of the two error-detecting codes (the last two fields in the header).

Table 7-2: Transport Layer Header

Field Name	Type	Notes
PlatformId	U2	Set to 0x3254
PacketId	U2	Incremented at every new command.
PacketSize	U4	Header + payload size
ContentType	U1	One of four possible values : Ping – 0x00 ACK – 0x01 NACK – 0x02 TYPE_MESSAGE – 0x05
HeaderChecksum	U1	LSByte of the two's complement sum of all the header bytes
PacketChecksum	U2	Fletcher's checksum on the full message (header + payload)

Message Layer

The message layer encapsulates a header + payload structure. The header format is similar for commands and responses. However, the response header has an additional field to report the result.

Table 7-3: Message Layer Header - Command

Field Name	Type	Notes
Module ID	U1	
Reserved1	U1	Set to 0
Command ID	U2	
Version	U4	Version of the command ID
Reserved2	U4	Set to 0
Payload Size	U4	
Reserved3	U4	Set to 0

Table 7-4: Message Layer Header - Response

Field Name	Type	Notes
Module ID	U1	
Reserved1	U1	Set to 0

Table 7-4: Message Layer Header - Response (Continued)

Field Name	Type	Notes
Response ID	U2	Identical to the respective command
Version	U4	Version of the response ID
Reserved2	U4	Set to 0
Payload Size	U4	
Result Status	U4	

In the firmware architecture, the protocol handling is distributed. Every module is responsible for its messages. The module ID gives the communication module a way to dispatch messages to each module.

The ***Module Identifiers*** table shows the recipient modules and identifiers.

Table 7-5: Module Identifiers

Module ID	Target Module
0x00	Reserved
0x01	Main Application / Bootloader
0x02	Processing Module
0x10	Camera (SNAP Framework)
0x12	IMU (SNAP Framework)
0x40	Storage Module (SNAP Framework)

The header fields include:

- Module ID — the module to which the command is redirected or the source of the response.
- Reserved — a part of the header reserved for module-specific data. For example, when multiple cameras exist in the camera module, it can be used for the unit ID.
- Command ID — Contains the ID of the command. IDs from 0 to 0xF are reserved as shown in the ***Reserved Command IDs*** table.
- Version — version of the software. Refer to this field when the payload of the message has changed and is not applicable to older software versions.
- Payload Size — The payload size can be computed starting from the full payload size available in the transport layer header (but this redundancy can avoid lot of coding mistake issues).
- Result Status — Response-specific error code. The ***Reserved Results Status*** table shows the codes that are reserved.

Table 7-6: Reserved Command IDs

Command ID	Description
0x00	Factory Reset
0x01	Reset
0x02	Reserved
0x03	Module SW Version
0x04 to 0xF	Reserved

Table 7-7: Reserved Results Status

Module ID	Reserved Result Status	Notes
0x00	OK	
0x01	Module ID Not Present	
0x02	Invalid Version	
0x03	Invalid Payload	Payload size or structure is invalid
0x04	Command Failed	
0x05	Invalid Command ID	Command ID does not exist
0x06	Command Unavailable	Command unavailable in the current state or mode. For example, trying to change IMU parameters while the sensor is in configuration mode generates a Command Unavailable message.
0x07 to 0xF	Reserved	

Operational Modes

Some of the commands require the device to be in a specific operational state (mode). The application defines the operational states. For the ADIS1700x application, the operational states are:

- Configuration Mode
- SmartCamera Mode
- Sensor Mode

The Bootloader application has only one operational state: Bootloader. The mode can be read or modified with the Get/Set Mode commands.

In general, commands devoted to tuning parameters or non-volatile data storage require that the device be in Configuration mode. Commands concerned with sensor measurements require the device to be in Smart Camera or Sensor Mode.

When the device cannot execute a command due to an invalid mode, a response with the `Result Status` field set to 0x06 Command Unavailable is sent to the host.

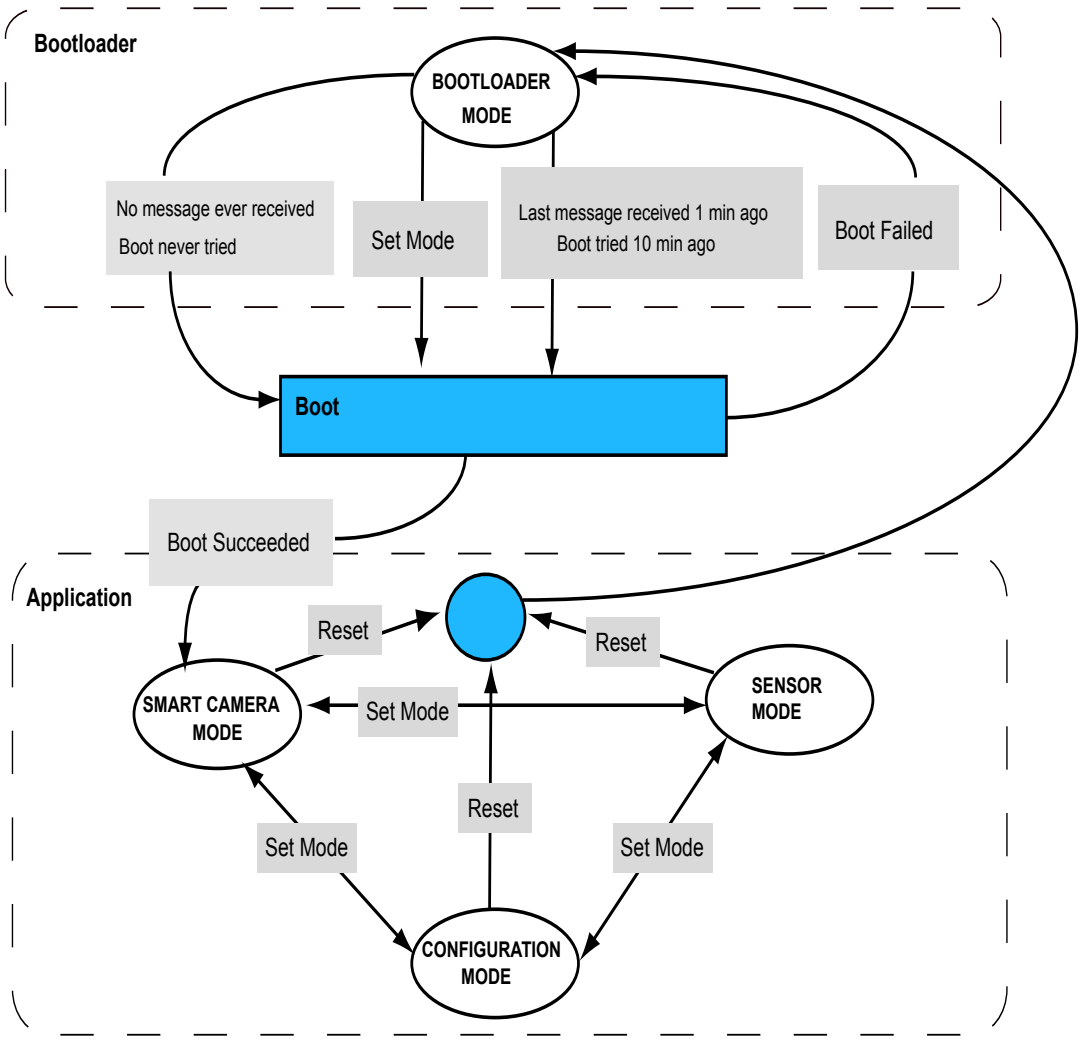


Figure 7-1: ADIS1700x Operating Modes

Bootloader Firmware

The Bootloader firmware is software that is not meant to change during the life of the product.The bootloader is run after a power cycle or hardware reset. It listens to the main communication link (for example, the USB) for a short interval (usually 10s). If no message is received, it launches the application, when a valid application is present. If not, it waits indefinitely for a host command.

The main features of the bootloader are:

- Allows the user to upgrade the factory application firmware with a new firmware (in this case, the factory firmware is not deleted but used as backup).
- Restores the factory application firmware if something goes wrong during the upgrade
- Allow a window to control the device during startup. If there are problems with the factory application firmware or upgrade application firmware, there is always few seconds after power cycle where the user can regain control of the device and upload new software.

Bootloader Messages

The bootloader module implements the commands shown in the ***Bootloader Commands*** table. Any command other than `Reset` sets the bootloader into service mode. When in service mode, the bootloader waits 10 minutes for a new command before trying to launch the application firmware.

Table 7-8: Bootloader Commands

ID	Command	Notes
0x01	Reset	Resets the device.
0x02	Reserved	
0x03	Module SW Version	
0x09	Get Device Status	
0x10	Get Device Info	Returns information like device serial number (unique device identification), product name and manufacturer.
0x11	Get Mode	Always returns 0x10 bootloader mode(for uniformity and coherence with the application).
0x12	Set Mode	Set to a mode different from 0x10 bootloader to boot the current firmware (usually the default mode of the application to boot).
0x14	Set Device Info	
0x20	Get Sections Info	
0x21	Clear Sections	
0x22	Read Section	
0x23	Write Section	
0x24	Validate Section	
0x2A	Create/Recover Restore Point	
0xFF00	Debug Message	

Any command other than Reset delays the attempt to launch the most updated application firmware by 1 minute.

Code Upload

When uploading a new code, the host shall:

1. Clear the associated section (Clear Sections)
2. Upload the data (use multiple Write Sections)
3. Validate the section (Validate Section)

Reset

The bootloader software triggers a hardware reset; the device boots from FLASH (first sector).

C++ API:

```
SnapSensor::SensorConnection::ApplicationModule::Reset()
```

Table 7-9: Reset Header - Command

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x01
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0
Reserved3	U4	0

Table 7-10: Reset Header - Response

Field Name	Type	Notes
Module ID	U1	0
Reserved1	U1	0
Command ID	U2	0x01
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0
Result Status	U4	0

Module SW Version

The SW Version command returns the software version of the module.

C++ API:

```
void
SnapSensor::SensorConnection::ApplicationModule::GetSoftwareVersion(
FrameworkModuleVersion *version)
```

Table 7-11: SW Version Command - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x03
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0
Reserved3	U4	0

Table 7-12: SW Version Response- Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x03
Version	U4	1
Reserved2	U4	0
Payload Size	U4	4
Result Status	U4	0

Table 7-13: SWVersion Payload Response

Field Name	Type	Notes
Release Type	U1	0 – production 1 – beta 2 – development; No debug information 3 – development – Has debug information
Major SW Version	U1	
Minor SW Version	U1	
Build	U1	

Get Device Status

The Get Device Status message provides information about software name and version and useful information for diagnostic like device status, error counts and last error message.

C++ API:

```
void SnapSensor::SensorConnection::ApplicationModule::
GetDeviceStatus(DeviceStatus *deviceStatus)
```

Table 7-14: Get Device Status Command - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x09
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0
Reserved3	U4	0

Table 7-15: Get Device Status Response- Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x09
Version	U4	1
Reserved2	U4	0
Payload Size	U4	76
Result Status	U4	0

Table 7-16: Get Device Status Payload - Response

Field Name	Type	Notes
Product ID	U4	See Appendix A
Product Name	U32	See Appendix A
SW Version	U24	Same format as Module SW Version command
Current State	U2	Reserved for future use
Errors Count	U2	
Last Error Message	U32	See Appendix A

Get Device Info

The Get Device Info message returns device information such as device serial number (unique device identification), device name and manufacturer, and user-defined metadata.

C++ API:

```
void SnapSensor::SensorConnection::ApplicationModule::
GetDeviceInfo(DeviceInfo *deviceInfo)
```

Table 7-17: Get Device Info Command - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x10
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0
Reserved3	U4	0

Table 7-18: Get Device Info Response - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x10
Version	U4	1
Reserved2	U4	0
Payload Size	U4	116 + N
Result Status	U4	0

Table 7-19: Get Device Info Response - Payload

Field Name		Type	Notes
Unique ID		U16	Unique identifier
Manufacturer ID		U4	Manufacturer unique identifier. See Appendix A.
Manufacturer Name		U32	Manufacturer name. See Appendix A.
Device ID		U4	Device unique identifier. See Appendix A.
Device Name		U32	Device name. See Appendix A.
HW Version		U16	
Metadata	Length	U4	Data length N; N ≤ 2048
	Type	U4	
	Version	U4	
	Data	U1 x N	

Set Device Info

The Set Device Info message is used to save to a persistent memory (generally FLASH) information such as device serial number (unique device identification), device name and manufacturer, and user-defined metadata.

C++ API:

```
void SnapSensor::SensorConnection::ApplicationModule::
SetDeviceInfo(DeviceInfo *deviceInfo)
```

Table 7-20: Set Device Info Command - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x14
Version	U4	1
Reserved2	U4	0
Payload Size	U4	116 + N
Reserved3	U4	0

Table 7-21: Set Device Info Command - Payload

Field Name		Type	Notes
Unique ID		U16	Unique identifier
Manufacturer ID		U4	Manufacturer unique identifier
Manufacturer Name		U32	Manufacturer name
Device ID		U4	Device unique identifier
Device Name		U32	Device name
HW Version		U16	
Metadata	Length	U4	Data length N; N ≤ 2048
	Type	U4	
	Version	U4	
	Data	U1 x N	

Table 7-22: Set Device Info Response - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0

Table 7-22: Set Device Info Response - Header (Continued)

Field Name	Type	Notes
Command ID	U2	0x14
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0
Result Status	U4	0

Get Sections Info

The Get Sections Info command returns a list of the sections in the non-volatile computer storage medium with its size and offset.

Table 7-23: Get Sections Info Command - Header

Field Name	Type	Notes
Module ID	U1	1 – Bootloader module 0x40– Storage module
Reserved1	U1	NV storage medium ID
Command ID	U2	0x20
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0

Table 7-24: Get Sections Info Response - Header

Field Name	Type	Notes
Module ID	U1	1 – Bootloader module 0x40– Storage module
Reserved1	U1	NV storage medium ID
Command ID	U2	0x20
Version	U4	1
Reserved2	U4	0
Payload Size	U4	N × 4 × 5 + 8
Result Status	U4	0

Table 7-25: Get Sections Info Response - Payload

Field Name		Type	Notes
Sector Length		U4	Length of the smaller erasable area in the storage medium in bytes. Offsets and sizes are expressed in terms of sectors.
Num Sections		U4	N
× N	ID	U4	Section data
	Offset [Sectors]	U4	
	Size [Sectors]	U4	
	Free Size [Sectors]	U4	
	Version	U4	

Clear Sections

The Clear Sections command sends a request to clear the sections specified in a list.

Table 7-26: Clear Section Command - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	NV storage medium ID
Command ID	U2	0x21
Version	U4	1
Reserved2	U4	0
Payload Size	U4	4 + 4 × N

Table 7-27: Clear Sections Command - Payload

Field Name		Type	Notes
Num Sections		U4	N
× N	ID (*1)		IDs of sections that are cleared

*1 The number of sections that are cleared can be larger than the ones indicated in the message. The objective of the bootloader is to preserve integrity. For example, clearing the boot section clears up all the sections.

Table 7-28: Clear Sections Response- Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	NV Storage medium ID
Command ID	U2	0x21

Table 7-28: Clear Sections Response- Header (Continued)

Field Name	Type	Notes
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0
Result Status	U4	0

Read Section

Reads a section. Offset and size are expected to be expressed in sectors. The sector size is provided by the Get Sections Info message.

Table 7-29: Read Section Command - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	NV Storage medium ID
Command ID	U2	0x22
Version	U4	1
Reserved2	U4	0
Payload Size	U4	12

Table 7-30: Read Section Command - Payload

Field Name	Type	Notes
Section ID	U4	Section ID to read
Offset [sectors]	U4	Relative offset with respect to the section start
Size [sectors]	U1	N

Table 7-31: Read Section Response- Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	NV storage medium ID
Command ID	U2	0x22
Version	U4	1
Reserved2	U4	0
Payload Size	U4	12 + Size
Result Status	U4	0

Table 7-32: Read Section Response - Payload

Field Name		Type	Notes
Offset [sectors]		U4	
Num Sections		U4	N; where N cannot exceed 8
M	Data	U1	M = N × Sector Size

Write Section

The Write Section command writes a section. Each response returns the data that was read back after the write operation (cache disabled). Whenever the bootloader receives a `Write Section` command, it first invalidates the section and then starts writing. The section is validated after a successful reception of the Validate Section command. Offset and size are expected to be expressed in sectors. The sector size is provided by Get Sections Info message.

Table 7-33: Write Section Command - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	NV storage medium ID
Command ID	U2	0x23
Version	U4	1
Reserved2	U4	0
Payload Size	U4	12 + Size

Table 7-34: Write Section Command - Payload

Field Name		Type	Notes
Section ID		U4	Section ID to write
Offset		U4	Relative offset with respect to the section start
Size		U4	N
× N	Data	U1	

Table 7-35: Write Section Response- Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	NV storage medium ID
Command ID	U2	0x23
Version	U4	1
Reserved2	U4	0

Table 7-35: Write Section Response- Header (Continued)

Field Name	Type	Notes
Payload Size	U4	12 + Size
Result Status	U4	0

Table 7-36: Write Section Response - Payload

Field Name	Type	Notes
Section ID	U4	
Offset [sectors]	U4	
Size [sectors]	U1	Number of sectors N; where N cannot exceed 8
× M	U1	M ≤ N × Sector size. The missing bytes are interpreted as zero.

Validate Section

The Validate Section command contains the content size, version, and CRC of the section. The following rules apply:

- It is applicable to all sections except the data section
- It is applicable only to invalid sections (cleared using the clear section command)
- If the verification of the CRC on the expected memory size within the section passes, then all the remaining bytes are set to 0. The version and size are stored in the table of the section and the section is declared valid.
- For code sections, the `Content Version` is the software version

Table 7-37: Validate Section Command - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	NV Storage medium ID
Command ID	U2	0x24
Version	U4	1
Reserved2	U4	0
Payload Size	U4	16

Table 7-38: Validate Section Command - Payload

Field Name	Type	Notes
Section ID	U4	Section ID to validate
Size [sectors]	U4	

Table 7-38: Validate Section Command - Payload (Continued)

Field Name	Type	Notes
Content Version	U4	
CRC	U4	

Table 7-39: Validate Section Response- Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	NV storage medium ID
Command ID	U2	0x24
Version	U4	1
Reserved2	U4	0
Payload Size	U4	4
Result Status	U4	0

Table 7-40: Validate Section Response - Payload

Field Name	Type	Notes
Computed CRC	U4	

Create/Recover Restore Point

A restore point consists of a “code plus data” bundle; an application code plus its initialization data.

The application can have one or more restore points (only one in the current implementation). When only one restore point is available, create a restore point of the factory configuration (application code and initialization data saved during production). When the product must be commissioned, create a restore point after commissioning. The benefit of the restore point is to create the possibility, at any time, to revert to a previous software / configuration when either the update or the initialization data modification failed.

C++ API:

```
void
SnapSensor::SensorConnection::ApplicationModule::CreateRestorePoint(
unsigned int RestorePointID)
```

```
void
SnapSensor::SensorConnection::ApplicationModule::RecoverRestorePoint(
unsigned int RestorePointID)
```

Table 7-41: Create/Recover Restore Point Command - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x2A
Version	U4	1
Reserved2	U4	0
Payload Size	U4	2
Reserved3	U4	0

Table 7-42: Create/Recover Restore Point Command - Payload

Field Name	Type	Notes
Create/Recover	U1	1 – Create 2 – Recover
Restore Point ID	U1	0 - N

Table 7-43: Create/Recover Restore Point Response - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x2A
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0
Result Status	U4	

Application Firmware

The application firmware implements the functionality required by the customer product. The simplest applications interface the sensors and upload the sensor data to a host PC or data router using a link (USB, UART, Network). The most complex applications also perform some processing (edge processing in the IOT terminology) and upload processed data to a host PC.

Main Application Messages

The main application module implements the commands shown in the ***Main Application Messages*** table. The availability column indicates the applicable modes for the message.

Table 7-44: Main Application Messages

ID	Command	Notes	Availability
0x01	Reset	Device is reset	All modes
0x02	Reserved		
0x03	Module SW Version	Equivalent to the bootloader command.	All modes
0x10	Get Device Info	Equivalent to the bootloader command.	All modes
0x11	Get Mode	Requests the framework mode	All modes
0x12	Set Mode	Sets the framework mode	All modes
0x15	Store/Restore Parameters to Flash	The framework initiates a procedure to store all the module parameters to the primary storage device.	Configuration mode
0x16	Get Production Parameters		All modes
0x17	Set Production Parameters		Configuration mode
0x20	Get Sections Info	Equivalent to the bootloader command.	Configuration mode
0x21	Clear Sections	Equivalent to the bootloader command.	Configuration mode
0x22	Read Section	Equivalent to the bootloader command.	Configuration mode
0x23	Write Section	Equivalent to the bootloader command.	Configuration mode
0x24	Validate Section	Equivalent to the bootloader command.	Configuration mode
0x2A	Create/Recover Restore Point	Equivalent to the bootloader command.	
0x40	Get Time	Requests the local time	All modes
0x41	Set Time	Sets the local time	All modes
0x60	Get LED Mode	Returns the mode of the board LEDs	All modes
0x61	Set LED Mode	Sets the mode of the board LEDs	All modes
0x70	Get ADC Reading		Configuration mode
0x100	Get Processor Register		Configuration mode
0x101	Set Processor Register		Configuration mode
0x102	External Memory Test		Configuration mode

Table 7-44: Main Application Messages (Continued)

ID	Command	Notes	Availability
0xFF00	Debug Message	Debug message to test the capability of the firmware to recover from software and hardware errors.	All modes (only on the development debug version of the software)

Get Mode

The Get Mode command requests the application mode. The default mode is Smart Camera mode.

C++ API:

```
void SnapSensor::SensorConnection::ApplicationModule::GetMode (SNAP_UINT32
*modeId)
```

Table 7-45: Get Mode Header - Command

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x11
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0
Reserved3	U4	

Table 7-46: Get Mode Header - Response

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x11
Version	U4	1
Reserved2	U4	0
Payload Size	U4	4
Result Status	U4	0

Table 7-47: Get Mode Response - Payload

Field Name	Type	Notes
Mode	U4	0 – Sensor Mode 1– Configuration Mode 2 – Smart Camera Mode

Set Mode

The Set Mode command sets the application mode. In some conditions, the operating mode cannot be set (for example, because the mode does not exist or because of a pending operation). The reason is always coded in the result status. The default mode is Smart Camera mode.

C++ API:

```
void SnapSensor::SensorConnection::ApplicationModule::SetMode (SNAP_UINT32
modeId)
```

Table 7-48: Set Mode Header - Command

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x12
Version	U4	1
Reserved2	U4	0
Payload Size	U4	4
Reserved3	U4	

Table 7-49: Set Mode Command - Payload

Field Name	Type	Notes
Mode	U4	0 – Sensor Mode 1 – Configuration Mode 2 – Smart Camera Mode

Table 7-50: Set Mode Header - Response

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x12

Table 7-50: Set Mode Header - Response (Continued)

Field Name	Type	Notes
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0
Result Status	U4	0 – mode was successfully changed 0x20 – invalid mode (for example, production modes normally do not implement the smart camera mode) 0x30 – mode could not be changed because of a pending operation (for example, uploading the `.dll`)

Store/Restore Parameters to Flash

The `StoreParametersToFlash()` and `RestoreParametersFromFlash()` commands are active in configuration mode. The commands have two purposes:

- Avoid continuous access of the data storage device during the tuning operations
- Confirm or abort changes

These commands shall be sent to the framework to confirm or abort changes by storing to the storage device or restoring from the storage device.

C++ API:

```
void
SnapSensor::SensorConnection::ApplicationModule::StoreParametersToFlash()
```

```
void
SnapSensor::SensorConnection::ApplicationModule::
RestoreParametersFromFlash()
```

Table 7-51: Store/Restore Parameters to Flash Command - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x15
Version	U4	1
Reserved2	U4	0
Payload Size	U4	1

Table 7-51: Store/Restore Parameters to Flash Command - Header (Continued)

Field Name	Type	Notes
Reserved3	U4	0

Table 7-52: Store/Restore Parameters to Flash Command - Payload

Field Name	Type	Notes
Store/Restore	U1	1 – Store 2 – Restore

Table 7-53: Store/Restore Parameters to Flash Response - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x15
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0
Result Status	U4	0 – Successful 1 – Invalid Version

Get Production Parameters

The Get Production Parameters command returns the FLASH information related to:

- Board ID
- Lens type
- Lens center offset

Lens center offset is computed by subtracting the lens center coordinates from the image center. The X axis of the image is parallel to the image rows and its direction is from left to right. The Y axis of the image is parallel to the image columns and its direction is from top to bottom.

C++ API:

```
void
SnapSensor::SensorConnection::ADIS1700xApplication::
GetProductionParameters (ADIS1700xProductionParameters
*productionParameters)
```

Table 7-54: Get Production Parameters Command - Header

Field Name	Bytes	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x16
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0
Reserved3	U4	

Table 7-55: Get Production Parameters Response - Header

Field Name	Bytes	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x16
Version	U4	1
Reserved2	U4	0
Payload Size	U4	N
Result Status	U4	0

Table 7-56: Get Production Parameters Response - Payload

Field Name	Type	Notes
Board ID	U4	
Lens SKU	U8	
Lens Offset X	I1	Offset in pixels
Lens Offset Y	I1	Offset in pixels

Set Production Parameters

The Set Production Parameters command is used to store to the FLASH information related to:

- Board ID
- Lens type
- Lens center offset

Lens center offset is computed by subtracting the lens center coordinates from the image center. The X axis of the image is parallel to the image rows and its direction is from left to right. The Y axis of the image is parallel to the image columns and its direction is from top to bottom.

C++ API:

```
void
SnapSensor::SensorConnection::ADIS1700xApplication::
SetProductionParameters (ADIS1700xProductionParameters
*productionParameters)
```

Table 7-57: Set Production Parameters Command - Header

Field Name	Bytes	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x17
Version	U4	1
Reserved2	U4	0
Payload Size	U4	18
Reserved3	U4	

Table 7-58: Set Production Parameters Response - Header

Field Name	Bytes	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x17
Version	U4	1
Reserved2	U4	0
Payload Size	U4	N
Result Status	U4	0

Table 7-59: Set Production Parameters Response - Payload

Field Name	Type	Notes
Board ID	U4	
Lens SKU	U8	
Lens Offset X	I1	Offset in pixels
Lens Offset Y	I1	Offset in pixels

Get Time

The Get Time command requests the local time. The time is provided as the number of seconds elapsed since January 1, 2000. When this number of seconds is expressed using an unsigned 32 bit, the number overflows in 2135.

C++ API:

```
void SnapSensor::SensorConnection::ApplicationModule::GetTime ( TimeSpan *
sinceY2000 )
```

Table 7-60: Get Time Command - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x40
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0
Reserved3	U4	

Table 7-61: Get Time Response - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x40
Version	U4	1
Reserved2	U4	0
Payload Size	U4	8
Result Status	U4	0

Table 7-62: Get Time Response - Payload

Field Name	Type	Notes
SecondsSince Y2000	U4	Seconds since 1/1/2000 at 00:00:00
Milliseconds	U4	

Set Time

The Set Time command sets the local time. The time is provided as the number of seconds elapsed since January 1, 2000. When this number of seconds is expressed using an unsigned 32 bit, the number overflows in 2135.

C++ API:

```
void SnapSensor::SensorConnection::ApplicationModule::SetTime ( TimeSpan
const * sinceY2000 )
```

Table 7-63: Set Time Command - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x41
Version	U4	1
Reserved2	U4	0
Payload Size	U4	8
Reserved3	U4	

Table 7-64: Set Time Command - Payload

Field Name	Type	Notes
SecondsSince Y2000	U4	Seconds since 1/1/2000 at 00:00:00
Milliseconds	U4	

Table 7-65: Set Time Response - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x41
Version	U4	1
Reserved2	U4	0
Payload Size	U4	1
Result Status	U4	0 if the time was successfully changed

Get LED Mode

The Get LED Mode command returns the mode of the board LEDs. Several modes can be configured. Multiple LEDs can be configured with one single message. See the Table 7-66 LED States table.

The Get LED Mode reports the current LED mode, the LED state and the LED light state.

The device firmware sets LEDs to on or off by switching the LED state. The host can read bit 6 of the `Mode` field in the Get LEDs Mode response to know the current LED state. But, the host cannot change the bit. The host can change the LED light state by overriding the LED state using modes 2 to 4.

When the mode is nominal fixed or nominal blinking, the LED light state is equal to the LED state (the LED is driven by the device).

C++ API:

```
void
SnapSensor::SensorConnection::ADIS1700xApplication::GetLedStatus(
LedStatus * ledStatus, SNAP_UINT16 statusCount, SNAP_UINT16 *
receivedStatusCount)
```

Table 7-66: LED States

INPUT		OUTPUT	
LED State **Device R/W** **Host Read only**	**Mode**	**Light State**	**Note**
0 Off	0: Nominal	0 Off	Led is off
1 On	0: Nominal	1 On	Led is on
0 Off	1: Nominal blinking	0 Off	Led is off
1 On	1: Nominal blinking	1 On	Led is blinking
N/A	2: Override always off	0 Off	Led is off
N/A	3: Override always on	1 On	Led is on
N/A	4: Override always blinking	1 On	Led is blinking

Table 7-67: Get LED Mode Command - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x60
Version	U4	1
Reserved2	U4	0
Payload Size	U4	2
Reserved3	U4	0

Table 7-68: Get LED Mode Command - Payload

Field Name	Type	Notes
Num LEDs	U2	N = 1– 32
LEDS IDs	N×U1	

Table 7-69: Get LED Mode Response - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x60
Version	U4	1
Reserved2	U4	0
Payload Size	U4	2 + 2 × N
Result Status	U4	0x40 – Invalid LED ID

Table 7-70: Get LED Mode Response - Payload

Field Name		Type	Notes
Num LEDs		U2	N = 1– 32
N× LED	ID	U1	0 – 255
	Mode	U1	Bits 0 (LSB) to 2 are for the mode 0 : Nominal (fixed light) when on 1 : Blinking when on 2 : Always off (override) 3 : Always on (override) 4 : Always blinking (override) Bit 6 – Led state is on Bit 7 (MSB) – Led light on/blinks

Table 7-71: LED ID Description

LED Name	ID	Notes
Status LEDs	0	

Set LED Mode

The Set LED command sets the mode of the board LEDs. Several different modes can be configured. Multiple LEDs can be configured with one single message. See the Table 7-66 LED States table.

The device firmware sets LEDs to on or off by switching the LED state. The host can read bit 6 of the `Mode` field in the Get LEDs Mode response to know the current LED state. But, the host cannot change the bit. The host can change the LED light state by overriding the LED state using modes 2 to 4.

When the mode is nominal fixed or nominal blinking, the LED light state is equal to the LED state (the LED is driven by the device).

C++ API:

```
void SnapSensor::SensorConnection::ADIS1700xApplication::SetLedStatus
(LedStatus * ledStatus, SNAP_UINT16 statusCount)
```

Table 7-72: Set LED Mode Command - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x61
Version	U4	1
Reserved2	U4	0
Payload Size	U4	2
Reserved3	U4	0

Table 7-73: Set LED Mode Command - Payload

Field Name	Type	Notes
Num LEDs	U2	N = 1– 32
LEDS IDs	N×U1	

Table 7-74: Set LED Mode Response - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x61
Version	U4	1
Reserved2	U4	0
Payload Size	U4	2 + 2 × N
Result Status	U4	0x40 – Invalid LED ID

Table 7-75: Set LED Mode Response - Payload

Field Name	Type	Notes
Num LEDs	U2	N = 1– 32

Table 7-75: Set LED Mode Response - Payload (Continued)

Field Name		Type	Notes
N× LED	ID	U1	0 – 255
	Mode	U1	Bits 0 (LSB) to 2 are for the mode 0 : Nominal (fixed light) when on 1 : Blinking when on 2 : Always off (override) 3 : Always on (override) 4 : Always blinking (override) Bit 6 – Led state is on Bit 7 (MSB) – Led light on/blinks

Get ADC Reading

The Get ADC Reading command returns voltage readings on specified ADC channels.

C++ API:

```
void
SnapSensor::SensorConnection::ADIS1700xApplication::GetAdcReading(
AdcReading *readings, SNAP_UINT16 channelCount, SNAP_UINT16
*receivedChannelCount)
```

Table 7-76: Get ADC Reading Command - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x70
Version	U4	1
Reserved2	U4	0
Payload Size	U4	2 × N + 2
Reserved3	U4	0

Table 7-77: Get ADC Reading Command - Payload

Field Name	Type	Notes
Num Channels	U2	N ≤ 4
Channel ID	N × U2	

Table 7-78: Get ADC Reading Response - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x70
Version	U4	1
Reserved2	U4	0
Payload Size	U4	2 × N + 2
Result Status	U4	0x20 Invalid Channel ID

Table 7-79: Get ADC Reading Response - Payload

Field Name		Type	Notes
Number of Channels		U2	N, number of channels to read (N ≤ 4)
N × Reading	ID	U2	Channel ID
	Value	U2	Voltage in mV

Get Processor Register

The Get Processor Register command returns the values of a list of processor registers. An address identifies each register. This command can access only memory-mapped registers..

Table 7-80: Get Processor Register Command - Header

Field Name	Type	Value
Module ID	U1	0x01
Reserved1	U1	0
Command ID	U2	0x100
Version	U4	1
Reserved2	U4	0
Payload Size	U4	4 + 8N
Reserved3	U4	

Table 7-81: Get Processor Register Command - Payload

Field Name		Type	Value	Notes
Number of Registers to Read		U4	N	N ≤ 32
× N	Register Address	U4		The value field is disregarded.
	Value	U4		

Table 7-82: Get Processor Register Response - Header

Field Name	Type	Value	Notes
Module ID	U1	0x01	
Reserved1	U1	0	
Command ID	U2	0x100	
Version	U4	1	
Reserved2	U4	0	
Payload Size	U4	4 +8N	
Result Status	U4		0 if the registers are successfully read

Table 7-83: Get Processor Register Response - Payload

Field Name		Type	Value	Notes
Number of Registers		U4	N	N ≤ 32
× N	Register Address	U4		N is the number of registers to read.
	Value	U4		

Set Processor Register

The Set Processor Register command sets the value of a list of memory mapped processor registers.

WARNING: Improper usage of this command can damage the processor or devices connected to it.

Table 7-84: Set Processor Register Command - Header

Field Name	Type	Value
Module ID	U1	0x01
Reserved1	U1	0
Command ID	U2	0x101
Version	U4	1
Reserved2	U4	0
Payload Size	U4	8N + 4
Reserved3	U4	

Table 7-85: Set Processor Register Command - Payload

Field Name		Type	Value	Notes
Number of Registers		U4	N	N ≤ 32
× N	Register Address	U4		N is the number of registers to be read.
	Value	U4		

Table 7-86: Set Processor Register Response - Header

Field Name	Type	Value	Notes
Module ID	U1	0x01	
Reserved1	U1	0	
Command ID	U2	0x101	
Version	U4	1	
Reserved2	U4	0	
Payload Size	U4	0	
Result Status	U4		0 if the registers have been successfully written

External Memory Test

The External Memory Test command performs a memory test and returns the status of the test.

Table 7-87: External Memory Test Command - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0x102
Version	U4	1
Reserved2	U4	0
Payload Size	U4	2
Reserved3	U4	0

Table 7-88: External Memory Test Command - Payload

Field Name	Type	Notes
Memory Type	U1	1 LPDDR 2 FLASH
Test Type	U1	1 Data Bus Test 2 Address Bus Test 3 Device Test

Table 7-89: External Memory Test Response - Header

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0

Table 7-89: External Memory Test Response - Header (Continued)

Field Name	Type	Notes
Command ID	U2	0x102
Version	U4	1
Reserved2	U4	0
Payload Size	U4	8
Result Status	U4	0

Table 7-90: External Memory Test Response - Payload

Field Name	Type	Notes
Memory Type	U2	1 LPDDR 2 FLASH
Test Type	U1	1 Data Bus Test 2 Address Bus Test 3 Device Test
Test Result	U2	0 if the test is successful
Failure Info	U4	

Debug Message

The Debug Message command injects errors into the application to test its capability to recover. This command is available only in the debug versions of the software.

Table 7-91: Debug Message Header - Command

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0xFF00
Version	U4	1
Reserved2	U4	0
Payload Size	U4	4
Reserved3	U4	0

Table 7-92: Debug Message Command - Payload

Field Name	Type	Notes
Command Code	U4	See the Table 7-94 Debug Message Command Codes table

Table 7-93: Debug Message Header - Response

Field Name	Type	Notes
Module ID	U1	1
Reserved1	U1	0
Command ID	U2	0xFF00
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0
Reserved3	U4	0

Table 7-94: Debug Message Command Codes

Command Code	Name	Notes
0	Infinite Loop	When the firmware receives this command, it starts an infinite loop waiting for the watchdog to expire and reset the device.
1	Camera Failure	Forces a camera failure. The application resets and restarts the camera.
2	IMU Failure	Forces an IMU failure. The application resets and restarts the IMU.

Camera Module Messages

The camera module implements the messages shown in the ***Camera Module Commands*** table.

Table 7-95: Camera Module Commands

ID	Command	Notes	Availability
0x03	Module SW Version	Equivalent to the bootloader command	All modes
0x11	Get Luminance Image	Returns an image from the camera	All modes
0x12	Set Luminance Image	Sets the image in the camera. The camera is halted. After this time images are provided through the data link. To exit from this mode, a reset command must be sent to the camera module.	Sensor Mode
0x20	Get ADVS200x Tuning Parameters	Returns the parameter values of the ADVS200x imager	Configuration mode
0x21	Set ADVS200x Tuning Parameters	Sets the parameter values of the ADVS200x imager	Configuration mode
0x22	Get ADVS200x LUT	Returns the values of the look-up table	Configuration mode
0x23	Set ADVS200x LUT	Sets the values of the look-up table	Configuration mode
0x80	Self Test		Configuration mode

Get Luminance Image

This message is used to get the image from the camera module. The image is received in chunks. The size of the chunks and the number of necessary chunks is determined by the client (the camera module in the framework). The image must to be packed. The number of chunks to request is indicated by the total chunks field in the response. The upload sequence can be restarted at any time by sending a new Get Luminance Image command with the chunk index set to 1. When not in Sensor mode, the host must quickly request the remaining chunks to avoid data corruption. In Smart Camera mode and in Configuration mode, sending a Get Luminance Image command with chunk index set to 1 initiates a new capture.

The response timeout for this command must be set to two times the inverse of the frame rate.

Every chunk must contain an integer number of pixels. However, the size of the last chunk can be different. The chunk size must indicate the size of the transmitted chunk. Image is transmitted first row first.

There are two versions of this message that are implemented. The first version is the legacy one, the second is the new one which includes a time tag.

C++ API:

```
void SnapSensor::SensorConnection::CameraModule::GetLuminanceImage (Image
dstImage, SNAP_UINT32 *frameIndex, TimeSpan *frameTime)
```

Table 7-96: Get Luminance Image Command - Header

Field Name	Type	Notes
Module ID	U1	0x10
Reserved1	U1	0
Command ID	U2	0x11
Version	U4	1
Reserved2	U4	0
Payload Size	U4	2
Reserved3	U4	0

Table 7-97: Get Luminance Image Command - Payload

Field Name	Type	Notes
Chunk Index	U4	1 – Total Chunks

Table 7-98: Get Luminance Image Response - Header

Field Name	Type	Notes
Module ID	U1	0x10
Reserved1	U1	0

Table 7-98: Get Luminance Image Response - Header (Continued)

Field Name	Type	Notes
Command ID	U2	0x11
Version	U4	1
Reserved2	U4	0
Payload Size	U4	N + 17
Result Status	U4	0x20 –Timeout. The buffer is no longer available. (The buffer overwritten by the camera while downloading the image). 0x30 – Invalid chunk index

Table 7-99: Get Luminance Image Response - Payload

Field Name		Type	Notes
Frame Index		U4	
Image Width		U2	Size of each row in pixels.
Image Height		U2	Number of rows in the final image
Chunk Size		U4	N in bytes
Chunk Index		U2	
Total Chunks		U2	
Bits per Pixel		U1	
× N	Data	U1	

Table 7-100: Get Luminance Image Command - Header (V2)

Field Name	Type	Notes
Module ID	U1	0x10
Reserved1	U1	0
Command ID	U2	0x11
Version	U4	2
Reserved2	U4	0
Payload Size	U4	2
Reserved3	U4	0

Table 7-101: Get Luminance Image Command - Payload (V2)

Field Name	Type	Notes
Chunk Index	U4	1 – Total Chunks

Table 7-102: Get Luminance Image Response - Header (V2)

Field Name	Type	Notes
Module ID	U1	0x10
Reserved1	U1	0
Command ID	U2	0x11
Version	U4	2
Reserved2	U4	0
Payload Size	U4	N + 17
Result Status	U4	0x20 –Timeout. The buffer is no longer available. (The buffer overwritten by the camera while downloading the image). 0x30 – Invalid chunk index

Table 7-103: Get Luminance Image Response - Payload (V2)

Field Name		Type	Notes
Frame Index		U4	
Reserved		U2	
DaysSince2000		U2	0..65535 (2000 … 2179)
Time of Day		U4	0…86400s with a resolution of 0.1 ms
Image Width		U2	Size of each row in pixels.
Image Height		U2	Number of rows in the final image
Chunk Size		U4	N in bytes
Chunk Index		U2	
Total Chunks		U2	
Bits per Pixel		U1	
× N	Data	U1	

Set Luminance Image

This command is used to set the image in the camera module. Once the message is received, the camera module switches to virtual camera mode. From this moment new images can only come from the host through this command. To exit from virtual camera mode, reset the module.

The image size must match the image size expected by the application. The image is transmitted in chunks. The image must be packed. Every chunk must contain an integer number of pixels. But, the size of the last chunk can be different. The chunk size must indicate the size of the transmitted chunk.

C++ API:

```
void SnapSensor::SensorConnection::CameraModule::SetLuminanceImage(Image
srcImage, SNAP_INT32 srcBitsPerPixel, SNAP_UINT32 srcFrameIndex)
```

Table 7-104: Set Luminance Image Command - Header

Field Name	Type	Notes
Module ID	U1	0x10
Reserved1	U1	0
Command ID	U2	0x12
Version	U4	1
Reserved2	U4	0
Payload Size	U4	N + 17
Reserved3	U4	

Table 7-105: Set Luminance Image Command - Payload

Field Name		Type	Notes
Frame Index		U4	
Image Width		U2	Size of each row in pixels
Image Height		U2	Number of rows in the final image
Chunk Size		U4	N in bytes
Chunk Index		U2	
Total Chunks		U2	
Bits per Pixel		U1	
× N	Data	U1	

Table 7-106: Set Luminance Image Response - Header

Field Name	Type	Notes
Module ID	U1	0x10
Reserved1	U1	0
Command ID	U2	0x12
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0

Table 7-106: Set Luminance Image Response - Header (Continued)

Field Name	Type	Notes
Result Status	U4	0x20 – Invalid image size. (The image size must match the image size expected by the application). If this is not the case, the device flags the error and ignores the message content. 0x30 – Unexpected chunk index. (This error is caused by a missing index). A chunk index of 0 indicates that the host wants to restart the image uploading. This operation is valid and no error is flagged.

Get ADVS200x Tuning Parameters

This command is sent to read the values of ADVS200x imager parameters. The IDs of the parameters to be read are sent in the command payload. The command and response payloads have a similar structure.

C++ API:

```
void SnapSensor::SensorConnection::CameraModule:: GetADVS200xParameters
(IADVS200xRegisters *dstADVS200xRegisters, IADVS200xLut *dstADVS200xLut)
```

Table 7-107: Get Tuning Parameters Command - Header

Field Name	Type	Value
Module ID	U1	0x10
Reserved1	U1	0
Command ID	U2	0x20
Version	U4	1
Reserved2	U4	0
Payload Size	U4	4 + 8N
Reserved3	U4	

Table 7-108: Get Tuning Parameters Command - Payload

Field Name		Type	Value	Notes
Num of Parameters		U4	N	
x N	Param ID	U4		N is the number of parameters to read. The parameter value is disregarded. The parameter ID contains the ID of the parameters to read. The last parameter ID is the null parameter (Parameter ID = 0xFFFF) which is used as a terminator.
	Value	U4		

Table 7-109: Get Tuning Parameters Response- Header

Field Name	Type	Value	Notes
Module ID	U1	0x10	
Reserved1	U1	0	
Command ID	U2	0x20	
Version	U4	1	
Reserved2	U4	0	
Payload Size	U4	4 +8N	
Result Status	U4	0	0 if the parameters have been successfully read

Table 7-110: Get Tuning Parameters Response - Payload

Field Name		Type	Value	Notes
Number of Parameters		U4	N	
× N	Param ID	U4		N is the number of parameters read. The last parameter is the terminator.
	Value	U4		

Set ADVS200x Tuning Parameters

The Set ADVS200x Tuning Parameters command sets the parameter values for the ADVS200x imager. The IDs of the parameters and their respective values are sent in the command payload. The response does not have a payload.

C++ API:

```
void SnapSensor::SensorConnection::CameraModule:: SetADVS200xParameters
(IADVS200xRegisters *srcADVS200xRegisters, IADVS200xLut *srcADVS200xLut)
```

Table 7-111: Set Tuning Parameters Command - Header

Field Name	Type	Value
Module ID	U1	0x10
Reserved1	U1	0
Command ID	U2	0x21
Version	U4	1
Reserved2	U4	0
Payload Size	U4	8N + 4
Reserved3		

Table 7-112: Get Tuning Parameters Command - Payload

Field Name		Type	Value	Notes
Num of Parameters		U4	N	
x N	Param ID	U4		N is the number of parameters to read. Each byte octet is a couple of 32-bit integers representing the parameter ID and its value. The last octet is the null parameter (Parameter ID = 0xFFFF) which is used as a terminator.
	Value	U4		

Table 7-113: Get Tuning Parameters Response- Header

Field Name	Type	Value	Notes
Module ID	U1	0x10	
Reserved1	U1	0	
Command ID	U2	0x21	
Version	U4	1	
Reserved2	U4	0	
Payload Size	U4	0	
Result Status	U4		0 if the parameters have been successfully read

Get ADVS200x LUT

The Get ADVS200x LUT command reads the values of the ADVS200x lookup table.

C++ API:

```
void SnapSensor::SensorConnection::CameraModule:: GetADVS200xLut
(IADVS200xLut *dstADVS200xLut)
```

Table 7-114: Get ADVS200x LUT Command - Header

Field Name	Type	Value
Module ID	U1	0x10
Reserved1	U1	0
Command ID	U2	0x22
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0

Table 7-115: Get ADVS200x LUT Response - Header

Field Name	Type	Value	Notes
Module ID	U1	0x10	
Reserved1	U1	0	
Command ID	U2	0x22	
Version	U4	1	
Reserved2	U4	0	
Payload Size	U4	4x1024	
Result Status	U4		0 if the LUT has been successfully read

Table 7-116: Get ADVS200x LUT Response - Payload

Field Name	Type	Notes
Lookup Table	U4x1024	The 1024 values of the lookup table

Set ADVS200x LUT

The `SetADVS200xLUT` command writes the values of the ADVS200x lookup table.

C++ API:

```
void SnapSensor::SensorConnection::CameraModule:: SetADVS200xLut
(IADVS200xLut *srcADVS200xLut)
```

Table 7-117: Set ADVS200x LUT Command - Header

Field Name	Type	Value
Module ID	U1	0x10
Reserved1	U1	0
Command ID	U2	0x23
Version	U4	1
Reserved2	U4	0
Payload Size	U4	4 x 1024

Table 7-118: Set ADVS200x LUT Command - Payload

Field Name	Type	Value	Notes
Lookup table	U4 x 1024		

Table 7-119: Set ADVS200x LUT Response- Header

Field Name	Type	Value	Notes
Module ID	U1	0x10	
Reserved1	U1	0	
Command ID	U2	0x23	
Version	U4	1	
Reserved2	U4	0	
Payload Size	U4	0	
Result Status	U4		0 if the LUT has been successfully set.

Self Test

The Self Test command runs the defined sequence of electrical tests. The response reports availability and result of each test.

The *Self Test ID Descriptions* table specifies the available self tests.

Self Test ID Descriptions

Test ID	Test Name	Description	Result Type	Notes
1	ADVS200x Reset	Modifies the first and last entry of the LUT. Resets the ADVS200x. Reads the same two entries and verifies that they match the default ADVS200x LUT values.	Result: Pass(0)/Fail(1) Result Value: 0	
2	PPI Port	Uses the ADVS200x PPI test mode. Verifies that each of the 13 pins of the PPI port can be set to 0 or 1 while all the other pins are set to the complementary value. This requires 26 tests.	Result: Pass(0)/Fail(n) Result Value: Test word that failed if any.	n > 0 is the subtest which failed if any. (n-1)%13 gives the pin which failed where PPI pins 0..9 are the data pins while pins 10..12 are H.Sync V.Sync and F.Sync. If n <= 13 (resp. n > 13) the test failed to set the failed pin to 1 (resp. to 0) while all the other PPI pins where set to 0 (resp. to 1).

Test ID	Test Name	Description	Result Type	Notes
3	Trig NIRQ LED Configures the LED pin.	Triggers an image capture. Trigger pin test passes if an image is receive through PPI. Verifies that the host processor receives an interrupt on the GPIO connected to NIRQ at the end of the image transfer. The LED pin test passes if the LED pin goes high for the entire duration of the exposure and Is low otherwise.	Result: Pass(0)/Fail(1) Result Value: (LSB) Trigger P/F (1/0) Bit 1 NIRQ Available Bit 2 NIRQ P/F (1/0) Bit 3 LED Available Bit 4 LED P/F (1/0)	Test is unavailable if trigger pin is not connected. Test passes if the test for the available pins passes.
6	GPIO 0	Subtest 1 : Tries to set this GPIO to 0 while setting all the other available GPIOs to 1 Subtest 2 : Tries to set this GPIO to 1 while setting all the other available GPIOs to 0	Result: Pass(0)/Fail(n) Result Value: Test word that failed if any.	n > 0 is the subtest which failed if any.
7	GPIO 1	Same as GPIO 0	Result: Pass(0)/Fail(n) Result Value: Test word that failed if any.	n > 0 is the subtest which failed if any.
8	GPIO 2	Same as GPIO 0	Result: Pass(0)/Fail(n) Result Value: Test word that failed if any.	n > 0 is the subtest which failed if any.
9	GPIO 3	Same as GPIO 0	Result: Pass(0)/Fail(n) Result Value: Test word that failed if any.	n > 0 is the subtest which failed if any.
10	PPI_DValid	Configures the GPIO connected to the PPI_DVALID signal as an interrupt source. Subtest 1 : Sets an all zeros word in the PPI FIFO and enables the PPI output. Checks that an interrupt has been received. Subtest 2 : Sets a 0x1FFF word in the PPI FIFO and enables the PPI output. Checks that an interrupt has been received.	Result: Pass(0)/Fail(n) Result Value: Test word that failed if any.	n > 0 is the subtest which failed if any.

Table 7-120: Self Test Command - Header

Field Name	Type	Value
Module ID	U1	0x10
Reserved1	U1	0
Command ID	U2	0x80
Version	U4	1
Reserved2	U4	0
Payload Size	U4	4 x N

Table 7-121: Self Test Command - Payload

Field Name		Type	Value	Notes
Test Count		U4	N	N ≤ 8
x N	Test ID	U1	1 - 10	Same test ID can be specified multiple times but is executed only once

Table 7-122: Self Test Response- Header

Field Name	Type	Value	Notes
Module ID	U1	0x10	
Reserved1	U1	0	
Command ID	U2	0x80	
Version	U4	1	
Reserved2	U4	0	
Payload Size	U4	4 + 4N	
Result Status	U4		0 – all the tests passed 0x20 – one of the tests failed

Table 7-123: Self Test Response - Payload

Field Name		Type	Value	Notes
Test Count		U4	N	
x N	Test ID	U1		
	Test Result	U1		0xFF – test not available
	Result Value	U2		

IMU Module Messages

The IMU module implements the messages shown in the ***IMU Module Messages*** table.

Table 7-124: IMU Module Messages

ID	Command	Description	Availabilty
0x11	Get Measurements	Retrieves IMU sample measurements from the IMU module	Sensor/Smart Camera mode
0x15	Get Temperature		For future use. (Sensor/Smart Camera mode)
0x20	Get Configuration	Retrieve the IMU configuration including the number of available axis, the measurement rate and the range for each axis	All modes
0x21	Set Configuration		Configuration mode
0x24	Get Parameters	Reads the IMU parameter values	Configuration mode
0x25	Set Parameters	Writes the IMU parameter values	Configuration mode

Get Measurements

The `GetMeasurements` message is used to retrieve IMU sample measurements from the IMU module.

The sample format field allows the user to define the structure of the measurement sample. Attempts to request an axis that is not present returns in an *invalid format* in the results status field of response message. In case of invalid format, the response message returns no samples but the sample format field indicates the measurements that are available.

There are two possible ways to arrange the data:

- Structure – a matrix arrangement where all the data in the same row belongs to the same epoch and the matrix is transmitted row-wise, first row first.
- Vectors – a matrix arrangement where all the data in the same row are homogeneous (for example, the first row contains the accelerometer X axis measurements). The vectors arrangement is the transpose of the structure arrangement.

The user can request an arbitrary number of samples using the homonymous field. The response message can return a smaller number of samples when the number indicated in the command exceeds the number of available measurements. When this happens, no error is reported in the result status of the response message. The number of samples field indicates the effective number of measurements in the message.

C++ API:

```
void SnapSensor::SensorConnection::ImuModule::GetMeasurements (SNAP_UINT16
measurementCount, SNAP_UINT32 *timeTag, SNAP_INT32 *accelerometerX,
SNAP_INT32 *accelerometerY, SNAP_INT32 *accelerometerZ, SNAP_UINT16
```

```
*returnedMeasurementCount, SNAP_UINT16 *lastMeasurementId, TimeSpan
*lastMeasurementTime)
```

Table 7-125: Get Measurements Command - Header

Field Name	Type	Notes
Module ID	U1	0x12
Reserved1	U1	0
Command ID	U2	0x11
Version	U4	2
Reserved2	U4	0
Payload Size	U4	4
Result	U4	0

Table 7-126: Get Measurements Command - Payload

Field Name	Type	Notes
Format	U2	See the Table 7-129 Get Measurement Format table
Number of Samples	U2	1– 1000

Table 7-127: Get Measurements Response - Header

Field Name	Type	Notes
Module ID	U1	0x12
Reserved1	U1	0
Command ID	U2	0x11
Version	U4	2
Reserved2	U4	0
Payload Size	U4	12 + N × M × 2 + M × 4
Result Status	U4	0x50 – Invalid Format

Table 7-128: Get Measurements Response - Payload

Field Name	Type	Notes
Format	U2	See the Table 7-129 Get Measurement Format table
Number of Samples	U2	N = 0 – 1000
Last Meas. ID	U2	0 – 65535
DaysSince2000	U2	0 – 65535 (years 2000 to 2179)

Table 7-128: Get Measurements Response - Payload (Continued)

Field Name	Type	Notes
Time Of Day	U4	0 – 86400s with resolution 0.1 ms
IMU Measurements	N × M	See theTable 7-129 Get Measurement Format table. M = size of requested data. Acceleration and angular rates are two bytes long while the time tag field is four bytes long. Data are sorted oldest first.

Table 7-129: Get Measurement Format

Bit Field Name	Bit	Notes
Time Tag	0	Set to 1 when the measurement is required
Accelerometer X	1	Set to 1 when the measurement is required
Accelerometer Y	2	Set to 1 when the measurement is required
Accelerometer Z	3	Set to 1 when the measurement is required
Structure/Vectors	15 (MSB)	Set to 1 when structure format is desired. (See the *Structure Format Example*). Set to 0 when vectors format is desired. (See the *Vector Format Example*).

Structure Format Example

The following code example show the IMU measurement using the structure format for:

Bitfield = 1+2+4+(1<<15)

```
ImuMeasurements = {
       TimeTag[0],
       Accelerometer X[0],
       Accelerometer Y[0],
       ……
       TimeTag[N-1],
       Accelerometer X[N-1],
       Accelerometer Y[N-1],
 };
```

Vector Format Example

The following code example show the IMU measurement using the vector format for:

Bitfield = 1+2+4+(1<<15)

```
ImuMeasurements = {
```

```
    TimeTag[0],
    ……
    TimeTag[N-1],
    Accelerometer X[0],
    ……
    Accelerometer X[N-1],
    Accelerometer Y[0],
    ……
    Accelerometer Y[N-1],
};
```

Get Configuration

The Get Configuration command is used to retrieve the IMU configuration. In particular, the command gets the number of available axes, the measurement rate, the range for each axis.

C++ API:

```
void SnapSensor::SensorConnection::ImuModule::
GetConfiguration(ImuConfiguration *imuConfiguration)
```

Table 7-130: Get Configuration Command - Header

Field Name	Type	Notes
Module ID	U1	0x12
Reserved1	U1	0
Command ID	U2	0x20
Version	U4	3
Reserved2	U4	0
Payload Size	U4	0
Reserved3	U4	0

Table 7-131: Get Configuration Response- Header

Field Name	Type	Notes
Module ID	U1	0x12
Reserved1	U1	0
Command ID	U2	0x20
Version	U4	3
Reserved2	U4	0
Payload Size	U4	88
Result Status	U4	0

Table 7-132: Get Configuration Response - Payload

Field Name	Type	Notes
Accelerometer Sampling Rate	U4	Milli Hertz resolution (mHz)
Accelerometer X axis	U4	Nominal Range (mg) Returns 0 if the measurement is not available
Accelerometer Y axis	U4	Nominal Range (mg) Returns 0 if the measurement is not available
Accelerometer Z axis	U4	Nominal Range (mg) Returns 0 if the measurement is not available

Set Configuration

The Set Configuration command stores the IMU configuration. Information provided for non-existing IMU axes is ignored. The configuration is stored in the device non-volatile memory.

If the user fails to provide an available nominal range, the device software will adjust it to the closest available nominal range.

C++ API:

```
void SnapSensor::SensorConnection::ImuModule::
SetConfiguration(ImuConfiguration *imuConfiguration)
```

Table 7-133: Set Configuration Command - Header

Field Name	Type	Notes
Module ID	U1	0x12
Reserved1	U1	0
Command ID	U2	0x21
Version	U4	3
Reserved2	U4	0
Payload Size	U4	88
Reserved3	U4	0

Table 7-134: Set Configuration Command - Payload

Field Name	Type	Notes
Accelerometer Sampling Rate	U4	Milli Hertz resolution (mHz)
Temperature Sampling Rate	U4	Reserved for future use.
Accelerometer Nominal Range	U2	(g)

Table 7-134: Set Configuration Command - Payload (Continued)

Field Name	Type	Notes
Accelerometer X axis Range	U4	Calibrated range (mg) Returns 0 if the measurement is not available
Accelerometer X axis Offset	I4	Offset (ug) Returns 0 if the measurement is not available or was not set.
Accelerometer Y axis	U4	Calibrated range (mg) Returns 0 if the measurement is not available
Accelerometer Y axis Offset	I4	Offset (ug) Returns 0 if the measurement is not available or was not set.
Accelerometer Z axis	U4	Calibrated range (mg) Returns 0 if the measurement is not available
Accelerometer Z axis Offset	I4	Offset (ug) Returns 0 if the measurement is not available or was not set.
Accelerometer Lower Cut-off Frequency	U4	Milli Hertz resolution (mHz)
Accelerometer Upper Cut-off Frequency	U4	Milli Hertz resolution (mHz)
Accelerometer Temperature Range	I2	Min = 0.1 °C; 0 if not available Max = 0.1 °C; 0 if not available

Table 7-135: Set Configuration Response- Header

Field Name	Type	Notes
Module ID	U1	0x12
Reserved1	U1	0
Command ID	U2	0x21
Version	U4	3
Reserved2	U4	0
Payload Size	U4	88
Result Status	U4	0

Table 7-136: Set Configuration Response - Payload

Field Name	Type	Notes
	U4	Milli Hertz resolution (mHz)

Table 7-136: Set Configuration Response - Payload (Continued)

Field Name	Type	Notes
Accelerometer Sampling Rate		
Temperature Sampling Rate	U4	Reserved for future use.
Accelerometer Nominal Range	U2	(g)
Accelerometer X axis Nominal Range	U4	True range (mg) Returns 0 if the measurement is not available
Accelerometer X axis Offset	I4	Offset (ug) Returns 0 if the measurement is not available or was not set.
Accelerometer Y axis	U4	Calibrated range (mg) Returns 0 if the measurement is not available
Accelerometer Y axis Offset	I4	Offset (ug) Returns 0 if the measurement is not available or was not set.
Accelerometer Z axis	U4	Calibrated range (mg) Returns 0 if the measurement is not available
Accelerometer Z axis Offset	I4	Offset (ug) Returns 0 if the measurement is not available or was not set.
Accelerometer Lower Cut-off Frequency	U4	Milli Hertz resolution (mHz)
Accelerometer Upper Cut-off Frequency	U4	Milli Hertz resolution (mHz)
Accelerometer Temperature Range	I2	Min = 0.1 °C; 0 if not available Max = 0.1 °C; 0 if not available

Get Parameters

The Get Parameters command is sent to read the values of a group of registers for the IMU. The ID of the register to read is sent in the command payload. The command and response payload have similar structures. The meaning of ID is specific to the IMU; it can be a 32-bit address for memory-mapped registers or an entry ID inside a predefined table of registers.

Table 7-137: Get Parameters Command - Header

Field Name	Type	Value	Notes
Module ID	U1	0x12	
Reserved1	U1	0	

Table 7-137: Get Parameters Command - Header (Continued)

Field Name	Type	Value	Notes
Command ID	U2	0x24	
Version	U4	1	
Reserved2	U4	0	
Payload Size	U4	4 + 8N	
Reserved3	U4		

Table 7-138: Get Parameters Command - Payload

Field Name		Type	Value	Notes
Number of Parameters		U4	N	N ≤ 32
x N	Parameter ID	U4		N is the number of parameters to read. The parameter value is disregarded. The parameter ID contains the ID of the parameters to read.
	Value	U4		

Table 7-139: Get Parameters Response - Header

Field Name	Type	Value	Notes
Module ID	U1	0x12	
Reserved1	U1	0	
Command ID	U2	0x24	
Version	U4	1	
Reserved2	U4	0	
Payload Size	U4	4 +8N	
Result Status	U4		0 if the parameters are successfully read

Table 7-140: Get Parameters Response - Payload

Field Name		Type	Value	Notes
Number of Parameters		U4	N	N ≤ 32
× N	Parameter ID	U4		N is the number of parameters read. The last parameter is the terminator.
	Value	U4		

Table 7-141: Register IDs

Name	Name	R/W	Reset Value	Num Bits
0x10000000	DEVID	R	11100101	8
0x1000001D	THRESH_TAP	R/W	00000000	8

Table 7-141: Register IDs (Continued)

Name	Name	R/W	Reset Value	Num Bits
0x1000001E	OFSX	R/W	00000000	8
0x1000001F	OFSY	R/W	00000000	8
0x10000020	OFSZ	R/W	00000000	8
0x10000021	DUR	R/W	00000000	8
0x10000022	Latent	R/W	00000000	8
0x10000023	Window	R/W	00000000	8
0x10000024	THRESH_ACT	R/W	00000000	8
0x10000025	THRESH_INACT	R/W	00000000	8
0x10000026	TIME_INACT	R/W	00000000	8
0x10000027	ACT_INACT_CTL	R/W	00000000	8
0x10000028	THRESH_FF	R/W	00000000	8
0x10000029	TIME_FF	R/W	00000000	8
0x1000002A	TAP_AXES	R/W	00000000	8
0x1000002B	ACT_TAP_STATUS	R	00000000	8
0x1000002C	BW_RATE	R/W	00001010	8
0x1000002D	POWER_CTL	R/W	00000000	8
0x1000002E	INT_ENABLE	R/W	00000000	8
0x1000002F	INT_MAP	R/W	00000000	8
0x10000030	INT_SOURCE	R	00000010	8
0x10000031	DATA_FORMAT	R/W	00000000	8
0x10000032	DATAX	R	00000000	16
0x10000034	DATAY	R	00000000	16
0x10000036	DATAZ	R	00000000	16
0x10000038	FIFO_CTL	R/W	00000000	8
0x10000039	FIFO_STATUS	R	00000000	8

Set Parameters

The Set Parameters command is used to set the values of custom parameters for the IMU sensor. The IDs of the parameters and their respective values are sent in the command payload. The response does not have a payload.

Table 7-142: Set Parameters Command - Header

Field Name	Type	Value
	U1	0x12

Table 7-142: Set Parameters Command - Header (Continued)

Field Name	Type	Value
Module ID		
Reserved1	U1	0
Command ID	U2	0x25
Version	U4	1
Reserved2	U4	0
Payload Size	U4	8N + 4
Reserved3	U4	

Table 7-143: Set Parameters Command - Payload

Field Name		Type	Value	Notes
Number of Parameters		U4	N	N ≤ 32
× N	Parameter ID	U4		N is the number of parameters to be read. Each byte octet is a couple of 32-bit integers representing the parameter ID and its value.
	Value	U4		

Table 7-144: Set Parameters Response - Header

Field Name	Type	Value	Notes
Module ID	U1	0x12	
Reserved1	U1	0	
Command ID	U2	0x25	
Version	U4	1	
Reserved2	U4	0	
Payload Size	U4	4 +8N	
Result Status	U4		0 if the parameters have been successfully written

Storage Module Messages

The storage module implements the commands shown in the ***Storage Commands*** table.

Table 7-145: Storage Commands

ID	Command	Notes	Availability
0x03	Module SW Version	Equivalent to the bootloader command	All modes
0x10	Get Device Info	Returns information related to the non-volatile storage device	All modes

Table 7-145: Storage Commands (Continued)

ID	Command	Notes	Availability
0x20	Get Sections Info	Equivalent to the bootloader command	All modes
0x21	Clear Sections	Equivalent to the bootloader command	Configuration mode/Bootloader mode
0x22	Read Section	Equivalent to the bootloader command	Configuration mode/Bootloader mode
0x23	Write Section	Equivalent to the bootloader command	Configuration mode/Bootloader mode
0x24	Validate Section	Equivalent to the bootloader command	Configuration mode/Bootloader mode
0x31	Get Log Data Summary		All modes
0x32	Get Log Data		All modes
0x33	Clear Log Data		Configuration mode

Get Device Info

The Get Device Info command returns device information such as device serial number (unique device identification), device name and manufacturer, and user-defined metadata.

C++ API:

```
void
SnapSensor::SensorConnection::StorageModule::GetFlashDeviceInfo(
FlashDeviceInfo *deviceInfo)
```

Table 7-146: Get Device Info Command - Header

Field Name	Type	Notes
Module ID	U1	0x40
Reserved1	U1	0
Command ID	U2	0x10
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0
Reserved3	U4	0

Table 7-147: Get Device Info Response - Header

Field Name	Type	Notes
Module ID	U1	0x40
Reserved1	U1	0
Command ID	U2	0x10
Version	U4	1
Reserved2	U4	0
Payload Size	U4	24
Result Status	U4	0

Table 7-148: Get Device Info Response - Payload

Field Name	Type	Notes
Unique ID	16xU8	Unique identifier
Manufacturer ID	U32	Manufacturer unique identifier. See Appendix A
Device ID	U32	Device unique identifier. See Appendix A

Clear Sections

The Clear Sections command sends a request to clear the sections specified in a list.

C++ API:

```
void SnapSensor::SensorConnection::StorageModule::ClearSection
(SNAP_UINT32 sectionId)
```

Table 7-149: Clear Section Command - Header

Field Name	Type	Notes
Module ID	U1	0x40
Reserved1	U1	NV storage medium ID
Command ID	U2	0x21
Version	U4	1
Reserved2	U4	0
Payload Size	U4	4 + 4 × N

Table 7-150: Clear Sections Command - Payload

Field Name		Type	Notes
Num Sections		U4	N
× N	ID (*1)		IDs of sections that are cleared

*1 The number of sections that are cleared can be larger than the ones indicated in the message. The objective of the bootloader is to preserve integrity. For example, clearing the boot section clears up all the sections.

Table 7-151: Clear Sections Response- Header

Field Name	Type	Notes
Module ID	U1	0x40
Reserved1	U1	NV Storage medium ID
Command ID	U2	0x21
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0
Result Status	U4	0

Read Section

Reads a section. Offset and size are expected to be expressed in sectors. The sector size is provided by the Get Sections Info message.

C++ API:

```
void SnapSensor::SensorConnection::StorageModule::ReadSection (void
*dstBuf, SNAP_UINT32 dstLengthBytes, SNAP_UINT32 srcSectionId, SNAP_UINT32
srcOffsetSectors, SNAP_UINT32 srcSectorLengthBytes)
```

Table 7-152: Read Section Command - Header

Field Name	Type	Notes
Module ID	U1	0x40
Reserved1	U1	NV Storage medium ID
Command ID	U2	0x22
Version	U4	1
Reserved2	U4	0
Payload Size	U4	12

Table 7-153: Read Section Command - Payload

Field Name	Type	Notes
Section ID	U4	Section ID to read
Offset [sectors]	U4	Relative offset with respect to the section start
Size [sectors]	U1	N

Table 7-154: Read Section Response- Header

Field Name	Type	Notes
Module ID	U1	0x40
Reserved1	U1	NV storage medium ID
Command ID	U2	0x22
Version	U4	1
Reserved2	U4	0
Payload Size	U4	12 + Size
Result Status	U4	0

Table 7-155: Read Section Response - Payload

Field Name		Type	Notes
Offset [sectors]		U4	
Num Sections		U4	N; where N cannot exceed 8
M	Data	U1	M = N × Sector Size

Write Section

The Write Section command writes a section. Each response returns the data that was read back after the write operation (cache disabled). Whenever the bootloader receives a `Write Section` command, it first invalidates the section and then starts writing. The section is validated after a successful reception of the Validate Section command. Offset and size are expected to be expressed in sectors. The sector size is provided by Get Sections Info message.

C++ API:

```
void SnapSensor::SensorConnection::StorageModule::WriteSection (void
*srcBuf, SNAP_UINT32 srcLengthBytes, SNAP_UINT32 dstSectionId, SNAP_UINT32
dstSectorOffset, SNAP_UINT32 dstSectorLengthBytes)
```

Table 7-156: Write Section Command - Header

Field Name	Type	Notes
Module ID	U1	0x40
Reserved1	U1	NV storage medium ID
Command ID	U2	0x23
Version	U4	1
Reserved2	U4	0
Payload Size	U4	4108

Table 7-157: Write Section Command - Payload

Field Name		Type	Notes
Section ID		U4	Section ID to write
Offset		U4	Relative offset with respect to the section start
Size		U4	N
× N	Data	U1	

Table 7-158: Write Section Response- Header

Field Name	Type	Notes
Module ID	U1	0x40
Reserved1	U1	NV storage medium ID
Command ID	U2	0x23
Version	U4	1
Reserved2	U4	0
Payload Size	U4	12 + Size
Result Status	U4	0

Table 7-159: Write Section Response - Payload

Field Name	Type	Notes
Section ID	U4	
Offset [sectors]	U4	
Size [sectors]	U1	Number of sectors N; where N cannot exceed 8
× M	U1	M ≤ N × Sector size. The missing bytes are interpreted as zero.

Validate Section

The Validate Section command contains the content size, version, and CRC of the section. The following rules apply:

- It is applicable to all sections except the data section
- It is applicable only to invalid sections (cleared using the clear section command)
- If the verification of the CRC on the expected memory size within the section passes, then all the remaining bytes are set to 0. The version and size are stored in the table of the section and the section is declared valid.
- For code sections, the `Content Version` is the software version

C++ API:

```
void SnapSensor::SensorConnection::StorageModule::WriteSection (void
SnapSensor::SensorConnection::StorageModule::ValidateSection (SNAP_UINT32
sectionId, FrameworkModuleVersion contentVersion, void *data, SNAP_UINT32
dataLength, SNAP_UINT32 sectorLength)
```

Table 7-160: Validate Section Command - Header

Field Name	Type	Notes
Module ID	U1	0x40
Reserved1	U1	NV Storage medium ID
Command ID	U2	0x24
Version	U4	1
Reserved2	U4	0
Payload Size	U4	16

Table 7-161: Validate Section Command - Payload

Field Name	Type	Notes
Section ID	U4	Section ID to validate
Size [sectors]	U4	
Content Version	U4	
CRC	U4	

Table 7-162: Validate Section Response- Header

Field Name	Type	Notes
Module ID	U1	0x40
Reserved1	U1	NV storage medium ID
Command ID	U2	0x24
Version	U4	1
Reserved2	U4	0
Payload Size	U4	4
Result Status	U4	0

Table 7-163: Validate Section Response - Payload

Field Name	Type	Notes
Computed CRC	U4	

Get Log Data Summary

This command requests the summary of the data currently logged on the device whose sequence ID is greater or equal than the command parameter “start index”. The data returned are (the following data are computed by the device by taking into account only logged items with sequence ID greater than the command parameter start index):

- Start Index : the first valid logged item sequence ID greater than the command parameter start index
- Total Logged Items
- Space Left : space left in bytes (zero after the first data logging buffer rollover)
- Logged Types Count: count of how many types of logged items are currently logged.
- Count of logged items for each type.

Table 7-164: Get Log Data Summary Command - Header

Field Name	Type	Notes
Module ID	U1	0x40
Reserved1	U1	0
Command ID	U2	0x31
Version	U4	1
Reserved2	U4	0
Payload Size	U4	4
Result Status	U4	0

Table 7-165: Get Log Data Summary Command - Payload

Field Name	Type	Notes
Start Index	U4	

Table 7-166: Get Log Data Summary Response - Payload

Field Name	Type	Notes
Module ID	U1	0x40
Reserved1	U1	0
Command ID	U2	0x31
Version	U4	1
Reserved2	U4	0
Payload Size	U4	16 + N × 4
Result Status	U4	0

Table 7-167: Get Log Data Summary Response - Payload

Field Name		Type	Notes
Start Index		U4	
Total Logged Items		U4	
Space Left		U4	Expressed in bytes
Logged Type Count		U4	N
× N	Type ID	U1	
	Count	U3	Number of logged items of type TypeID

Get Log Data

The Get Log Data command requests the content of a certain number of items of the desired types starting at a specific index. The desired types are the ones included in the required types list. The device replies by trying to fit the content of the indicated number of items or as many as possible in the output buffer in the same order in which they were logged. The logged item content is always preceded by a header containing type, total size, sequence ID, offset and chunk size. Each new header is 32-bit aligned. The offset and chunk size are needed whenever the logged item cannot fit into the communication buffer. Otherwise, they are set respectively to 0 and total size.

Use the following procedure to download all the logged items of the desired types consists:

1. Send a first command with start index 0, offset 0, `Num Items` equal to a big number (for example 2^{31}). Flag the desired types in the Types Filter Mask.
2. If the response indicates a number or remaining items greater than zero, send a new command. But, set the `Start Index` and `Offset` to the values indicated in the response by next index and next offset.
3. Repeat until the number of remaining items indicated in the response `Remaining Items` field is zero.

   Whenever the response `Next Offset` field is not zero, the last logged item content could not fit inside the output buffer. For example, suppose that the log index is set to the type and size shown in the ***Example Device Log Buffer Content*** table and the buffer size is 50kb..

Table 7-168: Example Device Log Buffer Content

Seq ID	Type	Size
101	1	1kB
102	2	45kB
103	10	80kB
104	1	1kB

If the user requests to download all the logged data, the sequence of commands and responses is shown in the ***Example Command*** and ***Example Response*** tables. The size of the headers when computing the offsets is ignored in this example.

Table 7-169: Example Command

#	Start Index	Offset	Items Count
1	0	0	1e9
2	103	3kB	2
3	103	53kB	2

Table 7-170: Example Response

#	Start Index	Items Count	Rem. Items	Next Index	Next Offset	Notes
1	100	4	2	103	3kB	
2	103	1	2	103	53kB	*1
3	0	2	0	0	0	

*1 The response contains the second chunk of item 103. Also chunks are preceded by a header, in this case: type 10, total size 80kB, sequence ID 103, offset 3kB and chunk size 50 kB (for the sake of simplicity we ignore in this example the size of the logged items headers when computing the offsets).

Table 7-171: Get Log Data Command - Header

Field Name	Type	Notes
Module ID	U1	0x40
Reserved1	U1	0
Command ID	U2	0x32
Version	U4	1
Reserved2	U4	0
Payload Size	U4	13 + N *1
Reserved3	U4	

*1 If the field `Num of Types` is set to 255, then N is 0, no Type ID list shall be added.

Table 7-172: Get Log Data Command - Payload

Field Name	Type	Notes
Start Index	U4	
Offset	U4	Start offset into the content of the logged item at index `Start Index`
Num Items	U4	Maximum number of items to return

Table 7-172: Get Log Data Command - Payload (Continued)

Field Name		Type	Notes
Num of Types		U1	N. Required types list count. When set to 255 the following list shall not be added to the command.
N^2 ×	Type ID	U1	Required types list item [0..254]. Type ID 255 codes the invalid type.

Table 7-173: Get Log Data Response - Payload

Field Name	Type	Notes
Module ID	U1	0x40
Reserved1	U1	0
Command ID	U2	0x32
Version	U4	1
Reserved2	U4	0
Payload Size	U4	where L_i is the chuck size
Result Status	U4	0

Table 7-174: Get Log Data Response - Payload

Field Name		Type	Notes
Start Index		U4	
Items Count		U4	N
Remaining Items		U4	[0..10]*1
Next Index		U4	
Next Offset		U4	
× N	Type ID	U1	
	Total Size	U3	
	Sequence ID	U4	
	Offset	U4	
	Chunk Size	U4	Let L_i be the chuck size $20 + 16N + \sum_{1}^{N-1} 4ceil(L_i/4) + L_N$

Table 7-174: Get Log Data Response - Payload (Continued)

Field Name		Type	Notes
	Logged Item Data	U1 × M	Padding for 32-bit alignment is required for all the items except the last one. $M_i = ceil(L_i/4) \times 4$

*1 This field does not return the exact number of remaining items because this operation is generally time consuming. If the number of remaining items is more than 10, it returns 10.

Clear Log Data

The Clear Log Data command clears the log event buffer.

Table 7-175: Clear Log Data Header - Command

Field Name	Type	Notes
Module ID	U1	0x40
Reserved1	U1	0
Command ID	U2	0x33
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0
Reserved3	U4	

Table 7-176: Clear Log Data Header - Response

Field Name	Type	Notes
Module ID	U1	0x40
Reserved1	U1	0
Command ID	U2	0x33
Version	U4	1
Reserved2	U4	0
Payload Size	U4	0
Result Status	U4	0 – Success

8 Appendix A

The following tables describe the device, manufacturer, product and software IDs used in the ADIS1700x communication protocol.

Table 8-1: Device Identification

Device ID	Device Name	Notes
0	BlipMini	
1	Blip	
2	ADIS1700x	
3	Boost	
4	EagleEye	
5	LGA_CAV	S2 Opcom Tester
6:127	Reserved	

Table 8-2: Software Identification

Software Type ID	Software Type Name	Notes
0	Bootloader	
1	Camera	Camera Application
2	Occupancy HR	
3	Parking Occupancy	
4	S2Tester	
5:127	Reserved	

Table 8-3: Manufacturer Identification

Manufacturer Type ID	Manufacturer Name	Notes
0	Analog Devices	
1	SDataWay	

Product Identification

The product ID and name are a combination of the software type ID and device ID .

PRODUCT_ID = (SOFTWARE_TYPE_ID + (DEVICE_ID << 8))

PRODUCT_NAME = DEVICE_NAME + ”_” + SOFTWARE_TYPE_NAME

For example, for the ADIS1700x Camera application where:

- Software Type ID = 1
- Device ID = 2
- Device Name = ADIS1700x
- Software Type Name = Camera

The PRODUCT_ID = 512 and the PRODUCT_NAME = "ADIS1700x-Camera"

Error Messages

The messages shown in the ***Error Messages*** table are reported in the `Last Error Message` field of the Get Device Status response.

Table 8-4: Error Messages

Error String	Description	Applicable Module
FW not valid, max launches exc.	Bootloader has expired the maximum number of attempts to launch (boot) an unvalidated application. The application cannot launch. The bootloader tries to recover a restore point, if any are available. If a restore point is not available, the bootloader waits for the user to update a new firmware.	Bootloader
No valid firmware	No valid firmware is present in the “Current Application Firmware” section	Bootloader
WD RST on Current App. FW	A watchdog reset caused the last system reset. It is likely that the watchdog was caused by the firmware loaded in the Current Application Firmware” section	Bootloader
Camera Failure. Resetting camera	Camera service failed to initialize or to start/stop the camera or to set/get registers or LUT.	All except bootloader