

## **CC26xx, CC13xx Sensor Controller Studio Version 2.6**

This document provides an overview of the CC26xx and CC13xx Sensor Controller and the Sensor Controller Studio tool.

### **Contents**

1	Introduction .....	2
2	Overview .....	2
3	Prerequisites.....	9
4	Installation .....	10
5	Sensor Controller Studio Tutorials .....	11
6	Sensor Controller Studio Walkthrough .....	13
7	References .....	28

### **List of Figures**

1	SimpleLink™ Academy Labs .....	11
2	TI Resource Explorer .....	12
3	Start Page .....	13
4	Help Viewer Example Page.....	14
5	Example Configuration: ADC Window Monitor .....	15
6	Project Settings Panel .....	16
7	Task Panel .....	17
8	Constants and Data Structures Panel .....	18
9	Task Code Editor Panel .....	19
10	I/O Mapping Panel .....	20
11	Code Generator Panel.....	21
12	Task Testing, Setup Tab .....	23
13	Task Testing, Graph Tab.....	24
14	Task Debugging Panel .....	25
15	Run-Time Logging Panel.....	26
16	Run-Time Logging Graph Tab.....	27

### **Trademarks**

SimpleLink, Code Composer Studio, LaunchPad are trademarks of Texas Instruments.  
 Arm, Cortex, Keil are registered trademarks of Arm Limited (or its subsidiaries).  
 IAR Embedded Workbench is a registered trademark of IAR Systems AB.  
 Windows is a registered trademark of Microsoft Corporation.  
 All other trademarks are the property of their respective owners.

## 1 Introduction

Sensor Controller Studio is used to write, test and debug code for the CC26xx and CC13xx Sensor Controller. The tool generates a set of C source files, which contains the Sensor Controller firmware image and allows the System CPU application to control and exchange data with the Sensor Controller.

This document provides an overview of the CC26xx and CC13xx Sensor Controller and the Sensor Controller Studio tool.

Prerequisites for the generated driver source code and the supplied examples are stated in [Section 3](#).

[Section 5](#) provides information about how to access tutorials through the TI Resource Explorer.

The tutorial in [Section 6](#) provides a complete walkthrough of the Sensor Controller Studio tool.

## 2 Overview

Sensor Controller Studio is used to write, test and debug code for the CC26xx and CC13xx Sensor Controller. The tool generates a Sensor Controller Interface driver, which is a set of C source files to be compiled into the System CPU (Arm® Cortex®-M3 and Arm Cortex-M4 processors) application. These source files contain the Sensor Controller firmware image, associated definitions, and generic functions that allow the System CPU application to control the Sensor Controller and exchange data.

The Sensor Controller is a small CPU core that is highly optimized for low power consumption and efficient peripheral operation. The Sensor Controller is located in the CC26xx and CC13xx auxiliary (AUX) power/clock domain and can perform simple background tasks autonomously and independently of the System CPU and the MCU domain power state. These **tasks** include the following:

- Analog sensor polling, using ADC or comparator
- Digital sensor polling, using SPI, I2C, or other protocols
- Capacitive sensing using current source, comparator, and time-to-digital converter (TDC)
- Waveform generation, for example for LCDs

The Sensor Controller is user programmable, using a simple programming language with syntax similar to C. This programmability allows for sensor polling and other tasks to be specified as sequential algorithms rather than static configuration of complex peripheral modules, timers, DMA, register-programmable state machines, event routing, and so on. The main advantages are:

- Flexibility
- Dynamic reuse of hardware resources
- Ability to perform simple data processing without the need for dedicated hardware
- Observability and debugging options

For example, the following code samples an analog sensor and notifies the System CPU application if needed:

```
// Select ADC input
adcSelectGpioInput(AUXIO_A_SENSOR_OUTPUT);

// Enable the ADC (fixed reference, 2.7 us sample time, manual trigger)
adcEnableSync(ADC_REF_FIXED, ADC_SAMPLE_TIME_2P7_US, ADC_TRIGGER_MANUAL);

// Sample the sensor and store the ADC value
adcGenManualTrigger();
adcReadFifo(output.adcValue);

// Disable the ADC
adcDisable();

// Notify the application if above the threshold
if (output.adcValue > SENSOR_THRESHOLD) {
    fwGenAlertInterrupt();
}
```

The Sensor Controller task algorithms can efficiently be evaluated, debugged, and verified using the Sensor Controller Studio. The resulting Sensor Controller Interface driver can be treated as a black box, and can be integrated painlessly into the System CPU application.

There is a command line interface (CLI) that can be used for build automation.

## 2.1 Sensor Controller and AUX Domain Hardware Overview

Sensor Controller Studio supports the following device families:

- CC13x0 and CC26x0
- CC13x2 and CC26x2

### 2.1.1 Hardware Functionality

The Sensor Controller is located in the AUX domain of the CC26xx and CC13xx devices, and has access to hardware functionality located in the AUX domain.

The Sensor Controller has, with few exceptions, no access to peripherals, RAM, flash or registers in MCU and AON domains. This separation allows the MCU domain to enter and exit standby mode independently of the Sensor Controller. The System CPU has access to all AUX domain peripherals.

[Table 1](#) lists the functionality of the hardware in the AUX domain.

**Table 1. Hardware Functionality**

AUX Domain and Sensor Controller	CC13x0, CC26x0	CC13x2, CC26x2
Sensor Controller Engine (power-optimized 16-bit CPU core)	Yes	Yes
AUX RAM (Sensor Controller instruction and data memory)	2 KB = 1024 × 16-bit words	4 KB = 2048 × 16-bit words
Programmable wake-up sources	1	3
Wake up from AON_RTC channel 2	Yes	Yes
<b>Analog Peripherals</b>		
ADC (12-bit, 200-kHz analog-to-digital converter)	Yes	Yes
COMP_A (continuous time comparator)	Yes	Yes, improved
COMP_B (low-power clocked comparator)	Yes	Yes, improved
ISRC (0 to 20-μA current source)	Yes	Yes
Reference DAC (8-bit DAC for COMP_A/COMP_B reference)	—	Yes
<b>Digital Peripherals</b>		
TDC (96-MHz time-to-digital converter)	Yes	Yes
Pulse counter (16-bit asynchronous)	Yes	Yes, improved
Timer0 (synchronous timer)	Yes, 16-bit	Yes, 16-bit, improved
Timer1 (synchronous timer)	Yes, 8-bit	Yes, 16-bit, improved
Timer2 (16-bit asynchronous PWM/sequencing timer)	—	Yes
Microsecond delay timer	—	Yes
AUX I/O controllers (8 pins each)	2	4
Hardware semaphores (for peripheral sharing with System CPU)	8	8
Multiply and accumulate accelerator (40-bit accumulator)	—	Yes
<b>Serial Interfaces</b>		
Serial peripheral interface (SPI)	Bit-banged	Yes, hardware peripheral
I2C master (400 kHz or 100 kHz)	Bit-banged	Bit-banged
<b>I/O Pins</b>		
Analog- and digital-capable general-purpose I/O pins	Up to 8	Up to 8
Digital-only general-purpose I/O pins	Up to 8	Up to 23

**Table 1. Hardware Functionality (continued)**

AUX Domain and Sensor Controller	CC13x0, CC26x0	CC13x2, CC26x2
<b>System</b>		
Observation signal output (AUX event bus signal to GPIO)	—	1
AON_BATMON read access (temperature and voltage)	—	Yes
AON_RTC read access (16-bit second, 16-bit subsecond)	—	Yes
MCU domain and power supply system status	—	Yes
VDDR recharge status and interaction	—	Yes

For an overview of the differences between the CC13x0, CC26x0 and the CC13x2, CC26x2 device families, see the *Chip Family Migration Guide* in the Sensor Controller Studio help viewer.

For detailed technical information about the Sensor Controller Engine CPU core, see the *Assembly Language Reference* in the Sensor Controller Studio help viewer.

### 2.1.2 Power and Clock Management

The sections that follow describe for each device family the available power modes and the clock frequencies in each power mode, depending on the state of the System CPU.

The main improvements in the CC13x2 and CC26x2 device family are:

- 2-MHz clock generated by RCOSC MF
- Low-power mode (power supply system is off, some analog and digital peripherals are unavailable)
- Faster wake-up times

#### 2.1.2.1 CC13x0 and CC26x0 Operation Modes

The Sensor Controller is in active mode when running task code and in standby mode otherwise. The Sensor Controller firmware framework, which is integrated in the generated driver, autonomously handles the transitions into and out of standby mode.

All peripheral modules are available in active mode, unless they are used by the System CPU. A small number of peripherals can be used in standby mode to trigger wakeup of the Sensor Controller.

[Table 2](#) lists the Sensor Controller/AUX domain clock source and frequency for the different modes of operation.

**Table 2. Clock Source and Frequency in the Sensor Controller and AUX Domain**

Operation State	Frequency and Source	System CPU and MCU Domain Active	System CPU and MCU Domain Standby
Active	Sensor Controller clock	24 MHz (SCLK_HF / 2)	24 MHz (SCLK_HF / 2)
	AUX bus clock	24 MHz (SCLK_HF / 2)	24 MHz (SCLK_HF / 2)
Standby with event trigger	Sensor Controller clock	24 MHz (SCLK_HF / 2)	32 kHz (SCLK_LF)
	AUX bus clock	24 MHz (SCLK_HF / 2)	32 kHz (SCLK_LF)
Standby without event trigger	Sensor Controller clock	24 MHz (SCLK_HF / 2)	No clock
	AUX bus clock	24 MHz (SCLK_HF / 2)	No clock

#### 2.1.2.2 CC13x2 and CC26x2 Operation Modes

The Sensor Controller is in active mode or low-power mode when running task code, and in standby mode otherwise.

The Sensor Controller firmware framework, which is integrated in the generated driver, handles the transitions into and out of standby mode autonomously.

The Sensor Controller configuration and task code can trigger transitions between the active and low-power modes. The Sensor Controller wakes up to the power mode that was used when it entered standby.

All peripheral modules are available in active mode, unless they are used by the System CPU. Peripheral modules that require active mode (SCLK\_HF and/or other system functionality) are not available in low-power mode. A small number of peripherals can be used in standby mode to trigger Sensor Controller wake-up.

[Table 3](#) shows the Sensor Controller and AUX domain clock source and frequency for the different modes of operation.

**Table 3. Clock Source and Frequency in the Sensor Controller and AUX Domain**

Operation State	Frequency and Source	System CPU and MCU Domain Active	System CPU and MCU Domain Standby
Active with Sensor Controller at 24 MHz	Sensor Controller clock	24 MHz (SCLK_HF / 2)	24 MHz (SCLK_HF / 2)
	AUX bus clock	24 MHz (SCLK_HF / 2)	24 MHz (SCLK_HF / 2)
Active with Sensor Controller at 2 MHz	Sensor Controller clock	2 MHz (derived from SCLK_HF / 2)	2 MHz (derived from SCLK_HF / 2)
	AUX bus clock	24 MHz (SCLK_HF / 2)	24 MHz (SCLK_HF / 2)
Low-power	Sensor Controller clock	2 MHz (derived from SCLK_HF / 2)	2 MHz (SCLK_MF)
	AUX bus clock	24 MHz (SCLK_HF / 2)	2 MHz (SCLK_MF)
Standby with Timer0/1 event trigger	Sensor Controller clock	32 kHz (derived from SCLK_HF / 2)	32 kHz (SCLK_LF)
	AUX bus clock	24 MHz (SCLK_HF / 2)	32 kHz (SCLK_LF)
Standby without Timer0/1 event trigger	Sensor Controller clock	No clock	No clock
	AUX bus clock	24 MHz (SCLK_HF / 2)	No clock

### 2.1.2.3 Communication With the System CPU Application

The Sensor Controller tasks and System CPU application exchange data through shared memory in the AUX RAM. Sensor Controller Studio generates type definitions and other needed definitions for easy access to these variables.

A pair of event signals is used by the Sensor Controller Interface driver to control the Sensor Controller tasks:

- To Sensor Controller: **CTRL** initiates a control operation that starts or stops Sensor Controller tasks.
- From Sensor Controller: **READY** indicates that the control interface is idle and ready for another control operation.

Another pair of event signals is used by the Sensor Controller to notify the System CPU and exchange data:

- From Sensor Controller: **ALERT** wakes up the System CPU if currently in standby mode, and generates a callback (interrupt).
- To Sensor Controller: **ACK** indicates that the System CPU has handled the last alert and is ready for another.

The event signaling is handled internally by the Sensor Controller Interface driver and the Sensor Controller firmware framework, but it is exposed to the application through function calls and callbacks.

## 2.2 Sensor Controller Interface Driver

The generated Sensor Controller Interface driver (SCIF) consists of three pairs of C header and source files that are used in the application running on the System CPU. The driver includes:

- The **driver setup** contains:
  - The AUX RAM image (containing machine code and data for the Sensor Controller)
  - Data structure definitions (for easy access to Sensor Controller configuration, state and input/output data)
  - Constant definitions (matching relevant constants used in the Sensor Controller code)

- I/O mapping, task data structure information, and other parameters for driver internal use
- A **generic application programming interface (API)** for:
  - Initializing the driver (including I/O pin configuration and event routing)
  - Task control (for starting, stopping and manually executing Sensor Controller tasks)
  - Task data exchange (for producing input data to and consume output data from Sensor Controller tasks)
  - Uninitializing the driver (allowing the application to switch SCIF driver setup)
- An **operating system abstraction layer (OSAL)** that:
  - Ensures seamless integration with the selected operating system, for example TI-RTOS
  - Ensures seamless integration with the power and clock management framework running on the System CPU

### 2.2.1 Tailored How-To-Use Guide

Sensor Controller Studio also outputs a tailored how-to-use guide (HTML document) together with the generated SCIF driver.

The guide is divided into two main sections:

- **Getting Started:** Basic steps to add the SCIF driver to the System CPU application.
- **Further Integration:** Detailed task control, other SCIF driver features, and helpful information.

The guide contains only relevant information for each generated SCIF driver.

The guide contains code snippets that can be copied and pasted directly into the application source code.

### 2.2.2 Doxygen Documentation

The SCIF driver header and source files are documented using Doxygen syntax. This documentation includes project-specific information, such as names, descriptions, I/O mapping overview, constants, task data structures, and resource specific APIs (for example for the "UART Emulator" resource).

For the generated SCIF driver, Sensor Controller Studio outputs a doxyfile (that is, a project file for Doxygen), and a Windows batch file for running Doxygen on this doxyfile.

## 2.3 Sensor Controller Programming Model

Sensor Controller Studio is project-based. All data associated with a Sensor Controller project is stored in one file (\*.scp). Each project may contain one or more Sensor Controller tasks (for example, capacitive sensing and ADC measurement). The output of a project is one SCIF driver.

The task code programming language uses a syntax that is similar to C, but has limited features compared to C because it specifically targets the instruction set and architecture of the Sensor Controller Engine. For example, there is only support for 16-bit variables.

Hardware peripherals and software algorithms are available as resources. For each task, a set of resources must be selected and configured to enable different types of functionality, such as:

- Analog and digital peripheral modules (ADC)
- Analog and digital general-purpose I/O (GPIO) pins
- Simple and complex software algorithms
- Bit-banged serial interfaces
- Task scheduling and event handling,
- Communication with the System CPU

Each task resource enable a set of procedures (equivalent to functions in C), with associated constants and variables. The task code can call these procedures to access hardware modules, firmware framework features, and optimized software algorithms.

It is possible to add user-specified constants and variables that can be linked to resource configuration values (for example, the number of ADC input pins).

The programming model and firmware framework are optimized for low overhead when communicating with the System CPU application, low AUX RAM memory footprint, and efficient use of the Sensor Controller's instruction set. To minimize power consumption, the Sensor Controller enters standby mode when it is idle.

## 2.4 Sensor Controller Tasks

A Sensor Controller task consists of the following components:

- Resources, with associated configurations, procedures and I/O pins
- Constants
- Data structures
- Task code blocks

Sensor Controller tasks are fully independent and cannot transfer data or control each other.

### 2.4.1 Data Structures

Each Sensor Controller task stores its global variables in a standardized set of data structures. The data structures are located in the AUX RAM, which is mapped into the memory address space of the System CPU and the DMA. The data structures for a task are:

- **cfg**: Used to perform run-time configuration of the task before the task is started
- **input**: Used to pass input data to the task (for example, dynamic parameters for an external sensor)
- **output**: Used to pass output data to the System CPU application (for example, accelerometer data)
- **state**: Internal variables used to store the state of the task between iterations

The output data structure can be single- or multiple-buffered. The task can alert the System CPU application (with interrupt generation) to request data exchange or the application can simply poll and access the data structure variables (if the System CPU wakes up periodically).

### 2.4.2 Task Code Blocks

The task algorithm is divided into four types of code blocks:

- **Initialization Code**: Runs one time when the task is started through the task control interface
  - Performs one-time hardware initialization to reduce execution time
  - Sets up task data and hardware for special behavior during the first execution
  - Schedules the next iteration of the Execution Code and (or) sets up an *Event Handler Code* trigger
- To run iterations of the Sensor Controller task, one or both of the following code blocks can be implemented:
  - **Execution Code**: Runs each time the task is scheduled for execution (based on periodic ticks from the real-time counter (RTC))
    - Executes the task algorithm, for example sampling capacitive touch buttons, simple data filtering, processing and buffering
    - Alerts the System CPU application to perform data exchange or signalize events when needed (for example when an array of ADC samples is full)
    - Schedules the next iteration of the Execution Code and (or) sets up an *Event Handler Code* trigger
  - **Event Handler Code**: Runs one time when the trigger that was set up occurs, (for example, an edge or level on an AUX I/O pin, or after a variable delay):
    - The timer trigger is typically used to run Sensor Controller tasks at irregular intervals or to follow up on actions performed by the Execution Code
    - The GPIO trigger is typically used to respond to external interrupts
    - Other event triggers are available, depending on the device family
    - Alerts the System CPU application to perform data exchange or signalize events when needed (for example, when accelerometer data indicates movement)



- Sets up another *Event Handler Code* trigger, if needed
- **Termination Code:** Runs one time when the task is stopped through the task control interface
  - Shuts down hardware left active between task iterations
  - Performs final (partial) data exchange, if needed

The CC13x0 and CC26x0 device family supports one event trigger and one Event Handler Code block per project.

The CC13x2 and CC26x2 device family supports three independent event triggers and up to three Event Handler Code blocks per project.

### 2.4.3 High-Level Program Flow

The Sensor Controller Engine does not support preemption, which means that one task code block cannot interrupt another task code block. While this is a limitation, it prevents actions of one task (for example SPI communication) from interfering with the measurements of another task (for example, capacitive touch).

A Sensor Controller task is typically executed in short iterations. It is possible to implement tasks that run for longer periods of time, but this will prevent other tasks from running during that time.

## 2.5 Task Testing and Debugging

The Sensor Controller Studio provides a generic, easy-to-use environment for ad hoc and exhaustive testing, and for low-level debugging of tasks.

Task testing can be performed one task at a time, using an XDS100v3, XDS110 or XDS200 JTAG debug probe for interfacing with the CC13xx and CC26xx device. While testing, the Sensor Controller Studio acts as the System CPU application and is responsible for controlling the Sensor Controller task. Values of all data structure members (from cfg, input, output and state) are logged after each task iteration. These values can be displayed graphically in Sensor Controller Studio and can also be saved to file for external analysis.

Low-level task code debugging allows for single-stepping instructions or running, with breakpoints, the initialization, execution, event handler and termination code blocks. Debugging is performed on the generated assembly code.

Because the Sensor Controller tasks will execute asynchronously with the System CPU application, and mostly while the MCU power/clock domain is in standby, there is normally little to be gained from debugging the Sensor Controller code together with the System CPU application code. This option is therefore not supported.

## 2.6 Run-Time Logging

The Sensor Controller Studio provides a generic, easy-to-use environment for evaluating and optimizing performance of tasks while these run at full speed, as they would in the actual application.

Run-time logging can be performed using a generic System CPU application image programmed by Sensor Controller Studio (through an XDS100v3 or XDS110 JTAG debug probe), or it can be performed using a custom application image programmed manually into flash. Commands and data are then transferred over UART, using a protocol based on the network processor interface (NPI).

The task data structures can be selected individually as either logged or editable. Logged data structures can be displayed graphically in Sensor Controller Studio and can also be saved to file for external analysis. Editable data structures can be modified by Sensor Controller Studio while the tasks are running on the Sensor Controller.



## 3 Prerequisites

### 3.1 Driver

The generated Sensor Controller Interface driver depends on register definitions and basic operating system functionality. These dependencies are included in the following software development kits (SDKs):

- SimpleLink™ CC13x0 SDK
- SimpleLink CC2640R2 SDK
- SimpleLink CC13x2 CC26X2 SDK (silicon revision E)

The driver is also compatible with previous SDK generations:

- TI-RTOS for CC26xx and CC13xx
- TI-RTOS for SimpleLink Wireless MCUs

The *Revision History* lists all compatible SDK releases for your current Sensor Controller Studio version. The generated driver will normally also be compatible with future SDK releases because it depends only on basic SDK functionality.

The driver source files are compatible with the following IDEs and compilers:

- IAR Embedded Workbench® for Arm (EWARM)
- Code Composer Studio™ IDE (TI compiler)
- Keil® MDK-ARM
- Arm GCC

### 3.2 Examples

Examples with associated files are configured and patched when opened through the Start Page panel of the Sensor Controller Studio. This allows support for different target chips (when supported by the example hardware platform) and for different SDK releases. Example project files are provided for the IAR EWARM and TI CCS toolchains.

The *Revision History* lists all supported SDK releases for your current Sensor Controller Studio version.

The example application project files require one of the following:

- For CC13x0 and CC26x0 devices:
  - IAR EWARM 7.80.4 or later
  - TI CCS 7.1.0 or later
- For CC13x2 and CC26x2 devices:
  - IAR EWARM 8.20.1 or later
  - TI CCS 7.3.0 or later

## 4 Installation

### 4.1 Sensor Controller Studio for Windows® (7/8.1/10)

By default, the Sensor Controller Studio will install to <Program Files>\Texas Instruments\Sensor Controller Studio. The installer requires administrator privileges to allow for the JTAG debug probe (XDS100v3, XDS110 and XDS200) driver installation. The JTAG interface is used for testing and debugging of the Sensor Controller tasks.

Once started, Sensor Controller Studio creates folders for examples, user projects and user-defined content under <My Documents>\Texas Instruments\Sensor Controller Studio.

#### 4.1.1 Update Service

Sensor Controller Studio 1.3.0 and later can check for new releases and for patches to your current installation. Patches are made available between releases to fix bugs and add new features, such as:

- Support for new SDK releases in the example configuration
- Bug fixes for SCIF driver and task code procedures
- New examples, task resources and task code procedures

Patches can be added and removed independently without running an installer.

### 4.2 Sensor Controller Studio CLI for Linux (64-Bit)

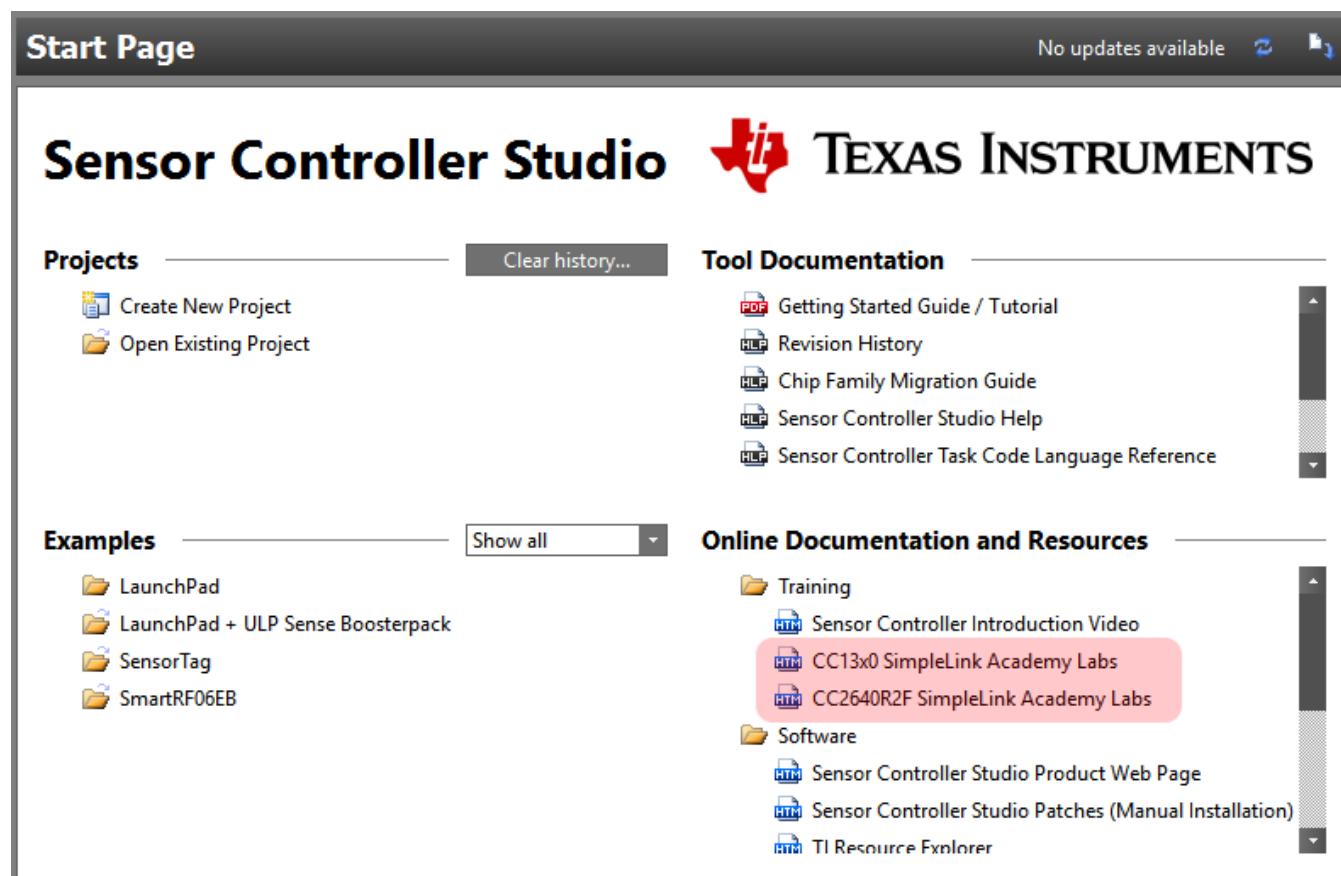
The command line interface (CLI) for Linux is provided as a minimal tarball. This CLI can be used to automate the Sensor Controller Studio code generation step.

See the included readme.md file for instructions and other relevant information.

## 5 Sensor Controller Studio Tutorials

Tutorials for Sensor Controller Studio are available in the TI Resource Explorer, under the SimpleLink Academy labs.

To access these tutorials, double-click the links for the SimpleLink Academy Labs in the Start Page panel of the Sensor Controller Studio (see [Figure 1](#)).



**Figure 1. SimpleLink™ Academy Labs**

Clicking these links opens the TI Resource Explorer in a Web browser (see [Figure 2](#)).

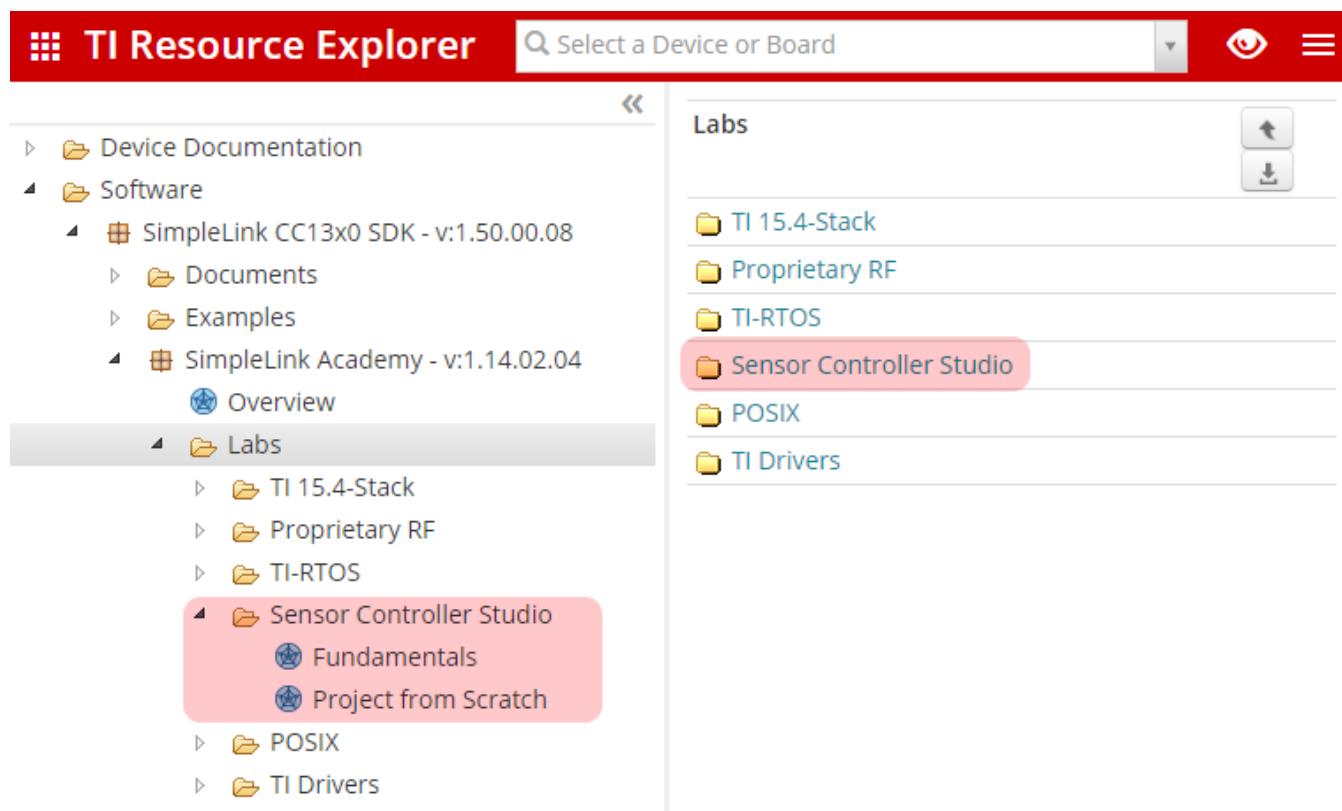


Figure 2. TI Resource Explorer

## 6 Sensor Controller Studio Walkthrough

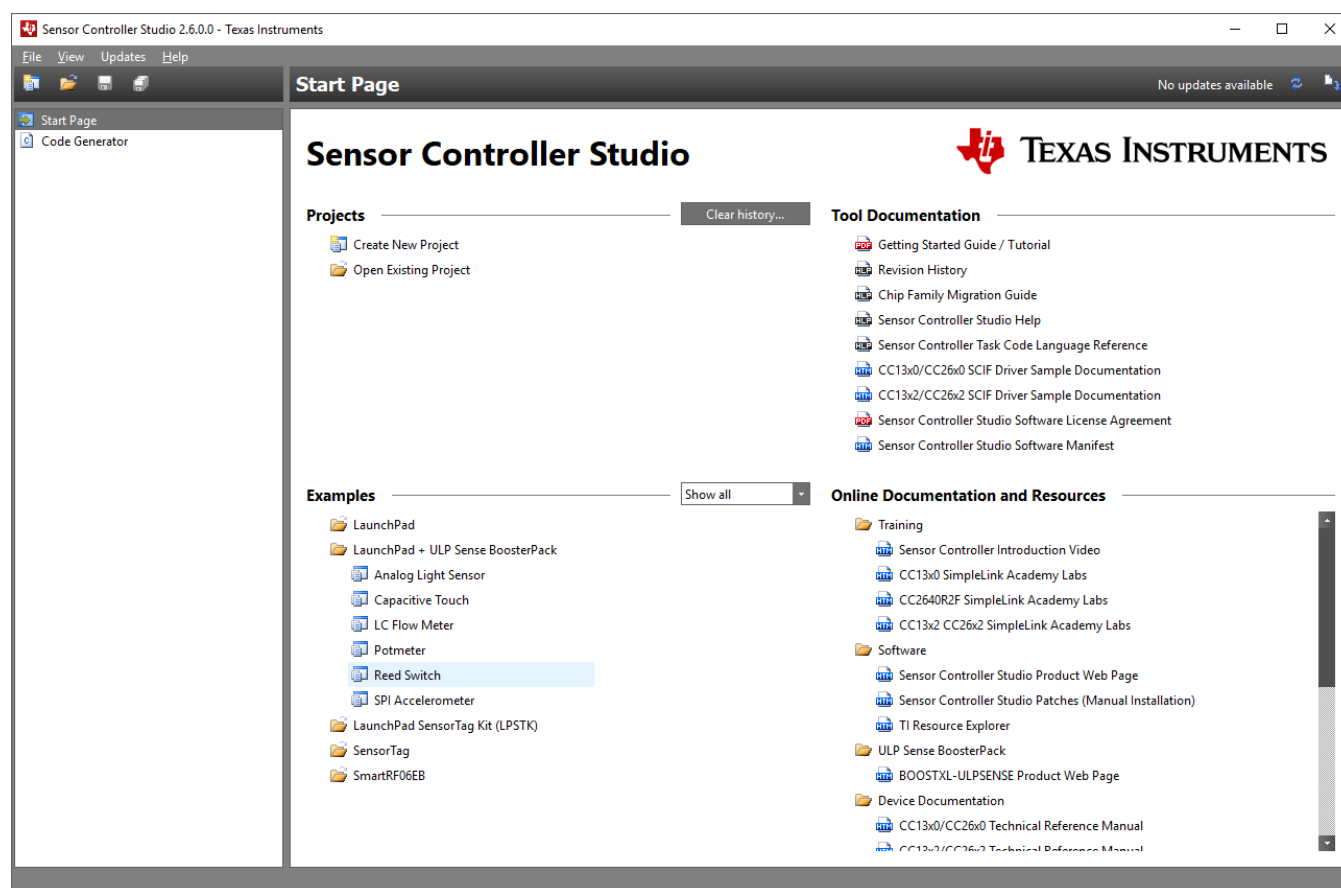
This section provides a detailed walkthrough of Sensor Controller Studio tool. The walkthrough opens the “ADC Window Monitor” example for the LaunchPad™ Development Kit, and then explains the different parts of the tool.

### 6.1 Start Page and Navigation

The Start Page provides easy access to projects, examples, documentation and online resources (see [Figure 3](#)).

The left part of the main window displays a project tree that is used for navigation. The right part of the main window displays the panel that is currently selected in the project tree (for example, the Start Page).

For efficient navigation, use the **Alt+Up/Down** keyboard shortcuts, the panel shortcuts found in the View menu, and the mouse back/forward buttons.



**Figure 3. Start Page**

Access preference settings for Sensor Controller Studio through the file menu or by pressing **Ctrl+P**. These control tool behaviors that are not related to specific Sensor Controller projects.

## 6.2 Documentation

Most of the Sensor Controller Studio documentation is provided in the Help Viewer. This documentation includes:

- Introduction and overview of the tool
- Panel documentation
- Resource and procedure documentation
- Sensor Controller task code programming language reference and other reference documents
- Revision history

Other documentation (for example, for the generated SCIF driver) can be accessed through the Start Page panel.

Press **F1** at any time to open the Help Viewer (see the example in [Figure 4](#)).

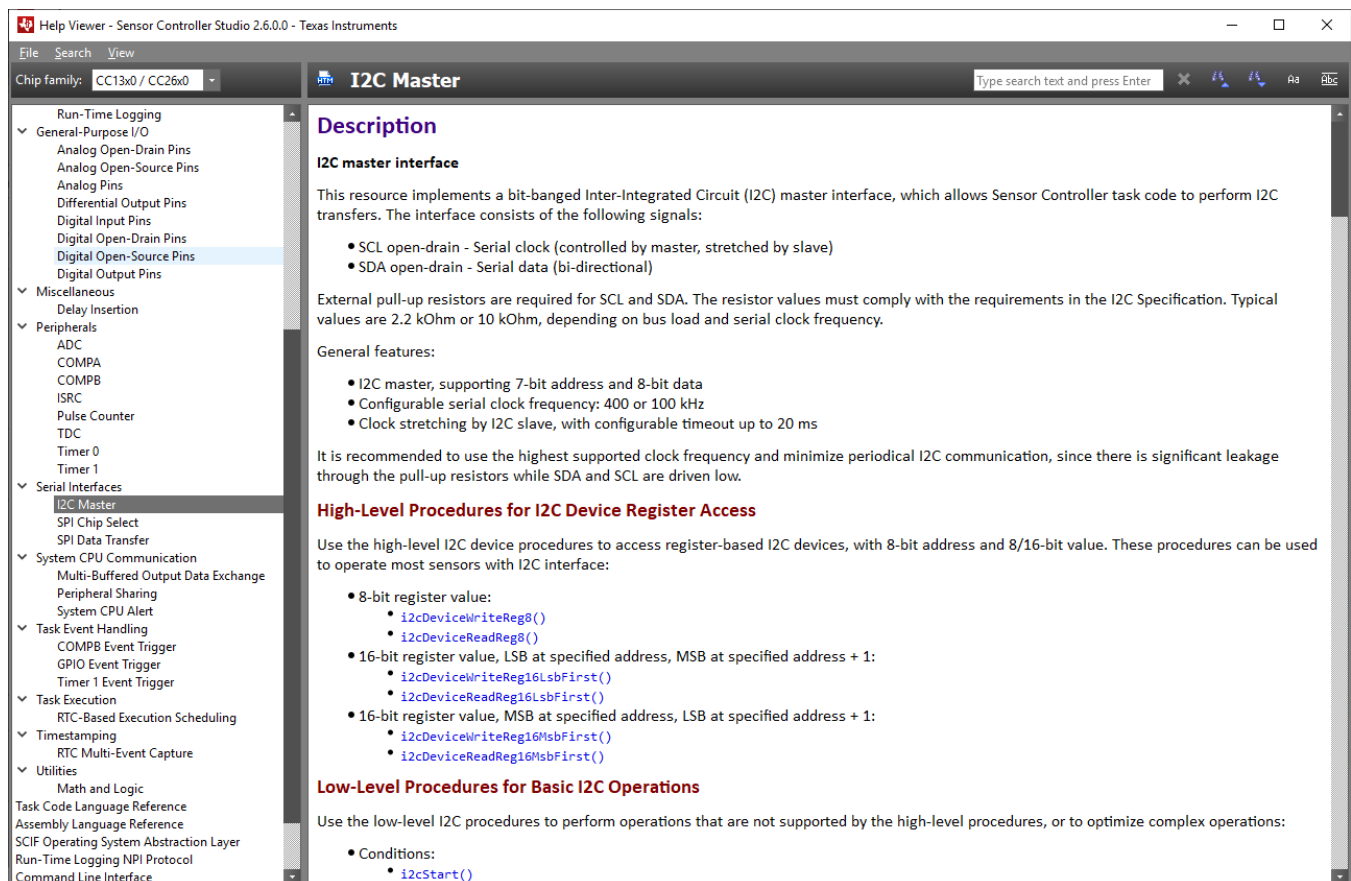


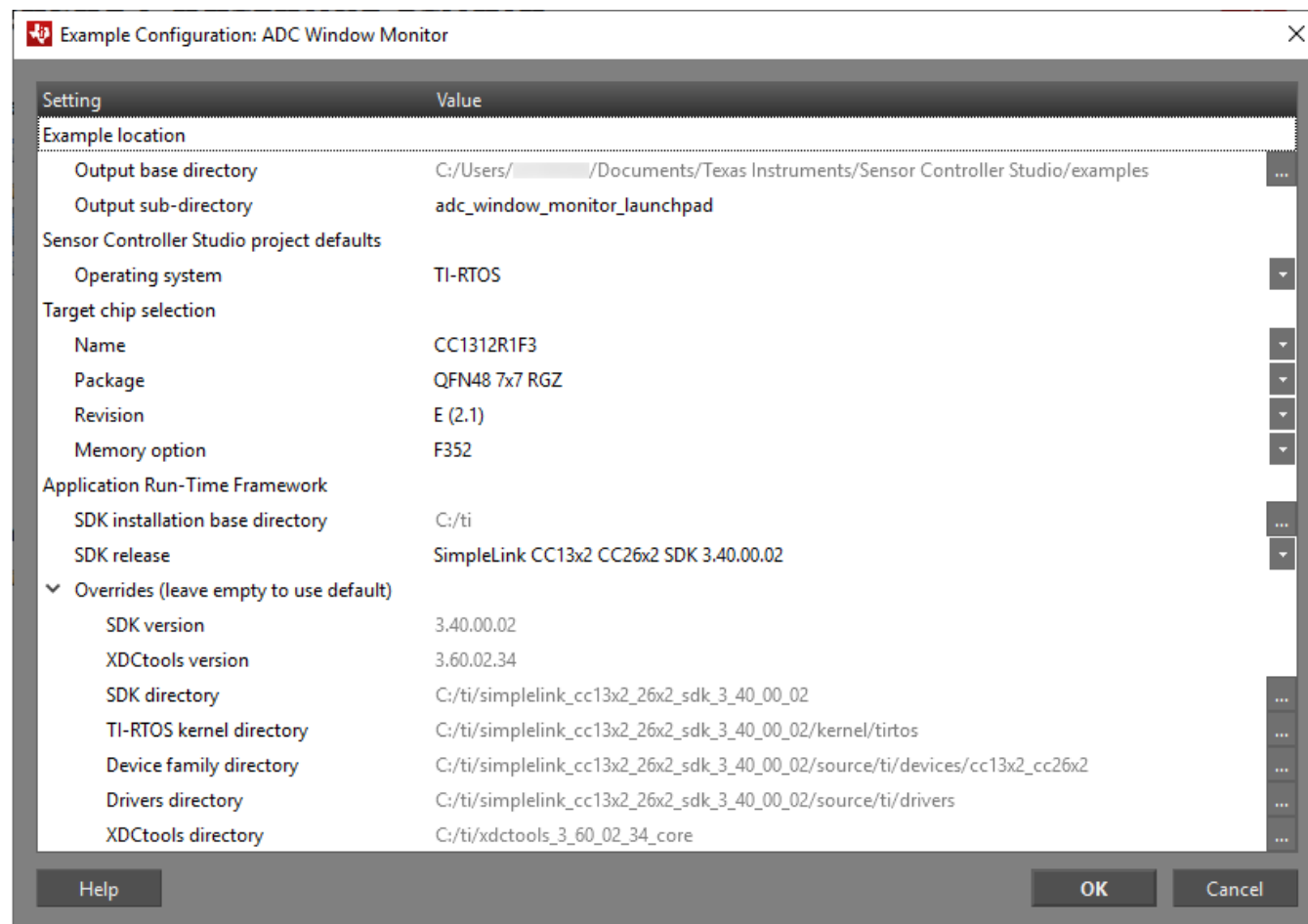
Figure 4. Help Viewer Example Page

### 6.3 Open the Example

In the "Examples" section of the Start Page, open the example configuration window by clicking on the "LaunchPad" folder and then double-clicking on "ADC Window Monitor."

This action will display the example configuration window (see [Figure 5](#)), which allows the examples to work seamlessly with the following:

- Different target chips
- Different SDK releases



**Figure 5. Example Configuration: ADC Window Monitor**

- Check that the settings for the target chip name and SDK release are correct, and update if necessary. You can use the base directory and override settings if the desired SDK is not yet supported, or if it is installed in a nondefault location.
- Press **OK**.

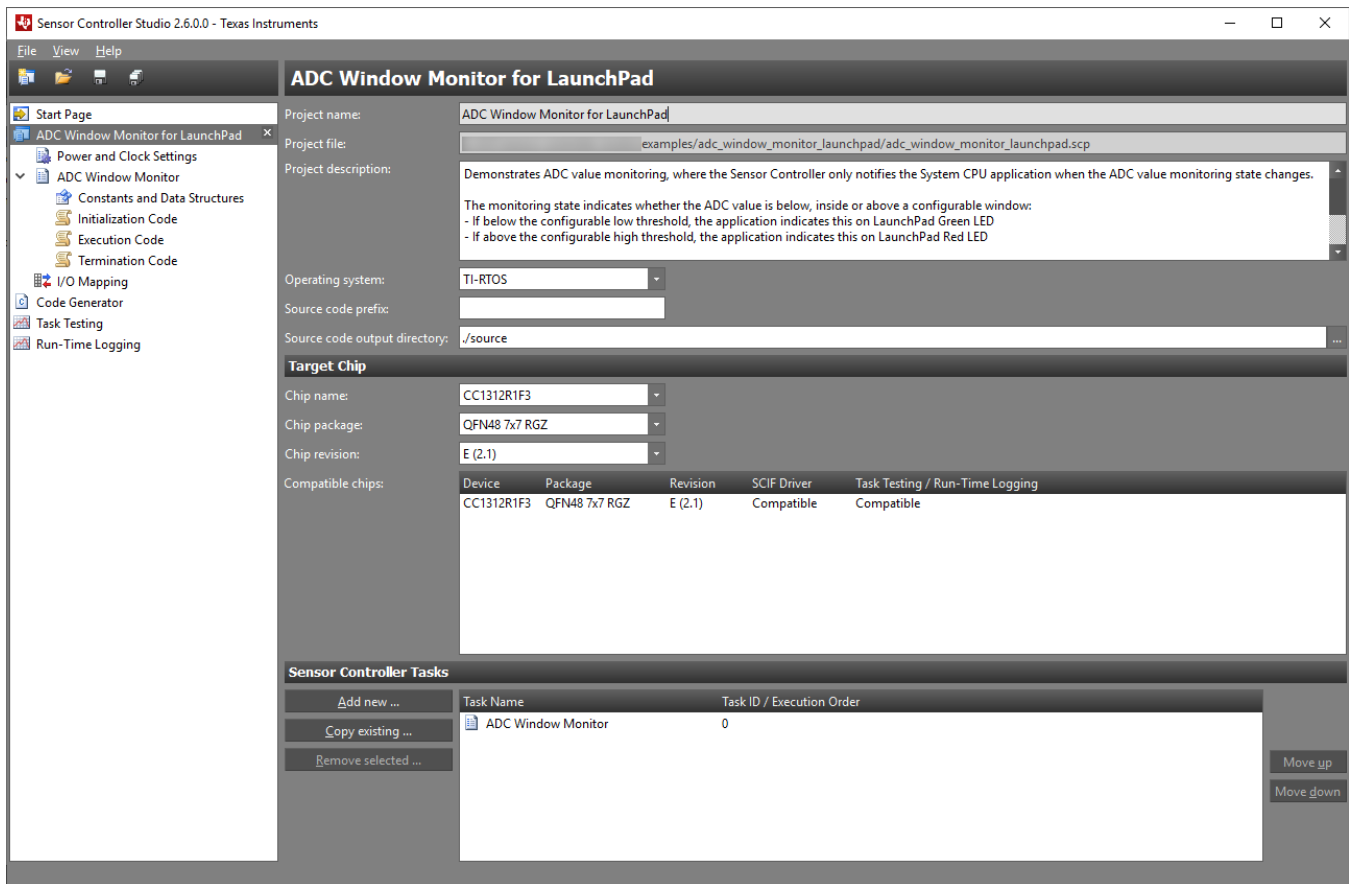
This action will patch and output the Sensor Controller project, application source code, and project files for IAR and CCS.

**NOTE:** While chip name, revision and package can be modified later in the Project panel, these settings will then be incompatible with the application project files. Instead, rerun the example configuration to create new application project files.



## 6.4 Project Panel

The project panel contains settings that apply to the entire project, and allows for tasks to be added and removed (see [Figure 6](#)).



**Figure 6. Project Settings Panel**

The project settings include:

- Project name and description: These are included in the generated SCIF driver setup header file.
- Operating system: Select the operating system that affects which SCIF OSAL files are output.
- Source code prefix (optional): If needed, specify a prefix that will be added to relevant parts of the driver setup (including file names). This allows for multiple SCIF driver setups in one application.
- Source code output directory: Specify the path of the generated output by the absolute path or path relative to the Sensor Controller project file (\*.scp).
- Target chip: Select the target chip by specifying name, revision, and package. These settings affect which task resources are available, I/O mapping, task testing, and runtime logging.

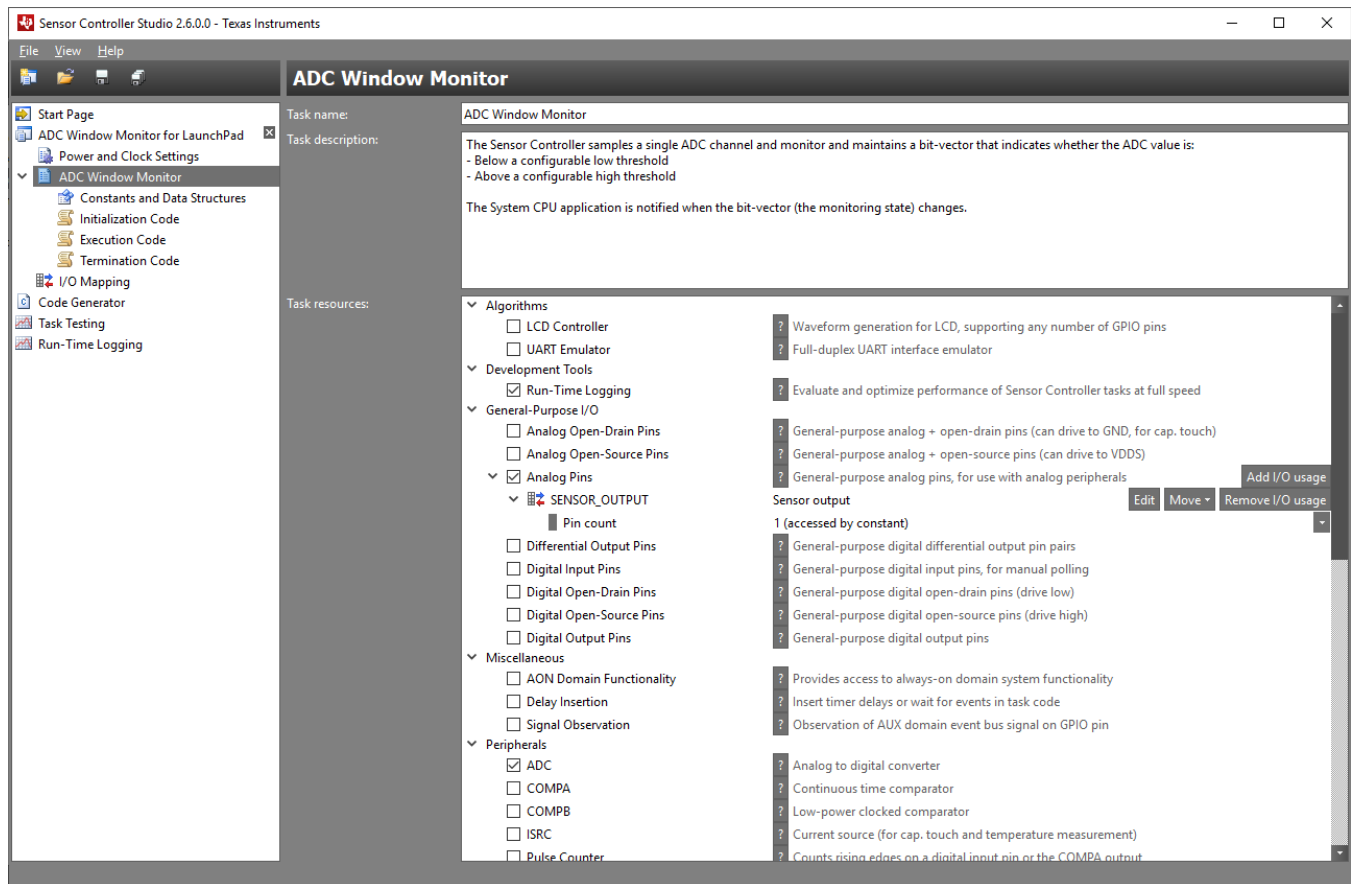
Sensor Controller tasks can be added to and removed from the project in the bottom section.

It is possible to have multiple projects open at the same time.

**NOTE:** The *Save project* and *Close project* commands apply only to the currently selected project in the project tree.

## 6.5 Task Panel Settings

The task panel contains settings that apply to one task, including resource selection and configuration (see the example in [Figure 7](#)).



**Figure 7. Task Panel**

The task name is used in the generated SCIF driver code to identify the task and must be unique per project.

The task description (also included in the generated driver) can be used to explain the task purpose, usage, electrical connections, and so forth.

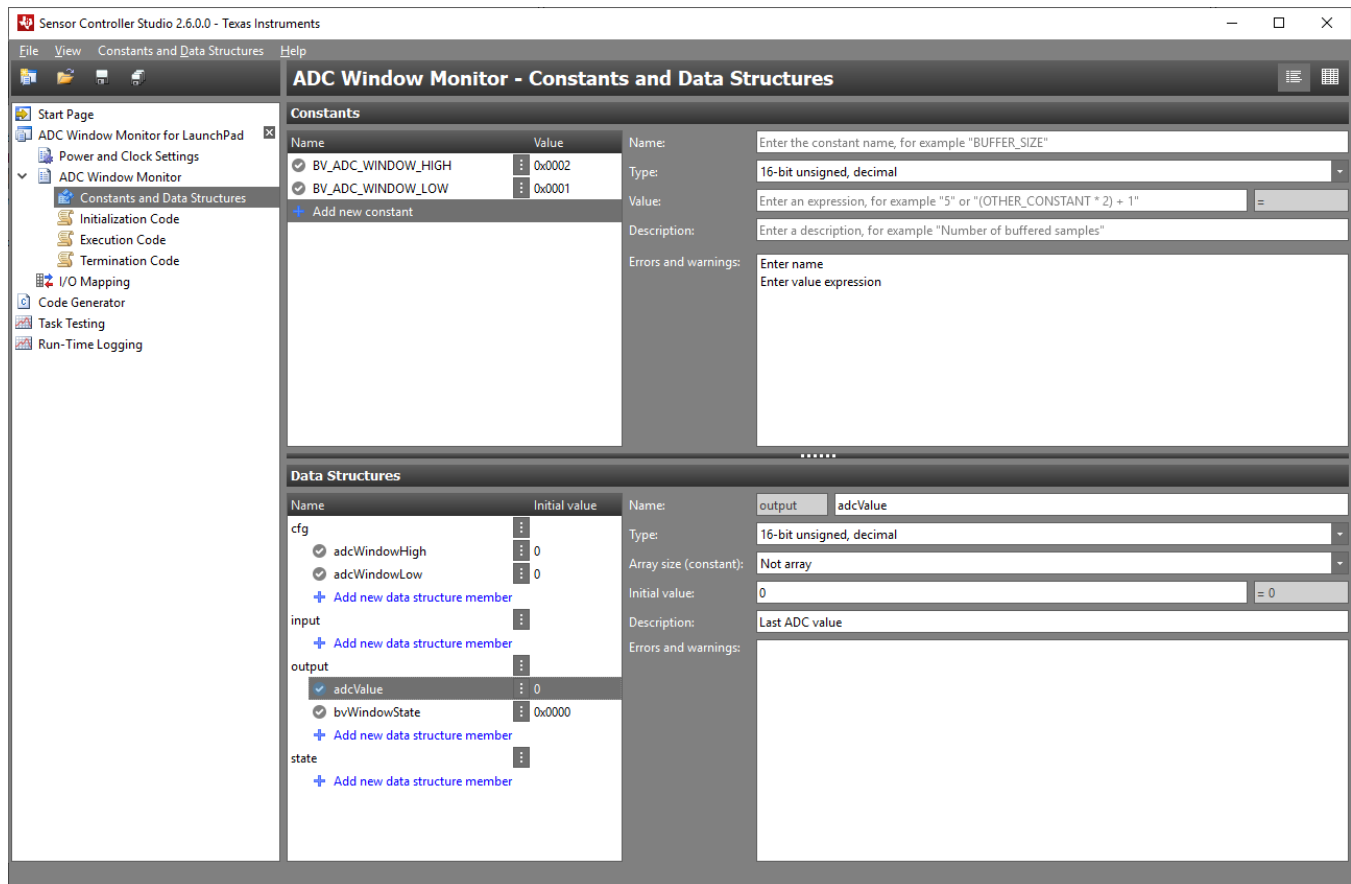
Task resources are selected to enable access to hardware modules, firmware framework functionality and various software algorithms. Some resources have configuration settings. For the "General Purpose I/O" resources, multiple usages with different name and configuration can be specified. Press the help buttons [?] to view resource documentation, including associated procedures, constants and variables.

For the ADC Window Monitor task, the following resources are selected:

- **ADC:** to measure the output of the light sensor
- **Analog Pins** (one instance): to connect the analog sensor output to the ADC
- **System CPU Alert:** to wake up and generate interrupts in the System CPU application
- **RTC-Based Execution Scheduling:** to trigger periodical execution of the task

## 6.6 Constants and Data Structures Panel

The constants and data structures panel is used to add, edit and remove user-defined constants and data structure variables (see [Figure 8](#)).



**Figure 8. Constants and Data Structures Panel**

Constants are available both to Sensor Controller task code, and to the System CPU application through C defines (with prefix to prevent name conflict).

Data structure variables are used to store Sensor Controller data between task iterations, and to exchange data with the System CPU application (through C structs).

There are two editor views/modes:

- Form view (default): Edit attributes for one constant and one data structure at a time
- Table view: Shows all attributes for all constants and data structures to allow more efficient editing

## 6.7 Task Code Editor Panels

There is one task code editor panel for each task code block. The Event Handler Code panels are only displayed when event trigger resources are enabled in the task panel.

To the left is the task code editor. Press **F2** to open the Task Code Language Reference document in the Help Viewer (see [Figure 9](#)).

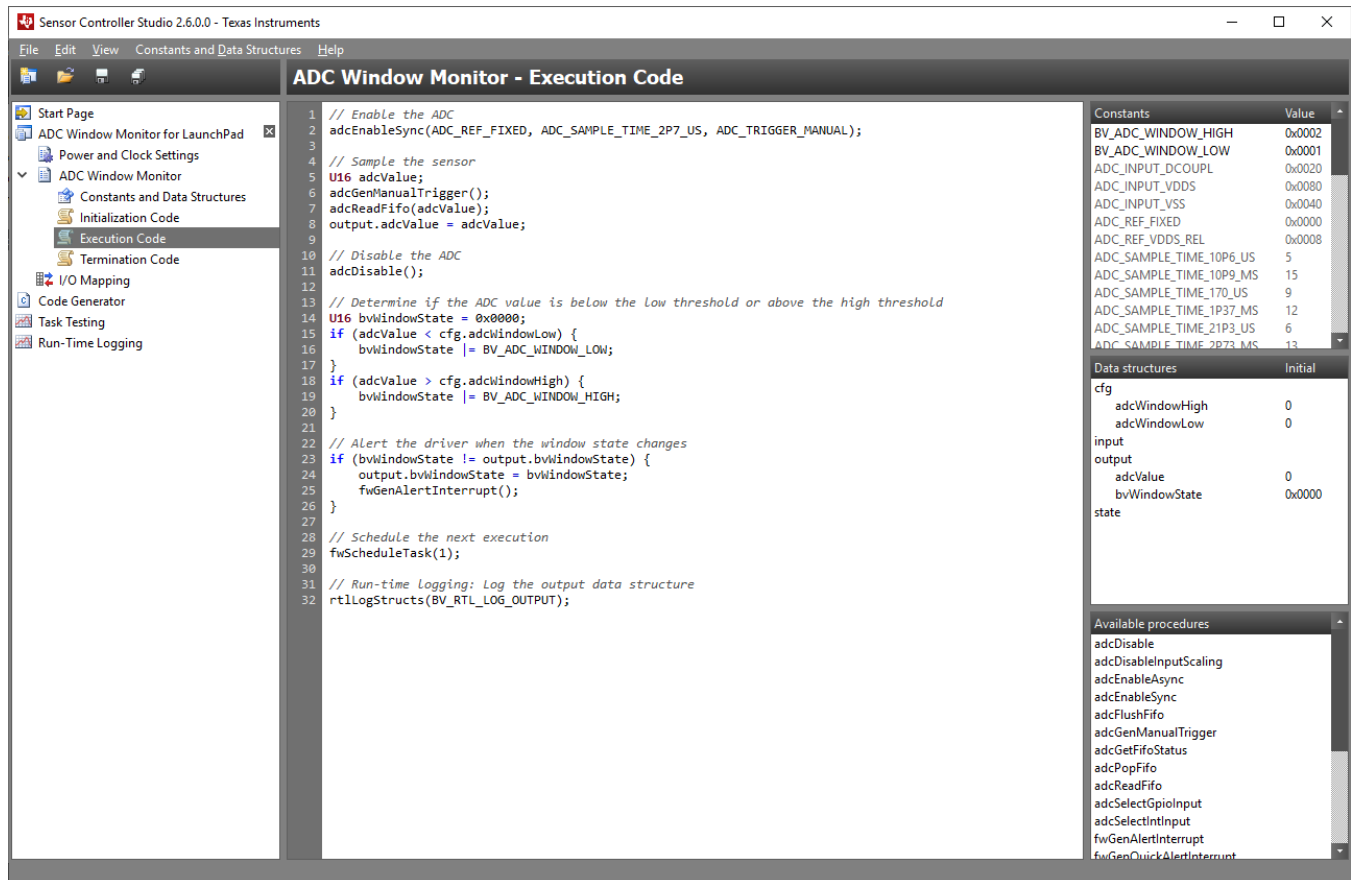


Figure 9. Task Code Editor Panel

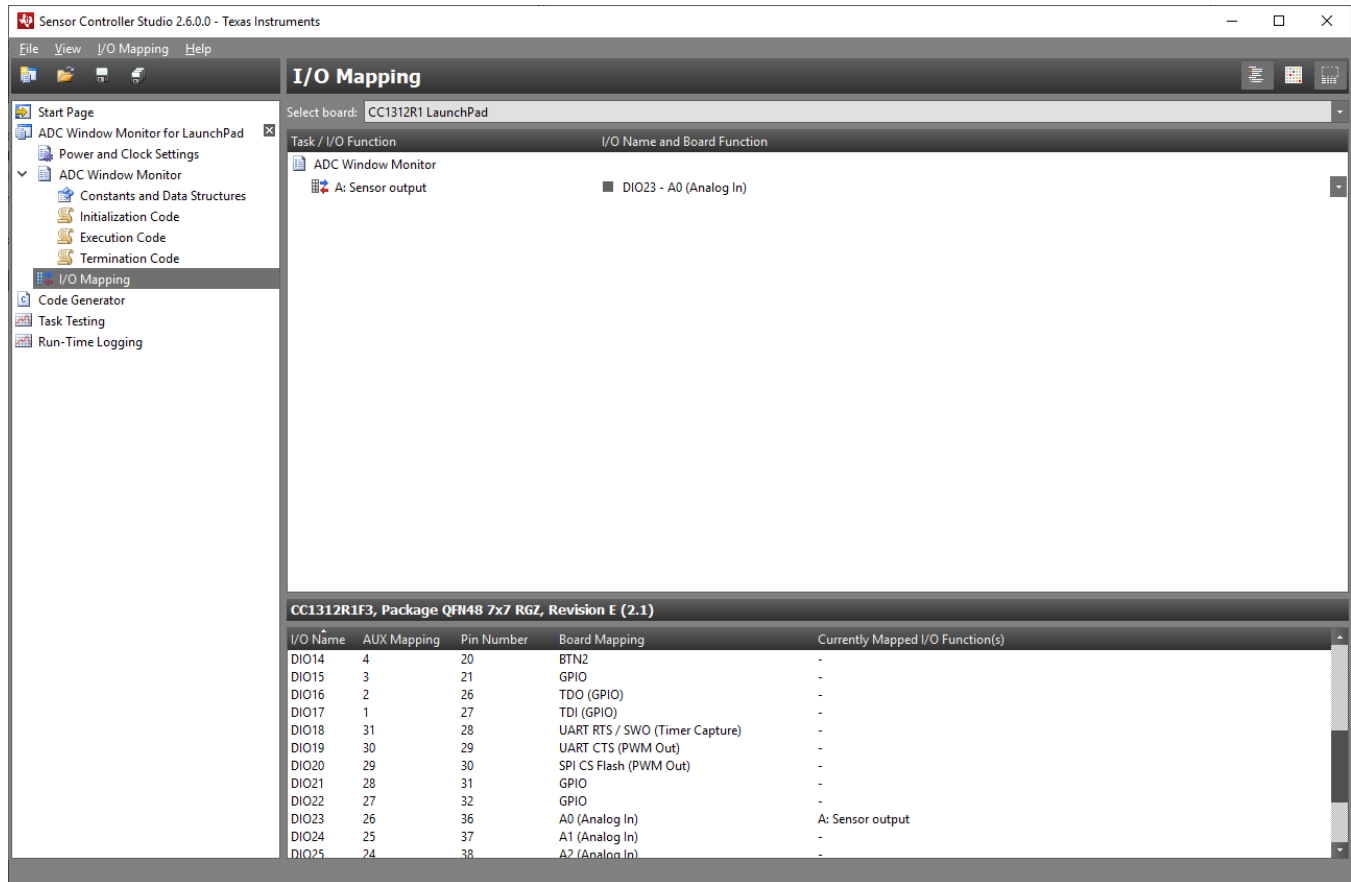
To the right (see [Figure 9](#)) are lists that show the following:

- Relevant constants with values for use in the task code
- The task data structure members with initial values
- Procedures that can be called from the task code

Pop-up documentation is displayed when typing the parameters and return values of a procedure. The pop-ups can also be triggered by moving the cursor to the procedure call and pressing **Ctrl+Space**.

## 6.8 I/O Mapping Panel

The I/O mapping panel is used to map the I/O functions enabled in the task panel to DIO pins on the selected target chip. The generated SCIF driver performs all necessary configuration for these pins (see [Figure 10](#)).



**Figure 10. I/O Mapping Panel**

Use the board selection to ease the I/O mapping.

There are two I/O mapping views:

- List mode (default): Select I/O pins for each I/O function from the drop-down lists
- Grid mode: Click on the cells in the grid to map I/O pins to I/O functions

Press Ctrl+Tab to switch view.

It is possible to share I/O pins between Sensor Controller tasks. For more details, see the panel help documentation.

## 6.9 Code Generator Panel

The code generator panel validates project and task settings, compiles task code, and outputs the SCIF driver (see [Figure 11](#)).

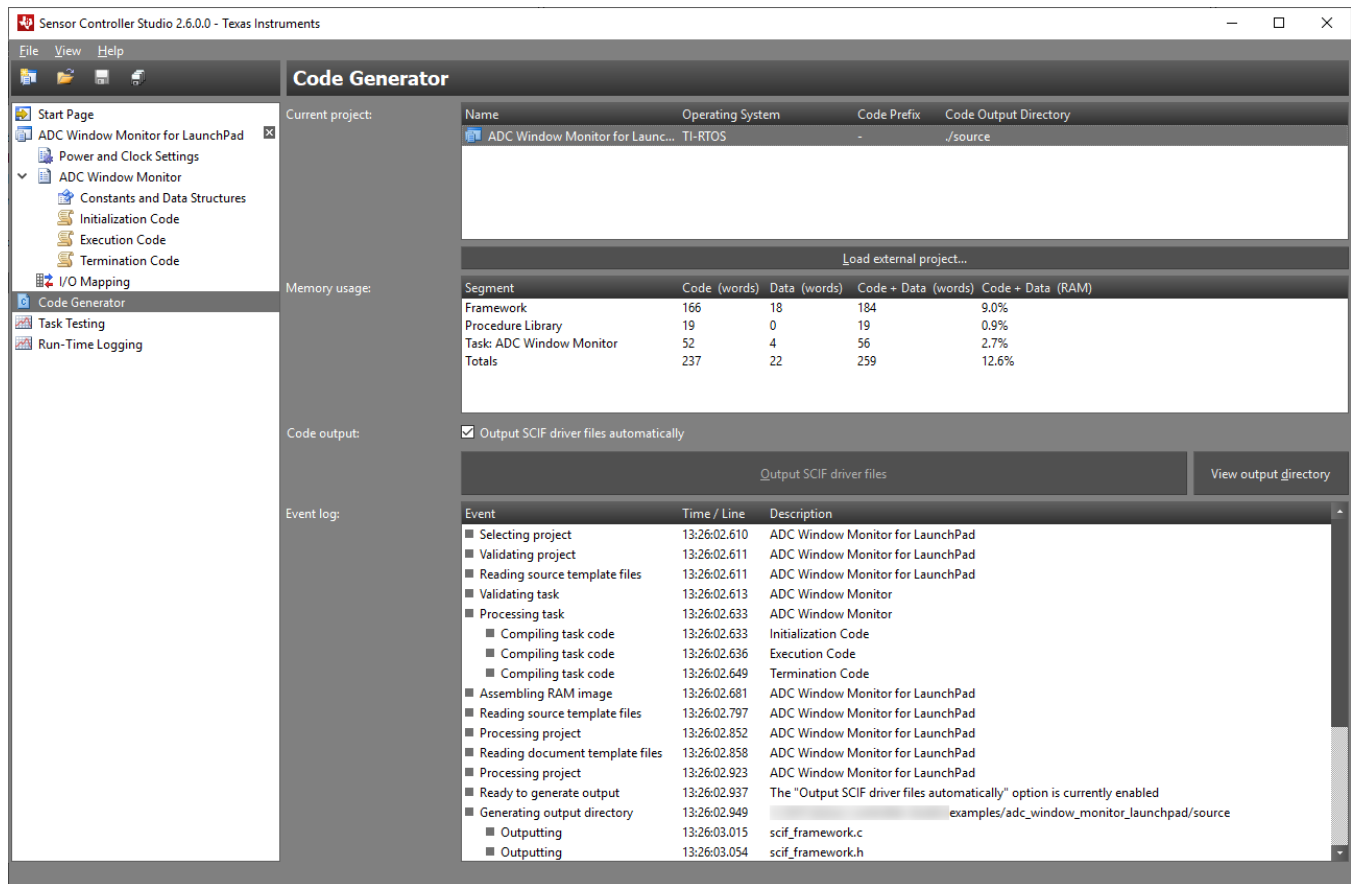


Figure 11. Code Generator Panel

When only one project is open, this project will be selected automatically. The code generator is triggered by:

- Entering the panel
- Changing the project selection
- Double-clicking on the selected project in the current project list

The event log will indicate errors with red icons , if there are any.

If AUX RAM image generation is successful, the memory usage will be displayed. This can be used to determine whether the tasks will fit, adjust data buffer sizes, and optimize the task code.

To trigger code output after successful code generation, either select the *Output SCIF driver files automatically* option, or press the *Output SCIF driver files* button. Press the *View output directory* button to explore the file output location.

The output includes a tailored "how-to-use" guide.

The SCIF driver is documented using Doxygen syntax. The output includes a doxyfile (Doxygen project file) and a Windows batch file for running Doxygen.

## 6.10 Compiling Example Applications in IAR or CCS

Source and project files for the example application are located in subdirectories next to the Sensor Controller project file (\*.scp).

To compile example projects, use the following:

- For CC13x0 and CC26x0 devices:
  - IAR EWARM 7.80.4 or later
  - TI CCS 7.1.0 or later
- For CC13x2 and CC26x2 devices:
  - IAR EWARM 8.20.1 or later
  - TI CCS 7.3.0 or later

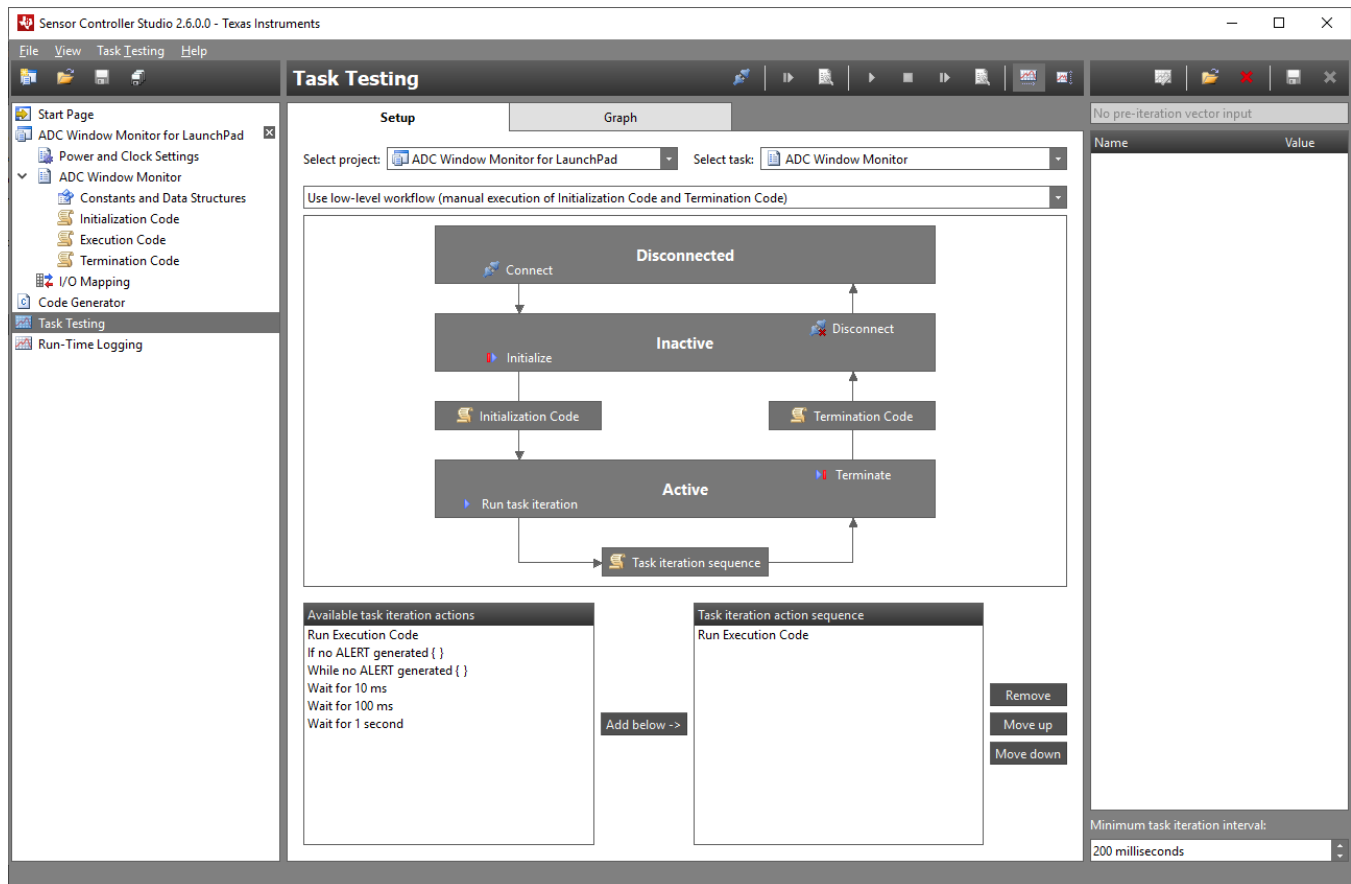
The ADC Window Monitor example comes with two different project files for IAR EWARM and CCS:

- **iar/ccs**: Operating system "None", without power management
- **iar\_tirtos/ccs\_tirtos**: Operating system "TI-RTOS", with power management



## 6.11 Task Testing Panel

The task testing panel can be used to test a Sensor Controller task on a physical CC26xx or CC13xx device (see [Figure 12](#)).



**Figure 12. Task Testing, Setup Tab**

During task testing, the Sensor Controller Studio takes over the role of the System CPU application, and it interacts with the Sensor Controller task through an XDS100, an XDS110, or an XDS200 JTAG debug probe.

The task testing panel runs whole iterations of the Initialization Code, Execution Code, Event Handler Code, and Termination Code task code blocks, and captures all task data structure values between task iterations. The data structure values are displayed graphically in the Graph tab of the task testing panel, and the values can be saved to file for external analysis.

It is also possible to automatically apply new data structure values that are loaded from file between specific task iterations.

For low-level debugging of a task, it is possible to run single iterations of the task code blocks in debug mode.

### 6.11.1 Task Testing Setup

At the top of the Setup tab, select the project and the task to trigger code generation. The code generation event log is displayed only if there are any errors.

Select either the low-level or simplified workflow, and observe the state diagram that is displayed in the middle of the Setup tab.

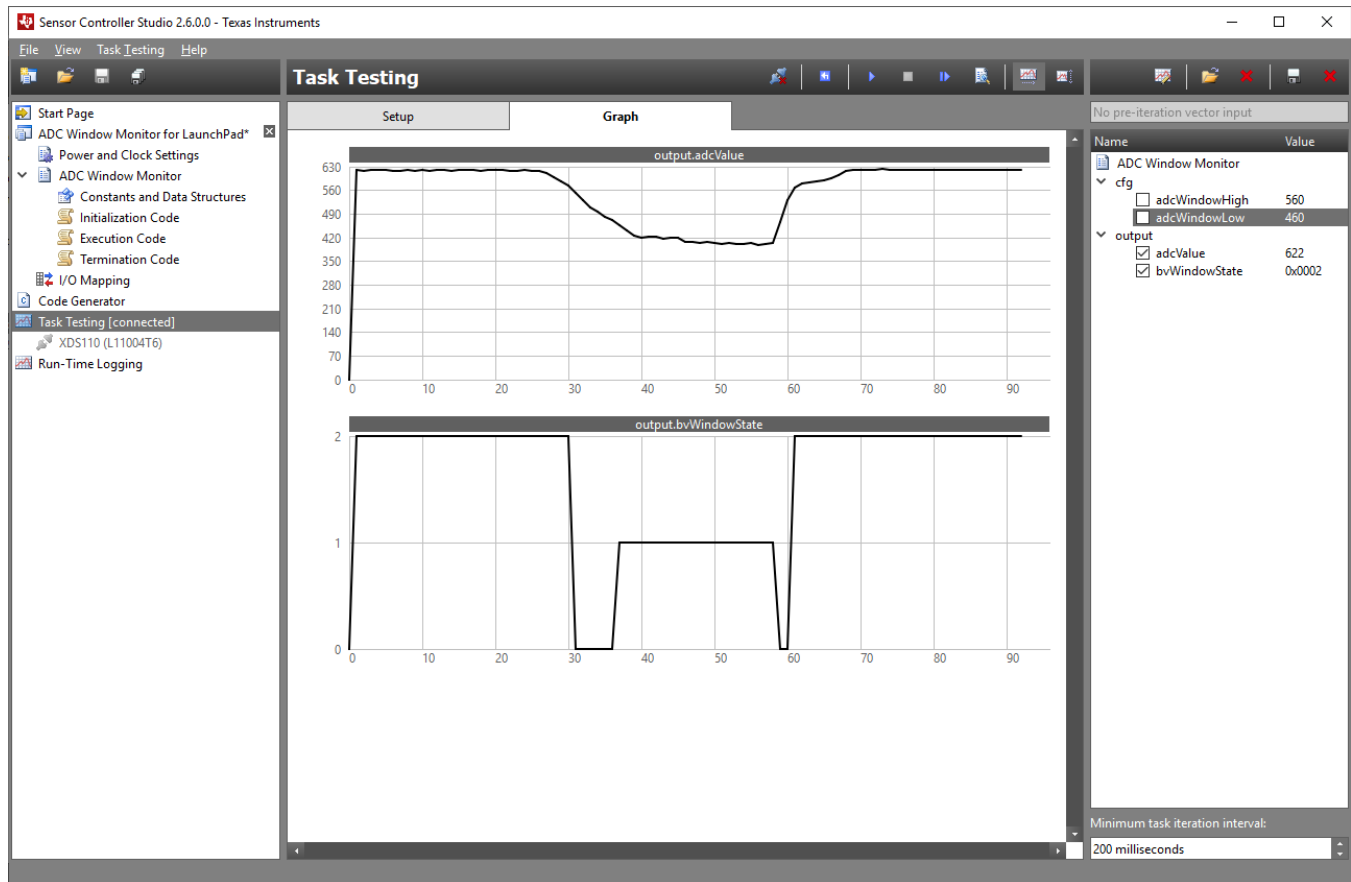
At the bottom of the Setup tab, specify a sequence of actions to be performed for each task iteration. Enter "Run Execution Code" for the ADC Window Monitor example.

**NOTE:** Each task code block in the action sequence runs at full speed. However, for most iteration actions, there is need for handshaking or data transfer over JTAG, which means that the task will not execute with the same timing during task testing as it will in the actual application.

### 6.11.2 Task Testing Session

Attach the target CC13xx or CC26xx device to the PC through an XDS100v3, an XDS110, or an XDS200 JTAG probe. Ensure that the correct target chip is selected in the project panel, then connect to the target.

When connected, the panel switches to the Graph tab, as shown in [Figure 13](#).



**Figure 13. Task Testing, Graph Tab**

In the right section of the panel, select the data structure members to be displayed in the Graph tab. There is one graph section for each data structure member because the value ranges typically vary. Use *Customize Graphs* to merge different data structure members into one graph section, with scaling and offset.

It is possible to:

- Single-step task iterations
- Run task iterations continuously
- Run single task iterations in debug mode (see [Section 6.11.4](#))

Use the **Ctrl** and **Shift** keys in combination with the mouse wheel to zoom and scroll the graph area.

### 6.11.3 Data Handling

Data structure values can be edited while the target is not running. In the right section of the panel:

1. Double-click on one of the values
2. Type the new value
3. Press Enter

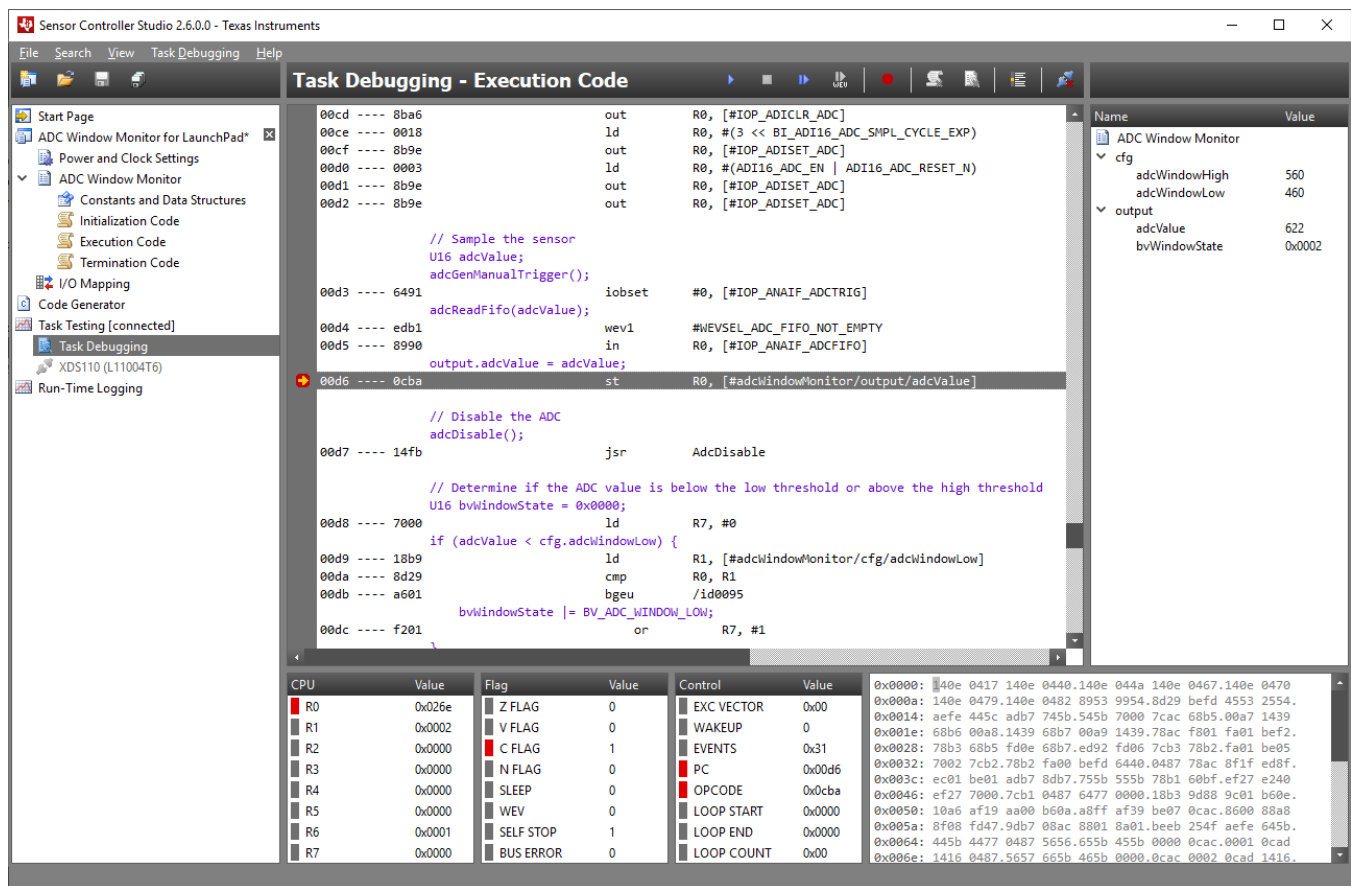
These modifications can also be automated by loading pre-iteration vectors (data structure values to be applied before task iterations) from a comma separated value (CSV) file:

- Load all or a subset of the data structure members. The first row specifies the member name for each column (for example, "cfg.adcWindowHigh" or "input.something.[2]").
- Vector 0 (second row) is applied before running the Initialization code, and then vector N is applied before the Nth task iteration.
- A special, optional first column "index" can be added to specify when each vector shall be applied (index N corresponds to vector N).

Similarly, it is possible to save post-iteration vectors (data structure values captured after each task iteration) to a CSV file. This can be used to transfer results to a spreadsheet, script, or other tool for further processing.

### 6.11.4 Task Debugging Panel

During task testing, it is possible to run single iterations of the Initialization, Execution, Event Handler, and Termination Code blocks in debug mode. Task debugging supports single-stepping and running with breakpoints in assembly code (see [Figure 14](#)).



**Task Debugging - Execution Code**

```

00cd ---- 8ba6      out      R0, [#IOP_ADICLR_ADC]
00ce ---- 0018      ld        R0, #(3 << BI_ADI16_ADC_SMPL_CYCLE_EXP)
00cf ---- 8b9e      out      R0, [#IOP_ADISET_ADC]
00d0 ---- 0003      ld        R0, #(ADI16_ADC_EN | ADI16_ADC_RESET_N)
00d1 ---- 8b9e      out      R0, [#IOP_ADISET_ADC]
00d2 ---- 8b9e      out      R0, [#IOP_ADISET_ADC]

// Sample the sensor
U16 adcValue;
adcGenManualTrigger();

00d3 ---- 6491      iobset   #0, [#IOP_ANAIF_ADCTRIG]
adcReadFifo(adcValue);

00d4 ---- edb1      wev1     #WEVSEL_ADC_FIFO_NOT_EMPTY
00d5 ---- 8990      in        R0, [#IOP_ANAIF_ADCFIFO]

output.adcValue = adcValue;

00d6 ---- 0cba      st        R0, [#adcWindowMonitor/output/adcValue]

// Disable the ADC
adcDisable();

00d7 ---- 14fb      jsr      AdcDisable

// Determine if the ADC value is below the low threshold or above the high threshold
U16 bvWindowState = 0x0000;

00d8 ---- 7000      ld        R7, #0
if (adcValue < cfg.adcWindowLow) {
00d9 ---- 18b9      ld        R1, [#adcWindowMonitor/cfg/adcWindowLow]
00da ---- 8d29      cmp      R0, R1
00db ---- a601      bgeu     /id0095
    bvWindowState |= BV_ADC_WINDOW_LOW;
00dc ---- f201      or       R7, #1
}

```

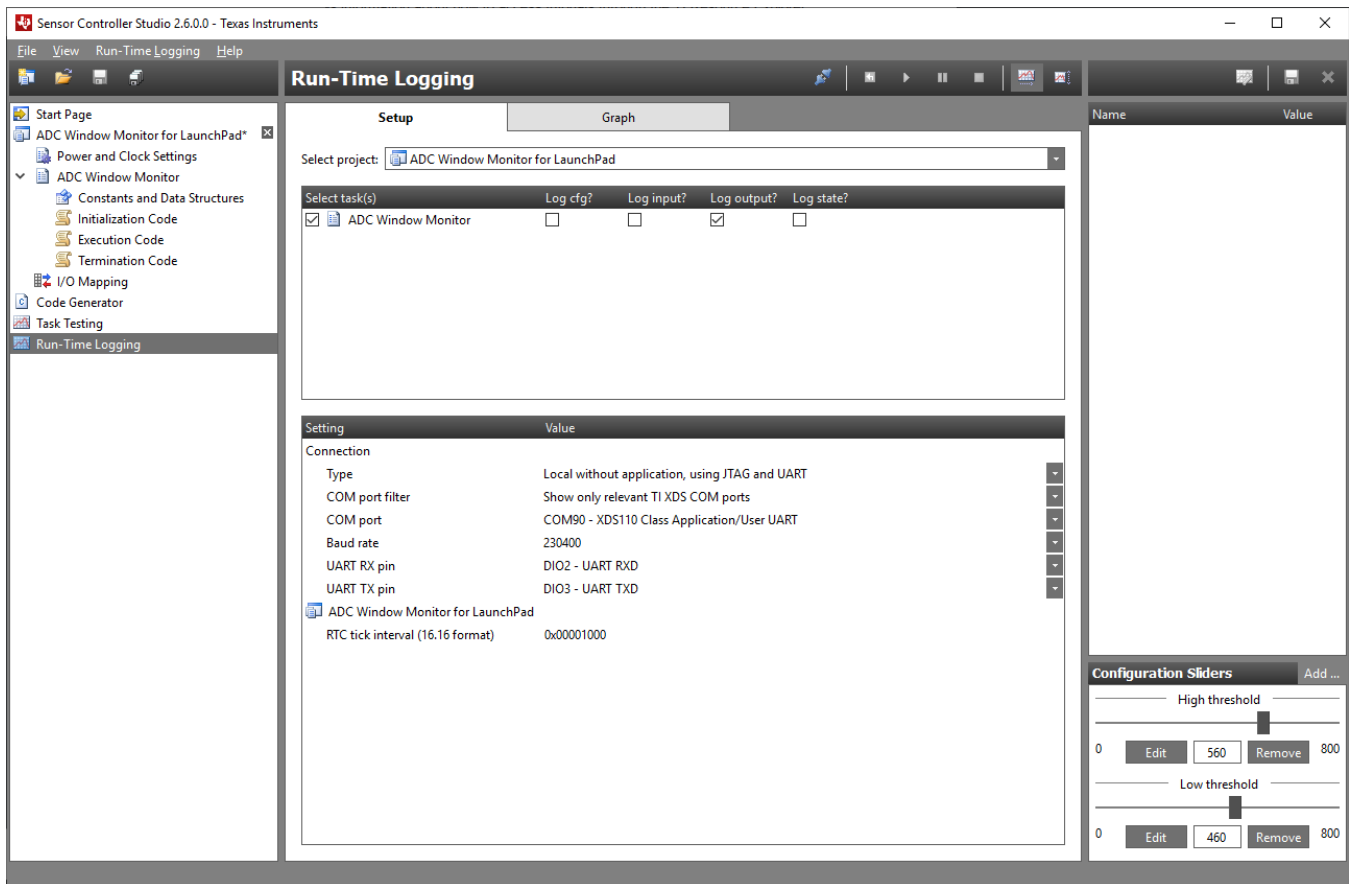
Name	Value
ADC Window Monitor	
cfg	
adcWindowHigh	560
adcWindowLow	460
output	
adcValue	622
bvWindowState	0x0002

CPU	Value	Flag	Value	Control	Value
R0	0x026e	Z FLAG	0	EXC VECTOR	0x00
R1	0x0002	V FLAG	0	WAKEUP	0
R2	0x0000	C FLAG	1	EVENTS	0x31
R3	0x0000	N FLAG	0	PC	0x00d6
R4	0x0000	SLEEP	0	OPCODE	0x0cba
R5	0x0000	WEV	0	LOOP START	0x0000
R6	0x0001	SELF STOP	1	LOOP END	0x0000
R7	0x0000	BUS ERROR	0	LOOP COUNT	0x00

Figure 14. Task Debugging Panel

## 6.12 Run-Time Logging Panel

The Run-Time Logging panel is used to evaluate and optimize performance of Sensor Controller tasks running at full speed on a physical CC26xx or CC13xx device (see [Figure 15](#)).



**Figure 15. Run-Time Logging Panel**

During run-time logging, the Sensor Controller runs autonomously while the System CPU transfers data structure information to and from Sensor Controller Studio. The logged data structures can be displayed graphically in Sensor Controller Studio, or the data can be saved to file for external analysis.

The JTAG interface is used to download a generic System CPU application to the target. The run-time logging panel then uses UART to transfer commands and data.

### 6.12.1 Run-Time Logging Setup

Run-Time Logging must be manually enabled for each task. Enable the Run-Time Logging resource in the Task panel, and use the run-time logging procedures in the task code to trigger data structure logging to Sensor Controller Studio.

At the top of the Setup tab, select the following:

- A project
- The task or tasks to be enabled for run-time logging
- Which data structures shall be logged or be editable

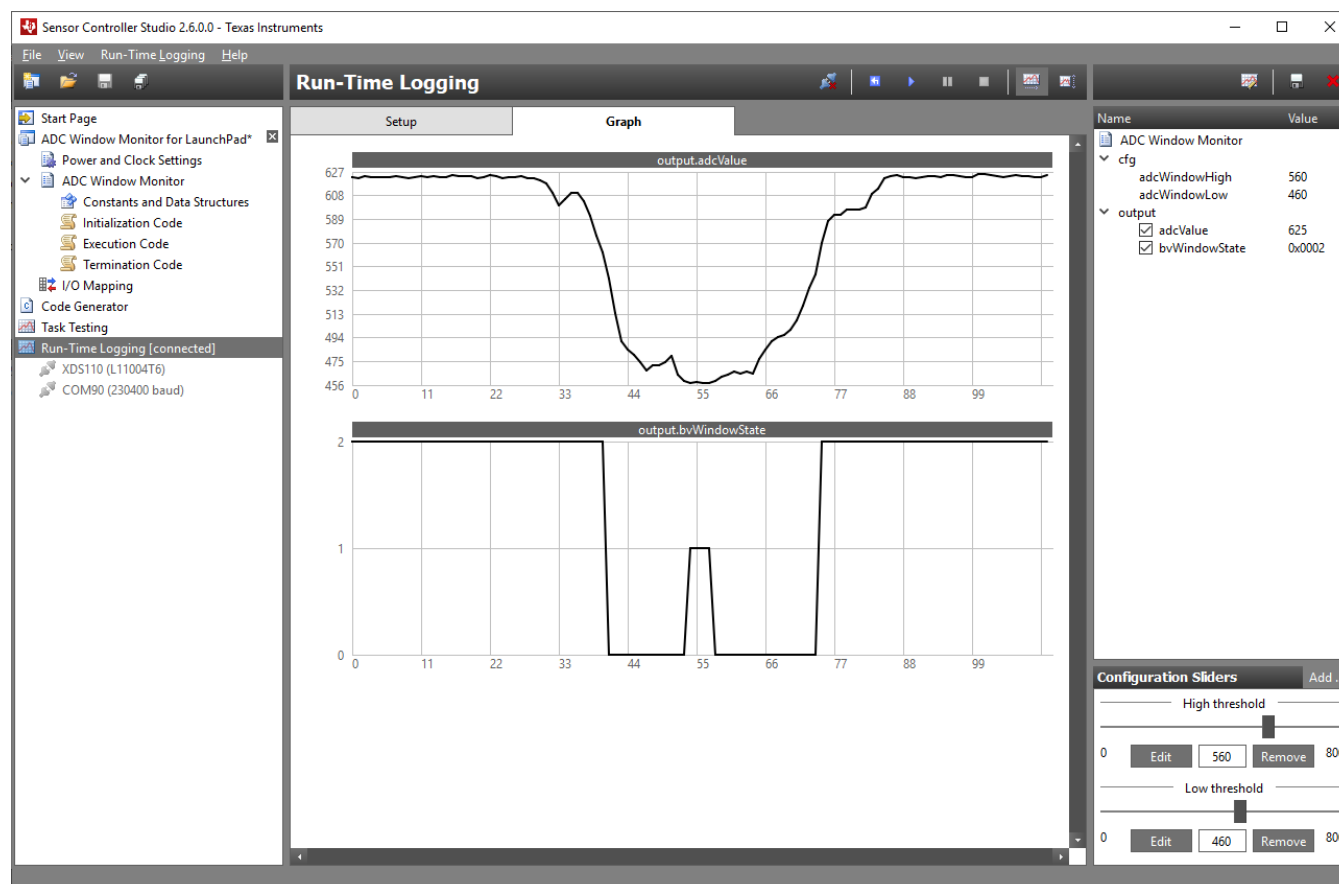
Attach the target CC13xx or CC26xx device to the PC through an XDS100v3 or an XDS110 JTAG probe (for example, a LaunchPad Development Kit).

Configure connection settings and project specific settings, if any. Project settings control behavior that is normally configured by the System CPU application, for example RTC tick interval.

## 6.12.2 Run-Time Logging Session

Ensure that the correct target chip is selected in the project panel, then connect to the target.

When connected, the panel switches to the Graph tab, as shown in [Figure 16](#).



**Figure 16. Run-Time Logging Graph Tab**

In the right section of the panel, select the logged data structure members to be displayed in the Graph tab. There is one graph section for each data structure member because value ranges typically vary. Use graph customization to merge different data structure members into one graph section with scaling and offset.

Editable data structure members can be modified at any time. The cfg data structure members can also be controlled by configuration sliders.

For run-time logging, it is only possible to run the Sensor Controller tasks at full speed.

Use the **Ctrl** and **Shift** keys in combination with the mouse wheel to zoom and scroll the graph area.

## 7 References

- [CC13x0, CC26x0 SimpleLink™ Wireless MCU Technical Reference Manual](#)
- [CC13x2, CC26x2 SimpleLink™ Wireless MCU Technical Reference Manual](#)

## Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

<b>Changes from J Revision (July 2019) to K Revision</b>	<b>Page</b>
• The version of this document is updated from 2.5 to 2.6. ....	<a href="#">2</a>
• Updated Clock Source and Frequency in the Sensor Controller and AUX Domain table. ....	<a href="#">5</a>
• Updated Start Page image.....	<a href="#">13</a>
• Updated Help Viewer Example Page image. ....	<a href="#">14</a>
• Updated Example Configuration: ADC Window Monitor image.....	<a href="#">15</a>
• Updated Project Settings Panel image. ....	<a href="#">16</a>
• Updated Task Panel image. ....	<a href="#">17</a>
• Updated Constants and Data Structures Panel image.....	<a href="#">18</a>
• Updated Task Code Editor Panel image.....	<a href="#">19</a>
• Updated I/O Mapping Panel image. ....	<a href="#">20</a>
• Updated Code Generator Panel image. ....	<a href="#">21</a>
• Updated Task Testing, Setup Tab image.....	<a href="#">23</a>
• Updated Task Testing, Graph Tab image. ....	<a href="#">24</a>
• Updated Task Debugging Panel image. ....	<a href="#">25</a>
• Updated Run-Time Logging Panel image. ....	<a href="#">26</a>
• Updated Run-Time Logging Graph Tab image. ....	<a href="#">27</a>



## IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to TI's Terms of Sale ([www.ti.com/legal/termsofsale.html](http://www.ti.com/legal/termsofsale.html)) or other applicable terms available either on [ti.com](http://ti.com) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2020, Texas Instruments Incorporated