

CC26xx/CC13xx Sensor Controller Studio Version 1.3

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

Contents

1	Introduction	2
2	Overview	2
3	Sensor Controller Studio Installation (Windows 7/8.1/10)	5
4	Prerequisites.....	6
5	Tutorial: Analog Light Sensor	6
6	References	19

List of Figures

1	Start Page	7
2	Example Configuration	8
3	Project Settings	9
4	Light Sensor Task Settings	10
5	Light Sensor Task Code.....	11
6	I/O Mapping	12
7	Code Generator Panel.....	13
8	Task Testing	14
9	Task Testing Operations	15
10	Task Testing Data Handling	17
11	Task Debugging Panel	18
12	Task Debugging	19

Code Composer Studio is a trademark of Texas Instruments.
 Keil is a registered trademark of ARM Germany GmbH and ARM Ltd.
 IAR Embedded Workbench is a registered trademark of IAR Systems AB.
 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/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/CC13xx Sensor Controller and the Sensor Controller Studio tool.

Prerequisites for the generated driver source code and the supplied examples are stated in [Section 4](#).

The tutorial in [Section 5](#) 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/CC13xx Sensor Controller. The tool generates a Sensor Controller Interface driver, which is a set of C source files that are compiled into the System CPU (ARM Cortex-M3) application. These source files contain the Sensor Controller firmware image and allow the System CPU application to control and exchange data with the Sensor Controller.

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/CC13xx auxiliary (AUX) power/clock domain, and can perform simple background tasks autonomously and independent of the System CPU and MCU domain power state. These **tasks** include such as:

- Analog sensor polling, using the ADC or comparator
- Digital sensor polling, using bit-banged SPI, I2C or other protocols
- Capacitive sensing, using the integrated current source, comparator and time-to-digital converter (TDC)

The Sensor Controller is user programmable, using a language that is very similar to C, and allows for sensor polling and other tasks to be specified as sequential algorithms. This replaces 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
- Improved debuggability

The Sensor Controller algorithms can efficiently be tested, debugged and verified using Sensor Controller Studio, allowing the generated source code to be integrated painlessly into the System CPU application.

2.1 Sensor Controller and AUX Domain Hardware

The Sensor Controller is located in the CC26xx/CC13xx AUX domain, and has access to:

- AUX RAM (instruction and data memory for the Sensor Controller)
 - 2 kB = 1024 Sensor Controller 16-bit instruction/data words
- Analog peripherals:
 - ADC (12-bit analog to digital converter, 200 kHz maximum sample rate)
 - COMPA (continuous time comparator)
 - COMPB (low-power clocked comparator)
 - ISRC (0-20 μ A current source)
- Digital peripherals:
 - TDC (high-precision time to digital converter)
 - AUX timers 0 and 1 (16-bit synchronous timers)
 - AUX I/O controllers
 - Hardware semaphores (used to share peripherals between the Sensor Controller and System CPU)

- I/O pins
 - I/O pins
 - Up to 8 digital-only general-purpose I/O (GPIO) pins

The Sensor Controller does not have access to MCU- or AON-domain peripherals, RAM, flash or registers. This allows the MCU domain to enter and exit standby mode independently of the Sensor Controller. The System CPU, on the other hand, has access to all AUX domain peripherals and the AUX RAM.

There are two event signals from the Sensor Controller to the MCU domain:

- **READY** indicates that the control interface used to start or stop Sensor Controller tasks is idle and ready.
- **ALERT** is used by Sensor Controller tasks to wake up the System CPU from standby and exchange data.

The Sensor controller is in active mode when running task code, and is 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.

Table 1 shows the Sensor Controller/AUX domain clock source and frequency for the different modes of operation:

Table 1. Sensor Controller/AUX Domain Clock Source and Frequency

	System CPU/MCU Domain Active	System CPU/MCU Domain Standby
Sensor Controller Active	SCLK_HF / 2 = 24 MHz	SCLK_HF / 2 = 24 MHz
Sensor Controller Standby With Event Trigger	SCLK_HF / 2 = 24 MHz	SCLK_LF = 32 kHz
Sensor Controller Standby Without Event Trigger	SCLK_HF / 2 = 24 MHz	No clock

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**, containing:
 - 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

2.3 Sensor Controller Programming Model

Driver setups are created in Sensor Controller Studio **projects**. Each project may contain one or more Sensor Controller **tasks**, for example, instance capacitive sensing and ADC measurement. The output of a project is a single SCIF driver.

For each task, a set of resources is selected and configured to enable different types of functionality:

- Analog and digital peripheral modules (ADC)
- Analog and digital general-purpose I/O (GPIO) pins
- Simple and complex software algorithms
- Bit-banged serial interfaces
- System functionality for task scheduling and event handling, System CPU communication, and so forth.

The task algorithm is implemented using a programming language similar to C.

The selected task resources enable sets of procedures (equivalent to functions in C), with associated constants and variables. The task code calls 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 or depend on resource defined constants.

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 firmware framework sets the Sensor Controller in a low-power standby mode when it is idle.

2.4 Sensor Controller Tasks

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 System CPU's and DMA's memory address space. 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 task's state 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 structures (if the System CPU wakes up periodically).

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

The task algorithm is divided into three or four code blocks:

- **Initialization Code**: Runs once 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 or sets up an *Event Handler Code* trigger
 - **Event Handler Code** (limited to one task per project): Triggered either by edge or level on a single AUX I/O pin, or after a variable delay with 4 kHz resolution
 - 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
 - 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 once 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

2.5 Task Testing and Debugging

Sensor Controller Studio provides a generic, quick and 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/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, which can be displayed graphically or 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.

3 Sensor Controller Studio Installation (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 (XDS100, 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".

3.1 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 TI-RTOS 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 Prerequisites

4.1 For Generated SCIF Driver

The generated Sensor Controller Interface driver depends on register definitions and OS-specific functionality:

- For OSAL None: CC13xxWare or CC26xxWare (included in the TI-RTOS for CC13xx and CC26xx installer)
- For OSAL TI-RTOS: TI-RTOS for CC13xx and CC26xx

The driver can be compiled using:

- IAR Embedded Workbench® for ARM
- TI Code Composer Studio™
- Keil® MDK-ARM
- ARM GCC

4.2 For Examples

Examples with associated files are configured and patched when opened through the Sensor Controller Studio Start Page panel. This allows support for different target chips (when supported by the example's hardware platform) and for different releases of TI-RTOS for CC13xx and CC26xx. Example project files are provided for the IAR EWARM and CCS toolchains.

The example application project files require one of the following:

- IAR EWARM 7.40.3 or later
- TI CCS 6.1.0 or later

5 Tutorial: Analog Light Sensor

This is a step-by-step walkthrough of the “Analog Light Sensor” example, in which the Sensor Controller samples the SFH5711 light sensor on the SmartRF06 Evaluation Board (SmartRF06EB).

The ADC value range (0-4095) is divided into a configurable number of bins, with run-time configurable hysteresis and bin thresholds. If the bin index changes, the Sensor Controller alerts the System CPU application, which reads the bin index and enables the corresponding number of LEDs (0 to 4) on the SmartRF06EB.

The SmartRF06EB is required for [Section 5.10](#) and [Section 5.10.4](#).

5.1 Start Page

The Start Page provides easy access to projects, examples, documentation and online resources. The left part of the main window displays a project tree that is used for navigation. For efficient navigation, use the *Alt + Up/Down* keyboard shortcuts and also mouse back and forward buttons.

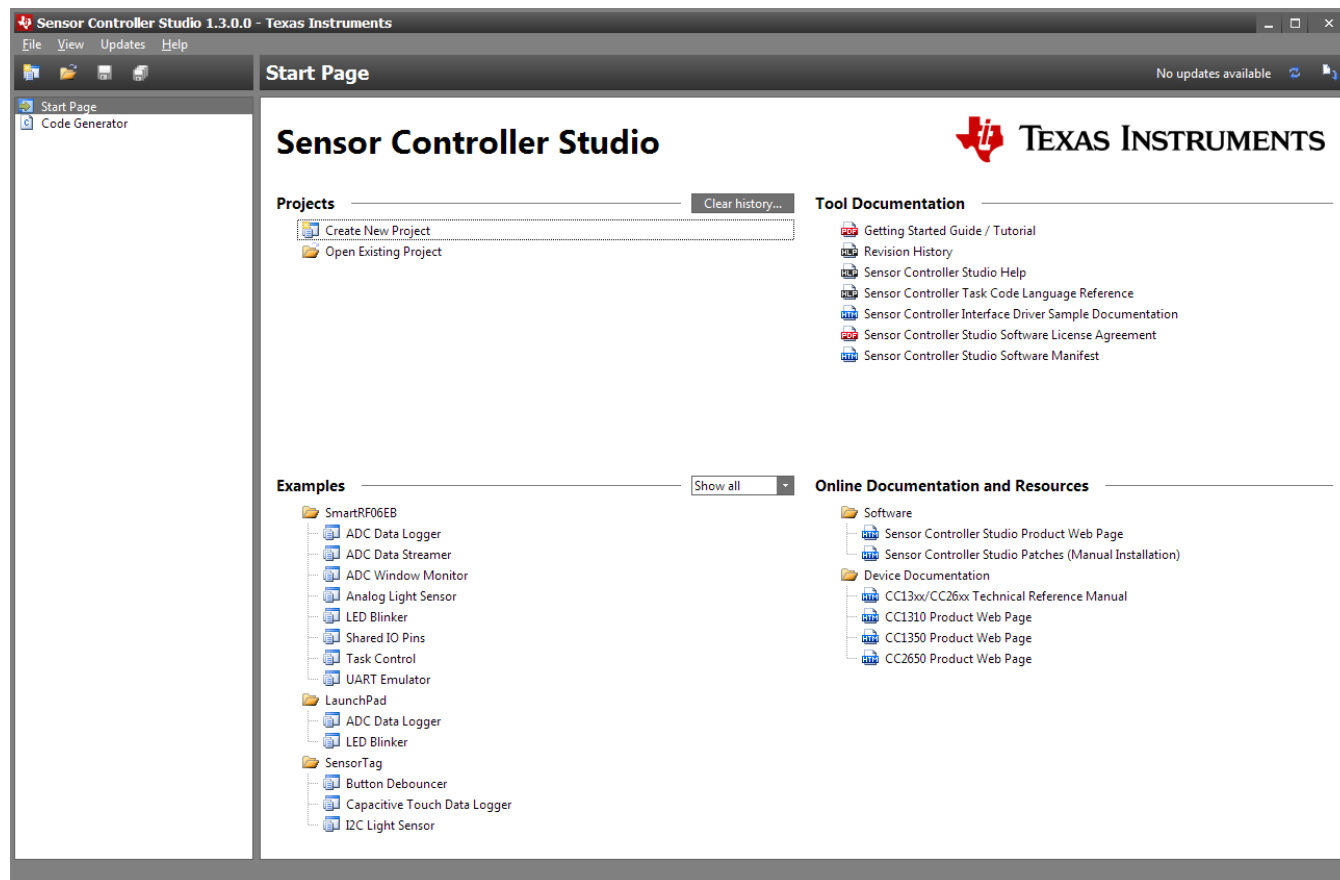


Figure 1. Start Page

Help documentation is accessible at any time by pressing *F1*.

5.2 Open the Example

Open the example configuration window by double-clicking on “Analog Light Sensor” under “Examples” on the Start Page.

- This will display the example configuration window, which allows the examples to work seamlessly with:
 - Different target chips (if the example's hardware platform allows it)
 - Releases of the TI-RTOS application run-time framework

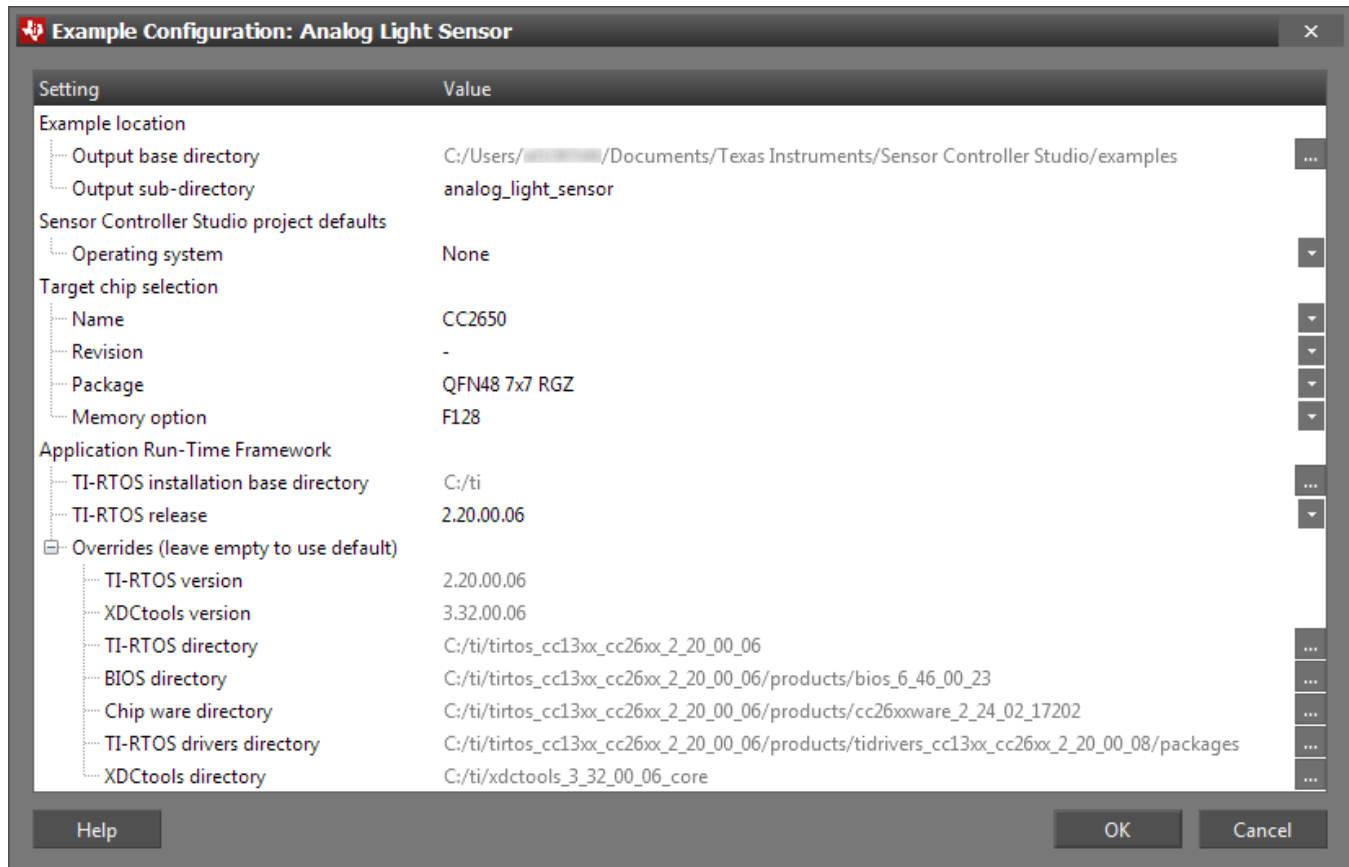


Figure 2. Example Configuration

- Check that the settings for the target chip name and TI-RTOS release are correct, and update if necessary.
- Press **OK**.

This will patch and output the Sensor Controller project application source code and project files.

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, re-run the example configuration to create new application project files.

5.3 Review Project Settings

Select the project panel and review the settings.

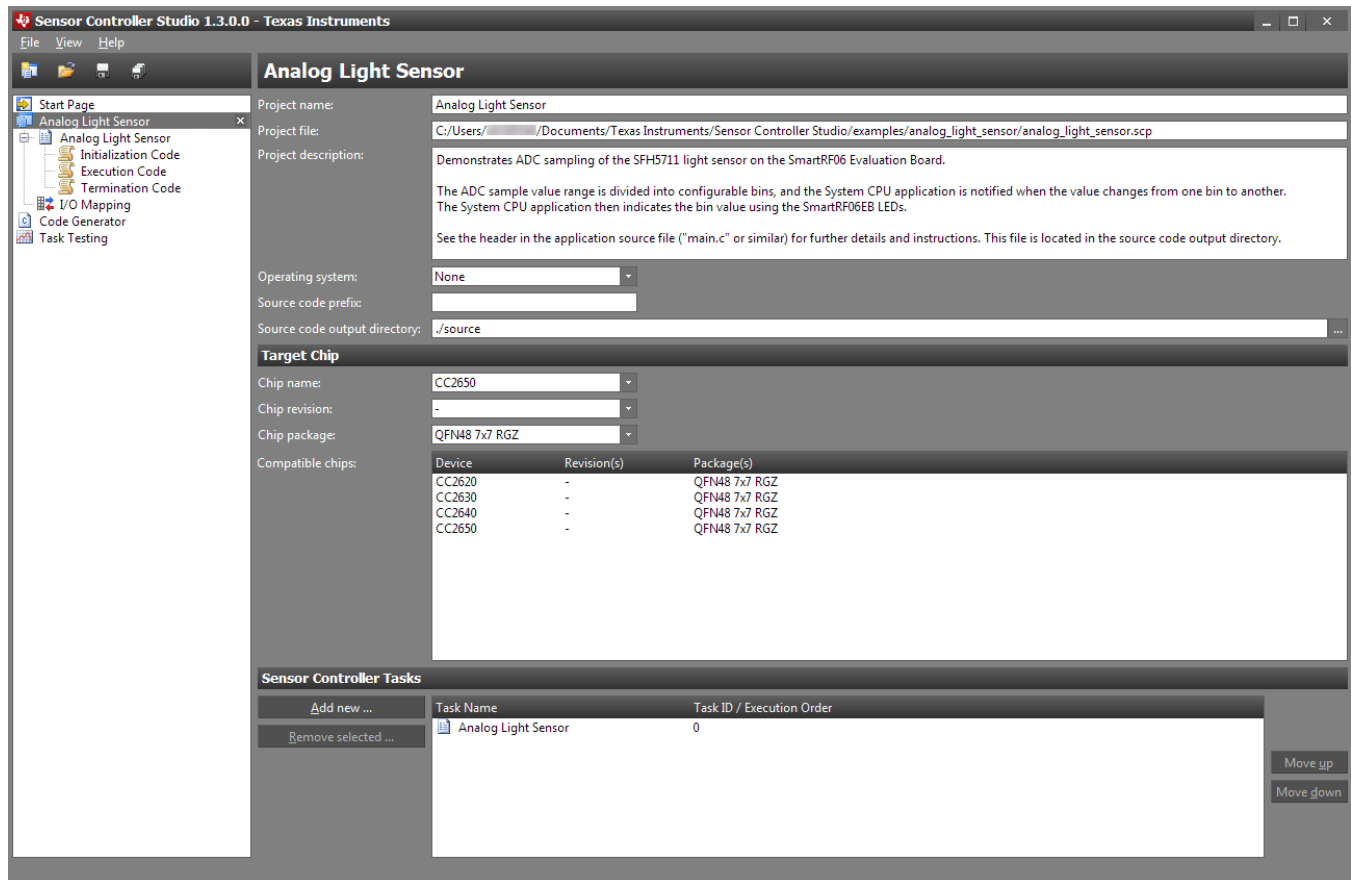


Figure 3. Project Settings

The project settings include:

- Project name and description, which are displayed in the header of the generated driver setup
- Operating system: Select the operating system that affects which OSAL files are output.
- Source code prefix (optional): Specify a (preferably) short prefix, which is added to relevant parts of the driver setup (including file names). This allows for multiple driver instances 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 target chip by specifying name, revision and package. This choice affects I/O mapping, and so forth.

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 that the *Save project* and *Close project* commands apply only to the currently selected project in the project tree.

5.4 Review Light Sensor Task Settings

Select the task panel and review the settings.

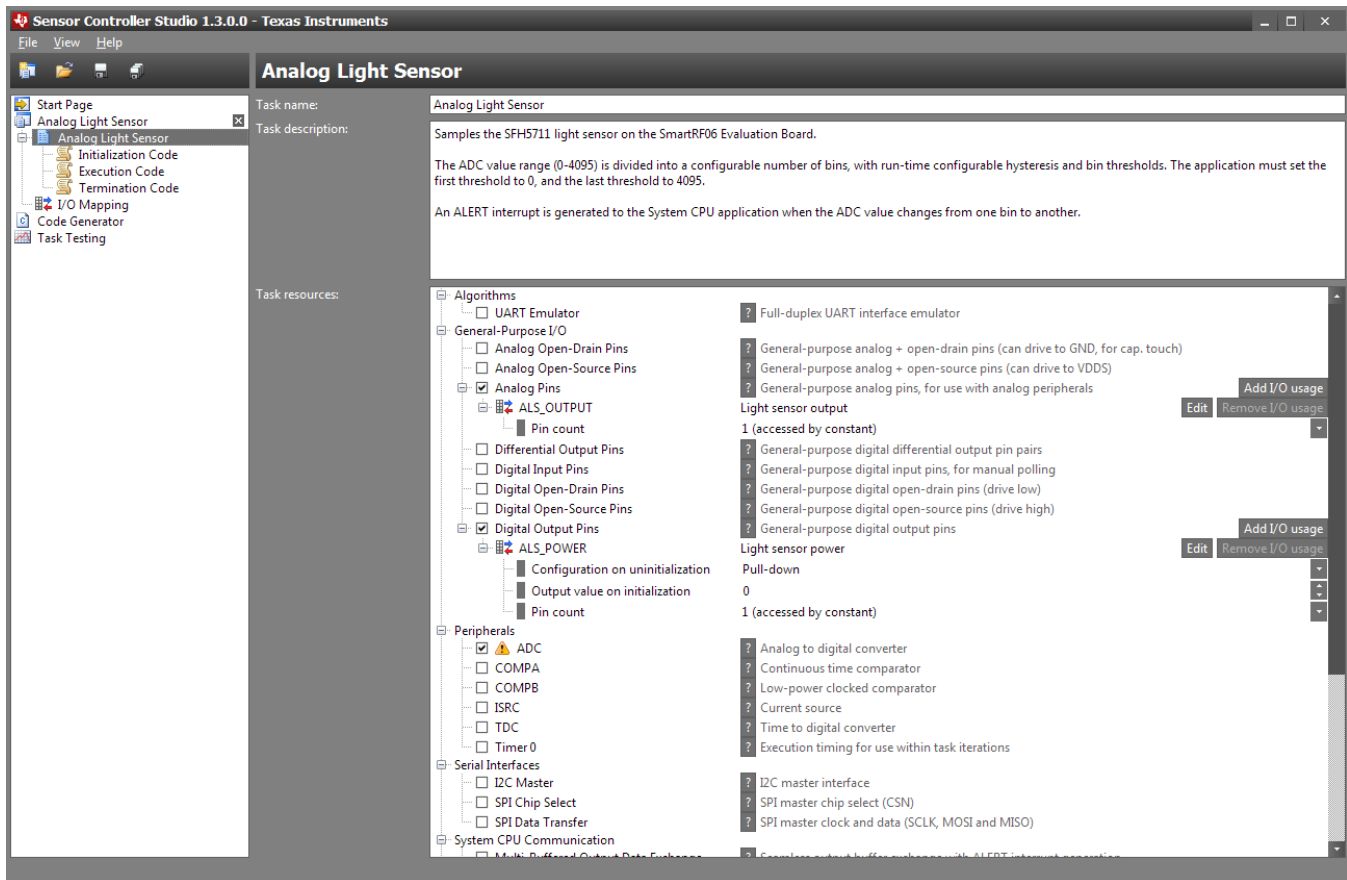


Figure 4. Light Sensor Task Settings

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.

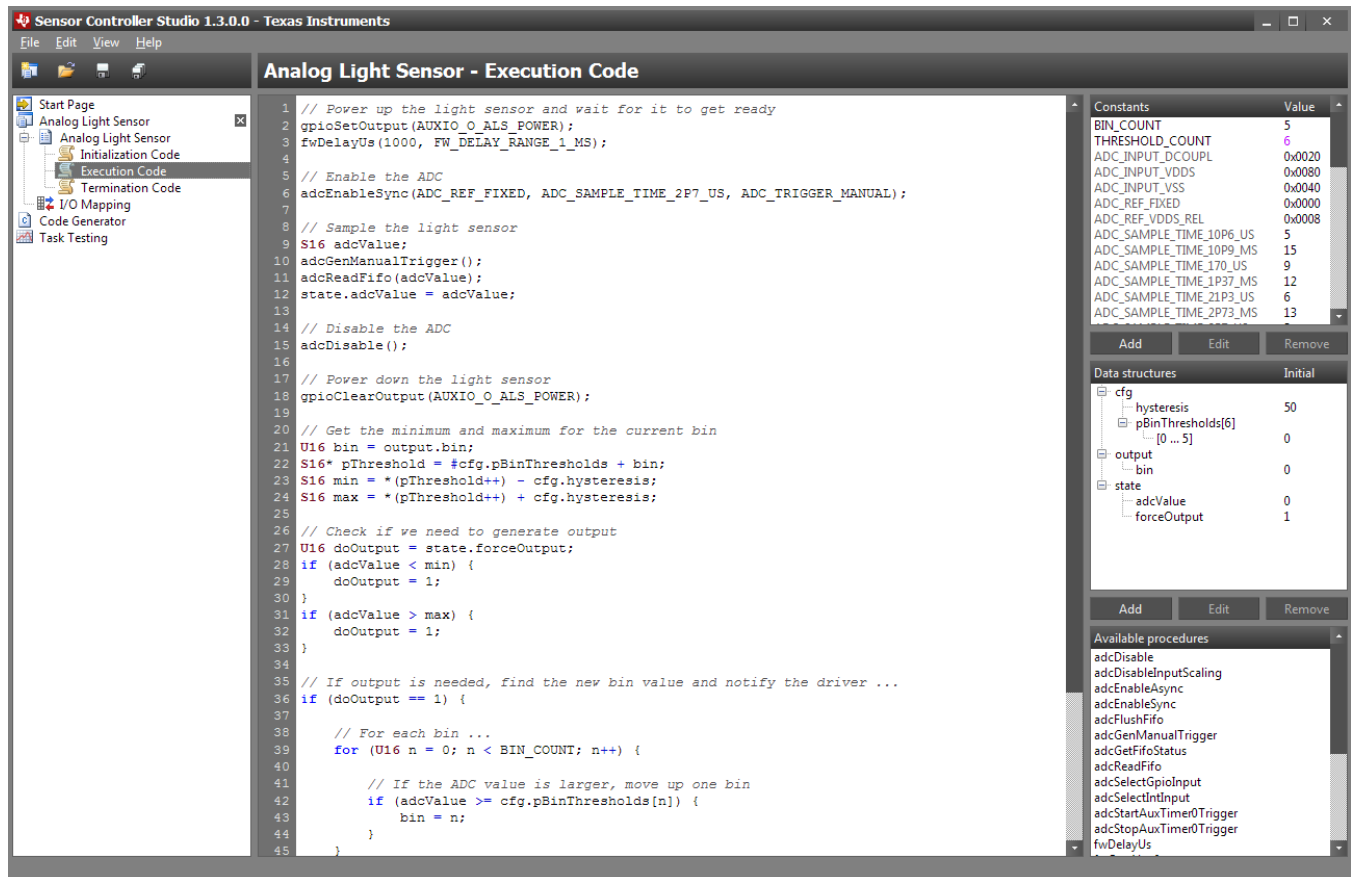
Task resources are selected to enable access to hardware modules, firmware framework functionality and various software algorithms. Some resources have configuration settings. For the "General Purpose I/O" resources, multiple usages with different name and configuration can be specified. Press the help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

For the Analog Light Sensor task the following resources are selected:

- **ADC:** to measure the output of the light sensor
- **Analog Pins** (single instance): to connect the light sensor output to the ADC
- **Digital Output Pins** (single instance): to control power to the light sensor
- **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

5.5 Review Light Sensor Task Code

Select the task code panels and review the initialization and execution code. The Termination code is not necessary for this task.



Analog Light Sensor - Execution Code

```

1 // Power up the light sensor and wait for it to get ready
2 gpioSetOutput(AUXIO_O_ALS_POWER);
3 fwDelayUs(1000, FW_DELAY_RANGE_1_MS);
4
5 // Enable the ADC
6 adcEnableSync(ADC_REF_FIXED, ADC_SAMPLE_TIME_2P7_US, ADC_TRIGGER_MANUAL);
7
8 // Sample the light sensor
9 S16 adcValue;
10 adcGenManualTrigger();
11 adcReadFifo(adcValue);
12 state.adcValue = adcValue;
13
14 // Disable the ADC
15 adcDisable();
16
17 // Power down the light sensor
18 gpioClearOutput(AUXIO_O_ALS_POWER);
19
20 // Get the minimum and maximum for the current bin
21 U16 bin = output.bin;
22 S16* pThreshold = #cfg.pBinThresholds + bin;
23 S16 min = *(pThreshold++) - cfg.hysteresis;
24 S16 max = *(pThreshold++) + cfg.hysteresis;
25
26 // Check if we need to generate output
27 U16 doOutput = state.forceOutput;
28 if (adcValue < min) {
29     doOutput = 1;
30 }
31 if (adcValue > max) {
32     doOutput = 1;
33 }
34
35 // If output is needed, find the new bin value and notify the driver ...
36 if (doOutput == 1) {
37
38     // For each bin ...
39     for (U16 n = 0; n < BIN_COUNT; n++) {
40
41         // If the ADC value is larger, move up one bin
42         if (adcValue >= cfg.pBinThresholds[n]) {
43             bin = n;
44         }
45     }
46 }

```

Constants

Constant	Value
BIN_COUNT	5
THRESHOLD_COUNT	6
ADC_INPUT_DCOUPL	0x0020
ADC_INPUT_VDD5	0x0080
ADC_INPUT_VSS	0x0040
ADC_REF_FIXED	0x0000
ADC_REF_VDD5_REL	0x0008
ADC_SAMPLE_TIME_10P6_US	5
ADC_SAMPLE_TIME_10P9_MS	15
ADC_SAMPLE_TIME_170_US	9
ADC_SAMPLE_TIME_1P37_MS	12
ADC_SAMPLE_TIME_21P3_US	6
ADC_SAMPLE_TIME_2P73_MS	13

Data structures

Data structure	Initial
cfg	
hysteresis	50
pBinThresholds[6]	
[0 ... 5]	0
output	
bin	0
state	
adcValue	0
forceOutput	1

Available procedures

- adcDisable
- adcDisableInputScaling
- adcEnableAsync
- adcEnableSync
- adcFlushFifo
- adcGenManualTrigger
- adcGetFifoStatus
- adcReadFifo
- adcSelectGpioInput
- adcSelectIntInput
- adcStartAuxTimer0Trigger
- adcStopAuxTimer0Trigger
- fwDelayUs

Figure 5. Light Sensor Task Code

The Sensor Controller Task Code Language Reference can be opened from the Start Page panel.

To the right are lists showing:

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

It is possible to add, edit and remove user-defined constants and data structure variables. The data structure variables are accessible from the System CPU application through C structs.

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**.

5.6 Review I/O Mapping

Select the I/O mapping panel and review the settings.

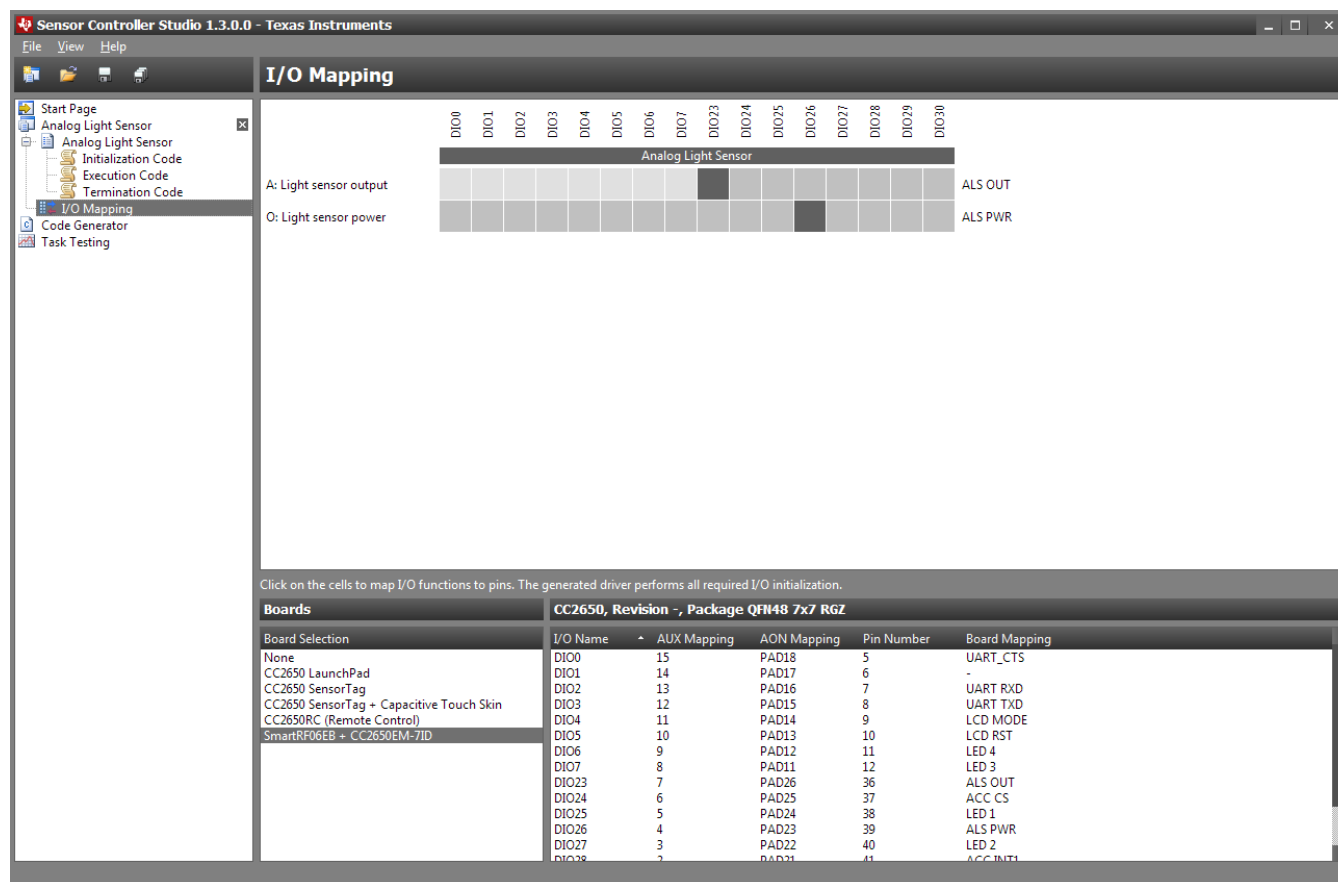


Figure 6. I/O Mapping

Map I/O functions (for example light sensor output) to I/O pins (for example DIO23) by clicking on the cells in the map. Note that some cells do not support analog signals. These cells are light grey. To ease setup for common development platforms, use the Board Selection.

5.7 Generate Code

Select the code generator panel (see [Figure 7](#)).

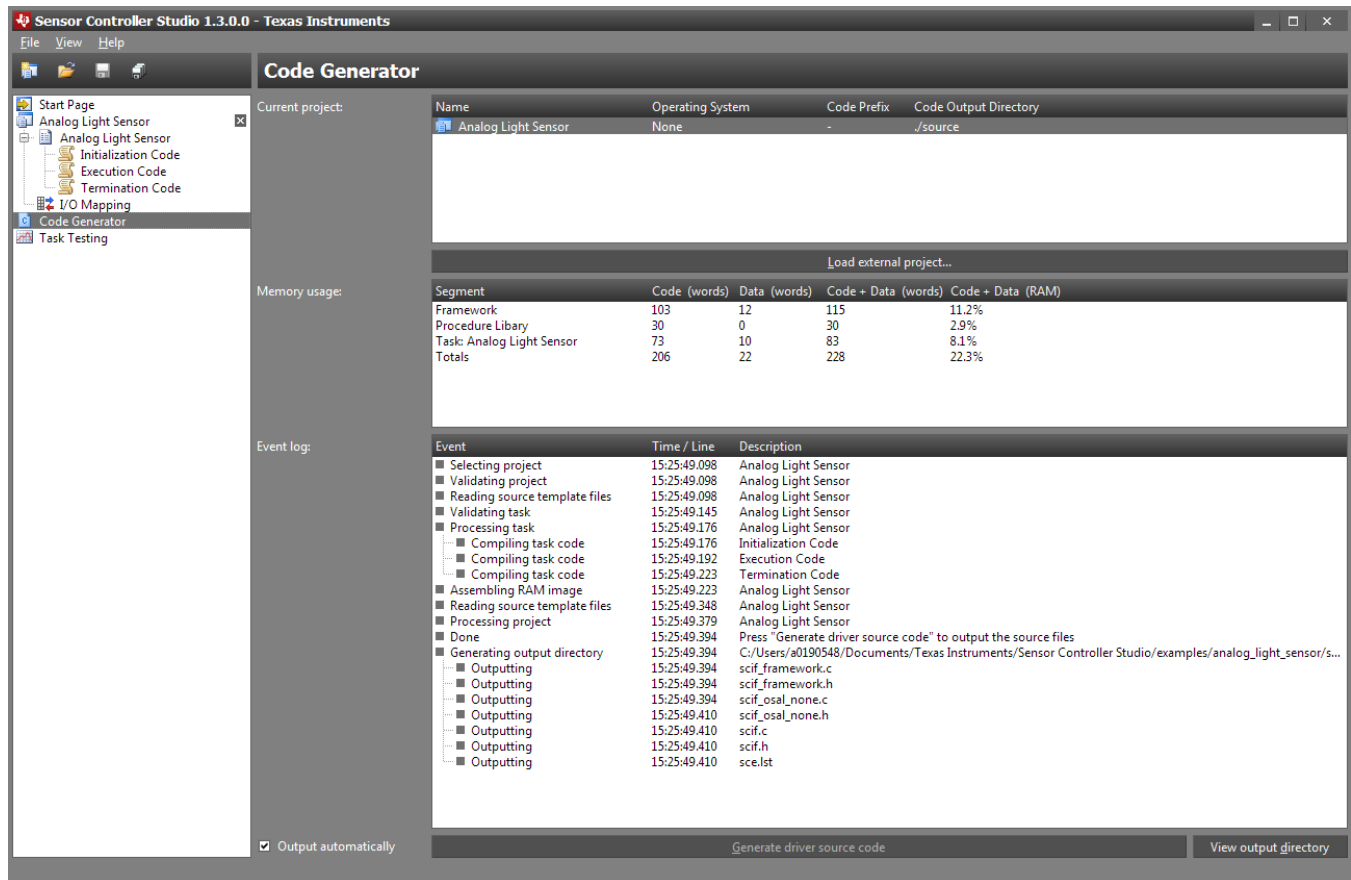


Figure 7. Code Generator Panel

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

- Entering the panel, in which case, it runs for the currently selected project
- Changing the project selection
- Double-clicking on the currently selected project

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 output the source files to be used with the System CPU application, press the *Generate driver source code* button. Use the *View output directory* button to quickly explore the file output location.

The generated SCIF driver is documented using Doxygen syntax. Sample documentation can be found in the start page panel.

5.8 Compiling the Application in IAR and CCS

The Analog Light Sensor example comes with two different project files for IAR EWARM and CCS:

- **iar/ccs:** Operating system "None", no power management
- **iar_tirtos/ccs_tirtos:** Operating system "TI-RTOS", configurable power management

Once Sensor Controller Studio has been launched, the example source and project files are located in <My Documents>\Texas Instruments\Sensor Controller Studio\examples\analog_light_sensor\project.

The Sensor Controller Studio example project is configured for one of the supported operating systems: "None" or "TI-RTOS". If the setting does not match the desired application project, change the *Operating system* setting in the Sensor Controller Studio project panel and re-run the code generator.

The examples require IAR EWARM 7.40.3 (or later) or CCS 6.1.0 (or later).

Select the desired version of the example application; compile and download it to the SmartRF06EB with the attached CC13xx or CC26xx EM.

5.9 Running the Application

The LEDs on the SmartRF06EB will indicate the intensity of the incoming light. In normally lit rooms, one or two LEDs should be on. Cover the light sensor to turn all LEDs off. Use a powerful light source (a mobile phone flash lamp) to turn on all the LEDs.

5.10 Task Testing

Select the Task Testing panel. This panel is only available when at least one project is open.

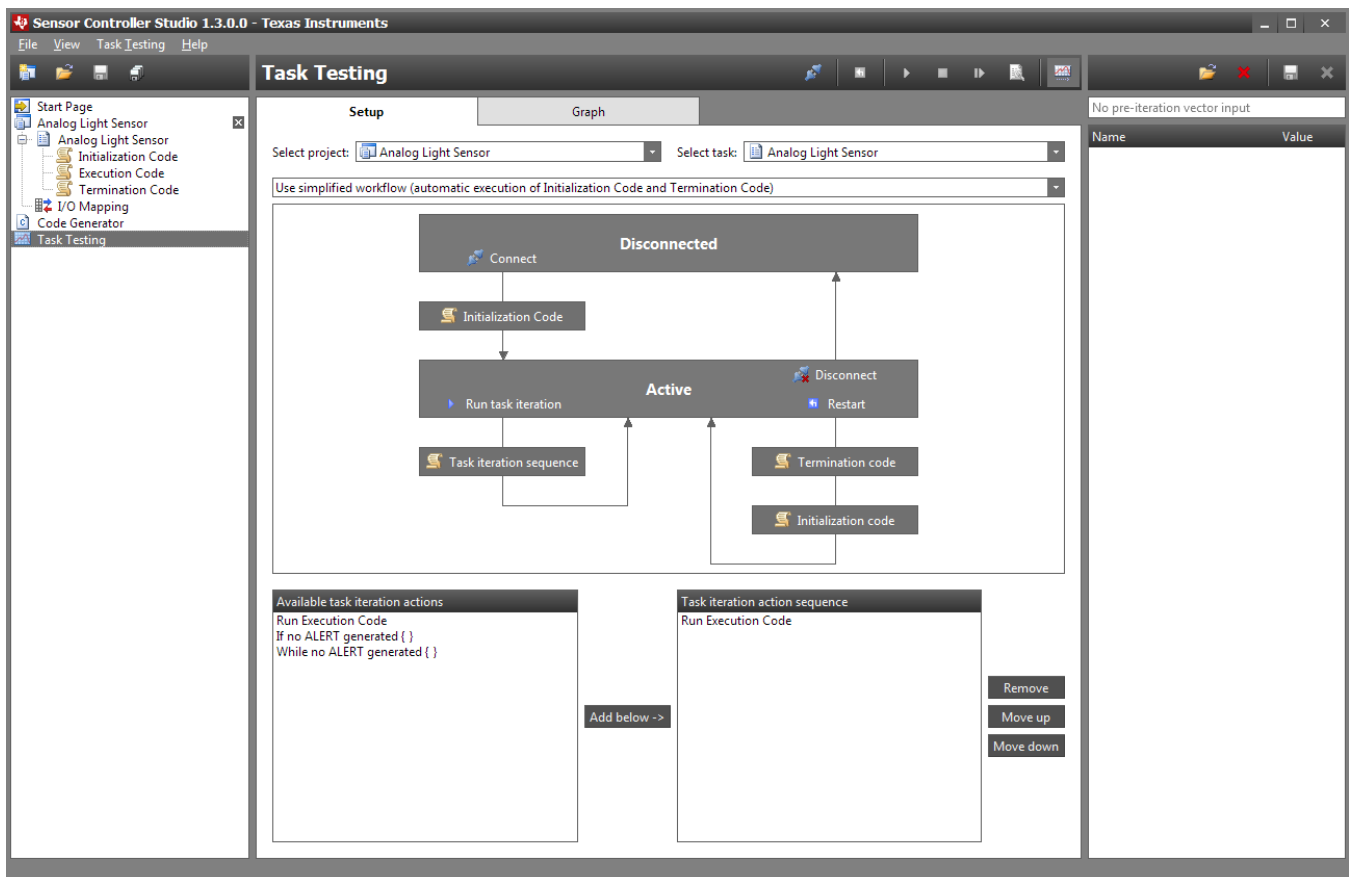


Figure 8. Task Testing

Entering the panel or selecting the project or task to be tested triggers code generation for the selected task only. The code generation log is displayed if there are any errors.

Select the simplified task testing workflow. This workflow is quicker to work with when testing task iterations, but does not allow for debugging of the initialization and termination code blocks.

The actions to be performed for each task iteration are set up in the bottom section of the panel. For the Analog Light Sensor, trigger **Run the Execution Code**.

5.10.1 Starting and Ending a Test Session

Use the menu, tool-bar, or shortcut key to **Connect** to the CC13xx/CC26xx devices through the XDS100, XDS110 or XDS200. If multiple JTAG debug probes are connected to the PC, a target selection dialog will be displayed.

Notice that the tool-bar button now changes to **Disconnect**. It is possible to end the test session at any time.

Task testing does not overwrite or modify the target's existing Flash memory contents. After disconnecting, reset or power-cycle the target to restart the flash application.

5.10.2 Task Testing Operations

When connected, the display changes.

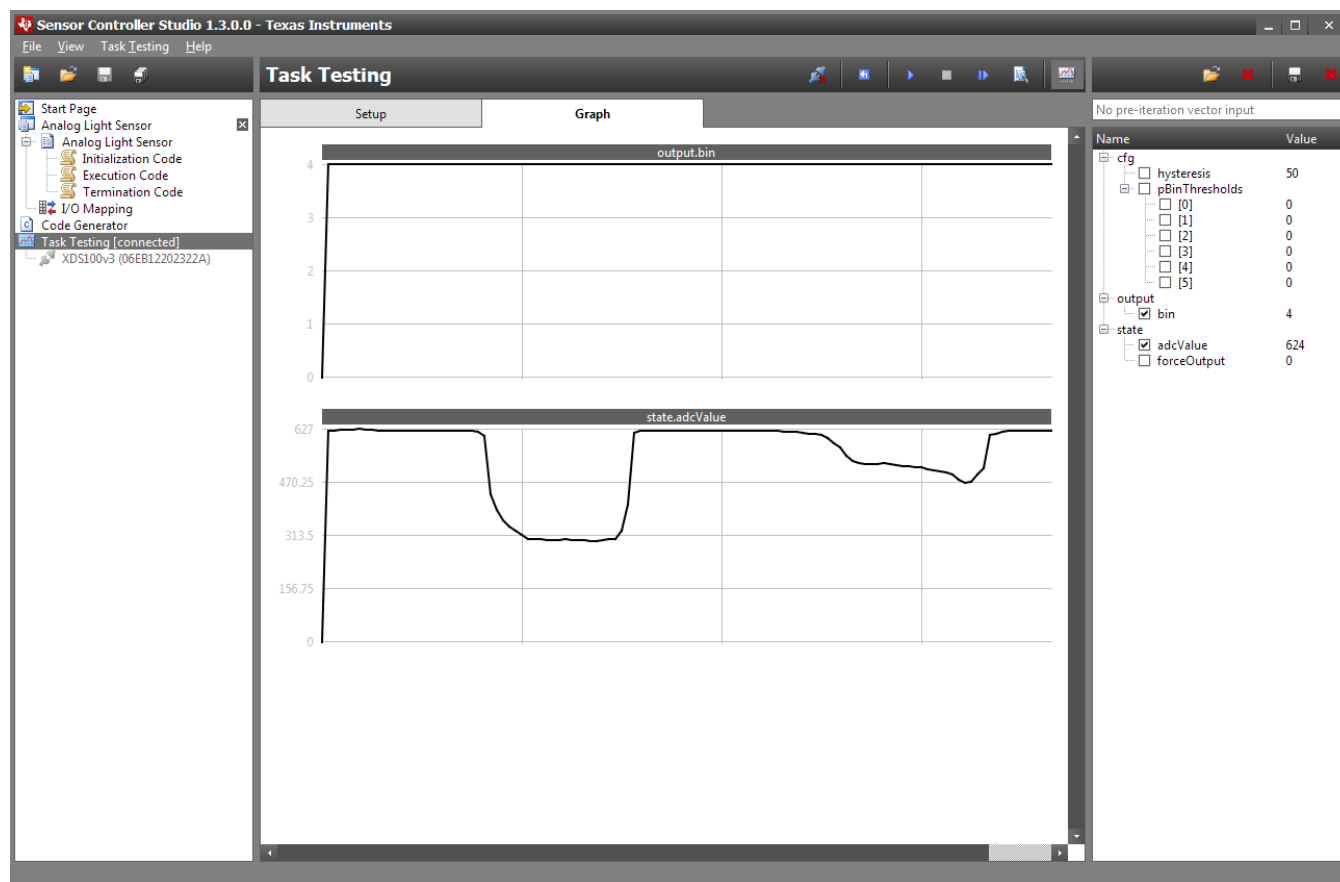


Figure 9. Task Testing Operations

The initialization begins to run when connected.

In the task data structures to the right, select output.bin and state.adcValue. To run the Analog Light Sensor task's execution code, trigger **Run Task Iterations Continuously**.

Move a dark object over the SmartRF06EB light sensor and observe `state.adcValue` dropping. The graph can be zoomed on the X-axis (using *Ctrl+Scroll*) and on the Y-axis (using *Shift+Scroll*). Auto-scrolling on the X-axis is enabled by default.

To stop the task iterations, trigger **Stop Continuous Task Iterations**.

5.10.3 Task Testing Data Handling

Notice that the bin value did not change due to all-zero bin thresholds.

It is possible to enter configuration parameters (and other data structure members) by double-clicking on the values, typing the new value, and pressing *Enter*. However, when working on a project and testing repeatedly, it is normally more efficient to use file-based data handling.

Pre-iteration data vectors (the data structure contents before each task code iteration) can be loaded from a CSV file. This can be used to load initial configuration and state, and change configuration and state during testing:

- Can load all or a subset of the data structure members. The first row specifies the member name for each column (for example, "cfg.hysteresis" or "cfg.pBinThresholds.[2]").
- Vector 0 is applied before running the Initialization code and then vector N before the N'th task iteration
- A special, optional first column "index" can be added to specify when each vector should be applied (index N corresponds to vector N)

Post-iteration data vectors (the data structure contents each task code iteration) can be saved to a CSV file. This can be used to transfer results to a spreadsheet or script for further processing or logging.

To load a suitable task configuration, access the *Task Testing* menu and trigger **Load Pre-Iteration Vector Input**. Select "<My Documents>\Texas Instruments\Sensor Controller Studio\examples\analog_light_sensor\task_testing\analog_light_sensor_cfg.csv".

The loaded CSV file takes effect when running the Initialization code. Trigger **Restart Task and Run Task Iterations Continuously** and observe that the `output.bin` now matches `state.adcValue`.

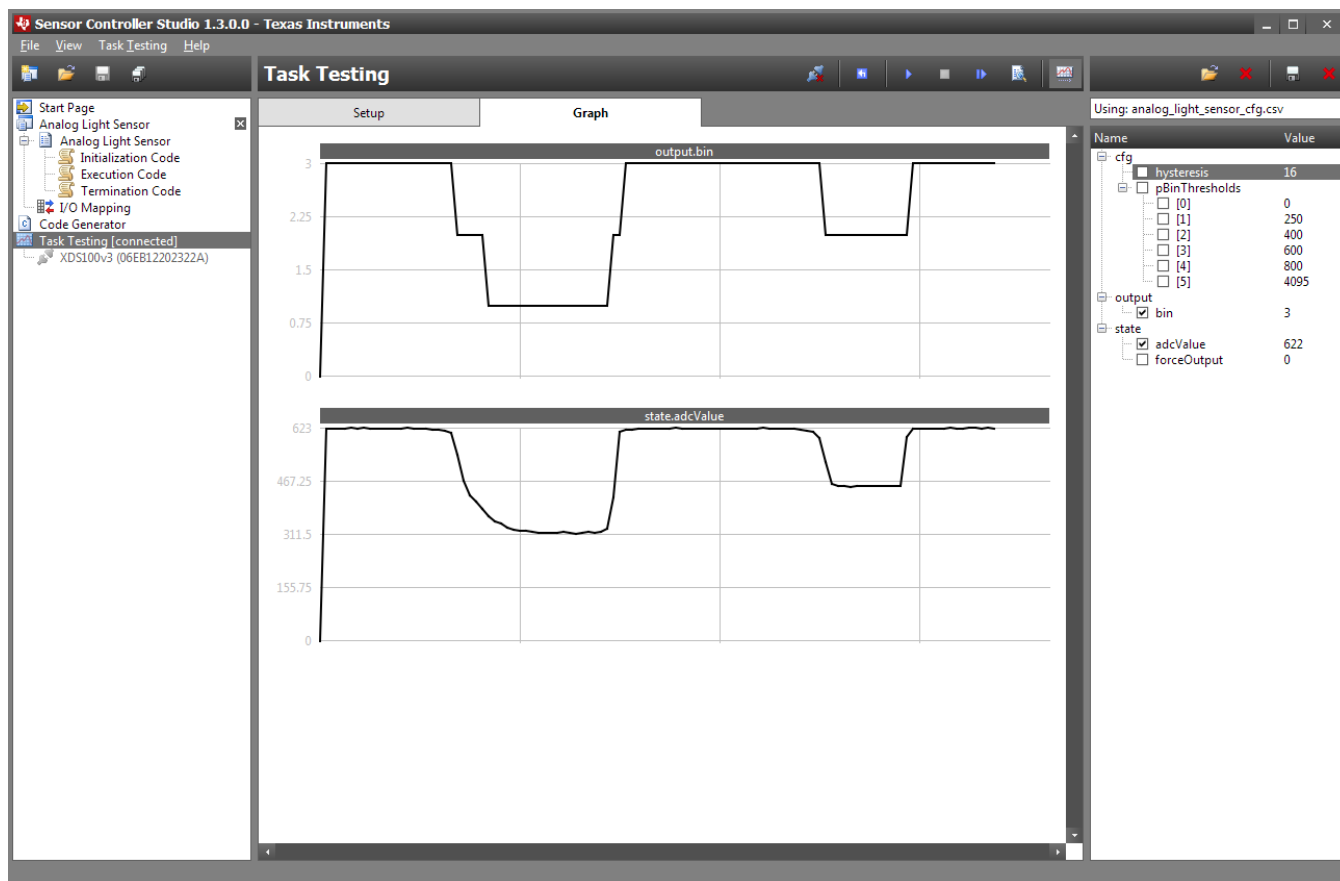


Figure 10. Task Testing Data Handling

5.10.4 Task Debugging

The Initialization, Execution, Event Handler and Termination Code can be run in debug mode:

- Manually: by triggering the Task Testing menu, tool-bar or shortcut key commands
- Automatically: when an iteration takes more than 1 second to complete, in which case the debug session begins where the task code is stuck

From the Task Testing panel, trigger the **Debug One Task Iteration**. This enables the debug mode and switches to the Task Debugging panel (see Figure 11).

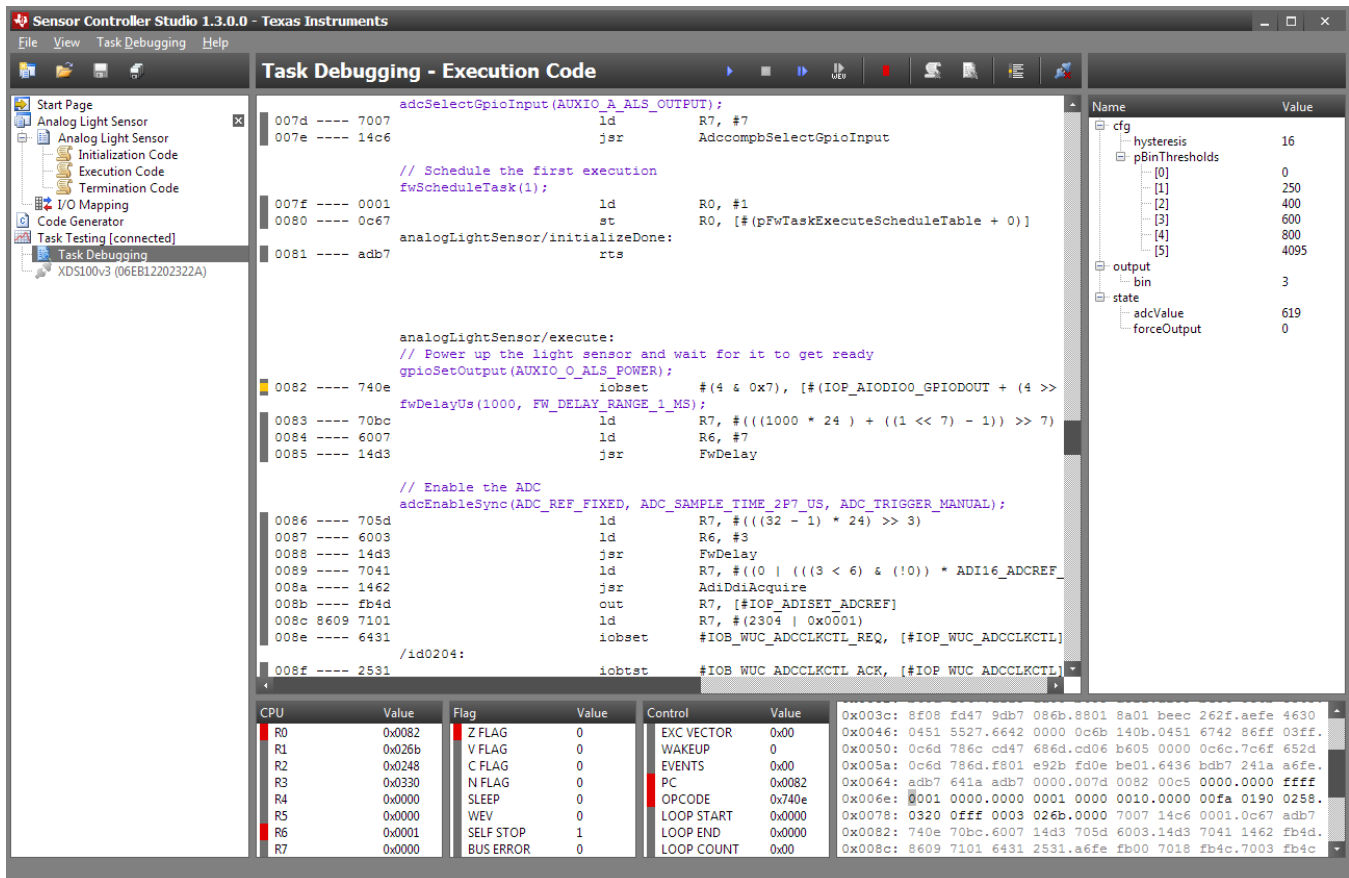


Figure 11. Task Debugging Panel

The debug session starts at the first instruction of the Execution Code. Single-step a few times and observe registers and states changing, and the current instruction indicator moving.

- Hold down **F11** to **Single-Step** until stuck on a wait for event (WEV) instruction. This happens when the WEV instruction waits for a pulsed event signal, since the debugging halts the Sensor Controller but not the hardware peripherals. Trigger **Single-Step From Blocking WEV** to end this condition.

- Setup a breakpoint in the task code by selecting the line and pressing **F9**, as shown in [Figure 12](#), and **Run** to it.

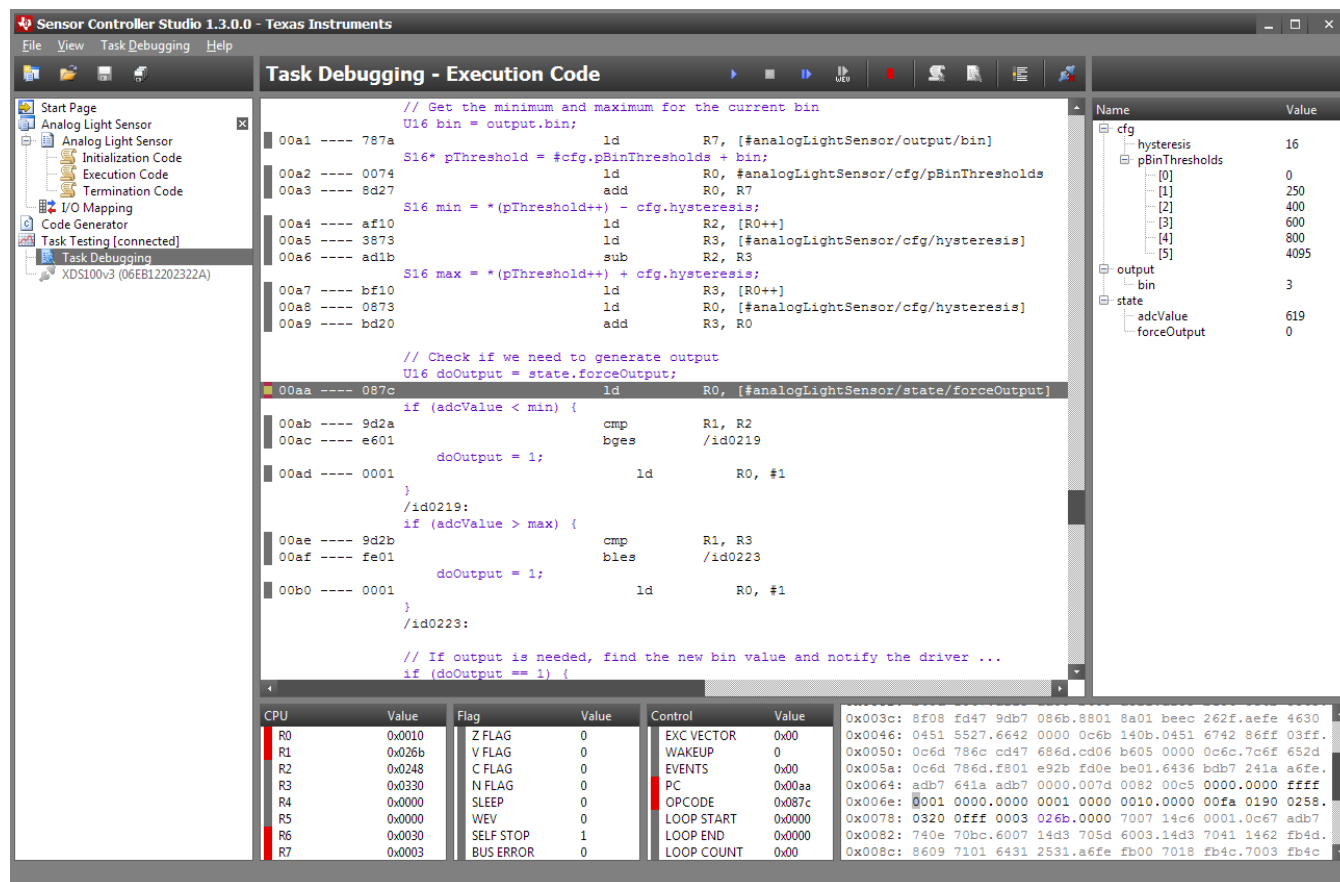


Figure 12. Task Debugging

- **Run** again. The Sensor Controller is now at the debug session exit point. **End Debugging Session** to return to the Task Testing panel.
- End the Task Testing session by triggering **Disconnect**.

6 References

CC13xx, CC26xx SimpleLink™ Wireless Technical Reference Manual (SWCU117)

Revision History

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

Changes from A Revision (March 2016) to B Revision		Page
• Added new Section 2.1 .		2
• Added new Section 3.1 .		6

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as "components") are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI's terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers' products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers' products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have **not** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products

Audio	www.ti.com/audio
Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DLP® Products	www.dlp.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
OMAP Applications Processors	www.ti.com/omap
Wireless Connectivity	www.ti.com/wirelessconnectivity

Applications

Automotive and Transportation	www.ti.com/automotive
Communications and Telecom	www.ti.com/communications
Computers and Peripherals	www.ti.com/computers
Consumer Electronics	www.ti.com/consumer-apps
Energy and Lighting	www.ti.com/energy
Industrial	www.ti.com/industrial
Medical	www.ti.com/medical
Security	www.ti.com/security
Space, Avionics and Defense	www.ti.com/space-avionics-defense
Video and Imaging	www.ti.com/video

TI E2E Community

e2e.ti.com