Application Note

Introducing the xSPI, OSPI, and QSPI Flash Differences

Rio Chan, FAE
Nilabh Anand, Application Manager

Abstract

This document teaches users the basic arch of and how to tune xSPI, OSPI, and QSPI serial flash memory interfaces for TI embedded processors and microcontrollers (MCUs).

Table of Contents

1 Introduction....2
2 Purpose....2
3 Knowing the SPI, GSPI, OSPI, xSPI, and eSPI for the TI SoC....2
3.1 Details of the SPI, GSPI, OSPI, xSPI, and eSPI....3
4 TI EP Device General Boot Modes....4
5 More Debug Tips....8
6 Summary....9
7 Conclusion....9
8 References....10

Trademarks

Jacinto™ and E2E™ are trademarks of Texas Instruments.
Linux® is a registered trademark of Linus Torvalds.
All trademarks are the property of their respective owners.

Copyright © 2026 Texas Instruments Incorporated

1 Introduction

This application note provides comprehensive technical guidance for implementing xSPI, OSPI, and QSPI flash memory interfaces on most TI EP products (high-end MCU and processors). Key topics include interface architecture, hardware configuration, driver implementation, pinmux setup, debugging methodologies, and flash memory porting procedures.

2 Purpose

The objectives of this document are:

- Define and differentiate between SPI, QSPI, OSPI, and xSPI protocols
- Explain hardware and software requirements for each protocol
- Describe TI MCU boot modes and flash configuration
- Provide practical implementation guidance and troubleshooting tips

3 Knowing the SPI, GSPI, OSPI, xSPI, and eSPI for the TI SoC

Serial flash interfaces vary significantly in data line count, signaling protocol, and performance characteristics. Table 3-1 shows an interface comparison regarding the signal clock speed versus throughput.

Table 3-1. Interface Comparison

Interface	Data Lines	Maximum Speed	Throughput	Use Case
SPI	1	50MHz	6.25MB/s	Legacy systems
QSPI	4	104MHz	52MB/s	Embedded systems
OSPI and xSPI	8	200MHz+	200MB/s	High-performance systems
eSPI	1 + Alert	100MHz	Async signal	Modern BIOS

These are the basic formulae for clock speed to throughput calculations:

1. SPI (50MHz)
   - Data lines: One
   - Clock edge: single data rate (SDR)
   - Formula: Throughput = (50MHz × 1 bit) / 8 bits/byte
   - Throughput: 6.25MB/s
2. QSPI (104MHz)
   - Data lines: Four
   - Clock edge: SDR
   - Formula: Throughput = (104MHz × four bits) / Eight bits/byte
   - Throughput: 52MB/s
   - Highlight: QSPI: Quad SPI with fixed 4-line operation, mature standard
3. OSPI and xSPI (200MHz+)
   - Data lines: Eight
   - Clock edge: double data rate (DDR / octal mode)
     - This samples data on both rising and falling clock edges, and effectively yields 2 bits per data line per cycle
   - Formula: Throughput = (200MHz × eight bits × 2 DDR) / Eight bits/byte
   - Throughput: 400MB/s
   - Highlight: OSPI: octal SPI with 8-line operation and higher throughput
   - Highlight: xSPI: flexible and supports multiple I/O modes (SDR and DDR)

Let us learn more about the terminology before doing a deep dive into the mechanism.

There are two roles in these kinds of interfaces: controller and peripheral, which define the control directions for each signal pin.

- Controller role: Used to send the chip select signal (CS) and clock source (CLK) to the controlled Flash (as peripheral)

Copyright © 2026 Texas Instruments Incorporated

- Peripheral role: Used by the Flash end, which is controlled by the CPU (SoC)

These are the common signaling pins:

- CS (or SSn or nSS): Chip select of the set number n
- CLK (or SCLK): Clock source
- IOn (as the IO0/IO1 through IO8): The input/output (I/O) data pins between the controller/CPU and peripheral/Flash
- DQ: Data bus
- DS (or DQS, WS, or RWDS): Data strobe pin. The read-write data strobe (RWDS) is used by the HyperRAM memory to indicate if an additional initial latency has to be inserted or not
- ALERT: This term is only applied in the *eSPI* protocol. The direction of Alert is from peripheral to controller, and is asynchronous in nature. Alert is used by eSPI peripherals to request service from eSPI controller. Alert pin is optional for single controller-single peripheral configuration
- Reset: Reset signal pin

3.1 Details of the SPI, GSPI, OSPI, xSPI, and eSPI

Place those five interfaces (SPI, GSPI, OSPI, xSPI, and ESPI) together to highlight the differences between each interface. In general cases, the OSPI is equal to the xSPI.

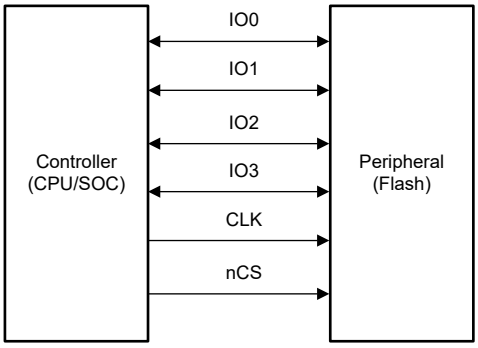

Figure 3-1. SPI Arch

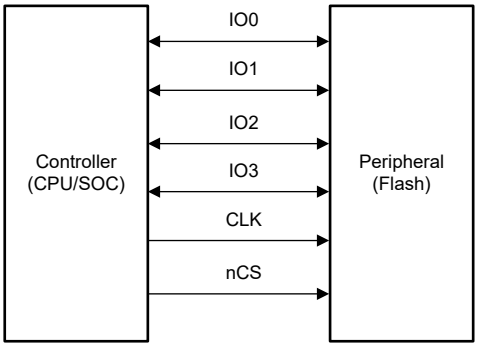

Figure 3-2. QSPI Arch

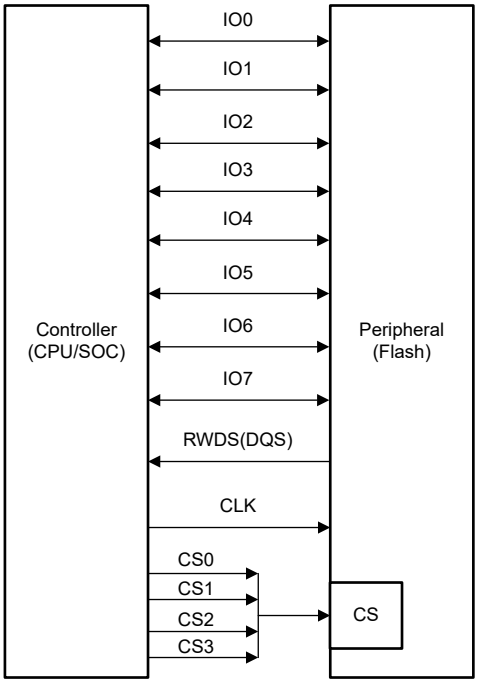

Figure 3-3. OSPI Arch

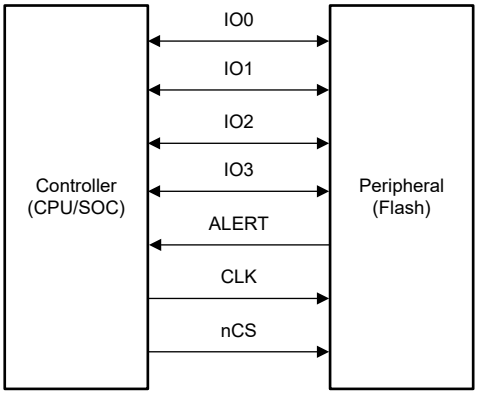

Figure 3-4. eSPI Arch

Copyright © 2026 Texas Instruments Incorporated

Generally, we can have this kind of arch:

- SPI = 1spi
- QSPI = Four sets of spi, which needs four sets of data I/O pins
- OSPI = xSPI = Eight sets of spi, which needs eight sets of data I/O pins
- eSPI = Enhanced SPI, which needs another *Alert* pin to notify CPU

4 TI EP Device General Boot Modes

There are two major TI EP products with the OSPI, QSPI, and xSPI boot mode:

- High-performance MCU: AM26xx family
  - The AM263x has QSPI IP, AM263Px (-SIP), AM261x has OSPI IP
- Processor: AM6xx family, for example TDA4VM

Table 4-1 shows the AM263PX-SIP boot modes. This supports OSPI and QSPI for 1S, 4S, and 8S mode.

Table 4-1. BOOTMODE Pin Mapping

Boot Mode	SPI0_D0_pad (SOP3)	SPI0_CLK_pad (SOP2)	QSPI_D1 (SOP1)	QSPI_D0 (SOP0)
OSPI (4S) - Quad Read Mode	0	0	0	-
UART	0	0	0	1
OSPI (1S) - Single Read Mode	0	0	1	0
OSPI (8S) - Octal Read Mode	0	0	1	1
DevBoot	1	0	1	1
xSPI 8D (SFDP)	1	1	0	0
Unsupported Boot Mode	All other combinations not defined above			

Table 4-2 shows the TDA4VM boot modes. We can see the OSPI and QSPI flash for the booting.

Table 4-2. Jacinto™ 7 Device Boot Modes

Boot Mode / Peripheral	Boot Media / Host	Used as Backup Boot Option	MCU Peripheral	Notes
No Boot	No media or host	N	N/A	No boot performed – debug mode
OSPI	OSPI flash	N	Y	On OSPI port
QSPI	QSPI flash / EEPROM	N	Y	On OSPI port
Hyperflash	Hyperflash NOR flash	N	Y	
SPI	SPI EEPROM	Y	Y	On OSPI port
I2C	I^2C EEPROM External host	Y	Y	I2C module is controller I2C module is peripheral
MMC/SD	MMC/SD card	Y	N	Boot from UDA or file system
eMMC	eMMC flash	N	N	Boot from BP; auto-fallback to file system
Ethernet	External host	Y	Y	BOOTP mode. RGMII or RMII PHY
PCIe	External host	N	N	
UART	External host	Y	Y	XMODEM protocol
GPMC NOR	NOR flash	N	N	
USB	External host	Y	N	USB device mode boot using DFU
UFS	UFS mass storage	N	N	

Copyright © 2026 Texas Instruments Incorporated

Typically, when the Flash is starting up, the initial driver configuration is set of 1s1s1s low speed for boot loader booting. This is to make sure the following boot stage can be finished. Then, when entering the RTOS or Linux® uboot or kernal, this changes to higher speed (4S or 8S speed) to maximize the code transaction for executing.

Three different kinds of flashes are used as examples to explain differences: GD25Q32E, MT25QU512ABB, and S28HS512TGAB.

- CYPRESS SEMPER FLASH (OCTAL)
  - Interface: OCTAL SPI (8-bit data)
  - Capacity: 512Mbit
  - Type: NOR Flash with octal interface
- GIGADEVICE QSPI NOR FLASH
  - Interface: QSPI (4-bit data) or standard SPI
  - Capacity: 512Mbit (see datasheet for variants)
  - Type: NOR Flash
- MICRON MT25QU512ABB8E12
  - Interface: Quad SPI (QSPI) capable
  - Capacity: 512Mbit (MT25QU = 512 in product code)
  - Type: NOR Flash

Interface differences:

- Cypress: OSPI (8-bit parallel data path)
- GigaDevice: QSPI (4-bit parallel data path)
- Micron: QSPI (4-bit parallel data path)

So, Cypress has twice the data bandwidth compared to GigaDevice.

If the user wants to adapt those different types of flashes onto TI EP platform, let us see how is going on the MCU processor. In general, the concepts on the MCU and processor are very similar.

Here are the AM263PX steps:

1. Prepare the flash configuration file using the steps mentioned.
2. Test basic flash read writes are working with the configuration mentioned in Step 1.
3. Take SDK example reference for Secondary Bootloader and Flasher Program and adapt it to your custom flash using the flash configuration generated in Step 1.
4. Power off, switch the Boot mode to *DEV* or *UART* if the flash programmer was JTAG, Uniflash, or UART Uniflash example.
5. Launch Uniflash, then program the SBL into external flash using the Flash programmer both developed in Step 2. Once flashing succeeds:
   a. Power off, switch the Boot mode to *OSPI*.
   b. Power on OSPI to boot the system.

AM263PX-SIP has flash configuration fixed so customer should use the steps mentioned in this link.

Here are the TDA4VM steps (Uniflash with UART):

1. Power off, switch the Boot mode to *UART*.
2. Using UART boot.
3. Launching Uniflash.
4. Power on, flashing the OSPI/QSPI flash.
5. Power off, switch the Boot mode to *OSPI*.
6. Power on OSPI to boot the system.

Figure 4-1 is recommended as the implementing step.

Copyright © 2026 Texas Instruments Incorporated

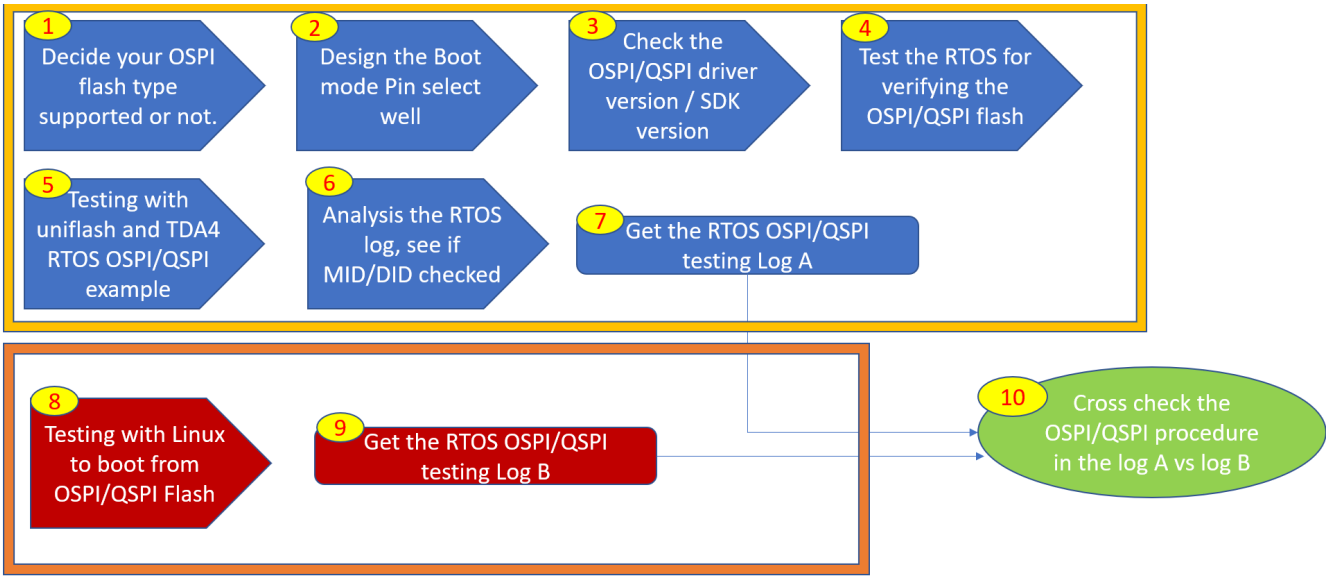

Figure 4-1. OSPI/QSPI Debug Flow

The following is a break down of the ten steps:

1. Confirm if the OSPI flash type supported or not.
   - Goal: Verify that your chosen flash memory IC physically and logically communicates with the TI processor (for example, TDA4, AM64x, AM263x).
   - Details: Check the flash datasheet against the TI SoC's technical reference manual (TRM). Verify whether the flash uses QSPI (4 data lines) or OSPI (8 data lines), if it uses STR versus DTR protocols, and whether it requires an 1.8V or 3.3V voltage rail. Confirm if the flash part is already natively included in the TI MCU+ SDK / Processor SDK board support library.
2. Design the Boot mode pin select well.
   - Goal: Wire the physical hardware configuration pins so the internal ROM bootloader targets the correct flash memory interface on power-up.
   - Details: On TI SoCs, specific GPIO pins function as BOOTMODE pins during power-on-reset (POR). User must design proper pullup and pulldown resistor networks to match the exact binary value needed for OSPI or QSPI booting. If designed poorly, the chip tries to boot from the wrong peripheral (like UART or SD card) and hang.
3. Check the OSPI/QSPI driver version and SDK version.
   - Goal: Make sure your software development kit natively supports the peripheral features you plan to utilize.
   - Details: Different versions of the TI SDK introduce critical bug fixes, updated driver APIs, and expanded flash compatibility tables. Check the SDK Release Notes to verify if your exact flash device requires custom driver patches or if it works out-of-the-box with the standard OSPI/QSPI low-level drivers.
4. Test the RTOS for verifying the OSPI/QSPI flash.
   - Goal: Set up a clean, minimal RTOS baseline environment to test basic read/ write hooks before introducing complex OS dependencies
   - Details: Use TI SysConfig to configure the clock tree, baud rates, pin-multiplexing and dummy cycles for your specific flash model. This step makes sure that the core OSPI peripheral controller registers are being initialized correctly by the software framework
5. Testing with Uniflash and TDA4 RTOS OSPI/QSPI example.
   - Goal: Execute physical hardware operations on the real target board
   - Details: Use the TI UniFlash tool over a JTAG or UART link to write a pre-compiled RTOS diagnostic application directly into the flash memory. Then, run the SDK's bundled OSPI/QSPI diagnostic test examples (like block erase, random sector writes, and data read back verification loops) to check basic signal stability.
6. Analyze the RTOS log and see if MID/DID checked.

Copyright © 2026 Texas Instruments Incorporated

- Goal: Verify that the microcontroller can reliably query the flash identity registers
- Details: Read the console output logs from Step 5. You are specifically looking for the successful detection of the Manufacturer ID (MID) and Device ID (DID). If the logs report 0x00000000, 0xFFFFFFFF, or wrong hex values, it points to a physical hardware assembly error, broken trace lines, missing power rails, or wrong clock/dummy cycle configurations.

7. Get the RTOS OSPI/QSPI testing Log A.
   - Goal: Capture a clean, definitive baseline performance and configuration reference
   - Details: Once step 6 is successful, run a full RTOS flash driver cycle (including standard speed read, high-speed read, and PHY tuning training if applicable). Export this console text output as Log A. This log represents how the flash responds when operating in a tightly controlled, standalone bare-metal or RTOS environment.
8. Testing with Linux to boot from OSPI/QSPI Flash.
   - Goal: Shift testing to the complex, full OS runtime pipeline
   - Details: Build and flash the Linux bootloader binaries (such as tiboot3.bin, tispl.bin, and u-boot.img). Attempt to boot the device cleanly into Linux using the flash interface. This step introduces a secondary, independent layer of drivers (U-Boot MTD driver and the Linux kernel SPI-NOR architecture) over the physical hardware
9. Get the RTOS OSPI/QSPI testing Log B.
   - Goal: Capture the flash transaction footprints during the multi-stage Linux boot sequence
   - Details: Collect the initialization printouts and memory mapping outputs from the U-Boot interface and Linux kernel log buffers (dmesg). Save this data sequence as Log B
10. Cross check the OSPI/QSPI procedure in Log A versus Log B.
    - Goal: Diagnose conflicts, register corruption, or resource mismanagement between software environments
    - Details: Compare Log A against Log B. Look for mismatches in clock frequencies, differences in configuration registers, or missing dummy cycle overrides. A common bug discovered here is that the flash works flawlessly under RTOS (Log A), but fails under Linux (Log B) because U-Boot or the Linux device tree configuration uses an incorrect node mapping or resource allocation, overriding working settings

Regarding the PHY training to those flashes, here is further information on why dummy cycles are needed:

- Flash needs internal time to fetch data from array and prepare output
- At high speeds, (100MHz+), this requires more clock cycles
- Dummy cycles provide PHY timing margin for clock and data phase alignment
- Without them = data corruption; with proper setting = reliable, high-speed reads

Three devices - dummy cycles that must be set:

1. CYPRESS Semper (Octal): 23 cycles at 200MHz
   - Uses data strobe (DS) for automatic PHY training
   - Built-in latency – no configuration needed
   - Highest performance (approximately 400MBps
2. GigaDevice (QSPI): 8 cycles at 100MHz
   - Configure using status register (DC bits [15:12])
   - Non-volatile setting (write once, persists)
   - Simple register-based configuration
3. Micron (QSPI): 10 cycles at 133MHz
   - Dual configuration: non-volatile and volatile registers
   - Both bits [15:12] and [7:4] = 1010
   - Most flexible, supports XIP mode

Users must understand:

- If undersetting dummy cycles, the flash data will be corruption
- If oversetting dummy cycles, the performance has the penalty
- If optimal setting, the device-specific balance is based on frequency and mode

Copyright © 2026 Texas Instruments Incorporated

5 More Debug Tips

There are three more important debug tips shown below:

1. DTR versus STR

   Users must check if you are using a double transfer rate (DTR) or a single transfer rate (STR). In most cases, a DTR is used. Users can find information on the STR and DTR from those three different Nor Flash datasheets as below.
2. Check if OSPI/QSPI Phy training is needed:

   In the RTOS code, the *sbl_ospi.c* file says:

   OSPI clock is set to 200MHz by RBL on J7200, J721S2, J784S4, and AM64x platforms.

   PHY mode cannot be used until sysfw is loaded and the OSPI clock is configured to 133MHz.

   Confirm if the OSPI/QSPI Phy training needs to be done before accessing the flash.
3. Dummy cycle versus read latency

   If the user looks in the SPI flash datasheet and the QSPI/OSPI code, there is a keyword (*dummy cycle*) that must match the read latency, so study the correspondent matching in between.

Copyright © 2026 Texas Instruments Incorporated

6 Summary

GD25Q32E:

- Bus modes: SDR is supported; DDR/DTR is not supported.
- PHY training: not required
  - Timing margins at the maximum 133MHz SDR frequency are wide enough that static board routing delay configuration is sufficient.
- Dummy cycles: default setting is eight cycles for standard fast reads. Cross check status register 3 (SR3) settings to verify the host controller matches the select command opcode.

MT25QU512ABB:

- Bus modes: STR is supported up to 166MHz; DTR is supported up to 90MHz
- PHY training: not required.

  Like the GigaDevice part, this part operates with comfortable timing margins in STR and SDR modes where dynamic training is unnecessary.
- Dummy cycles: defaults to 10 cycles for high-speed STR operations out of the box (configurable using volatile and nonvolatile configuration registers

S28HS512TGAB:

- Bus modes: SDR is supported up to 200MHz; DDR is supported up to 200MHz (yielding a 400MB/s throughput over the 8-bit bus)
- PHY TRAINING: mandatory

  Operating at 200MHz DDR results in an extremely narrow data valid window (data eye). Host-side PHY training is required to dynamically align the read web data strobe (RWDS) and maintain signal integrity.
- Dummy cycles: requires 16 cycles when operating in SDR mode at 200MHz (or DDR mode at 133MHz)

7 Conclusion

This application note has provided a comprehensive overview of serial flash interfaces supported by TI MCUs. The choice between SPI, QSPI, OSPI, and xSPI depends on system performance requirements, power constraints, and PCB design complexity.

For production systems requiring high performance and reliability, OSPI with proper PHY training and configuration represents the optimal solution. Careful attention to dummy cycle configuration and signal integrity provides trouble-free operation across the full operating temperature range.

For additional support and technical questions, contact TI field application engineers or visit the TI E2E™ Forum.

Copyright © 2026 Texas Instruments Incorporated

8 References

- Texas Instruments, *AM263P4/AM2612 Custom Flash integration FAQ*, E2E support forum
- Texas Instruments, *AM261x OSPI, QSPI Flash Selection Guide*, user's guide
- Texas Instruments, *SDK reference example for SBL*, webpage.
- Texas Instruments, *SDK reference example for flasher: JTAG and UART*, webpage.
- Texas Instruments, *AM26x MCU Boot Flow*, webpage.
- Texas Instruments, *AM26x Academy*, resource academy.

Copyright © 2026 Texas Instruments Incorporated

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, regulatory 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 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, TI’s General Quality Guidelines, or other applicable terms available either on 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. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Copyright © 2026, Texas Instruments Incorporated

Last updated 10/2025