Application Note

QSPI Usage on External Flash

Jojo Arthur

ABSTRACT

The MSPM33C321A microcontroller (MCU) integrates a Quad Serial Peripheral Interface (QSPI) controller that connects to external serial NOR flash memory over four data lines, providing sustained data transfer of up to 20MB/s. This application note describes how to connect and configure external QSPI flash on the MSPM33C321A, using the LP-MSPM33C321A LaunchPad™ development kit and its onboard 128Mbit MX25L12833F NOR flash as the reference platform. Measured results are presented for a data-streaming use case in which the MCU reads bulk data from external flash and retransmits it over one or two SPI ports concurrently. The document is intended for embedded system designers who need external memory for asset or data storage in end equipment such as optical communication modules, display-based user interfaces, and audio playback systems.

Table of Contents

Trademarks

1 Introduction

Many embedded applications need more non-volatile storage than is practical to integrate on-chip. Graphical assets for a display, calibration and configuration records for an optical transceiver, or compressed audio clips for a voice prompt system can each consume several megabytes, while the application code itself fits comfortably in on-chip flash. External serial NOR flash devices provide this bulk storage at low cost, in small packages, and with a simple six-signal interface.

The MSPM33C321A is a mixed-signal MCU based on the 160MHz Arm® Cortex®-M33 core with up to 1MB of on-chip flash and 256kB of SRAM, both with ECC. Among its communication peripherals is a dedicated QSPI controller that interfaces to external serial flash memory over four data lines and supports data transfer of up to 20MB/s. Combined with two DMA controllers (16 channels total), the QSPI enables the device to move large blocks of data from external flash to on-chip SRAM or directly on to other interfaces with minimal CPU involvement.

This application note covers three topics: the QSPI controller feature set and its hardware connection to external NOR flash, a measured performance example that streams data read from external flash out of one or two SPI ports, and practical design considerations for using this data path in optical modules, display applications, and audio storage.

2 Detailed Description

2.1 QSPI Controller Overview

The QSPI controller on the MSPM33C321A is designed specifically for external serial flash memory and provides the following key features:

- Single, dual (bi), and quad data operations with serial clock rates up to 40MHz (VDD ≥ 2.7V; up to 20MHz at VDD ≥ 1.71V), for data transfer up to 20MB/s in quad mode
- Programmable interface supporting QSPI Mode-0 and Mode-3 clock polarity/phase
- Support for both 3-byte and 4-byte address external QSPI flash memories
- Preconfigured read frame formats with automatic dummy clock insertion for optimized read operations
- Configurable automatic prefetch function for continuous data reads without CPU intervention
- Separate transmit and receive FIFOs (up to 4 locations deep) with DMA support
- Four chip-select lines (QSPI_CS0 through QSPI_CS3) for multiple external memory devices

The frame format is the central configuration choice for the QSPI. The QSPICTL0.QSPIFORMAT field selects how the instruction, address, and data phases of a transaction are distributed across the IO lines, using a three-letter notation in which each letter describes one phase as Single, Dual, or Quad lane. Configuration and status frames (command writes, status reads) use the SPI or QPI half-duplex formats, while data-read frames support the six formats listed in Table 1. Note that QQQ mode is the same as QPI mode, and that only the clock combinations SPO=0/SPH=0 (Mode-0) and SPO=1/SPH=1 (Mode-3) are supported.

Table 2-1. QSPI Data-Read Frame Formats (QSPICTL0.QSPIFORMAT)

Format	Code	Instruction	Address	Data	Typical Use
SSS	8h	Single	Single	Single	Normal / fast read
SSD	9h	Single	Single	Dual	Fast read dual output
SDD	Ah	Single	Dual	Dual	Fast read dual IO
SSQ	Bh	Single	Single	Quad	Fast read quad output
SQQ	Ch	Single	Quad	Quad	Fast read quad IO
QQQ (QPI)	Dh	Quad	Quad	Quad	Full QPI operation

For each data-read format, the controller is programmed with the instruction opcode (QSPICMDBYTE), address width (QSPIADDRMODE), optional performance byte (QSPIPERFBYTE), and the dummy clock count (QSPIDUMMYCLK) required by the selected flash read command; the dummy clocks give the NOR flash time to retrieve the requested data. The serial clock is derived from SYSCLK through the CLKDIV divider and CLKCTL prescaler, up to the 40MHz maximum, and a delayed data sampling option (CLKCTL.DSAMPLE) or internal clock loopback (QSPITIMING.DATSMPLDY) can be used to compensate board and flash output delays at high clock rates.

The QSPI peripheral is mapped at base address 0x4003.2000 (non-secure) / 0x5003.2000 (secure), uses interrupt line 47 (QSPI0), and connects to the DMA controllers through triggers 18 (QSPI RX) and 19 (QSPI TX). For register-level details, see the QSPI chapter of the MSPM33C3x 160-MHz Microcontrollers Technical Reference Manual.

2.2 Hardware Connection to External NOR Flash

The LP-MSPM33C321A LaunchPad development kit includes a Macronix MX25L12833F external NOR flash connected to the QSPI interface and is used as the reference hardware for this document. The MX25L12833F is a 128Mbit (16MB) 3V serial NOR flash with a 256-byte page size, addressable entirely with 3-byte (24-bit) addressing, so QSPIADDRMODE remains in its 3-byte setting for the full array. The device exposes a Quad Enable (QE) bit in its status register, which must be set before any quad-lane operation. Table 2 lists the signal mapping between the NOR flash and the LaunchPad pins.

Table 2-2. External NOR Flash to QSPI Pin Mapping (LP-MSPM33C321A)

NOR Flash Pin	Pin Function	LaunchPad Pin	QSPI Signal
1	CS	PC0	QSPI_CS0
2	IO1	PA14	QSPI_IO1
3	IO2	PA13	QSPI_IO2
4	GND	—	—
5	IO0	PA12	QSPI_IO0
6	SCLK	PB16	QSPI_CLK
7	IO3	PB15	QSPI_IO3
8	VCC	—	—

Two board-level details are worth noting when working with the LaunchPad. First, the external NOR flash is powered through jumper J12, which must be populated to supply the 3.3V rail to the memory. Second, to route the QSPI signals to an external connection through header J11, five 0-ohm resistors (R77 through R81) on the back of the board must be soldered; these are left unpopulated by default to avoid noise on the QSPI signals.

Table 2-3 summarizes the QSPI frame formats supported by the MX25L12833F for program and read operations, as validated on the LaunchPad. The device supports all four read formats, including full QQQ (QPI) operation; for programming, page program is available in SSS, SQQ, and QQQ formats, while SSQ page program is not supported. The measurements and software flow in this document use QPI (QQQ) mode for both program and read.

Table 2-3. MX25L12833F Supported QSPI Frame Formats

Frame Format	Page Program	Read
SSS	✓	✓
SSQ	✗	✓
SQQ	✓	✓
QQQ (QPI)	✓	✓

For a custom board design, keep the QSPI signal traces short and length-matched, and note that the maximum QSPI clock frequency depends on the supply voltage: 40MHz requires VDD ≥ 2.7V, while operation from 1.71V limits the clock to 20MHz. The datasheet QSPI timing parameters (input setup/hold and output valid/hold times) should be checked against the selected flash device across the intended voltage and temperature range.

2.3 Software Configuration

The MSP M33 SDK provides a QSPI flash driver (QSPI_FLASH.h) layered on top of the DriverLib QSPI module (DL_QSPI). The SDK example qspi_flash_readback_dma demonstrates the complete life cycle used in this application note: driver initialization, flash identification, quad-mode enablement, erase, DMA-based page program, and DMA-based quad read with verification.

The qspi_flash_readback_DMA example is provided in the MSP M33 SDK as a No-RTOS project with GCC, IAR, and TI Clang compiler variants. The project separates the reusable QSPI flash driver from the application:

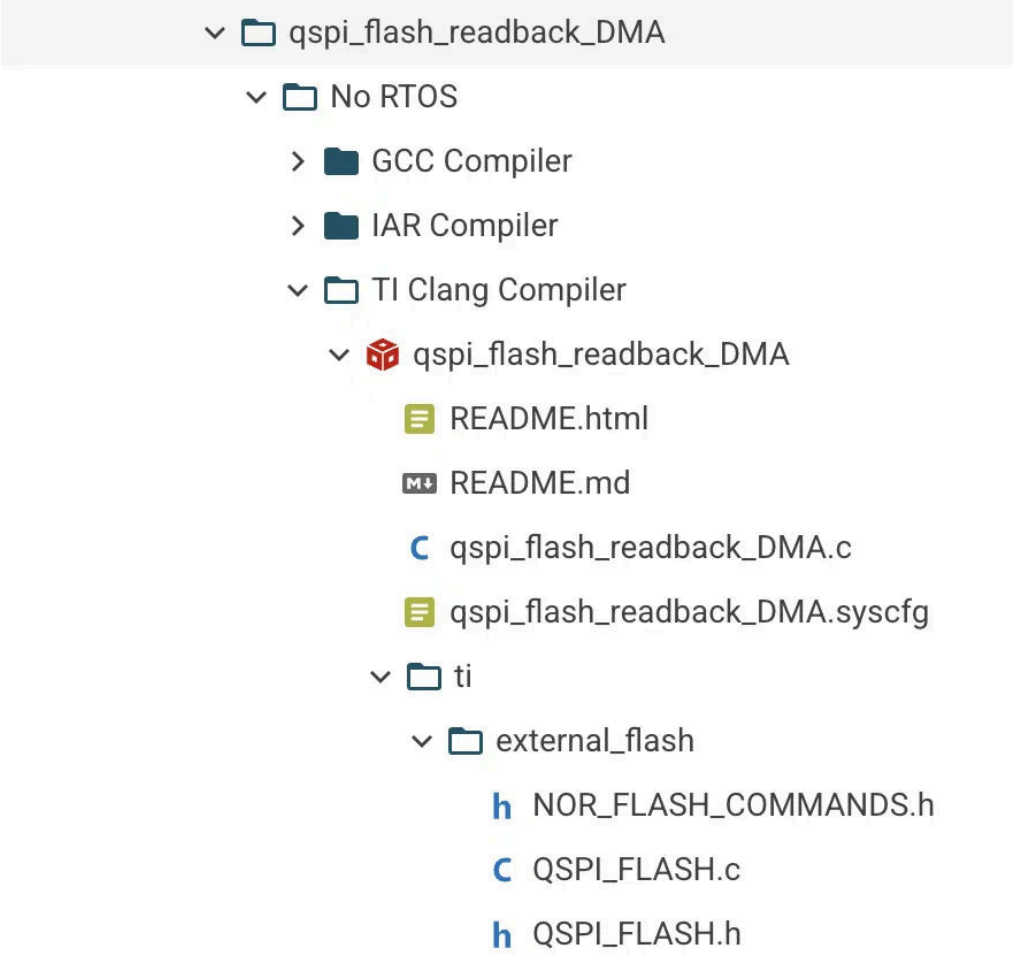

Figure 2-1. QSPI_flash_readback_DMA Project Structure

NOR_FLASH_COMMANDS.h centralizes the flash-specific opcodes (read ID, write status, enable QPI, sector erase, page program, quad I/O read) and the dummy-cycle counts, so retargeting the example to a different NOR flash typically only requires editing this header against the new flash datasheet. QSPI_FLASH.c/.h implement the QSPIFLASH driver on top of DriverLib (DL_QSPI), and the .syscfg file generates the pin multiplexing, clock tree, and DMA channel configuration through SysConfig.

The example application executes the following sequence, checking the returned status after each step and latching an error flag on the first failure:

1. Initialize the device (SYSCFG_DL_init), enable the QSPI interrupt, and bind the QSPIFLASH driver handle
2. Read the JEDEC ID in SPI half-duplex mode to confirm the flash is present
3. Set the QE bit in the flash status register, then enable the QPI mode
4. Erase the target sector in QPI half-duplex mode
5. Page program the test data with QSPIFLASH_writeDMA (automatic page-boundary handling)
6. Read the data back with QSPIFLASH_readDMA using the quad I/O read command
7. Disable the QPI mode and de-initialize the QSPI

8. Verify the readback buffer word-for-word and report pass/fail on the Launchpad LEDs

The subsections below walk through the key steps of this flow with code excerpts from the example.

2.3.1 Driver Initialization and Flash Identification

After SYSCFG_DL_init() configures the pins, clocks, and DMA channels generated by SysConfig, the application enables the QSPI interrupt in the NVIC, binds the driver handle to the QSPI instance, and reads the JEDEC ID to confirm the flash device is present. Commands that must run before quad mode is enabled use the SPI half-duplex transfer mode:

```
params.qspiRegs = QSPI;

status = QSPIFLASH_init(&handle, &params);

commandConfig.instruction = READ_ID_CMD;

commandConfig.transferMode = QSPIFLASH_TRANSFER_MODE_SPI_HALF_DUPLEX;

status = QSPIFLASH_readID(&handle, id, sizeof(id), &commandConfig);
```

The 3-byte ID returned (manufacturer ID, memory type, capacity) is the recommended first checkpoint when bringing up a new board: a correct ID confirms the pin mapping in Table 1, the J12 power jumper, and the single-bit SPI path before any quad-mode configuration is attempted.

2.3.2 Enabling Quad Mode and Programming the Flash

Quad operation requires two steps on the flash side: setting the Quad Enable (QE) bit in the flash status register with QSPIFLASH_setQEBit(), then switching the device into QPI mode with QSPIFLASH_enableQuadMode(). From that point, erase and program commands use the QPI half-duplex transfer mode. The command structure carries the instruction opcode, address size (3-byte or 4-byte per the QSPI controller's addressing support), and target address:

```
commandConfig.instruction = SECTOR_ERASE_CMD;

commandConfig.addressSize = QSPIFLASH_ADDRESS_SIZE_3BYTE;

commandConfig.address = ADDRESS;

commandConfig.transferMode = QSPIFLASH_TRANSFER_MODE_QPI_HALF_DUPLEX;

status = QSPIFLASH_eraseSector(&handle, &commandConfig);
```

Page programming uses QSPIFLASH_writeDMA(), which automatically splits a buffer across flash page boundaries. In the example, a single call programs 512 bytes (two 256-byte pages) with the driver issuing one page-program command per page and invoking the application's DMA start callback for each transfer.

2.3.3 DMA-Based Quad Read

The read path is the performance-critical direction for the streaming use case in this document. The example configures a quad I/O read with the dummy-cycle count required by the fast-read command, selects a 16-bit FIFO word size to halve the number of DMA transactions, and hands the transfer to QSPIFLASH_readDMA():

```
commandConfig.instruction = QUAD_IO_READ_CMD;

commandConfig.addressSize = QSPIFLASH_ADDRESS_SIZE_3BYTE;

commandConfig.address = ADDRESS;

commandConfig.dummyCycles = DUMMY_CYCLES_QUAD_IO_READ;

commandConfig.transferMode = QSPIFLASH_TRANSFER_MODE_QPI;

dmaConfig.startDMA = startDMARxTransfer;

dmaConfig.dmaCompleteFlag = &gDMARXDataTransferred;

DL_QSPI_setWordSize(QSPI, DL_QSPI_WORD_SIZE_16);
```

status = QSPIFLASH_readDMA(&handle, readback_data, sizeof(readback_data), &commandConfig, &dmaConfig);

The driver does not own the DMA channel; instead, the application supplies a start callback so the same code works with any channel assignment. The callback points the channel at the QSPI RX data register, sets the transfer size in 16-bit words, and enables the channel:

```
void startDMARxTransfer(const void *destAddr, uint32_t sizeBytes)
{
*(dmaConfig.dmaCompleteFlag) = false;
DL_QSPI_enableInterrupt(QSPI_0_INST, DL_QSPI_INTERRUPT_DMA_DONE_RX);
DL_DMA_setSrcAddr(DMA0, DMA0_CH0_CHAN_ID, (uint32_t)(&(QSPI->RXDATA)));
DL_DMA_setDestAddr(DMA0, DMA0_CH0_CHAN_ID, (uint32_t) destAddr);
DL_DMA_setTransferSize(DMA0, DMA0_CH0_CHAN_ID, sizeBytes / sizeof(uint16_t));
DL_DMA_enableChannel(DMA0, DMA0_CH0_CHAN_ID);
}
```

Completion is signaled by the QSPI DMA_DONE_RX interrupt (QSPI0, interrupt line 47). The handler sets the completion flag polled by the driver and disables the interrupt until the next transfer, a pattern that keeps the interrupt service routine to a few instructions. The write direction mirrors this structure on a second DMA channel targeting the QSPI TX data register.

Three practical notes from the example apply to any application built on this driver:

- Select DL_QSPI_WORD_SIZE_16 when the transfer length is even and the DMA moves 16-bit words; fall back to 8-bit word size for odd byte counts
- Re-arm the DMA_DONE interrupt inside the start callback and disable it in the handler, so a stale interrupt cannot fire between transfers
- After streaming completes, QSPIFLASH_disableQuadMode() and QSPIFLASH_deinit() return the flash to SPI mode, which keeps the device recoverable by standard tools and bootloaders

The example verifies the readback buffer word-for-word against the programmed pattern and reports pass/fail on the LaunchPad LEDs, providing a self-checking starting point before the application replaces the test pattern with real asset data.

2.4 Measured Performance: Flash Read With SPI Streaming

A common system architecture uses the MCU as a data pump: bulk data is stored in external QSPI flash and, at run time, is read out and streamed to one or more downstream consumers over SPI. To characterize this path, data blocks of 128KB to 1MB were first read from the external NOR flash and then transmitted out of the device over SPI, using one SPI port and two SPI ports transmitting concurrently. On the receive side, the transmitted bytes were simply captured and discarded with no post-sampling handshake, so the results represent the raw streaming capability of the transmit path. Table 2-4 lists the measured transfer times.

Figure 2-2 shows the data path exercised by this measurement. Data flows from the external NOR flash through the QSPI controller into an SRAM buffer under DMA control, then out through one or two SPI transmit paths, each serviced by its own DMA channel. The Cortex-M33 CPU only orchestrates the transfers; it does not copy data.

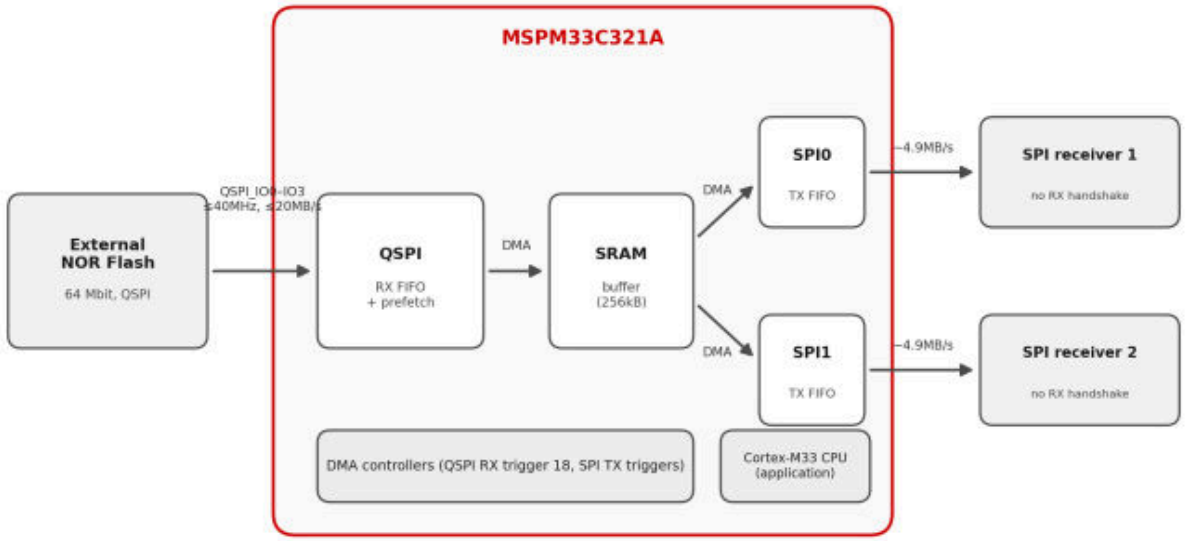

Figure 2-2. External Flash to Dual-SPI Streaming Path

Table 2-4. Measured Transfer Time, External Flash Read Plus SPI Transmit

Data Size	1 SPI Transmitting	2 SPI Transmitting
128KB	26.81ms	26.82ms
256KB	52.80ms	52.83ms
512KB	104.79ms	104.84ms
1MB	208.76ms	208.90ms

Two observations follow from these results. First, the transfer time scales linearly with the data size (doubling the payload doubles the time to within a fraction of a percent), which indicates a fully pipelined, DMA-driven data path with negligible per-block software overhead. The effective sustained throughput is approximately 4.9MB/s per SPI port, or roughly 39Mbit/s on the wire.

Second, adding a second SPI transmitter is essentially free: the 1MB transfer takes 208.76ms with one SPI port and 208.90ms with two ports streaming the same amount of data each, a difference of less than 0.1%. Because the QSPI read side (up to 20MB/s) is faster than the SPI transmit side, and because each SPI port is serviced by its own DMA channel, the two output streams run in parallel without contending for CPU time or bus bandwidth. This doubles the aggregate output bandwidth to approximately 9.8MB/s.

Note that these figures were measured without any receive-side handshake after sampling. In a real system where the receiver applies flow control (for example, a ready line or a command/response protocol), the effective throughput is set by the receiver and will be lower; the values in Table 2 represent the upper bound of the MSPM33C321A transmit path itself.

2.5 Application Use Cases

2.5.1 Optical Modules

Optical transceivers and communication modules typically store large calibration tables, per-channel tuning coefficients, and firmware images that exceed the capacity economical to keep on-chip. An external QSPI flash holds this data, and the streaming architecture measured above maps directly onto the module's internal topology: the MSPM33C321A reads coefficient blocks from flash and distributes them over independent SPI ports to multiple DACs, laser driver ICs, or cross-point devices concurrently. The near-zero cost of the second SPI stream means configuration of parallel optical lanes does not serialize, reducing module bring-up and reconfiguration time. The device's 125°C extended temperature rating and its listing for communication-module applications make it well suited to this environment.

2.5.2 Display Applications

Graphical user interfaces require image assets, fonts, and animation frames that quickly outgrow on-chip flash. Storing these assets in external QSPI flash and streaming them to the display controller is the standard pattern, demonstrated by the LaunchPad out-of-box example, which plays a video from the 64Mbit NOR flash on the onboard 128×32 OLED. As a sizing reference, a 128KB frame or asset block moves through the flash-to-SPI path in under 27ms, and sustained streaming of approximately 4.9MB/s supports continuous animation on small and mid-size SPI displays. Larger assets can be prefetched into SRAM using the QSPI automatic prefetch function while the previous frame is still being clocked out.

2.5.3 Audio Storage and Playback

Voice prompts, alert tones, and audio clips are naturally stored in external flash. The MSPM33C321A pairs the QSPI with two digital audio interfaces supporting full-duplex I2S and 16-slot TDM, so compressed or PCM audio can be read from flash and delivered to an external codec or amplifier with DMA on both sides of the transfer. The measured streaming bandwidth of approximately 4.9MB/s is orders of magnitude above the rate required for even multi-channel audio (16-bit stereo at 48kHz requires only 192kB/s), leaving ample headroom for the CPU to run the application, or for the audio path to coexist with a display path sharing the same external flash through separate QSPI chip selects.

3 Summary

The QSPI controller on the MSPM33C321A provides a simple, high-bandwidth connection to external NOR flash, with quad-mode transfers up to 20MB/s, automatic prefetch, and full DMA integration. Measured results on the LP-MSPM33C321A LaunchPad show that data read from external flash can be streamed out over SPI at approximately 4.9MB/s per port, and that a second concurrent SPI stream adds essentially no time penalty, doubling aggregate output bandwidth. This flash-to-interface data-pump architecture serves a wide range of end equipment, including coefficient distribution in optical modules, asset streaming for displays, and audio clip playback, while leaving the Cortex-M33 CPU free for application processing.

4 References

1. Texas Instruments, *MSPM33C321x Mixed-Signal Microcontrollers*, datasheet.
2. Texas Instruments, *LP-MSPM33C321A Evaluation Module*, user's guide.
3. Texas Instruments, *MSPM33 C3-Series 160MHz Microcontrollers*, technical reference manual.
4. Texas Instruments, MSP Software Development Kit (SDK), qspi_flash_readback_dma example, available in TI Resource Explorer

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.

Last updated 10/2025