Application Note

MSPM0 and MSPM33 Devices: Retaining IO State Through Bankswap

Erik Vaughn

ABSTRACT

MCUs typically drive multiple signals to other ICs on a PCB using GPIOs. The voltage level of these GPIOs can be critical to maintain for the lifecycle of the MCU. Most MCU's require a reset to perform a firmware update. This causes a problem for applications that both require a firmware update and need to maintain a certain GPIO voltage. To maintain the GPIO voltage through a firmware update the MSPM0 and MSPM33 devices provide a methodology to latch the IO state before performing the reset required for a bootprocess. To set these latches the device must enter a low power mode called shutdown but this causes an additional problem of waking up the device. This document goes over how to trigger these latches, how to automatically wake up the MCU after shutdown, how to build a software application to achieve this functionality, and use cases for the boot process.

Table of Contents

Trademarks

All trademarks are the property of their respective owners.

1 Introduction

For applications where GPIO's state needs to be maintained during the MCU's life cycle such as applications where the MCU is enabling LDO's, biasing analog signals, or turning on separate ICs, maintaining these GPIO's voltage state while the MCU is powered becomes critical. Typically this information on GPIO output voltage and IOMUX configuration are kept inside of the MCU's memory mapped registers (MMR). When the MCU resets these MMRs are reset causing these external voltage signals to be lost. For use cases where the MCU may have a firmware update while it is actively driving a GPIO to properly trigger the firmware update the MCU will have to issue a system reset. This system reset will cause the MCU to lose the MMR information causing the GPIO voltage to be lost. This causes a problem that can be resolved by adding additional hardware to the circuit which would increase the overall cost of the system. To help prevent this added cost the MSPM0 and MSPM33 products include latches inside of the IOCell structure to maintain IO state through updates.

These latches inside the IOMUX are available on every pin on the MSPM0 and MSPM33 devices. These latches inside the IOMUX maintain any internal pullup/pulldown resistors and GPIO output voltage. To trigger these latches the MCU must enter shutdown and wake up. This process forces the MCU to go through a primary or secondary bootloader as well. By adding this one step before updating the application image, the GPIO voltage can be properly maintained for as long as needed.

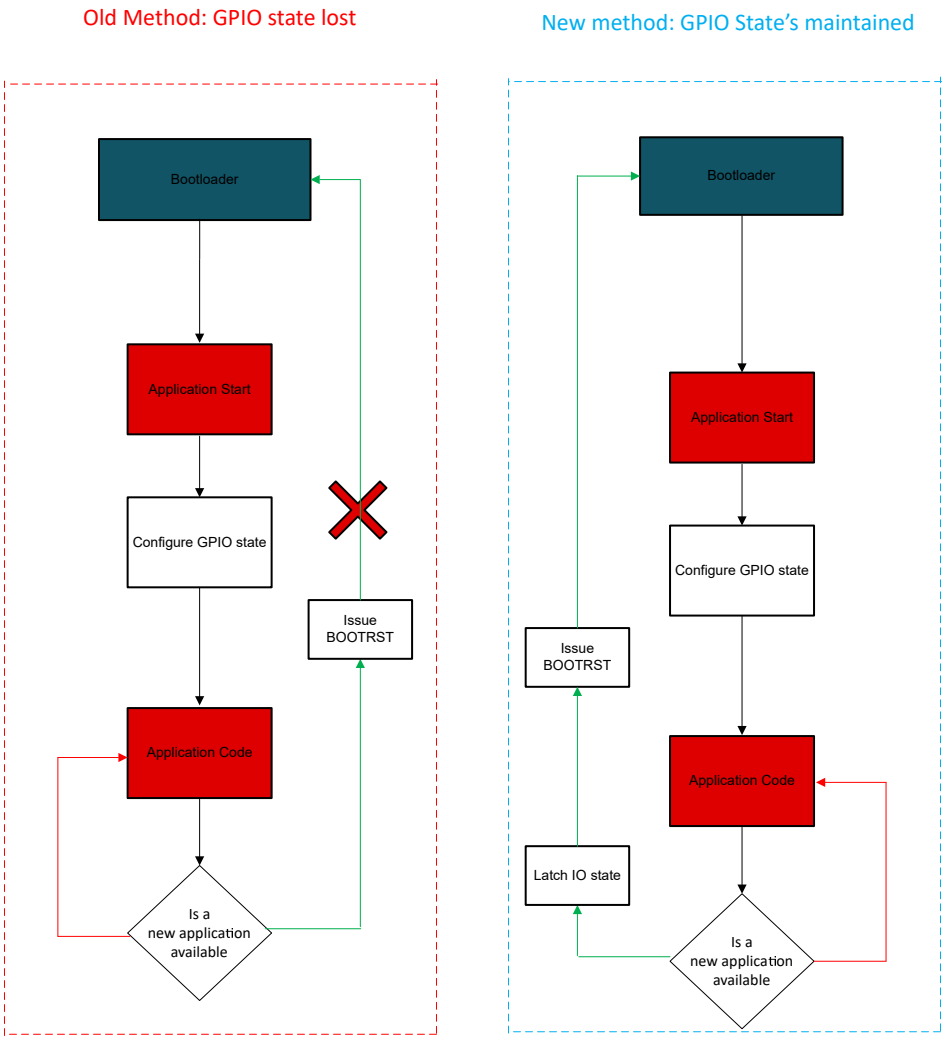

Figure 1-1. Standard Boot Process Compared vs Maintained GPIO State Boot Process

1.1 Firmware Updates on MSPM0 and MSPM33 MCU

MSPM0 and MSPM33 devices have a variety of ways to support a firmware update process. Typically this is achieved using primary (Some times called ROM bootloader or BSL) or secondary bootloaders (sometimes referred to flash bootloaders). These bootloaders utilize some communication peripheral to read an update from an external device to write a new application image to the code memory. Some of these bootloaders provide security feature such as the customer secure code (CSC) which verifies if a new application image has the proper signature to guarantee the new firmware came from a secure element. All of the bootloaders previously mentioned follow a similar flow which requires the MCU to execute a reset to properly run the bootloader either through the ROM bootloader or the secondary bootloader.

The MSPM0 and MSPM33 provide a variety of reset options as seen in Figure 1-2. The red box's show the different types of reset that could possibly be triggered by the MCU. The BCR and BSL are also highlighted in the Figure 1-2 graphic. The BSL is the bootstrap loader and the BCR is the boot configuration routine which together from the ROM bootloader. This ROM bootloader has the capability to trigger a new firmware update. We can then see any reset above the BCR and BSL boxes would trigger the firmware update we need such as a brownout reset (BOR), power on reset (POR), and boot reset (BOOTRST).

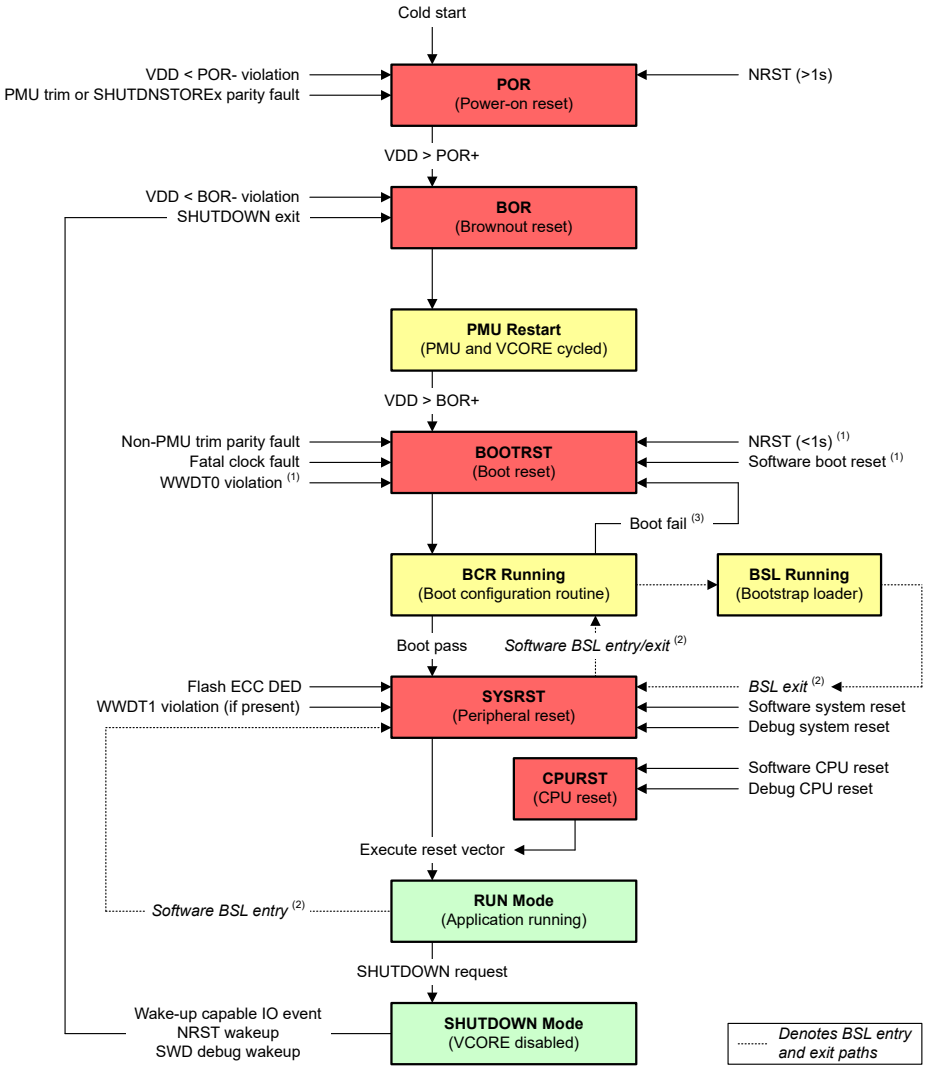

Figure 1-2. Device Reset Flow

1.2 Power Modes on the MSPM0 and MSPM33

On the MSPM0 and MSPM33 devices there are five different power modes: RUN, SLEEP, STOP, STANDBY, and SHUTDOWN. To trigger these GPIO states the device must enter a low power mode state called shutdown. What's unique about the shutdown power mode when compared to the other modes is all peripherals and the MCU are in a deep sleep mode where only configured peripherals can wake up the MCU. See Figure 1-3 as a high level reference for the different operating modes.

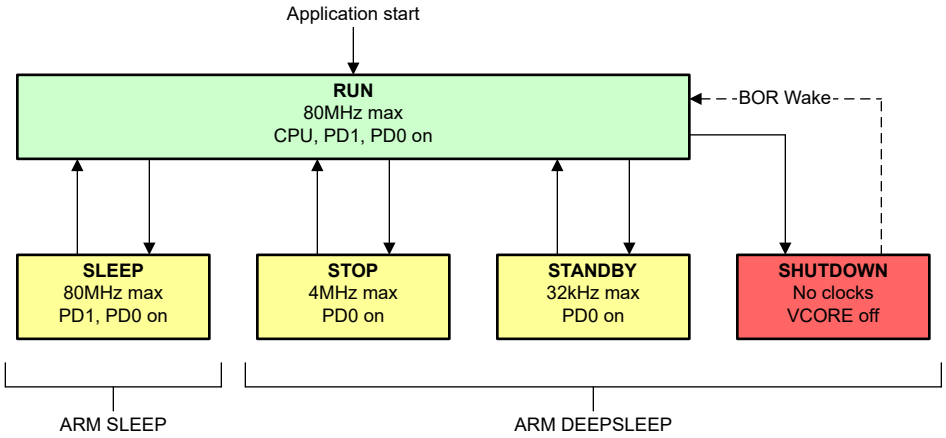

Figure 1-3. MSPM0G Operating Modes

As shown in Figure 1-3 , when exiting the shutdown mode for the device a BOR is triggered. This causes the MCU to trigger a reset and start the application code again. This is convienient if the goal is to execute a firmware update that would need to be triggered at the start of the code. However before the BOR is triggered the device must wake up from shutdown. To wake up the MCU from shutdown, use either a wake-capable IO, a debug connection, or NRST. To automate this process a wake capable IO is the easiest method rather than issuing a NRST or creating a debug connection. This document further discussess this in Section 2.

2 How the IO Latches Work

To properly understand how the latches can be used for the secure boot process, first the process for how these latches are triggered, through what states the IO state are maintained and how to automate the wake up procedure needs to be understood. This section discusses the hardware that allows the functionality for latching the GPIO voltage level, How to trigger these latches through entering shutdown, and how to automatically wake up the device after entering shutdown.

2.1 How the IO State Gets Latched

Triggering the IOMUX Latch

Inside of the IO mux there is a specific latch that keeps the GPIO's logic level. See the SHUTDOWN latches in the graphic below. These latches get set automatically once the MCU enters shutdown. After the latch is set, the GPIO state is maintained until the RELEASE bit in the SHDINIOREL register is set. Here are the steps taken for latching and unlatching the IO state.

1. When the SHUTDOWN signal asserts, the D-Q latches in the output path capture DOUT, DRV (drive strength), PIPU (pullup enable), and PIPD (pulldown enable).
2. The latches maintain state independent of the GPIO MMRs
3. The release signal (driven by writing SHDINIOREL) resets the latches and returns IO control to the normal MMR-driven path.

This state is maintained through BOOTRST, BCR, BSL, SYSRST, and CPURST, all resets lower in the hierarchy shown in Maintaining IO State During the Bootprocess, until software releases the latches through SHDINIOREL. See the PMCU chapter of the technical reference manual for more details.

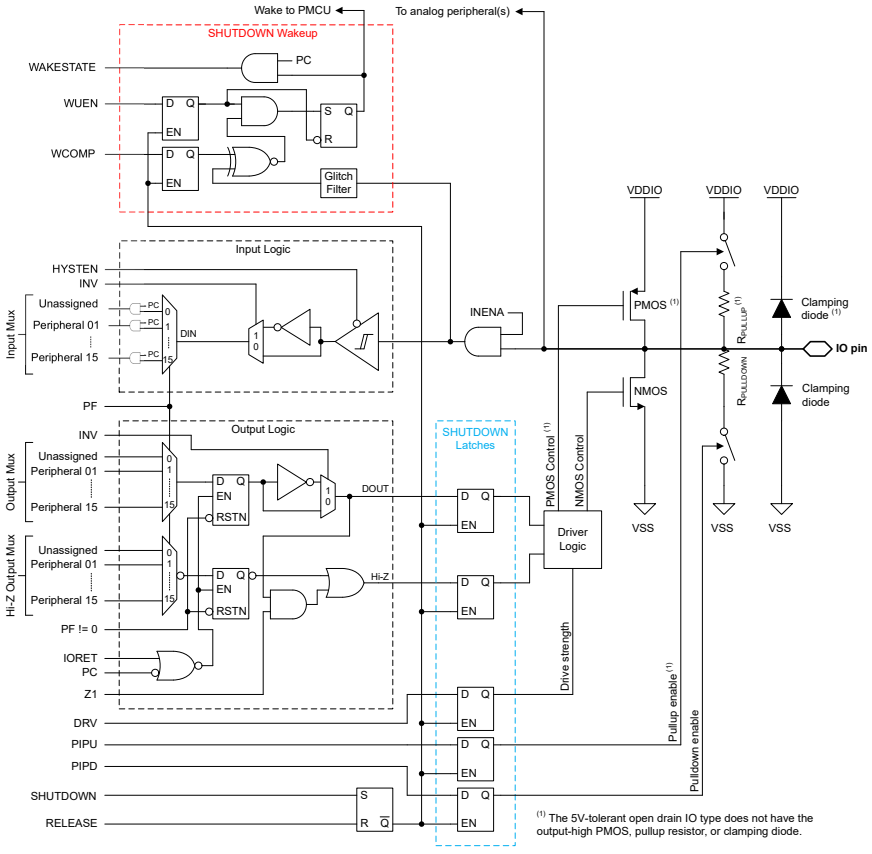

Figure 2-1. IOMUX Structure

Waking up from Shutdown

To properly latch the IO state the MCU just needs to enter shutdown. To verify the MCU can immediately wake up without any issues a separate IO with wake functionality can be configured with an internal pull up resistor. If the logic level for this wake functionality is high then the MCU immediately wakes up after the device enter shutdown.

To verify the MSPM0 or MSPM33 device has shutdown capable IO's, check the device datasheet. The following table shows generic IO types supported on the MSPM0 or MSPM33 devices and the features for each IO type. If the GPIO is an SDIO with wake, ODIO, or HDIO it supports wake up shutdown.

Table 2-1. Digital IO Features by IO Type

BUFFER TYPE	INVERSION CONTROL	DRIVE STRENGTH CONTROL	HYSTERESIS CONTROL	PULLUP RESISTOR	PULLDOWN RESISTOR	WAKEUP LOGIC
SDIO (standard drive)	Y			Y	Y	
SDIO (standard drive) with wake(1)	Y			Y	Y	Y
HDIO (High drive)	Y	Y		Y	Y	Y
HSIO (High speed)	Y	Y		Y	Y	
ODIO (5V-tolerant open drain)	Y		Y		Y	Y

2.2 Maintaining IO State During the Bootprocess

Since the IO state is lost after a boot reset due to the GPIO register information being lost, the previously mentioned internal latches for the IOMUX can be utilized for holding the IO state until the application can properly reconfigure them. To trigger these latches the MCU must enter shutdown and after the MCU wakes from the shutdown the device goes through the BCR, BSL, or any secondary bootloader configured. Through this process the MCU maintains the IO state properly until the main application can then properly reconfigure the IO state and release the IOMUX latches.

To configure an application to latch the IO state, follow these steps:

1. Configure GPIOs to the desired state.
2. Configure a singular GPIO with wake from shutdown capabilities with an internal pull up resistor and to wake when high.
3. Put the device into shutdown.
4. After wakeup, release the IO latches by writing to SHDINIOREL before reconfiguring MMRs.

Using the above steps the device wakes up immediately from shutdown and restart the boot sequence. Waking up from shutdown triggers a BOR. This forces the device to go through the boot reset process. See graphic below for what happens after a BOR.

3 Configuring Software Around Shutdown IO Functionality

Now that the mechanics behind how the IO latch has been understood we can utilize this functionality to build out a structure for how software can be generally architected to put the device into shutdown while maintaining the GPIO state. This section discusses how to build an application around the IOMUX latches so the MCU can properly go through a bootloader while maintaining the GPIO state.

3.1 Configuring software to wake from shutdown

If the goal is to reset the MCU either through a SYSRST or other types of resets while maintaining the IO state, the user can configure an internal pull up resistor on a GPIO with wake capabilities and set the GPIO to wake when high. After doing this step the user can put the device into shutdown and the MCU wakes up with no other input needed.

To configure the MCU to latch the IO state there's two major steps:

1. Configure an IO with wake from shutdown functionality with an internal pull-up resistor and to wake when a high voltage is received.
2. Put the MCU into shutdown

For the first step use the code example below for configuring the wake IO.

```
DL_GPIO_initDigitalInputFeatures(Wake_IO_PIN_0_IOMUX,
    DL_GPIO_INVERSION_DISABLE, DL_GPIO_RESISTOR_PULL_UP,
    DL_GPIO_HYSTERESIS_DISABLE, DL_GPIO_WAKEUP_ON_1);
```

For the second point you can just use the below API's

```
DL_SYSCTL_setPowerPolicySHUTDOWN();
__WFI();
```

The above lines puts the device into shutdown mode. Since a GPIO with wake capabilities has been configured to wake when high and has a pull up resistor connected the GPIO, the MCU wakes up immediately after entering shutdown.

After the IO's have been latched the user must release the IO state for the GPIO functionality to properly be updated. To do this the below API can be used that sets the RELEASE bit in the SHDINIOREL register directly.

```
/* Release shutdown IO latches - call this after MMRs have been re-initialized */
DL_SYSCTL_releaseShutdownIO();
```

3.2 Using State Retention IOs With Bootloaders

This feature is useful for MCU bootloaders where the IO state can be lost during the boot process or during a bankswap. For usecases where certain IOs are used for biasing signals or locked into a known digital state during both applications, you can use the software shown before to configure the IOs state. When developing an application that uses a bootloader, Figure 3-1can be used as a general idea for how to structure a program to latch the GPIO state. In this example, Application A detects a new firmware image is available and needs to perform a bankswap. Before entering shutdown it configures its output GPIOs to the desired hold state. On wakeup from shutdown (which triggers a BOR), the bootloader determines which application to execute. Each application starts by reconfiguring its GPIO state and releasing the shutdown IO latches via SHDINIOREL before running the main application loop.

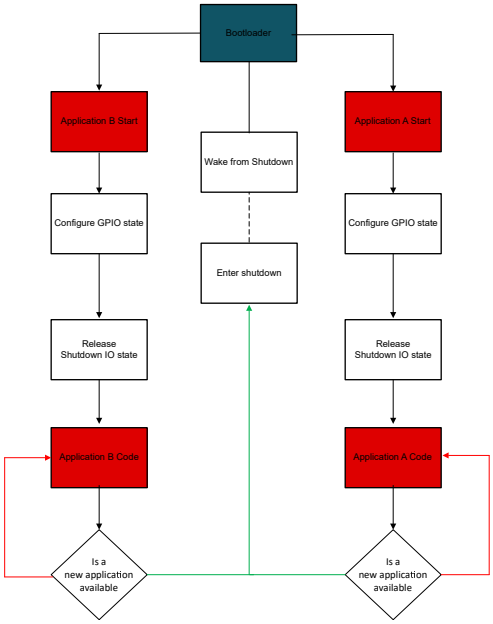

Figure 3-1. Bootflow for Utilizing IO Latch Functionality

Determining if the MCU Woke from Shutdown

If the application image uses different power modes and has multiple different ways the MCU could restart its application image. To determine if the MCU has woken up from shutdown and the RELEASE bit in the SHDINIOREL register needs to be set the RSTCAUSE register can be read. Use the below code as an example.

```
if (DL_SYSCTL_getResetCause() == DL_SYSCTL_RESET_CAUSE_SHUTDOWN) {
    /* Device woke from shutdown - IO latches are active */
    /* Re-initialize peripherals, then release latches */
    DL_SYSCTL_releaseShutdownIO();
}
```

If the MCU uses a CSC or other bootloader that issued a SYSRST before the application image has started, the RSTCAUSE register can be compared against the corresponding software triggered SYSRST ID value. See the PMCU chapter of the technical reference manual for more details.

3.3 Shutdown Memory

Additionally these MCU's allow the user to store the up to four bytes of data inside the SYSCTL peripheral. You can store this data inside of the SHUTDNSTORE0 - SHUTDNSTORE3 registers. As long as the data is stored before entering shutdown the SHUDNSTORE registers are maintained similar to how the GPIO state is maintained.

Shutdown Memory Software

For storing this shutdown storage byte use the following API.

```
/* Before shutdown: store application state */
DL_SYSCTL_setShutdownStorageByte(DL_SYSCTL_SHUTDOWN_STORAGE_BYTE_0, myStateValue);
```

After waking up from shutdown the following API can be used to read the corresponding register directly

```
/* After wakeup: retrieve stored state */
uint8_t myStateValue = DL_SYSCTL_getShutdownStorageByte(DL_SYSCTL_SHUTDOWN_STORAGE_BYTE_0);
```

Use Cases for Shutdown Memory

For use cases using bankswap or bootloaders where the MMR's and SRAM were lost between a bankswap, the user can write them directly in this memory instead. This is useful for when the device computed data during run time that have been calculated only when the previous application was running. One example of this is a counter that incremented when a device on an I2C bus communicated.

4 Summary

For applications where the GPIO voltage must be maintained during the whole lifecycle of the MCU the MSPM0 and MSPM33 MCU's can utlize the shutdown IO feature within the IO cell to latch the voltage properly. By putting the device into a low power mode called shutdown the latches can be set and the MCU can preconfigure the GPIO's as a pull up resistor to immediately wake up the MCU. By configuring your software with this in mind you can utlize the previously mentioned code to build software that put's the device into shutdown to both trigger the bootprocess and latch the IO state. With this feature shown a software project can be easily configured to not lose a IO voltage when performing a firmware update.

5 References

- Texas Instruments, *MSPM33 C3-Series 160MHz Microcontrollers Technical Reference Manual*, technical reference manual.

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