*Application Note*

# *Enhanced OSPI PHY Tuning Algorithm for Sitara™ Processors on MCU+ SDK*

*Nikhil Jain, Aryamaan Chaurasia, Vaibhav Kumar, Soumya Tripathy, Brian Elies, Nuruddin Mahmood*

**ABSTRACT**

This Application Note provides an overview of the OSPI DQS PHY Tuning algorithm, a new feature introduced in MCU+ SDK 11.02, designed to optimize high-speed serial NOR and NAND flash access on Texas Instruments' Sitara™ processor family, including AM243x, AM62x, AM62Ax, AM62Dx, AM62Px, AM64x, and AM275x devices. The PHY Tuning process automatically calibrates three timing parameters which include Tx DLL, Rx DLL, and read delay. These parameters compensate for board-level propagation delays, process, voltage, and temperature variations. The target audience includes embedded software engineers, system architects, and hardware designers working with OSPI flash memory on Sitara processors.

## Table of Contents



## Trademarks

# 1 Introduction

Modern embedded systems require fast, reliable access to external flash memory for code execution, and data storage. The OSPI enables high-bandwidth communication with serial NOR and NAND flash devices, with single data rate (SDR) and dual data rate (DDR) mode supporting high data rates in eight-lane configurations.

At these high-speeds, signal timing is critical. Small variations in board trace lengths, temperature changes, voltage fluctuations, and component tolerances can corrupt data if the OSPI controller samples data at the wrong time. The OSPI PHY (Physical Layer) module includes programmable delay lines that allows fine-grained control of signal timing, but finding the optimal delay settings manually is impractical.

To determine the delay settings, the PHY Tuning algorithm searches the parameter space to identify stable operating points. This application note explains how the algorithm works, what factors affect tuning success, and the steps performed in DQS Tuning algorithm.

# 2 Terminology

PHY:

Refers to the PHY (Physical Layer) mode of the OSPI driver. PHY mode uses a specialized timing circuit to manage memory data transfers. In this mode, each reference clock cycle generates one complete memory clock cycle for standard transfers or half a cycle for double-speed transfers. The system offers four different timing configurations, using either internal signals or external feedback from the memory chip.

When PHY is enabled, the input clock divider is bypassed. As a result, the effective frequency is the input clock frequency. The PHY tuning algorithm calculates the optimal tuning point by varying rxDLL, txDLL, and Read Delay. To learn more about this, refer here.

QSPI:

Quad Serial Peripheral Interface is an enhanced SPI variant using 4 data lines (DQ0-DQ3) for serial data transfer. Supports single/dual/quad modes for different transfer phases, achieving up to 4x bandwidth improvement over standard SPI while maintaining backward compatibility.

OSPI:

Octal Serial Peripheral Interface is an advanced SPI variant using 8 data lines (DQ0-DQ7) for serial data transfer. Supports all QSPI modes plus octal mode, enabling even higher bandwidth. It can operate with or without DQS (Data Strobe) signal for source-synchronous data capture.

SDR:

Single Data Rate mode transfers data on a single edge of the clock signal, sending one bit per clock cycle per data line. This is the simpler and more traditional clocking scheme that provides good reliability at moderate speeds. In octal SDR mode with 8 data lines, the theoretical maximum data rate is 8 bits per clock cycle.

DDR:

Double Data Rate mode transfers data on both rising and falling edges of the clock signal, effectively doubling the data throughput compared to SDR mode. In octal DDR mode with 8 data lines, data is transferred 16 bits per clock cycle (8 bits per edge × 2 edges).

Protocol (Command-Address-Data):

The protocol mode format is WR-WR-WR, where the first WR represents the command bit width and rate, the second WR represents the command modifier bit width and rate, and the third WR represents the data bit width and rate. The bit width (W) may be 1 or 8 bits. The rate (R) is either S for SDR or D for DDR. SDR transfers the same value on both rising and falling clock edges, while DDR may transfer different values on each edge.

For example, 1S-1S-1S means all phases use 1-bit wide SDR. The notation 8D-8D-8D means all phases use 8-bit wide DDR.

DQS:

Data Strobe is a source-synchronous signal provided by Flash device during Flash read operations in DDR mode. The DQS edges are aligned to the center of the valid data windows, allowing the controller to sample data at the optimal time. DQS is used in DDR mode, and not in SDR mode.

Reference Clock:

Reference clock is the input clock signal to the OSPI controller. It is typically provided by a system clock. This clock is divided down to generate the serial clock that is sent to the Flash device.

DLL:

Delay Locked Loop is a digital circuit that generates precise, programmable delays for signal timing control. The DLL consists of a chain of delay elements, typically inverters or buffers, whose propagation delays can be adjusted. The loop locks to a reference by continuously adjusting the delay chain until the output aligns with a feedback signal. In the OSPI PHY, separate Transmit and Receive DLLs control outgoing and incoming signal timing with resolution of one delay element step, providing fine-grained control over setup and hold times.

Tx DLL:

Transmit DLL is a programmable delay line in the PHY that adjusts when the controller drives outgoing data and commands to the Flash. It also ensures that the setup and hold time requirements are met. Values typically range from 0 to 127.

Rx DLL:

Receive DLL is a programmable delay line in the PHY that adjusts when the controller samples incoming data from the Flash. The DLL delays the sampling clock or DQS signal to align it with the center of the data valid window. Values typically range from 0 to 127.

Read Delay:

Read Delay is an additional programmable delay applied to the data capture timing, measured in reference clock cycles. It provides coarse adjustment complementing the fine-grained DLL settings. Values typically range from 0 to 4.

OTP:

Optimal Tuning Point is the specific combination of Tx DLL, Rx DLL, and Read Delay values selected by the tuning algorithm. It represents the center of a stable operating region with maximum margins in all directions.

Metastability Gap:

The Metastability Gap is a parameter space that is not uniformly passing or failing. Instead, it comprises a diagonal gap between regions where timing is metastable and reads fail, or boundaries that shift with temperature, voltage, and other environmental factors. This gap represents the transition zone where the sampling clock edge moves across the data transition edge. This is fundamentally unstable and must be avoided.

Radius Verification:

Radius Verification is a validation technique where all parameter combinations within a circular area around a candidate tuning point are tested. It ensures the selected point has adequate margins in all directions.

Diagonal Search:

Diagonal Search is the core search strategy used by the latest tuning algorithm. Instead of searching horizontally and vertically through the parameter space, the algorithm searches along the 45-degree diagonals that effectively traverse both passing regions and metastability gaps.

# 3 PHY Tuning Algorithm

The PHY tuning is the automated calibration process that determines the optimal timing parameters, when PHY is enabled.

The process involves:

- Writing a test pattern, known as the attack vector, to the flash memory.
- Systematically testing different combinations of timing parameters.
- Verifying flash reads of the test pattern at each parameter combination.
- Identifying regions where flash reads consistently succeed.
- Selecting an optimal tuning point (OTP) within the stable region with maximum margins.

The tuning algorithm must complete successfully before the system can reliably access flash memory at high speeds. Without tuning, systems need conservative timing settings that limit performance, or risk data corruption at extreme values of temperature and voltage.

**Note**

The algorithm, steps, parameters, and corresponding values have been configured in accordance with MCU+ SDK 11.02.

# 4 Key Tuning Parameters

## 4.1 Parameter Configurations for Non-DQS PHY Tuning Algorithm

| Parameter | Purpose | Typical Values | Impact |
|---|---|---|---|
| Minimum Read Delay | Lower bound for Read Delay search | 0 | Defined by hardware |
| Maximum Read Delay | Upper bound for Read Delay search | 3 | Defined by hardware |
| RxDLL Search - TxDLL High End | Highest value of Tx DLL to search in RxDLL window | 127 | Fixed as the TxDLL value during RxDLL window search |
| RxDLL Low Search Start | Starting Rx DLL for lower boundary search | 0 | Defines minimum Rx DLL to begin window search |
| RxDLL High Search End | Ending Rx DLL for higher boundary search | 127 | Defines maximum Rx DLL to search for window end |
| RxDLL and TxDLL Search Step | Step size for incrementing/ decrementing Rx and Tx DLL during search | 8 | Smaller values provide finer resolution but increase tuning time |

## 4.2 Parameter Configurations for DQS PHY Tuning Algorithm

| Parameter | Purpose | Typical Values | Impact |
|---|---|---|---|
| Search Radius | Size of circular area to validate around candidate tuning point | 10 | Larger values provide more margin but increase tuning time |
| Minimum Pass Size | Minimum squared length of a passing region must have to be considered valid | 100 | Larger values reject narrow regions that lack sufficient margin |
| Consecutive Pass Points | Number of adjacent passing points required to confirm a metastability gap | 10 | Higher values improve stability detection but can be harder to validate |
| Consecutive Fail Points | Number of adjacent failing points required to confirm a metastability gap | 5 | Helps distinguish genuine gaps from isolated failures |
| Diagonal Shift | How much to shift the search diagonal if the main diagonal fails | 10 | Larger values speed up the search but provide less thorough coverage |
| Maximum Diagonal Shift | Upper limit on diagonal shifting before declaring failure | 70 | Determines how exhaustively the parameter space is searched |
| Read Delay Search Step | Step size when initially locating valid Read Delay values | 16 | Larger values complete faster but can miss narrow passing regions |
| Minimum DLL Value | Lower bound for DLL parameter search | 0 | Defined by hardware |
| Maximum DLL Value | Upper bound for DLL parameter search | 127 | Defined by hardware |
| Minimum Read Delay | Lower bound for Read Delay search | 0 | Defined by hardware |
| Maximum Read Delay | Upper bound for Read Delay search | 4 | Defined by hardware |

# 5 Prerequisites for PHY Tuning Algorithm

## 5.1 Hardware Requirements

### *5.1.1 Flash Device Preparation*

- Verify that the flash device is properly initialized and accessible.
- The device must support the intended protocol mode.
- Erase flash from the region where the attack vector is to be written.

### *5.1.2 PHY Configuration*

- The reference clock must be stable and running at the correct frequency.
- Set the PHY mode (Master or Bypass) according to operating frequency.
- Configure the DLL lock mode (full cycle or half cycle) appropriately.

## 5.2 Attack Vector

The attack vector is a specific 128-byte pattern designed to reliably detect timing violations.

The pattern includes:

- Alternating bit patterns that create frequent data transitions.
- Long runs of zeros and ones to test for offset issues.
- Single-bit transitions to test bit-to-bit isolation.
- Even and odd byte alignments to cover both sampling phases in DQS mode.

Write the attack vector to a known flash address before tuning begins. Keep this address:

- In an accessible, non-protected region of flash.
- At a location that does not interfere with bootloader or application code.
- Properly erased before writing the pattern.

During tuning, the algorithm repeatedly reads from this address and compares the result to the expected pattern. Any mismatch indicates a timing violation at that parameter combination.

## 5.3 Passing vs Failing Region

The delay values for DQS PHY Tuning algorithm with Half Cycle Lock are stored in a three-dimensional array with dimensions:

- First dimension: Read delay values [0 to 4] - 5 total values.
- Second dimension: Transmit DLL values [0 to 127] - 128 total values.
- Third dimension: Receive DLL values [0 to 127] - 128 total values.

This results in an array of size [5][128][128] containing 81,920 total elements. The array is indexed as: array[readDelay][TxDLL][RxDLL].

Each array element stores a simple binary result:

- Passing regions are marked with value: 1 (indicating successful attack vector read).
- Failing regions are marked with value: 0 (indicating read error or data mismatch).

For example:

- array[2][64][64] = 1: At read delay=2, Tx DLL=64, Rx DLL=64, the test has passed.
- array[2][65][65] = 0: At read delay=2, Tx DLL=65, Rx DLL=65, the test has failed.

## 5.4 Master vs Bypass Mode

The OSPI PHY can operate in two fundamentally different modes that change how the DLL delay values are interpreted and applied.

The MCU+ SDK 11.02 selects the appropriate mode automatically based on the configured reference clock frequency:

- Reference Clock < 166MHz: Bypass mode is selected.
- Reference Clock ≥ 166MHz: Master mode is selected.

### *5.4.1 Bypass Mode*

In Bypass Mode, the DLL values directly control the number of physical delay elements used. Each delay element is a fixed buffer or gate that adds a small, absolute propagation delay. Bypass Mode follows direct mapping between the Tx DLL and the number of hardware elements used to delay the transmit clock signal. The total delay is then the sum of individual element delays. The delay in this mode is a fixed amount of time, independent of the operating clock frequency.

### *5.4.2 Master Mode*

In Master mode, a master DLL circuit continuously measures the clock period by locking the delay chain to produce either a full clock cycle or half clock cycle of delay. The Transmit and Receive DLL values are then interpreted as fractions of this reference delay, not as absolute delay element counts. There are two types of clocking mechanisms supported in MCU+ SDK 11.02:

- Full Cycle Lock: If the Master DLL locks to a full clock cycle, a DLL value of 64 represents 64/128 = 50% of one complete clock period.
- Half Cycle Lock: If the Master DLL locks to a half clock cycle, a DLL value of 64 represents 64/128 = 50% of a half cycle, which equals 25% of a full clock period.

In Master Mode, the delay is proportional to the clock period, automatically scaling with frequency. Half cycle lock provides finer resolution for precise timing control within smaller portions of the clock cycle. In this mode, delays are specified as fractions of the clock period, maintaining proper timing relationships regardless of the absolute clock frequency. This is essential for reliable operation at high-speeds where setup and hold times must be carefully controlled relative to clock edges.

# 6 Need for a Newer Tuning Algorithm

The new DQS algorithm addresses critical reliability limitations in the previous algorithm. The old approach selected tuning point based primarily on region size and temperature-dependent placement, without validating margins around the selected point, resulting in a tuning point that was positioned closer to the extremes, making the system more susceptible to noise-related variations. The previous approach used corner-finding with fixed Tx/Rx searches and binary search along boundaries, which could miss narrow regions or mischaracterize irregular passing region geometries. Additionally, the old approach lacked fallback strategies when the primary search failed. The new PHY tuning algorithm introduces circular radius verification to provide adequate margins in all directions, explicit metastability gap detection using consecutive passing and failing point counts which filters out isolated noise, diagonal search that follows the natural geometry of DQS passing regions, geometric midpoint calculation for optimal placement, minimum region size enforcement, and diagonal shifting as a fallback strategy. These enhancements help deliver reliable operation across diverse board designs, flash devices, and PVT conditions.

## 6.1 Temperature Variations

Temperature has a significant effect on semiconductor timing:

- Higher Temperature results in slower operation. As temperature rises, electrons move through transistors more slowly due to increased atomic vibration in the silicon crystal lattice.
- Lower Temperature results in faster operation. At cold temperatures, the atomic lattice is less active, allowing electrons to move more freely and quickly through transistors.
- Over a typical operating temperature range (-40°C to +125°C), delay elements can change the propagation delay characteristics, causing a shift of DLL positions in the parameter space.

The metastability gap moves as temperature changes because the relative timing between signals shifts. The newer DQS tuning algorithm compensates by selecting a point with sufficient margin so that the passing region remains valid across the full temperature range.

A comparison of the OTP values computed by the old and new algorithms is presented in the figure below.

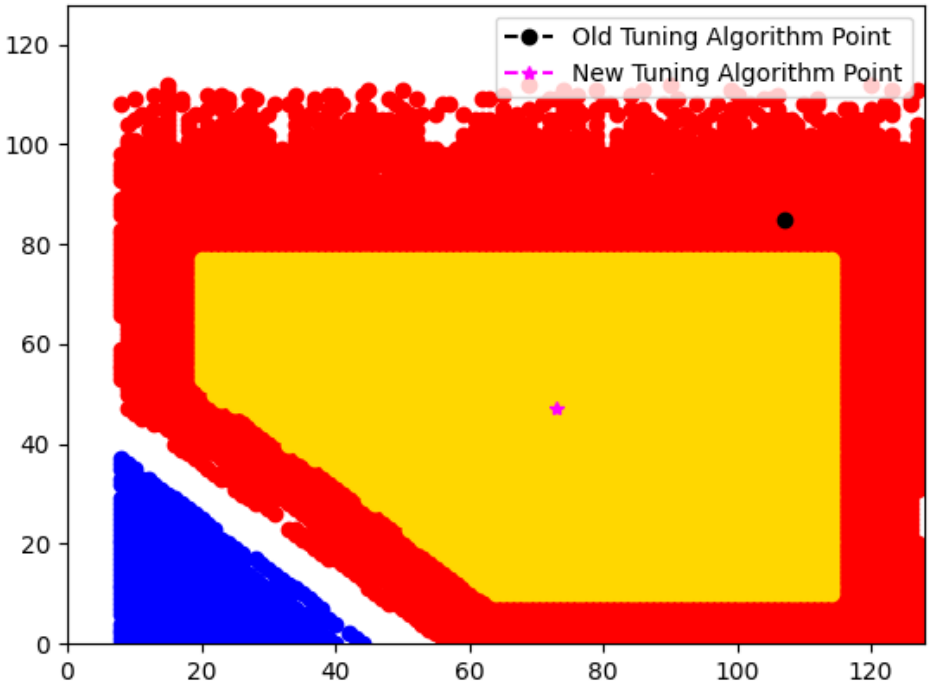


**Figure 6-1. Comparison of old and new OTP**

# 7 Algorithm Implementation

## 7.1 DQS PHY Tuning Algorithm

The DQS tuning algorithm uses a diagonal search strategy to identify optimal timing parameters for high-speed operation with DQS signaling.

### *7.1.1 Diagonal Selection*

The algorithm begins searching along a diagonal from (0,0) to (127,127), representing equal increments in TX and RX DLL values. If no valid tuning point is found, the diagonal is shifted upward by 10 points (increasing Y-offset) or rightward by 10 points (increasing X-offset) in an alternating pattern. Shifts continue up to a maximum of 70 units from the original diagonal, enabling systematic coverage of the parameter space while maintaining a 45-degree slope.

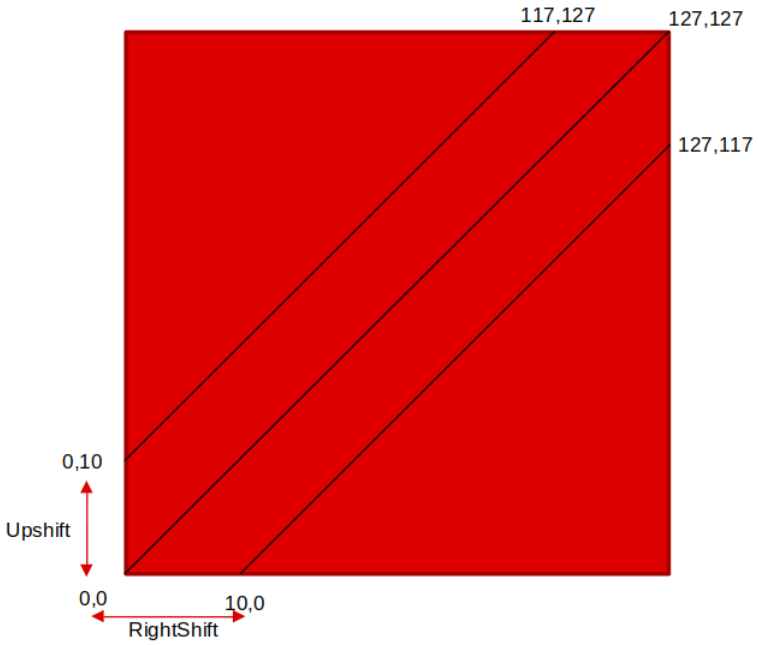


**Figure 7-1. Diagonal Selection**

### *7.1.2 Valid Read Delays Selection*

Starting at the minimum read delay, the algorithm performs a coarse search in steps of 16 along the diagonal to identify regions where reads succeed. A passing point confirms a valid operating region exists at that read delay setting. Once found, fine-grained searching in steps of one maps the complete valid region. The process repeats for each read delay value. If no passing points are found across all read delays, the diagonal is shifted to find a new potential operating point.

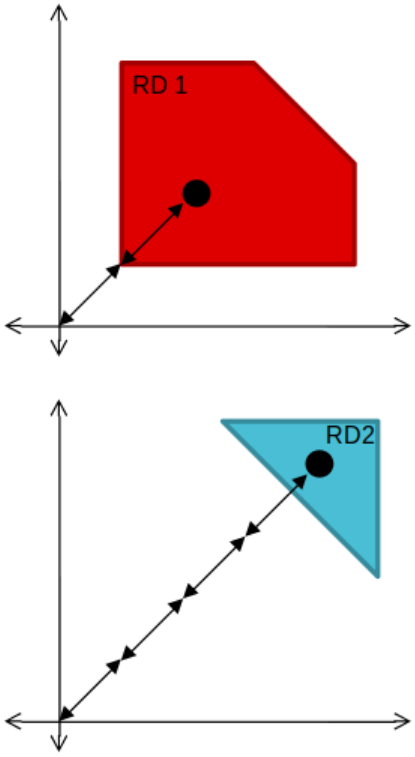


**Figure 7-2. Valid Read Delays Selection**

### *7.1.3 Corner Points Identification*

The algorithm locates precise boundaries of passing regions along the diagonal. For the minimum valid read delay, the algorithm searches from the diagonal start point by incrementing Tx DLL and Rx DLL in steps of 1 until finding the first passing point (lower boundary). For the maximum valid read delay, the algorithm searches from the diagonal end point by decrementing both DLLs until finding the first passing point (upper boundary). These corner points define the complete range where the interface operates reliably.

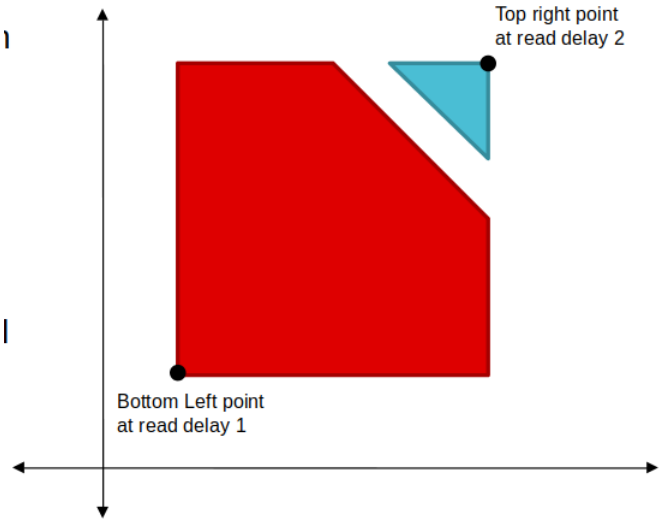


**Figure 7-3. Corner Points Identification**

#### 7.1.3.1 Corner Point Selection for Only One Read Delay Value

When identical read delays exist at both endpoints, a single passing region is present. The algorithm finds corner points of the line with the highest number of consecutive passing points, representing a stable sampling window. If the line length, which is the distance between corner points, exceeds the minimum pass length threshold, tuning point selection proceeds. Otherwise, a failure is reported as the window is too narrow for reliable operation.

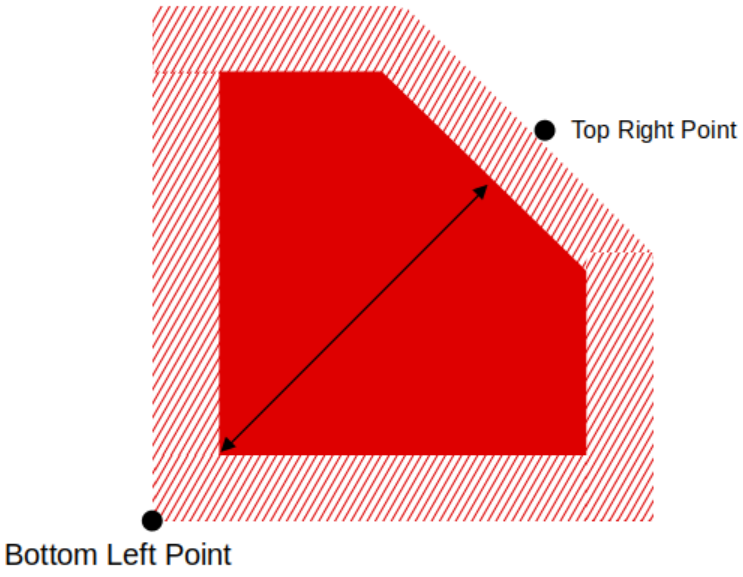


**Figure 7-4. Corner Point Selection for One Read Delay Value**

#### 7.1.3.2 Corner Point Selection for Two Different Read Delay Values

When different read delays appear at diagonal endpoints, two passing regions exist separated by a metastability gap. The algorithm refines corner points by finding points with minimum consecutive passing points, testing multiple adjacent points to improve stability. For each valid point, the algorithm identifies the first failing point and confirms minimum consecutive failing points to establish clear boundaries. Regions without sufficient consecutive passing points are eliminated as unstable. The algorithm searches upward along the diagonal for the minimum read delay and downward for the maximum read delay.

After refining, the algorithm identifies the line with the largest number of passing points. If either line length exceeds minimum pass length, tuning point selection proceeds; otherwise, calibration failure is reported.

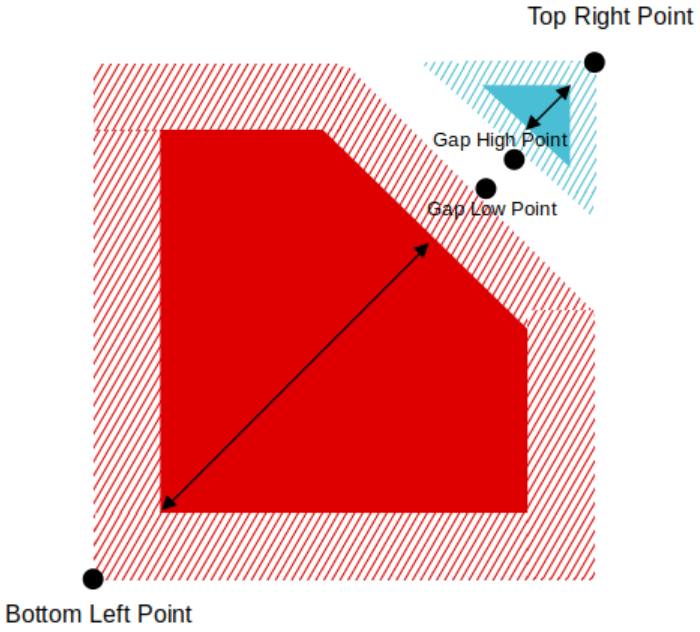


**Figure 7-5. Corner Point Selection for Two Different Read Delay Values**

### *7.1.4 Tuning Point Selection*

The algorithm selects the region with larger length and calculates the midpoint (midpoint1). Corner points of a line perpendicular to the diagonal at midpoint1 are identified, representing consecutive passing points. The midpoint of the perpendicular line (midpoint2) is calculated, representing the center width of the stable region. Radius verification tests whether all points within a circle of specified radius around midpoint2 pass. If successful, midpoint2 becomes the optimal tuning point.

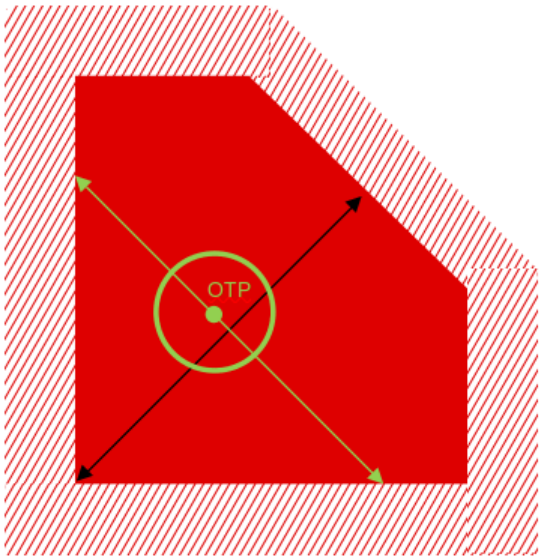


**Figure 7-6. Tuning Point Selection at Midpoint**

If radius verification fails, the perpendicular line is divided into two segments at midpoint1. The midpoint of the larger segment (midpoint3) undergoes radius verification. If successful, midpoint3 becomes the tuning point.

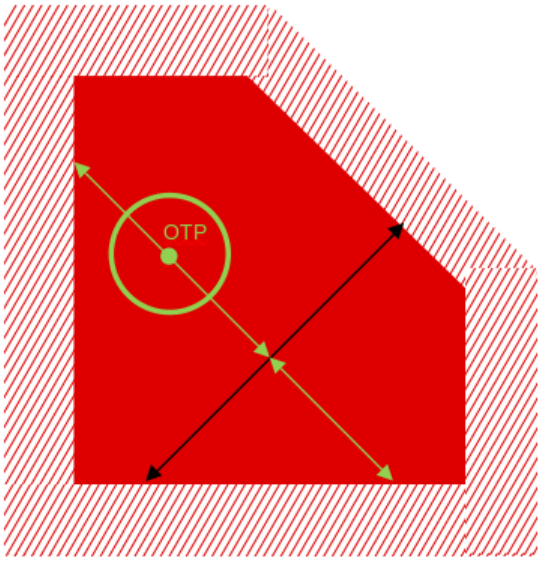


**Figure 7-7. Tuning Point Selection at Midpoint of Larger Segment**

If both midpoint2 and midpoint3 fail, the second read delay region is evaluated if the length exceeds minimum pass length, repeating the selection process.

If all regions and segments fail, the algorithm reports tuning failure and selects a different diagonal, following a predefined pattern to enable comprehensive coverage. Each diagonal represents different delay parameter

combinations, offering multiple opportunities to find a stable operating region.

In case of PHY failures, see the PHY Failure section in the *Common Bring-up Issues and Debugging* Application Note.

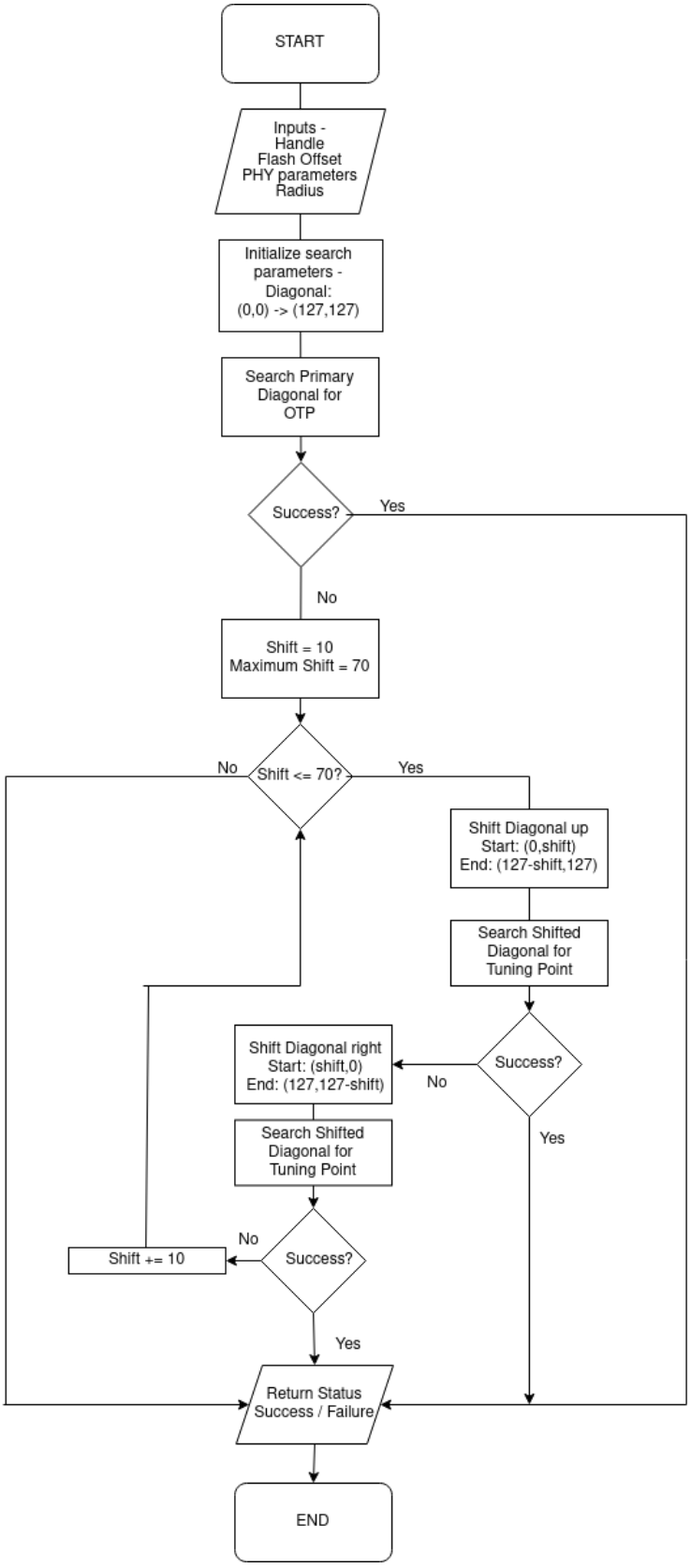


**Figure 7-8. Flowchart of DQS Tuning Algorithm**

## 7.2 Non - DQS PHY Tuning Algorithm

The Non-DQS tuning algorithm uses a simplified window-based approach designed for lower-speed operation without DQS signal support. Unlike the two-dimensional diagonal search in the DQS algorithm, the Non-DQS algorithm performs a one-dimensional search along the Rx DLL axis while maintaining a fixed Tx DLL value.

### *7.2.1 Fix Tx DLL Value*

The algorithm begins by setting the Tx DLL to a predetermined high value. This fixed configuration provides adequate setup time for most flash devices that support Non-DQS mode and reduces the search problem from two dimensions to one dimension. By eliminating the need to optimize Tx DLL, the algorithm can now focus exclusively on finding the optimal Rx DLL timing.

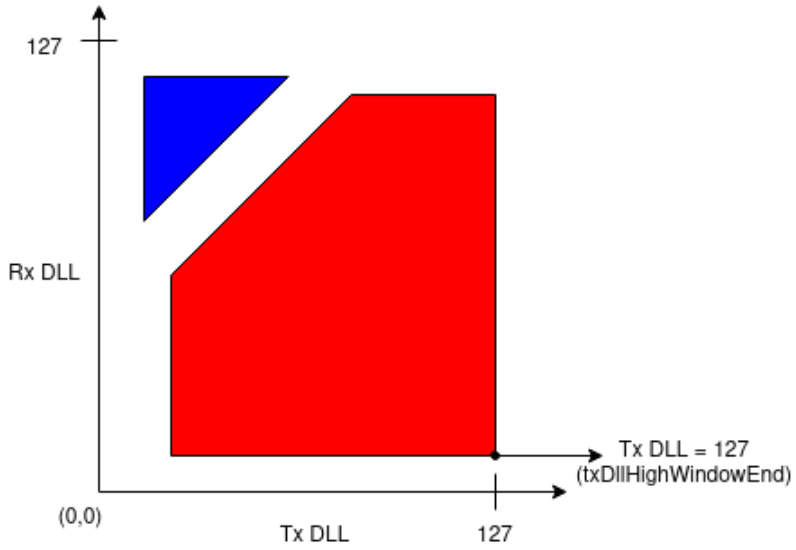


**Figure 7-9. Fix Tx DLL Value**

### *7.2.2 Find Rx Window 1*

Starting at the minimum read delay value, the algorithm searches for the first valid Rx DLL window. The Rx DLL is incremented from the minimum value while testing attack vector reads at each step. The first DLL value where reads succeed becomes the window start (rxStart1), and the last value before failures resume becomes the window end (rxEnd1). The window size is calculated as, rxWindow1 = rxEnd1 - rxStart1.

If no passing window is found at the current read delay, the algorithm increments the read delay and repeats the search. This process continues until a window is found or the maximum read delay is exceeded. The passing window represents a contiguous range of receive DLL values where interface timing meets all setup and hold requirements.

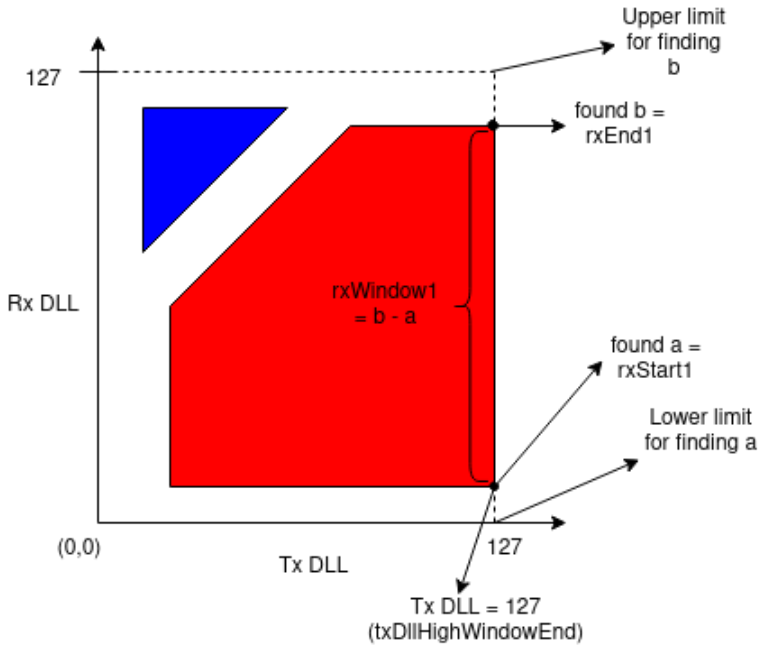


**Figure 7-10. Selection of Rx Window 1**

### *7.2.3 Find Rx Window 2*

After identifying the first window, the algorithm attempts to locate a second window at the next higher read delay value (read delay of window 1 + 1). The same search process is repeated to find rxStart2 and rxEnd2, and the second window size is calculated as rxWindow2 = rxEnd2 - rxStart2.

If no second window is found, rxWindow2 is set to 0, which is not considered a failure. Different read delay values shift the relative timing between the reference clock and data sampling point. Testing multiple read delay values so that the algorithm finds the largest available window and provides maximum margin against environmental variations.

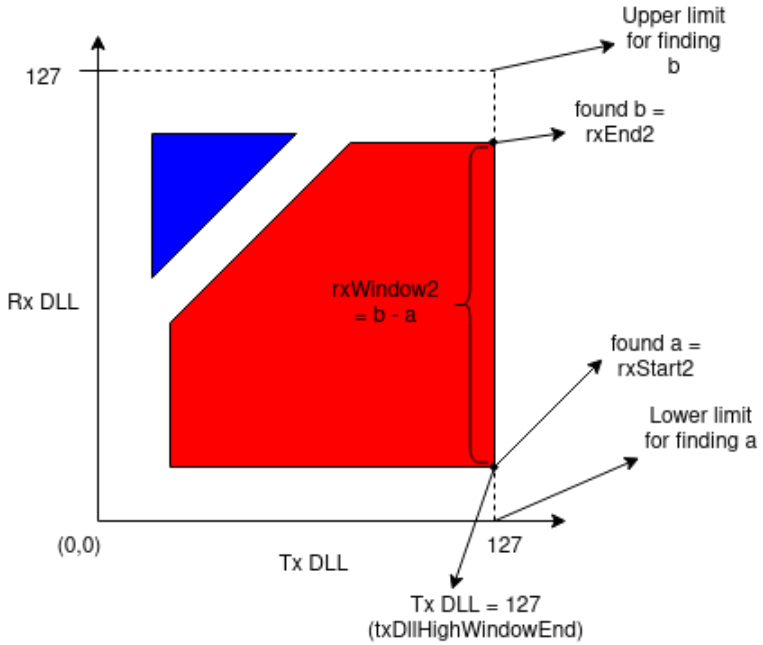


**Figure 7-11. Selection of Rx Window 2**

### 7.2.4 Choose Larger Rx Window

The algorithm compares the two window sizes and selects the larger one. If rxWindow2 is larger than rxWindow1, Window 2 is selected with the associated parameters (rxStart2, rxEnd2, and read delay). Otherwise, Window 1 is used. Larger windows indicate greater timing margin and better tolerance for temperature changes, voltage fluctuations, and manufacturing variations.

### 7.2.5 Calculate the OTP

The final tuning point is determined using three parameters:

1. Transmit DLL: The fixed value from Step 1.
2. Read Delay: The Read Delay associated with the selected window.
3. Receive DLL: The midpoint of the selected window = rxStart + (rxEnd - rxStart) / 2.

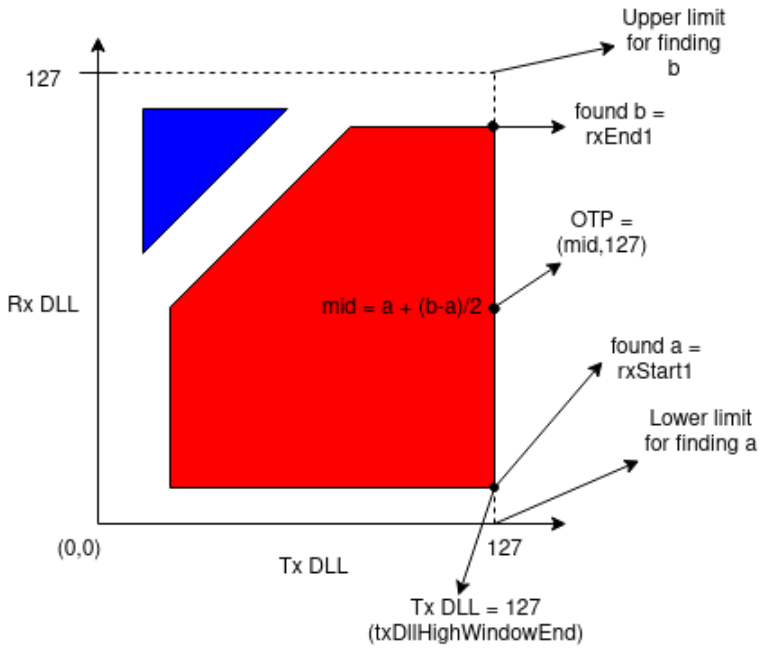


**Figure 7-12. OTP Calculation**

### 7.2.6 Temperature Consideration

The internal temperature sensor of the SoC, which is the VTM module, measures the current die temperature. Using a reference temperature of 42.5°C, the algorithm calculates a temperature-dependent offset:

Temperature Factor = (Current Temperature - 42.5°C) / 165°C × Window Size × 0.75

The adjusted Receive DLL then equals the difference between the midpoint and Temperature Factor. At higher temperatures, the tuning point shifts toward lower receive DLL values because delays increase with temperature. At lower temperatures, the tuning point shifts toward higher values because delays decrease.

After calculating the tuning point, the parameters are applied to the PHY hardware registers. Validation reads of the attack vector confirm successful tuning before the algorithm returns success.

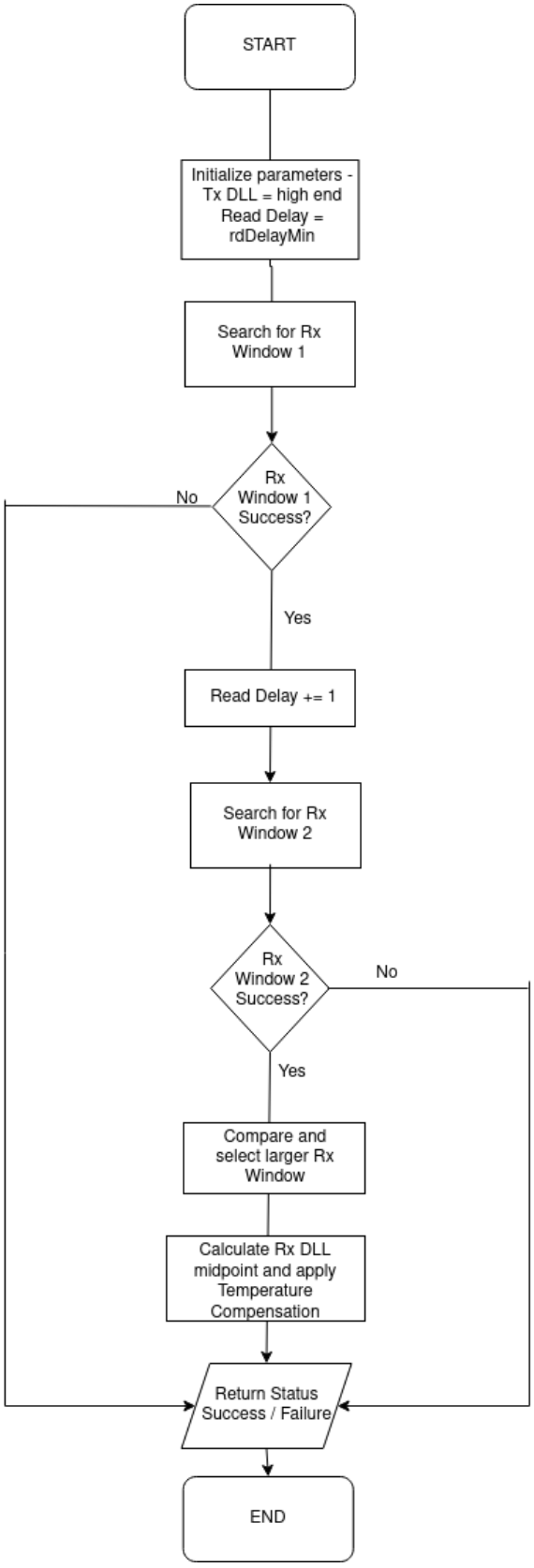


**Figure 7-13. Flowchart of Non-DQS Tuning Algorithm**

# 8 Tuning Enhancements

## 8.1 Tuning Time Optimization – Skip Tuning Feature

MCU+ SDK 11.02 has a feature to skip PHY Tuning. This feature reduces boot time for later stages. For systems with multi-stage boot processes, tuning can be performed once in the first stage, such as in the primary bootloader, and reused in subsequent stages by performing the following steps:

1. First boot stage performs the complete tuning and stores the parameter values in hardware registers.
2. When Skip Tuning is enabled in SysConfig, subsequent stages retrieve the saved parameters for direct application by the driver.

## 8.2 Runtime Validation – Validate OTP

MCU+ SDK 11.02 has a feature to Validate OTP. When enabled in SysConfig, applications re-validate the tuning point during PHY enabled flash read operation. The validation process works by performing the following steps:

1. During flash read operations, OTP Validation is automatically invoked if enabled in SysConfig.
2. The feature performs a diagonal check around the current tuning point within a circular area of the configured radius.
3. If validation fails, the PHY Tuning algorithm re-runs automatically.

# 9 Summary

The OSPI PHY tuning algorithm automatically calibrates timing parameters to enable reliable high-speed flash memory access. DQS mode uses a diagonal search strategy that efficiently characterizes passing regions, calculates optimal midpoints, and verifies margins through circular area testing. Non-DQS mode uses a simpler one-dimensional window search with temperature compensation. The latest DQS tuning algorithm compensates for manufacturing variations and environmental changes affecting semiconductor timing characteristics. Engineers can optimize using skip-tuning for faster boot, or pre-characterization for production.

# 10 References

1. Texas Instruments, *AM62x MCU+ SDK* .
2. Texas Instruments, *OSPI Controller PHY Tuning Algorithm*, Application Note
3. Texas Instruments, *OSPI PHY Tuning Algorithm*, Github Repository
4. Texas Instruments, *Throughput Characterization of OSPI and QSPI Serial NOR/NAND Flash Operation on MCU+ SDK* , Application Note
5. Texas Instruments, *xSPI Custom Flash Debug Guide for MCU+SDK* , Application Note

# 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