



xSPI PHY Configuration and Training

Contributed by Manoj Chitneedi

Rev 1 – July 8, 2025

Introduction

This application note describes the expanded serial peripheral interface (xSPI) mode and provides programming and board design guidelines. It also describes the PHY training process and the usage of the ADI SSLD PHY driver which incorporates the PHY training API. This note applies to the ADSP-2183x and ADSP-SC83x processors families. It is an addendum to the processor data sheet^[1] and the hardware reference manual^[2].

PHY

xSPI PHY is responsible for handling the low-level timings of the data, address, and control signals between the memory device and controller. Before the host can communicate with the memory device, the PHY must be initialized and locked. The soft PHY contains three synthesizable delay lines (DLL) that are built from 256 delay elements.

- Master DLL—Primary purpose is to measure how many delay elements are needed to match the XSPI_CLK clock period.
- TXDLL—Uses the measurement of the master DLL to influence how much delay to add to transmitted data as a percentage of the clock period (to improve the data-eye at the memory device).
- RXDDL—Used to shift the read sampling clock as a percentage of the clock period to find the ideal sampling point for the inbound read data.

Before this can happen, the master DLL must first be set up, which is done immediately following power-on reset (PoR). This is known as PHY locking. PHY locking is a fully internal PHY function that aims to lock the master DLL. Until the PHY is locked, there can be no access to/from the memory device.

Optionally, master DLL can be disabled by master DLL disabled mode. TXDLL and RXDLL cannot be programmed based on a percentage of the clock period and instead must be set up manually. This is a suitable mode when operating at slower data rates where the master DLL does not have enough delay elements to measure the complete clock period. Running at these slower rates does not require such fine tuning of the DLLs (particularly if running at SDR as well) and therefore, manual setup is acceptable.

PHY Training

PHY training is a software routine implemented to help with fine tuning the PHY to find the perfect sampling point, thus allowing the PHY to achieve operating xSPI at the highest possible xSPI_CLK frequency. A memory device read operation is performed iteratively with all combinations of RXDLL and TXDLL. A known pattern of read values is used to verify the read data. The optimal value of any wide range of passing patterns of RXDLL and TXDLL are chosen to configure the DLL registers. When the operating temperature drifts on either side, selecting the correct DLL values from a wide range of passing values provides a sustainable operation of the PHY.

- In master DLL enabled mode (recommended mode), DLL can be precisely delayed in $1/256^{\text{th}}$ steps of the clock period.
- On the transmit side, the DQ line can be delayed using the TXDLL.
- On the receive side, the sampling clock (both DQS/RWDS and non-DQS modes) is delayed by the RXDLL.
- Initial values of TXDLL and RXDDL are based on setup times, skewing between DQS and the data, and board delay. All affect where the data-eye may begin.

Initial values for the DLLs can be calculated with the help of the script provided. However, performing a full training sequence is the only way to calculate the most optimum delay value to find the center of the data-eye.

Temperature Impact on Passing Range

Processor die temperature affects the IO line delays and can cause a change in the passing range boundaries. The passing range of TXDLL and RXDLL can shrink or extend based on the operating temperature, board design, and other factors.

Figure 1 and Figure 2 show the RXDLL and TXDLL passing range for different temperature ranges. It is recommended to choose the mid value of the passing range for optimal operation when the silicon temperature drifts (even after calibration and DLL configuration).

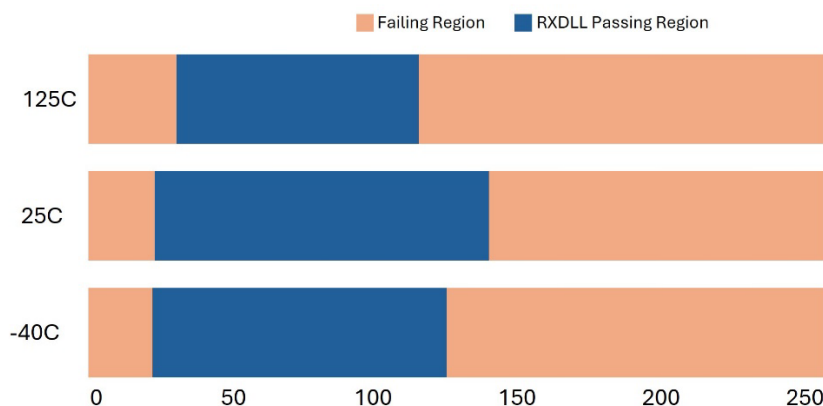


Figure 1. RXDLL Passing Range vs. Temperature

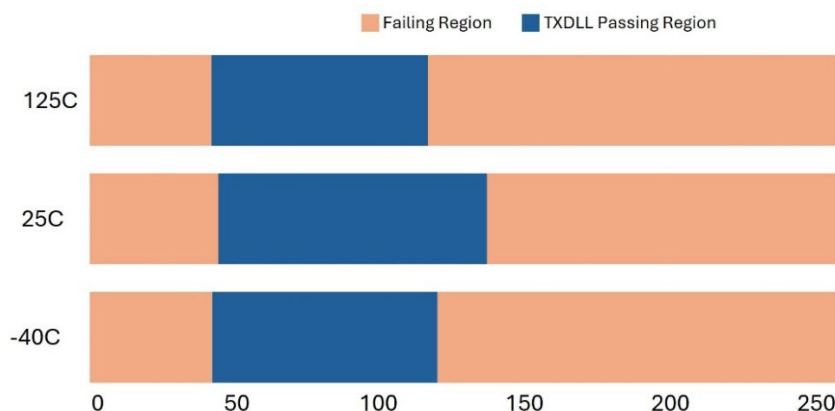


Figure 2. TXDLL Passing Range vs. Temperature



In Figure 1 and Figure 2, passing values were determined by randomly selecting sample values during multiple in-house experiments. Passing ranges and values vary with respect to operating frequency, clock source, and other factors.

Generating Initial PHY Configuration Settings

The attached zip file (*ee-468_xSPI_InitialPHYConfig.zip*) contains a Python script (*Initial_PHY_Config.py*) that generates the initial PHY configuration settings based on the provided input details. The script uses *flash_phy_reg_input.txt* as the input file, which contains the following parameters:

- **dqs_en**—Configures the device in read mode, where it generates a strobe (DQS or RWDS).
- **lpbk_en**—Configures the device in read mode, where it does not generate a strobe (DQS or RWDS). In this case, the controller uses an internal loopback clock for sampling.
- **clk_period**—Defines the flash clock period.
- **flash_setup** —Specifies the flash setup timing, which is the time from the last negative clock edge to valid data (for read commands).
- **board_delay**—Estimated board delay, which includes the total round-trip delay for the signals either from simulations or as calculated below. This value helps determine the timing for data read capture. Example formula: $\text{board_delay} = \text{PCB trace to device} + \text{PCB trace from device}$.
- **tdqs** —Defines the maximum timing skew between the DQS (RWDS) strobe and the stable data window (DQ).
- **tx_flash_setup** — Specifies the minimum setup timing for data transmitted by the controller (referred as tDVCH).

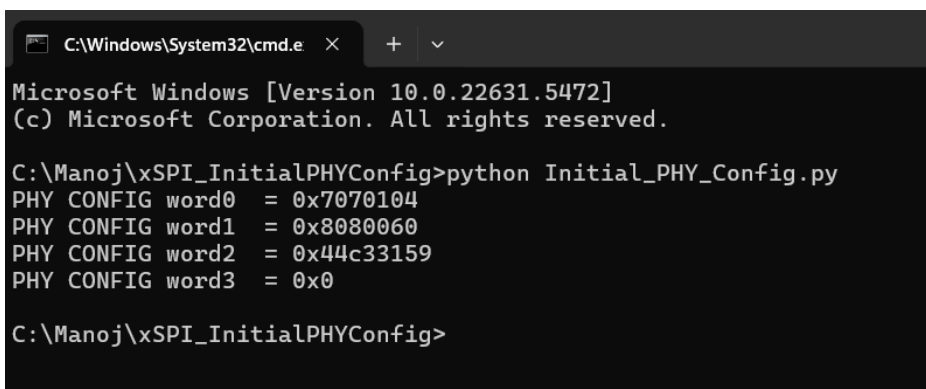


All above timing parameters should be specified in ns. Even if a parameter does not exist for the device, make sure a dummy value is passed and ignore the corresponding output file.

For example, for the S26KL512S HyperFlash part, the inputs to be provided for 125 MHz DQS_EN case are as follows:

dqs_en	1
lpbk_en	0
clk_period	8.000000
flash_setup	5.50000 (t_{CKD} in data sheet)
board_delay	4.90000 (from simulations)
tdqsq	0.6000 (t_{DSS} in data sheet)
tx_flash_setup	0.8 (t_{IS} in data sheet)

Use the above output words generated from script to program the PHY after initialization and booting of flash controller is over.



```

C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.22631.5472]
(c) Microsoft Corporation. All rights reserved.

C:\Manoj\xSPI_InitialPHYConfig>python Initial_PHY_Config.py
PHY CONFIG word0 = 0x7070104
PHY CONFIG word1 = 0x8080060
PHY CONFIG word2 = 0x44c33159
PHY CONFIG word3 = 0x0

C:\Manoj\xSPI_InitialPHYConfig>

```

Figure 3. Screenshot of Script Output with 4 Words Generated

The parameters calculated have been validated in our verification but depending on the use case and memory models used, they do not need to be optimal. The calculations should be treated as a reference.

PHY Training Algorithm

The ADI SSLD PHY driver incorporates a PHY training sequence designed to tune and select the optimal RXDLL and TXDLL values based on the chosen clock configurations. This training algorithm supports all xSPI transfer modes, including single, DPI, dual IO, QPI, quad IO, OPI, and octal IO, as well as both SDR and DDR modes.

The algorithm works by reading a known pattern from xSPI memory and iterating through multiple RXDLL and TXDLL values. By evaluating various parameters, it identifies the optimal RXDLL and TXDLL values from a wider range of passing values. This approach ensures stable operation of the xSPI PHY, even if temperature fluctuations occur after training.

Training Sequence

Before initiating the training sequence, the user must program the flash device with the desired training data sequence once, using a very low speed—such as 16 MHz—where PHY training is not required. The static configuration file includes a few initial settings, as outlined below.

```
/* PHY training configurations */

/*! Valid Start Point of TXDLL */
#define ADI_XSPI_TXDLL_START_POSITION      (5u)

/*! Valid Start Point of RXDLL */
#define ADI_XSPI_RXDLL_START_POSITION      (5u)

/*! Stable Region Start point of TXDLL */
#define ADI_XSPI_TXDLL_STABLE_REGION      (10u)

/*! TXDLL Retry Stride value */
#define ADI_XSPI_TXDLL_RETRY_STRIDE        (5u)

/*! RXDLL passing range margin */
#define ADI_XSPI_RXDLL_PASSING_RANGE      (10u)

/*! TXDLL passing range margin */
#define ADI_XSPI_TXDLL_PASSING_RANGE      (10u)

/*! Training Data Size */
#define XSPI_TRAINING_DATA_SIZE            (32u)
```

Figure 4. Input Parameters from Static Config File

In the input macros listed above:

- ADI_XSPI_RXDLL_START_POSITION and ADI_XSPI_TXDLL_START_POSITION define the starting delay cell values for sweeping up to 255.
- ADI_XSPI_TXDLL_STABLE_REGION is used in Step 2 (see below) as an offset added to the TXDLL value generated by the script, ensuring a reliable TXDLL pass value.
- ADI_XSPI_TXDLL_RETRY_STRIDE defines the retry stride value of TXDLL used during training.
- ADI_XSPI_RXDLL_PASSING_RANGE and ADI_XSPI_TXDLL_PASSING_RANGE specify the minimum number of delay cells passing. Choosing a higher value tends to be more conservative and may sometimes result in the PHY training failing to produce valid TXDLL and RXDLL values.
- XSPI_TRAINING_DATA_SIZE indicates the number of bytes utilized during training.

It is recommended not to change the values of START_POSITION, TXDLL_STABLE_REGION, and RETRY_STRIDE. Use the defaults defined in the static configuration file to maintain training reliability.

The following steps are involved in the PHY training:

1. *Initialization*

- ❑ Generate the initial TXDLL value via script.
- ❑ Set the starting RXDLL value to 5, as defined in the static config file.

2. *Establish Stable TXDLL Start*

- ❑ Shift the TXDLL by 10 steps (as per the static config file) to move into a stable, passing region.
- ❑ Label this new TXDLL value as the *TXDLL actual start value*.

3. *Sweep for Valid RXDLL Window*

- ❑ Iterate RXDLL from RXDLL_START to 255.
- ❑ Determine RXDLL_MIN, RXDLL_MAX, and compute RXDLL_MID (Max + Min)/2.
- ❑ If the valid window (RXDLL_MAX – RXDLL_MIN) is < 10, increment RD_DEL_SEL by 1 and repeat the sweep.
- ❑ If all RD_DEL_SEL values fail to yield a window ≥ 10 , return to Step 2 to try a new TXDLL actual start value.

4. *Fine-Tune TXDLL*

- ❑ Set RXDLL to the midpoint (RXDLL_MID) found in Step 3.
- ❑ Sweep TXDLL from TXDLL_START to 255 to identify TXDLL_MIN, TXDLL_MAX.
- ❑ Compute midpoint: $\text{TXDLL_MID} = ((\text{TXDLL_MAX} - \text{TXDLL_MIN}) / 2) + \text{TXDLL_MIN}$.
- ❑ Ensure all values between TXDLL_MIN and TXDDL_MAX pass.

5. *Final RXDLL Validation*

- ❑ With TXDLL fixed at its midpoint, resweep RXDLL to verify the final range.
- ❑ Expect a passing window where $\text{RXDLL_MAX} - \text{RXDLL_MIN} \geq 10$.
- ❑ The final RXDLL_MID should remain unchanged or deviate by no more than ± 5 from the previous value.

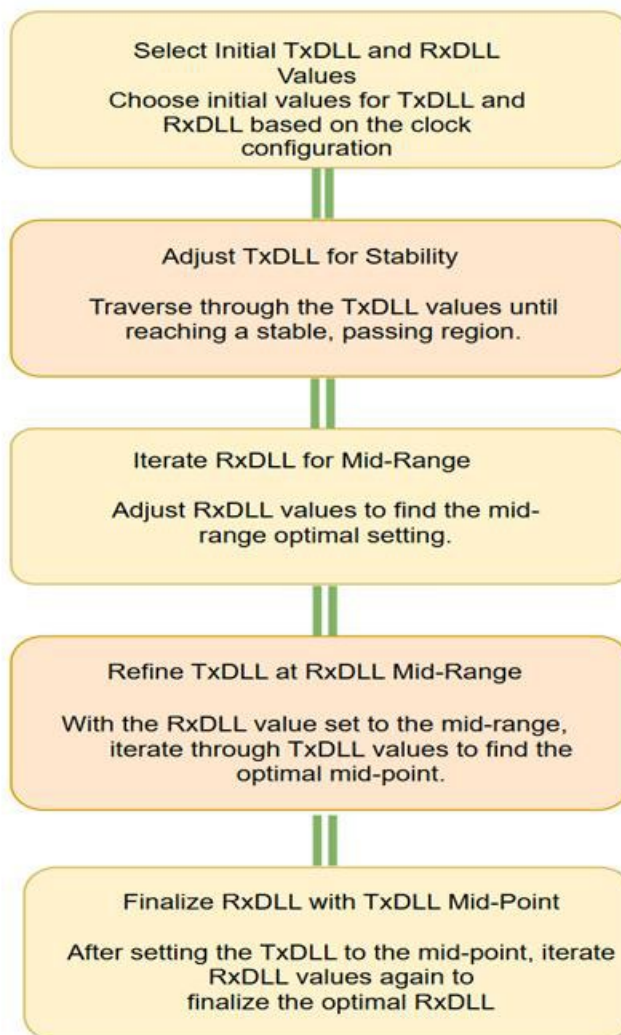


Figure 5. PHY Training Flow

Offline Vs Online PHY Training

Offline PHY training involves executing the training algorithm on a limited set of parts using a specific setup. The resulting PHY configuration values are then applied uniformly across all parts, eliminating the need to run PHY training on each unit. This method supports a maximum xSPI operating frequency of 62.5 MHz in non-DQS mode and 80 MHz in DQS mode.

For boot scenarios, the configuration values can be programmed into OTP (One-Time Programmable memory) or included in the boot stream. During boot, the ROM code initializes the PHY with these preset values before accessing the boot stream via the xSPI peripheral.

Online PHY training performs the training algorithm on every individual part, selecting optimal PHY configuration values per unit. This allows for higher performance, enabling up to 80 MHz in non-DQS mode and 125 MHz in DQS mode. To enable this method during boot, initialization code is required to run PHY training dynamically on each device.

SSLD PHY Configuration Structure

The xSPI SSLD driver provides configuration structures that define the initial configuration for PHY generated from the script. Listing 1 shows the configuration settings. This structure must be configured before calling `adi_xspi_PhyInit` or `adi_xspi_PhyTraining` API.

```
/**
 * \struct ADI_XSPI_PHYCONFIGURATION
 * Defines configuration for PHY mode of operation.
 *
 * */
typedef struct
{
    uint32_t XspiPhyWord0; /* xSPI PHY Word 0 */
    uint32_t XspiPhyWord1; /* xSPI PHY Word 1 */
    uint32_t XspiPhyWord2; /* xSPI PHY Word 2 */
    uint32_t XspiPhyWord3; /* xSPI PHY Word 3 */
}ADI_XSPI_PHYCONFIGURATION;
```

Listing 1. Configuration Settings

SSLD PHY Calibration Example

```
eResult = adi_xspi_Open( (uint8_t *)nDriver_Memory + ADI_SPU_MEMORY_SIZE,
    ADI_XSPI_MEMORY_SIZE,
    XSPI_ADI_MDMAUSED,
    MDMA_CHANNEL_USED,
    phDevice
);

if(eResult != ADI_XSPI_SUCCESS)
{
    printf("Failed to Open XSPI driver. Error code returned %d \n", eResult);
    return eResult;
}

eResult = adi_xspi_RegisterCallback ( *phDevice,
    NULL,
    XspiMdmaCallback,
    phDevice);

if(eResult != ADI_XSPI_SUCCESS)
{
    printf("Failed to Register XSPI Callback. Error code returned %d \n", eResult);
    return eResult;
}

Phy_Config.XspiPhyWord0 = 0x07070104;
Phy_Config.XspiPhyWord1 = 0x08080060;
Phy_Config.XspiPhyWord2 = 0x44c33159;
Phy_Config.XspiPhyWord3 = 0x00000000;

eResult = adi_xspi_PhyTraining(*phDevice, &DAC_Program_Command, &DAC_Read_Command, &Phy_Config);
```

Listing 2. Snippet Code from CCES

For the S71KL512SC0BHB003 device operating at 125 MHz and using the PHY configuration generated by the script above, the estimated training time is approximately 17 ms. This is a reference value and may vary with the specific component or PCB layout.

Hardware and Design Guidelines

For high-speed signals, maintain good signal integrity in the PCB design. The following general layout guidelines are recommended to prevent signal integrity problems.

- To maintain the timing margins, match trace lengths of all data lines within +/- 3 mils relative to the xSPI_DQS signal. The xSPI_DQS signal should be matched with respect to the clock signal.
- Route all data signals and xSPI_DQS on the same signal layer, having the same ground reference. Changing the ground reference plane can change the trace impedance.
- All data signals and strobes should have the same number of vias and layer changes. This design helps to ensure that the data signals along with the accompanying strobe will have the same effective delay.
- Avoid routing over a split plane as the return current path is not able to follow the signal trace. If a signal must be routed over two different reference planes, add a stitching capacitor between the reference planes and place the capacitor close to the signal path.

- If the signals change layers and reference planes from one ground plane to another, add ground vias or capacitors close to the layer change vias to avoid return path discontinuities.
- Use a ground plane as the primary reference or return paths for all signals. Whenever a power layer is used as the reference plane, it is important to ensure that the power layer is low-noise and there is proper stitching at the reference plane transitions to guarantee return path continuity.
- Maintain at least 3x spacing of the trace width for clock and strobe signals from other signal traces to minimize noise coupling.
- Route all the signals with the controlled impedance (typically 50 ohm) to reduce signal reflection.
- Use minimum number of vias in the clock trace. A via causes impedance discontinuity and signal reflections.
- The xSPI_DQS trace should be shorter than all data lines. Route the clock and strobe line (DQS) as short as possible. Try to avoid using serpentine routing and maintain a straight trace as much as possible, to minimize impedance variation. The maximum recommended trace length for the xSPI signals is 4 inches.
- Keep stubs as short as possible to avoid reflections. Keep stub propagation delay to <20% of the signal rise time.
- The xSPI_MISO signal requires a pull-up resistor. To ensure the correct signal level on the data pins during tri-state (Hi-Z), external pull-ups can be used on data pins (D0-D7).
- The pull-up resistor on the slave select signal ensures that the memory is deselected when the pin is in high-impedance mode, such as during reset.

References

- [1] *ADSP-21834/21835/21836/21837/ADSP-SC834/SC835 High Performance SHARC-FX DSP Core With Arm-Based Connectivity Preliminary Data Sheet, Rev PrE, May 2024*
- [2] *ADSP-2183x/ADSP-SC83x SHARC-FX Processor Hardware Reference, Rev 0.1. March 2024, Analog Devices, Inc.*

Document History

Revision	Description
<i>Rev 1 – July 2025 by Manoj Chitneedi</i>	Initial Release.