**NXP Semiconductors**
Application Notes

Document Number: AN13563

Rev. 0, 02/2022

# S32G QuadSPI Deep Dive

by: NXP Semiconductors

## 1. Introduction

This application note provides an in-depth description of the QuadSPI controller based on the S32G family of devices, mainly including the following parts:

- Primarily it explains the main supported features of the QuadSPI controller.
- To facilitate the use or porting of the external flash memory devices, the configuration of the QuadSPI pins, clocks, and registers is described in detail.
- The S32G chip supports loading images and booting from external flash via QuadSPI interface, the requirements of the external flash device which can support booting, as well as reconfiguration parameters are introduced.
- How to add new a flash algorithm via the Flash SDK is also described.
- In addition, NXP provides flash driver FLS based on RTD/MCAL, how to configure the FLS driver via the EB tool is also included in this document.
- Finally, for the debugging of QSPI, some tips, and examples of physical layer waveform analysis are given.

**Contents**

# 2. QuadSPI overview

S32G is the ideal solution for gateway and safety domain controller. S32G  does not have internal flash, the QuadSPI interface is used to connect external flash memory. To improve the efficiency of accessing external flash, QuadSPI has many features to accelerate the access efficiency. The main support features are:

- Command-driven interface: Flexible sequence engine to support various flash memory vendor devices
- Multiple operation modes: Interface like standard SPI but optionally utilizes 2(Dual), 4(Quad), or 8(Octal) data lines to transfer
- Multiple sampling modes: Can support SDR(Single Data Rate)/STR(Single Transfer Rate) and DDR(Double Data Rate)/DTR(Double Transfer Rate) mode to further increase throughput
- Multiple access modes: AHB master to read RX buffer data through AMBA AHB (64-bit width interface) or IPS registers space (32-bit access) and fill TX buffer via IPS register space (32-bit access)
- Supports for all types of addressing: Typically, 24-bit/32-bit addressing

**NOTE**

- Currently, all tests and waveforms in this document are based on G2, and does not completely cover G3. The values used in this document are for S32G2, they may differ for S32G3.
- This document's focus is on the Macronix serial NOR flash (MX25UW51245G) used in NXP evaluation platforms (S32G_VNP_RDB2).

# 3. QuadSPI configuration

## 3.1. Pin configuration

### 3.1.1. External signals

The external signals of the QuadSPI controller can be divided into five types according to a different function, as shown in the following table.

**Table 1. The external signals of the QuadSPI controller**

| Function | QSPI Signal | Pin Name | MSCR/IMCR |
|---|---|---|---|
| Chip Select | QSPI_CS_A0 | PG_04 | MSCR[100] = 0x00203021 |
| | QSPI_CS_A1 | PG_05 | MSCR[101] = 0x00203021 |
| | QSPI_CS_B0 | PD_00 | MSCR[48] = 0x00203002 |
| | QSPI_CS_B1 | PD_01 | MSCR[49] = 0x00203002 |
| Clock | QSPI_CK_A | PG_00 | MSCR[96] = 0x00200021 |

| Function | QSPI Signal | Pin Name | MSCR/IMCR |
|---|---|---|---|
| | QSPI_CK_A_b | PG_01 | MSCR[97] = 0x00200021 |
| | QSPI_CK_2A | PG_02 | MSCR[98] = 0x00200021 |
| | QSPI_CK_2A_b | PG_03 | MSCR[99] = 0x00200021 |
| | QSPI_CK_B | PD_06 | MSCR[54] = 0x00200002 |
| | QSPI_CK_B_b | PD_07 | MSCR[55] = 0x00200002 |
| Data I/O | QSPI_DATA_A_O[0:7] | PF_05 - PF12 | MSCR[85:92] = 0x00280021 |
| | QSPI_DATA_A_I[0:7] | | IMCR[28:35] = 0x00000002 |
| | QSPI_DATA_B_O[0:7] | | |
| | QSPI_DATA_B_I[0:7] | | |
| Data Strobe | QSPI_DQS_A_O | PF_13 | MSCR[93] = 0x00280021 |
| | QSPI_DQS_A_I | | IMCR[36] = 0x00000002 |
| | QSPI_DQS_B_O | | |
| | QSPI_DQS_B_I | | |
| Interrupt | QSPI_INTA_b | PF_14 | MSCR[94] = 0x00003020 |
| | | | IMCR[37] = 0x00000002 |
| | QSPI_INTB_b | | |

There are some recommendations for QuadSPI external signals:

- Chip select: Chip select is an active low signal, thus an external resistor (2K-10K) is needed to pull up a QuadSPI chip select signal to ensure the default value will be high.
- Clock: Connect to differential clock input for some Hyper-Flash devices, if needed (like S26KS512S2).
- Interrupt: Connect to flash ECC error-out signal, if available.

### 3.1.2. A pitfall for interrupt pin (PF_14)

Due to the limitation of IO resources, the user may want to use the QuadSPI controller interrupt pin (PF_14) as a GPIO output pin.

When booting from external QuadSPI flash, BootROM sets INT# pin (PF_14) as the primary function (QSPI_INTA_b). The specific configuration are, MSCR[94] = 0x83020, IMCR[37]=0x02. Once PF_14 is set as QuadSPI A interrupt input and enable internal pull-up. The block diagram of the configuration is shown in the following figure.

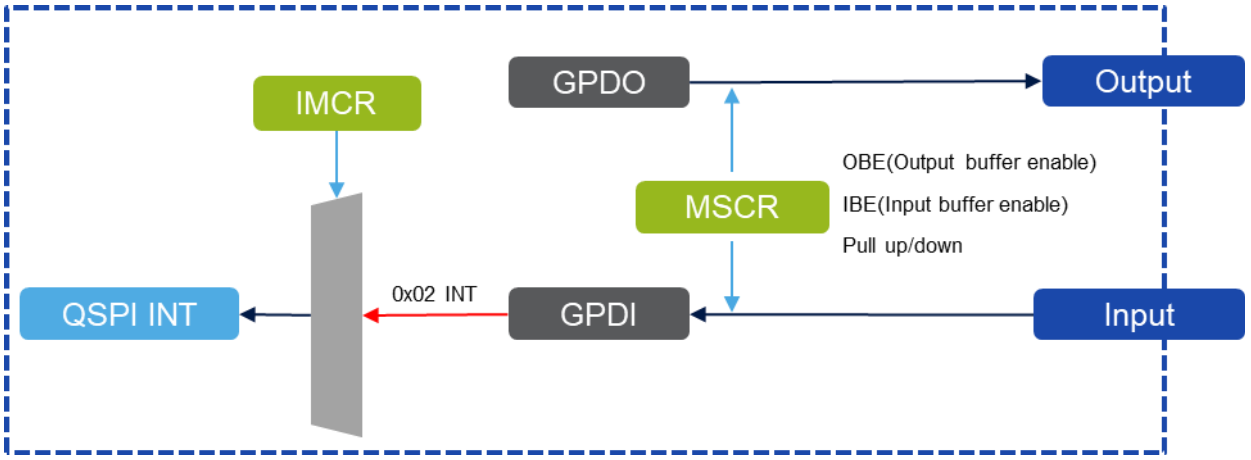


Figure 1. **Configurate PF_14 pin as the primary function**

There are some traps when trying to set this pin as a GPIO output:

- If IMCR setting is not changed, after MSCR is set to output first, GPDI will go to the low level, causing QSPI access failure.
- If external input is low level (such as directly grounded), it may cause QSPI boot failure.

There is a separate muxing control for this pin, it can be changed to a normal GPIO pin. IMCR[37] has three possible values, i.e.- 0x0, 0x1, and 0x2 which signifies “disable high”, “disable low” and “PF_14”. Since Interrupt pin to QSPI is active low, writing 0x0 on IMCR[37] or MSCR[549] to make it as “disable high”. Please note that MSCR[549] = IMCR[37]. MSCR[94] SSS value should be configured as 0 to select pin as GPIO.

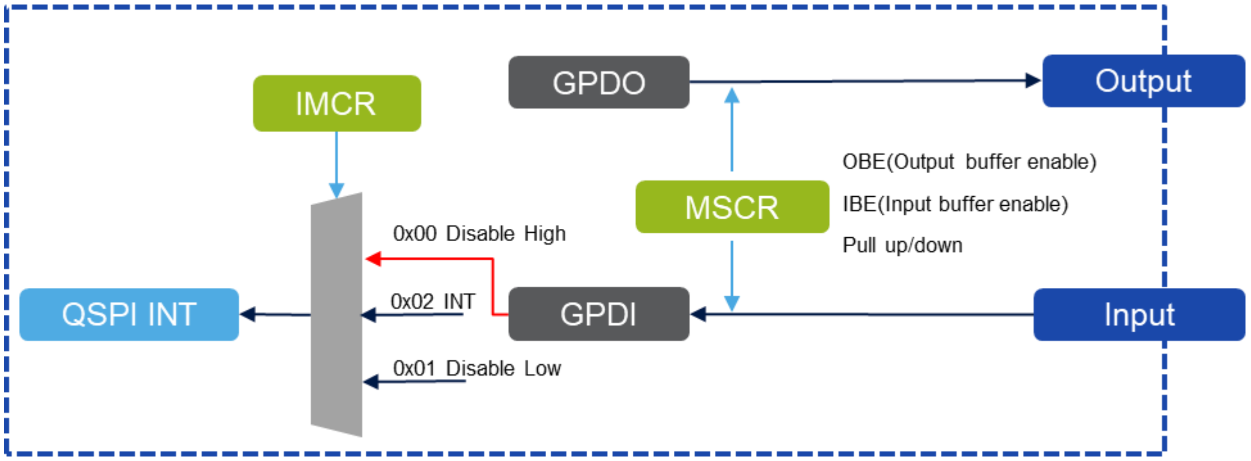


Figure 2. **Configurate PF_14 pin as GPIO output function**

Some points to remember for PF_14:

- Configure IMCR before MSCR in configuration sequence when setting this pin as a normal GPIO function

- Ensure that PF_14 remains high when booting from external flash, otherwise, it may cause boot failure

## 3.2. Clock configuration

The QuadSPI module can be divided into two clock domains, the SCLK clock domain and the Host clock domain as shown in the following figure.

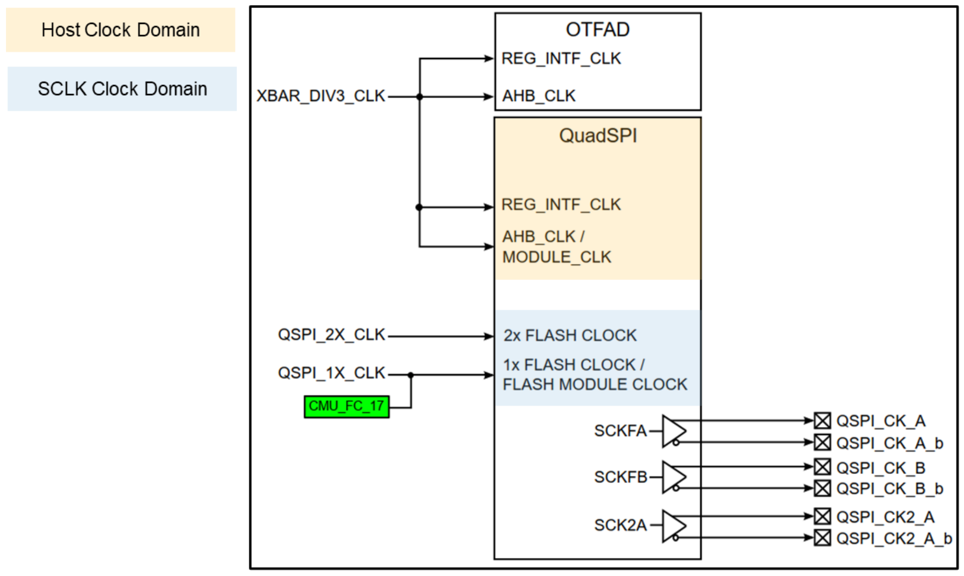


Figure 3. **The clock domains of the QuadSPI module**

The SCLK(Serial Flash Clock) clock domain is sourced from QSPI_1X_CLK/QSPI_2X_CLK. The Host clock domain is sourced from XBAR_DIV3_CLK.

### 3.2.1. Serial flash clock domain

The QSPI_1X_CLK and QSPI_2X_CLK are used for the protocol clock signals, that is serial flash clock.

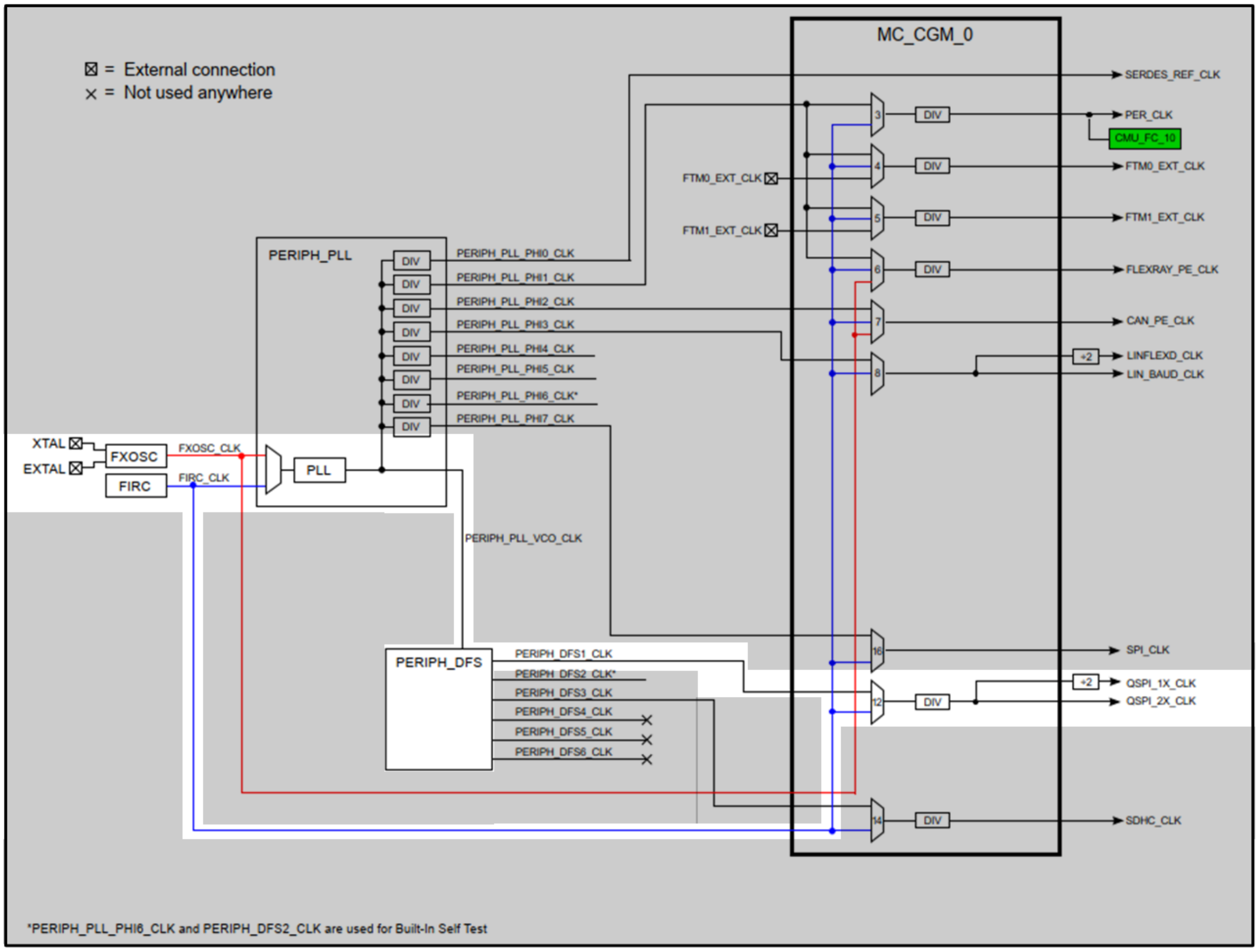


Figure 4. **The serial flash clock domain**

As shown in the above figure, the clock tree shows the paths of the serial flash clock (QSPI_1_CLK):

- FXOSC → PLL → PERIPH_FES → PERIPH_DFS1_CLK → MC_CGM_0_MUX_12 → QSPI_1/2_CLK
- FIRC→ MC_CGM_0_MUX_12 → QSPI_1/2_CLK
- FIRC→ PLL → PERIPH_FES → PERIPH_DFS1_CLK → MC_CGM_0_MUX_12 → QSPI_1/2_CLK

The path described in the first bullet is the recommended clock path, other two are not recommended to be used by application. The input clock source of PLL is selected by the REFCLKSEL field in PLLCLKMUX(PLL Clock Multiplexer). For example, PLLCLKMUX[REFCLKSEL] = 1 means select FXOSC(40 MHz) as PLL clock source.

The RDIV field in PLLDV(PLL Divider) sets the input clock divider for PLL. For example, PLL[RDIV] = 1 means the PLL input clock pre-divider is 1 and the MFI field in PPLDV specified establishes the multiplication factor applied to the reference frequency. For example, PLLDIV[MFI]=50, then the PLL VCO output is 40/1*50 = 2000 MHz.

DFS IP takes the input clock from PLL and generates multiple phases of the clock. Six independent phase dividers then use these phases. The following equation describes the relationship between the input and output clock of each phase divider:

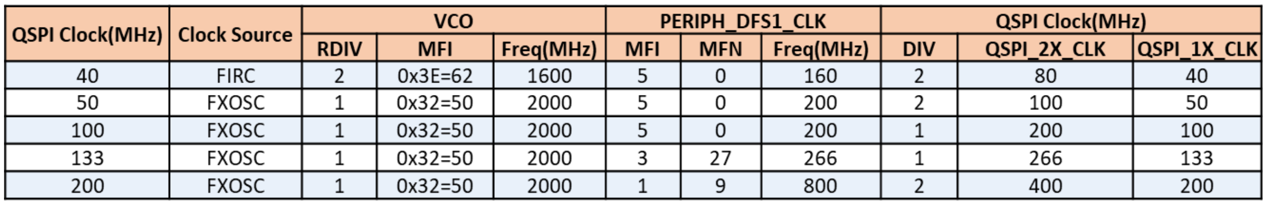
$$f_{\mathrm{dfsclkout}}n = \frac{f_{\mathrm{dfsclkin}}n}{2\times\left(\mathrm{DVPORTn[MFI]}+\frac{\mathrm{DVPORTn[MFN]}}{36}\right)}$$

For example, if DVPORT0[MFI]=1, DVPORT0[MFN]=9 in peripheral PLL, then the PERIPH_DFS1_CLK output is 2000/(2*(1+9/36))=2000/2.5 = 800 MHz

The QSPI_1/2_CLK clock is output by clock mux 12. The source clock for clock mux 12 is selected by the SELCTL field in MUX_12_CSC(Clock Mux 12 Select Control Register). For example, MUX_12_CSC[SELCTL] = 26 means select PERIPH_DFS1_CLK as clock mux 12 clock source.

If the DE field in MUX_12_DC_0 (Clock Mux 12 Divider 0 Control Register) is enabled, then the frequency of QSPI_2_CLK is divided by the output of clock mux 12. For  example, MUX_12_DC_0[DE]=1 and MUX_12_DC_0[DIV]=1, the QSPI_2_CLK should be 800 MHz/2 = 400 MHz, that is serial flash clock (QSPI_1_CLK) is 400 MHz/2 = 200 MHz.

The following screenshot shows the typical values of the related registers mentioned above when configuring some common QuadSPI clocks.



| QSPI Clock(MHz) | Clock Source | VCO | | | PERIPH_DFS1_CLK | | | QSPI Clock(MHz) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | RDIV | MFI | Freq(MHz) | MFI | MFN | Freq(MHz) | DIV | QSPI_2X_CLK | QSPI_1X_CLK |
| 40 | FIRC | 2 | 0x3E=62 | 1600 | 5 | 0 | 160 | 2 | 80 | 40 |
| 50 | FXOSC | 1 | 0x32=50 | 2000 | 5 | 0 | 200 | 2 | 100 | 50 |
| 100 | FXOSC | 1 | 0x32=50 | 2000 | 5 | 0 | 200 | 1 | 200 | 100 |
| 133 | FXOSC | 1 | 0x32=50 | 2000 | 3 | 27 | 266 | 1 | 266 | 133 |
| 200 | FXOSC | 1 | 0x32=50 | 2000 | 1 | 9 | 800 | 2 | 400 | 200 |

Figure 5. **Typical serial flash clock value settings**

**NOTE**

In the above table the FXOSC is assumed to be 40 MHz.

## 3.2.2. **Host clock domain**

As can be seen in Figure 2, the REG_INTF_CLK, AHB_CLK, and MODULE_CLK are sourced from the XBAR_DIV3_CLK signal.

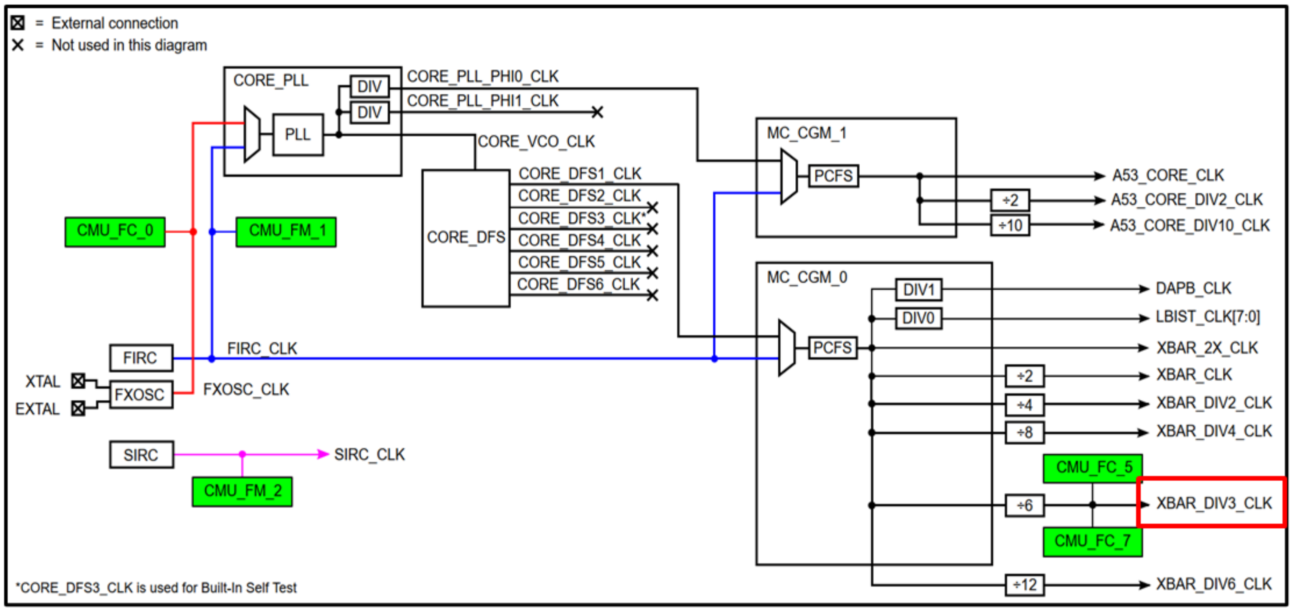


Figure 6. **The XBAR_DIV3_CLK clock**

The frequency of XBAR_DIV3_CLK is imposed by other system requirements and is already configured, the value is 133 MHz.

### 3.2.3. QuadSPI clock limitation

The maximum protocol frequencies can be found in the device datasheet. There are some limitations for the QuadSPI clock:

- The ratio of bus clock period to flash clock period must not be less than 1:5.
    - This means SCLK freq/Bus CLK freq > 1/5, thus SCLK freq > 26.67M
- Derive DDR octal writes using the following equation:
    - (3 x flash clock period) + (3 x bus clock period) < (8 x flash clock period)
    - This means SCLK freq/Bus clk freq < 5/3, thus SCLK freq < 222.167M

## 3.3. Register configuration

Before accessing the external flash device via the QuadSPI controller, we need to configure the QuadSPI controller correctly. Below are some general configuration steps:

- Configure flash memory address
    - Serial flash memory top address
    - Serial flash device address & address configuration
- Configure DQS sampling method
- Configure DLL and DQS delay chain

- Configure the data input hold requirement of external flash memory
- Configure the LUTs with QuadSPI command sequences
- Configure AHB access

**NOTE**

For each register field, the write access conditions are specified in the detailed resister description, please refer to S32G RM for more information.

## 3.3.1. Flash memory address

### 3.3.1.1. Serial flash memory top address

QuadSPI can support single mode, serial mode, and parallel mode for the supported flash memory ports A and B. The three modes are shown in the following figures.

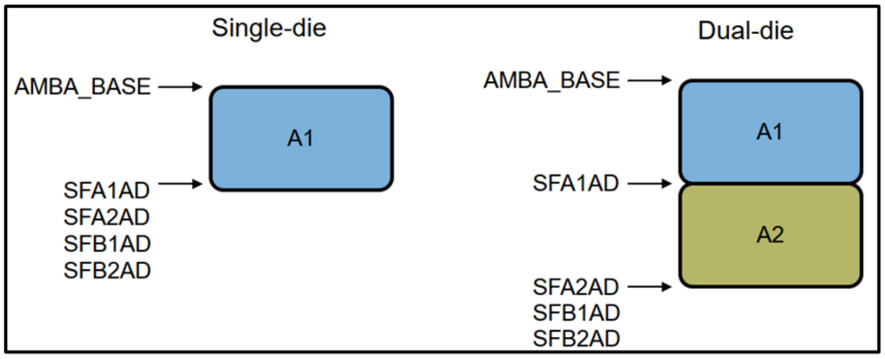


Figure 7. **Single mode for single-die and dual-die**

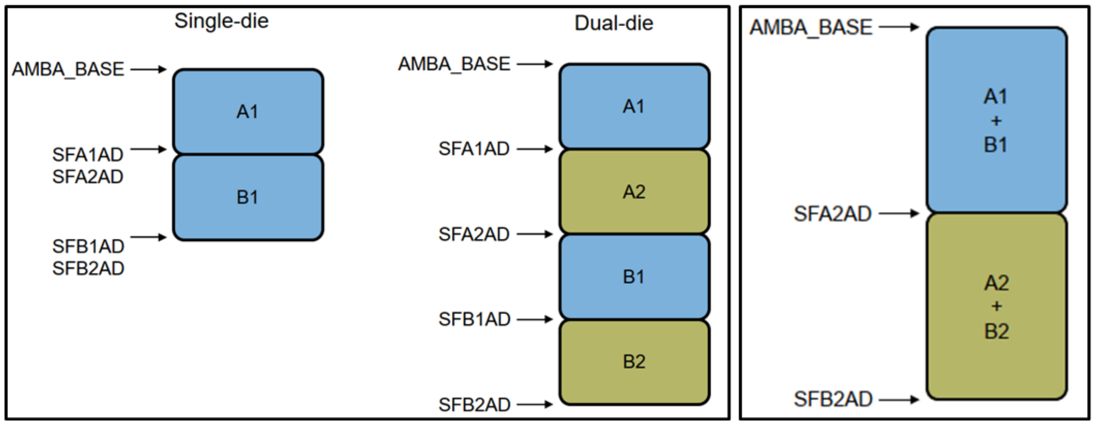


Figure 8. **Serial mode and parallel mode**

The size of the flash memory devices is mapped with the system memory space based on the configurations of the SFA1AD, SFA2AD, SFB1AD, and SFB2AD registers. Take 64 MB MX25UW51245 Nor flash device as an example in single mode. The settings are as follows:

SFA1AD = SFA2AD = SFB1AD = SFB2AD = 0x4000000

#### 3.3.1.2. Serial flash device address

**SFAR**

When sending an address to access external flash, a dedicated register SFAR (Serial Flash Memory Address Register) is required. The address must be within the external flash range of the system memory map (0x0000_0000 – 0x1FFF_FFFF). The QuadSPI module automatically translates this address on the memory map to the flash memory. If an SFM command does not need any address, then SFAR should be set to the QuadSPI_AMBA_BASE (0x0000_0000).

**SFACR**

SFACR (Serial Flash Address Configuration Register) is used to configure the address requirements that are specific to serial flash memory. Define the width of the column address in the SFACR[CAS] field.

If no column address is required, SFACR[CAS] must be 0. In 24-bit mode, when SFACR[CAS] is 3, then bits 3-26 are sent to the flash memory as its page address, and bits 2-0 are sent as its column address. The total number of address bits requested by the flash memory, as its page and column address, must not be more than 32 bits.

Indicate if the memory is word or byte-addressable in the SFACR[WA] field. If SFACE[WA] is 1 and the incoming address is 2004h, the controller re-maps this address to access the flash memory location 1002h.

### 3.3.2. Supported DQS sampling method

There are two supported DQS sampling methods, pad loopback and external DQS. The sampling method is set by the MCR[DQS_FA/B_SEL] field.

- Pad Loopback: The internal clock is loop-backed from the dummy internal pad to compensate for data pad delays and used as the sampling strobe signal. Pad Loopback is recommended for SDR mode up to 133 MHz  and DDR mode up to 66 MHz.
- External DQS: The data strobe signal(DQS/RWDS) is an output from the flash memory device that indicates when data is being transferred from the flash memory to the host controller. External DQS is recommended for DDR mode higher than 66 MHz. The data strobe signal (DQS/RWDS) is used to sample the read data. Both DQS and the data sent by the flash memory move in the same direction, therefore, it is relatively easier to achieve at higher frequencies.

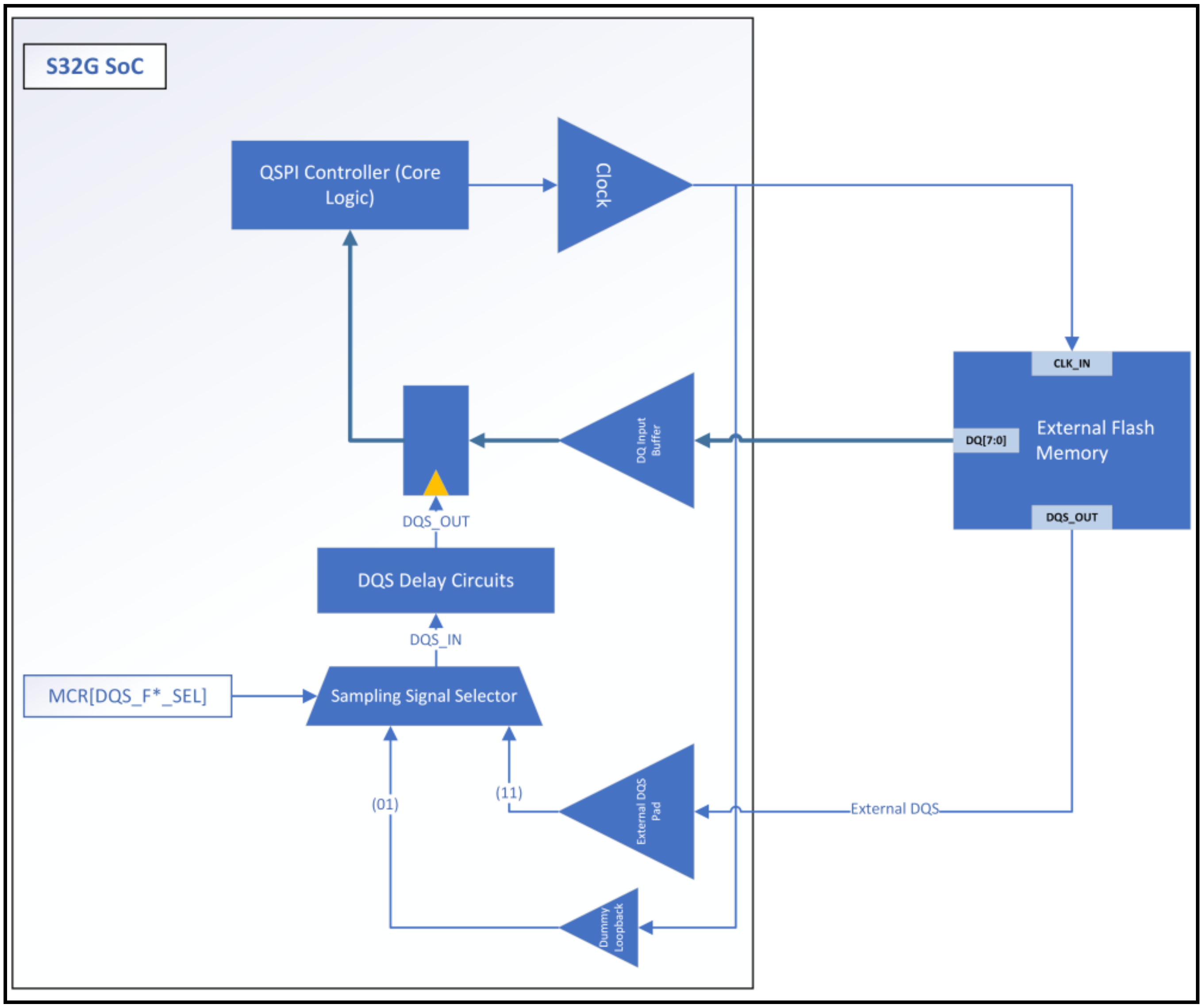


Figure 9. **Sampling signal generation**

## 3.3.3. DLL and DQS delay chain

The DLL is a general-purpose, dynamically adaptive clock delay module. It provides the ability to select a quantized delay (infractions of the clock period) regardless of on-chip variations such as process, voltage, and temperature (PVT).

The ‘DQS delay circuits’ are used to delay the selected signal and generate the actual strobe signal used to sample the incoming read data from external QSPI Flash.

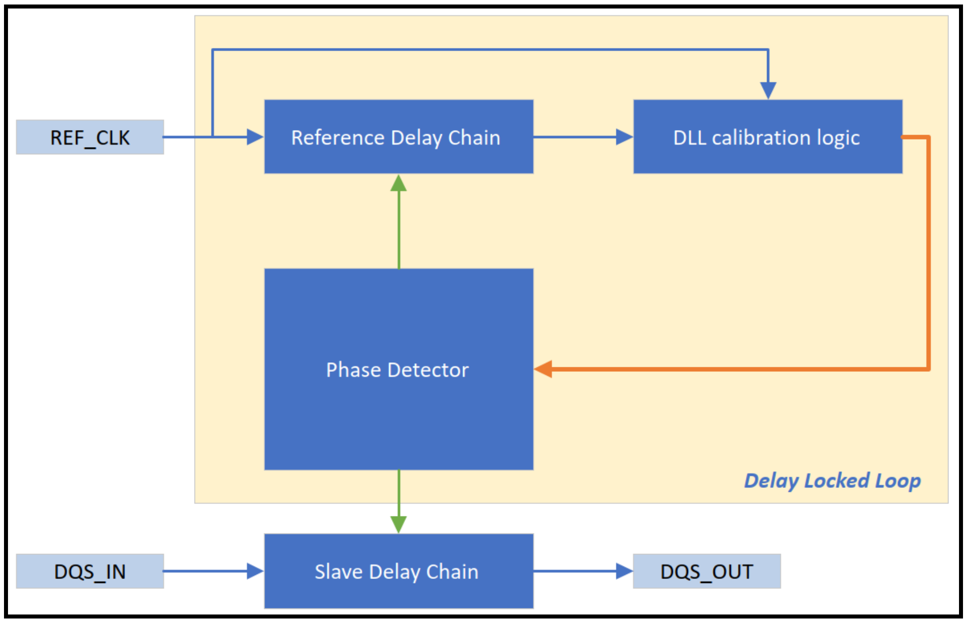


Figure 10. **The DQS delay circuits**

The DLL bypass mode and DLL auto update mode are supported in S32G, the specific programming sequence for the corresponding mode is as follows:

- DLL Bypass Mode: The DLL is disabled in this mode. The slave delay chain is directly configured using register fields DLLCR*[SLV_DLY_COARSE] and DLLCR*[SLV_DLY_OFFSET]. This is a static configuration (performed by application software or QSPI device driver). It is recommended to use in SDR mode.

  DLL bypass mode setting steps:

  1. Program DLLCRA[SLV_EN]=1, DLLCRA[SLV_DLL_BYPASS]=1, and DLLCRA[SLAVE_AUTO_UPDT]=0.
  2. Program the following fields to provide the desired DQS delay for sampling, DLLCRA[SLV_FINE_OFFSET], DLLCRA[SLV_DLY_COARSE], and DLLCR[FREQEN]. See the chip-specific QuadSPI information for the supported programming settings.
  3. Program DLLCRA[SLV_UPD]=1 to load these values in the slave delay chain.
  4. Check the slave delay chain update status by polling DLLSR[SLVA_LOCK]=1 and clear DLLCRA[SLV_UPD] after confirming the update state.

- DLL Auto Update mode: The DLL is enabled in this mode. Performs the ‘search’ to identify the right delay chain settings to generate T/16 (and multiples of T/16) delay using the slave delay chain. The slave delay chain is automatically reconfigured every time the DLL regains the lock condition. No application S/W intervention is required, once the DLL is configured initially. It is recommended to be used in DDR mode. Steps to step up the DLL auto-update mode setting:

  1. Program DLLCRA[SLV_EN]=1, DLLCRA[SLV_DLL_BYPASS]=0, and DLLCRA[SLAVE_AUTO_UPDT]=1.

2. Program the DLL configuration by using DLLCRA[DLL_REFCNTR] and DLLCRA[DLLRES]. See the chip-specific QuadSPI information for the supported DLL configuration settings.
3. Program the slave settings to delay DQS by using the fields, DLLCRA[SLV_FINE_OFFSET], DLLCRA[SLV_DLY_OFFSET], and DLLCR[FREQEN]. See the chip-specific QuadSPI information for the supported settings.
4. If offset delay needs to be updated on the slave chain, program DLLCRA[SLV_UPD]=1.
5. Enable DLL by programming DLLCRA[DLLEN]=1 and reset DLLCRA[SLV_UPD]=0. The slave delay chain is updated automatically and can be checked by polling DLLSR[SLVA_LOCK]==1

## 3.3.4. Data input hold requirement of flash memory

The Flash memory configuration register(FLSHCR) contains the timings that are specific to the flash memory device.

In the SDR mode:

MCR[DDR_EN] = 0, FLSHCR[TDH] = 0, FLSHCR[TCSS] = 3, FLSHCR[TCSH] = 3

In the DDR mode:

MCR[DDR_EN] = 1, FLSHCR[TDH] = 1, FLSHCR[TCSS] = 3, FLSHCR[TCSH] = 3

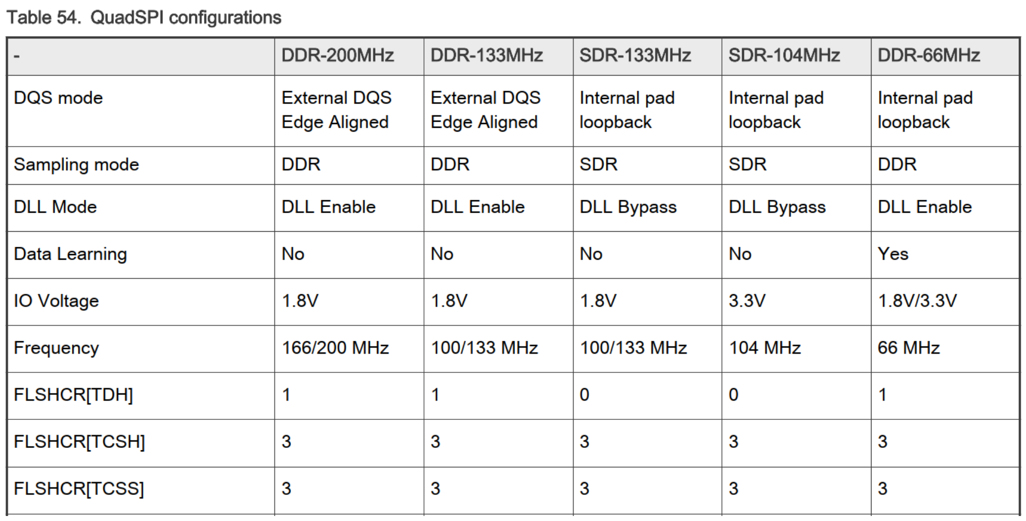


Table 54. QuadSPI configurations

| - | DDR-200MHz | DDR-133MHz | SDR-133MHz | SDR-104MHz | DDR-66MHz |
|---|---|---|---|---|---|
| DQS mode | External DQS Edge Aligned | External DQS Edge Aligned | Internal pad loopback | Internal pad loopback | Internal pad loopback |
| Sampling mode | DDR | DDR | SDR | SDR | DDR |
| DLL Mode | DLL Enable | DLL Enable | DLL Bypass | DLL Bypass | DLL Enable |
| Data Learning | No | No | No | No | Yes |
| IO Voltage | 1.8V | 1.8V | 1.8V | 3.3V | 1.8V/3.3V |
| Frequency | 166/200 MHz | 100/133 MHz | 100/133 MHz | 104 MHz | 66 MHz |
| FLSHCR[TDH] | 1 | 1 | 0 | 0 | 1 |
| FLSHCR[TCSH] | 3 | 3 | 3 | 3 | 3 |
| FLSHCR[TCSS] | 3 | 3 | 3 | 3 | 3 |

Figure 11. **QuadSPI configurations**

## 3.3.5. Look Up Table(LUT)

The sequences for a particular external QuadSPI flash are stored in the LUT. The LUT consists of up to 16 possible sequences that can be programmed in the LUT, and up to 10 instructions-operand pairs in one sequence. Writing the IPCR[SEQID] field with a LUT index will trigger execution.

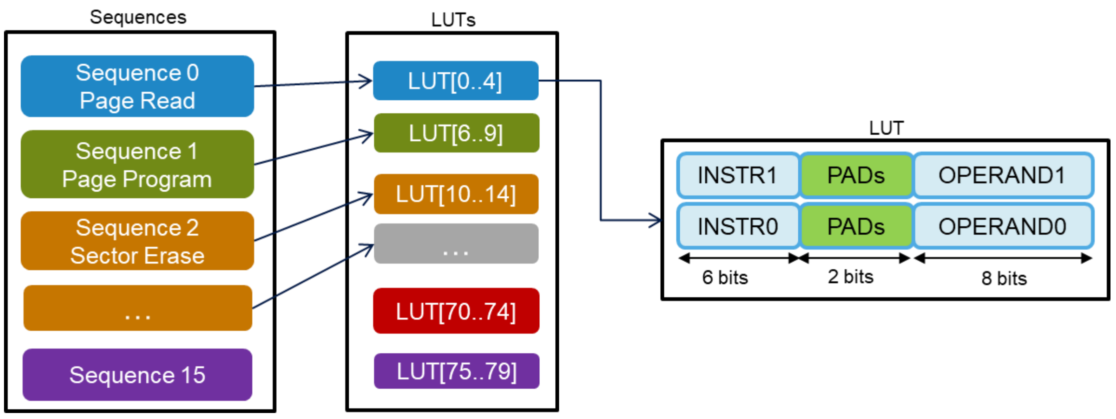
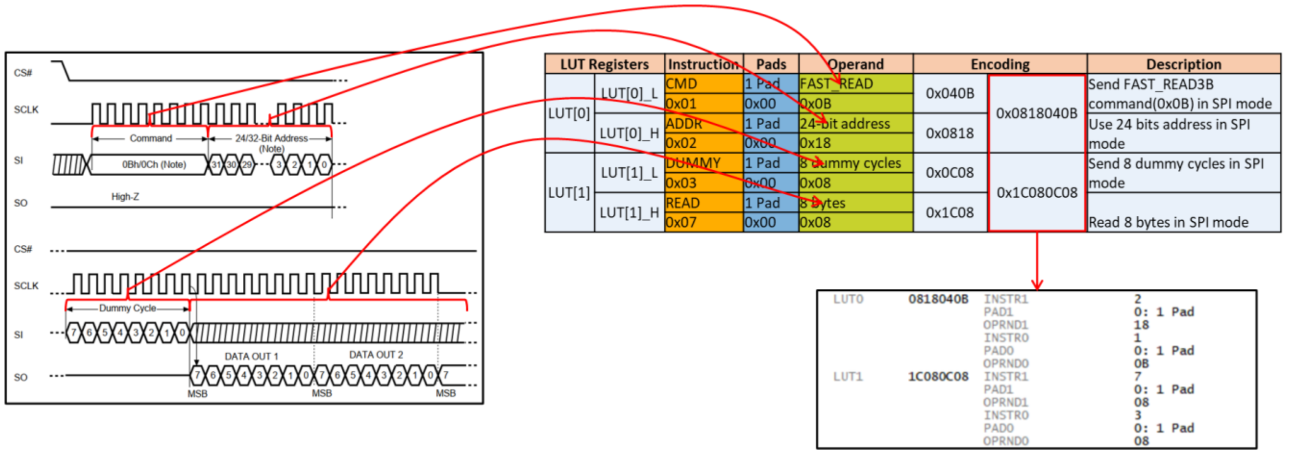


Figure 12. **The LUTs**

### 3.3.5.1. Example to configure the LUT

Transform the FAST_READ3B sequence into LUT sequence:

1. Command(0x0B)
2. Address bits(0x18)
3. Dummy cycles(0x08)
4. Read data bytes(0x08)



| LUT Registers | | Instruction | Pads | Operand | Encoding | | Description |
|---|---|---|---|---|---|---|---|
| LUT[0] | LUT[0]_L | CMD | 1 Pad | FAST_READ | 0x040B | 0x0818040B | Send FAST_READ3B command(0x0B) in SPI mode |
| | | 0x01 | 0x00 | 0x0B | | | |
| | LUT[0]_H | ADDR | 1 Pad | 24-bit address | 0x0818 | | Use 24 bits address in SPI mode |
| | | 0x02 | 0x00 | 0x18 | | | |
| LUT[1] | LUT[1]_L | DUMMY | 1 Pad | 8 dummy cycles | 0x0C08 | 0x1C080C08 | Send 8 dummy cycles in SPI mode |
| | | 0x03 | 0x00 | 0x08 | | | |
| | LUT[1]_H | READ | 1 Pad | 8 bytes | 0x1C08 | | Read 8 bytes in SPI mode |
| | | 0x07 | 0x00 | 0x08 | | | |

```
LUT0    0818040B  INSTR1       2
                  PAD1         0: 1 Pad
                  OPRND1       18
                  INSTR0       1
                  PAD0         0: 1 Pad
                  OPRND0       0B
LUT1    1C080C08  INSTR1       7
                  PAD1         0: 1 Pad
                  OPRND1       08
                  INSTR0       3
                  PAD0         0: 1 Pad
                  OPRND0       08
```

Figure 13. **Transform the FAST_READ3B sequence into LUT sequence**

## 3.3.6. Peripheral bus access

The peripheral bus allows the software to write and read QSPI module registers including RX and TX buffers. Read, write and erase can be implemented over the peripheral bus.

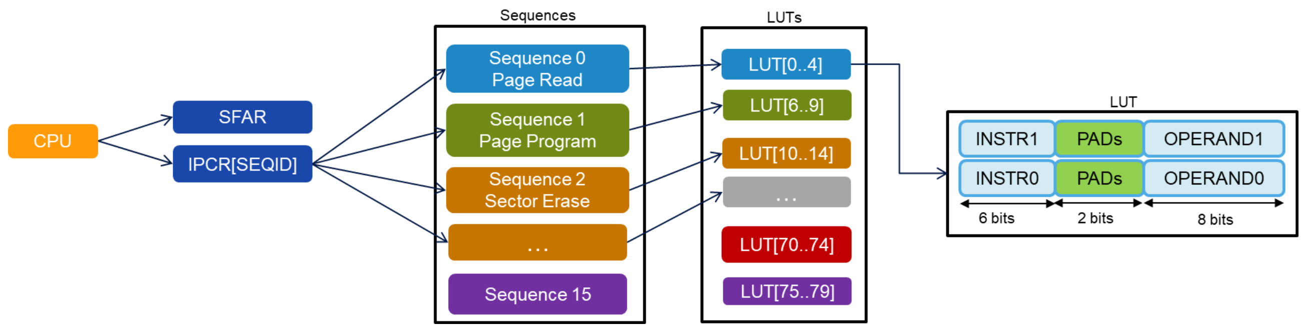


Figure 14. **Peripheral bus access**

## 3.3.7. AHB bus access

AHB bus allows AHB masters to read external QSPI flash as the system memory. External flash memory can be mapped to system memory at the address range, 0x0000_0000 – 0x1FFF_FFFF. The SEQID field in the Buffer Generic Configuration Register (BFGENCR) must be configured with the ID of one of the sequences defined in LUTs. By default, BFGEBCR[SEQID] is set to 0, it points to the default read sequence(the basic read command is 0x03).

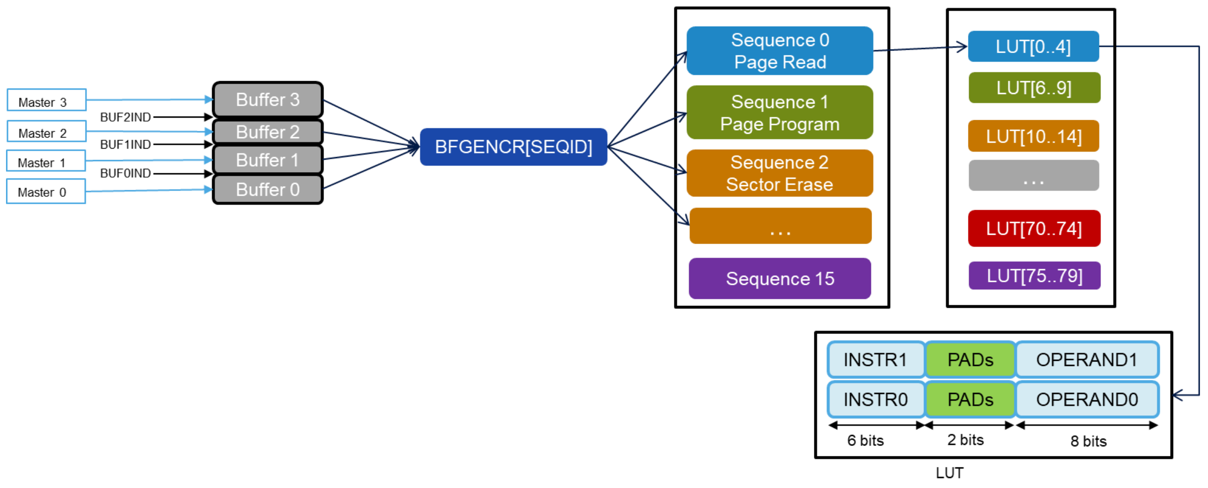
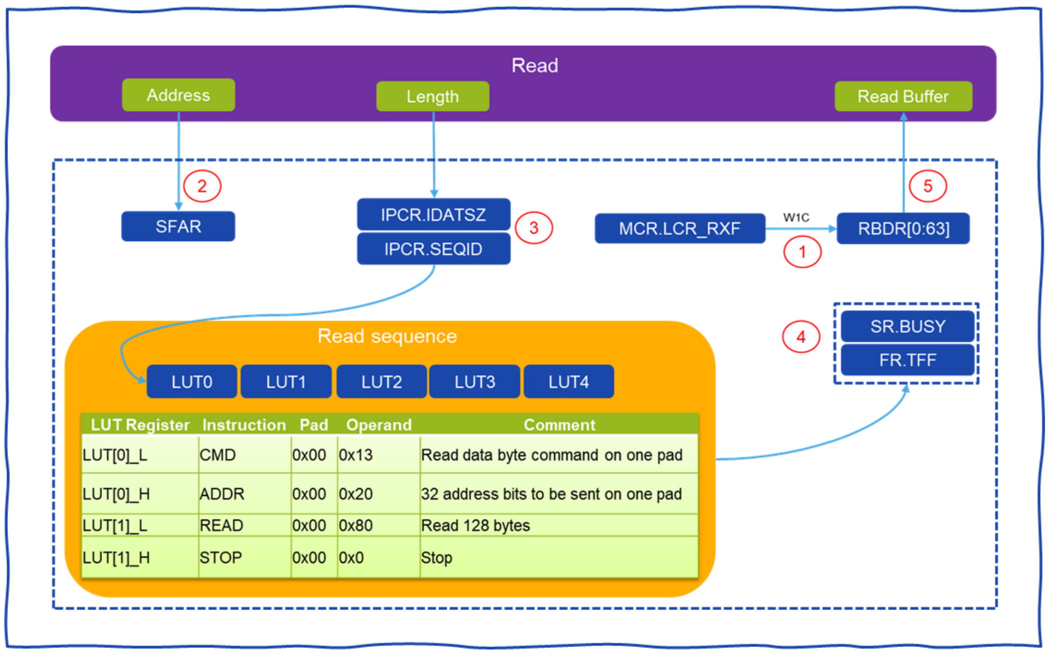


Figure 15. **AHB bus access**

## 3.3.8. Read the external flash device

### 3.3.8.1. Read – IP command

The steps to generate a read sequence via peripheral bus is shown in the following figure.



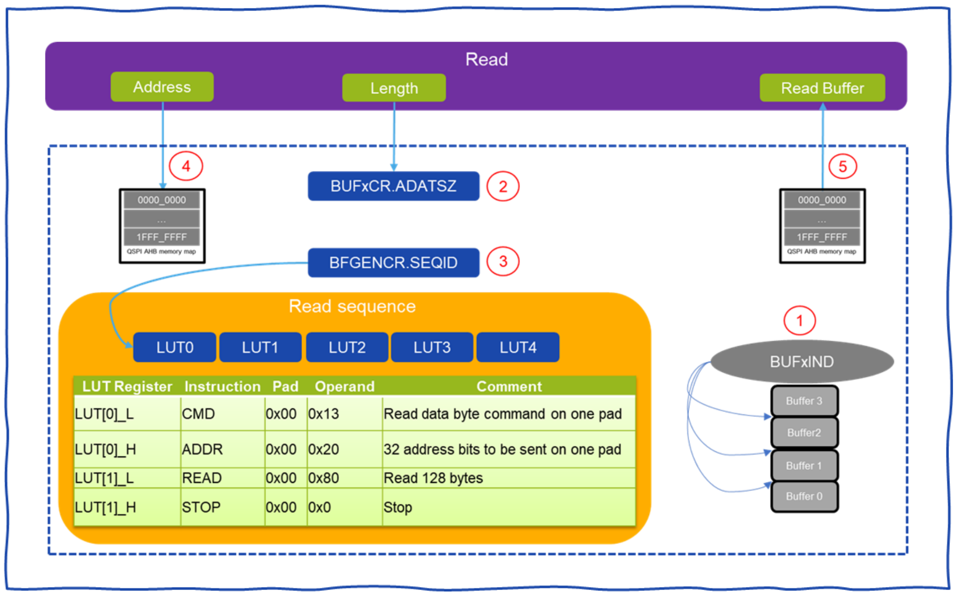
| LUT Register | Instruction | Pad | Operand | Comment |
|---|---|---|---|---|
| LUT[0]_L | CMD | 0x00 | 0x13 | Read data byte command on one pad |
| LUT[0]_H | ADDR | 0x00 | 0x20 | 32 address bits to be sent on one pad |
| LUT[1]_L | READ | 0x00 | 0x80 | Read 128 bytes |
| LUT[1]_H | STOP | 0x00 | 0x0 | Stop |

Figure 16. **Read - IP command**

1. Write 1 to clear RX buffer in MCR[LCR_RXF] field
2. Write the address that needs to be accessed in SFAR
3. Write the length of data to be read in IPCR[IDASZ], write the IPCR[SEQID] to trigger the read sequence
4. Check SR[BUSY] to wait for the module is idle, and check FR[TFF] to wait for IP command transaction to be finished
5. Read data from RBDR

### 3.3.8.2. **Read – AHB command**

The steps to generate a read sequence via AHB bus is shown in the following figure.



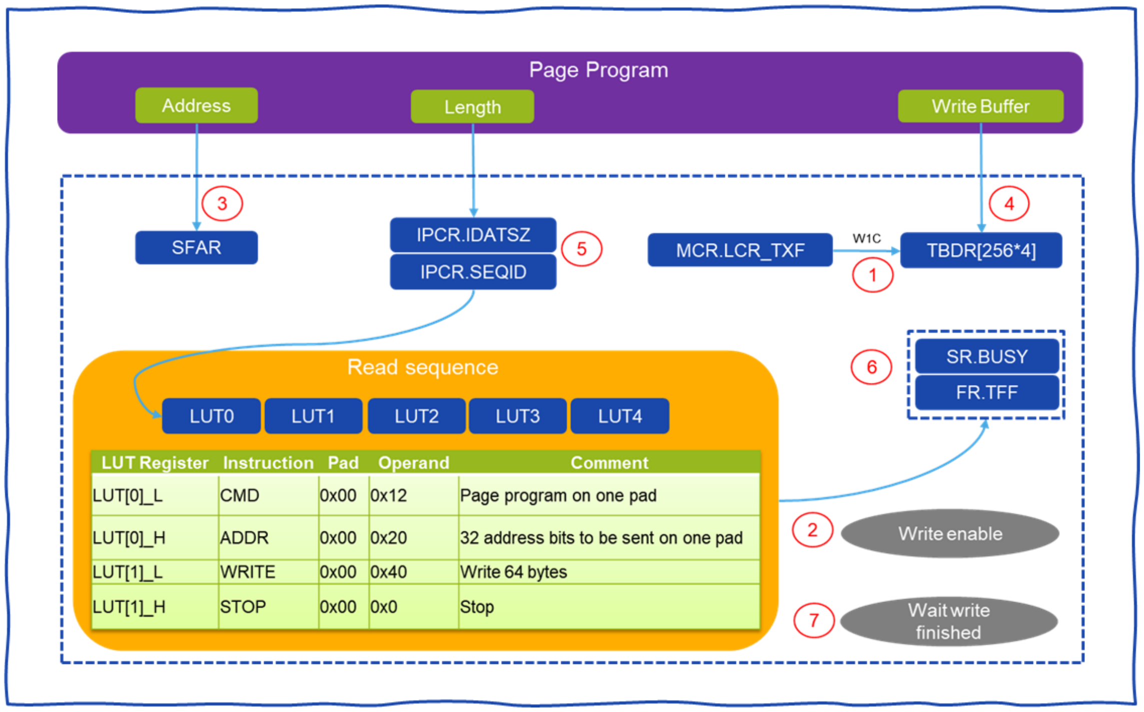
| LUT Register | Instruction | Pad | Operand | Comment |
|---|---|---|---|---|
| LUT[0]_L | CMD | 0x00 | 0x13 | Read data byte command on one pad |
| LUT[0]_H | ADDR | 0x00 | 0x20 | 32 address bits to be sent on one pad |
| LUT[1]_L | READ | 0x00 | 0x80 | Read 128 bytes |
| LUT[1]_H | STOP | 0x00 | 0x0 | Stop |

Figure 17. **Read – AHB command**

1. Configure flexible read AHB buffer size in BUFxIND
2. Typically, BUF0IND=BUF1IND=BUF2IND = 0, which means the size of buffer0, buffer1, and buffer2 is 0. The buffer3 is 1024 bytes.
3. Configure for any read access routed to buffer0 ~ buffer3 in BUFxCR
4. Optionally, buffer3 may be configured as an “all master” buffer by writing 1 to BUF3CR[ALLMST]
5. Set the amount of data to be fetched from the flash memory on every missed access in BUFxCR[ADATSZ] field
6. Configure the correct sequence ID in the BFGEBCR[SEQID] field
7. Choose a start address for reading in the memory mapped area
8. Read data from the memory mapped area directly

### 3.3.9. **Write the external flash device**

The steps to generate a write sequence via peripheral bus is shown in the following figure.



| LUT Register | Instruction | Pad | Operand | Comment |
|---|---|---|---|---|
| LUT[0]_L | CMD | 0x00 | 0x12 | Page program on one pad |
| LUT[0]_H | ADDR | 0x00 | 0x20 | 32 address bits to be sent on one pad |
| LUT[1]_L | WRITE | 0x00 | 0x40 | Write 64 bytes |
| LUT[1]_H | STOP | 0x00 | 0x0 | Stop |

Figure 18. **Write the external flash**

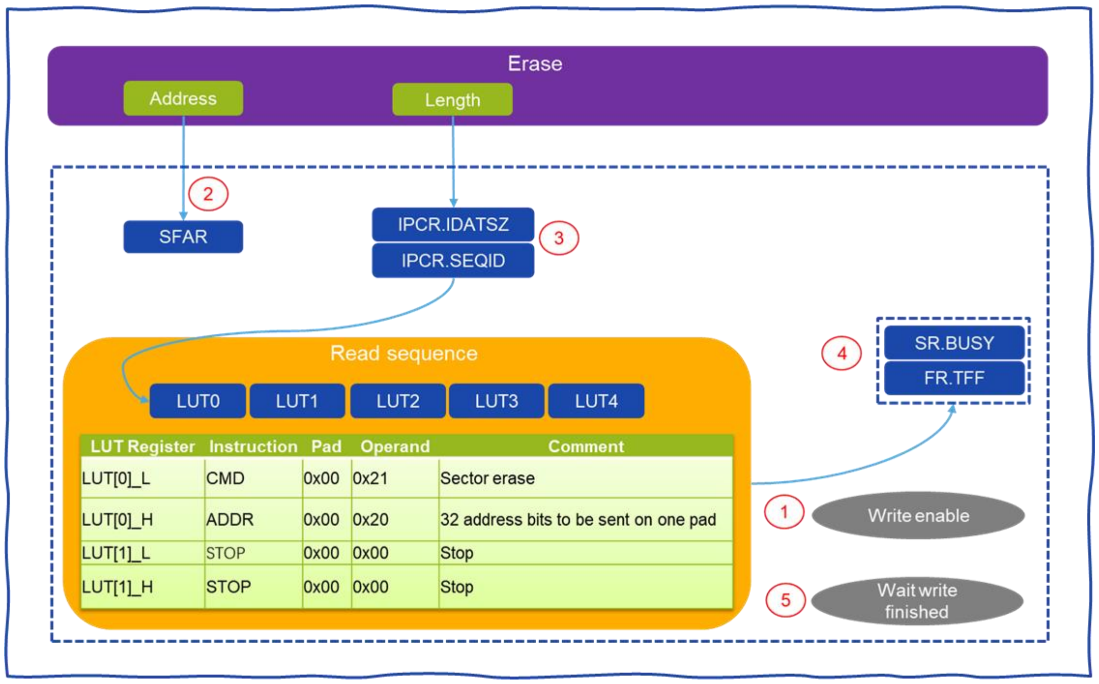
1. Write 1 to clear TX buffer in MCR[LCR_TXF] field
2. Write Enable
   a) Send the Write Enable(WREN) command to external flash memory to set Write Enable Latch(WEL) bit in its status register
3. Write the address that needs to be programmed in SFAR
4. Write the program data into TBDR
5. Write the length of data to be written in IPCR[IDASZ], write the IPCR[SEQID] to trigger the program sequence
6. Check SR[BUSY] to wait for the module is idle, and check FR[TFF] to wait for the IP command transaction finished
7. Wait to write finished
   a) Wait the Write In Progress(WIP) bit become 0

**NOTE**

AHB write is supported on S32G but with restrictions, mainly AHB writes use case is for Hyperram, which is not included in this document.

### 3.3.10. Sector erase

The steps to generate a sector erase sequence via peripheral bus is shown in the following figure.



| LUT Register | Instruction | Pad | Operand | Comment |
|---|---|---|---|---|
| LUT[0]_L | CMD | 0x00 | 0x21 | Sector erase |
| LUT[0]_H | ADDR | 0x00 | 0x20 | 32 address bits to be sent on one pad |
| LUT[1]_L | STOP | 0x00 | 0x00 | Stop |
| LUT[1]_H | STOP | 0x00 | 0x00 | Stop |

Figure 19. **Sector erase**

1. Write Enable
   a) Send the Write Enable(WREN) command to external flash memory to set Write Enable Latch(WEL) bit in its status register
2. Write the start address that needs to be erased in SFAR
3. Write the length of data to be erased in IPCR[IDASZ], write the IPCR[SEQID] to trigger the sector erase sequence
4. Check SR[BUSY] to wait for the module is idle, and check FR[TFF] to wait for the IP command transaction finished
5. Wait to write finished
   a) Wait the Write In Progress(WIP) bit become 0

# 4. QuadSPI Boot

When choosing boot from an external NOR flash device, BootROM will perform QuadSPI controller configuration. For different flash device types, BootROM adopts different configuration methods.

## 4.1. BootROM supported QuadSPI flashes

For Non-Hyper flash, like Quad and Octal flash devices that support with 1-bit mode, BootROM will set QuadSPI controller in 1-bit SDR mode. For those devices, BootROM can perform QuadSPI controller configuration in two phases:

**Initial configuration phase:** The initial configuration is used to read the reconfiguration data from the address 0x200. BootROM sets 40 MHz (For S32G3 it is set as 30 MHz) in the 1-bit mode for Quad and Octal flash memories at this phase. If the reconfiguration data is not present, BootROM firmware continues the booting process with the initial QuadSPI configuration.

**Final Configuration phase:** BootROM reconfigures the QuadSPI controller based on details provided in the QuadSPI reconfiguration data, and prepare the clock source per required frequency in reconfiguration data. It thens, configure the external Flash device for the intended mode of operation, like SPI mode to OPI mode.

BootROM supports only reading via an AHB interface, it does not support any write operations to QuadSPI flash memory. Some details configurations can be passed via RCON/Fuses, when booting from QuadSPI. After reset, the index 0 of the LUT[0..4] is programmed with a basic read sequence.

The flash devices (Single, Dual, Quad, or Octal), can start in 1-bit SDR mode, and the initial read command should be 0x03. Typically the read sequence of flash devices is shown in the following table:

Table 2. **Non-Hyper-flash initial read sequence**

| Instruction | Pad | Operand | Comment |
|---|---|---|---|
| CMD | 0x00 | 0x03 | Read data byte command on one pad |
| ADDR | 0x00 | 0x18 | 24 address bits to be sent on one pad |
| READ | 0x00 | 0x08 | Read 64 bits |
| JMP_ON_CS | 0x00 | 0x0 | Jump to instruction 0 (CMD) |

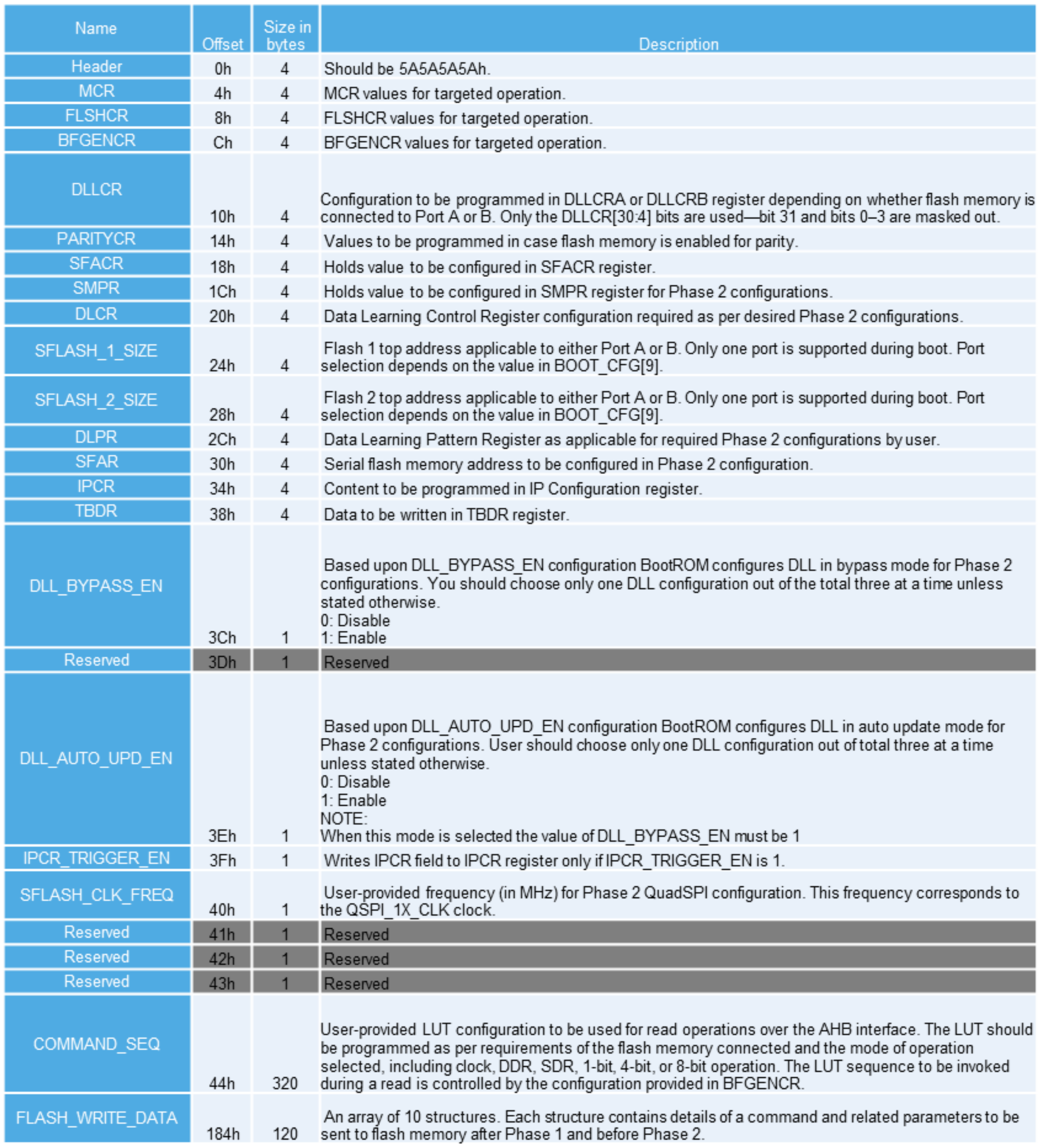
For Hyper flash devices, BootROM sets QuadSPI controller in 8-bit mode. And it should support the initial read command shown in the following table:

Table 3. **Hyper-flash initial read sequence**

| Instruction | Pad | Operand | Comment |
|---|---|---|---|
| CMD_DDR | 0x03 | 0xA0 | Read data byte command |
| ADDR_DDR | 0x03 | 0x18 | 3 Bytes address |
| CADDR_DDR | 0x03 | 0x10 | 2 Bytes column address |
| DUMMY | 0x03 | 0x0F | 16 dummy cycles |
| READ_DDR | 0x03 | 0x40 | Read 0x40 bytes data |
| STOP | 0x00 | 0x00 | Stop |

## 4.2. The reconfiguration parameter

For better performance, you may need to reconfigure flash memory using the reconfiguration parameters, after the initial configuration phase is over. BootROM supports a list of 10 commands (CMDs) to complete it. Each of these commands and associated data are encoded in 12 bytes. The structure of the reconfiguration parameter is shown in the following figure..



| Name | Offset | Size in bytes | Description |
|---|---|---|---|
| Header | 0h | 4 | Should be 5A5A5A5Ah. |
| MCR | 4h | 4 | MCR values for targeted operation. |
| FLSHCR | 8h | 4 | FLSHCR values for targeted operation. |
| BFGENCR | Ch | 4 | BFGENCR values for targeted operation. |
| DLLCR | 10h | 4 | Configuration to be programmed in DLLCRA or DLLCRB register depending on whether flash memory is connected to Port A or B. Only the DLLCR[30:4] bits are used—bit 31 and bits 0–3 are masked out. |
| PARITYCR | 14h | 4 | Values to be programmed in case flash memory is enabled for parity. |
| SFACR | 18h | 4 | Holds value to be configured in SFACR register. |
| SMPR | 1Ch | 4 | Holds value to be configured in SMPR register for Phase 2 configurations. |
| DLCR | 20h | 4 | Data Learning Control Register configuration required as per desired Phase 2 configurations. |
| SFLASH_1_SIZE | 24h | 4 | Flash 1 top address applicable to either Port A or B. Only one port is supported during boot. Port selection depends on the value in BOOT_CFG[9]. |
| SFLASH_2_SIZE | 28h | 4 | Flash 2 top address applicable to either Port A or B. Only one port is supported during boot. Port selection depends on the value in BOOT_CFG[9]. |
| DLPR | 2Ch | 4 | Data Learning Pattern Register as applicable for required Phase 2 configurations by user. |
| SFAR | 30h | 4 | Serial flash memory address to be configured in Phase 2 configuration. |
| IPCR | 34h | 4 | Content to be programmed in IP Configuration register. |
| TBDR | 38h | 4 | Data to be written in TBDR register. |
| DLL_BYPASS_EN | 3Ch | 1 | Based upon DLL_BYPASS_EN configuration BootROM configures DLL in bypass mode for Phase 2 configurations. You should choose only one DLL configuration out of the total three at a time unless stated otherwise.<br>0: Disable<br>1: Enable |
| Reserved | 3Dh | 1 | Reserved |
| DLL_AUTO_UPD_EN | 3Eh | 1 | Based upon DLL_AUTO_UPD_EN configuration BootROM configures DLL in auto update mode for Phase 2 configurations. User should choose only one DLL configuration out of total three at a time unless stated otherwise.<br>0: Disable<br>1: Enable<br>NOTE:<br>When this mode is selected the value of DLL_BYPASS_EN must be 1 |
| IPCR_TRIGGER_EN | 3Fh | 1 | Writes IPCR field to IPCR register only if IPCR_TRIGGER_EN is 1. |
| SFLASH_CLK_FREQ | 40h | 1 | User-provided frequency (in MHz) for Phase 2 QuadSPI configuration. This frequency corresponds to the QSPI_1X_CLK clock. |
| Reserved | 41h | 1 | Reserved |
| Reserved | 42h | 1 | Reserved |
| Reserved | 43h | 1 | Reserved |
| COMMAND_SEQ | 44h | 320 | User-provided LUT configuration to be used for read operations over the AHB interface. The LUT should be programmed as per requirements of the flash memory connected and the mode of operation selected, including clock, DDR, SDR, 1-bit, 4-bit, or 8-bit operation. The LUT sequence to be invoked during a read is controlled by the configuration provided in BFGENCR. |
| FLASH_WRITE_DATA | 184h | 120 | An array of 10 structures. Each structure contains details of a command and related parameters to be sent to flash memory after Phase 1 and before Phase 2. |

Figure 20. **The structure of the reconfiguration parameter**

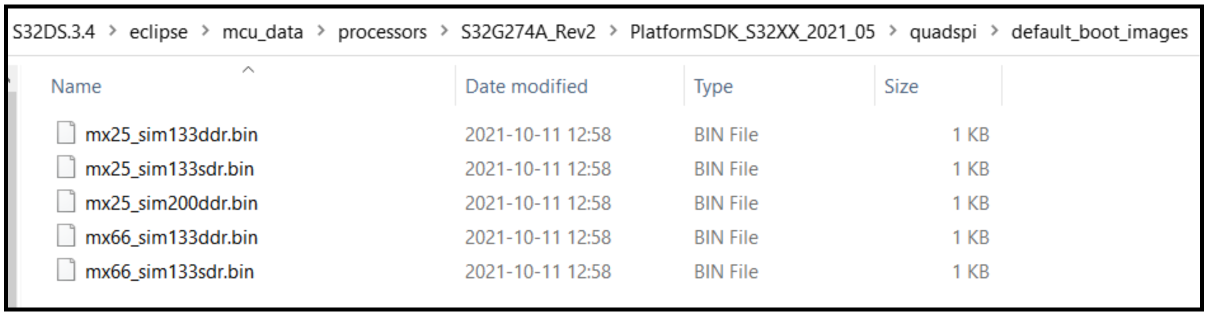
The reconstruction parameters includes the following three parts:

1. The first 17 32-bit words of the data are fields defining configuration register values, that the BootROM firmware uses to configure the QuadSPI controller
2. The LUTs as part of the reconfiguration data must contain at least one read sequence

3. The commands part: each of these commands and associated data are encoded in 3 words (12 bytes):
   a) Word 0(bytes 0-3): Command configuration
   b) Word 1(bytes 4-7): Flash configuration/Status Register address
   c) Word 2(bytes 11-8): Data to be sent to flash

The NXP provided reconfiguration images can be found at the below path.

C:\NXP\S32DS.3.4\eclipse\mcu_data\processors\S32G274A_Rev2\PlatformSDK_S32XX_2021_05\quadspi\default_boot_images



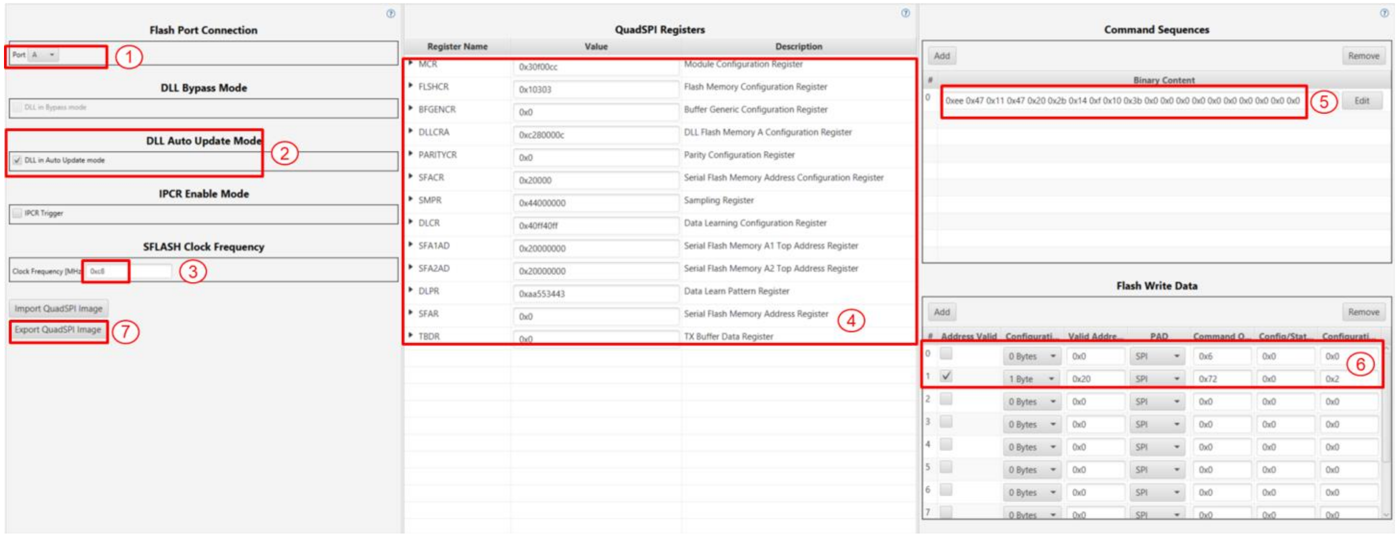
S32DS.3.4 › eclipse › mcu_data › processors › S32G274A_Rev2 › PlatformSDK_S32XX_2021_05 › quadspi › default_boot_images

| Name | Date modified | Type | Size |
|---|---|---|---|
| mx25_sim133ddr.bin | 2021-10-11 12:58 | BIN File | 1 KB |
| mx25_sim133sdr.bin | 2021-10-11 12:58 | BIN File | 1 KB |
| mx25_sim200ddr.bin | 2021-10-11 12:58 | BIN File | 1 KB |
| mx66_sim133ddr.bin | 2021-10-11 12:58 | BIN File | 1 KB |
| mx66_sim133sdr.bin | 2021-10-11 12:58 | BIN File | 1 KB |

Figure 21. **NXP provided reconfiguration images**

## 4.2.1. Generate reconfiguration image using S32 config tool

The reconfiguration image can be generated from the S32 config tool. After installing the S32DS, and S32G related plugins, you can open the QuadSPI reconfiguration image interface, which is shown in the following figure.



Flash Port Connection

Port: A

DLL Bypass Mode

☐ DLL in Bypass mode

DLL Auto Update Mode

☑ DLL in Auto Update mode

IPCR Enable Mode

☐ IPCR Trigger

SFLASH Clock Frequency

Clock Frequency [MHz]: 0xc8

QuadSPI Registers

| Register Name | Value | Description |
|---|---|---|
| MCR | 0x30f00cc | Module Configuration Register |
| FLSHCR | 0x10303 | Flash Memory Configuration Register |
| BFGENCR | 0x0 | Buffer Generic Configuration Register |
| DLLCRA | 0xc280000c | DLL Flash Memory A Configuration Register |
| PARITYCR | 0x0 | Parity Configuration Register |
| SFACR | 0x20000 | Serial Flash Memory Address Configuration Register |
| SMPR | 0x44000000 | Sampling Register |
| DLCR | 0x40ff40ff | Data Learning Configuration Register |
| SFA1AD | 0x20000000 | Serial Flash Memory A1 Top Address Register |
| SFA2AD | 0x20000000 | Serial Flash Memory A2 Top Address Register |
| DLPR | 0xaa553443 | Data Learn Pattern Register |
| SFAR | 0x0 | Serial Flash Memory Address Register |
| TBDR | 0x0 | TX Buffer Data Register |

Command Sequences

| # | Binary Content |
|---|---|
| 0 | 0xee 0x47 0x11 0x47 0x20 0x2b 0x14 0xf 0x10 0x3b 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 |

Flash Write Data

| # | Address Valid | Configurati... | Valid Addre... | PAD | Command O... | Config/Stat... | Configurati... |
|---|---|---|---|---|---|---|---|
| 0 | ☐ | 0 Bytes | 0x0 | SPI | 0x6 | 0x0 | 0x0 |
| 1 | ✓ | 1 Byte | 0x20 | SPI | 0x72 | 0x0 | 0x2 |
| 2 | ☐ | 0 Bytes | 0x0 | SPI | 0x0 | 0x0 | 0x0 |
| 3 | ☐ | 0 Bytes | 0x0 | SPI | 0x0 | 0x0 | 0x0 |
| 4 | ☐ | 0 Bytes | 0x0 | SPI | 0x0 | 0x0 | 0x0 |
| 5 | ☐ | 0 Bytes | 0x0 | SPI | 0x0 | 0x0 | 0x0 |
| 6 | ☐ | 0 Bytes | 0x0 | SPI | 0x0 | 0x0 | 0x0 |
| 7 | ☐ | 0 Bytes | 0x0 | SPI | 0x0 | 0x0 | 0x0 |

Figure 22. **Generate reconfiguration image using S32 config tool**

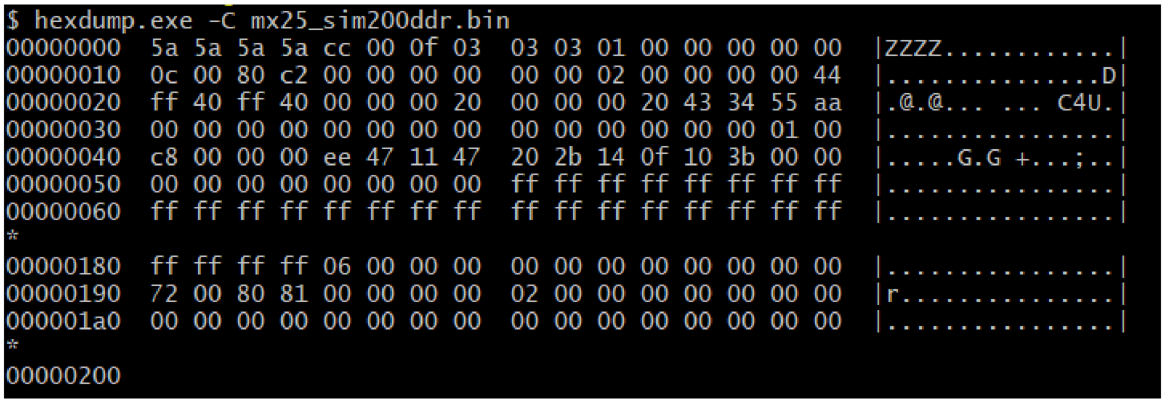
1. The flash port connection: QuadSPI module has two flash memory port. BootROM supports booting from both port A and port B.

2. Enable the DLL auto update mode.
3. Configure the serial flash clock frequency to 200 MHz.
4. Reconfigure the QuadSPI controller register value match with OPI-DTR mode.
5. Configure the OPI-DTR read sequence into LUTs
6. The flash write data, aim to switch flash device from SPI mode to DTR mode.
7. Export the reconfiguration parameter image.

### 4.2.2. Reconfiguration parameters analysis

The following figure shows the content of the QuadSPI reconfiguration parameters, that are exported from the S32 config tool.

The first four bytes (0x5a5a5a5a )represent the header of the reconfiguration parameters. The next 16 32-bit words of data represent the register values that the QSPI module needs to be reconfigured. Then there is the OPI-DTR read sequence which should be programmed into the LUTs and finally is the command part, the main purpose is to switch the external flash from single-wire SPI mode to 8-wire OPI mode.



```
$ hexdump.exe -C mx25_sim200ddr.bin
00000000  5a 5a 5a 5a cc 00 0f 03  03 03 01 00 00 00 00 00  |ZZZZ............|
00000010  0c 00 80 c2 00 00 00 00  00 00 02 00 00 00 00 44  |...............D|
00000020  ff 40 ff 40 00 00 00 20  00 00 00 20 43 34 55 aa  |.@.@... ... C4U.|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 01 00  |................|
00000040  c8 00 00 00 ee 47 11 47  20 2b 14 0f 10 3b 00 00  |.....G.G +...;..|
00000050  00 00 00 00 00 00 00 00  ff ff ff ff ff ff ff ff  |................|
00000060  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00000180  ff ff ff ff 06 00 00 00  00 00 00 00 00 00 00 00  |................|
00000190  72 00 80 81 00 00 00 00  02 00 00 00 00 00 00 00  |r...............|
000001a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000200
```

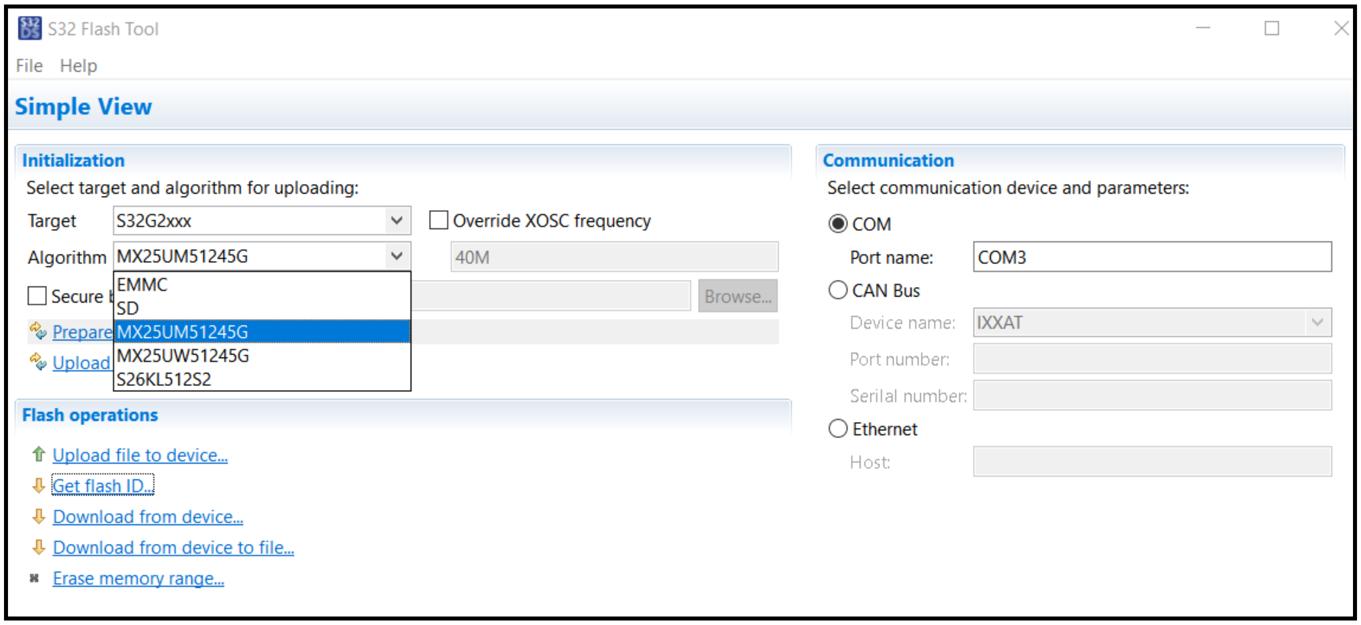
Figure 23. **Reconfiguration parameters analysis**

To improve the performance, the reconfiguration parameters should be programmed in flash memory at offset 200h.

# 5. Flash SDK usage

## 5.1. S32 flash tool

The S32 flash tool programs external flash devices such as QSPI, SD, and EMMC. The S32 flash tool can support both GUI interface and command-line interface, as shown in the following figure.


S32 Flash Tool

File Help

Simple View

Initialization

Select target and algorithm for uploading:

Target: S32G2xxx

Algorithm: MX25UM51245G

- EMMC
- SD
- MX25UM51245G
- MX25UW51245G
- S26KL512S2

Override XOSC frequency: 40M

Communication

Select communication device and parameters:

[✓ COM] Port name: COM3

CAN Bus: Device name: IXXAT

Ethernet

Figure 24. **S32 flash tool – GUI version**

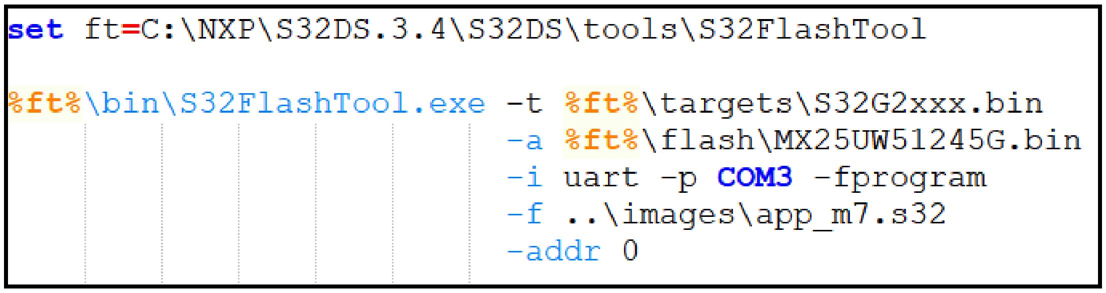

```
set ft=C:\NXP\S32DS.3.4\S32DS\tools\S32FlashTool

%ft%\bin\S32FlashTool.exe -t %ft%\targets\S32G2xxx.bin
                          -a %ft%\flash\MX25UW51245G.bin
                          -i uart -p COM3 -fprogram
                          -f ..\images\app_m7.s32
                          -addr 0
```

Figure 25. **S32 flash tool – command-line version**

For each external flash device, there is a flash device-specific flash algorithm file, such as EMMC, SD, QuadSPI devices. For S32G, the supported communication interface between the host (PC) and the target device (S32G2) are UART and CAN. The most commonly used is the UART interface.

## 5.2. How does the flash tool work

After the communication interface (like UART0 in S32G) is connected correctly, the flash tool then sends the target file (like S32G2xxx.bin) to the target device. Let's take S32G as an example, the target device is Cortex-M7_0 core. The Cortex-M7_0 core increases the UART speed from 48000 bps (For G3 is 115200 bps) to 115200*8 bps to load the algorithm file. The flash algorithm file is downloaded by S32 Flash Tool to the target device SRAM, where it will be executed by the target device (Cortex-M7_0 core). The S32 Flash Tool then sends commands to the flash algorithm along with the image to be programmed to the external device. The flash algorithm performs the programming of the image to the external flash device.

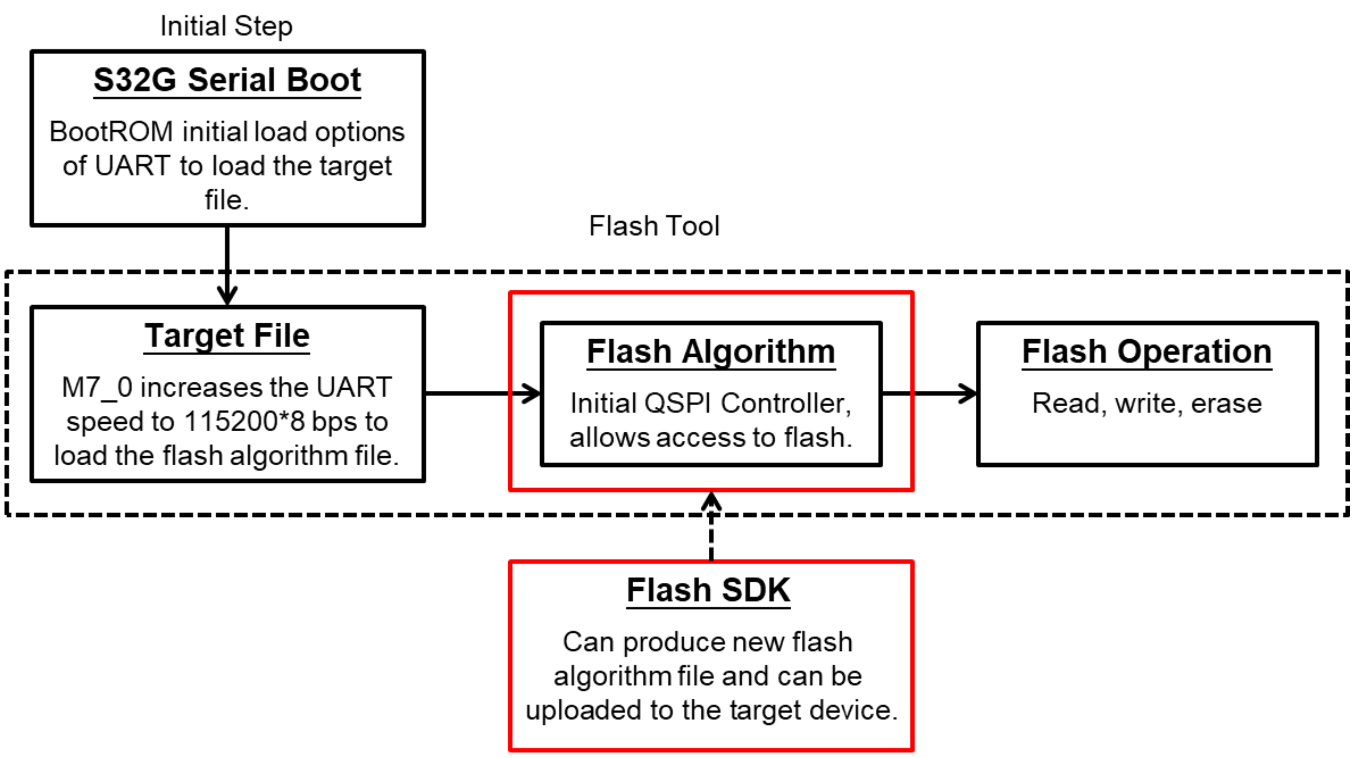

Initial Step
S32G Serial Boot
BootROM initial load options of UART to load the target file.
Flash Tool
Target File
M7_0 increases the UART speed to 115200*8 bps to load the flash algorithm file.
Flash Algorithm
Initial QSPI Controller, allows access to flash.
Flash Operation
Read, write, erase
Flash SDK
Can produce new flash algorithm file and can be uploaded to the target device.

Figure 26. **S32 flash tool workflow**

The Flash SDK provides the capability to produce new flash algorithm files, which can be uploaded to the target device by the S32 Flash Tool and then used to program images to the associated external flash device. This example project, as provided, will build and output a binary file similar to the MX25UM51245G.bin file, which includes in the S32 Flash Tool. The example project is designed to build for Arm® Cortex-M7_0 core.

## 5.3. Flash SDK usage

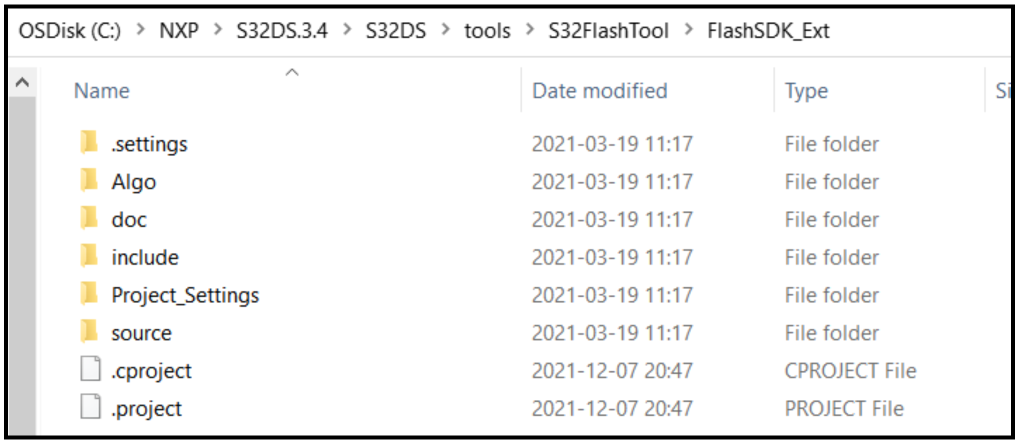
### 5.3.1. How to obtain the flash SDK

The Flash SDK is provided in the form of the S32 Design Studio for an S32 Platform project. After installing S32 Design Studio the Flash SDK project can be found at the following path:

C:\NXP\S32DS.3.4\S32DS\tools\S32FlashTool\FlashSDK_Ext

For any help refer to the document on how to import, modify, and build the Flash SDK project:

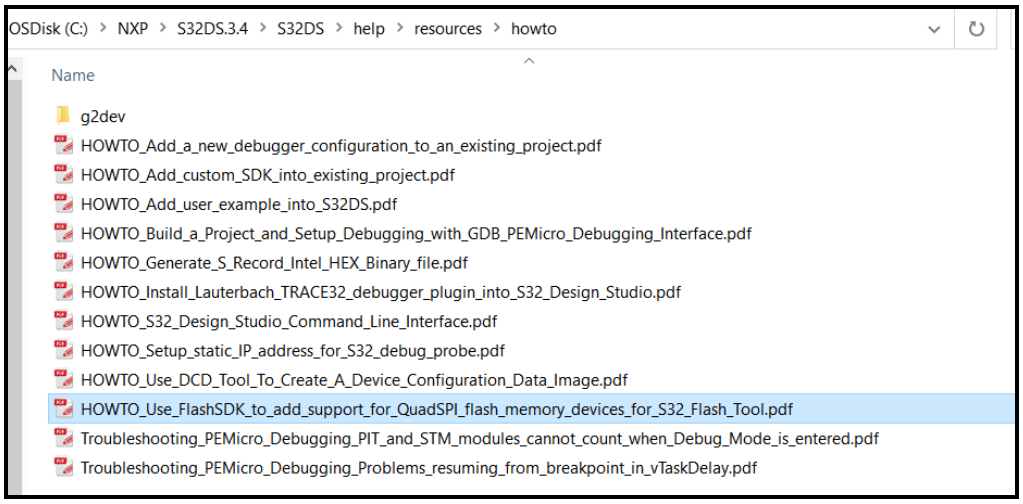
C:\NXP\S32DS.3.4\S32DS\help\resources\howto\HOWTO_Use_FlashSDK_to_add_support_for_Quad SPI_flash_memory_devices_for_S32_Flash_Tool.pdf



OSDisk (C:) > NXP > S32DS.3.4 > S32DS > tools > S32FlashTool > FlashSDK_Ext

| Name | Date modified | Type |
|---|---|---|
| .settings | 2021-03-19 11:17 | File folder |
| Algo | 2021-03-19 11:17 | File folder |
| doc | 2021-03-19 11:17 | File folder |
| include | 2021-03-19 11:17 | File folder |
| Project_Settings | 2021-03-19 11:17 | File folder |
| source | 2021-03-19 11:17 | File folder |
| .cproject | 2021-12-07 20:47 | CPROJECT File |
| .project | 2021-12-07 20:47 | PROJECT File |

Figure 27. **Flash SDK project**



OSDisk (C:) > NXP > S32DS.3.4 > S32DS > help > resources > howto

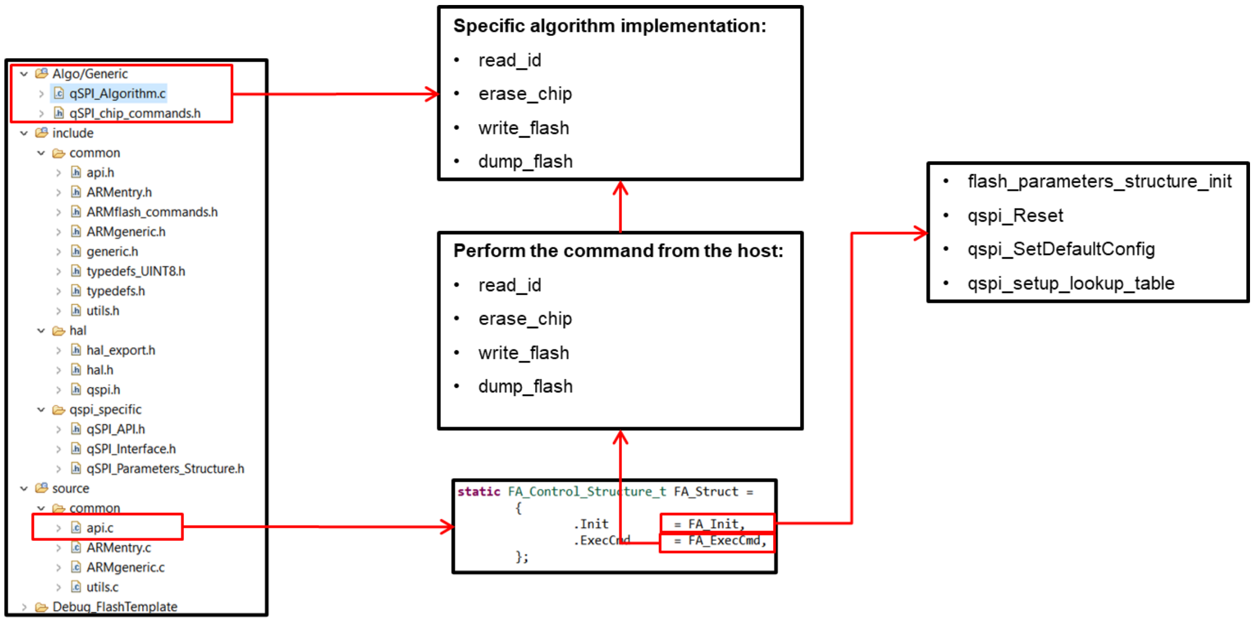
Name

- g2dev
- HOWTO_Add_a_new_debugger_configuration_to_an_existing_project.pdf
- HOWTO_Add_custom_SDK_into_existing_project.pdf
- HOWTO_Add_user_example_into_S32DS.pdf
- HOWTO_Build_a_Project_and_Setup_Debugging_with_GDB_PEMicro_Debugging_Interface.pdf
- HOWTO_Generate_S_Record_Intel_HEX_Binary_file.pdf
- HOWTO_Install_Lauterbach_TRACE32_debugger_plugin_into_S32_Design_Studio.pdf
- HOWTO_S32_Design_Studio_Command_Line_Interface.pdf
- HOWTO_Setup_static_IP_address_for_S32_debug_probe.pdf
- HOWTO_Use_DCD_Tool_To_Create_A_Device_Configuration_Data_Image.pdf
- HOWTO_Use_FlashSDK_to_add_support_for_QuadSPI_flash_memory_devices_for_S32_Flash_Tool.pdf
- Troubleshooting_PEMicro_Debugging_PIT_and_STM_modules_cannot_count_when_Debug_Mode_is_entered.pdf
- Troubleshooting_PEMicro_Debugging_Problems_resuming_from_breakpoint_in_vTaskDelay.pdf

Figure 28. **The help document for Flash SDK**

### 5.3.2. The structure of flash SDK

The Flash SDK project is intended to simplify the creation of a new algorithm. It includes generic implementation of basic IO operations such as read/write/erase.



```
static FA_Control_Structure_t FA_Struct =
        {
                .Init           = FA_Init,
                .ExecCmd        = FA_ExecCmd,
        };
```

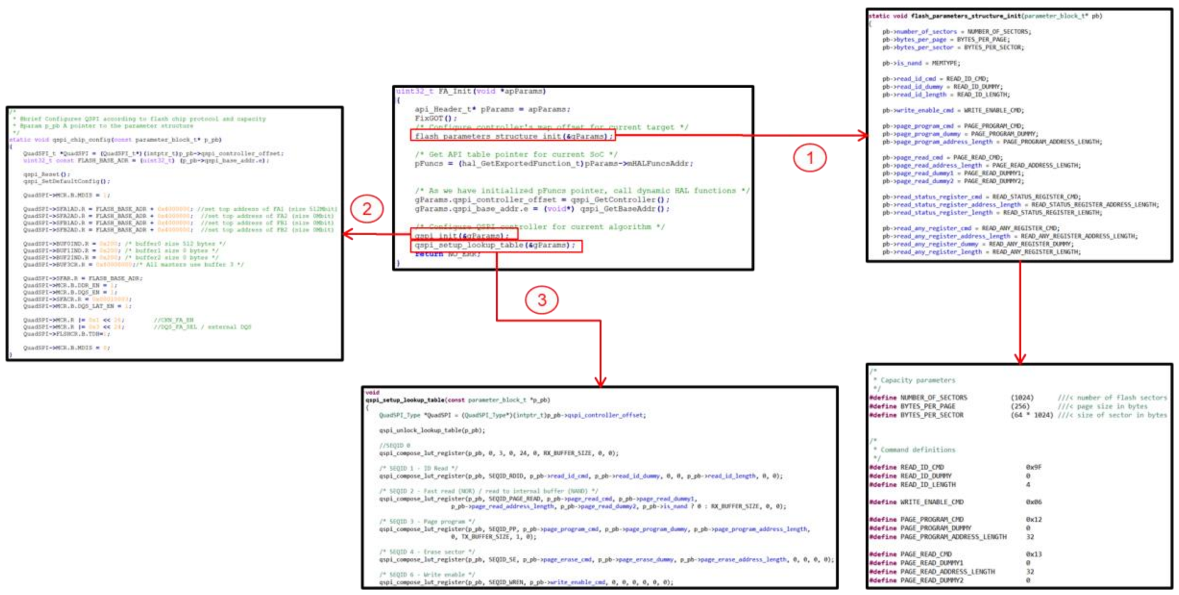
Figure 29. **The structure of flash SDK**

The most important here is the FlashSDK_Ext/Algo/Generic directory containing the implementation of flash driver functionality. To implement your algorithm, it should be enough to change source files in FlashSDK_Ext/Algo/Generic.

The entrance of the Flash SDK is the FA_Struct structure. Once the algorithm file is loaded, the FA_Init function will be called to initialize the QPSI controller. S32G calls the FA_ExeCmd function to execute the read/write/erase command received from the host.

## 5.3.3. How to create a new flash algorithm by flash SDK

The following steps shows how to take command from the host to call different functions.



```c
/*
 * @brief Configures QSPI according to flash chip protocol and capacity
 * @param p_pb A pointer to the parameter structure
 */
static void qspi_chip_config(const parameter_block_t* p_pb)
{
    QuadSPI_t *QuadSPI = (QuadSPI_t*)(intptr_t)p_pb->qspi_controller_offset;
    uint32_t const FLASH_BASE_ADR = (uint32_t) (p_pb->qspi_base_addr.e);

    qspi_Reset();
    qspi_SetDefaultConfig();

    QuadSPI->MCR.B.MDIS = 1;

    QuadSPI->SFA1AD.R = FLASH_BASE_ADR + 0x4000000; //set top address of FA1 (size 512Mbit)
    QuadSPI->SFA2AD.R = FLASH_BASE_ADR + 0x4000000;  //set top address of FA2 (size 0Mbit)
    QuadSPI->SFB1AD.R = FLASH_BASE_ADR + 0x4000000;  //set top address of FB1 (size 0Mbit)
    QuadSPI->SFB2AD.R = FLASH_BASE_ADR + 0x4000000;  //set top address of FB2 (size 0Mbit)

    QuadSPI->BUF0IND.R = 0x200; /* buffer0 size 512 bytes */
    QuadSPI->BUF1IND.R = 0x200; /* buffer1 size 0 bytes */
    QuadSPI->BUF2IND.R = 0x200; /* buffer2 size 0 bytes */
    QuadSPI->BUF3CR.R = 0x80000000;/* All masters use buffer 3 */

    QuadSPI->SFAR.R = FLASH_BASE_ADR;
    QuadSPI->MCR.B.DDR_EN = 1;
    QuadSPI->MCR.B.DQS_EN = 1;
    QuadSPI->SFACR.R = 0x00010003;
    QuadSPI->MCR.B.DQS_LAT_EN = 1;

    QuadSPI->MCR.R |= 0x1 << 26;        //CKN_FA_EN
    QuadSPI->MCR.R |= 0x3 << 24;        //DQS_FA_SEL / external DQS
    QuadSPI->FLSHCR.B.TDH=1;

    QuadSPI->MCR.B.MDIS = 0;
}
```

```c
uint32_t FA_Init(void *apParams)
{
    api_Header_t* pParams = apParams;
    FixGOT();
    /* Configure controller's map offset for current target */
    flash_parameters_structure_init(&gParams);

    /* Get API table pointer for current SoC */
    pFuncs = (hal_GetExportedFunction_t)pParams->mHALFuncsAddr;


    /* As we have initialized pFuncs pointer, call dynamic HAL functions */
    gParams.qspi_controller_offset = qspi_GetController();
    gParams.qspi_base_addr.e = (void*) qspi_GetBaseAddr();

    /* Configure QSPI controller for current algorithm */
    qspi_init(&gParams);
    qspi_setup_lookup_table(&gParams);
    return NO_ERR;
}
```

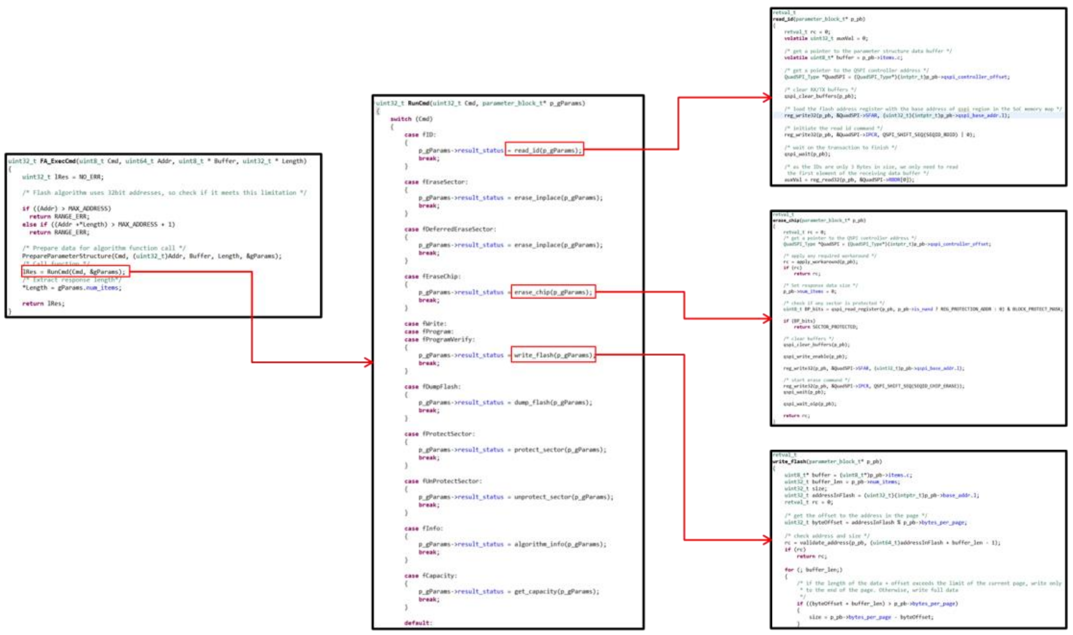
```c
static void flash_parameters_structure_init(parameter_block_t* pb)
{
    pb->number_of_sectors = NUMBER_OF_SECTORS;
    pb->bytes_per_page = BYTES_PER_PAGE;
    pb->bytes_per_sector = BYTES_PER_SECTOR;

    pb->is_nand = MEMTYPE;

    pb->read_id_cmd = READ_ID_CMD;
    pb->read_id_dummy = READ_ID_DUMMY;
    pb->read_id_length = READ_ID_LENGTH;

    pb->write_enable_cmd = WRITE_ENABLE_CMD;

    pb->page_program_cmd = PAGE_PROGRAM_CMD;
    pb->page_program_dummy = PAGE_PROGRAM_DUMMY;
    pb->page_program_address_length = PAGE_PROGRAM_ADDRESS_LENGTH;

    pb->page_read_cmd = PAGE_READ_CMD;
    pb->page_read_address_length = PAGE_READ_ADDRESS_LENGTH;
    pb->page_read_dummy1 = PAGE_READ_DUMMY1;
    pb->page_read_dummy2 = PAGE_READ_DUMMY2;

    pb->read_status_register_cmd = READ_STATUS_REGISTER_CMD;
    pb->read_status_register_address_length = READ_STATUS_REGISTER_ADDRESS_LENGTH;
    pb->read_status_register_length = READ_STATUS_REGISTER_LENGTH;

    pb->read_any_register_cmd = READ_ANY_REGISTER_CMD;
    pb->read_any_register_address_length = READ_ANY_REGISTER_ADDRESS_LENGTH;
    pb->read_any_register_dummy = READ_ANY_REGISTER_DUMMY;
    pb->read_any_register_length = READ_ANY_REGISTER_LENGTH;
```

```c
void
qspi_setup_lookup_table(const parameter_block_t *p_pb)
{
    QuadSPI_Type *QuadSPI = (QuadSPI_Type*)(intptr_t)p_pb->qspi_controller_offset;

    qspi_unlock_lookup_table(p_pb);

    //SEQID 0
    qspi_compose_lut_register(p_pb, 0, 3, 0, 24, 0, RX_BUFFER_SIZE, 0, 0);

    /* SEQID 1 - ID Read */
    qspi_compose_lut_register(p_pb, SEQID_RDID, p_pb->read_id_cmd, p_pb->read_id_dummy, 0, 0, p_pb->read_id_length, 0, 0);

    /* SEQID 2 - Fast read (NOR) / read to internal buffer (NAND) */
    qspi_compose_lut_register(p_pb, SEQID_PAGE_READ, p_pb->page_read_cmd, p_pb->page_read_dummy1,
                              p_pb->page_read_address_length, p_pb->page_read_dummy2, p_pb->is_nand ? 0 : RX_BUFFER_SIZE, 0, 0);

    /* SEQID 3 - Page program */
    qspi_compose_lut_register(p_pb, SEQID_PP, p_pb->page_program_cmd, p_pb->page_program_dummy, p_pb->page_program_address_length,
                              0, TX_BUFFER_SIZE, 1, 0);

    /* SEQID 4 - Erase sector */
    qspi_compose_lut_register(p_pb, SEQID_SE, p_pb->page_erase_cmd, p_pb->page_erase_dummy, p_pb->page_erase_address_length, 0, 0, 0, 0);

    /* SEQID 6 - Write enable */
    qspi_compose_lut_register(p_pb, SEQID_WREN, p_pb->write_enable_cmd, 0, 0, 0, 0, 0, 0);
```

```c
/*
 * Capacity parameters
 */
#define NUMBER_OF_SECTORS              (1024)      ///< number of flash sectors
#define BYTES_PER_PAGE                 (256)       ///< page size in bytes
#define BYTES_PER_SECTOR               (64 * 1024) ///< size of sector in bytes


/*
 * Command definitions
 */
#define READ_ID_CMD                     0x9F
#define READ_ID_DUMMY                   0
#define READ_ID_LENGTH                  4

#define WRITE_ENABLE_CMD                0x06

#define PAGE_PROGRAM_CMD                0x12
#define PAGE_PROGRAM_DUMMY              0
#define PAGE_PROGRAM_ADDRESS_LENGTH     32

#define PAGE_READ_CMD                   0x13
#define PAGE_READ_DUMMY1                0
#define PAGE_READ_ADDRESS_LENGTH        32
#define PAGE_READ_DUMMY2                0
```

Figure 30. **Create a new flash algorithm**

1. Initial the flash parameter's structure, including the supported command of the specific external flash device.
2. Initial the QuadSPI controller according to the specific flash chip protocol and capacity.
3. Set up LUT according to the requirement of the external flash device.
4. Modify the algorithm implementation of specific flash chip operations (such as reading, writing, and erasing).



```c
uint32_t FA_ExecCmd(uint8_t Cmd, uint64_t Addr, uint8_t * Buffer, uint32_t * Length)
{
    uint32_t lRes = NO_ERR;

    /* Flash algorithm uses 32bit addresses, so check if it meets this limitation */

    if ((Addr) > MAX_ADDRESS)
      return RANGE_ERR;
    else if ((Addr +*Length) > MAX_ADDRESS + 1)
      return RANGE_ERR;

    /* Prepare data for algorithm function call */
    PrepareParameterStructure(Cmd, (uint32_t)Addr, Buffer, Length, &gParams);
    /* Call function */
    lRes = RunCmd(Cmd, &gParams);
    /* Extract response length*/
    *Length = gParams.num_items;

    return lRes;
}
```

```c
uint32_t RunCmd(uint32_t Cmd, parameter_block_t* p_gParams)
{
    switch (Cmd)
    {
        case fID:
        {
            p_gParams->result_status = read_id(p_gParams);
            break;
        }

        case fEraseSector:
        {
            p_gParams->result_status = erase_inplace(p_gParams);
            break;
        }

        case fDeferredEraseSector:
        {
            p_gParams->result_status = erase_inplace(p_gParams);
            break;
        }

        case fEraseChip:
        {
            p_gParams->result_status = erase_chip(p_gParams);
            break;
        }

        case fWrite:
        case fProgram:
        case fProgramVerify:
        {
            p_gParams->result_status = write_flash(p_gParams);
            break;
        }

        case fDumpFlash:
        {
            p_gParams->result_status = dump_flash(p_gParams);
            break;
        }

        case fProtectSector:
        {
            p_gParams->result_status = protect_sector(p_gParams);
            break;
        }

        case fUnProtectSector:
        {
            p_gParams->result_status = unprotect_sector(p_gParams);
            break;
        }

        case fInfo:
        {
            p_gParams->result_status = algorithm_info(p_gParams);
            break;
        }

        case fCapacity:
        {
            p_gParams->result_status = get_capacity(p_gParams);
            break;
        }

        default:
```

```c
retval_t
read_id(parameter_block_t* p_pb)
{
    retval_t rc = 0;
    volatile uint32_t auxVal = 0;

    /* get a pointer to the parameter structure data buffer */
    volatile uint8_t* buffer = p_pb->items.c;

    /* get a pointer to the QSPI controller address */
    QuadSPI_Type *QuadSPI = (QuadSPI_Type*)(intptr_t)p_pb->qspi_controller_offset;

    /* clear RX/TX buffers */
    qspi_clear_buffers(p_pb);

    /* load the flash address register with the base address of qspi region in the SoC memory map */
    reg_write32(p_pb, &QuadSPI->SFAR, (uint32_t)(intptr_t)p_pb->qspi_base_addr.l);

    /* initiate the read id command */
    reg_write32(p_pb, &QuadSPI->IPCR, QSPI_SHIFT_SEQ(SEQID_RDID) | 0);

    /* wait on the transaction to finish */
    qspi_wait(p_pb);

    /* as the IDs are only 3 Bytes in size, we only need to read
     the first element of the receiving data buffer */
    auxVal = reg_read32(p_pb, &QuadSPI->RBDR[0]);
```

```c
retval_t
erase_chip(parameter_block_t* p_pb)
{
    retval_t rc = 0;
    /* get a pointer to the QSPI controller address */
    QuadSPI_Type *QuadSPI = (QuadSPI_Type*)(intptr_t)p_pb->qspi_controller_offset;

    /* apply any required workaround */
    rc = apply_workaround(p_pb);
    if (rc)
        return rc;

    /* Set response data size */
    p_pb->num_items = 0;

    /* check if any sector is protected */
    uint8_t BP_bits = qspi_read_register(p_pb, p_pb->is_nand ? REG_PROTECTION_ADDR : 0) & BLOCK_PROTECT_MASK;

    if (BP_bits)
        return SECTOR_PROTECTED;

    /* clear buffers */
    qspi_clear_buffers(p_pb);

    qspi_write_enable(p_pb);

    reg_write32(p_pb, &QuadSPI->SFAR, (uint32_t)p_pb->qspi_base_addr.l);

    /* start erase command */
    reg_write32(p_pb, &QuadSPI->IPCR, QSPI_SHIFT_SEQ(SEQID_CHIP_ERASE));
    qspi_wait(p_pb);

    qspi_wait_oip(p_pb);

    return rc;
}
```

```c
retval_t
write_flash(parameter_block_t* p_pb)
{
    uint8_t* buffer = (uint8_t*)p_pb->items.c;
    uint32_t buffer_len = p_pb->num_items;
    uint32_t size;
    uint32_t addressInFlash = (uint32_t)(intptr_t)p_pb->base_addr.l;
    retval_t rc = 0;

    /* get the offset to the address in the page */
    uint32_t byteOffset = addressInFlash % p_pb->bytes_per_page;

    /* check address and size */
    rc = validate_address(p_pb, (uint64_t)addressInFlash + buffer_len - 1);
    if (rc)
        return rc;

    for (; buffer_len;)
    {
        /* if the length of the data + offset exceeds the limit of the current page, write only
         * to the end of the page. Otherwise, write full data
         */
        if ((byteOffset + buffer_len) > p_pb->bytes_per_page)
        {
            size = p_pb->bytes_per_page - byteOffset;
        }
```

Figure 31. **Create a new flash algorithm**

# 6. Flash Driver configuration method – EB tresos

## 6.1. The FLS Driver

The flash driver provides services for reading, writing, and erasing flash memory. A configuration interface for external flash setting/resetting external flash memory is connected via the microcontroller's data/address buses (memory-mapped access). The flash drive then uses the handlers/drivers for those buses to access the external flash memory device.

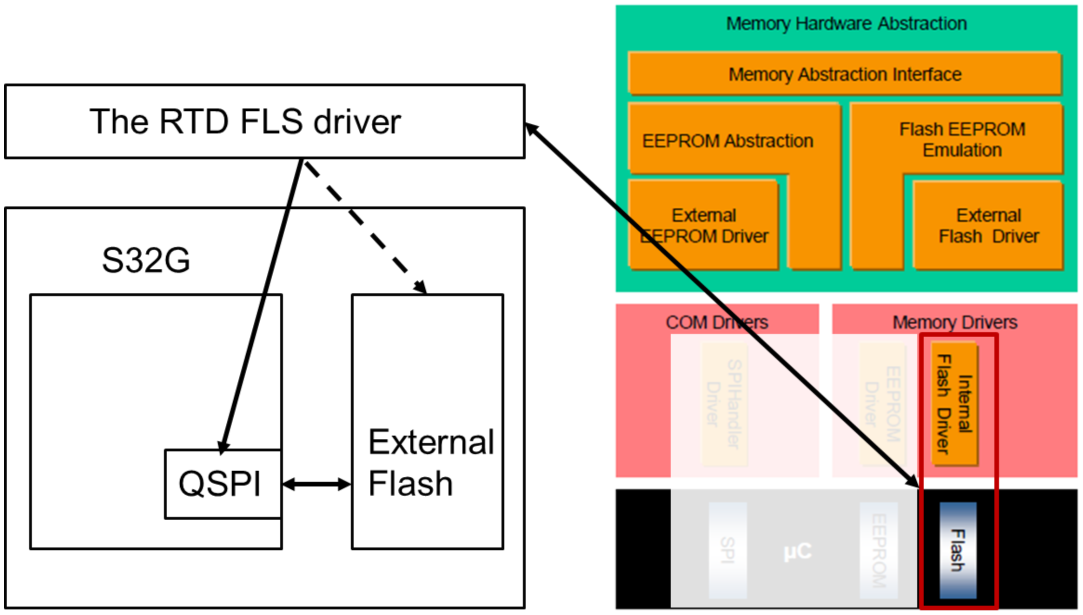
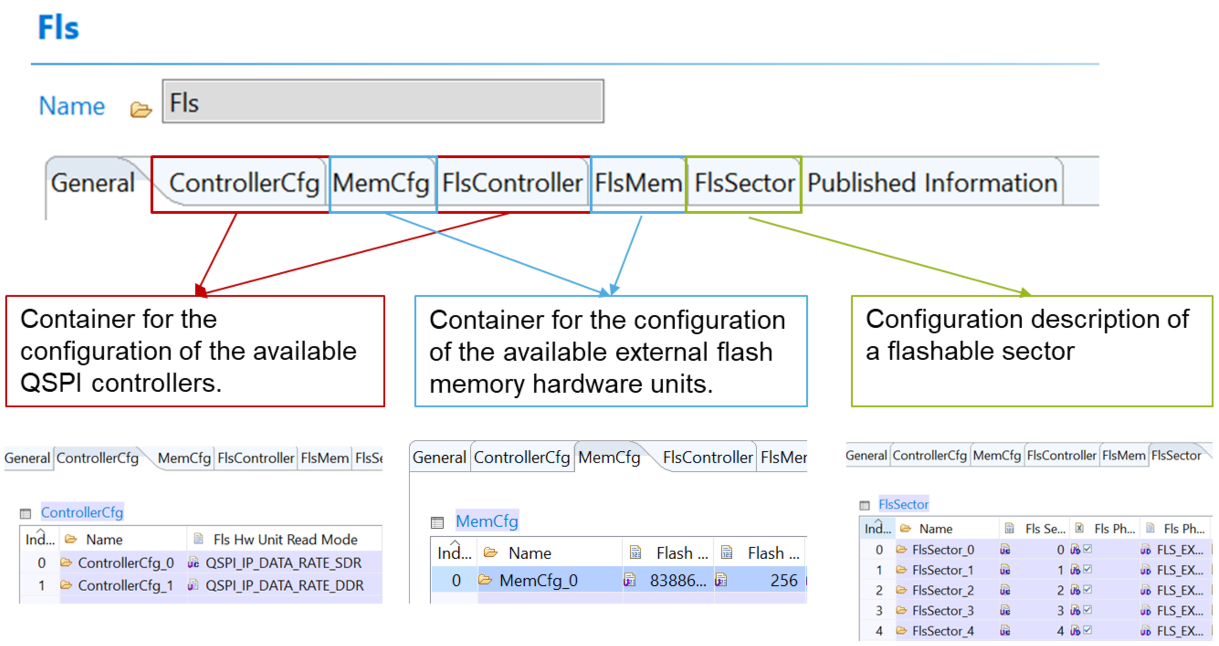


Figure 32. **The FLS driver**

## 6.2. The FLS example from RTD 2.0.0

The FLS driver of RTD 2.0.0 configures the QuadSPI controller and the external flash to implement the flash operation. The configuration includes the following three parts:

1. FLS Controller: Container for the configuration of the available QSPI controllers. It configures the S32G QSPI controller to make it work with a specific mode
2. FLS Memory: Container for the configuration of the available external flash memory hardware units. These configurations are used for the communication mode with the external flash chip. In this example. The extern flash chip is MX25UW51245G
3. FLS Sector: Configuration description of a flushable sector.



Fls

Name: Fls

Tabs: General | ControllerCfg | MemCfg | FlsController | FlsMem | FlsSector | Published Information

ControllerCfg, FlsController: Container for the configuration of the available QSPI controllers.

MemCfg, FlsMem: Container for the configuration of the available external flash memory hardware units.

FlsSector: Configuration description of a flashable sector

ControllerCfg

| Ind... | Name | Fls Hw Unit Read Mode |
|---|---|---|
| 0 | ControllerCfg_0 | QSPI_IP_DATA_RATE_SDR |
| 1 | ControllerCfg_1 | QSPI_IP_DATA_RATE_DDR |

MemCfg

| Ind... | Name | Flash ... | Flash ... |
|---|---|---|---|
| 0 | MemCfg_0 | 83886... | 256 |

FlsSector

| Ind... | Name | Fls Se... | Fls Ph... | Fls Ph... |
|---|---|---|---|---|
| 0 | FlsSector_0 | 0 | ☑ | FLS_EX... |
| 1 | FlsSector_1 | 1 | ☑ | FLS_EX... |
| 2 | FlsSector_2 | 2 | ☑ | FLS_EX... |
| 3 | FlsSector_3 | 3 | ☑ | FLS_EX... |
| 4 | FlsSector_4 | 4 | ☑ | FLS_EX... |

Figure 33. **The FLS example from RTD2.0.0**

The FLS driver of RTD 2.0.0 configures the QuadSPI controller and the external flash to implement the flash operation. The example FLS driver initializes the external flash through three phases.

1. The BootROM of the QuadSPI boot can configure the external flash to achieve the high-speed communication with the OPI mode via the QuadSPI parameter of IVT.
2. The QuadSPI driver initializes the external flash via the rest command of the external flash. The external flash turns the default state that is the SPI mode. To configure the external flash, the QuadSPI controller is set to the corresponding mode
3. The Fls driver turns the OPI mode of the external flash and QuadSPI controller again to increase the QSPI performance.

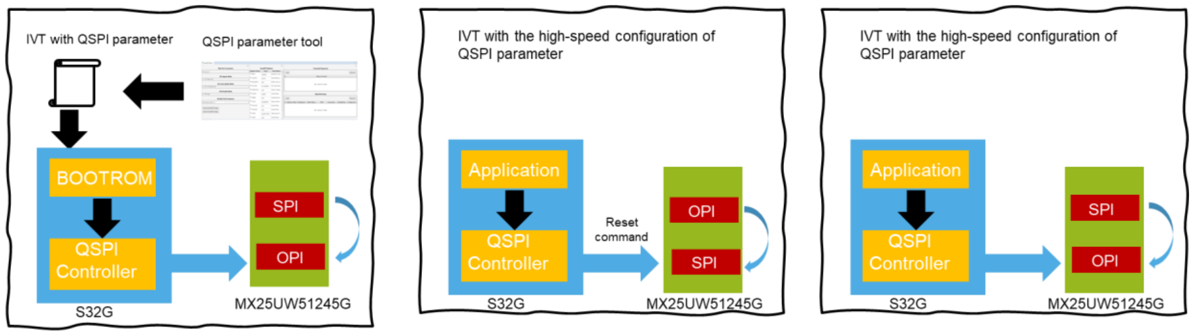
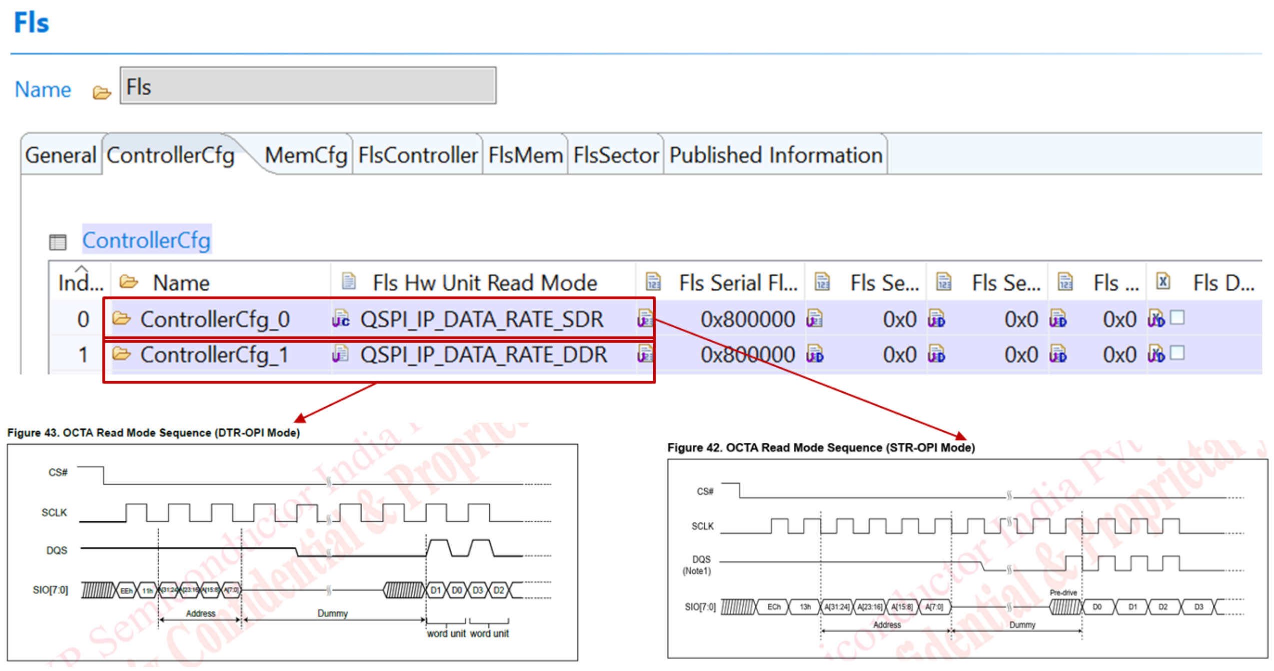


Figure 34. **Initializes the external flash via FLS driver**

The FLS example has two types of configurations for the QSPI controller to fit the external flash MX25UW51245G.

1. ControllerCfg_0 shows the configuration for the SPI mode of the external flash.
2. ControllerCfg_1 shows the configuration of the OPI mode of the external flash.



| Ind... | Name | Fls Hw Unit Read Mode | Fls Serial Fl... | Fls Se... | Fls Se... | Fls ... | Fls D... |
|---|---|---|---|---|---|---|---|
| 0 | ControllerCfg_0 | QSPI_IP_DATA_RATE_SDR | 0x800000 | 0x0 | 0x0 | 0x0 | ☐ |
| 1 | ControllerCfg_1 | QSPI_IP_DATA_RATE_DDR | 0x800000 | 0x0 | 0x0 | 0x0 | ☐ |

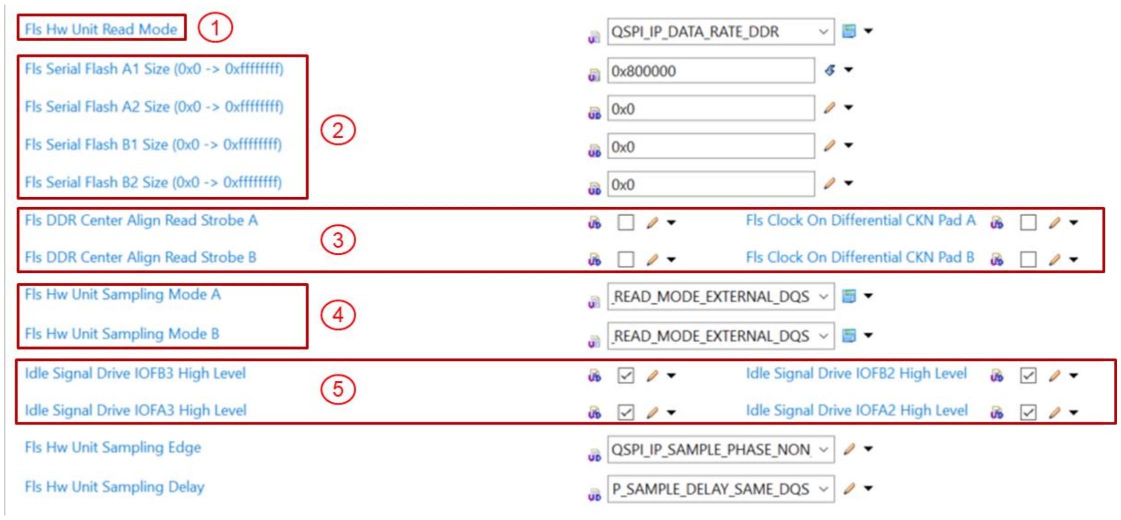
Figure 35. **The configurations of FLS driver**

**NOTE**

The read command as an example shows the SPI and OPI mode. For more information refer to the MX25UW51245G reference manual.

## 6.2.1. FLS controller configuration

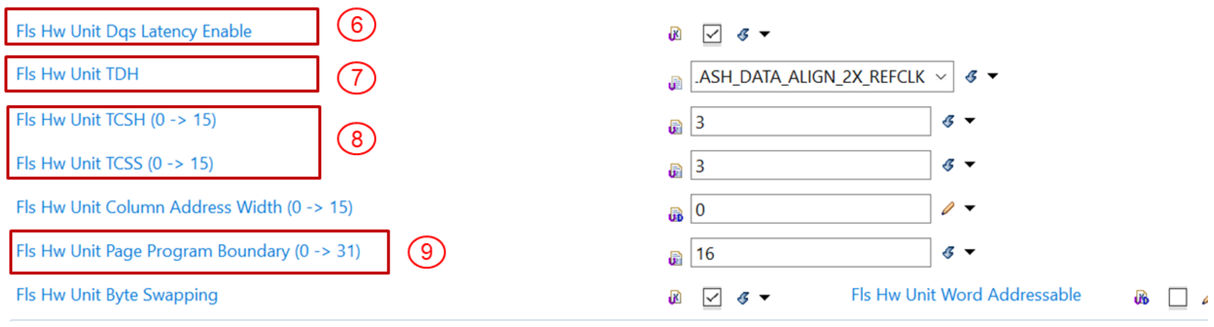
### 6.2.1.1. FLS controller



| | |
|---|---|
| Fls Hw Unit Read Mode | QSPI_IP_DATA_RATE_DDR |
| Fls Serial Flash A1 Size (0x0 -> 0xffffffff) | 0x800000 |
| Fls Serial Flash A2 Size (0x0 -> 0xffffffff) | 0x0 |
| Fls Serial Flash B1 Size (0x0 -> 0xffffffff) | 0x0 |
| Fls Serial Flash B2 Size (0x0 -> 0xffffffff) | 0x0 |
| Fls DDR Center Align Read Strobe A | ☐ |
| Fls DDR Center Align Read Strobe B | ☐ |
| Fls Clock On Differential CKN Pad A | ☐ |
| Fls Clock On Differential CKN Pad B | ☐ |
| Fls Hw Unit Sampling Mode A | READ_MODE_EXTERNAL_DQS |
| Fls Hw Unit Sampling Mode B | READ_MODE_EXTERNAL_DQS |
| Idle Signal Drive IOFB3 High Level | ☑ |
| Idle Signal Drive IOFA3 High Level | ☑ |
| Idle Signal Drive IOFB2 High Level | ☑ |
| Idle Signal Drive IOFA2 High Level | ☑ |
| Fls Hw Unit Sampling Edge | QSPI_IP_SAMPLE_PHASE_NON |
| Fls Hw Unit Sampling Delay | P_SAMPLE_DELAY_SAME_DQS |

Figure 36. **FLS controller 1/3**

1. The hardware unit read mode:
   - QSPI_IP_DATA_RATE_SDR (single data rate) which samples incoming data on a single edge.
   - QSPI_IP_DATA_RATE_DDR (double data rate) which samples incoming data on both edges.
2. Size of flash device connected to the side of the controller. Set to 0 if no flash device is connected. The MX25UW51245G support a 512 Mb memory size.
3. Enable CK2 output 90-degree phase shifted clock for flash. It also enables the clock on differential CKN pad of Flash, MX25UW51245G does not support differential CKN.
4. DQS clock for sampling read data at Flash QuadSPI port:
   - SPI_IP_READ_MODE_INTERNAL_DQS
   - QSPI_IP_READ_MODE_LOOPBACK
   - QSPI_IP_READ_MODE_LOOPBACK_DQS
   - QSPI_IP_READ_MODE_EXTERNAL_DQS
5. Idle Signal Drive. This bit determines the logic level output of the QuadSPI module is driven to in the inactive state.



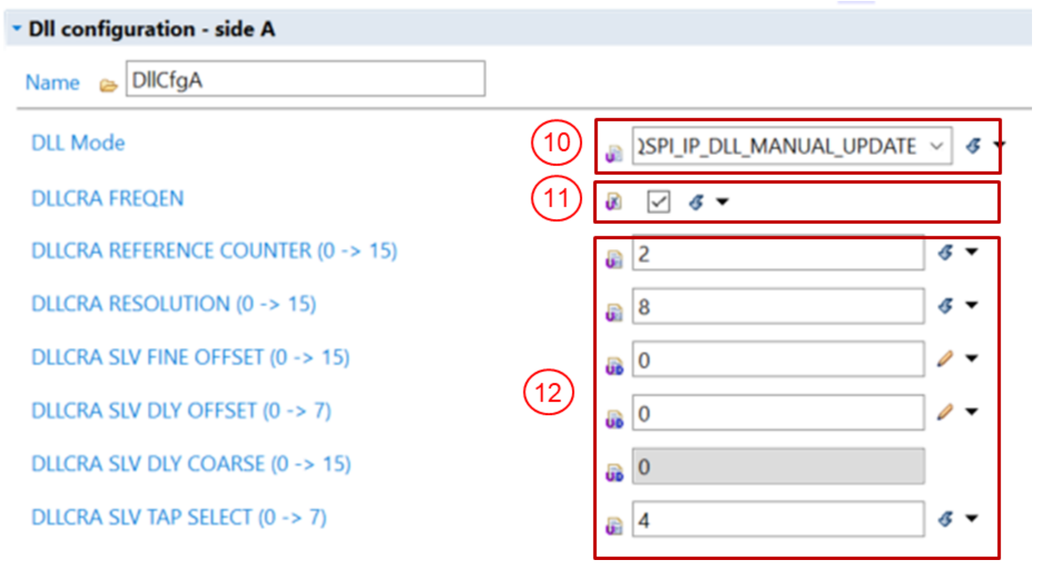
| | |
|---|---|
| Fls Hw Unit Dqs Latency Enable (6) | ✓ |
| Fls Hw Unit TDH (7) | .ASH_DATA_ALIGN_2X_REFCLK |
| Fls Hw Unit TCSH (0 -> 15) (8) | 3 |
| Fls Hw Unit TCSS (0 -> 15) (8) | 3 |
| Fls Hw Unit Column Address Width (0 -> 15) | 0 |
| Fls Hw Unit Page Program Boundary (0 -> 31) (9) | 16 |
| Fls Hw Unit Byte Swapping | ✓ |
| Fls Hw Unit Word Addressable | ☐ |

Figure 37. **FLS controller 2/3**

6. DQS Latency Enable. It is used to support external devices which add latency cycles in the DQS signal. Remarkably DQS Latency is applicable only for DQS sampling mode, FLS_EXTERNAL_DQS.
7. TDH: Serial flash data in hold time.
   - QSPI_IP_FLASH_DATA_ALIGN_REFCLK = Data aligned with the pose of the Internal reference clock of QuadSPI.
   - QSPI_IP_FLASH_DATA_ALIGN_2X_REFCLK = Data aligned with 2x serial flash half clock.
8. TCSH and TCSS

- TCSH: Serial flash CS hold time in terms of serial flash clock cycles. A bigger value will release the CS signal later after the transaction ends. The actual delay between chip select and clock is defined as, TCSH = 1 SCK CLK if N= 0/1 else, N SCK CLK if N>1, where N is the setting of TCSH
- TCSS: Serial flash CS setup time in terms of serial flash clock cycles. A bigger value will pull the CS signal earlier before the transaction starts. The actual delay between chip select and clock is defined as, TCSS = 0.5 SCK CLK if N= 0/1 else, N+0.5 SCK CLK if N>1, where N is the setting of TCSS.

9. Page program boundary
   - Flash specific Page Program boundary size should be programmed here during format. The DLL is a general-purpose, dynamically adaptive clock delay module. It provides the ability to select a quantized delay (in fractions of the clock period) regardless of on-chip variations such as process, voltage, and temperature (PVT). The DLL is suitable for applications where accurate delay adjustment is required, such as in case of DDR interfaces.


Dll configuration - side A

Name: DllCfgA

| | | |
|---|---|---|
| DLL Mode | 10 | )SPI_IP_DLL_MANUAL_UPDATE |
| DLLCRA FREQEN | 11 | ✓ |
| DLLCRA REFERENCE COUNTER (0 -> 15) | 12 | 2 |
| DLLCRA RESOLUTION (0 -> 15) | | 8 |
| DLLCRA SLV FINE OFFSET (0 -> 15) | | 0 |
| DLLCRA SLV DLY OFFSET (0 -> 7) | | 0 |
| DLLCRA SLV DLY COARSE (0 -> 15) | | 0 |
| DLLCRA SLV TAP SELECT (0 -> 7) | | 4 |

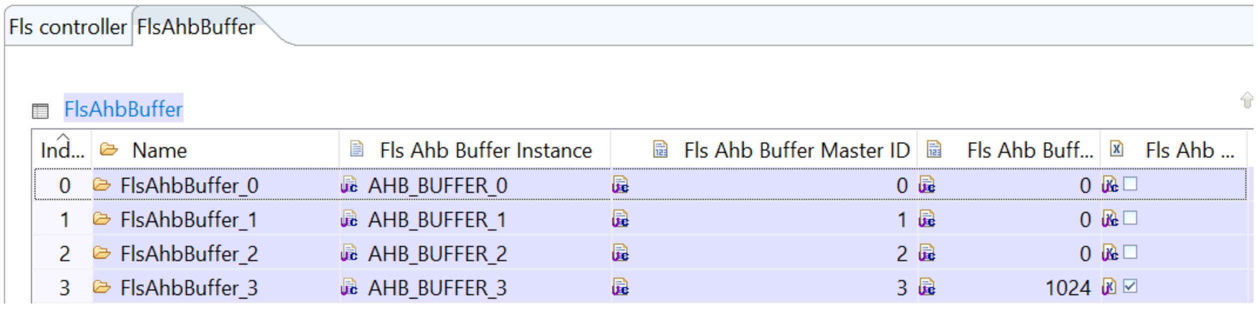
Figure 38. **FLS controller 3/3**

10. Choose the mode of DLL feature for the flash interface.
11. Frequency enables for flash. These are 60-133 MHz (low freq) and 133-200 MHz (high freq)
    - Disable - Selects delay-chain for low frequency of operation.
    - Enable - Selects delay-chain for high frequency of operation.
12. For the parameters of DLL refer to the device reference manual.

### 6.2.1.2. FLS AHB buffer

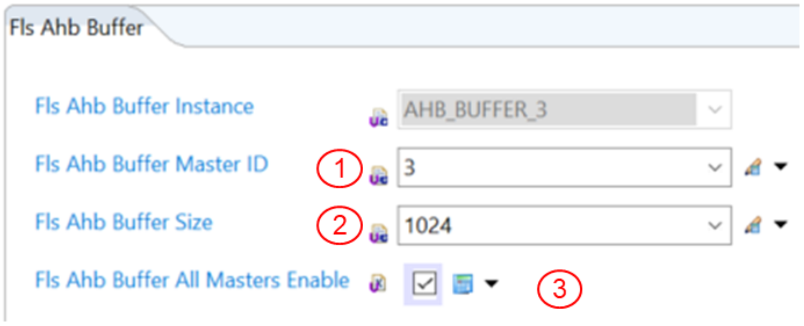
The size of each of these buffers is configurable with the minimum size being 0 bytes and maximum size being the size of the complete buffer instantiated (1024 bytes).

If an instance is not present, the corresponding AHB buffer will be configured with size 0.The size of the AHB_BUFFER_3 instance will be configured to at least the selected size, or more, up until the maximum value is reached.


Fls controller | FlsAhbBuffer

FlsAhbBuffer

| Ind... | Name | Fls Ahb Buffer Instance | Fls Ahb Buffer Master ID | Fls Ahb Buff... | Fls Ahb ... |
|---|---|---|---|---|---|
| 0 | FlsAhbBuffer_0 | AHB_BUFFER_0 | 0 | 0 | ☐ |
| 1 | FlsAhbBuffer_1 | AHB_BUFFER_1 | 1 | 0 | ☐ |
| 2 | FlsAhbBuffer_2 | AHB_BUFFER_2 | 2 | 0 | ☐ |
| 3 | FlsAhbBuffer_3 | AHB_BUFFER_3 | 3 | 1024 | ☑ |


Fls Ahb Buffer

| | |
|---|---|
| Fls Ahb Buffer Instance | AHB_BUFFER_3 |
| Fls Ahb Buffer Master ID (1) | 3 |
| Fls Ahb Buffer Size (2) | 1024 |
| Fls Ahb Buffer All Masters Enable | ☑ (3) |

Figure 39. **FLS AHB buffer**

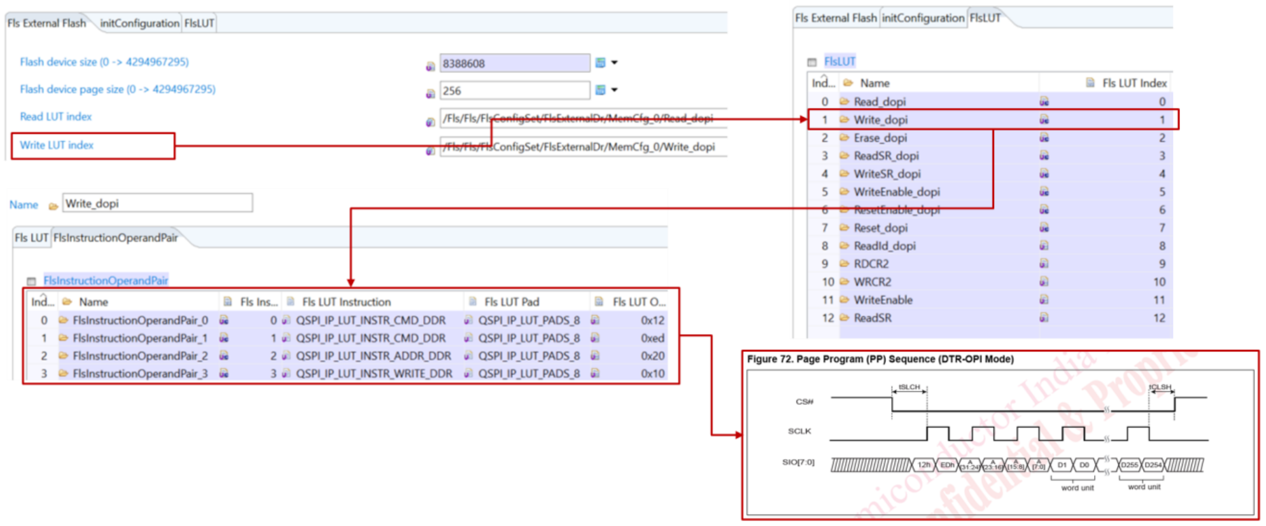
1. The ID of the AHB master associated with this buffer. Any AHB access with this master port number is routed to this buffer. It must be ensured that the master IDs associated with all buffers must be different.
2. The size allocated to this AHB Buffer instance. The minimum size is 8 bytes, the maximum size is the entire AHB Buffer.
3. When set, buffer3 acts as an all-master buffer. Any AHB access with a master port number not matching with the master ID of buffer0 or buffer1 or buffer2 is routed to buffer3. When set, the Master ID parameter for this buffer is ignored.

## 6.2.2. FLS memory configuration

To get the parameters for configuring the external flash refer to the MX25UW51245G reference manual.

- **Flash device size** is the size in bytes of this flash device. The size of the external flash MX25UW51245G is 512 Mb.
- **Flash device page size** is the maximum amount of data that the flash device can write in a single write operation. For example, The Page Program (PP/PP3B/PP4B) instruction of MX25UW51245G programs only the last 256 data bytes sent to the device. The Flash device page size is 256.

For information on **Read LUT index** and **Write LUT index** refer to the Fls LUT. This container is for the configuration of the Look Up Table holding all the Instruction/Operands sequences.



Fls External Flash | initConfiguration | FlsLUT

| | |
|---|---|
| Flash device size (0 -> 4294967295) | 8388608 |
| Flash device page size (0 -> 4294967295) | 256 |
| Read LUT index | /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_0/Read_dopi |
| Write LUT index | /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_0/Write_dopi |

Name: Write_dopi

Fls LUT | FlsInstructionOperandPair

FlsInstructionOperandPair

| Ind... | Name | Fls Ins... | Fls LUT Instruction | Fls LUT Pad | Fls LUT O... |
|---|---|---|---|---|---|
| 0 | FlsInstructionOperandPair_0 | 0 | QSPI_IP_LUT_INSTR_CMD_DDR | QSPI_IP_LUT_PADS_8 | 0x12 |
| 1 | FlsInstructionOperandPair_1 | 1 | QSPI_IP_LUT_INSTR_CMD_DDR | QSPI_IP_LUT_PADS_8 | 0xed |
| 2 | FlsInstructionOperandPair_2 | 2 | QSPI_IP_LUT_INSTR_ADDR_DDR | QSPI_IP_LUT_PADS_8 | 0x20 |
| 3 | FlsInstructionOperandPair_3 | 3 | QSPI_IP_LUT_INSTR_WRITE_DDR | QSPI_IP_LUT_PADS_8 | 0x10 |

Fls External Flash | initConfiguration | FlsLUT

FlsLUT

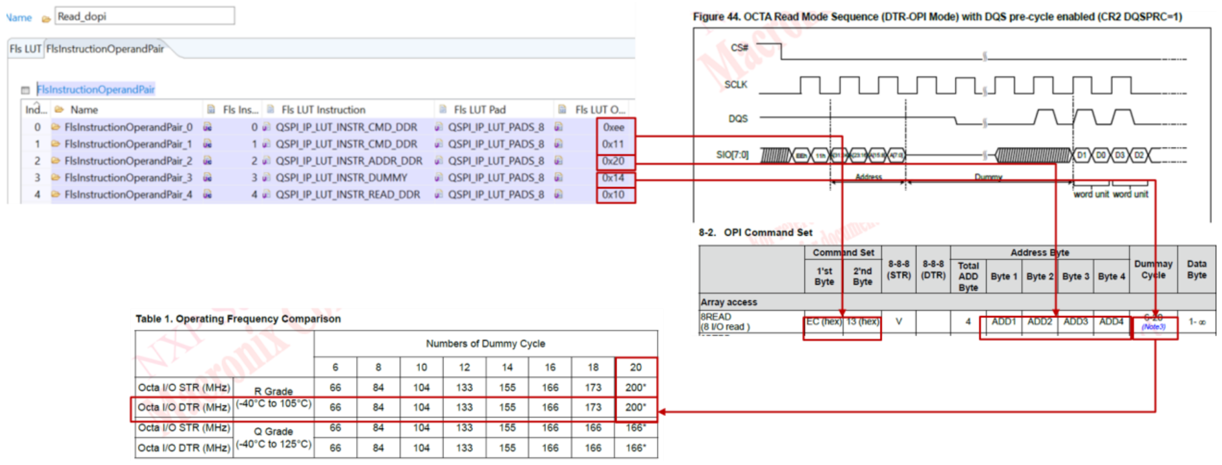
| Ind... | Name | Fls LUT Index |
|---|---|---|
| 0 | Read_dopi | 0 |
| 1 | Write_dopi | 1 |
| 2 | Erase_dopi | 2 |
| 3 | ReadSR_dopi | 3 |
| 4 | WriteSR_dopi | 4 |
| 5 | WriteEnable_dopi | 5 |
| 6 | ResetEnable_dopi | 6 |
| 7 | Reset_dopi | 7 |
| 8 | ReadId_dopi | 8 |
| 9 | RDCR2 | 9 |
| 10 | WRCR2 | 10 |
| 11 | WriteEnable | 11 |
| 12 | ReadSR | 12 |

Figure 72. Page Program (PP) Sequence (DTR-OPI Mode)

Figure 40. **FLS memory configuration**

The instruction operand pair of the FlsLut Read_dopi refer to the MX25UW51245G reference manual Chapter 8 COMMAND SET, it shows all operation command. Follow the below steps to set the instruction operand pair.

1. Create the command instruction and the operand refer to the READ command directly. The operand is 0xee and the 0x13
2. Create the address instruction and the operand refer to the READ command, the address byte is 32 bit. The operand is 0x20.
3. Create the dummy instruction and the operand refer to the QSPI clock and the reference manual of MX25UW51245G, the QSPI clock is 200 MHz. The operand is 0x14.
4. Create the read instruction and the operand is the word unit that is 0x10



Name: Read_dopi

Fls LUT | FlsInstructionOperandPair

FlsInstructionOperandPair

| Ind... | Name | Fls Ins... | Fls LUT Instruction | Fls LUT Pad | Fls LUT O... |
|---|---|---|---|---|---|
| 0 | FlsInstructionOperandPair_0 | 0 | QSPI_IP_LUT_INSTR_CMD_DDR | QSPI_IP_LUT_PADS_8 | 0xee |
| 1 | FlsInstructionOperandPair_1 | 1 | QSPI_IP_LUT_INSTR_CMD_DDR | QSPI_IP_LUT_PADS_8 | 0x11 |
| 2 | FlsInstructionOperandPair_2 | 2 | QSPI_IP_LUT_INSTR_ADDR_DDR | QSPI_IP_LUT_PADS_8 | 0x20 |
| 3 | FlsInstructionOperandPair_3 | 3 | QSPI_IP_LUT_INSTR_DUMMY | QSPI_IP_LUT_PADS_8 | 0x14 |
| 4 | FlsInstructionOperandPair_4 | 4 | QSPI_IP_LUT_INSTR_READ_DDR | QSPI_IP_LUT_PADS_8 | 0x10 |

Figure 44. OCTA Read Mode Sequence (DTR-OPI Mode) with DQS pre-cycle enabled (CR2 DQSPRC=1)

8-2. OPI Command Set

| | Command Set | | 8-8-8 (STR) | 8-8-8 (DTR) | Address Byte | | | | | Dummay Cycle | Data Byte |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1'st Byte | 2'nd Byte | | | Total ADD Byte | Byte 1 | Byte 2 | Byte 3 | Byte 4 | | |
| Array access | | | | | | | | | | | |
| 8READ (8 I/O read ) | EC (hex) | 13 (hex) | V | | 4 | ADD1 | ADD2 | ADD3 | ADD4 | 6-20 (Note3) | 1- ∞ |

Table 1. Operating Frequency Comparison

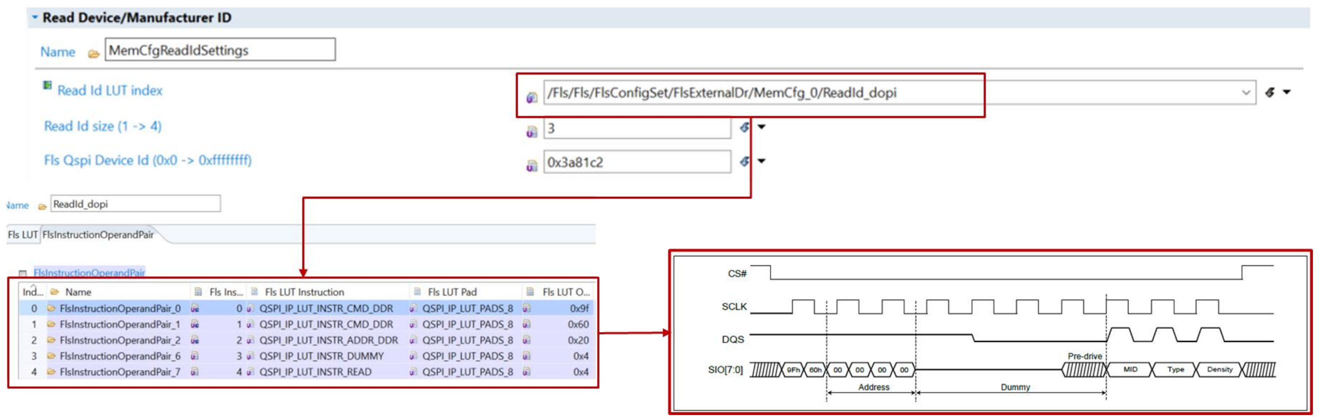
| | | Numbers of Dummy Cycle | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| Octa I/O STR (MHz) | R Grade (-40°C to 105°C) | 66 | 84 | 104 | 133 | 155 | 166 | 173 | 200* |
| Octa I/O DTR (MHz) | | 66 | 84 | 104 | 133 | 155 | 166 | 173 | 200* |
| Octa I/O STR (MHz) | Q Grade (-40°C to 125°C) | 66 | 84 | 104 | 133 | 155 | 166 | 166 | 166* |
| Octa I/O DTR (MHz) | | 66 | 84 | 104 | 133 | 155 | 166 | 166 | 166* |

Figure 41. **FLS Read_dopi**

### 6.2.2.1. **Read ID operation**

During the initialization of the external flash driver, the FLS module checks the hardware ID of the external flash device against the corresponding published parameter. If a hardware ID mismatch occurs, the FLS module would report the error code.

For more information on the Read Device/Manufacturer ID command refer to the Read Identification command of MX25UW51245G reference manual.



Read Device/Manufacturer ID

Name: MemCfgReadIdSettings

Read Id LUT index: /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_0/ReadId_dopi

Read Id size (1 -> 4): 3

Fls Qspi Device Id (0x0 -> 0xffffffff): 0x3a81c2

Name: ReadId_dopi

Fls LUT | FlsInstructionOperandPair

FlsInstructionOperandPair

| Ind... | Name | Fls Ins... | Fls LUT Instruction | Fls LUT Pad | Fls LUT O... |
|---|---|---|---|---|---|
| 0 | FlsInstructionOperandPair_0 | 0 | QSPI_IP_LUT_INSTR_CMD_DDR | QSPI_IP_LUT_PADS_8 | 0x9f |
| 1 | FlsInstructionOperandPair_1 | 1 | QSPI_IP_LUT_INSTR_CMD_DDR | QSPI_IP_LUT_PADS_8 | 0x60 |
| 2 | FlsInstructionOperandPair_2 | 2 | QSPI_IP_LUT_INSTR_ADDR_DDR | QSPI_IP_LUT_PADS_8 | 0x20 |
| 3 | FlsInstructionOperandPair_6 | 3 | QSPI_IP_LUT_INSTR_DUMMY | QSPI_IP_LUT_PADS_8 | 0x4 |
| 4 | FlsInstructionOperandPair_7 | 4 | QSPI_IP_LUT_INSTR_READ | QSPI_IP_LUT_PADS_8 | 0x4 |

Figure 42. **Read ID**

Configured value of Fls Qspi Device Id = 0x3A81C2, it means

- Memory density: 0x3A
- Memory type: 0x81
- Manufacturer ID: 0xC2

For more information refer to MX25UW51245G reference manual.

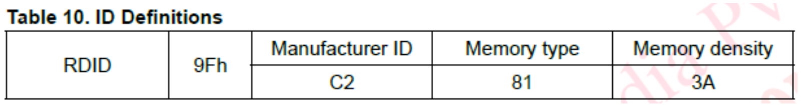


**Table 10. ID Definitions**

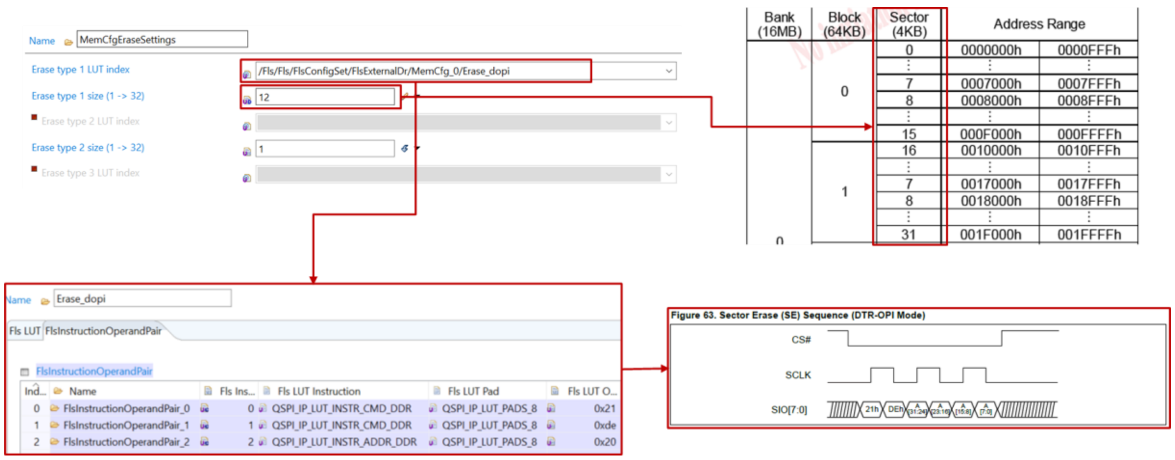
| RDID | 9Fh | Manufacturer ID | Memory type | Memory density |
|---|---|---|---|---|
|  |  | C2 | 81 | 3A |

Figure 43. **ID Definitions**

The configured READ_ID LUT sequence schedules a read id command (ex: RDID 0x9F) with a reading length of 3 bytes.

### 6.2.2.2. **Set container for erase command**

Erase command refers to the Sector Erase instruction of MX25UW51245G reference manual.



Name: MemCfgEraseSettings

| | |
|---|---|
| Erase type 1 LUT index | /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_0/Erase_dopi |
| Erase type 1 size (1 -> 32) | 12 |
| Erase type 2 LUT index | |
| Erase type 2 size (1 -> 32) | 1 |
| Erase type 3 LUT index | |

Name: Erase_dopi

Fls LUT FlsInstructionOperandPair

FlsInstructionOperandPair

| Ind... | Name | Fls Ins... | Fls LUT Instruction | Fls LUT Pad | Fls LUT O... |
|---|---|---|---|---|---|
| 0 | FlsInstructionOperandPair_0 | 0 | QSPI_IP_LUT_INSTR_CMD_DDR | QSPI_IP_LUT_PADS_8 | 0x21 |
| 1 | FlsInstructionOperandPair_1 | 1 | QSPI_IP_LUT_INSTR_CMD_DDR | QSPI_IP_LUT_PADS_8 | 0xde |
| 2 | FlsInstructionOperandPair_2 | 2 | QSPI_IP_LUT_INSTR_ADDR_DDR | QSPI_IP_LUT_PADS_8 | 0x20 |

| Bank (16MB) | Block (64KB) | Sector (4KB) | Address Range | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0000000h | 0000FFFh |
| | | ⋮ | ⋮ | ⋮ |
| | | 7 | 0007000h | 0007FFFh |
| | | 8 | 0008000h | 0008FFFh |
| | | ⋮ | ⋮ | ⋮ |
| | | 15 | 000F000h | 000FFFFh |
| | 1 | 16 | 0010000h | 0010FFFh |
| | | ⋮ | ⋮ | ⋮ |
| | | 7 | 0017000h | 0017FFFh |
| | | 8 | 0018000h | 0018FFFh |
| | | ⋮ | ⋮ | ⋮ |
| | | 31 | 001F000h | 001FFFFh |

Figure 44. **Set container for erase command**

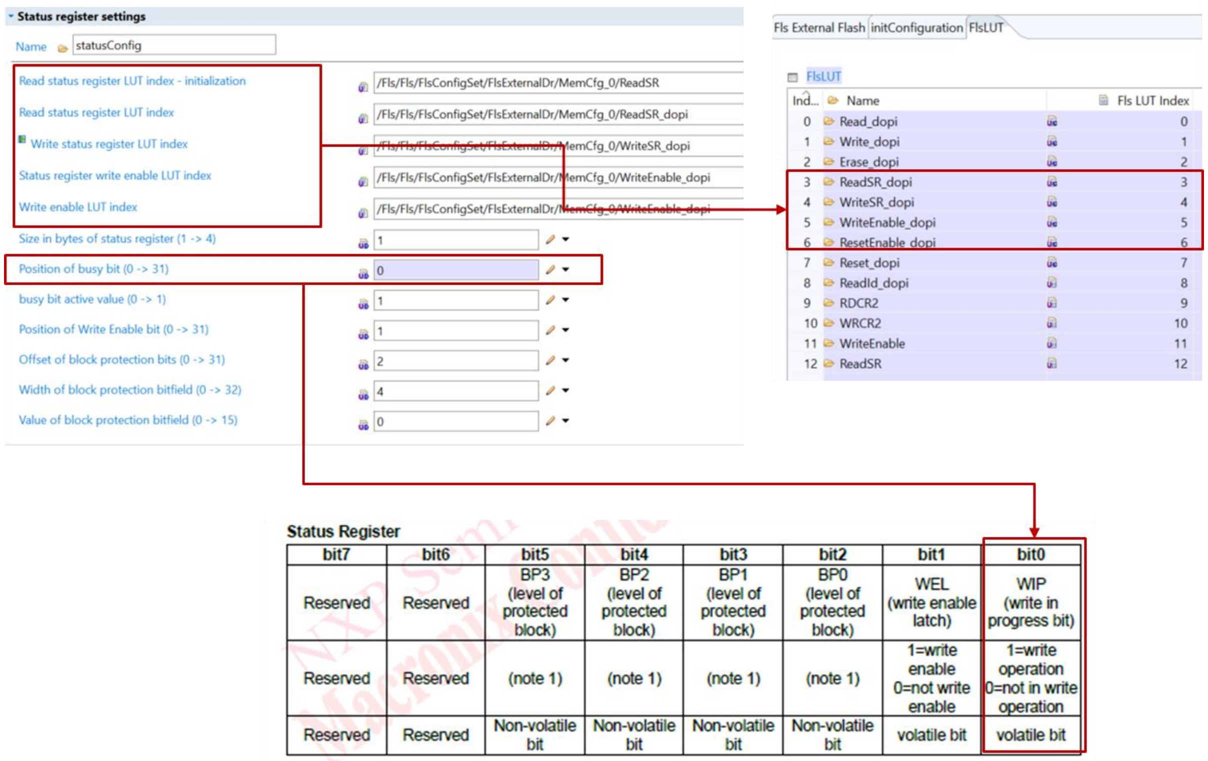
The size in bytes of the erased area is 2 ^ size; e.g. 0x0C means 4 Kbytes.

## 6.2.2.3. Set container for status register

The **WIP bit** of the status registers MX25UW51245G indicates whether the device is busy in program/erase/write progress. And the **WEL bit** needs to be set to "1" before the device can accept the program and erase instructions

Status register settings container for settings related to the status register of the flash device:

1. The instruction operand pair of the status register refer to the chapter 8 COMMAND SET of the reference manual of MX25UW51245G
2. The parameter of the status of referring to the status register of the MX25UW51245G reference manual. For example, the size of the status register is 8 bit, the size in bytes of the status register is 1.



Status register settings

| | |
|---|---|
| Name | statusConfig |
| Read status register LUT index - initialization | /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_0/ReadSR |
| Read status register LUT index | /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_0/ReadSR_dopi |
| Write status register LUT index | /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_0/WriteSR_dopi |
| Status register write enable LUT index | /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_0/WriteEnable_dopi |
| Write enable LUT index | /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_0/WriteEnable_dopi |
| Size in bytes of status register (1 -> 4) | 1 |
| Position of busy bit (0 -> 31) | 0 |
| busy bit active value (0 -> 1) | 1 |
| Position of Write Enable bit (0 -> 31) | 1 |
| Offset of block protection bits (0 -> 31) | 2 |
| Width of block protection bitfield (0 -> 32) | 4 |
| Value of block protection bitfield (0 -> 15) | 0 |

Fls External Flash | initConfiguration | FlsLUT

FlsLUT

| Ind... | Name | Fls LUT Index |
|---|---|---|
| 0 | Read_dopi | 0 |
| 1 | Write_dopi | 1 |
| 2 | Erase_dopi | 2 |
| 3 | ReadSR_dopi | 3 |
| 4 | WriteSR_dopi | 4 |
| 5 | WriteEnable_dopi | 5 |
| 6 | ResetEnable_dopi | 6 |
| 7 | Reset_dopi | 7 |
| 8 | ReadId_dopi | 8 |
| 9 | RDCR2 | 9 |
| 10 | WRCR2 | 10 |
| 11 | WriteEnable | 11 |
| 12 | ReadSR | 12 |

Status Register

| bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 |
|---|---|---|---|---|---|---|---|
| Reserved | Reserved | BP3 (level of protected block) | BP2 (level of protected block) | BP1 (level of protected block) | BP0 (level of protected block) | WEL (write enable latch) | WIP (write in progress bit) |
| Reserved | Reserved | (note 1) | (note 1) | (note 1) | (note 1) | 1=write enable 0=not write enable | 1=write operation 0=not in write operation |
| Reserved | Reserved | Non-volatile bit | Non-volatile bit | Non-volatile bit | Non-volatile bit | volatile bit | volatile bit |

Figure 45. **Set container for status register**

## 6.2.2.4. Set container for switch to OPI mode

The MX25UW51245G product provides both Volatile and Non-Volatile configuration registers to set the device operation condition in CR2 (Configuration Register 2). The Volatile configuration register bits to temporarily change the device operation. Volatile-CR2 bits can be set to change the setting originally set by NV-CR2 bits. For example, changing the operation mode of flash from single-wire SPI mode to OPI mode.

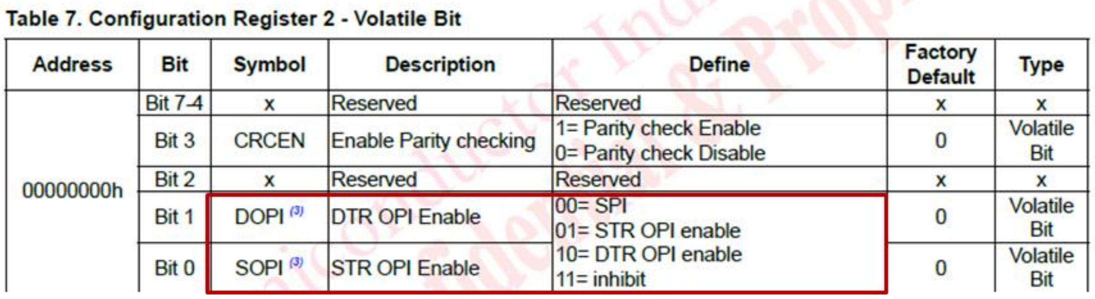


Table 7. Configuration Register 2 - Volatile Bit

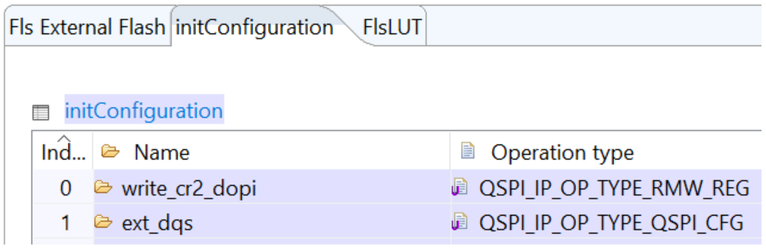
| Address | Bit | Symbol | Description | Define | Factory Default | Type |
|---|---|---|---|---|---|---|
| 00000000h | Bit 7-4 | x | Reserved | Reserved | x | x |
| | Bit 3 | CRCEN | Enable Parity checking | 1= Parity check Enable<br>0= Parity check Disable | 0 | Volatile Bit |
| | Bit 2 | x | Reserved | Reserved | x | x |
| | Bit 1 | DOPI (3) | DTR OPI Enable | 00= SPI<br>01= STR OPI enable<br>10= DTR OPI enable<br>11= inhibit | 0 | Volatile Bit |
| | Bit 0 | SOPI (3) | STR OPI Enable | | 0 | Volatile Bit |

Figure 46. **Set container for switch to OPI mode**

To clear the volatile-CR2 setting, users can initiate a power-on action or reset cycle, and the device returns to the default status set by NV-CR2 bits. Refer to Set container for reset command for more details about the reset cycle.

In this example, the following configurations make the QuadSPI controller and external flash switch to the OPI-DTR mode:

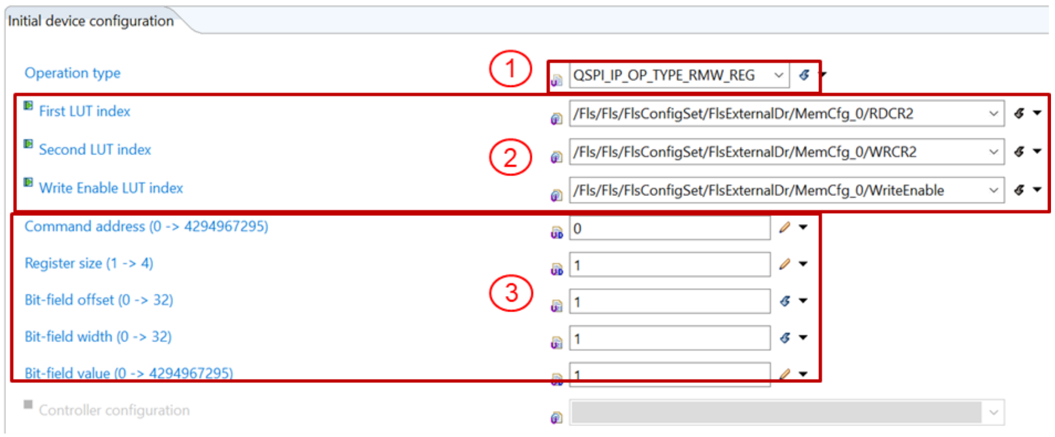
- write_cr2_dopi: It makes the external flash device switch to the OPI mode
- ext_dqs: It makes the QSPI controller of S32G switch to the ControllerCfg_1 which is the OPU mode.


Fls External Flash | initConfiguration | FlsLUT

initConfiguration

| Ind... | Name | Operation type |
|---|---|---|
| 0 | write_cr2_dopi | QSPI_IP_OP_TYPE_RMW_REG |
| 1 | ext_dqs | QSPI_IP_OP_TYPE_QSPI_CFG |

Figure 47. **Init configuration**


Initial device configuration

| | |
|---|---|
| Operation type | QSPI_IP_OP_TYPE_RMW_REG |
| First LUT index | /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_0/RDCR2 |
| Second LUT index | /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_0/WRCR2 |
| Write Enable LUT index | /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_0/WriteEnable |
| Command address (0 -> 4294967295) | 0 |
| Register size (1 -> 4) | 1 |
| Bit-field offset (0 -> 32) | 1 |
| Bit-field width (0 -> 32) | 1 |
| Bit-field value (0 -> 4294967295) | 1 |
| Controller configuration | |

Figure 48. **Initial device configuration**

1. The following can be one of the operation type:
   - QSPI_IP_OP_TYPE_CMD - Simple command
   - QSPI_IP_OP_TYPE_WRITE_REG - Write value in external flash register
   - QSPI_IP_OP_TYPE_RMW_REG - RMW command on external flash register QSPI_IP_OP_TYPE_READ_REG - Read external flash register until expected value is read
   - QSPI_IP_OP_TYPE_QSPI_CFG - Re-configure QSPI controller
2. This describes the list of operations which must be performed at initialization phase to bring the memory in the desired operating state, like from SPI to OPI mode.
3. The detail information of CR2 provided to the Fls driver helps to enable the DTR OPI mode.

#### 6.2.2.5. Set container for the reset command

Container related to software reset command, for resetting the flash device. This reset procedure applies only at driver initialization. It might be different from the normal reset command, depending on the initial state of the flash. For example, the mode external flash may be changed from SPI mode to OPI mode which is modified by the reconfiguration parameter.

The Software Reset operation combines two instructions, Reset-Enable (RSTEN) command following a Reset (RST) command. It returns the device to standby mode. All the volatile bits and settings will be cleared then, which makes the device return to the default status as power on.

**Figure 129. Reset Sequence (DTR-OPI mode)**

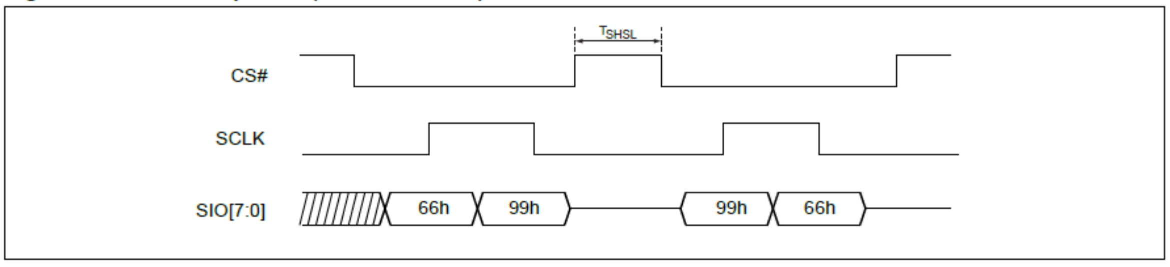
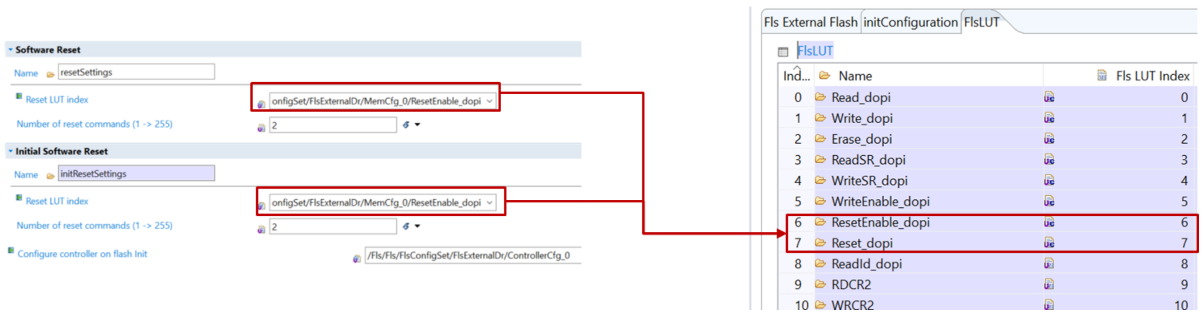


Figure 49. **Reset sequence**

Reference to the configuration which will be used for initializing the controller when the flash device is initialized. This is needed for devices that need to change controller configuration during device initialization. In this example, after software resetting, the flash device returns to a standby mode which is the SPI mode. The QSPI re-apply the configuration for the flash device of standby mode.



**Software Reset**

| | |
|---|---|
| Name | resetSettings |
| Reset LUT index | onfigSet/FlsExternalDr/MemCfg_0/ResetEnable_dopi |
| Number of reset commands (1 -> 255) | 2 |

**Initial Software Reset**

| | |
|---|---|
| Name | initResetSettings |
| Reset LUT index | onfigSet/FlsExternalDr/MemCfg_0/ResetEnable_dopi |
| Number of reset commands (1 -> 255) | 2 |
| Configure controller on flash Init | /Fls/Fls/FlsConfigSet/FlsExternalDr/ControllerCfg_0 |

Fls External Flash | initConfiguration | FlsLUT

FlsLUT

| Ind... | Name | Fls LUT Index |
|---|---|---|
| 0 | Read_dopi | 0 |
| 1 | Write_dopi | 1 |
| 2 | Erase_dopi | 2 |
| 3 | ReadSR_dopi | 3 |
| 4 | WriteSR_dopi | 4 |
| 5 | WriteEnable_dopi | 5 |
| 6 | ResetEnable_dopi | 6 |
| 7 | Reset_dopi | 7 |
| 8 | ReadId_dopi | 8 |
| 9 | RDCR2 | 9 |
| 10 | WRCR2 | 10 |

Figure 50. **Configure the reset sequence in FLS driver**

### 6.2.3. FLS controller configuration

The FLS driver provides services for reading, writing, and erasing flash memory and it combines configured flash memory sectors into one linear address space. The FLS module combines all available flash memory areas into one linear address space. It always start at address 0 and continue without any gap.

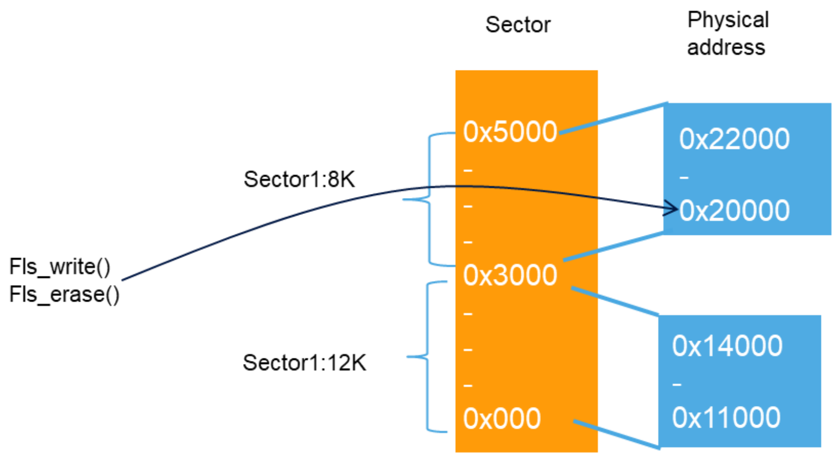
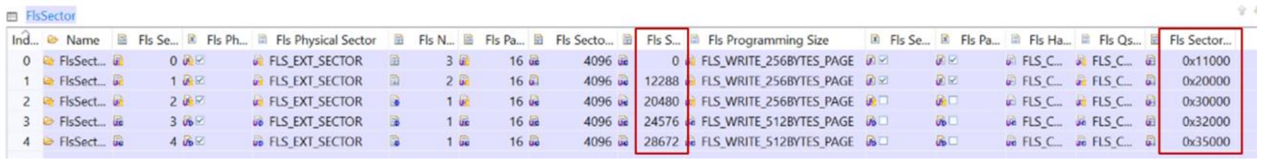


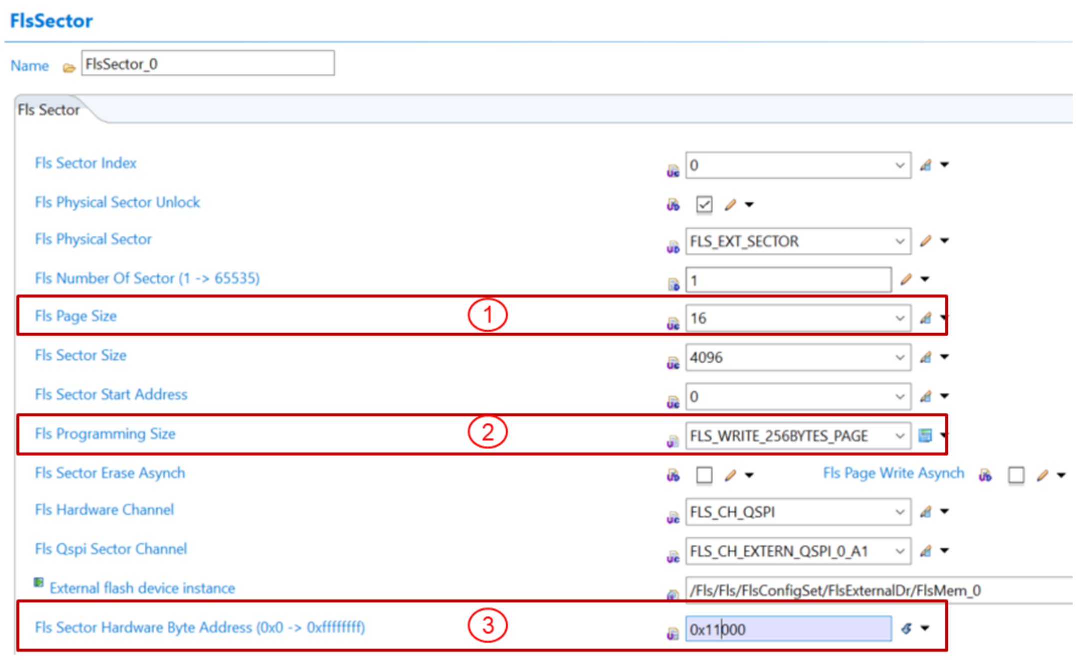
Figure 51. **Flash memory sectors**

Fls Sector Start Address for FlsSector_0 will be 0x0000 and Fls Sector Start Address for FlsSector_1 will be 0x1000 (4096). If user want to write FlsSector_1, user need to write to the logical address 0x1000 - 0x1FFF. If user want to erase it, user need to erase sector from address 0x1000 with size 0x1000.


FlsSector

| Ind... | Name | Fls Se... | Fls Ph... | Fls Physical Sector | Fls N... | Fls Pa... | Fls Secto... | Fls S... | Fls Programming Size | Fls Se... | Fls Pa... | Fls Ha... | Fls Qs... | Fls Sector... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | FlsSect... | 0 | ✓ | FLS_EXT_SECTOR | 3 | 16 | 4096 | 0 | FLS_WRITE_256BYTES_PAGE | ✓ | ✓ | FLS_C... | FLS_C... | 0x11000 |
| 1 | FlsSect... | 1 | ✓ | FLS_EXT_SECTOR | 2 | 16 | 4096 | 12288 | FLS_WRITE_256BYTES_PAGE | ✓ | ✓ | FLS_C... | FLS_C... | 0x20000 |
| 2 | FlsSect... | 2 | ✓ | FLS_EXT_SECTOR | 1 | 16 | 4096 | 20480 | FLS_WRITE_256BYTES_PAGE | ☐ | ☐ | FLS_C... | FLS_C... | 0x30000 |
| 3 | FlsSect... | 3 | ✓ | FLS_EXT_SECTOR | 1 | 16 | 4096 | 24576 | FLS_WRITE_512BYTES_PAGE | ☐ | ☐ | FLS_C... | FLS_C... | 0x32000 |
| 4 | FlsSect... | 4 | ✓ | FLS_EXT_SECTOR | 1 | 16 | 4096 | 28672 | FLS_WRITE_512BYTES_PAGE | ☐ | ☐ | FLS_C... | FLS_C... | 0x35000 |

Figure 52. **Flash memory sectors in FLS driver**


FlsSector

Name: FlsSector_0

Fls Sector

| | |
|---|---|
| Fls Sector Index | 0 |
| Fls Physical Sector Unlock | ✓ |
| Fls Physical Sector | FLS_EXT_SECTOR |
| Fls Number Of Sector (1 -> 65535) | 1 |
| Fls Page Size ① | 16 |
| Fls Sector Size | 4096 |
| Fls Sector Start Address | 0 |
| Fls Programming Size ② | FLS_WRITE_256BYTES_PAGE |
| Fls Sector Erase Asynch | ☐ |
| Fls Page Write Asynch | ☐ |
| Fls Hardware Channel | FLS_CH_QSPI |
| Fls Qspi Sector Channel | FLS_CH_EXTERN_QSPI_0_A1 |
| External flash device instance | /Fls/Fls/FlsConfigSet/FlsExternalDr/FlsMem_0 |
| Fls Sector Hardware Byte Address (0x0 -> 0xffffffff) ③ | 0x11000 |

Figure 53. **Configure flash memory sectors in FLS driver**

1. In MX25UW51245G, the ECC algorithm uses a Hamming code that can correct a single bit error per 16-Byte page. It is recommended that data be programmed in multiples of 16 bytes using the Page Program command instead of programming a byte or a word at a time using the Program command. Each group of 16 bytes must fall within the same 16-Byte boundary
2. Page size is the maximum amount of data that the flash device can write in a single write operation(Maximum 256 bytes in MX25UW51245G )
3. The hardware address of this sector, as needed by the external flash device is applicable only to external sectors. This value is used to access the hardware sector on the attached device and will be sent as a parameter of flash commands. It should be completed to meet the requirements of the external flash memory type and configured operating mode.

# 7. QuadSPI debug

## 7.1. Waveform analysis example – waveform decoding(read ID)

After initializing the clock, pins, and registers of the QuadSPI controller, you can use the RDID instruction shown in the following figure to read the external flash ID in order to determine whether the specific flash sequence is correct.

**Figure 11. Read Identification (RDID) Sequence (SPI mode)**

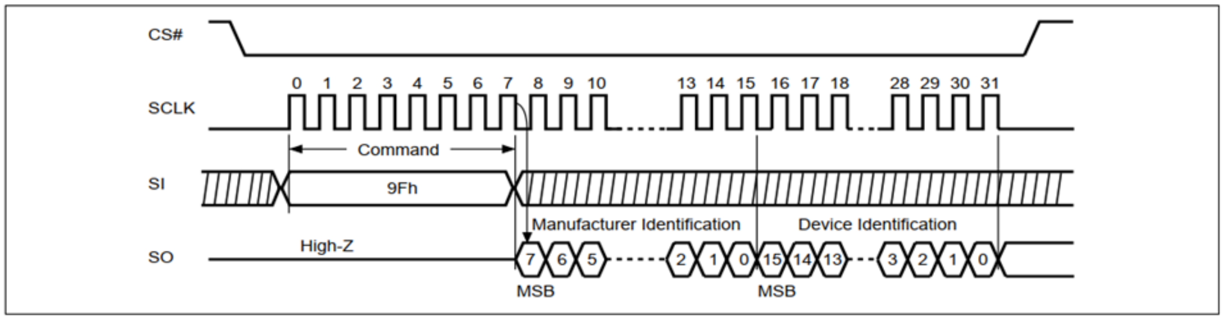


Figure 54. **Read identification sequence in SPI mode**

From the waveform captured by the oscilloscope, you can analyze the waveform after sending the read ID command with SPI mode. The readout data from the external flash device are 0xC2, 0x81, and 0x3A which match with the expected ID in the MX25UW51245G flash device datasheet.

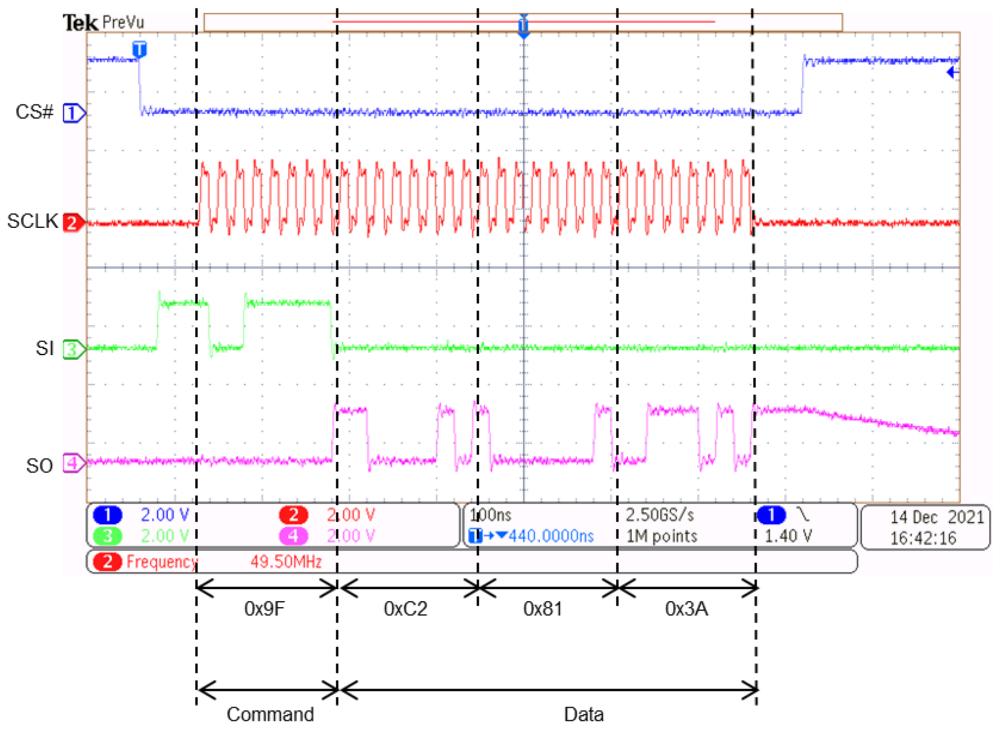


Figure 55. **Waveform decoding – Read ID**

## 7.2. Waveform analysis example – fast read command

According to the MX25UW51245G flash datasheet, the max value of the READ3B command (0x03) clock frequency is 66 MHz. While the FAST_READ3B (0x0B) can be up to 133 MHz, mainly because of the addition of the dummy cycle.

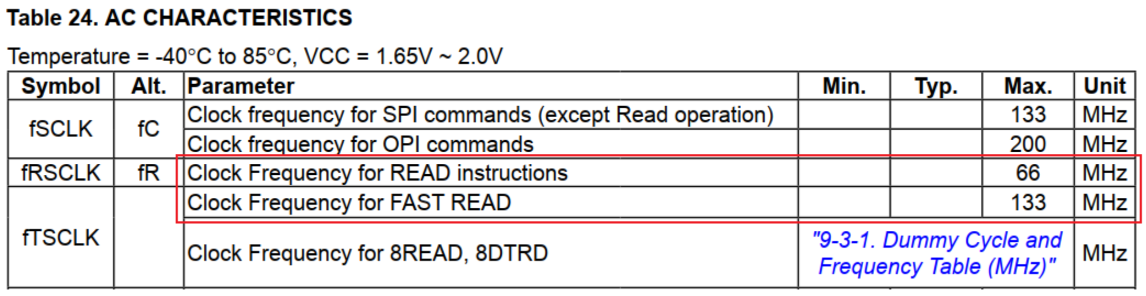


**Table 24. AC CHARACTERISTICS**

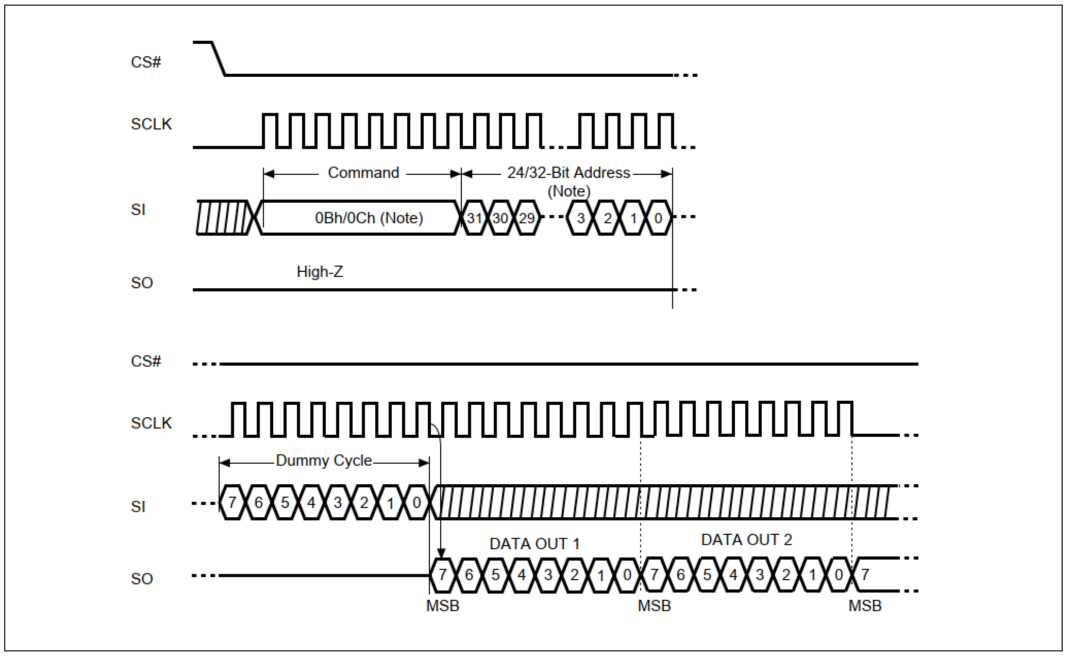
Temperature = -40°C to 85°C, VCC = 1.65V ~ 2.0V

| Symbol | Alt. | Parameter | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| fSCLK | fC | Clock frequency for SPI commands (except Read operation) | | | 133 | MHz |
| | | Clock frequency for OPI commands | | | 200 | MHz |
| fRSCLK | fR | Clock Frequency for READ instructions | | | 66 | MHz |
| fTSCLK | | Clock Frequency for FAST READ | | | 133 | MHz |
| | | Clock Frequency for 8READ, 8DTRD | *"9-3-1. Dummy Cycle and Frequency Table (MHz)"* | | | MHz |

Figure 56. **AC characteristics**

From the FAST_READ sequence shown in the following figure, you can know that after sending the FAST_READ command 0x0B/0x0C, a proper dummy cycle is needed. Incorrect dummy cycles will cause error data when using the FAST_READ3B command.

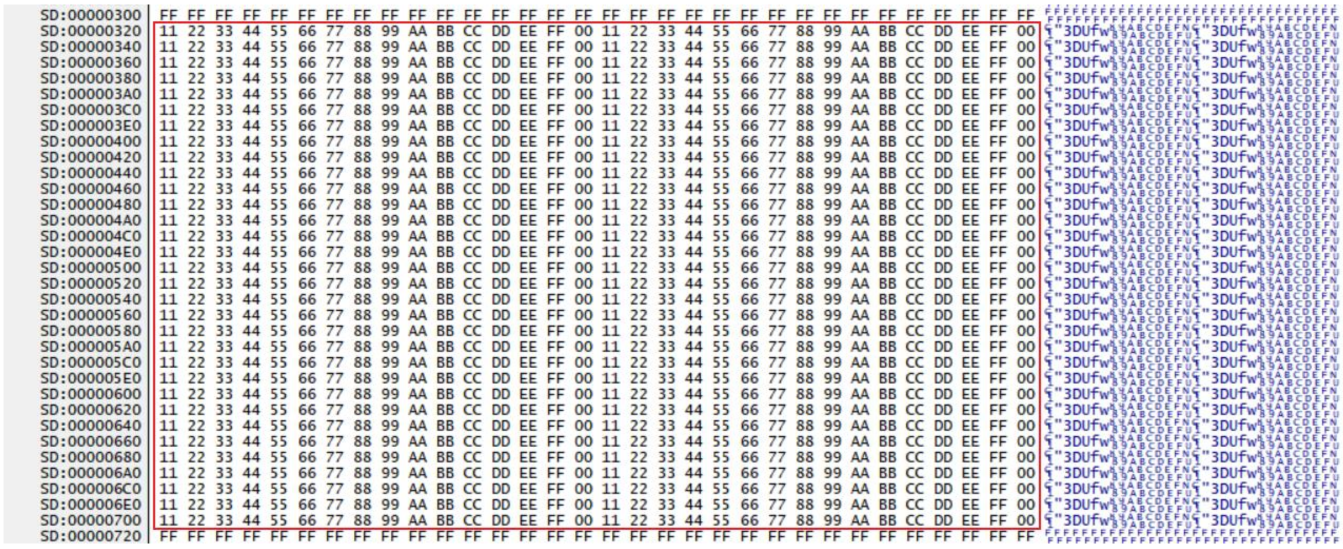
**Figure 41. Read at Higher Speed (FAST_READ/FAST_READ3B/FAST_READ4B) Sequence (SPI Mode only)**



Note: The number of address cycles are based on different address mode. In 3-Byte command operation, it is 24-bit. In 4-Byte command operation, it is 32-bit.

Figure 57. **Fast read sequence**

To test the FAST_READ3B (0x0B) command, write 1024 bytes test data into external flash at the start address 0x320. The specific data content is shown in the following figure.



```
SD:00000300 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
SD:00000320 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000340 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000360 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000380 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:000003A0 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:000003C0 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:000003E0 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000400 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000420 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000440 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000460 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000480 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:000004A0 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:000004C0 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:000004E0 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000500 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000520 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000540 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000560 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000580 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:000005A0 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:000005C0 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:000005E0 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000600 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000620 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000640 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000660 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000680 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:000006A0 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:000006C0 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:000006E0 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000700 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 ."3DUfw.ª»ÌÝîÿ.."3DUfw.ª»ÌÝîÿ.
SD:00000720 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
```

Figure 58. **The test data**

When using the dummy cycles specified in the MX25UW51245G flash datasheet (dummy(8)), the read data is the same as the test data. The data is: 0x11, 0x22, 0x33...

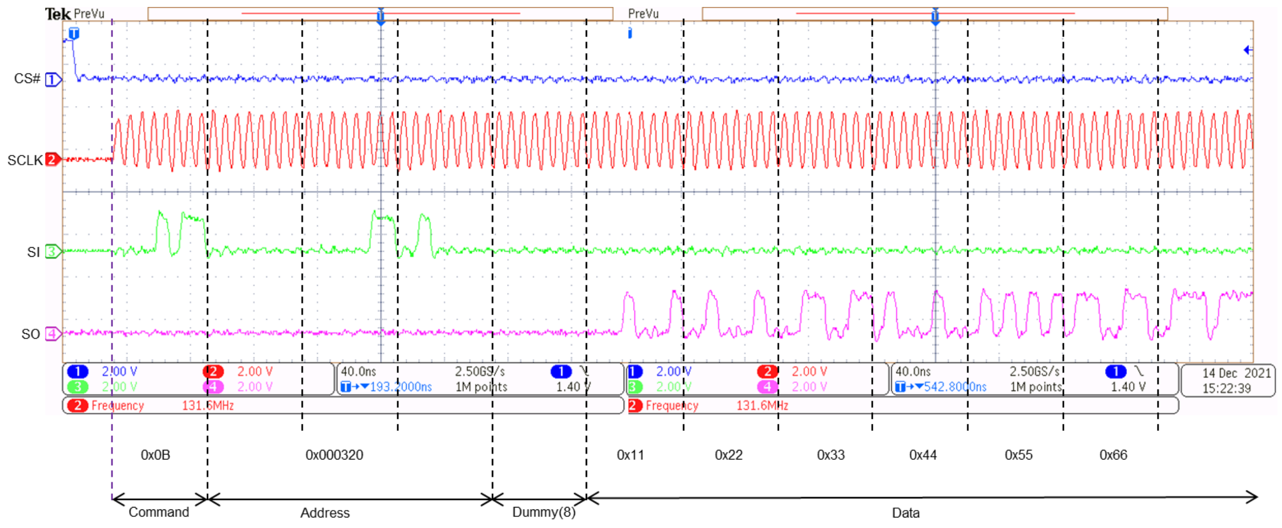


Figure 59. **Waveform decoding – fast read with dummy(8)**

While the dummy cycle(dummy(9)) used is different from the one specified in the MX25UW51245G flash datasheet, the read data is different from the test data. The data is: 0x22, 0x44, 0x66...

The specific physical layer data content is shown in the following figure.

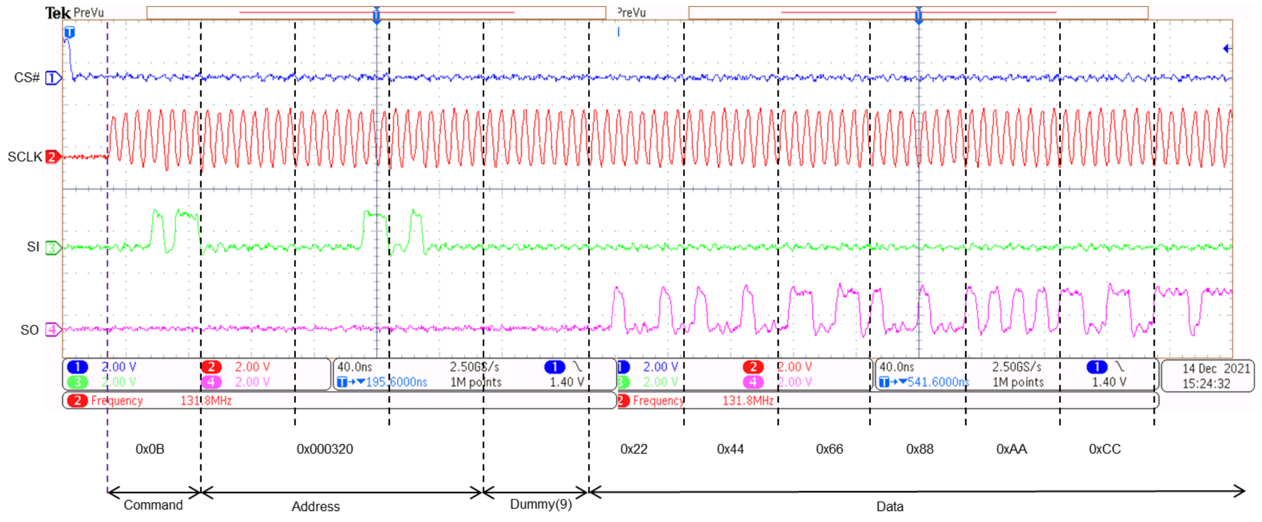


Figure 60. **Waveform decoding – fast read with dummy(9)**

## 7.3. Some debug tips for QuadSPI

- Please pay attention to the QuadSPI pin configuration, such as whether the differential clock is required from the external flash device, whether the interrupt pin is in low-level input, etc.
- Ensure clock settings meet the clock restrictions mentioned in the QuadSPI Clock configuration section.

- Due to BootROM setting the 500ms timeout restriction when downloading the application image, downloading a very large image with default settings is not optimal. It is recommended to use the reconfiguration parameter to increase the flash speed.
- A Lauterbach script can be helpful to read the flash device ID for quick verification, if available.
  - The LB  script can be obtained from the following location after installing the trace32 debug tool: C:\T32\demo\arm\flash\s32g274-qspi.cmm
- Use an oscilloscope to check whether the timing of the physical layer meets the requirements of the specific flash chip datasheet.

## 8. References

1. S32G Reference Manual
2. S32G Data Sheet
3. AN12808 - QSPI Timing Configuration

***How to Reach Us:***

**Home Page:**
nxp.com

**Web Support:**
nxp.com/support

Document Number: AN13563
Rev. 0
02/2022