Rev. 2, 8/2024

S32G Add GD QSPI NOR Support

by John.Li NXA08200

This article describes the GD QSPI NOR flash support on the S32G platform. The test platform is:

- S32G3 RDB3+GD25LX256E 32MB QSPI NOR flash.

G2 and G3 are basically the same in terms of QSPI NOR controller, so this article should also be applicable to G2 platform.

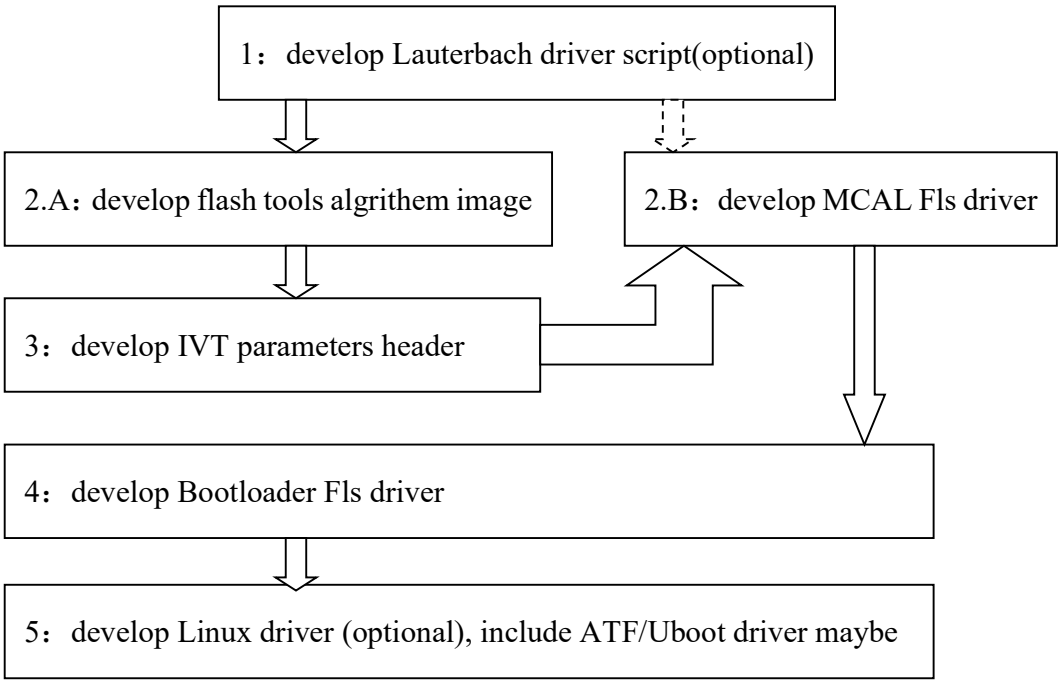
Ver.	History	Author
V1	● Created this doc	JohnLi
V2	● Traslate to Eng.	JohnLi

Content

1 Background and References

1.1 Background

This article takes GD GD25LX256E as an example to illustrate how to replace a new QSPI NOR flash on the S32G platform. In addition to hardware connection, the software development process includes:



The description is as follows:

1. You can use Lauterbach script to drive QSPI NOR. There are two main application scenarios:

- When the board is brought up, the simplest ID reading driver of the Lauterbach script is used to verify the hardware.
- When QSPI NOR fails to start for mass production products, Lauterbach script can be used to simulate the behavior of ROM code reading QSPI NOR.

Another situation is:

- Use Lauterbach script to directly realize burning image, but this requires Lauterbach company to provide driver script and algorithm image.

Lauterbach is used for debugging purposes, so it is not a part that must be developed, it is optional.

2. Since the flash tool is required to burn the image, the QSPI NOR flash algorithm image used by the flash tool needs to be developed first.
3. You can also use Lauterbach to debug the MCAL Fls driver and its test code directly.

4. The IVT QSPI NOR flash parameter header needs to be developed, so that the image burned in can be quickly started.
5. Generally, the first image to start is the bootloader. The bootloader requires the support of the Fls driver, which is mostly the same as the Fls driver in Mcal.
6. Generally speaking, only M7 can access QSPI NOR flash through MCAL or bootloader, so the support of a QSPI NOR flash will be completed after completing the above two to five steps. However, some customers will consider using Linux kernel in the production line to quickly burn QSPI NOR flash, so they need to complete the Linux drive (optional).
7. Because most customers use Bootloader to load BL2 of ATF, while BL2 loads the rest of ATF from eMMC, then ATF loads uboot from eMMC, and uboot loads the kernel, it is generally unnecessary to implement ATF/Uboot QSPI NOR flash drive (as the most unlikely option).

1.2 References

This article is developed based on S32G3 RDB3 board+GD25LX256E QSPI NOR flash:

Catalog	Name	Catalog2	Comments
Doc	S32G2RM.pdf S32G3RM.pdf	S32G RM	Download from www.nxp.com/s32g
SW RTD Mcal	SW32G_RTD_4.4_4.0.2	BSP SW+doc	Dowload from www.nxp.com Personal account
SW Boot loader	Platform_Software_Integration_S32G3_2023_02.exe	BSP SW+doc	Dowload from www.nxp.com Personal account
Doc	S32G_Bootloader_V*.pdf	Bootloader Customization Doc	https://community.nxp.com/ t5/NXP-Designs-Knowledge-Base/ S32G-Bootloader-Customzition/ ta-p/1519838
SW Linux	BSP37	BSP Doc	Dowload from www.nxp.com Personal account
Tools	S32Design Studio 3.4.3 or 3.5.3	S32DS	Dowload from www.nxp.com Personal account : Refer to the QuadSPI configuration tool, compile Flash_SDK, and Flash tool.
Doc	AN13563: S32G QuadSPI Deep Dive Application note	AppNotes	Download from www.nxp.com/s32g Some contents of this article overlap with it
Doc	S32G_RTD_MCAL_V*.pdf	AppNotes	https://community.nxp.com/t5/ NXP-Designs-Knowledge-Base/

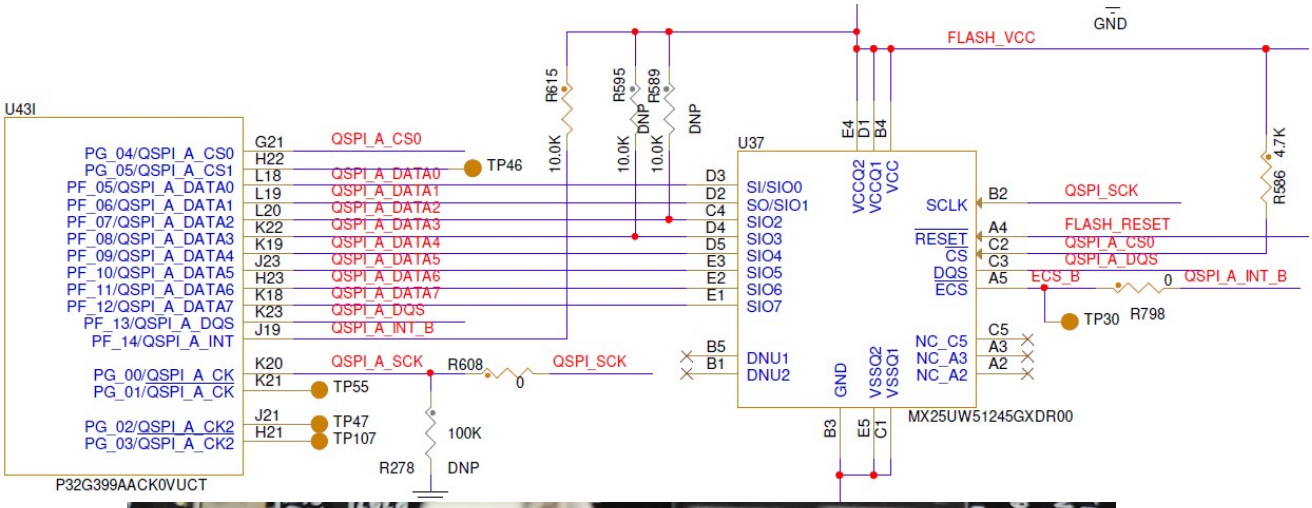
			S32G-MCAL-customization-application-doc /ta-p/1399899 RTD MCAL driver sample customization doc
Doc	AN12808: Quad SPI (QSPI) Timing Configuration on the S32G2 Vehicle Network Processor Application Note	AppNotes	Download from www.nxp.com/s32g Please refer to this document for register configuration in high-speed mode.
Doc	S32G_QSPINOR_Customization_*.pdf	AppNotes	从nxp community 下载 https://community.nxp.com/t5/ NXP-Designs-Knowledge-Base/ S32G-QSPI-Nor-customization-doc/t a-p/1399906 For the configuration of flash timing header, Flash tools SDK project customization (used to develop the QSPI NOR binary of flash tools) For uboot customization and kernel driver customization, please refer to this document. Some contents of this article overlap with it
Doc	MX25UW51245G.pdf	Macronix QSPI NOR datasheet	
Doc	DS-00762-GD25LX256E-Rev1.1_Automotive.pdf	GD QSPI NOR datasheet	Get the support from GD
Doc	S32G_How_to_Develop_QSPI_Script_*.pdf	QSPI Lauterbach Script deveopment doc	https://community.nxp.com/t5/ NXP-Designs-Knowledge-Base/ S32G-QSPI-Nor-customization-doc /ta-p/1399906 Some contents of this article overlap with it

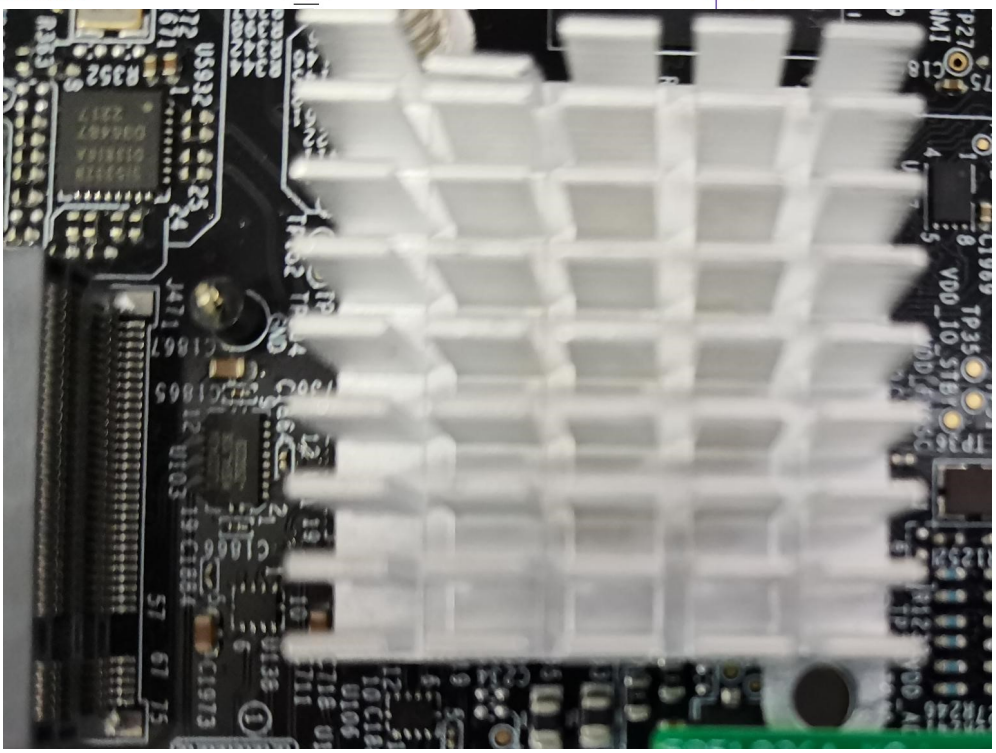
Note: Since this article develops each QSPI NOR flash related driver for the purpose of independent testing, it does not consider the problem of matching all software versions. For officially developed software version matching, it is recommended to use bundle release:

www.nxp.com/s32g->S32G3->Design Resources->Software->Automotive Software Package Manager->DOWNLOAD->input the account->S32G3->Integrated Software Bundle.

1.3 Hardware Link

The schematic diagram of S32G3 RDB3 connecting QSPI NOR is as follows: The MACRONIX MX25UW51245G flash is used. Flash is generally designed to be pin to pin compatible, so we directly replace it with GD25LX256E.





For hardware design related notes, please refer to section 3.1: Pin configuration of the document <<AN13563.pdf: S32G QuadSPI Deep Dive>>.

On the software, NXP default release software already supports MX25UW51245G, so this article will describe the software modification process by comparing it with GD25LX256E. At the same time, sometimes refer to Micron MT35XU256 (512) ABA.

2 Lauterbach Script development(Optional)

Refer to the document <<S32G_How_to_Development_QSPI_Script_ *. Pdf>> to learn how to develop the Lauterbach script driver. This paper compares the different configurations between the two Flash models, mainly the LUT configurations. Consider two functions:

- QuadSPI_ReadID
- After switching to DOPI mode (QuadSPI_InitDOPI_DLL_AutoUpdateMode_100MHz), quickly read QSPI NOR flash: QuadSPI_Read32BytesDOPI

2.1 Preparing the refer script

Copy C:\NXP\SW32G_RTD_4.4_4.0.2\eclipse\plugins\Fls_TS_T40D11M40I2R0\examples\EBT\S32G3\Fls_Example_S32G399A_M7\debug\device.cmm twice,Modify to device_gd_readid.cmm and device_gd.cmm. Remove the parts irrelevant to M7_0 startup, Disable WDG and QSPI NOR.

2.2 QuadSPI_ReadID

device_gd _readid.cmm main function is:

```
GOSUB PERIPH_PLL
GOSUB PERIPH_DFS1_QSPI_66MHz
GOSUB QuadSPI_PinMux_CLKEnable
GOSUB QuadSPI_Init
GOSUB QuadSPI_ReadID
```

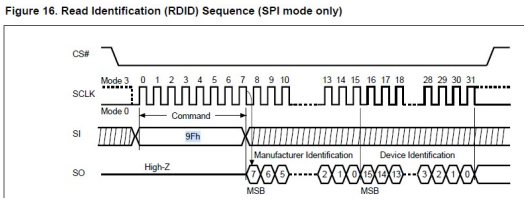
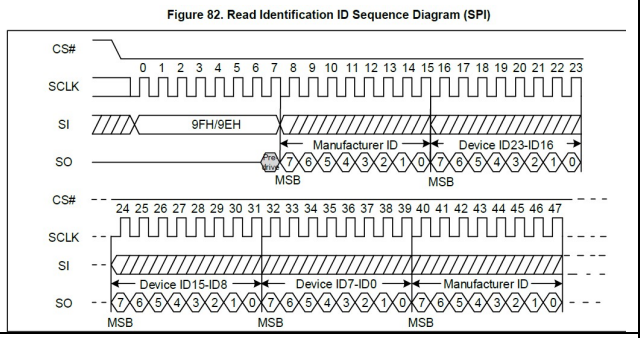
In addition, because BYTE SWAP is different, the following codes need to be modified:

```
    ; write sequence ID and assert Read id command
Data.Set A:&QSPI_Cntl_BASE+0x08 %Long  (5.<<24.)  ; LUT25  and sequence
PRINT "1st 0x" Data.Long(A:&QSPI_Cntl_BASE+0x200)>>24. " (Density)"
PRINT "2nd 0x" (Data.Long(A:&QSPI_Cntl_BASE+0x200)>>16.)&0xFF " (Device ID)"
PRINT "3rd 0x" (Data.Long(A:&QSPI_Cntl_BASE+0x200)>>8.)&0xFF  " (Manufacture)"
PRINT "4th 0x" Data.Long(A:&QSPI_Cntl_BASE+0x200)&0xFF
```

QuadSPI_ReadID function call: (Note that according to JEDEC requirements, all QSPI NOR flash manufacturer's ReadID commands should be the same).

```
|->
; write sequence ID and assert Read id command
  Data.Set A:&QSPI_Cntl_BASE+0x08 %Long  (5.<<24.)  ; LUT20  and sequence
```

Refer to MX25UW51245G design GD25LX256E command sequence：

	MX25U51245G(Reference)				GD25LX256E(Design)			
1：Lauterbach code	;Program LUT25 with READ_ID Data.Set A:&QSPI_Cntl_BASE+0x374 %LE %Long 0x0818049F ; SEQID 5 Data.Set A:&QSPI_Cntl_BASE+0x378 %LE %Long 0x00001C03 Data.Set A:&QSPI_Cntl_BASE+0x37C %LE %Long 0x0				;Program LUT25 with READ_ID Data.Set A:&QSPI_Cntl_BASE+0x374 %LE %Long 0x0818049F ; SEQID 5 Data.Set A:&QSPI_Cntl_BASE+0x378 %LE %Long 0x00001C03 //0x00001C04 Data.Set A:&QSPI_Cntl_BASE+0x37C %LE %Long 0x0			
Details		Instr(6bits)	Pads(2bits)	Operand(8bits)		Instr(6bits)	Pads(2bits)	Operand(8bits)
	049f	0x01(CMD)	0x0(1 bit)	0x9F(RDID)	049f	0x01(CMD)	0x0(1 bit)	0x9F/0x9E(RDID)
	0818	0x2(ADDR)	0x0(1 bit)	0x18(24 Addr bits to be sent on 1 pad)	0818	0x2(ADDR)	0x0(1 bit)	0x18(24 Addr bits to be sent on 1 pad)
	1c03	0x7(READ)	0x0(1 bit)	0x3 write data size in byte	1c04 1c03	0x7(READ)	0x0(1 bit)	0x4 write data size in byte Considering compatibility, you can also read only 3 bytes
Timing diagram	Figure 16. Read Identification (RDID) Sequence (SPI mode only) 				Figure 82. Read Identification ID Sequence Diagram (SPI) 			
Comments	The RDID instruction is for reading the manufacturer ID of 1-byte and followed by				The Read Identification (RDID) command allows the 8-bit manufacturer identification to be read, followed by			

<table>
<tr><td></td><td>Device ID of 2-byte
Table 11. ID Definitions
<table><tr><th colspan="2">Command Type</th><th colspan="3">MX25U51245G</th></tr><tr><td>RDID</td><td>9Fh</td><td>Manufacturer ID
C2</td><td>Memory type
25</td><td>Memory density
3A</td></tr></table></td><td>three Bytes of device identification. The device identification indicates the memory type in the first Byte, and the memory capacity of the device in the second Byte</td></tr>
<tr><td>Print</td><td colspan="2">PRINT "1st 0x" Data.Long(A:&QSPI_Cntl_BASE+0x200)>>24. " (Density)"
PRINT "2nd 0x" (Data.Long(A:&QSPI_Cntl_BASE+0x200)>>16.)&0xFF " (Device ID)"
PRINT "3rd 0x" (Data.Long(A:&QSPI_Cntl_BASE+0x200)>>8.)&0xFF " (Manufacture)"
PRINT "4th 0x" Data.Long(A:&QSPI_Cntl_BASE+0x200)&0xFF</td></tr>
</table>

2.3 Configure QSPI NOR to DOPI mode

device_gd.cmm main function call:

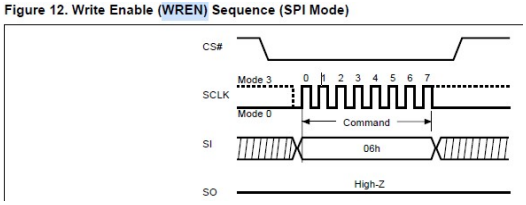
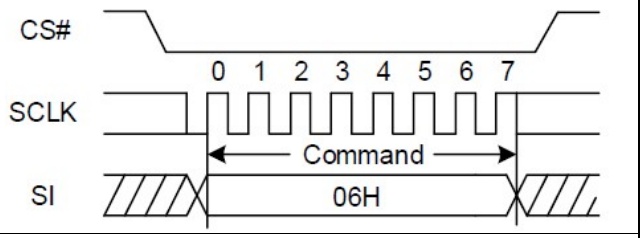
```
GOSUB PERIPH_PLL_1600MHZ
GOSUB PERIPH_PLL_DFS1_800MHZ
GOSUB QuadSPI_PinMux_CLKEnable
GOSUB QuadSPI_InitDOPI_DLL_AutoUpdateMode_100MHz
GOSUB QuadSPI_Read32BytesDOPI
```

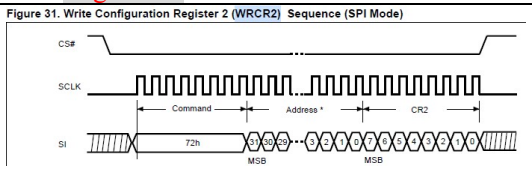
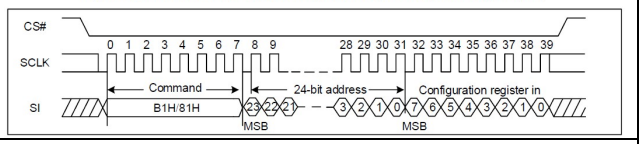
QuadSPI_InitDOPI_DLL_AutoUpdateMode_100MHz, call:

```
|-> ; write sequence ID and assert WriteEnable id command
  Data.Set A:&QSPI_Cntl_BASE+0x08 %Long  (2.<<24.)  ; sequence
|->
 ; We assume we are after a reset, in SPI mode 1X SDR
 Data.Set A:&QSPI_Cntl_BASE+0x154 %LE %Long 0x00000002 //TX Buffer Data Regsiter=2
; Program LUT60 Write CONFIG2 REGISTER - SPI mode with value to switch to DOPI mode. From this point on, all LUT seqs should be DDR OPI mode compatible
  Data.Set A:&QSPI_Cntl_BASE+0x08 %Long  (12.<<24.)  ; sequence
```

Refer MX25UW51245G design GD25LX256E Command sequence:

<table>
<tr><td></td><td colspan="4">MX25U51245G(Reference)</td><td colspan="4">GD25LX256E(Design)</td></tr>
<tr><td>Lauterbach code (writeenable)</td><td colspan="4">;Program LUT10 with WRITE_ENABLE
Data.Set A:&QSPI_Cntl_BASE+0x338 %LE %Long 0x00000406 ; SEQID 2
Data.Set A:&QSPI_Cntl_BASE+0x33C %LE %Long 0x0</td><td colspan="4">;Program LUT10 with WRITE_ENABLE
Data.Set A:&QSPI_Cntl_BASE+0x338 %LE %Long 0x00000406 ; SEQID 2
Data.Set A:&QSPI_Cntl_BASE+0x33C %LE %Long 0x0</td></tr>
<tr><td>Details</td><td></td><td>Instr(6bits</td><td>Pads(</td><td>Operand(8bits)</td><td></td><td>Instr(6bits)</td><td>Pads(</td><td>Operand(8bits)</td></tr>
</table>

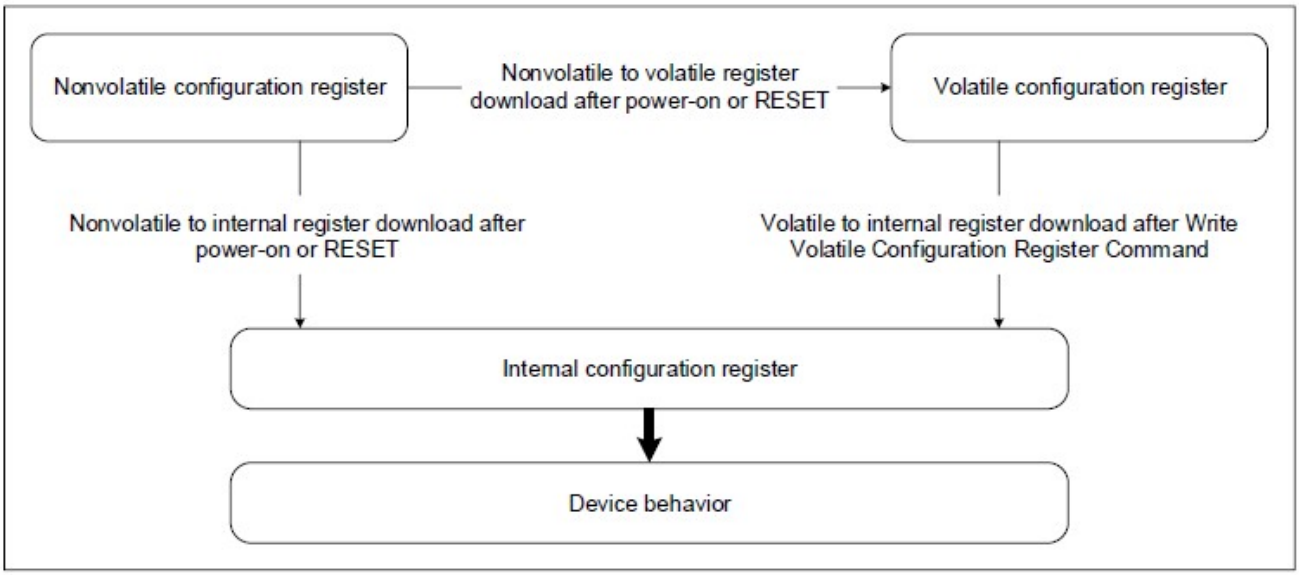
		)	2bits)				2bits)	
	0406	0x01(CMD)	0x0(1 bit)	0x06(WREN)	0406	0x01(CMD)	0x0(1 bit)	0x06(WREN) same
Timing diagram	Figure 12. Write Enable (WREN) Sequence (SPI Mode)  				Figure 16. Write Enable Sequence Diagram (SPI)  			
Comments	The Write Enable (WREN) instruction is for setting Write Enable Latch (WEL) bit. For those instructions like PP/ PP4B, 4PP/4PP4B, SE/SE4B, BE32K/BE32K4B, BE/BE4B, CE, and WRSR, which are intended to change the device content WEL bit should be set every time after the WREN instruction setting the WEL bit.				The Write Enable (WREN) command is for setting the Write Enable Latch (WEL) bit. The Write Enable Latch (WEL) bit must be set prior to every Page Program (PP), Sector Erase (SE), Block Erase (BE), Chip Erase (CE), Write Status Register (WRSR), Write Extended Address Register (WEAR), Write Nonvolatile/Volatile configure register and Erase/Program Security Registers command.			
Lauterbach code (switch to DOPI mode)	;Program LUT60 Write CONFIG2 REGISTER - SPI mode Data.Set A:&QSPI_Cntl_BASE+0x400 %LE %Long 0x08200472 ; SEQID 12 Data.Set A:&QSPI_Cntl_BASE+0x404 %LE %Long 0x00002001 Data.Set A:&QSPI_Cntl_BASE+0x408 %LE %Long 0x00000000				;Program LUT60 Write CONFIG2 REGISTER - SPI mode Data.Set A:&QSPI_Cntl_BASE+0x400 %LE %Long 0x08180481 ; SEQID 12 Data.Set A:&QSPI_Cntl_BASE+0x404 %LE %Long 0x00002001 Data.Set A:&QSPI_Cntl_BASE+0x408 %LE %Long 0x00000000			
Details		Instr(6bits)	Pads(2bits)	Operand(8bits)		Instr(6bits)	Pads(2bits)	Operand(8bits)
	0472	0x01(CMD)	0x0(1 bit)	0x72(WRCR2)	0481	0x01(CMD)	0x0(1 bit)	0xB1/81(WRCR)
	0820	0x02(ADDR)	0x0(1 bit)	0x20(32 Addr bits to be sent on 1 pad)	0818	0x02(ADDR)	0x0(1 bit)	0x18(24 Addr bits to be sent on 1 pad)
	2001	0x8(WRITE)	0x0(1 bit)	0x01 write data size in byte	2001	0x8(WRITE)	0x0(1 bit)	0x01 write data size in byte
Write value code:	; We assume we are after a reset, in SPI mode 1X SDR Data.Set A:&QSPI_Cntl_BASE+0x154 %LE %Lo				; We assume we are after a reset, in SPI mode 1X SDR Data.Set A:&QSPI_Cntl_BASE+0x154 %LE %Long 0x000000e7 //TX Buffer Data Regsiter=0xe7 means			

<table>
<tr><td></td><td>ng 0x00000002 //TX Buffer Data Regsiter=2</td><td>Octal DTR with DQS</td></tr>
<tr><td>Timing diagram</td><td>Figure 31. Write Configuration Register 2 (WRCR2) Sequence (SPI Mode)
 
</td><td>Figure 26. Write Nonvolatile/Volatile Configuration Register Sequence Diagram (SPI)
 
</td></tr>
<tr><td>Comments</td><td>The WRCR2 instruction is for changing the values of Configuration Register 2. Before sending WRCR2 instruction,
the Write Enable (WREN) instruction must be decoded and executed to set the Write Enable Latch (WEL) bit in
advance.
9-3. Configuration Register 2
<table>
<tr><th>Address</th><th>Bit</th><th>Name</th><th>Description</th><th>Default</th></tr>
<tr><td rowspan="2">000h</td><td>Bit 0</td><td>SOPI (STR OPI Enable)</td><td>0= STR OPI disable
1= STR OPI enable</td><td>0</td></tr>
<tr><td>Bit 1</td><td>DOPI(DTR OPI Enable)</td><td>0= DTR OPI disable
1= DTR OPI enable</td><td>0</td></tr>
</table></td><td>The Write Nonvolatile/Volatile Configuration Register (WRCR) command allows new values to be written to the Nonvolatile/Volatile Configuration Register. Before it can be accepted, a Write Enable (WREN) command must previously have been executed
Table 9 Volatile Configuration Register
<table>
<tr><th>Addr</th><th>Settings</th><th>bit7</th><th>bit6</th><th>bit5</th><th>bit4</th><th>bit3</th><th>bit2</th><th>bit1</th><th>bit0</th><th>Description</th></tr>
<tr><td rowspan="7"><0></td><td rowspan="7">I/O mode</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>SPI with DQS (Default)</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>SPI W/O DQS</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>Octal DTR with DQS</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>Octal DTR W/O DQS</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>Octal STR with DQS</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>Octal STR W/O DQS</td></tr>
<tr><td colspan="8">Others</td><td>Reserved</td></tr>
</table></td></tr>
</table>

Note that for GD25LX256E:

Internal configuration register settings that cannot be directly accessed by the user during QSPI NOR configuration. The user can use WRITE NOVOLATILE configuration register to change the default configuration after power on. The information of the nonvolatile configuration register overwrites the internal configuration register during power on or after reset.

The user can use WRITE VOLATILE configuration REGISTER to change the configuration during device operation. After the command is executed, the information from the volatile configuration register immediately overwrites the internal configuration register after the WRITE command is completed.



Therefore, when writing the configuration register, use the command 0x81 to write the volatile register, which takes effect directly.

Write Volatile Configuration Register	81h	1-1-1	0	8-8-8	8-8-8	0	3(4)	1

2.4 Use DOPI mode READ_8DTRD

QuadSPI_Read32BytesDOPI, Call:

```
|->; write sequence ID and assert Read command
  Data.Set A:&QSPI_Cntl_BASE+0x08 %Long  ((7.<<24.)+32.) ; sequence 7 + 32 bytes to be
```

Refer MX25UW51245G design GD25LX256E Command sequence:

Note that the MX25U51245G recommendation for the dummy setting is:

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*

So at 200Mhz, it is set to 0x14=20 clocks.

GD25LX256E recommendation:

Table 10 Clock Frequencies of TFBGA-24 (5x5 Ball Array)

Number of Dummy Clock Cycle	Octal I/O FAST READ		OPI DTR
	STR	DTR	
4	40	40	40
6	84	84	84
8	104	104	104
10	133	133	133
12	152	152	152
14	166	166	166
16 and above	166	200	200

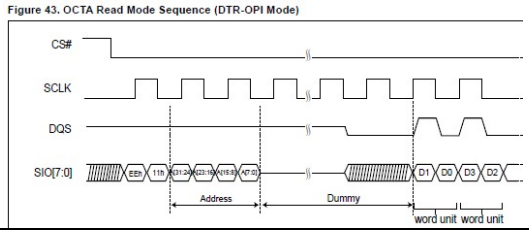
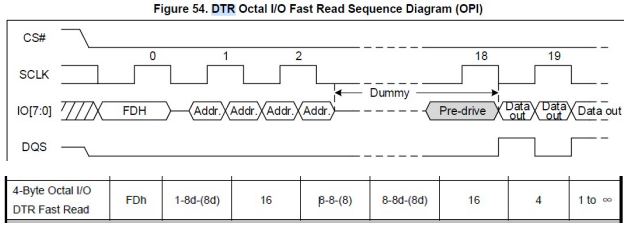
It can be set to 0x10=16 clocks in 200Mhz OPI-DTR mode. Please refer to the table for specific values:

Table 13 Commands (Extended/Octal SPI)

Command name	Code	Extended SPI		Octal SPI			Addr. Bytes	Data Bytes
		CMD-Addr-Data	Dummy Clock Cycles	CMD- Addr-Data (S-D-D)	CMD- Addr-Data (S-S-S)	Dummy Clock Cycles		

	MX25U51245G(Reference)				GD25LX256E(Design)			
Lauterbach code	;Program LUT35 with 8DTRD - READ DOPI mode Data.Set A:&QSPI_Cntl_BASE+0x39C %LE %Long 0x471147EE ; SEQID 7 Data.Set A:&QSPI_Cntl_BASE+0x3A0 %LE %Long 0x0C142B20 Data.Set A:&QSPI_Cntl_BASE+0x3A4 %LE %Long 0x00003B01 Data.Set A:&QSPI_Cntl_BASE+0x3A8 %LE %Long 0x00000000				;Program LUT35 with 8DTRD - READ DOPI mode Data.Set A:&QSPI_Cntl_BASE+0x39C %LE %Long 0x470247FD ; SEQID 7 Data.Set A:&QSPI_Cntl_BASE+0x3A0 %LE %Long 0x0F142B20 Data.Set A:&QSPI_Cntl_BASE+0x3A4 %LE %Long 0x00003B01 Data.Set A:&QSPI_Cntl_BASE+0x3A8 %LE %Long 0x00000000			
Details		Instr(6bits)	Pads(2bits)	Operand(8bits)		Instr(6bits)	Pads(2bits)	Operand(8bits)
	47ee	0x11(CMD_DDR)	0x3(8 bit)	0xee 0x11(Octa I/O DTR read)	47fd	0x11(CMD_DDR)	0x3(8 bit)	0xfd (OCTAL I/O FAST READ with DDR ADDRESS and DATA)
	4711	0x11(CMD_DDR)	0x3(8 bit)		4702	0x11(CMD_DDR)	0x3(8 bit)	0x02(0xfd 的补码)
	2b20	0xA(ADDR_DDR)	0x3(8 bit)	0x20(32 Addr bits to be sent on 4 pad)	2b20	0xA(ADDR_DDR)	0x3(8 bit)	0x20(32 Addr bits to be sent on 4 pad)
	0c14	0x3(DUMMY)	0x0(1 bit)	0x14(20 dummy cycles)	0f10	0x3(DUMMY)	0x3(8 bit) Dummy 命令	0x10(16 dummy cycles)

S32G ADD GD FLASH SUPPORT

							可用 1bit or 3 bit	
	3b10	0x14(READ_DDR)	0x3(8 bit)	0x10(Read 16 Bytes on 4 pad)	3b10	0x14(READ_DDR)	0x3(8 bit)	0x10(Read 16 Bytes on 8 pad)
Timing Diagram								
Comments	The 8DTRD instruction enable DTR Octa throughput of Serial Flash in read mode. An DOPI Enable bit of Configuration Register 2 must be set to "1" before sending the DTR Octa READ instruction.				The Octal I/O DTR Read command enables Double Transfer Rate throughput on Octal I/O of Serial Flash in read mode. The address (interleave on 8 I/O pins) is latched on both rising and falling edge of SCLK, and data (interleave on 8 I/O pins) shift out on both rising and falling edge of SCLK. The 8-bit address can be latched-in at one clock edge, and 8-bit data can be read out at one clock edge, which means 8 bits at rising edge of clock, the other 8 bits at falling edge of clock. The first address Byte can be at any location. The address is automatically increased to the next higher address after each Byte data is shifted out, so the whole memory can be read out at a single Octal I/O DTR Read command. The address counter rolls over to 0 when the highest address has been reached.				
Print	PRINT "1st 0x" Data.Long(A:&QSPI_Cntl_BASE+0x200) PRINT "2nd 0x" Data.Long(A:&QSPI_Cntl_BASE+0x204) PRINT "3rd 0x" Data.Long(A:&QSPI_Cntl_BASE+0x208) PRINT "4th 0x" Data.Long(A:&QSPI_Cntl_BASE+0x20C) PRINT "5th 0x" Data.Long(A:&QSPI_Cntl_BASE+0x210)							

2.5 Test report

Set the RDB3 board to download mode, or burn the image with the wrong IVT head in QSPI NOR, and start:

1. Run Lauterbach: t32marm.exe: File ->Open script...=device_gd_readid.cmm.
2. Then click the area command in the command bar to see the printed ID value in the RX buffer:

```
file C:\Work\S32G\Application notes\qspi_nor\GD\development\1.lauterbach\device_gd _read
1st 0x19 (Density)
2nd 0x68 (Device ID)
3rd 0x0C8 (Manufacture)
4th 0x0FF
```

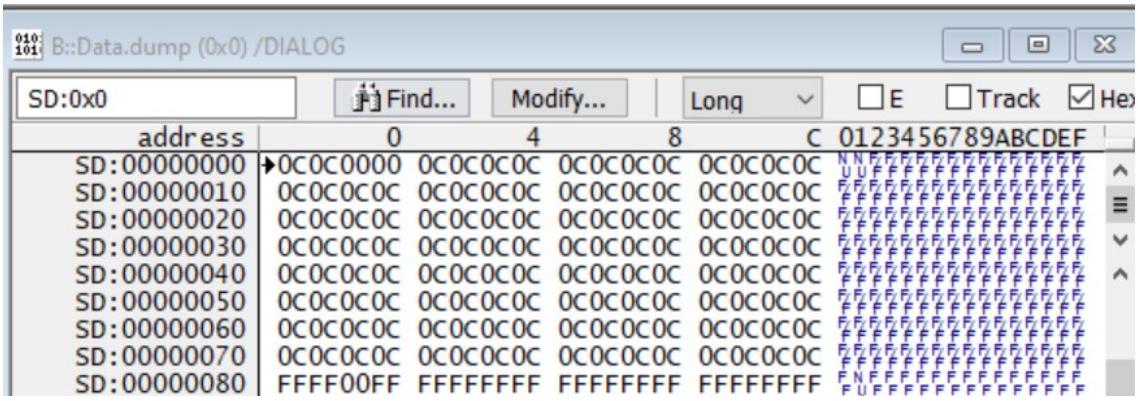
1. Run again Lauterbach: t32marm.exe：File->Open script…= device_gd.cmm。

```
file C:\Work\S32G\Application notes\qspi_nor\GD\development\1.lauterbach\device_gd.cmm
1st 0x0
2nd 0x68 (Device ID)
3rd 0x0C8 (Density)
4th 0x0FF (Manufacture)
1st 0x0C0C0000
2nd 0x0C0C0C0C
3rd 0x0C0C0C0C
4th 0x0C0C0C0C
5th 0x0C0C0C0C
```

2. Then in the menu View ->dump..., enter the address 0x0:

Start address (hex)	End address (hex)	Size (KB)	40-bit Master Description 32-bit Master Description (except M7) M7 Description HSE M7 Description	A53 CC FlexNOC Slave port	M7 Default Cache mode	M7 Bus	M7 Memory Space	M7 Memory Type
Code								
0x00_0000_0000	0x00_1FFF_FFFF	524288	QSPI AHB Buffer	s_flash	WT	AXIM	Code	Normal

It can be seen that the AHB address has read QSPI NOR content:



3 Flash tool algorithm image development

Refer to <<AN13563: S32G QuadSPI Deep Dive Application Note>>, 5.3 Flash SDK usage，Understand the image development method of Flash tool:

Image development is based on the following principles:

1. Generally speaking, Flash of different models from the same manufacturer will use the same command word and the same register design.

2. At present, the existing algorithms of the S32DS flash tool are mirrored in the directory C: NXP S32DS. 3.4 S32DS tools S32FlashTool flash, including:

Num.	Company	Model
1	Micron	MT35XU02GCBA.bin
2	MACRONIX	MX25UM51245G.bin
3		MX25UW12A45G_R52.bin
4		MX25UW51245G.bin
5	CYPRESS	S26KL512S2.bin
6		S26KS512S.bin
7		S70FS01GS.bin

So if you use the flash of the above three companies, you can first try to use the image burning algorithm similar to it to see whether it will succeed. First select the same model, then select the same model but different sizes, and finally select the models and sizes that may be different.

3. QSPI NOR flash manufacturers will consider a certain degree of compatibility, so you can first compare whether the command word and register definitions are the same.

4. Finally, we will consider according to 5.3 Flash SDK usage develops a new algorithm image.

3.1 Algorithms implemented by Flash SDK

According to 5.3 The Flash SDK usage shows that the algorithms implemented by the flash SDK include:

- Read_id

```
/* 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);
```

- Erase_chipset

```
/* SEQID 5 - Chip Eraser*/
  qspi_compose_lut_register(p_pb, SEQID_CHIP_ERASE, CHIP_ERASE_CMD, 0, 0, 0, 0, 0, 0);
```

- Write_flash

```
/* 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);
```

- Dump_flash

```
/* 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);
```

```
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->write_any_register_cmd = WRITE_ANY_REGISTER_CMD;
        pb->write_any_register_address_length = WRITE_ANY_REGISTER_ADDRESS_LENGTH;
        pb->write_any_register_dummy = WRITE_ANY_REGISTER_DUMMY;
        pb->write_any_register_length = WRITE_ANY_REGISTER_LENGTH;

#define CHIP_ERASE_CMD                                                    0x60
/*
 * 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

#define CHIP_ERASE_CMD                                         0x60

#define WRITE_ANY_REGISTER_CMD                                 0x72
#define WRITE_ANY_REGISTER_LENGTH                        1
#define WRITE_ANY_REGISTER_ADDRESS_LENGTH                32
#define WRITE_ANY_REGISTER_DUMMY                         0
```

3.2 Develop new flash source code

Refer doc C:\NXP\S32DS.3.4\S32DS\help\resources\howto\<<HOWTO_Use_FlashSDK_to_add_support_for_QuadSPI_flash_memory_devices_for_S32_Flash_Tool.pdf>>, to build S32DS FlashSDK project, Note:

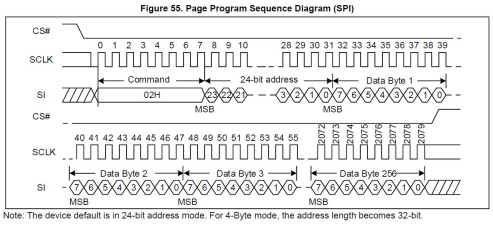
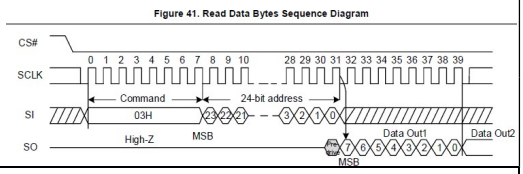
- On FlashSDK:Release_FlashTemple right click->Build Configurations->Set Active:can switch to Release or Debug Temple.
- On FlashSDK:Release_FlashTemple right click-->Build Project, It can be compiled. The result is in the Console window. The compiled image is in: C:\NXP\S32DS.3.4\S32DS\tools\S32FlashTool\FlashSDK_Ext\Release_FlashTemplate\FlashSDK.bin.

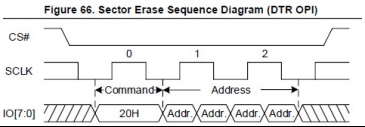
Develop a new Flash algorithm to drive the image. The main modifications are the file:C:\NXP\S32DS.3.4\S32DS\tools\S32FlashTool\FlashSDK_Ext\Algo\Generic\qspi_chip_commands.h.

Compare the command words of the two flash models with other definitions as follows (note that the current Flash SDK uses low-speed SPI mode, not high-speed mode):

	MX25UM51245G	GD25LX256E	Comments
NOR	0 ///< NOR memory type		pb->is_nand = MEMTYPE;//=NOR=0, do not support QSPI Nand
NAND	1// NAND Memory type. not supported		
MEMTYPE	NOR		
BLOCK_PROTECT_MASK	(x3C)		//status register protect bit 5~2
TOP_BOTTOM_MASK	(x08)		//configure register bit3 top area or bottom area protect but GD have no this register, so keep //it to avoid compiling error. current flash tool have no protect ability
WEL	(1 << 1)		///<Write Enable Latch bit of configuration register //should be status register
WIP	(1 << 0)		///<Write in Progress bit of configuration register //should be status register
BYTES_PER_PAGE	(256)		///< page size in bytes //- 256 Bytes per programmable page
NUMBER_OF_SECTORS	(1024)	(1024) (8192)	number of flash sectors Because the code uses sector program, the sector needs to use 1024 and 4 * 1024 instead of 8192 and 4 * 1024
BYTES_PER_SECTOR	(64 * 1024)	(64 * 1024) (4 * 1024)	size of sector in bytes
REG_PROTECTION_ADDR	0x00		address of the register that holds the protection bits, if exists
REG_STATUS_ADDR	0x00		address of the register that holds the status bits, if exists
READ_ID_CMD	0x9F	0x9F	Refer 2.1
READ_ID_DUMMY	0	0	
READ_ID_LENGTH	4	4	
WRITE_ENABLE_CMD	0x06	0x06	Refer 2.1
PAGE_PROGRAM_CMD	0x12	0x12	9.19 Page Program (PP) (02H/12H)
PAGE_PROGRAM_DUMM	0	0	sending Page Program command → 3-Byte address

S32G ADD GD FLASH SUPPORT

Y			or 4-Byte address on SI
PAGE_PROGRAM_ADDRESS_LENGTH	32	32	 0x12 is Four byte command word
PAGE_READ_CMD	0x13	0x13	9.14 Read Data Bytes (READ) (03H/13H)
PAGE_READ_DUMMY1	0	0	0x13 is Four byte command word
PAGE_READ_ADDRESS_LENGTH	32	32	
PAGE_READ_DUMMY2	0	0	
READ_STATUS_REGISTER_CMD	0x05	0x05	Refer 5.3.3.9
READ_STATUS_REGISTER_DUMMY	0	0	
READ_STATUS_REGISTER_ADDRESS_LENGTH	0	0	
READ_STATUS_REGISTER_LENGTH	1	1	
WRITE_STATUS_REGISTER_CMD	0x01	0x01	Refer 2.2
WRITE_STATUS_REGISTER_DUMMY	0	0	
WRITE_STATUS_REGISTER_ADDRESS_LENGTH	0	0	
WRITE_STATUS_REGISTER_LENGTH	1	1	
SECTOR_ERASE_CMD	0xDC	0xDC	Refer 5.3.3.2
SECTOR_ERASE_DUMMY	0	0	
SECTOR_ERASE_ADDRESS_LENGTH	32	32	
SUBSECTOR_ERASE_CMD	0x21	0x21	9.22 Sector Erase (SE) (20H/21H)
SUBSECTOR_ERASE_DUMMY	0	0	

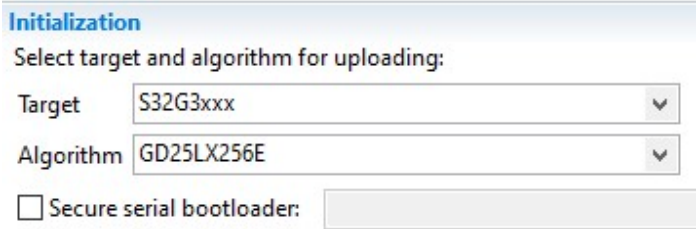
SUBSECTOR_ERASE_ADDRESS_LENGTH	32	32	
CHIP_ERASE_CMD	0x60	0x60	
CLEAR_STATUS_REGISTER_CMD	0	0	which means have no this command
READ_ANY_REGISTER_CMD	0x71	0x85	Since the SPI line low speed mode is adopted, the configure register does not need to be operated. Refer 5.3.3.8
READ_ANY_REGISTER_LENGTH	1	1	
READ_ANY_REGISTER_ADDRESS_LENGTH	32	24	
READ_ANY_REGISTER_DUMMY	0	0	
WRITE_ANY_REGISTER_CMD	0x72	0x81	Since the SPI line low speed mode is adopted, the configure register does not need to be operated. Refer 2.2
WRITE_ANY_REGISTER_LENGTH	1	1	
WRITE_ANY_REGISTER_ADDRESS_LENGTH	32	24	
WRITE_ANY_REGISTER_DUMMY	0	0	
READ_ID_CMD1_OPI	0x9F	0x9F	Refer 5.3.3.7
READ_ID_CMD2_OPI	0xF9	0xF9	
READ_ID_DUMMY_OPI	0	0	
READ_ID_LENGTH_OPI	4	4	

Therefore, the source file does not need to be modified. You can work directly by compiling the image directly.

Copy C:\NXP\S32DS.3.4\S32DS\tools\S32FlashTool\FlashSDK_Ext\Release_FlashTemplate\FlashSDK.bin to C:\NXP\S32DS.3.4\S32DS\tools\S32FlashTool\flash. Rename to GD25LX256E.bin。Then modify: C:\NXP\S32DS.3.4\S32DS\tools\S32FlashTool\configs\flash_devices.xml,add GD25LX256E.bin in:

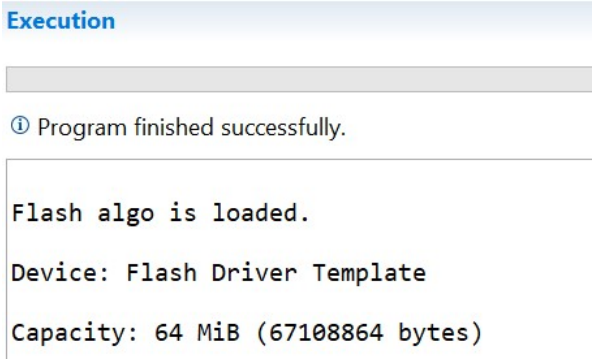
```
<algorithm>
    <id>GD25LX256E</id>
    <name>GD25LX256E</name>
    <path>flash/GD25LX256E.bin</path>
</algorithm>
```

In this way, you can see in the algorithm image drop-down box of the flash tool:

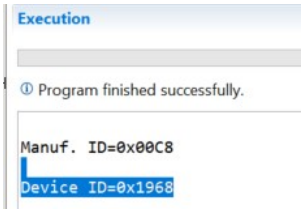


3.3 Test Report

- Use the Flash tool to load the algorithm：Upload target and algorithm to hardware:

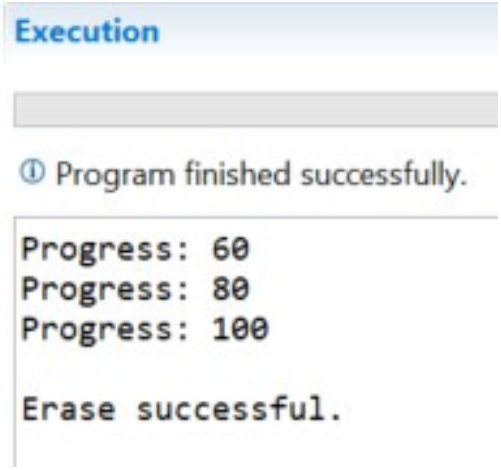


- Use Get flash ID after success: check whether it is successful:

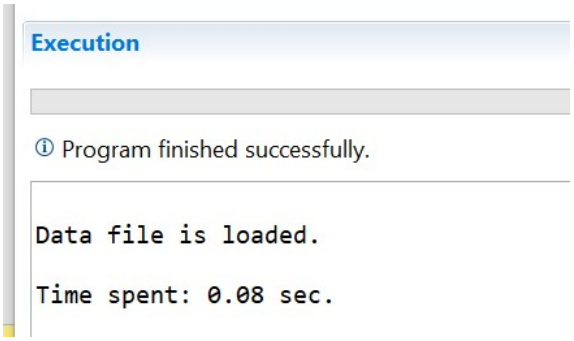


Same with datasheet.

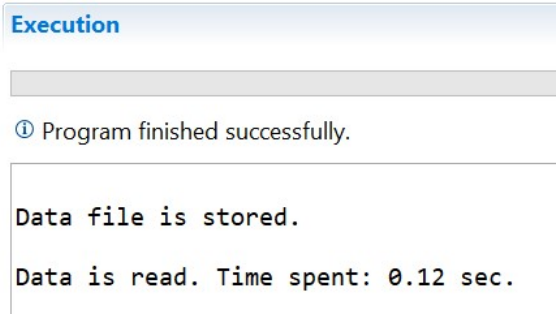
- Then use the erasing function of Flash tool: Erase memory range to see whether the erasing is successful:

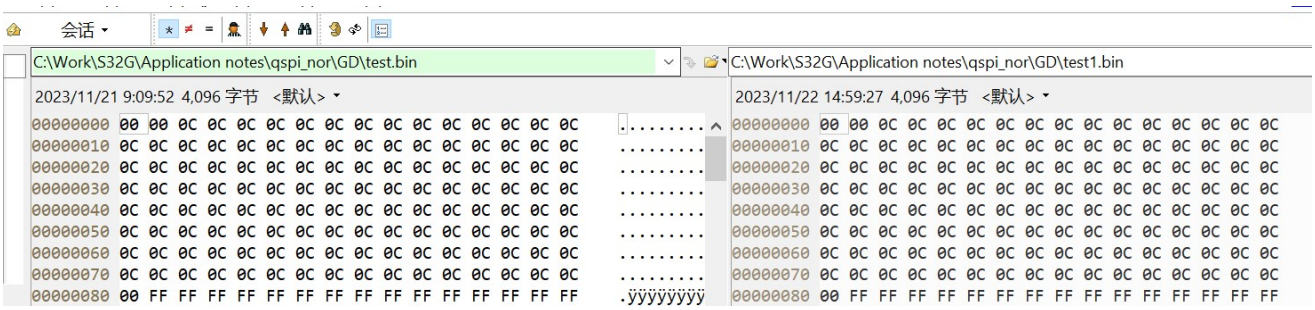


- Test the flash tool's burning function:Upload file to device:



- Finally, the Flash tool was used to read the image burned in for comparison:





4 Develop IVT Parameter Header

Refer doc <<AN13563: S32G QuadSPI Deep Dive Application note>>, Chapter 4 QuadSPI Boot and doc<<S32G_QSPINOR_Customization_*.pdf>>, Chapter 4 S32G QSPI NOR flash parameter header customization. The following is a comparison of the configurations of three flash models in 200Mhz, DDR, External DQS, Auto Update/Bypass, and a comparison of the appropriate configurations of GD25LX256E:

	Macronix MX25UW51245G	Micron MT35XU256ABA		GD GD25LX256E
	200Mhz	200Mhz		200Mhz
	DDR	DDR		DDR
	External DQS	External DQS	External DQS	External DQS
	Auto Update	Auto Update	Bypass	Auto Update
Flash Port Connection	A	A	A	A
DLL Bypass mode	No	No	Yes	No
DLL Auto Update Mode	Yes	Yes	No	Yes
IPCR Enable Mode	No	No	No	No
SFLASH Clock Frequency	0xc8	0xc8	0xc8	0xc8
MCR	0x30f00cc	0x30f00cc	0x30f00cc	0x30f00cc
	DQS_FA_ SEL=3	DQS_FA_ SEL=3	DQS_FA_ SEL=3	DQS_FA_ SEL=3
FLSHCR	0x10303	0x10303	0x10303	0x10303
BFGENCR	0x0	0x0	0x0	0x0
DLLCRA	0xc280000c	0xc280000c	0x40000506	0xc280000c
	DLLEN=1	DLLEN=1	DLLEN=0	DLLEN=1
	FREQEN=1	FREQEN=1	FREQEN=1	FREQEN=1
	DLL_REFCNTR=2	DLL_REFCNTR=2	DLL_REFCNTR=0	DLL_REFCNTR=2

	DLLRES=8	DLLRES=8	DLLRES=0	DLLRES=8
	SLV_FINE OFFSET=0	SLV_FINE OFFSET=0	SLV_FINE OFFSET=0	SLV_FINE OFFSET=0
	SLV_DLY OFFSET-=0	SLV_DLY OFFSET-=0	SLV_DLY OFFSET-=0	SLV_DLY OFFSET-=0
	SLV_DLY COARSE=0	SLV_DLY COARSE=0	SLV_DLY COARSE=5	SLV_DLY COARSE=0
	SLAVE_AUTO _UPDT =1	SLAVE_AUTO _UPDT =1	SLAVE_AUTO _UPDT =0	SLAVE_AUTO _UPDT =1
	SLV_EN=1	SLV_EN=1	SLV_EN=1	SLV_EN=1
	SLV_DLL_ BYPASS=0	SLV_DLL_ BYPASS=0	SLV_DLL_ BYPASS=1	SLV_DLL_ BYPASS=0
	SLV_UPD=0	SLV_UPD=0	SLV_UPD=0	SLV_UPD=0
PARITYCR	0x0	0x0	0x0	0x0
SFACR	0x20000	0x0	0x0	0x0
	Byte Swapping=1	Byte Swapping=0	Byte Swapping=0	Byte Swapping=0
SMPR	0x44000000	0x44000000	0x44000000	0x44000000
DLCR	0x40ff40ff	0x40ff40ff	0x40ff40ff	0x40ff40ff
SFA1AD	0x20000000	0x20000000	0x20000000	0x20000000
SFA2AD	0x20000000	0x20000000	0x20000000	0x20000000
DLPR	0xaa553443	0xaa553443	0xaa553443	0xaa553443
SFAR	0x0	0x0	0x0	0x0
TBDR	0x0	0x0	0x0	0x0
lut[0] command sequence	0xee 0x47 0x11 0x47	0xfd 0x47 0x2 0x47	0xfd 0x47 0x2 0x47	0xfd 0x47 0x2 0x47
lut[1] command sequence	0x20 0x2b 0x14 0xf	0x20 0x2b 0x10 0xf	0x20 0x2b 0x10 0xf	0x20 0x2b 0x10 0xf
lut[2] command sequence	0x10 0x3b 0x0 0x0	0x10 0x3b 0x0 0x0	0x10 0x3b 0x0 0x0	0x10 0x3b 0x0 0x0
lut[3] command sequence	0	0	0	0
lut[…] command sequence	0	0	0	0
lut[79] command sequence	0	0	0	0
Flash Write Data[0]	No/0byte/0x0/spi	No/0byte/0x0/spi	No/0byte/0x0/spi	No/0byte/0x0/spi

	/0x6/0x0/0x0	/0x6/0x0/0x0	/0x6/0x0/0x0	/0x6/0x0/0x0
Flash Write Data[1]	Yes/1byte/0x20/spi /0x72/0x0/0x2	Yes/1byte/0x18/spi /0x81/0x0/0xe7	Yes/1byte/0x18/spi /0x81/0x0/0xe7	Yes/1byte/0x18/spi /0x81/0x0/0xe7
Flash Write Data[…]	0	0	0	0
Flash Write Data[9]	0	0	0	0

It can be seen that the main differences are:

4.1 S32G QSPI Controllder configuration difference

1. IPCR Trigger

Because:

31.13.1.1 Functional description

Boot ROM supports boot from a variety of flash memories, providing flexibility for choosing the configuration parameters for which the controller must be programmed during boot. The configuration parameters can be based on product requirements. For better performance, the QuadSPI controller supports high-speed operations in DDR and SDR modes, which requires specific configurations to achieve correct data sampling at such a high rate. The QuadSPI controller supports reading data over an AHB interface or an IP interface—however, boot ROM supports only read via an AHB interface. Boot ROM does not support any write operations to QuadSPI flash memory.

Therefore, it is not necessary to configure to IP interface mode, IPCR Trigger does not need to be checked, and the registers to be configured for AHB mode are:

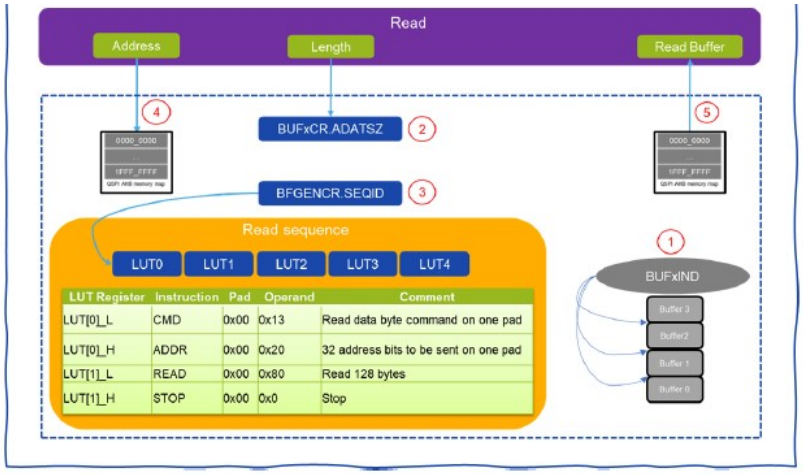


Figure 17. **Read – AHB command**

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

Therefore, it is necessary to ensure that:

- BUF0IND= BUF1IND= BUF2IND=0 //so BUFFER3=1024 bytes
- BUF3CR[ALLMST]=1 // All masters can access BUFFER3
- BUF3CR[ADATSZ]// It doesn't matter if the value is 0, it will be reset by the configuration of the SEQID
- BFGEBCR[SEQID]=0 // The LUT is configured as 0, so the IVT header needs to configure the fast read as LUT0.

Use an empty flash. In the QSPI NOR flash startup mode, after the startup fails, connect the lauterbach and confirm as

follows:

IPCR	00000000	SEQID	0	PAR_EN	0
		IDATSZ	0000		
FLSHCR	00000303	TDH	0: Data aligned with the posedge of internal..	TCSH	3
		TCSS	3		
BUF0CR	0000000B	ADATSZ	00	MSTRID	11
BUF1CR	00000001	ADATSZ	00	MSTRID	1
BUF2CR	00000002	ADATSZ	00	MSTRID	2
BUF3CR	80000003	ALLMST	1	ADATSZ	00
		MSTRID	3		
BFGENCR	00000000	PAR_EN	0	SEQID	0
BUF0IND	00000000	TPINDX0	00		
BUF1IND	00000000	TPINDX1	00		
BUF2IND	00000000	TPINDX2	00		

Therefore, the default code configuration of ROM Code meets the requirements. In addition:

LUTKEY	5AF05AF0	KEY	5AF05AF0		
LCKCR	00000002	UNLOCK	1	LOCK	0
LUT0	08180403	INSTR1	2	PAD1	0: 1 Pad
		OPRND1	18	INSTR0	1
		PAD0	0: 1 Pad	OPRND0	03
LUT1	24001C08	INSTR1	9	PAD1	0: 1 Pad
		OPRND1	00	INSTR0	7
		PAD0	0: 1 Pad	OPRND0	08
LUT2	00000000	INSTR1	0	PAD1	0: 1 Pad

So the LUT0 used by ROM Code by default is:

Table 228. Reset sequence

Instruction	Pad	Operand	Comment
CMD	0h	3h	Read data byte command on one pad
ADDR	0h	18h	24 address bits to be sent on one pad

Table continues on the next page...

Table 228. Reset sequence (continued)

Instruction	Pad	Operand	Comment
READ	0h	8h	Read 64 bits
JMP_ON_CS	0h	0h	Jump to instruction 0 (CMD)

2. DQS mode select

Refer doc <<AN13563: S32G QuadSPI Deep Dive Application note>>, Chapter 3.3.2. Supported DQS sampling method, and doc <<AN12808: QSPI Timing Configuration>>, Chapter 3 Sampling the read data from QSPI Flash memory, understand the DQS mode select.

25-24 DQS_FA_SEL	DQS clock for sampling read data at flash memory A Selects DQS clock for sampling read data at flash memory A QuadSPI port 00b - Reserved 01b - Pad loopback 10b - Reserved 11b - External DQS NOTE In case of an padloopback selection to access port A, port B cannot be programmed for external DQS and vice-versa.

Note:

- In order to improve the access speed, it is recommended to use the External DQS mode in the DDR mode, while the DDR mode generally needs to work at more than 133Mhz, and usually works at 200Mhz. The maximum use of Pad loopback in the DDR mode can only be 66Mhz.
- For 133Mhz SDR mode, Pad loopback mode can be used instead of External DQS mode.
- External DQS mode requires QSPI NOR to output clock to S32G, so QSPI NOR itself and PCB connection will affect the quality of DQS signal, so in addition to hardware measurement, 133Mhz SDR Pad loopback mode can also be used as a reference test.

2. Difference between Auto Update mode and Bypass mode:

Refer to <<AN13563: S32G QuadSPI Deep Dive Application note>>, chapter 3.3.3 DLL and DQS delay chain and <<AN12808: QSPI Timing Configuration>>, Chapter 4: DQS delay circuits, learn about the selection methods of DQS delay circuits,

The setting method of DLL Bypass mode is (dynamic code):

A. Set DLLCRA [SLV_EN]=1, DLLCRA [SLV_DLL_BYPASS]=1 DLLCRA [SLAVE_AUTO_UPT]=0.

B. Program the following fields to provide the DQS delay DLLCRA [SLV_FINE_OFFSET], DLLCRA [SLV_DLY_COARSE] and DLLCR [FREQEN] required for sampling. For supported programming settings, see chip specific QuadSPI information.

C. Set DLLCRA [SLV_UPD]=1 to load these values into the slave delay chain.

D. Check the update status from the delay chain by polling DLLSR [SLVA_LOCK]=1, and clear DLLCRA [SLV_UPD] after confirming the update status

So the final register setting is:

SLV_EN=1, SLV_DLL_BYPASS=1, SLAVE_AUTO_UPT=0, SLV_FINE_OFFSET=0 or a value (fine call can be set to 0 first), SLV_DLY_COARSE=5, FREQEN=1 (200Mhz is set to 1133Mhz is set to 0), SLV_UPD=changes from 1 to 0, and finally to 0.

The configuration method of DLL Auto Update mode is (dynamic code):

A. Program DLLCRA [SLV_EN]=1, DLLCRA/SLV_DLL_BYPASS]=0, DLLCRA [SLAVE_AUTO_UPT]=1.

B. Use DLLCRA [DLL_REFCNTR] and DLLCRA [DLLRES] to program the DLL configuration. For supported DLL configuration settings, see chip specific QuadSPI information.

C. The slave settings are programmed to delay DQS by using the fields DLLCRA [SLV_FINE_OFFSET], DLLCRA [SLV_DLY_OFFSET] and DLLCR [FFREQEN]. See chip specific QuadSPI information for supported settings.

D. If the offset delay needs to be updated on the slave chain, the program DLLCRA [SLV_UPD]=1.

E. Enable DLL by programming DLLCRA [DLLEN]=1, and reset DLLCRA [SLV_UPD]=0. The slave delay chain will be updated automatically, and can be checked by polling DLLSR [SLVA_LOCK]==1

So the final register setting is:

SLV_EN=1, SLV_DLL_BYPASS=0, SLAVE_AUTO_UPT=1, DLL_REFCNTR=2, DLLRES=8, SLV_FINE_OFFSET=0, SLV_DLY_OFFSET=0, FFREQEN=1, SLV_UPD=0 from 1, finally 0, DLLEN=1.

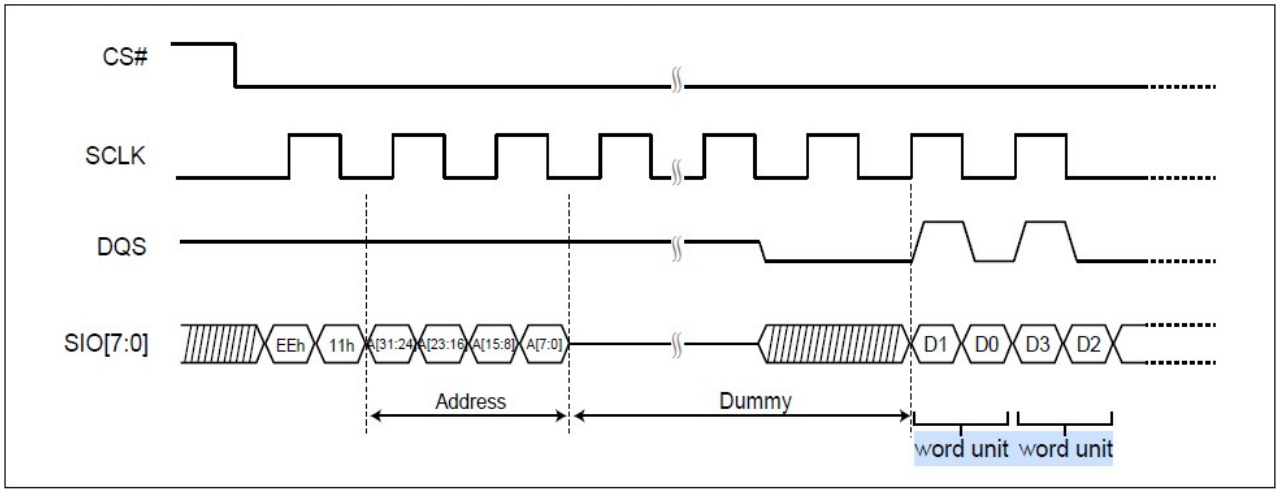
3. BYTE Swapping difference:

Byte swapping define as follows:

17 BYTE_SWAP	Byte swapping In case of Octal DDR mode, this field controls whether a word unit composed of 2 bytes from posedge and negedge of a single DQS cycle needs to be swapped. 0b - One word of two bytes at [nth, n+1th] address 1b - One word of two bytes at [n+1th, nth] address

Refer MX25UW51245G flash datasheet:

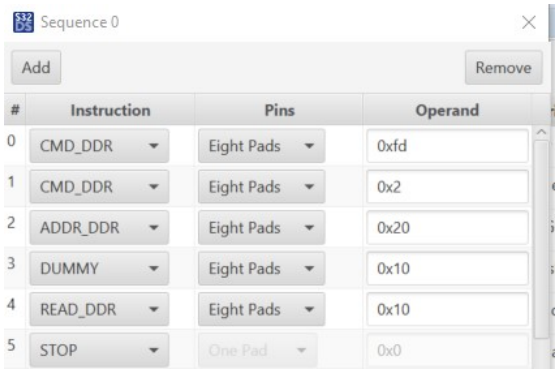
Figure 43. OCTA Read Mode Sequence (DTR-OPI Mode)



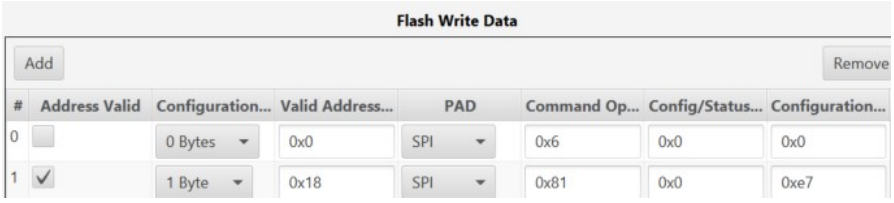
So for a word unit, the high byte comes first and the low byte comes last, so it needs to be set to swap here. Micron and GD do not have this requirement.

4.2 QSPI Configuration Difference

- Refer to Section 2.3 to configure GD25LX256E Command Sequences using DOPI mode READ_8DTRD



- Refer to section 2.2, configure QSPI NOR to DOPI mode, configure Flash Write Data, set GD25LX256E Flash to write enable, and then set QSPI NOR to DOPI mode.



Store as: GD_QSPI_Parametes_200M_DDR__ExternalDQS_Autoupdate.bin

4.3 Test Report

Refer to the description in the document <<S32G_RTD_MCAL_V *. Pdf>>, compile a DIO lighting example, pack it, and note:

- Configure QuadSPI parameters Select the IVT QSPI header image exported from S32DS: GD_QSPI_Parameteres_200M_DDR__ExternalDQS_Autoupdate.bin.
- DCD section is used to initialize SRAM, or it is not necessary to select: C:\NXP\Integration_Reference_Examples_S32G3_2023_02\code\framework\realtime\swc\bootloader\platforms\S32G3XX\res\flash\S32G3XX_DCD_InitSRAM.bin.
- The example of DIO lighting requires that GPIO switch SW11 be set to on.

Then set it to QSPI NOR startup mode, power on and start, you can see the U128 RGB light flashing. Prove successful startup.

Note:

With reference to the document <<S32G_QSPINOR_Customize_ *. Pdf>>, the larger image in section 10.2 cannot be started from QSPI-NOR without a parameter header, so:

- If the IVT QSPI NOR parameter header development is not completed, you can use the default 1 bit low-speed mode of ROM to start a small image, such as the Bootloader image, to avoid block bringing up.
- In order to accelerate the starting speed, it is recommended to complete the development of IVT QSPI NOR parameter header and add parameter header in IVT.
- In order to accelerate the startup speed, it is recommended to use 200Mhz, DDR, External DQS, Auto update mode to better adapt to temperature and other environmental factors, so if possible, try to use Auto update mode.
- If there are problems in the final high-speed mode development, you can use 133Mhz SDR Padloopback, bypass mode ->200 Mhz DDR external DQS bypass mode ->200 Mhz DDR external DQS Auto update mode to gradually upgrade the development in order to avoid block bringing up.

5 Develop MCAL Fls driver

Refer to the document <<AN13563: S32G QuadSPI Deep Dive Application note>>, chapter 6 Flash Driver configuration method – EB tresos， Understand the development of Flash MCAL Fls driver. Note that the development of the Fls driver can use Lauterbach debugging, so this work can be arranged flexibly after the development of the Lauterbach script driver (optional), or after the development of the Flash tool algorithm image/IVT parameter header.

In addition, NXP uses EB by default to configure Flash drivers, while some other Autosar vendors, such as Vector, use Davinci configuration work to configure Flash drivers. The interfaces of the two are different and the contents are the same. This article describes EB configuration.

As mentioned in the reference document, the Fls driver configuration includes three parts: the S32G Flash controller, the Flash Memory and the Fls sector. If replaces a new Flash. The main work focuses on the modification of the Flash Memory.

5.1 MCAL Fls Driver Project Details

5.1.1 MCAL Fls Driver Project

Take SW32G_RTD_4.4_4.0.2 as sample:

EB tresos Studio 27.1->File->Import->General->Existing Pojects into Workspace->Next->Select root directory->Browse to C:\NXP\SW32G2_RTD_4.4_4.0.2\eclipse\plugins\ Fls_TS_T40D11M40I2R0\examples\EBT\ S32G3\ Fls_Example_S32G399A_M7\TresosProject

Copy projects intow workspace->Finish.

Right click the project name Fls_Example_S32G399A_M7, and select Generate Project. The configuration source code file is generated. If you need to modify the configuration, save it and regenerate it.

Open the properties of the EB project and select Configuration Project ->Code Generator. The generated code will be placed in this relative path by default: ".. .. Generate". Therefore, after generating the code according to the previous step, the code will be placed in this relative path. At this time, you need to manually set all files in this relative directory (for example, the default workspace is under C:\EB\tresos\ ), copy C:\EB\tresos\generate to C:\NXP\SW32G2_RTD_4.4_4.0.2\eclipse\plugins\ Fls_TS_T40D11M40I2R0\examples\EBT\S32G3\ Fls_Example_S32G399A_M7\generate\.

Open Make file:

C:\NXP\SW32G2_RTD_4.4_4.0.2\eclipse\plugins\Fls_TS_T40D11M40I2R0\examples\EBT\S32G3\ Fls_Example_S32G399A_M7\project_parameters.mk

Modify the following parameters according to the path of your PC:

- TOOLCHAIN = gcc //Defaul MCAL using GCC.
- GCC_DIR= C:/NXP/S32DS.3.4/S32DS/build_tools/gcc_v9.2/gcc-9.2-arm32-eabi //S32DS GCC compiler path
- TRESOS_DIR= C:/EB/tresos // The installation path of EB Tresos Studio corresponding to this RTD.
- T32_DIR= C:/T32 //The installation path of Lauterbach's debugging software T32.
- PLUGINS_DIR // The path of RTD Plugins is relative by default, and generally does not need to be modified.
- MCAL_MODULE_LIST := BaseNXP Det Rte Fls MemIf Mcu Port Rm // Other Mcal modules that the Fls driver depends on.

In file Makefile defined:

```
ifneq (,$(findstring S32G3,$(EXAMPLE_DERIVATIVE)))
```

- FAMILY := S32G3XX

Launch Cygwin and enter:

C:\NXP\SW32G_RTD_4.4_4.0.2\eclipse\plugins\Fls_TS_T40D11M40I2R0\examples\EBT\S32G3\ Fls_Example_S32G399A_M7

Input command:

```
make build
```

wait for compiling finished.

5.1.2 Fls driver source code

```
Main()
|->Mcu_Init(NULL_PTR);
|->Mcu_InitClock(McuClockSettingConfig_0);
|-> Mcu_DistributePllClock();
|->Port_Init(NULL_PTR);
```

```
|-> Fls_Init(NULL_PTR);
|   |->Fls_IPW_Init();
|   |   |->Fls_IPW_InitControllers
|   |   |   |->Qspi_Ip_ControllerInit
|   |   |   |   |->Qspi_Ip_Disable
|   |   |   |   |->Qspi_Ip_ConfigureController
|   |   |   |   |   |->Qspi_Ip_ConfigureControllerA
|   |   |   |   |   |   |->Qspi_Ip_SetMemMapSizeA
|   |   |   |   |   |   |->Qspi_Ip_SetIdleLineValuesA
|   |   |   |   |   |   |->Qspi_Ip_SetCenterAlignedStrobeA
|   |   |   |   |   |   |->Qspi_Ip_SetDifferentialClockA
|   |   |   |   |   |->Qspi_Ip_SetSerialFlashAddress
|   |   |   |   |   |   |->Qspi_Ip_SetAddrOptions
|   |   |   |   |   |   |->Qspi_Ip_SetByteSwap
|   |   |   |   |   |->Qspi_Ip_SetRxCfg
|   |   |   |   |   |->Qspi_Ip_SetCsTime
|   |   |   |   |   |   |->Qspi_Ip_SetCsHoldTime
|   |   |   |   |   |   |->Qspi_Ip_SetCsSetupTime
|   |   |   |   |   |->Qspi_Ip_ConfigureReadOptions
|   |   |   |   |   |   |->QSPI_DQS_Enable
|   |   |   |   |   |   |->QSPI_DQS_LatEnable
|   |   |   |   |   |   |->QSPI_DDR_Enable
|   |   |   |   |   |   |->Qspi_Ip_SetDataInHoldTime
|   |   |   |   |   |   |->Qspi_Ip_SetDQSSourceA
|   |   |   |   |   |   |->Qspi_Ip_SetRxDLLTapA
|   |   |   |   |   |->Qspi_Ip_ConfigureChipOptions
|   |   |   |   |->Qspi_Ip_Enable
|   |   |   |   |->Qspi_Ip_SwReset
|   |   |   |   |->Qspi_Ip_ConfigureDLL
|   |   |   |   |   |->Qspi_Ip_ConfigureDLLA
|   |->Fls_IPW_InitMemories
|   |   |->Qspi_Ip_Init
//Reset QSPI
```

```
| | | |->Qspi_Ip_InitReset(instance, pConfig->initResetSettings.resetCmdLut,
pConfig->initResetSettings.resetCmdCount, state);
/
| | | | |->Qspi_Ip_InitLutSeq /* Copy sequence in LUT registers */
| | | | |->Qspi_Ip_IpCommand /* Run QSPI command */
| | | |->Qspi_Ip_InitDevice
| | | | |->Qspi_Ip_InitOperation
/ /Initialize (Reset) QSPI NOR to DTR-OPI mode according to the configuration on the InitConfiguration page.
case QSPI_IP_OP_TYPE_RMW_REG:
        /* Change a bitfield in the register */
        status = Qspi_Ip_InitRMWReg(instance, &initOperations[initOp]);
case QSPI_IP_OP_TYPE_QSPI_CFG:
        /* Re-initialize QSPI controller with the given configuration */
        (void)Qspi_Ip_ControllerDeinit(state->connection->qspiInstance);
        status = Qspi_Ip_ControllerInit(state->connection->qspiInstance, initOperations[initOp].ctrlCfgPtr);
| | |->Qspi_Ip_AhbReadEnable /* Configure the AHB reads for flash unit "cnt" */
| |->Fls_IPW_CheckDevicesId();
| | |->Fls_IPW_DeviceIdMatches
| | | |->Qspi_Ip_ReadId
Qspi_Ip_RunReadCommand(instance,
                        state->configuration->readIdSettings.readIdLut,
                        0U,
                        data,
                        NULL_PTR,
                        state->configuration->readIdSettings.readIdSize);
|->Fls_InitBuffers();
|->Fls_Erase(LOGICAL_START_ADDR, NUMBER_OF_EXTERNAL_SECTOR *
EXTERNAL_SECTOR_SIZE);
| |-> FLS_JOB_ERASE : Fls_DoJobErase
| | |-> Fls_IPW_SectorErase
| | | |->Qspi_Ip_EraseBlock
| | | | |->Qspi_Ip_BasicErase
| | | | | |->Qspi_Ip_SerialflashSectorErase
| | | | | | |->Qspi_Ip_WriteEnable
```

```
|   |   |   |   |   |   |->Qspi_Ip_RunCommand(instance, eraseLut, address);
|->Fls_Write(LOGICAL_START_ADDR, TxBuffer, FLS_BUF_SIZE);
…
|->Fls_Read(LOGICAL_START_ADDR, RxBuffer_IP, FLS_BUF_SIZE);
…
|->Fls_Compare(LOGICAL_START_ADDR, TxBuffer, FLS_BUF_SIZE);
…
|-> Fls_GetAhbData();
…
```

So pay attention to the reinitialization configuration of QSPI NOR in the InitConfiguration configuration page.

See Section 5.4 for testing.

5.2 FlsMem Configuration page

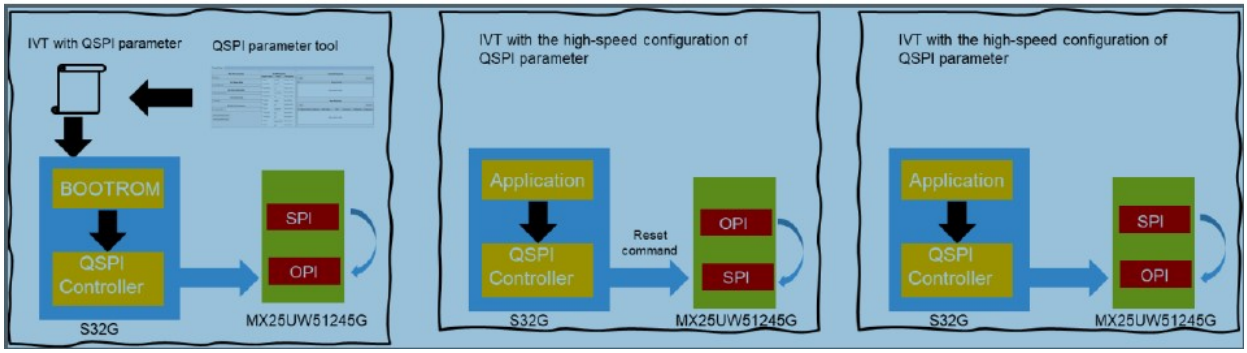
Fls_Example_S32G399A_M7->somId(…)->Fls(…)->Fls->FlsMem-> FlsMem_0：

- Flash Device Name= Gigadevice
- Flash memory alignment (1 -> 16) =1 // Address alignment required for external Flash (1, 2 or 4 bytes...) in OCTA DTR mode (DOPI)
- Enable Ahb Direct Reads = Checked // After setting, QSpi_Ip_AhbReadEnable() will be called from Fls_Init() to allow reading through AHB. The application can be read directly through the address mapping of the Flash device. That is, in addition to the IP access method, you can also read the content mapped by QSPI NOR flash from the AHB address starting with 0x0.
- Flash memory device initial configuration= /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI // The configuration reference that will be used to initialize the Flash device.
- QSPI controller instance= /Fls/Fls/FlsConfigSet/FlsExternalDr/FlsController_0 // The QSPI controller instance to which this Flash device is connected.
- // Note that the above two items are the configurations during QSPI NOR initialization. The description of AN13563 is:

1. The bootROM booted by QuadSPI can be configured with external Flash, and high-speed communication in OPI mode can be realized through the QuadSPI parameters of IVT.

2. The QuadSPI driver initializes the external Flash by sending the reset command to the external Flash. After reset, the external Flash becomes the default state of SPI mode. You need to reconfigure the external Flash and set the QuadSPI controller to the corresponding mode (usually OPI mode)

3. Re configure the external Flash and QuadSPI controllers in OPI mode for the Fls drive to improve QSPI performance.



So Mcal Fls does not depend on the default state of Flash. It will reset to the initialization state, and then reinitialize Flash. This is different from the Fls driver in Bootloader. Therefore, the Fls example has two types of QSPI controller configurations to adapt to external Flash.

1. ControllerCfg_0 shows the configuration of SPI mode of external Flash. Used to initialize Flash in SPI mode.

2. ControllerCfg_1 displays the configuration of OPI mode of external Flash. OPI mode for normal operation.

- Connection type= QSPI_IP_SIDE_A1 // The connection type between the flash device and the controller: QSPI_IP_SIDE_A1-A1 side serial Flash.

5.3 MemCfg Configuration page

Fls_Example_S32G399A_M7->somId(…)->Fls(…)->Fls->MemCfg-> MemCfg_DOPI：

5.3.1 Fls External Configuration page

- Flash device size (0x0 -> 0xffffffff) =0x2000000 // The size of this Flash device (in bytes), GD25LX256E is 32MB.
- Flash device page size (0 -> 4294967295) =256 // The page size (in bytes) of this Flash device. The page size is the maximum amount of data that the Flash device can write in a single write operation// GD25LX256E is 256 Bytes per programmable page
- Read LUT index =/Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/Read_dopi // Reference to the LUT sequence ID that will be used for the read operation, using DOPI mode.
- Write LUT index=/Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/Write_dopi // Reference to the LUT sequence ID that will be used for write operations, using DOPI mode.
- Read Id LUT Index= /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/ReadId_dopi // Refer to the LUT sequence ID, which will be used to read the device/manufacturer ID.
- Read Id size (0 -> 4)= 3 The size of the information returned by the readId command (in bytes)// Generally, it is 1-byte manufacure id and 2-byte device id.

- Fls Qspi Device Id = 0x19:68:C8 // QSPI NOR memory ID. If the related "FLS_E_UNEXPECTED_FLASH_ID" error is enabled, the configured value will be checked according to the value read from memory during initialization. Use the configured read_ID LUT sequence to read the memory ID from the memory. Note: This parameter can only be configured when using Read Id LUT index reference.。

  GD25LX256E is:

Table of ID Definitions
GD25LX256E

Operation Code	M7-M0	ID23-ID16	ID15-ID8	ID7-ID0
9FH/9EH	C8	68	19	FF

And MX25UW51245G= 0x3A:81:C2

Table 10. ID Definitions

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

- Erase type 1 LUT index = /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/Erase_dopi // Erase the LUT sequence ID reference for type 1.
- Erase type 1 size (1 -> 32)=12 // The size of the erased area (in bytes): 2 ^ size; For example, 0x0C represents 4K bytes - Sector of 4K Byte
- Read status register LUT index - initialization = /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/ReadSR // Read the LUT sequence ID reference of the status register command. This sequence is used for the initialization phase. For example, if the initial state of Flash is SPI, this should be a SPI sequence.
- Read status register LUT index = /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/ReadSR_dopi // Read the LUT sequence ID reference of the status register command. The normal mode is DOPI mode.
- Write status register LUT index=/Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/WriteSR_dopi // LUT sequence ID reference for write status register command.
- Status register write enable LUT index=/Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/WriteEnable_dopi // Status register writes LUT sequence ID reference of enable command
- Write enable LUT index=/Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/WriteEnable_dopi // Write the LUT sequence ID reference of the enable command.

  MX25UW51245G's status register of is defined as follows:
- Size in bytes of status register (1 -> 4) =1 // Size of the status register (in bytes)
- 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)

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

So the status registers of corresponding GD25LX256E are defined as follows: they are the same.

- Size in bytes of status register (1 -> 4) =1 // Size of the status register (in bytes)
- 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)=5 ， Value of block protection bitfield (0 -> 15)=0

No.	Bit Name	Description	Note
S7	SRP0	Status Register Protection	Non-volatile writable
S6	BP4	Block Protect Bits	Non-volatile writable
S5	BP3	Block Protect Bits	Non-volatile writable
S4	BP2	Block Protect Bits	Non-volatile writable
S3	BP1	Block Protect Bits	Non-volatile writable
S2	BP0	Block Protect Bits	Non-volatile writable
S1	WEL	Write Enable Latch	Volatile, read only
S0	WIP	Erase/Write In Progress	Volatile, read only

- resetSettings.Reset LUT index= /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/RuntimeReset // eference to the LUT sequence ID from the first command of the reset sequence. Reset command in Runtime status.
- resetSettings. Number of reset commands (1 -> 255) =2 // Number of commands in reset sequence
- initResetSettings.Reset LUT index= /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/ InitReset // Reference to the LUT sequence ID from the first command of the reset sequence. Reset command in SPI 1 line mode during initialization.
- initResetSettings. Number of reset commands (1 -> 255) =2 / Number of commands in reset sequence

- Configure controller on flash Init= /Fls/Fls/FlsConfigSet/FlsExternalDr/ControllerCfg_SDR // Initialization is in SPI SDR 1 line mode.

5.3.2 InitConfigureation Configuration page

This configuration page describes the list of operations that must be performed during initialization to keep the memory in the required operation state. For example, activate XPI mode and 4-byte addressing.

5.3.2.1 Write_cr2_dopi

- Operation type = QSPI_IP_OP_TYPE_RMW_REG // he operation type can be one of the following: QSPI_IP_OP_TYPE_RMW_REG - RMW command on external Flash register
- First LUT index= /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/RDCR // RDCR2 // Index of the first command sequence in Lut; For the RMW type, this is the read command.//Note that the DOPI mode is set in the configuration register 2 for MX25UW51245G, but in the configuration register for GD25LX256E, so the name should be modified first to avoid misunderstanding. The name of the subsequent FlsLUT table should also be modified accordingly
- Second LUT index= /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/WRCR // WRCR2 // The index of the second command sequence in Lut is only used for RMW type, which is a write command.
- Write Enable LUT Index= /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/WriteEnable // Write the index of the enable command, if necessary, before writing the command. For write and RMW operations only.
- //MX25UW51245G's configuration register 2 is defined as:

9-3. Configuration Register 2

Address	Bit	Name	Description	Default
000h	Bit 0	SOPI (STR OPI Enable)	0= STR OPI disable 1= STR OPI enable	0
	Bit 1	DOPI(DTR OPI Enable)	0= DTR OPI disable 1= DTR OPI enable	0
200h	Bit 0	DQSPRC (DTR DQS pre-cycle)	0= 0 cycle 1= 1 cycle	0
	Bit 1	DOS (DQS on STR mode)	0= Disable 1= Enable	0
	Bit [6:4]	DQSSKW (DQ to DQS Skew)	Refer to *"DQ to DQS Skew Table"*	000
300h	Bit [2:0]	DC (Dummy cycle)	Refer to *"Dummy Cycle and Frequency Table (MHz)"*	000
500h	Bit 0	PSB (Pattern Select Bit)	Refer to *"Preamable Pattern Select Bit Table"*	0

- Command address (0 -> 4294967295) =0 //address, if command use it
- Register size (1 -> 4) =1 //The size of the configuration register in bytes.
- Bit-field offset (0 -> 32) =1, Bit-field width (0 -> 32)=1, Bit-field value (0 -> 4294967295)=1

  //GD25LX256E is:

Table 8. Nonvolatile Configuration Register

Addr	Settings	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Description
<0>	I/O mode	1	1	1	1	1	1	1	1	SPI with DQS (Default)
		1	1	0	1	1	1	1	1	SPI W/O DQS
		1	1	1	0	0	1	1	1	Octal DTR with DQS
		1	1	0	0	0	1	1	1	Octal DTR W/O DQS
		1	0	1	1	0	1	1	1	Octal STR with DQS
		1	0	0	1	0	1	1	1	Octal STR W/O DQS
		Others								Reserved

So need configure to:

- Command address (0 -> 4294967295) =0 // address, if command use it
- Register size (1 -> 4) =1 // The size of the configuration register in bytes.
- Bit-field offset (0 -> 32) =0, Bit-field width (0 -> 32)=8, Bit-field value (0 -> 4294967295)=231=0xE7

5.3.2.2 Ext_dqs

- Operation type = QSPI_IP_OP_TYPE_QSPI_CFG // The operation type can be one of the following: QSPI_IP_OP_TYPE_QSPI_CFG – Reconfigure QSPI controller
- Controller configuration = /Fls/Fls/FlsConfigSet/FlsExternalDr/ControllerCfg_DDR_DQS_External // Reference to the configuration that will be used to initialize the controller. Only valid for QSPI_IP_OP_TYPE_QSPI_CFG operation. After initializing QSPI NOR to DOPI mode, reconfigure the control to DDR DQS external mode.

5.3.3 FlsLUT Configuration page

Configuration page used to configure the lookup table containing all instruction/operand sequences. A sequence consists of a series of up to 8 instruction/operand pairs, which can store up to 4 LUTs. These LUTs will be executed whenever a command is triggered to the external Flash. Note that this is the most important part of modifying a new Flash, and it needs to be modified according to the data manual of QSPI NOR.

Because previously in Chapter 2/4, we have analyzed:

- Read_dopi:

	MX25U51245G(Reference)	GD25LX256E(Design)

EB Conf.	FlsInstructionOperandPair Ind... Name .. Fls LUT Instruction Fls LUT Pad Fls... 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	FlsInstructionOperandPair Ind... Name .. Fls LUT Instruction Fls LUT Pad Fls... 0 FlsInstructionOperandPair_0 0 QSPI_IP_LUT_INSTR_CMD_DDR QSPI_IP_LUT_PADS_8 0xfd 1 FlsInstructionOperandPair_1 1 QSPI_IP_LUT_INSTR_CMD_DDR QSPI_IP_LUT_PADS_8 0x2 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 0x10 4 FlsInstructionOperandPair_4 4 QSPI_IP_LUT_INSTR_READ_DDR QSPI_IP_LUT_PADS_8 0x10

- WriteEnable: do not change.

	MX25U51245G(Reference)	GD25LX256E(Design)
EB Conf.	FlsInstructionOperandPair Ind... Name .. Fls LUT Instruction Fls LUT Pad Fls... 0 FlsInstructionOperandPair_0 0 QSPI_IP_LUT_INSTR_CMD QSPI_IP_LUT_PADS_1 0x6	FlsInstructionOperandPair Ind... Name .. Fls LUT Instruction Fls LUT Pad Fls... 0 FlsInstructionOperandPair_0 0 QSPI_IP_LUT_INSTR_CMD QSPI_IP_LUT_PADS_1 0x6

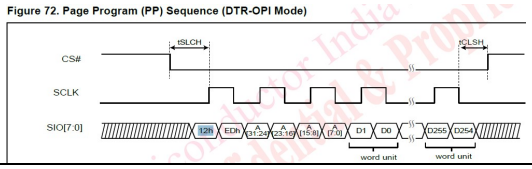
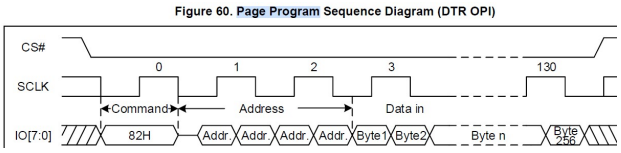
- WRCR2 modified to WRCR command sequence:

	MX25U51245G(Reference)	GD25LX256E(Design)
EB Conf.	FlsInstructionOperandPair Ind... Name .. Fls LUT Instruction Fls LUT Pad Fls... 0 FlsInstructionOperandPair_0 0 QSPI_IP_LUT_INSTR_CMD QSPI_IP_LUT_PADS_1 0x72 1 FlsInstructionOperandPair_1 1 QSPI_IP_LUT_INSTR_ADDR QSPI_IP_LUT_PADS_1 0x20 2 FlsInstructionOperandPair_2 2 QSPI_IP_LUT_INSTR_WRITE QSPI_IP_LUT_PADS_1 0x1	FlsInstructionOperandPair Ind... Name .. Fls LUT Instruction Fls LUT Pad Fls... 0 FlsInstructionOperandPair_0 0 QSPI_IP_LUT_INSTR_CMD QSPI_IP_LUT_PADS_1 0x81 1 FlsInstructionOperandPair_1 1 QSPI_IP_LUT_INSTR_ADDR QSPI_IP_LUT_PADS_1 0x18 2 FlsInstructionOperandPair_2 2 QSPI_IP_LUT_INSTR_WRITE QSPI_IP_LUT_PADS_1 0x1

Therefore, this section only analyzes the remaining items:

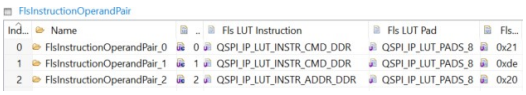
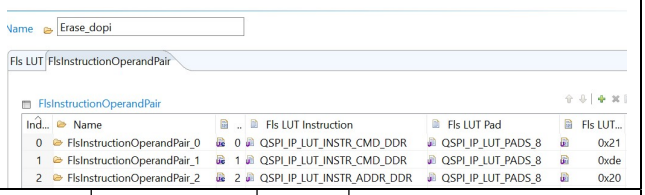
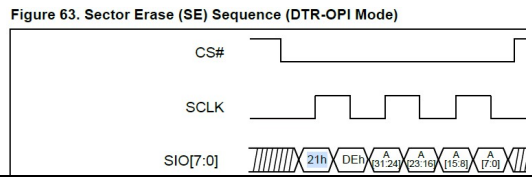
5.3.3.1 Write_dopi

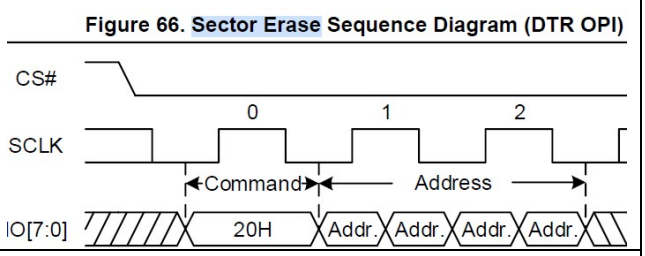
	MX25U51245G(Reference)				GD25LX256E(Design)			
EB Conf.	FlsInstructionOperandPair Ind... Name .. Fls LUT Instruction Fls LUT Pad Fls... 0 FlsInstructionOperandPair_0 0 QSPI_IP_LUT_INSTR_CMD_DDR QSPI_IP_LUT_PADS_8 0x12 1 FlsInstructionOperandPair_1 1 QSPI_IP_LUT_INSTR_CMD_DDR QSPI_IP_LUT_PADS_8 0xed 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				FlsInstructionOperandPair Ind... Name .. Fls LUT Instruction Fls LUT Pad Fls... 0 FlsInstructionOperandPair_0 0 QSPI_IP_LUT_INSTR_CMD_DDR QSPI_IP_LUT_PADS_8 0x82 1 FlsInstructionOperandPair_1 1 QSPI_IP_LUT_INSTR_CMD_DDR QSPI_IP_LUT_PADS_8 0x7d 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			
Details		Instr(6bits)	Pads(2bits)	Operand(8bits)		Instr(6bits)	Pads(2bits)	Operand(8bits)
	4712	0x11(CMD_DDR)	0x3(8 bit)	0x12 0xed(Page Program PP4B)	4782	0x11(CMD_DDR)	0x3(8 bit)	0x82(Page program DTR OPI)
	47ed	0x11(CMD_DDR)	0x3(8 bit)		477d	0x11(CMD_DDR)	0x3(8 bit)	0x7d(0x82 inverted code)
	2b20	0xA(ADDR_DDR)	0x3(8 bit)	0x20(32 Addr bits to be sent on 4 pad)	2b20	0xA(ADDR_DDR)	0x3(8 bit)	0x20(32 Addr bits to be sent on 4 pad)
	3f10	0xF(WRI	0x3(8	0x10(write 16	3f10	0xF(WRIT	0x3(8	0x10(write 16 Bytes on

		TE_DDR)	bit)	Bytes on 4 pad)		E_DDR)	bit)	4 pad)
Timing Dialog							
Comments	The Page Program (PP/PP3B/PP4B) instruction is for programming the memory to be "0". A Write Enable (WREN) instruction must be executed to set the Write Enable Latch (WEL) bit before sending each Page Program (PP/ PP3B/PP4B) command.				The Octal Page Program command is for programming the memory using eight pins: IO[7:0]. A Write Enable (WREN) command must previously have been executed to set the Write Enable Latch (WEL) bit before sending the Page Program command.			

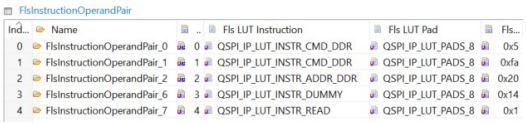
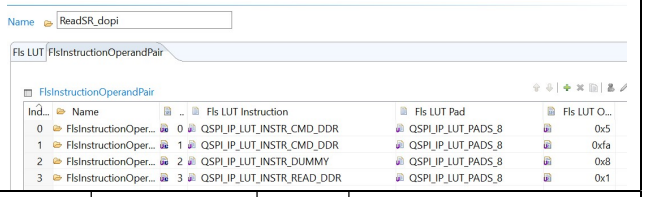
Note that the native MX25U51245G uses the Page Program (PP/PP3B/PP4B) command 02h/12h, which is the same as GD25LX256E. Here, it is modified as: Octal Page Program (82H/84H), so it is also possible to use the old command word. Here, it can be modified or not modified.

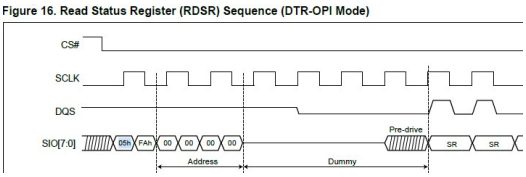
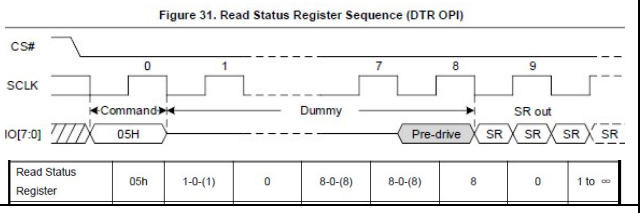
5.3.3.2 Erase_dopi: same, do not need modification

	MX25U51245G(Reference)				GD25LX256E(Design)			
EB Conf.								
Details		Instr(6bits)	Pads(2bits)	Operand(8bits)		Instr(6bits)	Pads(2bits)	Operand(8bits)
	4721	0x11(CMD_DDR)	0x3(8 bit)	0x21 0xde(Sector Erase SE4B)	4721	0x11(CMD_DDR)	0x3(8 bit)	0x21 (Sector Erase DTR OPI)
	47de	0x11(CMD_DDR)	0x3(8 bit)		47de	0x11(CMD_DDR)	0x3(8 bit)	0xde(0x21's inverted code)
	2b20	0xA(ADDR_DDR)	0x3(8 bit)	0x20(32 Addr bits to be sent on 4 pad)	2b20	0xA(ADDR_DDR)	0x3(8 bit)	0x20(32 Addr bits to be sent on 4 pad)
Timing Dialog					**9.22 Sector Erase (SE) (20H/21H)**			

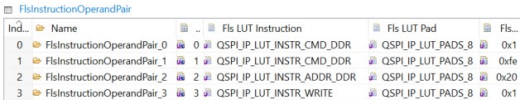
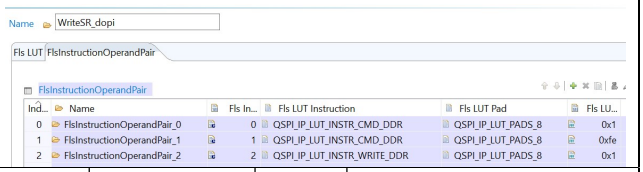
		
Comments	The Sector Erase (SE/SE3B/SE4B) instruction is for erasing the data of the chosen sector to be "1". The instruction is used for any 4K-byte sector. A Write Enable (WREN) instruction must execute to set the Write Enable Latch (WEL) bit before sending the Sector Erase (SE/SE3B/SE4B).	The Sector Erase (SE) command is erased the all data of the chosen sector. A Write Enable (WREN) command must previously have been executed to set the Write Enable Latch (WEL) bit.

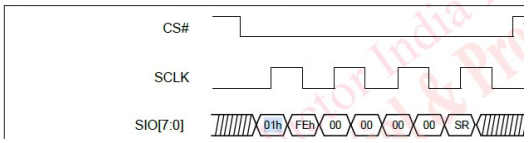
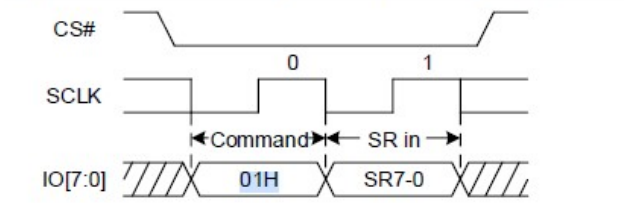
5.3.3.3 ReadSR_dopi

	MX25U51245G(Reference)				GD25LX256E(Design)			
EB Conf.								
Details		Instr(6bits)	Pads(2bits)	Operand(8bits)		Instr(6bits)	Pads(2bits)	Operand(8bits)
	4705	0x11(CMD_DDR)	0x3(8 bit)	0x05 0xfa(RDSR DTR-OPI mode)	4705	0x11(CMD_DDR)	0x3(8 bit)	0x05 (RDSR DTR-OPI)
	47fa	0x11(CMD_DDR)	0x3(8 bit)		4702	0x11(CMD_DDR)	0x3(8 bit)	0xfa(0x05 inverted code)
	2b20	0xA(ADDR_DDR)	0x3(8 bit)	0x20(32 Addr bits to be sent on 4 pad)				
	0f14	0x3(DUMMY)	0x3(8 bit)	0x14(20 dummy cycles)	0f08	0x3(DUMMY)	0x3(8 bit)	0x08(8 dummy cycles)
	1f01	0x7(READ)	0x3(8 bit)	0x1 (Read 1 Bytes on 4 pad)	2b01	0xE(READ_DDR)	0x3(8 bit)	0x1(Read 1Byte on 8 pad) Because it is 8bit mode,change to READ_DDR

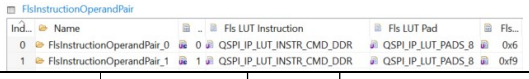
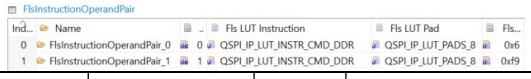
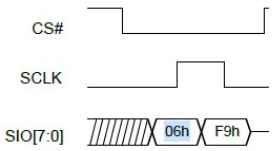
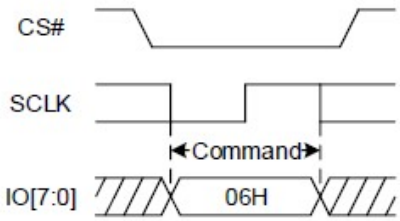
Timing Dialog		
Comments	The RDSR instruction is for reading Status Register Bits. The Read Status Register can be read at any time (even in program/erase/write status register condition). It is recommended to check the Write in Progress (WIP) bit before sending a new instruction when a program, erase, or write status register operation is in progress	The Read Status Register (RDSR) command is for reading the Status Register. The Status Register may be read at any time, even while a Program, Erase or Write Status Register cycle is in progress. When one of these cycles is in progress, it is recommended to check the Write in Progress (WIP) bit before sending a new command to the device. It is also possible to read the Status Register continuously. The SO will output Status Register bits S7~S0.

5.3.3.4 WriteSR_dopi

	MX25U51245G(Reference)				GD25LX256E(Design)			
EB Conf.								
Details		Instr(6bits)	Pads(2bits)	Operand(8bits)		Instr(6bits)	Pads(2bits)	Operand(8bits)
	4701	0x11(CMD_DDR)	0x3(8 bit)	0x01 0xfe(WRSR DTR-OPI Mode)	4701	0x11(CMD_DDR)	0x3(8 bit)	0x01 (WRSR OPI)
	47fe	0x11(CMD_DDR)	0x3(8 bit)		47fe	0x11(CMD_DDR)	0x3(8 bit)	0xfe(0x01 inverted code)
	2b20	0xA(ADDR_DDR)	0x3(8 bit)	0x20(32 Addr bits to be sent on 4 pad)				
	2301	0x08(WRITE)	0x3(8 bit)	0x01(write 1 Bytes on 4 pad)	3F01	0xF(WRITE_DDR)	0x3(8 bit)	0x01(write 1 Bytes on 4 pad) Because it is 8bit mode,change to WRITE_DDR

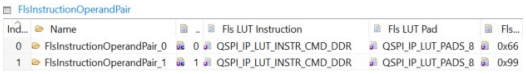
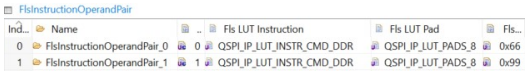
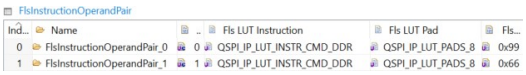
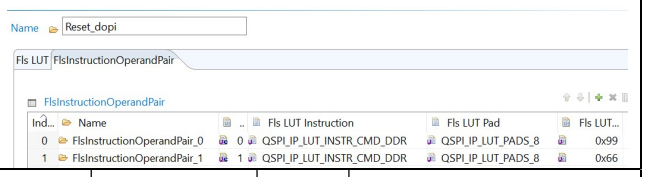
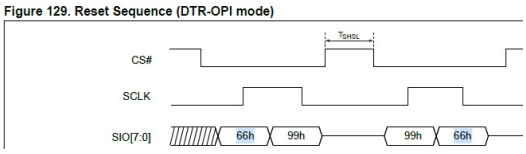
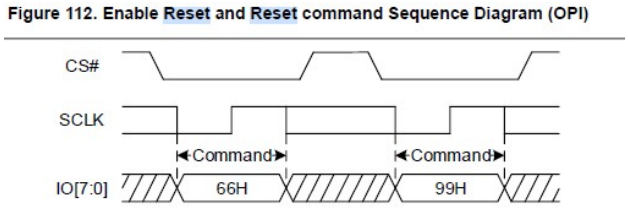
Timing Dialog	Figure 24. Write Status Register (WRSR) Sequence (DTR-OPI Mode)  	Figure 23. Write Status Register Sequence Diagram (OPI) 
Comments	The WRSR instruction is for changing the values of Status Register Bits and Configuration Register Bits. Before sending WRSR instruction, the Write Enable (WREN) instruction must be decoded and executed to set the Write Enable Latch (WEL) bit in advance. The WRSR instruction can change the value of Block Protect (BP3, BP2, BP1, BP0) bits to define the protected area of memory	The Write Status Register (WRSR) command allows new values to be written to the Status Register. Before it can be accepted, a Write Enable (WREN) command must previously have been executed. After the Write Enable (WREN) command has been decoded and executed, the device sets the Write Enable Latch (WEL).

5.3.3.5 WriteEnable_dopi: same, do not need modifcation

	MX25U51245G(Reference)				GD25LX256E(Design)			
EB Conf.	 FlsInstructionOperandPair Ind... Name .. Fls LUT Instruction Fls LUT Pad Fls... 0 FlsInstructionOperandPair_0 0 QSPI_IP_LUT_INSTR_CMD_DDR QSPI_IP_LUT_PADS_8 0x6 1 FlsInstructionOperandPair_1 1 QSPI_IP_LUT_INSTR_CMD_DDR QSPI_IP_LUT_PADS_8 0xf9				 FlsInstructionOperandPair Ind... Name .. Fls LUT Instruction Fls LUT Pad Fls... 0 FlsInstructionOperandPair_0 0 QSPI_IP_LUT_INSTR_CMD_DDR QSPI_IP_LUT_PADS_8 0x6 1 FlsInstructionOperandPair_1 1 QSPI_IP_LUT_INSTR_CMD_DDR QSPI_IP_LUT_PADS_8 0xf9			
Details		Instr(6bits)	Pads(2bits)	Operand(8bits)		Instr(6bits)	Pads(2bits)	Operand(8bits)
	4706	0x11(CMD_DDR)	0x3(8 bit)	0x06 0xf9(WREN DTR-OPI Mode)	4706	0x11(CMD_DDR)	0x3(8 bit)	0x06 (WREN OPI Mode)
	47f9	0x11(CMD_DDR)	0x3(8 bit)		47f9	0x11(CMD_DDR)	0x3(8 bit)	0xf9(0x06 inverted code)
Timing Dialog	Figure 7. Write Enable (WREN) Sequence (DTR-OPI Mode)  				Figure 17. Write Enable Sequence Diagram (OPI)  			
Comments	The Write Enable (WREN) instruction is for setting Write Enable Latch (WEL) bit. For those instructions like PP/ PP3B/PP4B, SE/SE3B/SE4B,				The Write Enable (WREN) command is for setting the Write Enable Latch (WEL) bit. The Write Enable Latch (WEL) bit must be set prior to every Page Program (PP), Sector Erase (SE), Block Erase (BE), Chip Erase (CE),			

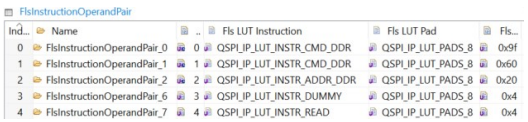
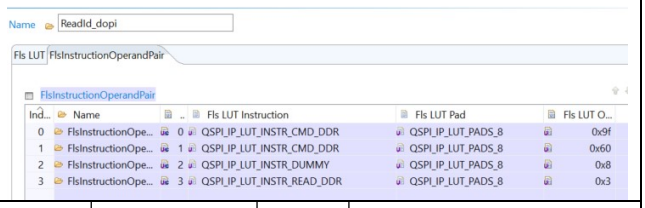
	BE/BE3B/BE4B, CE, WRSR, WRCR2, SBL, WRFBR, ESFBR, WRSCUR, WRLR, WSPB and ESSPB which are intended to change the device content WEL bit should be set every time after the WREN instruction setting the WEL bit. WREN is is also required before initiation of write-to-buffer sequence (WRBI command).	Write Status Register (WRSR), Write Extended Address Register (WEAR), Write Nonvolatile/Volatile configure register and Erase/Program Security Registers command.

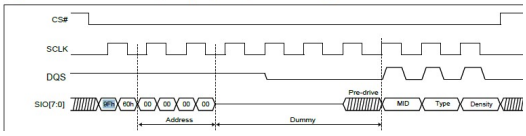
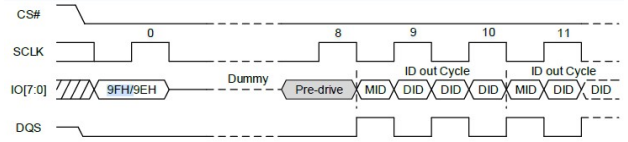
5.3.3.6 ResetEnable_dopi/Reset_dopi: same, do not need modifcation

	MX25U51245G(Reference)				GD25LX256E(Design)			
ResetEnable_dopi EB Conf.								
ResetEnable_dopi Details		Instr(6bits)	Pads(2bits)	Operand(8bits)		Instr(6bits)	Pads(2bits)	Operand(8bits)
	4766	0x11(CMD_DDR)	0x3(8 bit)	0x66 0x99(RSTEN DTR-OPI mode)	4766	0x11(CMD_DDR)	0x3(8 bit)	0x66 (RSTEN)
	4799	0x11(CMD_DDR)	0x3(8 bit)		4799	0x11(CMD_DDR)	0x3(8 bit)	0x99(0x66 inverted code)
Reset_dopi EB Conf.								
Reset_dopi Details		Instr(6bits)	Pads(2bits)	Operand(8bits)		Instr(6bits)	Pads(2bits)	Operand(8bits)
	4799	0x11(CMD_DDR)	0x3(8 bit)	0x99 0x66(RST DTR-OPI mode)	4799	0x11(CMD_DDR)	0x3(8 bit)	0x99 (RST)
	4766	0x11(CMD_DDR)	0x3(8 bit)		4766	0x11(CMD_DDR)	0x3(8 bit)	0x66(0x99 的补码)
Timing Dialog								

Comments	The Software Reset operation combines two instructions: Reset-Enable (RSTEN) command following a Reset (RST) command. It returns the device to a standby mode. All the volatile bits and settings will be cleared then, which makes the device return to the default status as power on. To execute Reset command (RST), the Reset-Enable (RSTEN) command must be executed first to perform the Reset operation. If there is any other command to interrupt after the Reset-Enable command, the Reset-Enable will be invalid. If the Reset command is executed during program or erase operation, the operation will be disabled, the data under processing could be damaged or lost. The reset time is different depending on the last operation	If the Reset command is accepted, any on-going internal operation will be terminated and the device will return to its default power-on state and lose all the current volatile settings, such as Volatile Status Register bits, Write Enable Latch status (WEL), Program/Erase Suspend status, Read Parameter setting (P7-P0), Deep Power Down Mode, Continuous Read Mode bit setting (M7-M0) . When Flash is in OPI Mode, DTR Mode or Continuous Read Mode (XIP), 66H&99H cannot reset Flash to power-on state. Therefore, it is recommended to send the following sequence to reset Flash in these modes: 1. 8CLK with IO<7:0>=all “H” or all “L”: ensure Flash quit XIP mode 2. OPI format 66H/99H: ensure Flash in OPI mode and DTR mode can be reset 3. SPI format 66H/99H: ensure Flash in SPI mode can be reset The “Enable Reset (66H)” and the “Reset (99H)” commands can be issued in either SPI or OPI mode.

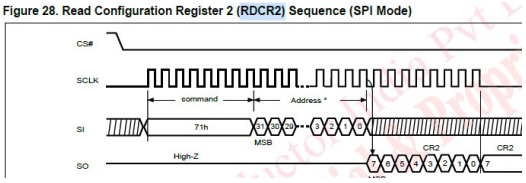
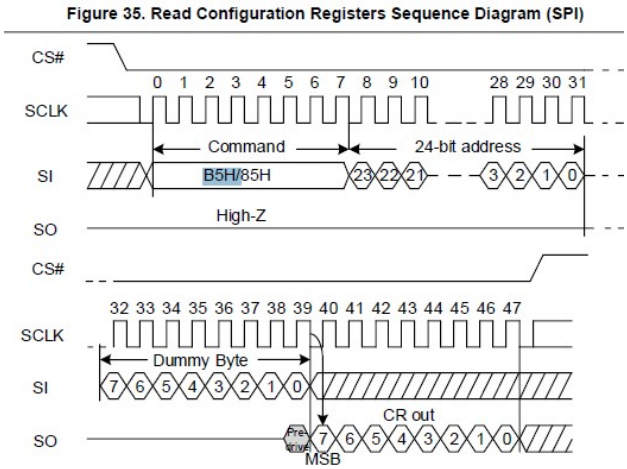
5.3.3.7 ReadId_dopi

	MX25U51245G(Reference)				GD25LX256E(Design)			
EB Conf.								
Details		Instr(6bits)	Pads(2bits)	Operand(8bits)		Instr(6bits)	Pads(2bits)	Operand(8bits)
	479f	0x11(CMD_DDR)	0x3(8 bit)	0x9f 0x60(RDID DTR-OPI mode)	479f	0x11(CMD_DDR)	0x3(8 bit)	0x9f/9e(RDID DTR-OPI)
	4760	0x11(CMD_DDR)	0x3(8 bit)		4760	0x11(CMD_DDR)	0x3(8 bit)	0x60/61(0x9f/9e inverted code)
	2b20	0xA(ADDR_DDR)	0x3(8 bit)	0x20(32 Addr bits to be sent on 4 pad)				
	0f04	0x3(DUMMY)	0x3(8 bit)	0x04(4 dummy cycles)	0f08	0x3(DUMMY)	0x3(8 bit)	0x08(8 dummy cycles)
	1f04	0x07(READ)	0x3(8 bit)	0x04(Read 4 Bytes on 4 pad)	3b04	0xE(READ_DDR)	0x3(8 bit)	0x04(Read 4 Bytes on 4 pad)

								Because it is 8bit mode, change to READ DDR
Timing Dialog	Figure 13. Read Identification (RDID) Sequence (DTR-OPI Mode)  				Figure 84. Read Identification ID Sequence Diagram (DTR, OPI)  Read Identification | 9Fh/9Eh | 1-0-(1) | 0 | 8-0-(8) | 8-0-(8) | 8 | 0 | 1 to 4			
Comments	The RDID instruction is for reading the manufacturer ID of 1-byte and followed by Device ID of 2-byte. The Macronix Manufacturer ID and Device ID are listed as *Table 10* ID Definitions				The Read Identification (RDID) command allows the 8-bit manufacturer identification to be read, followed by three Bytes of device identification. The device identification indicates the memory type in the first Byte, and the memory capacity of the device in the second Byte. The Read Identification (RDID) command while an Erase or Program cycle is in progress, is not decoded, and has no effect on the cycle that is in progress. The Read Identification (RDID) command should not be issued while the device is in Deep Power-Down Mode.			

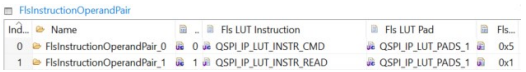
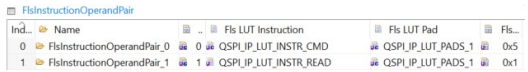
5.3.3.8 Change RDCR2 to RDCR

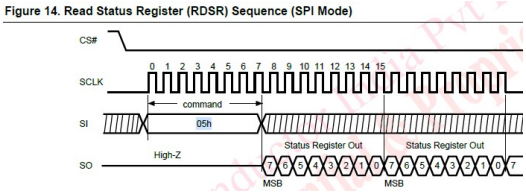
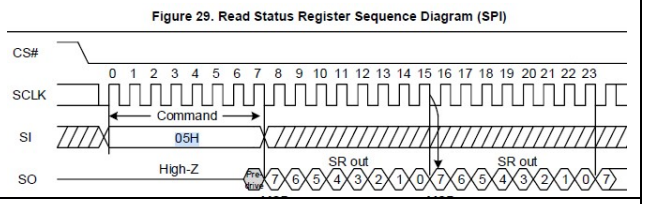
	MX25U51245G(Reference)				GD25LX256E(Design)			
EB Conf.	RDCR2 FlsInstructionOperandPair Ind... | Name | .. | Fls LUT Instruction | Fls LUT Pad | Fls... 0 | FlsInstructionOperandPair_0 | 0 | QSPI_IP_LUT_INSTR_CMD | QSPI_IP_LUT_PADS_1 | 0x71 1 | FlsInstructionOperandPair_1 | 1 | QSPI_IP_LUT_INSTR_ADDR | QSPI_IP_LUT_PADS_1 | 0x20 2 | FlsInstructionOperandPair_2 | 2 | QSPI_IP_LUT_INSTR_READ | QSPI_IP_LUT_PADS_1 | 0x1				RDCR: FlsInstructionOperandPair Ind... | Name | .. | Fls LUT Instruction | Fls LUT Pad | Fls... 0 | FlsInstructionOperandPair_0 | 0 | QSPI_IP_LUT_INSTR_CMD | QSPI_IP_LUT_PADS_1 | 0x85 1 | FlsInstructionOperandPair_1 | 1 | QSPI_IP_LUT_INSTR_ADDR | QSPI_IP_LUT_PADS_1 | 0x18 2 | FlsInstructionOperandPair_2 | 2 | QSPI_IP_LUT_INSTR_DUMMY | QSPI_IP_LUT_PADS_1 | 0x8 3 | FlsInstructionOperandPair_3 | 3 | QSPI_IP_LUT_INSTR_READ | QSPI_IP_LUT_PADS_1 | 0x1			
Details		Instr(6bits)	Pads(2bits)	Operand(8bits)		Instr(6bits)	Pads(2bits)	Operand(8bits)
	0471	0x01(CMD)	0x0(1 bit)	0x71(RDCR2 SPI Mode)	0485	0x01(CMD_)	0x0(1 bit)	0xb5/85 (Read Nonvolatile/Volatile Configuration Register) Attention, read Volatile register.
	0820	0x02(ADDR)	0x0(8 bit)	0x20(32 Addr bits to be sent on 4 pad)	0818	0x02(ADDR)	0x0(1 bit)	0x18(24 Addr bits to be sent on 1 pad)
					0c08	0x3(DUMMY)	0x0(1 bit)	0x08(8 dummy cycles)
	1c01	0x07(REA	0x0(1	0x01(Read 1	1c01	0x07(REA	0x0(1	0x01(Read 1 Bytes on 1

		D)	bit)	Bytes on 1 pad)		D)	bit)	pad)
Timing Dialog								
Comments	The RDCR2 instruction is for reading Configuration Register 2.				The Read Nonvolatile/Volatile Configuration Register command is for reading the Nonvolatile/Volatile Configuration Registers. It is followed by a 3-Byte address (A23-A0) or a 4-Byte address (A31-A0) and a dummy Byte, and each bit is latched-in on the rising edge of SCLK. Then the Configuration Register, at that address, is shifted out on SO, and each bit is shifted out, at a Max frequency fC, on the falling edge of SCLK. Read Nonvolatile/Volatile Configuration Register command, while an Erase, Program or Write cycle is in progress, is rejected without having any effects on the cycle that is in progress.			

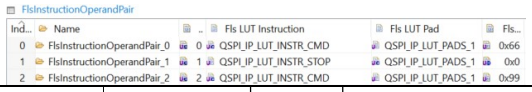
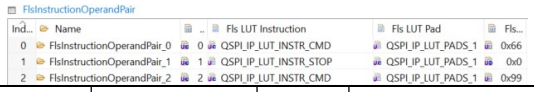
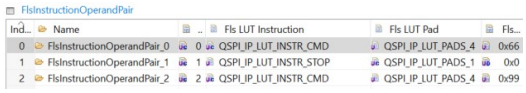
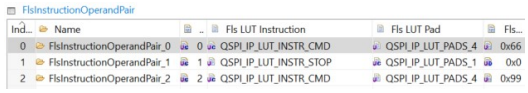
Read Nonvolatile Configuration Register	B5h	1-1-(1)	8	8-8-(8)	8-8-(8)	8	3(4)	1
Read Volatile Configuration Register	85h	1-1-(1)	8	8-8-(8)	8-8-(8)	8	3(4)	1

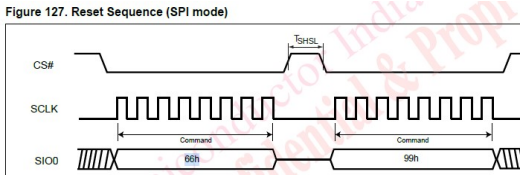
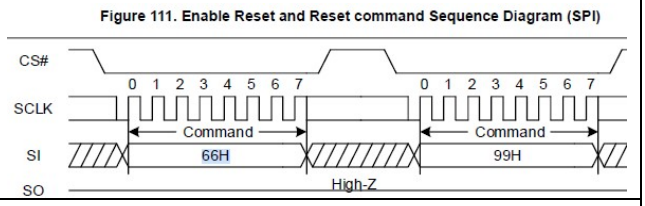
5.3.3.9 ReadSR: same, do not need modifcation

	MX25U51245G(Reference)				GD25LX256E(Design)			
EB Conf.								
Details		Instr(6bits)	Pads(2bits)	Operand(8bits)		Instr(6bits)	Pads(2bits)	Operand(8bits)
	0405	0x01(CMD	0x0(1 bit)	0x05(RDSR SPI mode)	0405	0x01(CMD	0x0(1 bit)	0x05(RDSR SPI mode)
	1c01	0x07(READ)	0x0(1 bit)	0x01(Read 1 Bytes on 1 pad)	1c01	0x07(READ)	0x0(1 bit)	0x01(Read 1 Bytes on 1 pad)

Timing Dialog		
Comments	The RDSR instruction is for reading Status Register Bits. The Read Status Register can be read at any time (even in program/erase/write status register condition). It is recommended to check the Write in Progress (WIP) bit before sending a new instruction when a program, erase, or write status register operation is in progress.	The Read Status Register (RDSR) command is for reading the Status Register. The Status Register may be read at any time, even while a Program, Erase or Write Status Register cycle is in progress. When one of these cycles is in progress, it is recommended to check the Write in Progress (WIP) bit before sending a new command to the device. It is also possible to read the Status Register continuously. The SO will output Status Register bits S7~S0

5.3.3.10 InitReset/ RuntimeReset: same, do not need modifcation

	MX25U51245G(Reference)				GD25LX256E(Design)			
InitRest EB Conf.								
InitRest Details		Instr(6bits)	Pads(2bits)	Operand(8bits)		Instr(6bits)	Pads(2bits)	Operand(8bits)
	0466	0x01(CMD)	0x0(1 bit)	0x66 (RSTEN SPI mode)	0466	0x01(CMD)	0x0(1 bit)	0x66 (RSTEN SPI mode)
	0000	0x0(STOP)	0x0(1 bit)	0x0	0000	0x0(STOP)	0x0(1 bit)	0x0
	0499	0x01(CMD)	0x0(1 bit)	0x99 (RST SPI mode)	0499	0x01(CMD)	0x0(1 bit)	0x99 (RST SPI mode)
RuntimeReset EB Conf.								
RuntimeReset Details		Instr(6bits)	Pads(2bits)	Operand(8bits)		Instr(6bits)	Pads(2bits)	Operand(8bits)
	0666	0x01(CMD)	0x2(4 bit)	0x66 (RSTEN SPI mode)	0666	0x01(CMD)	0x1(1 bit)	0x66 (RSTEN SPI mode)
	0000	0x0(STOP)	0x0(1 bit)	0x0	0000	0x0(STOP)	0x0(1 bit)	0x0
	0699	0x01(CM	0x2(4	0x99 (RST SPI	0699	0x01(CMD)	0x1(1	0x99 (RST SPI mode)

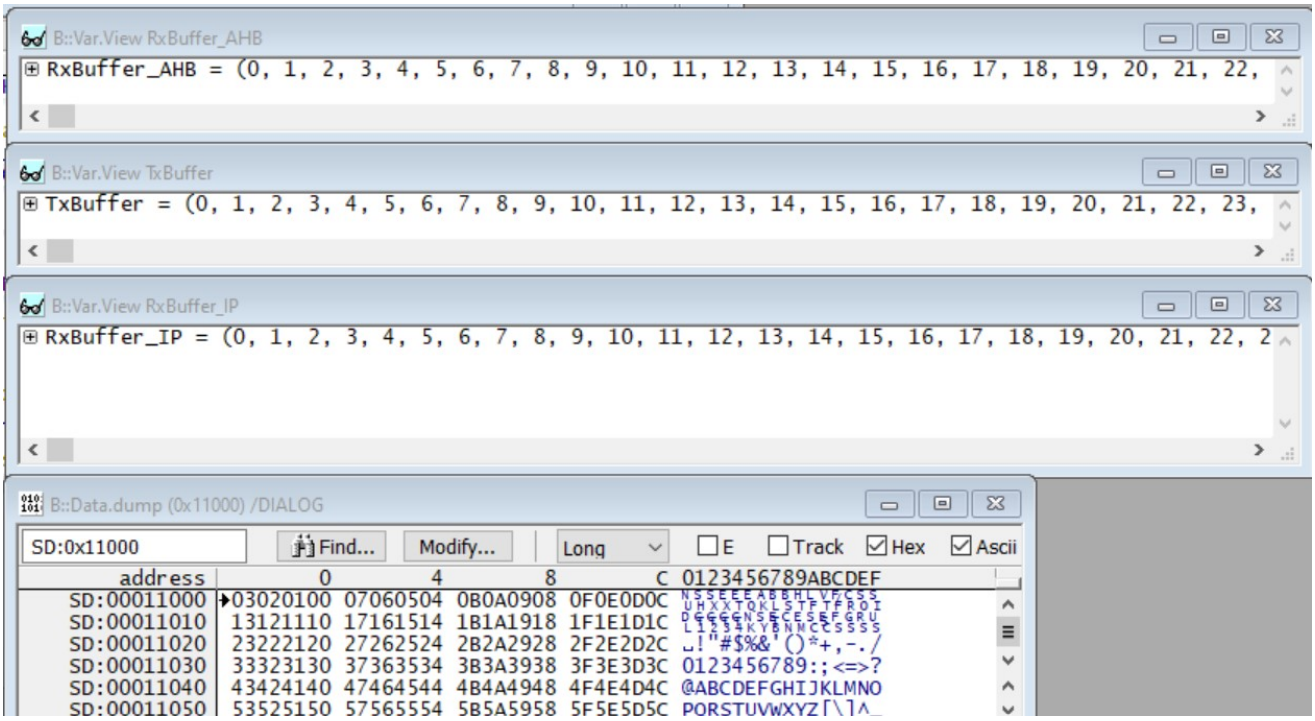
		D)	bit)	mode)			bit)	
Timing Dialog								
Comments	同 ResetEnable_dopi/Reset_dopi				同 ResetEnable_dopi/Reset_dopi			

5.4 Test Report

Use Lauterbach load the script: C:\NXP\SW32G_RTD_4.4_4.0.2\eclipse\plugins\Fls_TS_T40D11M40I2R0\examples\EBT\S32G3\Fls_Example_S32G399A_M7\debug\run.cmm，Stop in main function entry, use Lauterbach to check Fls_Init, Fls_Write, Fls_Read, function call result:

```
/* Compare data in external sector to TxBuffer buffer */
  Fls_Compare(LOGICAL_START_ADDR, TxBuffer, FLS_BUF_SIZE);
  while (MEMIF_IDLE != Fls_GetStatus())
  {
     Fls_MainFunction();
  }
  /* Check last job */
  ExampleAssert(MEMIF_JOB_OK == Fls_GetJobResult()); //可以看到写读后比较成功。
```

And Fls_GetAhbData call result, and then check address: (#define PHYSICAL_START_ADDR 0x11000U /* The HW start address corresponding to the logical address 0 */)：



It indicates that the write, IP read and AHB read data are consistent, which indicates that the drive works normally.

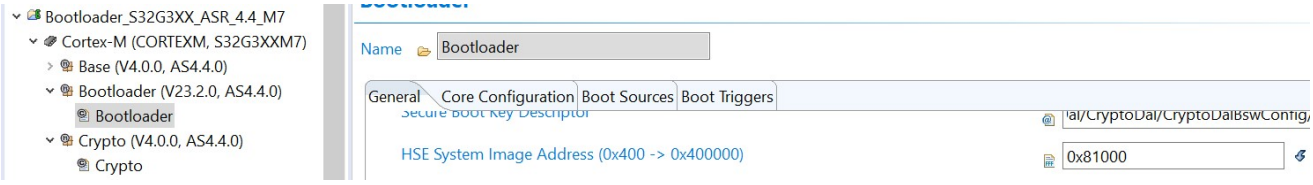
6 Develop Bootloader Project Fls Drivedr

6.1 Bootloader Project Details

Taking the version Integration_Reference_Examples_S32G3_2023_02 as an example, create and modify the Bootloader project according to the document <<S32G_Bootloader_V *. Pdf>>. Note:

- Because Bootloader uses QSPI NOR DMA driver to carry boot image, and uses IP driver to operate QSPI NOR operation related to secure, we choose to remove XRDC and eMMC, but retain secure boot function.
- Turn off the software debugging point.
- After removing eMMC, delete relevant eMMC boot sources in Boot Sources of Bootloader
- Deleted C:\EB\tresos\workspace\Bootloader_S32G3XX_ASR_4.4_M7\output\include, output, src. Delete before generating to prevent old files left by previous generation.
- For GPT problems encountered during compilation, the <<S32G_Bootloader_V *. Pdf>> document explains how to fix.

- Since the IVT configuration SYS-IMG address is 0x81000, the corresponding modification:



- Compiling script modified as follows:

  C:\NXP\Integration_Reference_Examples_S32G3_2023_02\code\framework\realtime\swc\bootloader\platforms\S32G3XX\build\configuration.bat

  SET TRESOS_DIR=C:/EB/tresos

  SET MAKE_DIR=C:/cygwin64

  ::SET GHS_DIR=

  SET GCC_DIR=C:/NXP/S32DS.3.4/S32DS/build_tools/gcc_v9.2

  SET TOOLCHAIN=gcc

  SET CORE=m7

  SET SRC_PATH_DRIVERS=C:/NXP/SW32_RTD_4.4_4.0.0/eclipse/plugins :: Note that this version of Bootloader corresponds to the original RTD version 4.4_4.0.0 by default

  :: SET SDHC_STACK_PATH=

  :: SET SRC_PATH_SAF=

  SET TRESOS_WORKSPACE_DIR=C:/EB/tresos/workspace/Bootloader_S32G3XX_ASR_4.4_M7/output

  SET HSE_FIRMWARE_DIR=C:/NXP/HSE_FW_S32G3_0_2_16_1

- Note that the secure boot will burn the internal anti rollback counter fuse when executing the publish sys img, so the secure boot function test will reduce the fuse resources of the counter. Since this article focuses on the operation of QSPI NOR flash, we modify the code:

C:\NXP\Integration_Reference_Examples_S32G3_2023_02\code\framework\realtime\swc\bootloader\generic\include\Bootloader.h

```
#define BL_ALIGN_4096B(x) BL_ALIGN_IMAGE_B(x, 12) //johnli Used to round the fls erase operation to 4KB
```

C:\NXP\Integration_Reference_Examples_S32G3_2023_02\code\framework\realtime\swc\bootloader\platforms\S32G3XX\src\Bootloader_Specific.c

```
Bl_ConfigureSecureBoot
|-> // Comment out the operation of sys img publish
/*
    if (E_OK == status)
    {
        status = CryptoDal_GetSysImage(
            &Bl_HseSysImage[0], &u32SysImageOffSet, &u32SysImageSize);
```

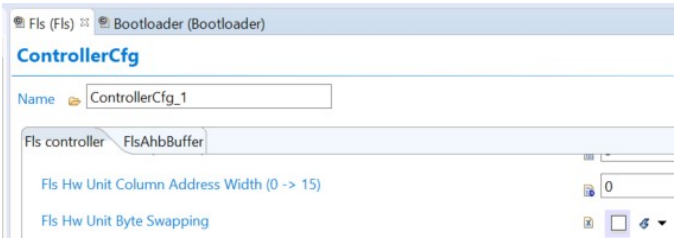
```
    }
*/

volatile int debug_erase;
volatile int debug_write;
|  |->Bl_SaveConfiguration
uint32_t u32SysImageSizeAligned = BL_ALIGN_4096B(u32SysImageSize); //johnli change from 1024 Modify the code to prevent erase data from not being 4KB aligned.
    while(debug_erase); // Stop the code here to use Lauterbach to check the operation of the Fls IP driver |  |->Fls_Erase(u32SysImageStorageAddr + u32SysImageOffset,
        u32SysImageSizeAligned);
while(debug_write);
|  |  |->Fls_Write(u32SysImageStorageAddr + u32SysImageOffset,
        (const uint8 *) pSysImage, u32SysImageSizeAligned);
```

- For the debug method of the Bootloader project, please refer to the document <<S32G_Bootloader_V *. Pdf>>.
- For the corresponding modification details of the Bootloader project, refer to the project source file issued with the document.

6.2 Difference of Bootloader and MCAL Fls Driver

For the modification of QSPI NOR configuration in MCAL Fls driver, the Bootloader still needs to be modified accordingly. In addition, there are the following differences between them:

	Bootloader Fls	MCAL Fls	Comments
	MemCfg_0 Configuration Page		
1	initResetSettings closed	initResetSettings open	Since Bootloader uses the IVT QSPI NOR parameter to initialize QSPI NOR, InitReset and Init reconfiguration are not considered in the driver. In FlsLUT FastRead/Write has no reference
2	initConfiguration closed FlsLUT is the same, Migrate the configuration of MCAL FlsLUT	initConfiguration Open FlsLUT is the same	

	FlsController Configuration Page		
3	ControllerCfg_1	ControllerCfg_SDR=0	MCAL is initialized to SDR mode, and then in initConfiguration, after QSPI NOR is initialized, switch to DDR mode. Bootloader does not need it, but directly initializes to DDR mode
4	ControllerCfg_1. Fls Hw Unit Byte Swapping=unchecked  	ControllerCfg_1. Fls Hw Unit Byte Swapping=unchecked	Note that when bootloader, IVT will be initialized to swapping=0, and the default bootloader is configured as Macronix=1. So after bootloader is copied and run to Fls_init, subsequent access to the AHB address will cause data inversion and failure, so it must be modified here
	ControllerCfg Configuration Page		
5	Cfg_0 is set to DDR Loopback, autoupdate mode, but this Cfg does not reference	Cfg_0 is set to SDR Loopback, bypass mode, and this Cfg is the mode during initialization	Cfg_1 is in DDR, external DQS, autoupdate mode The Cfg_1 is the same, the Bootloader is initialized to this mode, and MCAL Fls is reconfigured to this mode in initConfiguration
	General Configuration Page		

6			Because the clock configurations of Bootloader and MCAL are different, the clock related configurations in the corresponding Fls drive are different
	FlsSector Configuration Page		
7			Sector configuration in Bootloader is generally aligned with the address of the Bootloader image, M7 image and A53 ATF image to be stored, and Mcal is an example

6.3 Image Package

For image packaging and burning, please refer to the document <<S32G_Bootloader_V *. Pdf>>.
Note:

- The IVT head needs the version developed in Chapter 4.
- Select G3 version for SRAM initialization DCD script: C:\NXP\Integration_Reference_Examples_S32G3_2023_02\code\framework\realtime\swc\bootloader\platforms\S32G3XX\res\flash\ S32G3XX_DCD_InitSRAM.bin.
- In addition, when you open the IVT tool, you need to preview and select a created G3 project, so that you can use the G3 20MB SRAM memory to check for out of bounds.
- Since we have chosen to use secure boot, we need to add an HSE image: C: NXP C:\NXP\HSE_FW_S32G3_0_2_16_1\hse\bin\ rev1.1_s32g3xx_hse_fw_0.20.0_2.16.1_pb221011.bin.pink
- Then it is also necessary to refer to the configuration of the Flash drive sector to adjust the distribution of IVT images to avoid overlap:

Sector is configured as:

Name Fls

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

FlsSector

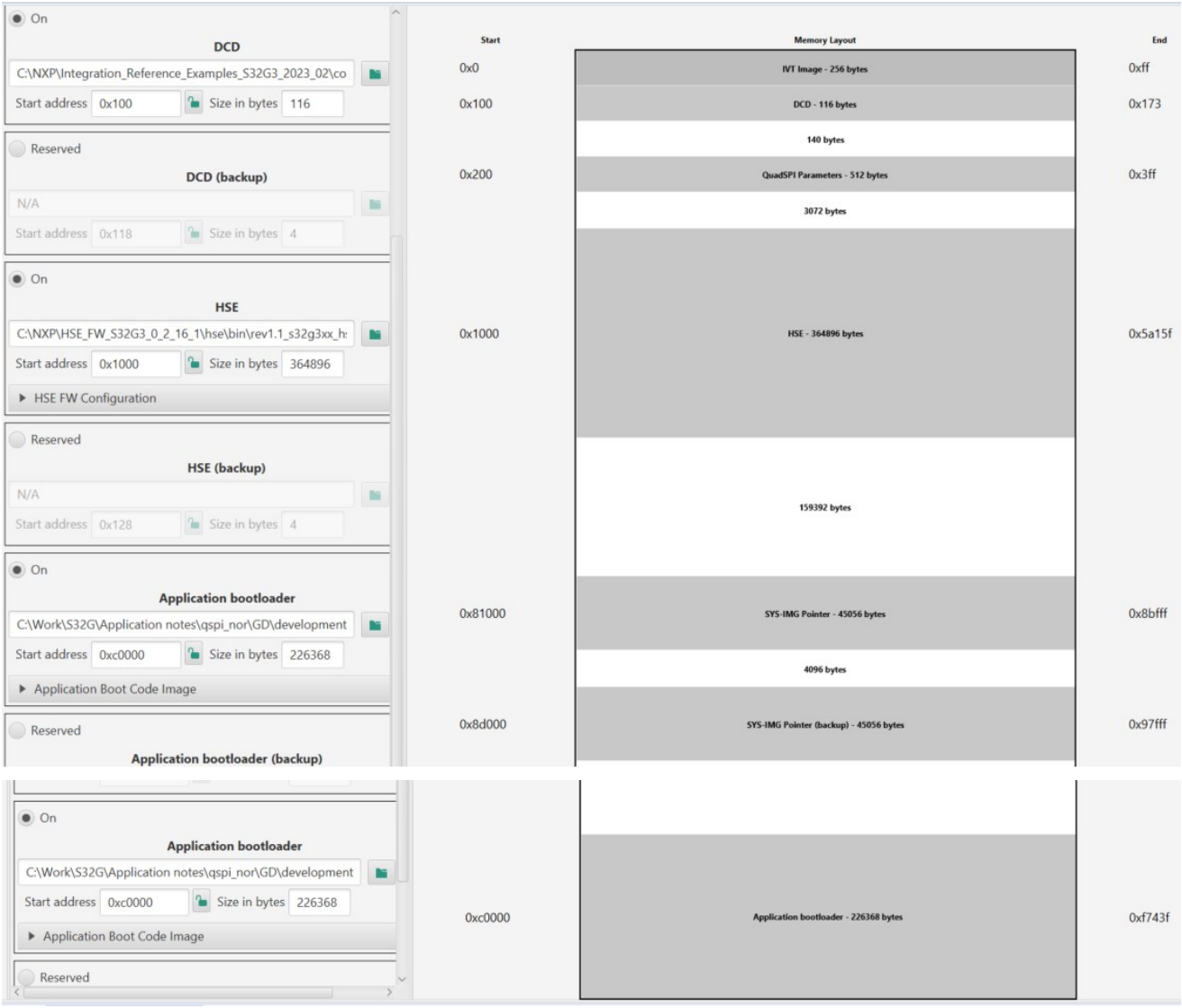
Ind...	Name	F...	Fls Physical Sect...	Fls Numb...	Fls ...	Fls Sector...	Fls Sector...
0	FlsSector_0	0	FLS_EXT_SECTOR	1	16	4096	0
1	FlsSector_1	1	FLS_EXT_SECTOR	1	16	782336	4096
2	FlsSector_2	2	FLS_EXT_SECTOR	1	16	4096	786432
3	FlsSector_3	3	FLS_EXT_SECTOR	1	16	4096	790528
4	FlsSector_4	4	FLS_EXT_SECTOR	1	16	4096	794624
5	FlsSector_5	5	FLS_EXT_SECTOR	1	16	4096	798720
6	FlsSector_6	6	FLS_EXT_SECTOR	1	16	4096	802816
7	FlsSector_7	7	FLS_EXT_SECTOR	1	16	4096	806912

So:

1. Start with 0x0 and store DCD (offset 0x100) and QSPI NOR header (offset 0x200) with size of 0x1000=4096.

2. Start with 0x1000 and store HSE FW and SYS-IMG with the size of 0xBF000=782336.

3. Use the part beginning with 0xc000 to store the bootloader image.



It can be seen that the images to be written are aligned to the 4KB Sector size.

6.4 Test Report

Use the Flash tool to burn the packaged Bootloader image into QSPI NOR flash, switch the startup mode to QSPI NOR flash, and then power on again:

Run the script using Lauterbach:

C:\NXP\Integration_Reference_Examples_S32G3_2023_02\code\framework\realtime\swc\bootloader\platforms\S32G3XX\build\cmm\connect_s32g3xx_m7.cmm, Then the Bootloader code will stay at the beginning. At this time, run the script to connect the debugger, that is, you can debug:

```
    uint32_t u32SysImageSizeAligned = BL_ALIGN_4096B(u32SysImageSize); //johnli change from 1024
641 while(debug_erase); //johnli for test
642 Fls_Erase(u32SysImageStorageAddr + u32SysImageOffset,
            u32SysImageSizeAligned);
644 while (MEMIF_IDLE != Fls_GetStatus())
    {
646     Fls_MainFunction();
    }
648 while(debug_write); //johnli for test
649 Fls_Write(u32SysImageStorageAddr + u32SysImageOffset,
            (const uint8 *) pSysImage, u32SysImageSizeAligned);
```

When the code runs to while (debug_erase) in Bl_SaveConfiguration; After changing debug_erase to 0, you can continue to run. Check the execution of the function Fls_Erase, and check the execution of Fls_Write in the same way.

7 Develop Linux Driver(Optional)

Refer to the document <<S32G_QSPINOR_Customize_ *. Pdf>>, Chapter 9: Software Customization Linux Kernel to learn about the customization method of Linux drivers. There is already an example of Micron MT35XU256ABA. Refer to the following to add GD25LX256E. Take BSP37 as an example.

The QSPI NOR driver of ATF and Uboot is similar to the kernel, and will not be detailed in this article.

7.1 Linux GD Driver Details

\BSP37\linux\drivers\mtd\spi-nor\Makefile:

```
spi-nor-objs            += gigadevice.o
spi-nor-objs            += macronix.o
spi-nor-objs            += micron-st.o
```

\BSP37\linux\drivers\mtd\spi-nor\gigadevice.c, At present, it is relatively primitive. Because Micron and GD flash are compatible, the source code of Micron can be used. You can refer to the example of Micron MT35XU256ABA in <<S32G_QSPINOR_Customize_ *. Pdf>>. Note that the code of BSP37 already supports MT35XU512ABA:

\BSP37\linux\drivers\mtd\spi-nor\micron-st.c

```
static const struct flash_info micron_parts[] = {
    { "mt35xu512aba", INFO(0x2c5b1a, 0, 128 * 1024, 512,
                        SECT_4K | USE_FSR | SPI_NOR_OCTAL_READ |
                        SPI_NOR_4B_OPCODES | SPI_NOR_OCTAL_DTR_READ |
                        SPI_NOR_OCTAL_DTR_PP |
                        SPI_NOR_IO_MODE_EN_VOLATILE)
      .fixups = &mt35xu512aba_fixups},
```

7.2 Modification of Clock

Linux QSPI NOR drive architecture is designed as:：drivers/mtd/spi-nor/core.c：

```
spi_nor_probe
|->spi_nor_scan
|   |->spi_nor_get_flash_info
|   |   |->spi_nor_read_id
struct spi_mem_op op =
                    SPI_MEM_OP(SPI_MEM_OP_CMD(SPINOR_OP_RDID, 1), //#define SPINOR_OP_RDID
            0x9f    /* Read JEDEC ID */
                        SPI_MEM_OP_NO_ADDR,
                        SPI_MEM_OP_NO_DUMMY,
                        SPI_MEM_OP_DATA_IN(SPI_NOR_MAX_ID_LEN, id, 1));
            ret = spi_mem_exec_op(nor->spimem, &op);
|   |   |   |->spi_nor_search_part_by_id(manufacturers[i]->parts,
                                    manufacturers[i]->nparts,
                                    id);
!memcmp(parts[i].id, id, parts[i].id_len))
                    return &parts[i];
```

Therefore, first read the ID value from the external QSPI NOR flash, and then use the ID value to match the data structure array of QSPI NOR related information.

The JEDEC clock for ID reading is not high, as follows: MACRONIX MX25UW51245G flash description is:

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

GD GD25LX256E is:

Symbol	Parameter	Min.	Typ.	Max.	Unit.
f_{C1}	Serial Clock Frequency for all instructions except Read (03H, 13H) in STR mode			166	MHz

At present, the QSPI NOR clock initialized by the S32G Linux BSP is 200Mhz, so there is a risk of incorrect ID reading (the ID value of GD25LX256E read using 200Mhz is incorrect in actual measurement). Since the current Linux SPI NOR driver architecture does not have an API for raising the

frequency after reading the ID, it is considered to use the frequency reduction directly in Linux, as shown in the QSPI NOR clock tree in Linux:

```
periphpll_sel 1 1 0 40000000 0 0 50000
periphpll_vco 8 8 0 2000000000 0 0 50000
|->periphll_dfs1 1 1 0 800000000 0 0 50000
   qspi_sel 1 1 0 800000000 0 0 50000
|   |->qspi_2x 1 1 0 400000000 0 0 50000
|   |   |->qspi_1x 2 2 0 200000000 0 0 50000
```

MC_CGM_0_AC12_DC_0 divide by 2=400Mhz, and divide by 3=266Mhz。

Therefore, without modifying the root clock, the frequency is reduced from the original 200Mhz to 133Mhz for real use, and is modified to:

7.2.1 ATF Modification

Fdts\s32cc.dtsi：

```
mc_cgm0: mc_cgm0@40030000 {
                compatible = "nxp,s32cc-mc_cgm0";
                assigned-clocks =…
                        <&plat_clks S32GEN1_CLK_QSPI_2X>;
                assigned-clock-parents =…
                        <&plat_clks S32GEN1_CLK_PERIPH_PLL_PHI7>;
                assigned-clock-rates =…
                        <S32GEN1_QSPI_2X_CLK_FREQ>;
        };
```

Include\dt-bindings\clock\s32gen1-clock-freq.h

```
#if defined(S32GEN1_QSPI_200MHZ)
#define S32GEN1_PERIPH_DFS1_FREQ      (800 * MHZ)
#define S32GEN1_QSPI_CLK_FREQ         (200 * MHZ)
#define S32GEN1_QSPI_2X_CLK_FREQ      (2 * S32GEN1_QSPI_CLK_FREQ)
#elif defined(S32GEN1_QSPI_166MHZ)
#define S32GEN1_PERIPH_DFS1_FREQ      (666666666)
#define S32GEN1_QSPI_CLK_FREQ         (166666666)
#define S32GEN1_QSPI_2X_CLK_FREQ      (333333333)
#elif defined(S32GEN1_QSPI_133MHZ)
#define S32GEN1_PERIPH_DFS1_FREQ      (800 * MHZ)
#define S32GEN1_QSPI_CLK_FREQ         (133333333)
#define S32GEN1_QSPI_2X_CLK_FREQ      (2 * S32GEN1_QSPI_CLK_FREQ)
```

Plat\nxp\s32\s32g\s32g3\s32g399ardb3\include\platform_def.h

```
#define   S32GEN1_QSPI_133MHZ //johnli for gd S32GEN1_QSPI_200MHZ 此值用用时钟初始化
```

Fdts\s32cc-nxp-flash-macronix.dtsi

```
&qspi {
    macronix_memory: mx25uw51245g@0 {
            compatible = "jedec,spi-nor";
            spi-max-frequency = <133333333>;\\<200000000>;
```

This DTS value is used for uboot drive settings, Therefore, there is no modification in Uboot.

7.2.2 Linux DTS Modification

Arch\arm64\boot\dts\freescale\s32cc.dtsi

```
qspi: spi@40134000 {…
spi-max-frequency =<133333333>;\\ <200000000>;
```

Arch\arm64\boot\dts\freescale\s32cc-nxp-flash-macronix.dtsi

```
&qspi {
    macronix_memory: mx25uw51245g@0 {
            compatible = "jedec,spi-nor";
            spi-max-frequency = <133333333>;\\<200000000>; \\ It is mainly modified here. This value is used to set the Linux drive clock, so the clock is modified to 133Mhz.
```

7.3 In DTS add GD flash Support

\linux\arch\arm64\boot\dts\freescale\s32cc-nxp-flash-macronix.dtsi

```
&qspi {
//    macronix_memory: mx25uw51245g@0 {
      gigadevice_memory: gd25lx256e@0 {
            compatible = "jedec,spi-nor";
            spi-max-frequency = <200000000>;
            spi-tx-bus-width = <8>; //8bit mode
            spi-rx-bus-width = <8>;
            reg = <0>;
            force-soft-reset;
            inverted-cmd-ext; // The second byte of DDR mode command is in inverted mode
            memory-default-octal-dtr; //Support 8bit DDR mode
```

\linux\arch\arm64\boot\dts\freescale\s32g-nxp-flash-macronix.dtsi

```
//&mcronix_memory {
&gigadevice_memory {
```

Then modify the compilation error:

```
diff --git a/arch/arm64/boot/dts/freescale/s32g2xxa-evb.dtsi b/arch/arm64/boot/dts/freescale/s32g2xxa-evb.dtsi
index a5df0a44bce2..8c866db48c05 100644
--- a/arch/arm64/boot/dts/freescale/s32g2xxa-evb.dtsi
+++ b/arch/arm64/boot/dts/freescale/s32g2xxa-evb.dtsi
+/*
 &qspi {
        mx25uw51245g@0 {
                spi-max-frequency = <166666666>;
        };
 };
+*/
diff --git a/arch/arm64/boot/dts/freescale/s32g3xxa-evb.dtsi b/arch/arm64/boot/dts/freescale/s32g3xxa-evb.dtsi
index 46845e7d0d3a..7083a9f9c297 100644
--- a/arch/arm64/boot/dts/freescale/s32g3xxa-evb.dtsi
+++ b/arch/arm64/boot/dts/freescale/s32g3xxa-evb.dtsi
+/*
 &qspi {
        mx25uw51245g@0 {
                spi-max-frequency = <166666666>;
        };
 };
+*/
```

7.4 Modify source code and add flash information structure

\BSP37\linux\drivers\mtd\spi-nor\micron-st.c

```
static const struct flash_info micron_parts[] = {
{ "mt35xu512aba"…,
        // Add a gd25lx256e. Note that the name should correspond to that in DTS.
```

3 MEMORY ORGANIZATION

GD25LX256E

Each device has	Each block has	Each sector has	Each page has	
32M	64/32K	4K	256	Bytes
128K	256/128	16	-	pages
8K	16/8	-	-	sectors
512/1K	-	-	-	blocks

```
{ "gd25lx256e ", INFO(0xc86819, 0, 4 * 1024, 8192, // ID refers to the previous article. Each sector is 4K, 8192 sectors in total
// The gd25lx256e does not need to operate the flag register, so the USE_FSR is removed, and the OCTAL_DTR read and write operations are reset in the fixup, so SPI_NOR_OCTAL_DTR_READ and SPI_NOR_OCTAL_DTR_PP do not need to be configured
                    SECT_4K | SPI_NOR_OCTAL_READ |
                    SPI_NOR_4B_OPCODES | SPI_NOR_OCTAL_DTR_READ |
                    SPI_NOR_IO_MODE_EN_VOLATILE)
    .fixups = &mt35xu512aba_fixups},
```

7.5 Modify the fixup of flash in source code to support DTR mode

Driver/mtd/spi-nor/Macronix.c:

```
static struct spi_nor_fixups mx25uw51245g_fixups = {
    .default_init = mx25uw51245g_default_init, // Used to configure QSPI to enter DTR mode
    .post_bfpt = mx25uw51245g_post_bfpt_fixup, // DTR mode for configuring write operations
    .post_sfdp = mx25uw51245g_post_sfdp_fixup,// DTR mode for configuring write and read operations
};
```

So refer to the fixup configuration of mx25uw51245g, and modify:

```
static struct spi_nor_fixups mt35xu512aba_fixups = {
    .default_init = mt35xu512aba_default_init,
    .post_sfdp = mt35xu512aba_post_sfdp_fixup, // bfpt is a repeated operation and does not need
};
static int spi_nor_micron_octal_dtr_enable(struct spi_nor *nor, bool enable)
{
#if 0
// Keep the configuration of 16 dummy cycles in the configure register, and do not need to modify
```

```
    if (enable) {
            /* Use 16 dummy cycles for memory array reads. */
…
    }
#endif
    ret = spi_nor_write_enable(nor);
…
            *buf = SPINOR_MT_OCT_DTR; #define SPINOR_MT_OCT_DTR          0xe7     /* Enable Octal
DTR. */
    op = (struct spi_mem_op)
            SPI_MEM_OP(SPI_MEM_OP_CMD(SPINOR_OP_MT_WR_ANY_REG, 1),
                    SPI_MEM_OP_ADDR(enable ? 3 : 4,
                            SPINOR_REG_MT_CFR0V, 1), // #define SPINOR_REG_MT_CFR0V
    0x00     /* For setting octal DTR mode */
                    SPI_MEM_OP_NO_DUMMY,
                    SPI_MEM_OP_DATA_OUT(1, buf, 1));

    if (!enable)
            spi_nor_spimem_setup_op(nor, &op, SNOR_PROTO_8_8_8_DTR);
…
#if 0 // Verification code, not required
    /* Read flash ID to make sure the switch was successful. */
…
#endif
    return 0;
}

static void mt35xu512aba_post_sfdp_fixup(struct spi_nor *nor)
{
   // Set the write operation to octal dtr mode
    nor->params->hwcaps.mask |= SNOR_HWCAPS_PP_8_8_8_DTR;
    spi_nor_set_pp_settings(&nor->params->page_programs[SNOR_CMD_PP_8_8_8_DTR],
                                                    SPINOR_OP_PP_4B,
SNOR_PROTO_8_8_8_DTR);
    //end
```

```
    /* Set the Fast Read settings. */ //Set the read operation to cotal dtr mode
    nor->params->hwcaps.mask |= SNOR_HWCAPS_READ_8_8_8_DTR;
    spi_nor_set_read_settings(&nor->params->reads[SNOR_CMD_READ_8_8_8_DTR],
                                0, 16/*johnli gd 20*/, SPINOR_OP_MT_DTR_RD,
                                SNOR_PROTO_8_8_8_DTR);
    nor->params->rdsr_dummy = 8; //johnli gd, read statue register dummy
    nor->params->rdsr_addr_nbytes = 1;//johnli gd :0;
    nor->params->quad_enable = NULL;
}
```

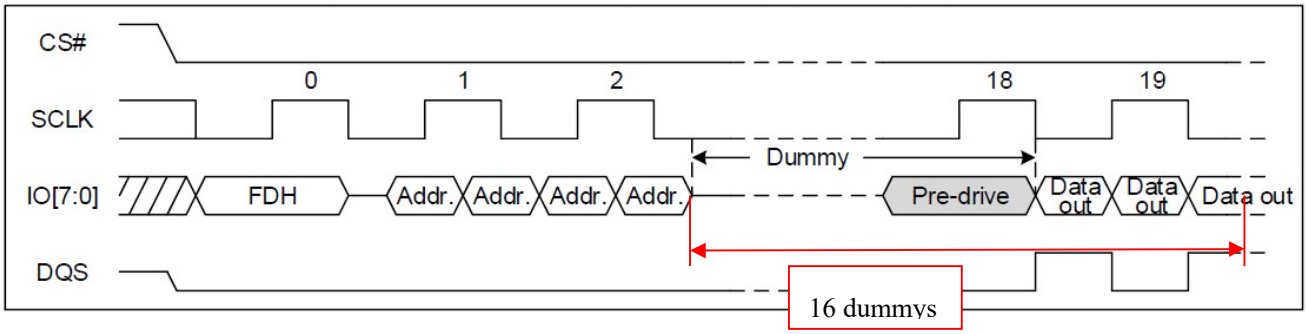
7.6 Turning Dummy Value to Solve the Misplacement Problem

The dummy read by DTR in function mt35xu512aba_post_sfdp_fixup is configured as 16, which is the same as the default configuration of the configuration register in QSPI NOR. After testing

```
root@s32g399ardb3:~# hexdump -v -n 0x100 /dev/mtd0
0000000 5b3e 8135 9776 160c 41a8 990c 66c4 1135
…
00000d0 77b8 a13e d49c 6f7d 2098 d6e2 d49b cb3e
00000e0 6fcf ffff ffff ffff ffff ffff ffff ffff
00000f0 ffff ffff ffff ffff ffff ffff ffff ffff
```

Therefore, the data is offset 2X15 bytes backward. According to the 8bit DTR mode, one clock transmits two bytes, so the dummy value should be 15 clocks ahead.

Figure 54. DTR Octal I/O Fast Read Sequence Diagram (OPI)



Therefore, it should be set to 1. Modify:

```
spi_nor_set_read_settings(&nor->params->reads[SNOR_CMD_READ_8_8_8_DTR],
                                0, 1/*johnli gd 20*/, SPINOR_OP_MT_DTR_RD,
                                SNOR_PROTO_8_8_8_DTR);
```

Test passed：

```
root@s32g399ardb3:~# hexdump -v -n 0x100 /dev/mtd0
0000000 0335 5403 4062 4c55 1b60 78ad 8df9 e45e
…
00000f0 a13e d49c 6f7d 2098 d6e2 d49b cb3e 6fcf
```

The reason why the controller can work only when it does not match the dummy setting in QSPI NOR flash requires the QSPI NOR flash manufacturer's instructions.

7.7 Test Report

Bunry the fsl-image-auto-s32g399ardb3.sdcard image to The TFcard and replace the ATF, Image, and DTB, and then insert the RDB3 board. The startup mode is set to SDcard startup, the USDHC dial is set to SD, connect the serial port and power supply, power on and start linux from the eMMC, and enter the shell:

- Boot information:

```
root@s32g399ardb3:~# dmesg |grep spi
[    0.671341] spi-nor spi6.0: gd25lx256e (32768 Kbytes)
[    0.676482] spi-nor spi6.0: mtd .name = 0.spi, .size = 0x2000000 (32MiB), .erasesize = 0x00001000
(4KiB) .numeraseregions                    = 0
[    0.688277] 7 fixed-partitions partitions found on MTD device 0.spi
[    0.694658] Creating 7 MTD partitions on "0.spi":
[    0.704940] mtd: partition "Flash-Image" extends beyond the end of device "0.spi" -- size truncated to 0x2000000
[    0.741416] mtd: partition "Rootfs" extends beyond the end of device "0.spi" -- size truncated to 0xf10000
```

- Device file:

```
root@s32g399ardb3:~# ls /dev/mtd*
/dev/mtd0    /dev/mtd1    /dev/mtd2    /dev/mtd3    /dev/mtd4    /dev/mtd5    /dev/mtd6
/dev/mtd0ro  /dev/mtd1ro  /dev/mtd2ro  /dev/mtd3ro  /dev/mtd4ro  /dev/mtd5ro  /dev/mtd6r
```

- Clock:

```
root@s32g399ardb3:~# cat /sys/kernel/debug/clk/clk_summary |grep qspi
 qspi_flash1x          2     2     0   133333333     0     0  50000     Y
 qspi_flash2x          0     0     0   266666666     0     0  50000     Y
 qspi_ahb              0     0     0   132206143     0     0  50000     Y
 qspi_reg              0     0     0   132206143     0     0  50000     Y
```

- MTD flash device should be erased before use, because it can only be written from 1 to 0, and can be read out after being erased, All are 0xff;

```
root@s32g399ardb3:~# mtd_debug erase /dev/mtd0 0 0x10000 // The erasure operation takes 4K byte sector size as the address alignment and data alignment
Erased 65536 bytes from address 0x00000000 in flash
root@s32g399ardb3:~# hexdump -v -n 0x100 /dev/mtd0 // For read/write operations, the 256 byte page size is used as the address alignment and data alignment

0000000 ffff ffff ffff ffff ffff ffff ffff ffff
…
00000f0 ffff ffff ffff ffff ffff ffff ffff ffff
0000100…
```

- Write data, read it several times to compare with each other, and compare with the original file. If it is

consistent, the drive works correctly.

```
dd if=/dev/random of=test.txt count=1 bs=256
1+0 records in
1+0 records out
256 bytes copied, 0.000168 s, 1.5 MB/s

root@s32g399ardb3:~# mtd_debug write /dev/mtd0 0 0x100 test.txt
Copied 256 bytes from test.txt to address 0x00000000 in flash

root@s32g399ardb3:~# hexdump -v -n 0x100 /dev/mtd0
0000000 0335 5403 4062 4c55 1b60 78ad 8df9 e45e
0000010 cac4 45e6 1d64 b9d0 9d41 cec9 81d0 5b3e
…
00000f0 a13e d49c 6f7d 2098 d6e2 d49b cb3e 6fcf

root@s32g399ardb3:~# hexdump -v -n 0x100 test.txt
0000000 0335 5403 4062 4c55 1b60 78ad 8df9 e45e
0000010 cac4 45e6 1d64 b9d0 9d41 cec9 81d0 5b3e
…
00000f0 a13e d49c 6f7d 2098 d6e2 d49b cb3e 6fcf
```