Rev. 1, 11/2023

S32G Add GD QSPI NOR Support

by John.Li NXA08200

本文说明在 S32G 平台上增加 GD QSPI NOR flash 的支持，测试使用平台为：

- S32G3 RDB3+ GD25LX256E 32MB QSPI NOR flash。

G2 与 G3 在 QSPI NOR 控制器方面基本相同，所以本文应当同样适用于 G2 平台。

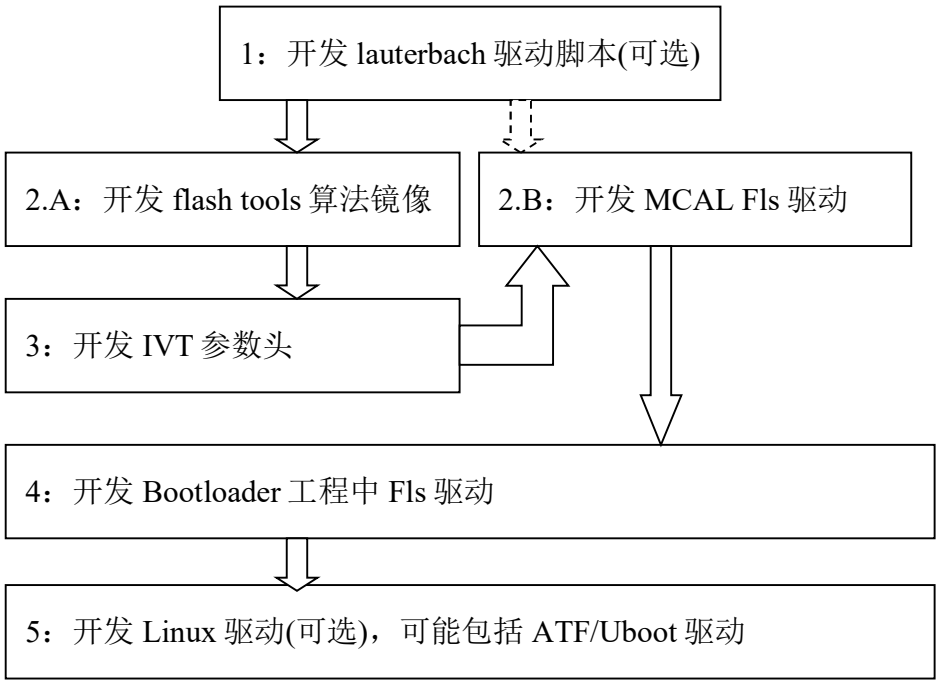
版本	历史	作者
V1	● 创建本文	JohnLi

目录

1 背景和参考资料

1.1 背景说明

本文以GD GD25LX256E为例说明如何在S32G平台上替换一颗新QSPI NOR flash，除了硬件连接外，软件开发流程包括：



说明如下：

1. 可以使用Lauterbach脚本来实现QSPI NOR的驱动，主要应用场景有两种：

- 在板子bring up的时候，使用Lauterbach脚本最简单的读ID驱动来验证硬件。
- 在量产产品出现QSPI NOR启动失败时，可以使用Lauterbach脚本来模拟ROM Code读取QSPI NOR的行为。

另外还有一种情况就是：

- 使用Lauterbach脚本来直接实现烧写镜像，但是这个需要Lauterbach公司提供驱动脚本和算法镜像。

Lauterbach是用于调试目的，所以不是必须要开发的部分。为可选部分。

2. 由于需要使用flash tool来烧写镜像，所以需要先开发flash tool使用的QSPI NOR flash算法镜像。
3. 也可以使用Lauterbach来直接调试MCAL Fls驱动及其测试代码。

4. 需要开发IVT QSPI NOR flash参数头，这样烧写进去的镜像才可以快速启动。
5. 一般最先启动的镜像是bootloader，bootloader需要Fls驱动支持，这个Fls驱动与Mcal中的Fls驱动大部分相同。
6. 一般来讲只能由M7通过MCAL或bootloader来访问QSPI NOR flash，所以一款QSPI NOR flash的支持在完成以上2~5步后就已经完成，不过有的客户会考虑在产线使用Linux内核来快速烧写QSPI NOR flash，所以需要完成Linux驱动(可选)。
7. 由于绝大部分客户的用户是用Bootloader来加载ATF的BL2，而BL2从eMMC中加载ATF其余部分，然后ATF再从eMMC加载uboot，uboot加载内核，所以一般不用实现ATF/Uboot QSPI NOR flash驱动(作为最不可能的可选)。

1.2 参考资料

本文基于 S32G3 RDB3 板+ GD25LX256E QSPI NOR flash 开发：

分类	名称	类别	说明
文档	S32G2RM.pdf S32G3RM.pdf	S32G 芯片手册	从www.nxp.com/s32g下载
软件 RTD Mcal	SW32G_RTD_4.4_4.0.2	BSP 软件+文档	从www.nxp.com个人帐号下载
软件 Boot loader	Platform_Software_Integration_S32G3_2023_02.exe	BSP 软件+文档	从www.nxp.com个人帐号下载
文档	S32G_Bootloader_V*.pdf	Bootloader 定制文档	https://community.nxp.com/ t5/NXP-Designs-Knowledge-Base/ S32G-Bootloader-Customzition/ ta-p/1519838
软件 Linux	BSP37	BSP 文档	从www.nxp.com个人帐号下载
工具	S32Design Studio 3.4.3 或者 3.5.3	S32DS	从www.nxp.com个人帐号下载 参考其中的 QuadSPI 配置工具，编译 Flash_SDK，及 Flash tool。
文档	AN13563: S32G QuadSPI Deep Dive Application note	AppNotes	从www.nxp.com/s32g下载 本文部分内容与之重叠
文档	S32G_RTD_MCAL_V*.pdf	AppNotes	https://community.nxp.com/t5/ NXP-Designs-Knowledge-Base/

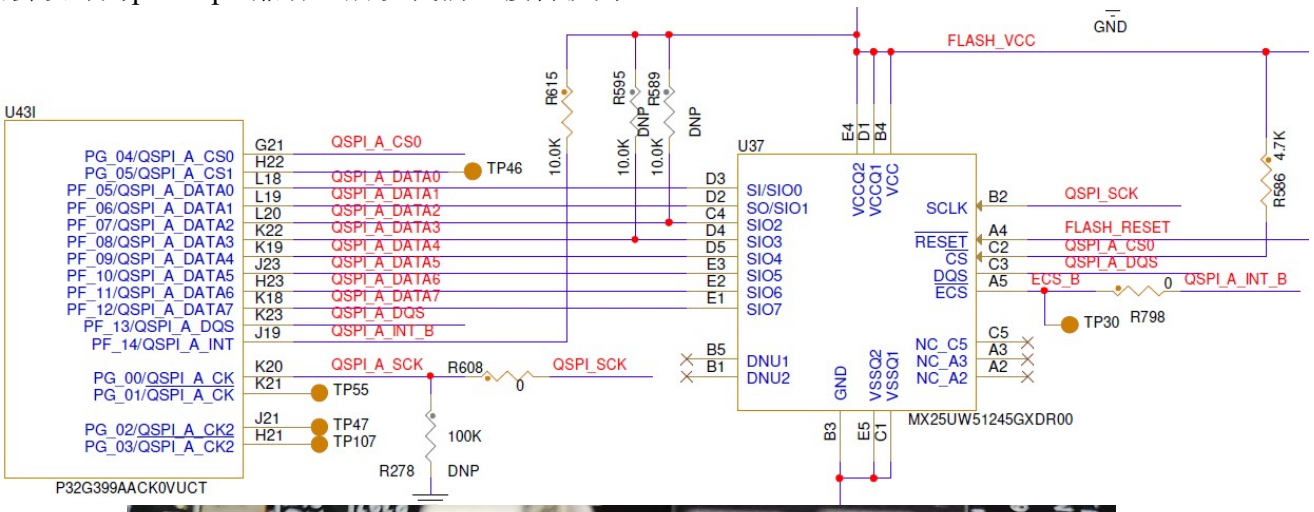
			S32G-MCAL-customization-application-doc /ta-p/1399899 RTD MCAL 驱动测试示例说明
文档	AN12808: Quad SPI (QSPI) Timing Configuration on the S32G2 Vehicle Network Processor Application Note	AppNotes	从www.nxp.com/s32g下载 高速模式下的寄存器配置请参考此文档。
文档	S32G_QSPINOR_定制_*.pdf	AppNotes	从nxp community 下载 https://community.nxp.com/t5/ NXP-Designs-Knowledge-Base/ S32G-QSPI-Nor-customization-doc/t a-p/1399906 关于 flash timing header 配置， flash tools SDK 工程定制(用于开发 flash tools 的 qspi nor binary) uboot 定制，内核驱动定制请参考此文档。 本文部分内容与之重叠
文档	MX25UW51245G.pdf	Macronix QSPI NOR 数据手册	
文档	DS-00762-GD25LX256E-Rev1.1_Automotive.pdf	GD QSPI NOR 数据手册	通过 GD 支持窗口获得
文档	S32G_How_to_Develop_QSPI_Script_*.pdf	QSPI Lauterbach 脚本驱动开发文档	https://community.nxp.com/t5/ NXP-Designs-Knowledge-Base/ S32G-QSPI-Nor-customization-doc /ta-p/1399906 本文部分内容与之重叠

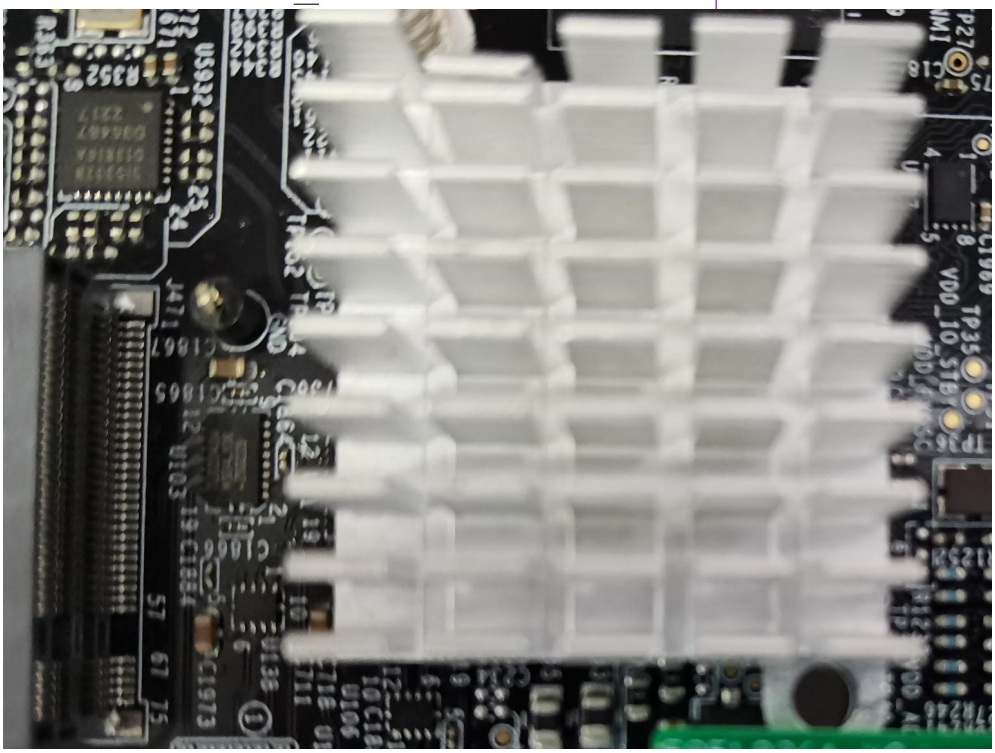
注意：由于本文对每个 QSPI NOR flash 相关驱动开发，是以单独测试为目的，所以没有考虑所有软件版本匹配的问题。对于正式开发的软件版本匹配，建议使用 bundle release：

www.nxp.com/s32g->S32G3->Design Resources->Software->Automotive Software Package Manager->DOWNLOAD->输入注册帐号->S32G3->Integrated Software Bundle。

1.3 硬件连接

S32G3 RDB3连接QSPI NOR原理图如下：使用的是MACRONIX MX25UW51245G flash。Flash一般会设计为pin to pin兼容，所以我们直接替换为GD GD25LX256E。





关于硬件设计相关的注意点，请参考文档《AN13563.pdf：S32G QuadSPI Deep Dive》3.1节：Pin configuration。

软件上NXP默认发布软件已经支持MX25UW51245G，所以本文会用它与GD GD25LX256E对比的方式来说明软件修改过程，同时，有时会参考Micron MT35XU256(512)ABA。

2 Lauterbach 脚本驱动开发(可选)

参考文档《S32G_How_to_Develop_QSPI_Script_*.pdf》了解如何开发 Lauterbach 脚本驱动，本文比较两款 Flash 之间的不同配置，主要是 LUT 配置不同，考虑两个功能：

- QuadSPI_ReadID
- 切换到 DOPI 模式后(QuadSPI_InitDOPI_DLL_AutoUpdateMode_100MHz)，快速读取 QSPI NOR flash：QuadSPI_Read32BytesDOPI

2.1 准备参考脚本

将 C:\NXP\SW32G_RTD_4.4_4.0.2\eclipse\plugins\Fls_TS_T40D11M40I2R0\examples\EBT\S32G3\Fls_Example_S32G399A_M7\debug\device.cmm 拷贝两份，修改为：device_gd _readid.cmm 和 device_gd.cmm。去掉与 M7_0 启动，Disable WDG 和 QSPI NOR 无关的部分。

2.2 QuadSPI_ReadID

device_gd _readid.cmm 主调用为：

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

另外，由于 BYTE SWAP 不同，所以需要修改以下代码：

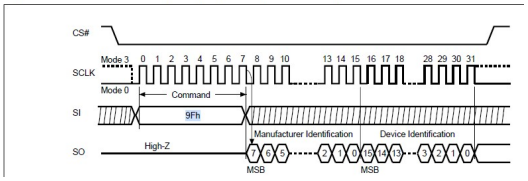
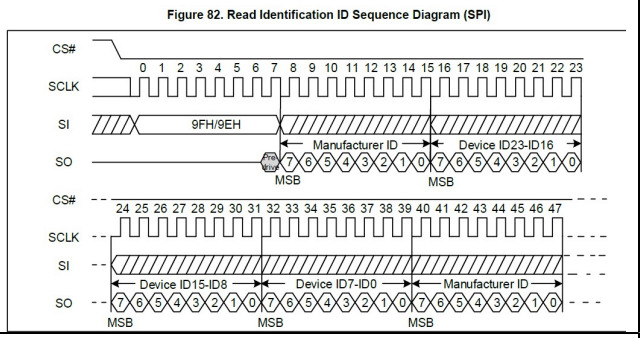
```
        ; write sequence ID and assert Read id command
   Data.Set A:&QSPI_Cntl_BASE+0x08 %Long  (5.<<24.)  ; LUT25  and sequence
   PRINT "1st 0x" Data.Long(A:&QSPI_Cntl_BASE+0x200)>>24. " (Density)"
   PRINT "2nd 0x" (Data.Long(A:&QSPI_Cntl_BASE+0x200)>>16.)&0xFF " (Device ID)"
   PRINT "3rd 0x"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

QuadSPI_ReadID，调用(注意，根据 JEDEC 要求，所有的 QSPI NOR flash 厂商的 ReadID 命令应该相同)：

```
|->
; write sequence ID and assert Read id command
```

Data.Set A:&QSPI_Cntl_BASE+0x08 %Long (5.<<24.) ; LUT20 and sequence

参考 MX25UW51245G 设计 GD25LX256E 命令序列：

<table>
<tr><td></td><td colspan="4">MX25U51245G(参考)</td><td colspan="4">GD25LX256E(设计)</td></tr>
<tr><td>1：Lauterbach 代码</td><td colspan="4">;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</td><td colspan="4">;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</td></tr>
<tr><td rowspan="5">具体分析</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td></tr>
<tr><td>049f</td><td>0x01(CMD)</td><td>0x0(1 bit)</td><td>0x9F(RDID)</td><td>049f</td><td>0x01(CMD)</td><td>0x0(1 bit)</td><td>0x9F/0x9E(RDID)</td></tr>
<tr><td>0818</td><td>0x2(ADDR)</td><td>0x0(1 bit)</td><td>0x18(24 Addr bits to be sent on 1 pad)</td><td>0818</td><td>0x2(ADDR)</td><td>0x0(1 bit)</td><td>0x18(24 Addr bits to be sent on 1 pad)</td></tr>
<tr><td>1c03</td><td>0x7(READ)</td><td>0x0(1 bit)</td><td>0x3 write data size in byte</td><td>1c04
1c03</td><td>0x7(READ)</td><td>0x0(1 bit)</td><td>0x4 write data size in byte
考虑兼容性，也可以只读 3 个 byte</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>时序图</td><td colspan="4">Figure 16. Read Identification (RDID) Sequence (SPI mode only)
  </td><td colspan="4">Figure 82. Read Identification ID Sequence Diagram (SPI)
  </td></tr>
<tr><td>说明</td><td colspan="4">The RDID instruction is for reading the manufacturer ID of 1-byte and followed by 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 rowspan="2">RDID</td><td rowspan="2">9Fh</td><td>Manufacturer ID</td><td>Memory type</td><td>Memory density</td></tr><tr><td>C2</td><td>25</td><td>3A</td></tr></table></td><td colspan="4">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</td></tr>
<tr><td>打印</td><td colspan="8">PRINT "1st 0x" Data.Long(A:&QSPI_Cntl_BASE+0x200)>>24. " (Density)"</td></tr>
</table>

	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

2.3 配置 QSPI NOR 为 DOPI 模式

device_gd.cmm 主调用如下：

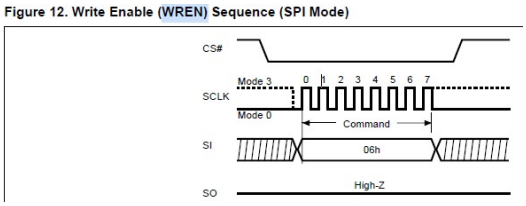
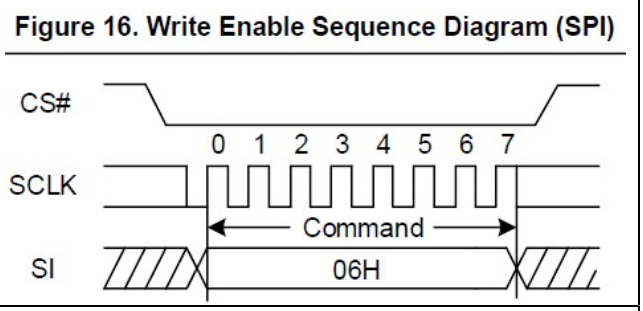
```
GOSUB PERIPH_PLL_1600MHZ
GOSUB PERIPH_PLL_DFS1_800MHZ
GOSUB QuadSPI_PinMux_CLKEnable
GOSUB QuadSPI_InitDOPI_DLL_AutoUpdateMode_100MHz
GOSUB QuadSPI_Read32BytesDOPI
```

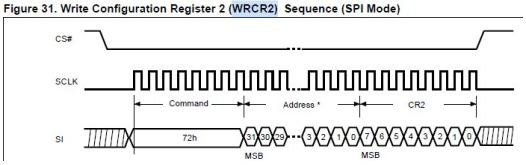
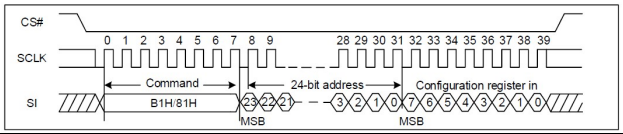
QuadSPI_InitDOPI_DLL_AutoUpdateMode_100MHz，调用：

```
|-> ; 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
```

参考 MX25UW51245G 设计 GD25LX256E 命令序列：

	MX25U51245G(参考)				GD25LX256E(设计)			
Lauterbach 代码 (writeenable)	;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				;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			
具体分析		Instr(6bits)	Pads(2bits)	Operand(8bits)		Instr(6bits)	Pads(2bits)	Operand(8bits)
	0406	0x01(CMD)	0x0(1bit)	0x06(WREN)	0406	0x01(CMD)	0x0(1bit)	0x06(WREN) 相同

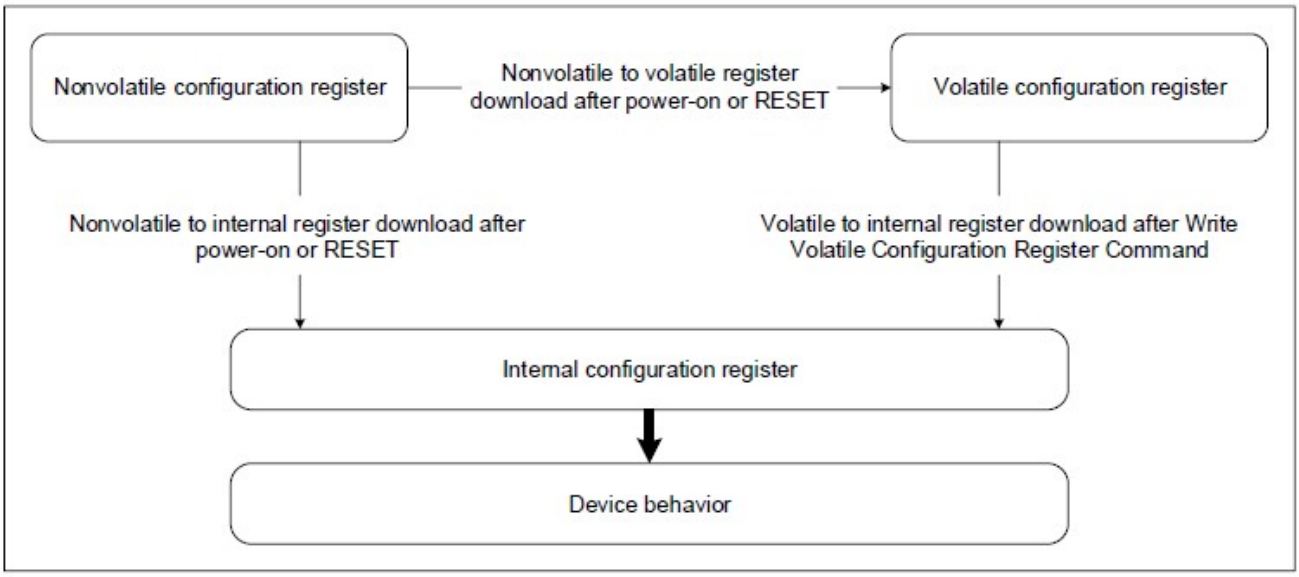
<table>
<tr><td>时序图</td><td colspan="4"> 
</td><td colspan="4"> 
</td></tr>
<tr><td>说明</td><td colspan="4">The Write Enable (WREN) instruction is for setting Write Enable Latch (WEL) bit. For those instructions like PP/
PP4B, 4PP/4PP4B, SE/SE4B,
BE32K/BE32K4B, BE/BE4B, CE, and WRSR, which are intended to change the
device content WEL bit should be set every time after the WREN instruction setting the WEL bit.</td><td colspan="4">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.</td></tr>
<tr><td>Lauterbach 代码 (switch to DOPI mode)</td><td colspan="4">;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</td><td colspan="4">;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</td></tr>
<tr><td rowspan="4">具体分析</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td></tr>
<tr><td>0472</td><td>0x01(CMD)</td><td>0x0(1 bit)</td><td>0x72(WRCR2)</td><td>0481</td><td>0x01(CMD)</td><td>0x0(1 bit)</td><td>0xB1/81(WRCR)</td></tr>
<tr><td>0820</td><td>0x02(ADDR)</td><td>0x0(1 bit)</td><td>0x20(32 Addr bits to be sent on 1 pad)</td><td>0818</td><td>0x02(ADDR)</td><td>0x0(1 bit)</td><td>0x18(24 Addr bits to be sent on 1 pad)</td></tr>
<tr><td>2001</td><td>0x8(WRITE)</td><td>0x0(1 bit)</td><td>0x01 write data size in byte</td><td>2001</td><td>0x8(WRITE)</td><td>0x0(1 bit)</td><td>0x01 write data size in byte</td></tr>
<tr><td>写入值代码：</td><td colspan="4">; 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</td><td colspan="4">; 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 Octal DTR with DQS</td></tr>
</table>

<table>
<tr><td>时序图</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>说明</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="9"><0></td><td rowspan="9">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>

注意，对 GD25LX256E 来说：

用户在做 QSPI NOR 配置时无法直接访问的内部配置寄存器设置。用户可以使用 WRITE NOVOLATILE configuration REGISTER（写入非挥发性配置寄存器）更改上电后的默认配置。非易失性配置寄存器的信息在上电期间或在 reset 之后覆盖内部配置寄存器。

用户可以在设备操作期间使用 WRITE VOLATILE configuration REGISTER 更改配置，命令执行之后，来自易失性配置寄存器的信息在 WRITE 命令完成后立即覆盖内部配置寄存器。



所以在写 configuration 寄存器时，使用命令 0x81 写 volatile 寄存器，直接生效。

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

2.4 使用 DOPI 模式 READ_8DTRD

QuadSPI_Read32BytesDOPI，调用：

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

参考 MX25 设计 GD25 命令序列：

注意，对于 dummy 的设置，MX25U51245G 的建议为：

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*

所以在 200Mhz 时，设置为 0x14=20 个 clock。

GD25LX256E 建议为：

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

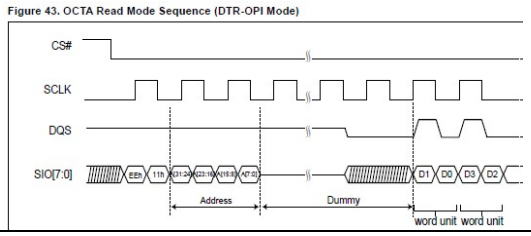
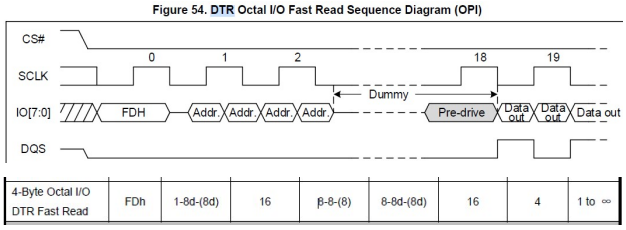
Number of Dummy Clock Cycle	Octal I/O FAST READ		OPI DTR
	STR	DTR	
4	40	40	40
6	84	84	84
8	104	104	104
10	133	133	133
12	152	152	152
14	166	166	166
16 and above	166	200	200

在 200Mhz OPI-DTR 模式下可以设置为 0x10=16 个 clock。具体值请参考表：

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(参考)				GD25LX256E(设计)			
Lauterbach 代码	;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			
具体分析		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 命令可用 1bit	0x10(16 dummy cycles)

							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)
时序图							
说明	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.		
打印	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 测试结果

将 RDB3 板设置为下载模式，或者在 QSPI NOR 烧入带错误 IVT 头的镜像，启动：

1. 运行 Lauterbach: t32marm.exe：File->Open script…= device_gd _readid.cmm。
2. 然后在命令栏里敲 area 命令，就可以看到打印出来的 RX buffer 里的 ID 值：

```
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
```

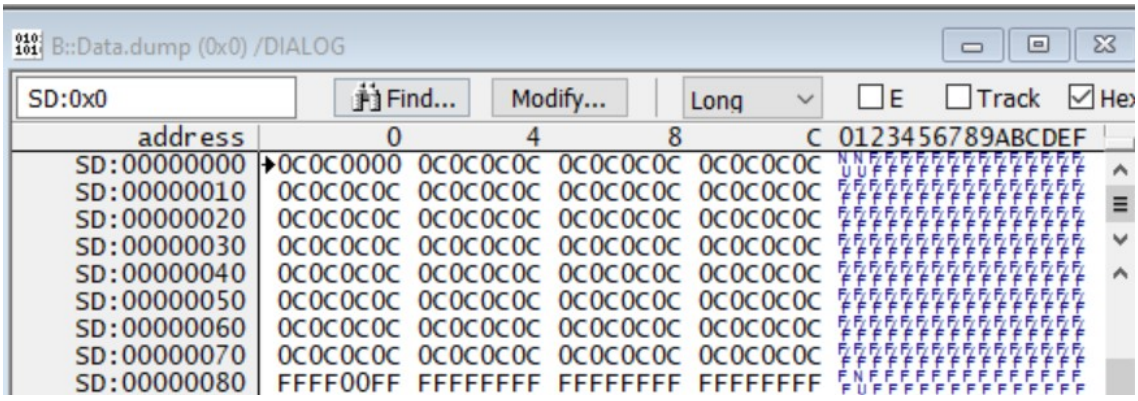
3. 重新运行 Lauterbach: t32marm.exe：File->Open script…= device_gd.cmm。

```
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
```

4. 然后在菜单 View->dump…里，输入地址 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

可以看到 AHB 地址有读出来的 QSPI NOR 内容：



3 Flash tool 算法镜像开发

参考文档《AN13563: S32G QuadSPI Deep Dive Application note》，5.3. Flash SDK usage，了解 Flash tool 镜像开发方法：

镜像开发基于以下原则：

1. 一般来说，同一家厂商的不同型号 Flash 之间，会使用相同的命令字，相同的寄存器设计。
2. 目前 S32DS flash tool 已有的算法镜像在目录 C:\NXP\S32DS.3.4\S32DS\tools\S32FlashTool\flash，包括：

序号	公司	型号
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

所以如果是使用以上三家公司的 flash，可以先尝试使用与之相近的算法镜像烧写，看是否会成功，先选同一型号的，再选同一型号但是尺寸不同的，最后选型号和尺寸可能都不同的。

3. QSPI NOR flash 厂家会考虑一定的兼容，所以可以先对比其命令字和寄存器定义是否相同。
4. 最后才考虑按照 5.3. Flash SDK usage 开发一个新的算法镜像。

3.1 Flash SDK 实现的算法

根据 5.3. Flash SDK usage 说明， flash SDK 实现的算法包括：

- 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 开发新的 flash 源代码

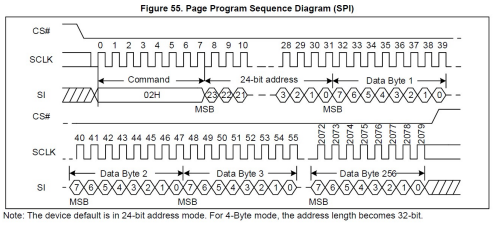
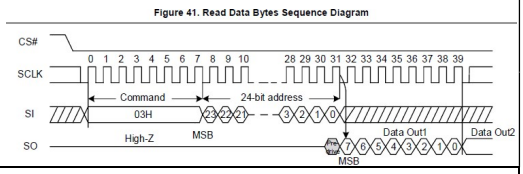
参考文档 C:\NXP\S32DS.3.4\S32DS\help\resources\howto\《HOWTO_Use_FlashSDK_to_add_support_for_QuadSPI_flash_memory_devices_for_S32_Flash_Tool.pdf》可以搭建 S32DS 的 FlashSDK 工程，注意：

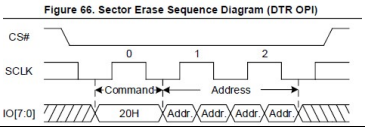
- 在 FlashSDK:Release_FlashTemple 上右击->Build Configurations->Set Active:可以切换 Release 或者 Debug Temple。
- 在 FlashSDK:Release_FlashTemple 上右击->Build Project，可以编译，结果在 Console 窗口，编译出来的镜像在：C:\NXP\S32DS.3.4\S32DS\tools\S32FlashTool\FlashSDK_Ext\Release_FlashTemplate\FlashSDK.bin。

开发一款新的 Flash 算法驱动镜像，主要的修改是文件：C:\NXP\S32DS.3.4\S32DS\tools\S32FlashTool\FlashSDK_Ext\Algo\Generic\ qSPI_chip_commands.h。

比较两款 flash 的命令字与其它定义如下(注意目前 Flash SDK 中使用的均是低速 SPI 模式，非高速模式)：

	MX25UM51245G	GD25LX256E	说明
NOR	0 ///< NOR memory type		pb->is_nand = MEMTYPE;//=NOR=0，说明不支持 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 由于代码中采用 sector programe 的方式，所以 sector 需要采用 1024 和 4*1024 的值，而不能使用 8192 与 4*1024 的值
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	见 2.1 节
READ_ID_DUMMY	0	0	
READ_ID_LENGTH	4	4	
WRITE_ENABLE_CMD	0x06	0x06	见 2.2 节
PAGE_PROGRAM_CMD	0x12	0x12	9.19 Page Program (PP) (02H/12H)
PAGE_PROGRAM_DUMM	0	0	sending Page Program command → 3-Byte address

Y			or 4-Byte address on SI
PAGE_PROGRAM_ADDRESS_LENGTH	32	32	 0x12 为四字节命令字
PAGE_READ_CMD	0x13	0x13	9.14 Read Data Bytes (READ) (03H/13H)
PAGE_READ_DUMMY1	0	0	0x13 为四字节命令字
PAGE_READ_ADDRESS_LENGTH	32	32	
PAGE_READ_DUMMY2	0	0	
READ_STATUS_REGISTER_CMD	0x05	0x05	见 5.3.3.9 节
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	见 2.2 节
WRITE_STATUS_REGISTER_DUMMY	0	0	
WRITE_STATUS_REGISTER_ADDRESS_LENGTH	0	0	
WRITE_STATUS_REGISTER_LENGTH	1	1	
SECTOR_ERASE_CMD	0xDC	0xDC	见 5.3.3.2 节
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	

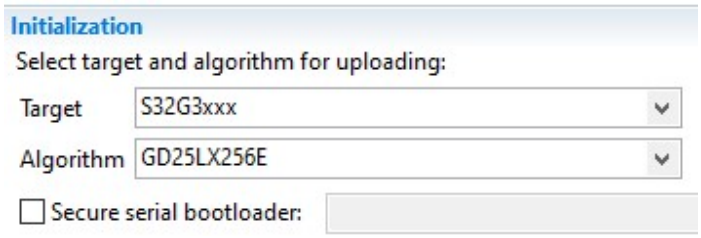
SUBSECTOR_ERASE_ADDRESS_LENGTH	32	32	
CHIP_ERASE_CMD	0x60	0x60	
CLEAR_STATUS_REGISTER_CMD	0	0	which means have no this command
READ_ANY_REGISTER_CMD	0x71	0x85	由于采用 SPI 一线低速模式，也不需要操作 configure 寄存器。 见 5.3.3.8 节
READ_ANY_REGISTER_LENGTH	1	1	
READ_ANY_REGISTER_ADDRESS_LENGTH	32	24	
READ_ANY_REGISTER_DUMMY	0	0	
WRITE_ANY_REGISTER_CMD	0x72	0x81	由于采用 SPI 一线低速模式，也不需要操作 configure 寄存器。 见 2.2 节
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	见 5.3.3.7 节
READ_ID_CMD2_OPI	0xF9	0xF9	
READ_ID_DUMMY_OPI	0	0	
READ_ID_LENGTH_OPI	4	4	

所以，源文件基本修改，直接编译出镜像就可以直接工作。

将 C:\NXP\S32DS.3.4\S32DS\tools\S32FlashTool\FlashSDK_Ext\Release_FlashTemplate\FlashSDK.bin 拷贝到 C:\NXP\S32DS.3.4\S32DS\tools\S32FlashTool\flash。重新命名为：GD25LX256E.bin。然后同样：修改 C:\NXP\S32DS.3.4\S32DS\tools\S32FlashTool\configs\flash_devices.xml，将 GD25LX256E.bin 增加上去：

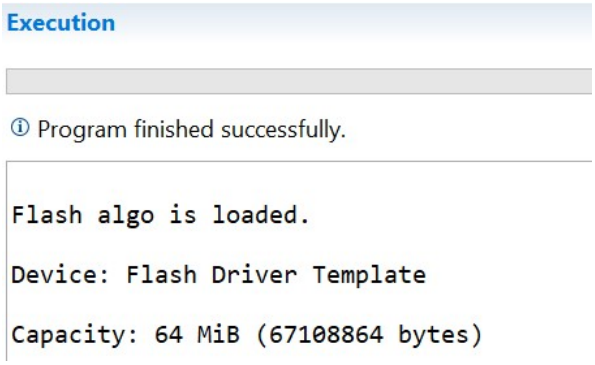
```
<algorithm>
    <id>GD25LX256E</id>
    <name>GD25LX256E</name>
    <path>flash/GD25LX256E.bin</path>
</algorithm>
```

这样在 flash tool 的算法镜像下拉框里就可以看到了：

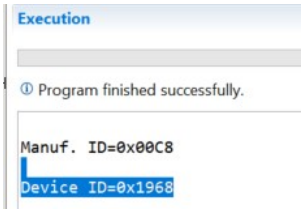


3.3 测试结果

- 使用 Flash 工具，加载算法：Upload target and algorithm to hardware:

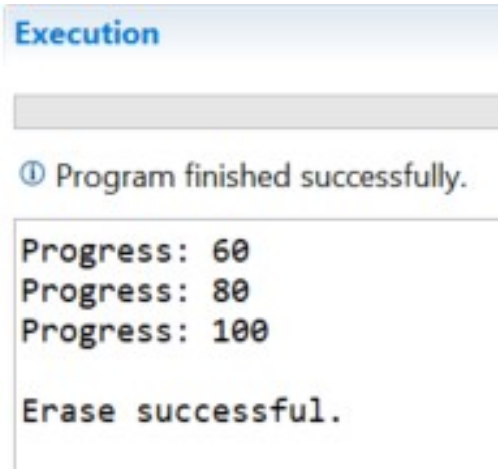


- 成功后使用 Get flash ID：看是否成功：

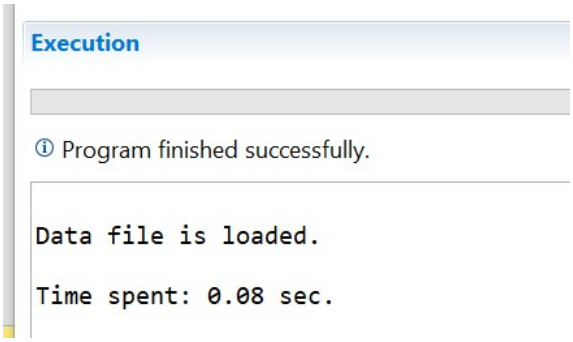


与数据手册相符。

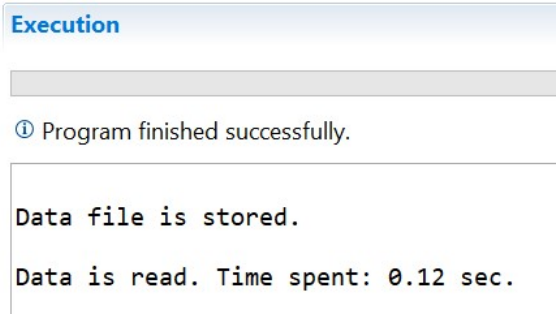
- 然后使用 Flash 工具的擦除功能：Erase memory range，看是否能擦除成功：

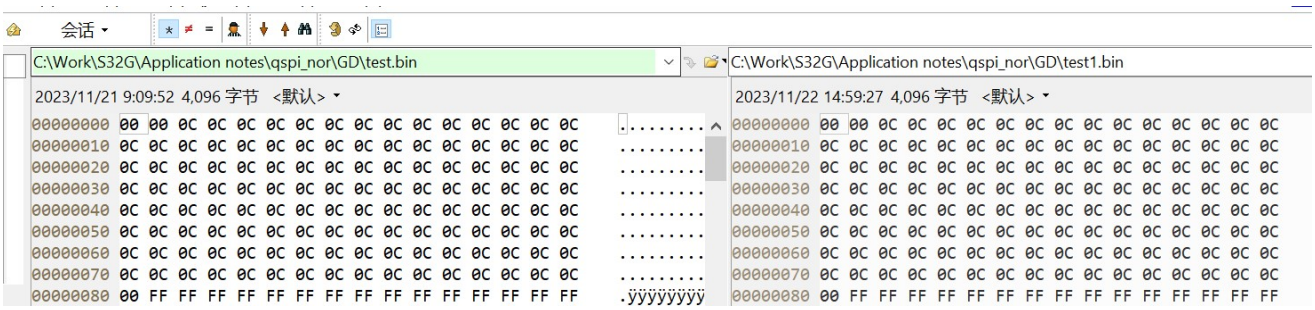


- 再测试 Flash 工具的烧写功能：Upload file to device:



- 最后再使用 Flash 工具读了烧写进去的镜像来做对比：





4 开发 IVT 参数头

参考文档《AN13563: S32G QuadSPI Deep Dive Application note》，章节 4 QuadSPI Boot 和文档《S32G_QSPINOR_定制_*.pdf》，章节 4 S32G QSPI NOR flash 配置表头定制，以下比较三款 flash 在 200Mhz，DDR，External DQS，Auto Update/Bypass 下的配置，及比较出 GD GD25LX256E 合适的配置：

	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

可以看出区别主要在：

4.1 S32G QSPI 控制器配置区别

1. IPCR Trigger

由于：

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.

所以不需要配置成 IP interface 模式，IPCR Trigger 不用勾选，而 AHB 模式需要配置的寄存器为：

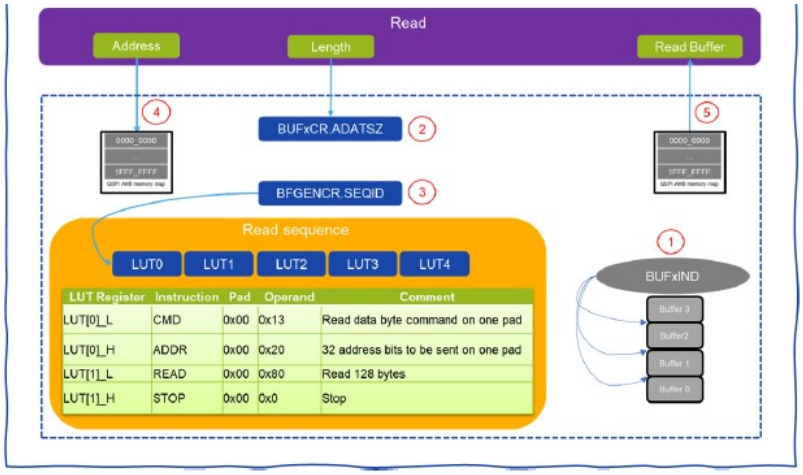


Figure 17. **Read – AHB command**

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

所以需要保证：

- BUF0IND= BUF1IND= BUF2IND=0 //这样 BUFFER3=1024 bytes
- BUF3CR[ALLMST]=1 //所有 master 都可以访问 BUFFER3
- BUF3CR[ADATSZ]值为 0 没有关系，会被 SEQID 的配置重置掉
- BFGEBCR[SEQID]=0 //LUT 配置为 0，所以 IVT 头需要把快速读配置为 LUT0。

使用空的 flash，在 QSPI NOR flash 启动模式下，启动失败后，连接上 lauterbach 确认如下：

```
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
```

所以 ROM Code 默认的代码配置符合要求，另外：

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

所以 ROM Code 默认使用的 LUT0 为：

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 模式选择

参考文档《AN13563: S32G QuadSPI Deep Dive Application note》，章节 3.3.2. Supported DQS sampling method，和文档《AN12808: QSPI Timing Configuration》，章节 3 Sampling the read data from QSPI Flash memory 了解 DQS 模式选择。

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.

注意：

- 为了提高访问速度，建议在 DDR 模式下选择使用 External DQS 模式，而 DDR 模式一般是需要工作在 133Mhz 以上，而通常是工作在 200Mhz，DDR 模式下使用 Pad loopback 最高只能是 66Mhz。
- 对于 133Mhz SDR 模式，可以使用 Pad loopback 模式，不能使用 External DQS 模式。
- External DQS 模式，需要 QSPI NOR 输出时钟到 S32G，所以 QSPI NOR 本身和 PCB 连线会影响到 DQS 信号的质量，所以除了硬件量测，使用 133Mhz SDR Pad loopback 模式也可以作为参考测试。

2. Auto Update 模式与 Bypass 模式的区别：

参考文档《AN13563: S32G QuadSPI Deep Dive Application note》，章节 3.3.3. DLL and DQS delay chain 和《AN12808: QSPI Timing Configuration》，章节 4 DQS delay circuits，了解 DQS delay circuits 的选择办法，

其中 DLL Bypass mode 的设置方法是(动态代码)：

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

B．对以下字段进行编程，以提供采样所需的 DQS 延迟 DLLCRA[SLV_FINE_OFFSET]，DLLCRA[SLV_DLY_COARSE]和 DLLCR[FREQEN]。有关支持的编程设置，请参阅特定于芯片的 QuadSPI 信息。

C．设置 DLLCRA[SLV_UPD]＝1 以将这些值加载到从延迟链中。

D．通过轮询 DLLSR[SLVA_LOCK]=1 检查从延迟链更新状态，并在确认更新状态后清除 DLLCRA[SLV_UPD]

所以最终寄存器的设置为：

SLV_EN=1，SLV_DLL_BYPASS=1, SLAVE_AUTO_UPT=0, SLV_FINE_OFFSET=0 或某个值(精细调用可以先设置为 0)，SLV_DLY_COARSE=5，FREQEN=1(200Mhz 设置为 1，133Mhz 设置为 0)，SLV_UPD=由 1 变 0，最终为 0。

而 DLL Auto Update mode 的配置方法是(动态代码)：

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

B. 使用 DLLCRA[DLL_REFCNTR]和 DLLCRA[DLLRES]对 DLL 配置进行编程。有关支持的 DLL 配置设置，请参阅特定于芯片的 QuadSPI 信息。

C. 通过使用字段 DLLCRA[SLV_FINE_OFFSET], DLLCRA[SLV_DLY_OFFSET]和 DLLCR[FFREQEN]对从属设置进行编程以延迟 DQS。有关支持的设置，请参阅特定于芯片的 QuadSPI 信息。

D. 如果需要在从链上更新偏移延迟，则程序 DLLCRA[SLV_UPD]=1。

E. 通过编程 DLLCRA[DLLEN]=1 启用 DLL，并重置 DLLCRA[SLV_UPD]=0。从延迟链会自动更新，可以通过轮询 DLLSR[SLVA_LOCK]==1 进行检查

所以最终寄存器的设置为：

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=由 1 变 0，最终为 0, DLLEN=1。

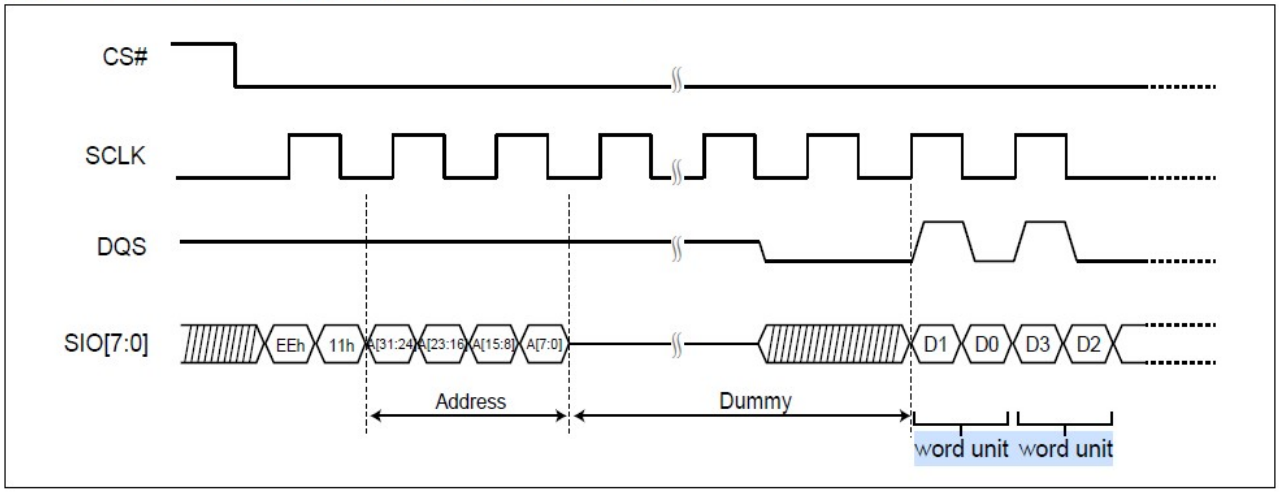
3. BYTE Swapping 不同。

Byte swapping 的定义如下：

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

参考 MX25UW51245G flash 数据手册：

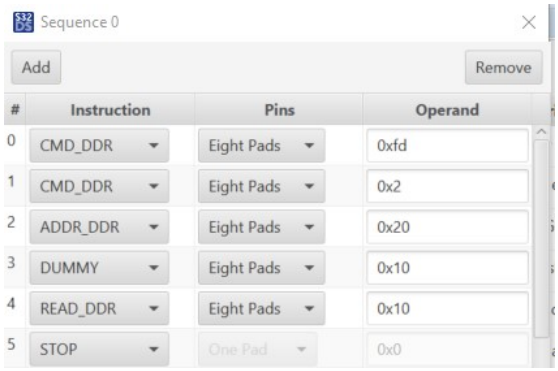
Figure 43. OCTA Read Mode Sequence (DTR-OPI Mode)



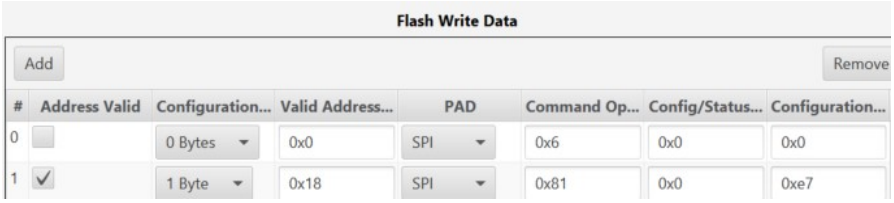
所以一个 word unit 是高位 byte 在前，低位 byte 在后的，所以这儿需要设置为 swap。Micron 和 GD 没有这个要求。

4.2 QSPI 的配置区别

- 参考 2.3 节 使用 DOPI 模式 READ_8DTRD 来配置 GD25LX256E 的 Command Sequences



- 参考 2.2 节 配置 QSPI NOR 为 DOPI 模式来配置 Flash Write Data 来设置 GD25LX256E Flash 为 write enable 后，再设置 QSPI NOR 为 DOPI 模式。



保存为：GD_QSPI_Parametes_200M_DDR__ExternalDQS_Autoupdate.bin

4.3 测试结果

参考文档《S32G_RTD_MCAL_V*.pdf》说明，编译一个 DIO 点灯示例，打好包，注意：

- Configure QuadSPI parameters 选择 S32DS 导出的 IVT QSPI 头镜像：GD_QSPI_Parametes_200M_DDR__ExternalDQS_Autoupdate.bin。
- DCD 段用于初始化 SRAM，或者不用选择：C:\NXP\Integration_Reference_Examples_S32G3_2023_02\code\framework\realtime\swc\bootloader\platforms\S32G3XX\res\flash\S32G3XX_DCD_InitSRAM.bin。
- DIO 点灯示例要求将 GPIO 开关 SW11 置成 on。

然后设置为 QSPI NOR 启动模式，上电启动，可以看到 U128 RGB 灯闪烁。证明启动成功。

注意：

参考文档《S32G_QSPINOR_定制_*.pdf》，章节 10.2 比较大的镜像如果不加参数头，无法从 QSPI-NOR 上启动，所以：

- 如果在没有完成 IVT QSPI NOR 参数头开发的情况下，可以使用 ROM 默认的 1bit 低速模式启动一个小镜像，比如说 Bootloader 镜像，以避免 block bring up。
- 为了加速启动速度，建议完成 IVT QSPI NOR 参数头开发，并在 IVT 中加上参数头。

- 为了加速启动速度，建议使用 200Mhz, DDR, External DQS, Auto update 模式下，对温度等环境因素适应性更好，所以如果可能，尽量采用 Auto update 模式。
- 如果最终的高速模式开发中有问题，可以使用 133Mhz SDR Padloopback，bypass 模式-> 200 Mhz DDR external DQS bypass 模式->200 Mhz DDR external DQS Auto update 模式的顺序逐步升级开发，以避免 block bring up。

5 开发 MCAL Fls 驱动

参考文档《AN13563: S32G QuadSPI Deep Dive Application note》，章节 6. Flash Driver configuration method – EB tresos，了解 Flash MCAL Fls 驱动的开发。注意，Fls 驱动的开发可以使用 Lauterbach 调试，所以此项工作可以安排在 Lauterbach 脚本驱动开发(可选)之后，或者在 Flash tool 算法镜像/IVT 参数头开发后，可以灵活安排。

另外，NXP 默认使用 EB 来配置 Flash 驱动，而一些其它 Autosar 供应商，如 Vector 使用 Davinci 配置工作来配置，两者界面不同，内容一致，本文以 EB 配置为说明。

如同参考文档所说，Fls 驱动配置包括三部分，S32G Flash 控制器，Flash Memory 和 Fls sector，替换一款新的 Flash，主要的工作的关注在 Flash Memory 部分的修改。

5.1 MCAL Fls 驱动工程说明

5.1.1 MCAL Fls 驱动工程

以 SW32G_RTD_4.4_4.0.2 为例：

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.

右键单击工程名 Fls_Example_S32G399A_M7，选择 Generate Project。即生成配置源代码文件。如需要修改配置，修改后保存再重新生成。

打开EB工程的属性，选择Configuration Project->Code Generator，会发现生成的代码默认会放在这个相对路径下："..\..\generate"。所以，按照上一步生成代码后，代码会放到这个相对路径下。这时需要手动将这个相对目录下的所有文件(比如说默认的workspace在C:\EB\tresos\下)，将C:\EB\tresos\generate拷贝到C:\NXP\SW32G2_RTD_4.4_4.0.2\eclipse\plugins\ Fls_TS_T40D11M40I2R0\examples\EBT\S32G3\ Fls_Example_S32G399A_M7\generate\。

用文本文档编辑器打开

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

根据自己PC的路径修改如下参数：

- TOOLCHAIN = gcc //默认MCAL是使用GCC，如需要可修改为GHS。
- GCC_DIR= C:/NXP/S32DS.3.4/S32DS/build_tools/gcc_v9.2/gcc-9.2-arm32-eabi //S32DS对应的GCC编译器的路径(AutoSAR推荐使用GHS)。
- TRESOS_DIR= C:/EB/tresos //该RTD对应的EB Tresos Studio的安装路径。
- T32_DIR= C:/T32 /Lauterbach的调试软件T32的安装路径。
- PLUGINS_DIR //RTD Plugins的路径，默认是相对路径，一般不用修改。
- MCAL_MODULE_LIST := BaseNXP Det Rte Fls MemIf Mcu Port Rm //Fls驱动依赖的Mcal其它模块。

而在 Makefile 中定义了：

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

- FAMILY := S32G3XX

使用cygwin进入路径:

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

键入命令：

```
make build
```

等待编译完成。

5.1.2 Fls 驱动源代码分析

```
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

/根据 InitConfigruation 页的配置来初始化(Reset)QSPI NOR，到 DTR-OPI 模式。

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();
…
```

所以注意 InitConfigureation 配置页中对 QSPI NOR 的重新初始化的配置。

测试见 5.4 节。

5.2 FlsMem 配置页

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

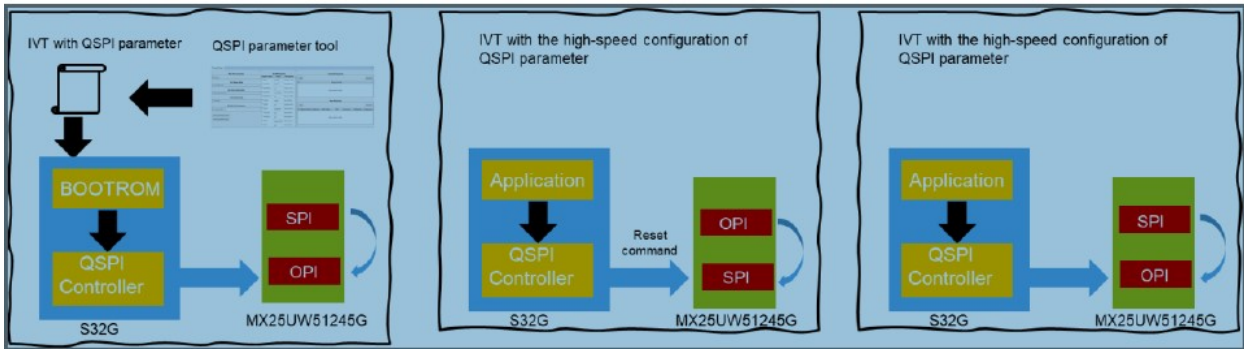
- Flash Device Name= Gigadevice
- Flash memory alignment (1 -> 16) =1 // OCTA DTR 模式（DOPI）中所需的外部 Flash（1、2 或 4 字节…）所需的地址对齐
- Enable Ahb Direct Reads = Checked //设置后，将从 Fls_Init()调用 Qspi_Ip_AhbReadEnable()，以允许通过 AHB 进行读取。该应用程序可以直接通过 Flash 设备的地址映射进行读取。也就是除了 IP access 的方法，也可以从 0x0 开头的 AHB 地址读到 QSPI NOR flash 映射过来的内容。
- Flash memory device initial configuration= /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI // 将用于初始化 Flash 设备的配置参考。
- QSPI controller instance= /Fls/Fls/FlsConfigSet/FlsExternalDr/FlsController_0 //此 Flash 设备所连接的 QSPI 控制器实例。

  //注意以上两条为 QSPI NOR 初始化时的配置，《AN13563》说明为：

1.QuadSPI 引导的 BootROM 可以配置外部 Flash，通过 IVT 的 QuadSPI 参数实现为 OPI 模式的高速通信。

2.QuadSPI 驱动程序通过向外部 Flash 发送 reset 命令初始化外部 Flash。Reset 后外部 Flash 变成 SPI 模式的默认状态。需要重新配置外部 Flash，将 QuadSPI 控制器设置为相应的模式(一般又配置为 OPI 模式)

3.Fls 驱动器再次配置外部 Flash 和 QuadSPI 控制器为 OPI 模式，以提高 QSPI 性能。



所以 Mcal Fls 是不依赖与 Flash 的默认状态的，它都会重置为初始化状态，然后重新初始化 Flash，这个与 Bootloader 中的 Fls 驱动不同，所以 Fls 示例有两种类型的 QSPI 控制器配置，以适应外部 Flash。

1.ControllerCfg_0 显示了外部 Flash 的 SPI 模式的配置。用于在 SPI 模式下初始化 Flash。

2.ControllerCfg_1 显示外部 Flash 的 OPI 模式的配置。用于正常工作的 OPI 模式。

- Connection type= QSPI_IP_SIDE_A1 // flash 设备与控制器的连接类型：QSPI_IP_SIDE_A1-A1 侧连接的串行 Flash。

5.3 MemCfg 配置页

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

5.3.1 Fls External 配置页

- Flash device size (0x0 -> 0xffffffff) =0x2000000 //此 Flash 设备的大小(以字节为单位)，GD25LX256E 为 32MB。
- Flash device page size (0 -> 4294967295) =256 //此Flash设备的页面大小(以字节为单位)。页面大小是Flash设备在单个写入操作中可以写入的最大数据量。//GD25LX256E为256 Bytes per programmable page
- Read LUT index =/Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/Read_dopi //将用于读取操作的 LUT 序列 ID 的参考,使用 DOPI 模式。
- Write LUT index=/Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/Write_dopi //将用于写入操作的 LUT 序列 ID 的参考, 使用 DOPI 模式。
- Read Id LUT Index= /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/ReadId_dopi //参考 LUT 序列 ID，该 ID 将用于读取设备/制造商 ID。
- Read Id size (0 -> 4)= 3 readId 命令返回的信息的大小（以字节为单位）。//一般为 1 字节 manufacutre id，两字节 device id。
- Fls Qspi Device Id = 0x19:68:C8 //外部内存 ID。如果启用了相关的“FLS_E_UNEXPECTED_FLASH_ID”错误，则在初始化时，将根据从内存读取的值检查配置的值。使用配置的 read_ID LUT 序列从存储器中读取存储器 ID。注意：只有在使用 Read Id LUT 索引引用时，才能配置此参数。

  GD25LX256E 为：

Table of ID Definitions
GD25LX256E

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

对应：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 //擦除类型 1 的 LUT 序列 ID 参考。
- Erase type 1 size (1 -> 32)=12 //擦除区域的大小(以字节为单位)：2^大小；例如 0x0C 表示 4K 字节 - Sector of 4K-Byte
- Read status register LUT index - initialization = /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/ReadSR //读取状态寄存器命令的 LUT 序列 ID 参考。此序列用于初始化阶段。例如，如果 Flash 的初始状态是 SPI，则这应该是 SPI 序列。
- Read status register LUT index = /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/ReadSR_dopi //读取状态寄存器命令的 LUT 序列 ID 参考。正常模式下是 DOPI 模式。
- Write status register LUT index=/Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/WriteSR_dopi //写入状态寄存器命令的 LUT 序列 ID 参考。
- Status register write enable LUT index=/Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/WriteEnable_dopi //状态寄存器写入启用命令的 LUT 序列 ID 参考
- Write enable LUT index=/Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/WriteEnable_dopi // 写入启用命令的 LUT 序列 ID 参考。

MX25UW51245G 的状态寄存器定义如下：

- Size in bytes of status register (1 -> 4) =1 //状态寄存器的大小(以字节为单位)
- 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

所以相应 GD GD25LX256E 的状态寄存器定义如下：是相同的。

- Size in bytes of status register (1 -> 4) =1 //状态寄存器的大小(以字节为单位)
- 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 //对来自重置序列的第一个命令的 LUT 序列 ID 的引用。Runtime 状态下的 reset 命令。
- resetSettings. Number of reset commands (1 -> 255) =2 //重置序列中的命令数
- initResetSettings.Reset LUT index= /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/ InitReset //对来自重置序列的第一个命令的 LUT 序列 ID 的引用。初始化时 SPI 1 线模式下的 reset 命令。
- initResetSettings. Number of reset commands (1 -> 255) =2 //重置序列中的命令数
- Configure controller on flash Init= /Fls/Fls/FlsConfigSet/FlsExternalDr/ControllerCfg_SDR //初始化是在在 SPI SDR 1 线模式下。

5.3.2 InitConfigureation 配置页

此配置页描述了在初始化时必须执行的操作列表，以使内存处于所需的操作状态。比如：激活 XPI 模式，激活 4 字节寻址。

5.3.2.1 Write_cr2_dopi

- Operation type = QSPI_IP_OP_TYPE_RMW_REG //操作类型可以是以下类型之一：QSPI_IP_OP_TYPE_RMW_REG-外部 Flash 寄存器上的 RMW 命令

- First LUT index= /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/RDCR // RDCR2 // Lut 中第一个命令序列的索引；对于 RMW 类型，这是读取命令 //注意设置 DOPI 模式在 MX25UW51245G 是在 configureation register 2 中，但是在 GD25LX256E 中是 configuration register 中，所以这儿先修改名字，以避免误会，之后的 FlsLUT 表也要相应修改名字
- Second LUT index= /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/WRCR // WRCR2 // Lut 中第二个命令序列的索引，仅用于 RMW 类型，这是写入命令。
- Write Enable LUT Index= /Fls/Fls/FlsConfigSet/FlsExternalDr/MemCfg_DOPI/WriteEnable //写入启用命令的索引，如果需要，在写入命令之前。仅用于写入和 RMW 操作。

  //MX25UW51245G 的配置寄存器 2 定义为：

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 //地址，如果在命令中使用
- Register size (1 -> 4) =1 //配置寄存器的大小(以字节为单位)。
- Bit-field offset (0 -> 32) =1, Bit-field width (0 -> 32)=1, Bit-field value (0 -> 4294967295)=1

  //GD25LX256E 定义为：

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

所以应该配置为：

- Command address (0 -> 4294967295) =0 //地址，如果在命令中使用
- Register size (1 -> 4) =1 //配置寄存器的大小(以字节为单位)。
- 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 //操作类型可以是以下类型之一：QSPI_IP_OP_TYPE_QSPI_CFG – 重新配置 QSPI 控制器
- Controller configuration = /Fls/Fls/FlsConfigSet/FlsExternalDr/ControllerCfg_DDR_DQS_External //对将用于初始化控制器的配置的引用。仅对 QSPI_IP_OP_TYPE_QSPI_CFG 操作有效。在初始化 QSPI NOR 为 DOPI 模式后，重新配置控制为 DDR DQS external 模式。

5.3.3 FlsLUT 配置页

用于配置包含所有指令/操作数序列的查找表的配置页。一个序列由一系列最多 8 个指令/操作数对组成，最多可存储 4 个 LUT，每当向外部 Flash 触发命令时，就会执行这些 LUT。注意，此项为修改一款新 Flash 最主要需要修改的地方，需要对照 QSPI NOR 的数据手册修改。

因为之前在第 2/4 章，已经分析过：

- Read_dopi：

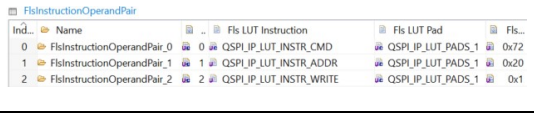
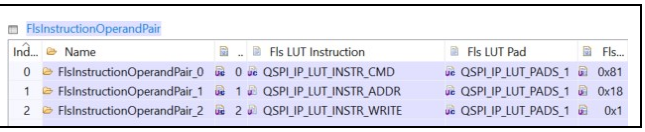
	MX25U51245G(参考)	GD25LX256E(设计)
EB 配置	FlsInstructionOperandPair Ind... Name .. Fls LUT Instruction Fls LUT Pad Fls... 0 FlsInstructionOperandPair_0 0 QSPI_IP_LUT_INSTR_CMD_DDR QSPI_IP_LUT_PADS_8 0xee 1 FlsInstructionOperandPair_1 1 QSPI_IP_LUT_INSTR_CMD_DDR QSPI_IP_LUT_PADS_8 0x11 2 FlsInstructionOperandPair_2 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：配置不变。

	MX25U51245G(参考)	GD25LX256E(设计)
EB 配置	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

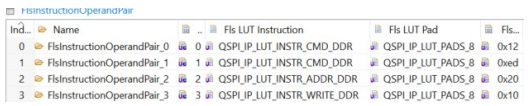
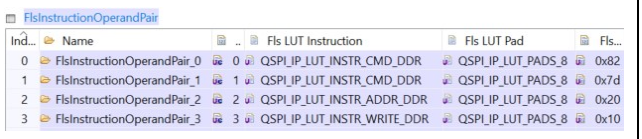
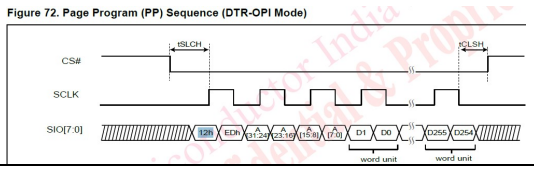
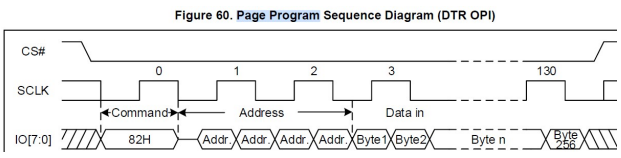
- WRCR2 修改为 WRCR 的序列：

	MX25U51245G(参考)	GD25LX256E(设计)

EB 配置		

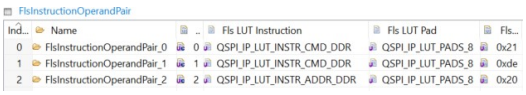
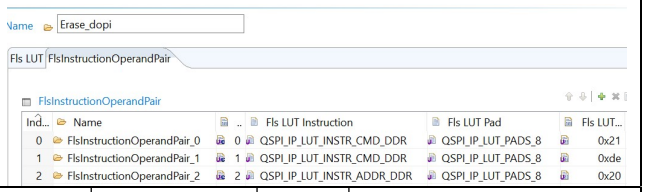
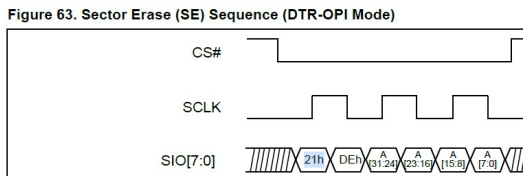
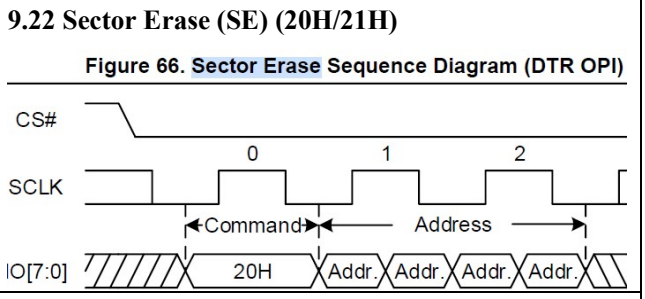
所以本节只分析剩余项：

5.3.3.1 Write_dopi

	MX25U51245G(参考)				GD25LX256E(设计)			
EB 配置								
具体分析		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 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)
	3f10	0xF(WRITE_DDR)	0x3(8 bit)	0x10(write 16 Bytes on 4 pad)	3f10	0xF(WRITE_DDR)	0x3(8 bit)	0x10(write 16 Bytes on 4 pad)
时序图								
说明	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.			

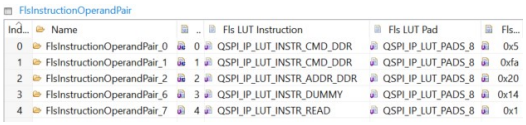
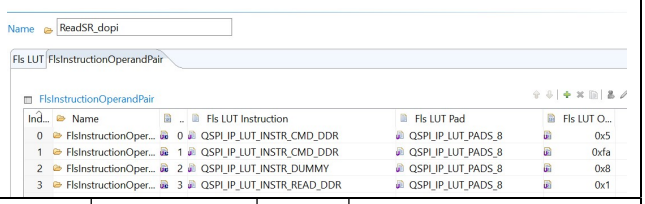
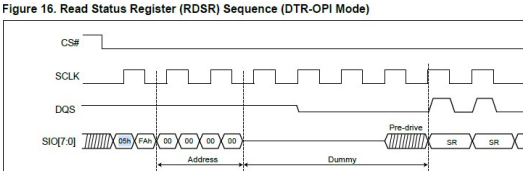
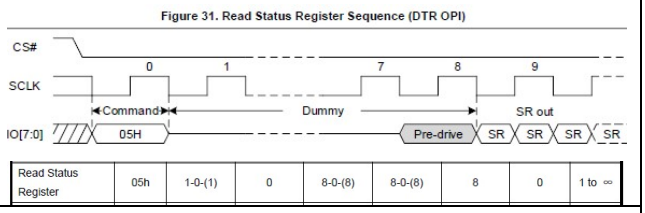
注意原生 MX25U51245G 使用 Page Program (PP/PP3B/PP4B)命令 02h/12h，这个与 GD25LX256E 相同，此处修改为：Octal Page Program (82H/84H)，所以使用旧命令字也可以，此处可以修改也可以不修改。

5.3.3.2 Erase_dopi:相同，不用修改

<table>
<tr><td></td><td colspan="4">MX25U51245G(参考)</td><td colspan="4">GD25LX256E(设计)</td></tr>
<tr><td>EB 配置</td><td colspan="4"> 
</td><td colspan="4"> 
</td></tr>
<tr><td rowspan="4">具体分析</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td></tr>
<tr><td>4721</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td rowspan="2">0x21
0xde(Sector Erase SE4B)</td><td>4721</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>0x21 (Sector Erase DTR OPI)</td></tr>
<tr><td>47de</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>47de</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>0xde(0x21 的补码)</td></tr>
<tr><td>2b20</td><td>0xA(ADDR_DDR)</td><td>0x3(8 bit)</td><td>0x20(32 Addr bits to be sent on 4 pad)</td><td>2b20</td><td>0xA(ADDR_DDR)</td><td>0x3(8 bit)</td><td>0x20(32 Addr bits to be sent on 4 pad)</td></tr>
<tr><td>时序图</td><td colspan="4"> 
</td><td colspan="4"> 
</td></tr>
<tr><td>说明</td><td colspan="4">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).</td><td colspan="4">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.</td></tr>
</table>

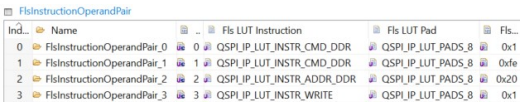
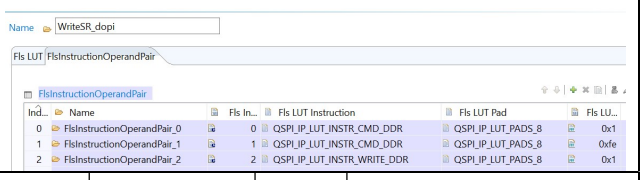
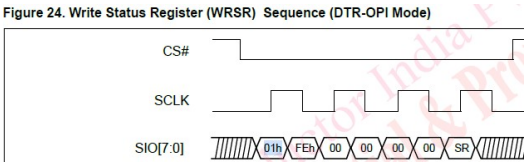
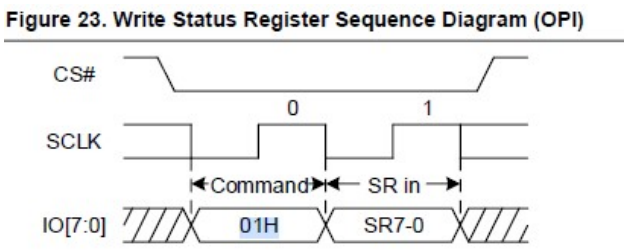
5.3.3.3 ReadSR_dopi

	MX25U51245G(参考)	GD25LX256E(设计)

<table>
<tr><td>EB 配置</td><td colspan="4">  </td><td colspan="4">  </td></tr>
<tr><td rowspan="6">具体分析</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td></tr>
<tr><td>4705</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td rowspan="2">0x05 0xfa(RDSR DTR-OPI mode)</td><td>4705</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>0x05 (RDSR DTR-OPI)</td></tr>
<tr><td>47fa</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>4702</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>0xfa(0x05 的补码)</td></tr>
<tr><td>2b20</td><td>0xA(ADDR_DDR)</td><td>0x3(8 bit)</td><td>0x20(32 Addr bits to be sent on 4 pad)</td><td></td><td></td><td></td><td></td></tr>
<tr><td>0f14</td><td>0x3(DUMMY)</td><td>0x3(8 bit)</td><td>0x14(20 dummy cycles)</td><td>0f08</td><td>0x3(DUMMY)</td><td>0x3(8 bit)</td><td>0x08(8 dummy cycles)</td></tr>
<tr><td>1f01</td><td>0x7(READ)</td><td>0x3(8 bit)</td><td>0x1 (Read 1 Bytes on 4 pad)</td><td>2b01</td><td>0xE(READ_DDR)</td><td>0x3(8 bit)</td><td>0x1(Read 1Byte on 8 pad)
因为是 8bit 模式修改为 READ_DDR</td></tr>
<tr><td>时序图</td><td colspan="4">  </td><td colspan="4">  </td></tr>
<tr><td>说明</td><td colspan="4">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</td><td colspan="4">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.</td></tr>
</table>

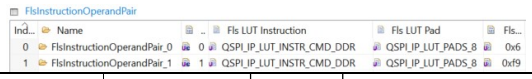
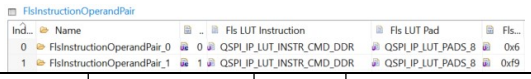
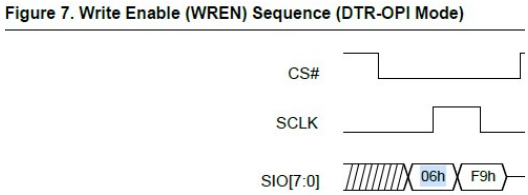
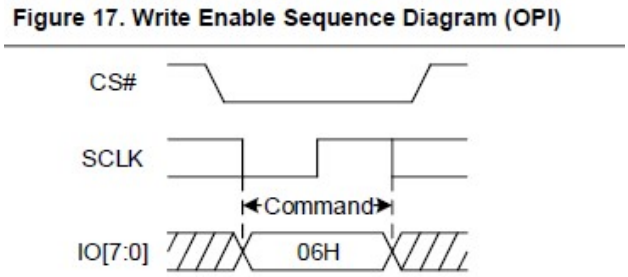
5.3.3.4 WriteSR_dopi

	MX25U51245G(参考)	GD25LX256E(设计)

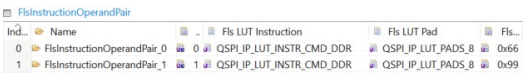
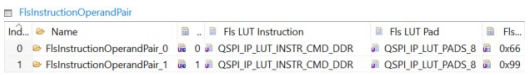
<table>
<tr><td>EB 配置</td><td colspan="4">  </td><td colspan="4">  </td></tr>
<tr><td rowspan="5">具体分析</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td></tr>
<tr><td>4701</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td rowspan="2">0x01 0xfe(WRSR DTR-OPI Mode)</td><td>4701</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>0x01 (WRSR OPI)</td></tr>
<tr><td>47fe</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>47fe</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>0xfe(0x01 的补码)</td></tr>
<tr><td>2b20</td><td>0xA(ADDR_DDR)</td><td>0x3(8 bit)</td><td>0x20(32 Addr bits to be sent on 4 pad)</td><td></td><td></td><td></td><td></td></tr>
<tr><td>2301</td><td>0x08(WRITE)</td><td>0x3(8 bit)</td><td>0x01(write 1 Bytes on 4 pad)</td><td>3F01</td><td>0xF(WRITE_DDR)</td><td>0x3(8 bit)</td><td>0x01(write 1 Bytes on 4 pad)
因为是 8bit 模式修改为 WRITE_DDR</td></tr>
<tr><td>时序图</td><td colspan="4">  </td><td colspan="4">  </td></tr>
<tr><td>说明</td><td colspan="4">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</td><td colspan="4">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).</td></tr>
</table>

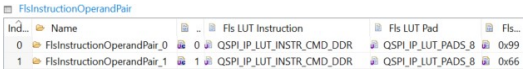
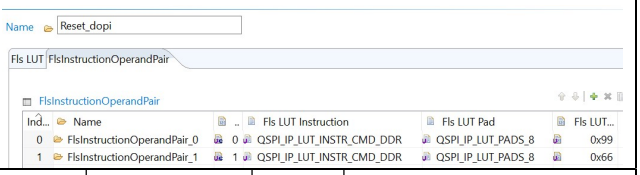
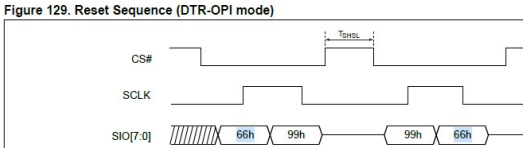
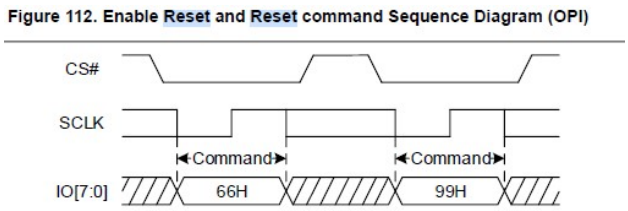
5.3.3.5 WriteEnable_dopi:相同，不用修改

	MX25U51245G(参考)	GD25LX256E(设计)

<table>
<tr><td>EB 配置</td><td colspan="4">  </td><td colspan="4">  </td></tr>
<tr><td rowspan="3">具体分析</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td></tr>
<tr><td>4706</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td rowspan="2">0x06
0xf9(WREN DTR-OPI Mode)</td><td>4706</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>0x06 (WREN OPI Mode)</td></tr>
<tr><td>47f9</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>47f9</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>0xf9(0x06 的补码)</td></tr>
<tr><td>时序图</td><td colspan="4">  </td><td colspan="4">  </td></tr>
<tr><td>说明</td><td colspan="4">The Write Enable (WREN) instruction is for setting Write Enable Latch (WEL) bit. For those instructions like PP/ PP3B/PP4B, SE/SE3B/SE4B, 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).</td><td colspan="4">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.</td></tr>
</table>

5.3.3.6 ResetEnable_dopi/Reset_dopi:相同，不用修改

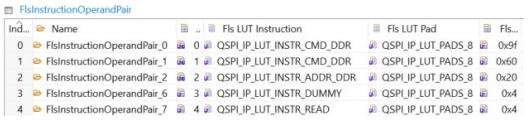
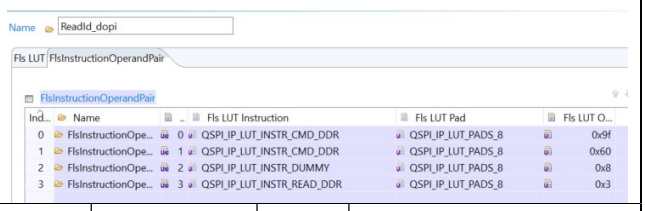
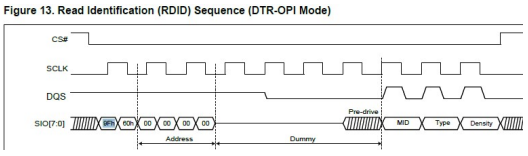
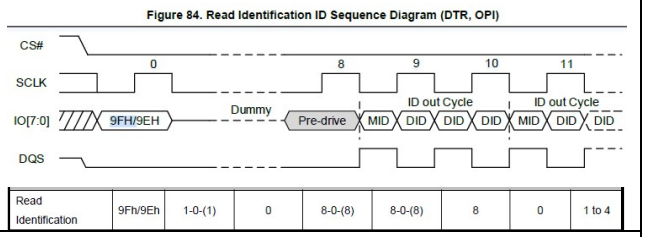
<table>
<tr><td></td><td colspan="4">MX25U51245G(参考)</td><td colspan="4">GD25LX256E(设计)</td></tr>
<tr><td>ResetEnable_dopi EB 配置</td><td colspan="4">  </td><td colspan="4">  </td></tr>
<tr><td rowspan="2">ResetEnable_dopi 具体分析</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td></tr>
<tr><td>4766</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>0x66
0x99(RSTEN</td><td>4766</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>0x66 (RSTEN)</td></tr>
</table>

<table>
<tr><td></td><td>4799</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>DTR-OPI mode)</td><td>4799</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>0x99(0x66 的补码)</td></tr>
<tr><td>Reset_dopi
EB 配置</td><td colspan="4">  </td><td colspan="4">  </td></tr>
<tr><td rowspan="3">Reset_dopi
具体分析</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td></tr>
<tr><td>4799</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td rowspan="2">0x99
0x66(RST DTR-OPI mode)</td><td>4799</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>0x99 (RST)</td></tr>
<tr><td>4766</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>4766</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>0x66(0x99 的补码)</td></tr>
<tr><td>时序图</td><td colspan="4">  </td><td colspan="4">  </td></tr>
<tr><td>说明</td><td colspan="4">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.

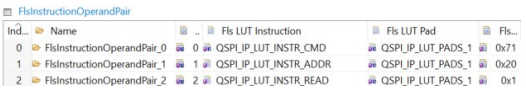
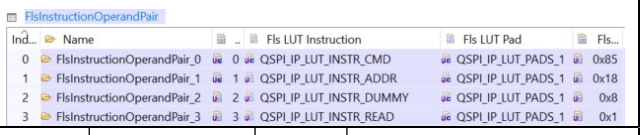
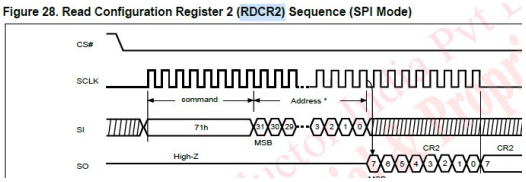
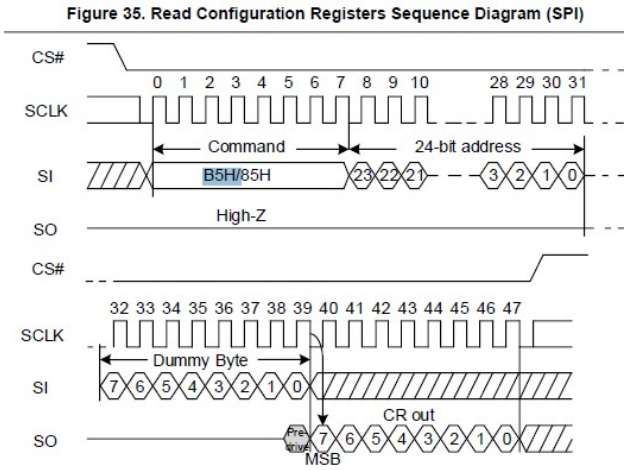
The reset time is different depending on the last operation</td><td colspan="4">If the Reset command is accepted, any on-going internal operation will be terminated and the device will return to its default power-on state and lose all the current volatile settings, such as Volatile Status Register bits, 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

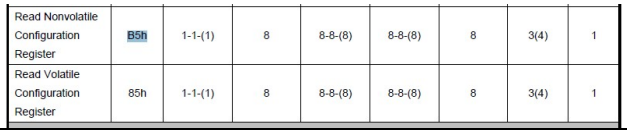
The “Enable Reset (66H)” and the “Reset (99H)” commands can be issued in either SPI or OPI mode.</td></tr>
</table>

5.3.3.7 ReadId_dopi

<table>
<tr><td></td><td colspan="4">MX25U51245G(参考)</td><td colspan="4">GD25LX256E(设计)</td></tr>
<tr><td>EB 配置</td><td colspan="4"> 
</td><td colspan="4"> 
</td></tr>
<tr><td rowspan="6">具体分析</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td></tr>
<tr><td>479f</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td rowspan="2">0x9f 0x60(RDID DTR-OPI mode)</td><td>479f</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>0x9f/9e(RDID DTR-OPI)</td></tr>
<tr><td>4760</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>4760</td><td>0x11(CMD_DDR)</td><td>0x3(8 bit)</td><td>0x60/61(0x9f/9e 的补码)</td></tr>
<tr><td>2b20</td><td>0xA(ADDR_DDR)</td><td>0x3(8 bit)</td><td>0x20(32 Addr bits to be sent on 4 pad)</td><td></td><td></td><td></td><td></td></tr>
<tr><td>0f04</td><td>0x3(DUMMY)</td><td>0x3(8 bit)</td><td>0x04(4 dummy cycles)</td><td>0f08</td><td>0x3(DUMMY)</td><td>0x3(8 bit)</td><td>0x08(8 dummy cycles)</td></tr>
<tr><td>1f04</td><td>0x07(READ)</td><td>0x3(8 bit)</td><td>0x04(Read 4 Bytes on 4 pad)</td><td>3b04</td><td>0xE(READ_DDR)</td><td>0x3(8 bit)</td><td>0x04(Read 4 Bytes on 4 pad)
因为是 8bit 模式修改为 READ_DDR</td></tr>
<tr><td>时序图</td><td colspan="4"> 
</td><td colspan="4"> 
</td></tr>
<tr><td>说明</td><td colspan="4">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</td><td colspan="4">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.</td></tr>
</table>

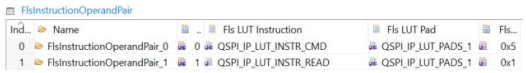
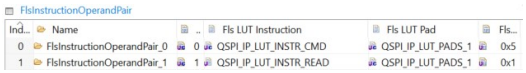
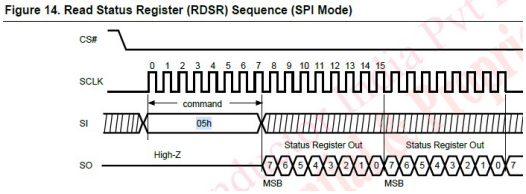
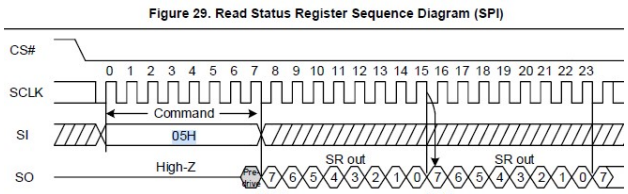
5.3.3.8 修改 RDCR2 为 RDCR

<table>
<tr><td></td><td colspan="4">MX25U51245G(参考)</td><td colspan="4">GD25LX256E(设计)</td></tr>
<tr><td>EB 配置</td><td colspan="4">RDCR2
 
</td><td colspan="4">RDCR:
 
</td></tr>
<tr><td rowspan="5">具体分析</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td></tr>
<tr><td>0471</td><td>0x01(CMD)</td><td>0x0(1 bit)</td><td>0x71(RDCR2 SPI Mode)</td><td>0485</td><td>0x01(CMD_)</td><td>0x0(1 bit)</td><td>0xb5/85 (Read Nonvolatile/Volatile Configuration Register)
注意，读取 Volatile 寄存器</td></tr>
<tr><td>0820</td><td>0x02(ADDR)</td><td>0x0(8 bit)</td><td>0x20(32 Addr bits to be sent on 4 pad)</td><td>0818</td><td>0x02(ADDR)</td><td>0x0(1 bit)</td><td>0x18(24 Addr bits to be sent on 1 pad)</td></tr>
<tr><td></td><td></td><td></td><td></td><td>0c08</td><td>0x3(DUMMY)</td><td>0x0(1 bit)</td><td>0x08(8 dummy cycles)</td></tr>
<tr><td>1c01</td><td>0x07(READ)</td><td>0x0(1 bit)</td><td>0x01(Read 1 Bytes on 1 pad)</td><td>1c01</td><td>0x07(READ)</td><td>0x0(1 bit)</td><td>0x01(Read 1 Bytes on 1 pad)</td></tr>
<tr><td>时序图</td><td colspan="4"> 
</td><td colspan="4"> 

 
</td></tr>
</table>

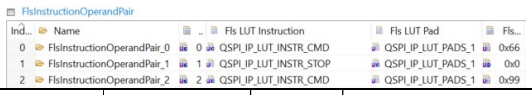
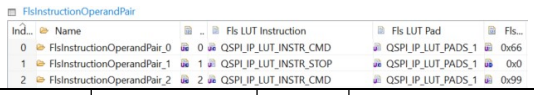
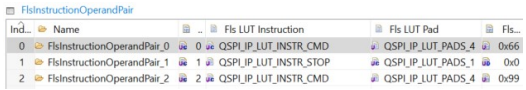
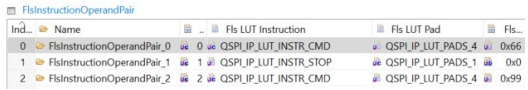
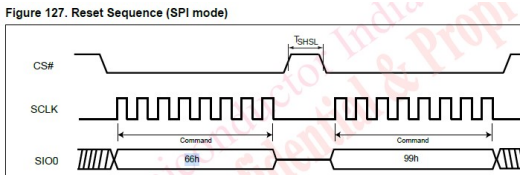
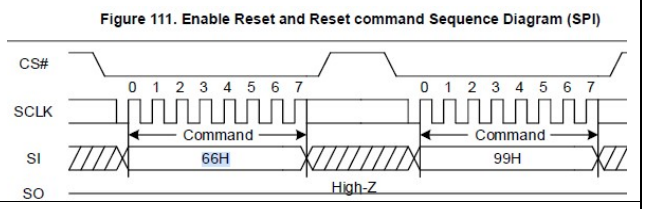
说明	The RDCR2 instruction is for reading Configuration Register 2.	The Read Nonvolatile/Volatile Configuration Register command is for reading the Nonvolatile/Volatile Configuration Registers. It is followed by a 3-Byte address (A23-A0) or a 4-Byte address (A31-A0) and a dummy Byte, and each bit is latched-in 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.

5.3.3.9 ReadSR: 相同，不用修改

	MX25U51245G(参考)				GD25LX256E(设计)			
EB 配置								
具体分析		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)
时序图								
说明	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: 相同，不用修改

	MX25U51245G(参考)	GD25LX256E(设计)

<table>
<tr><td>InitRest
EB 配置</td><td colspan="4">  </td><td colspan="4">  </td></tr>
<tr><td>InitRest
具体分析</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td></tr>
<tr><td></td><td>0466</td><td>0x01(CMD)</td><td>0x0(1 bit)</td><td>0x66 (RSTEN SPI mode)</td><td>0466</td><td>0x01(CMD)</td><td>0x0(1 bit)</td><td>0x66 (RSTEN SPI mode)</td></tr>
<tr><td></td><td>0000</td><td>0x0(STOP)</td><td>0x0(1 bit)</td><td>0x0</td><td>0000</td><td>0x0(STOP)</td><td>0x0(1 bit)</td><td>0x0</td></tr>
<tr><td></td><td>0499</td><td>0x01(CMD)</td><td>0x0(1 bit)</td><td>0x99 (RST SPI mode)</td><td>0499</td><td>0x01(CMD)</td><td>0x0(1 bit)</td><td>0x99 (RST SPI mode)</td></tr>
<tr><td>RuntimeReset
EB 配置</td><td colspan="4">  </td><td colspan="4">  </td></tr>
<tr><td>RuntimeReset
具体分析</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td><td></td><td>Instr(6bits)</td><td>Pads(2bits)</td><td>Operand(8bits)</td></tr>
<tr><td></td><td>0666</td><td>0x01(CMD)</td><td>0x2(4 bit)</td><td>0x66 (RSTEN SPI mode)</td><td>0666</td><td>0x01(CMD)</td><td>0x1(1 bit)</td><td>0x66 (RSTEN SPI mode)</td></tr>
<tr><td></td><td>0000</td><td>0x0(STOP)</td><td>0x0(1 bit)</td><td>0x0</td><td>0000</td><td>0x0(STOP)</td><td>0x0(1 bit)</td><td>0x0</td></tr>
<tr><td></td><td>0699</td><td>0x01(CMD)</td><td>0x2(4 bit)</td><td>0x99 (RST SPI mode)</td><td>0699</td><td>0x01(CMD)</td><td>0x1(1 bit)</td><td>0x99 (RST SPI mode)</td></tr>
<tr><td>时序图</td><td colspan="4">  </td><td colspan="4">  </td></tr>
<tr><td>说明</td><td colspan="4">同 ResetEnable_dopi/Reset_dopi</td><td colspan="4">同 ResetEnable_dopi/Reset_dopi</td></tr>
</table>

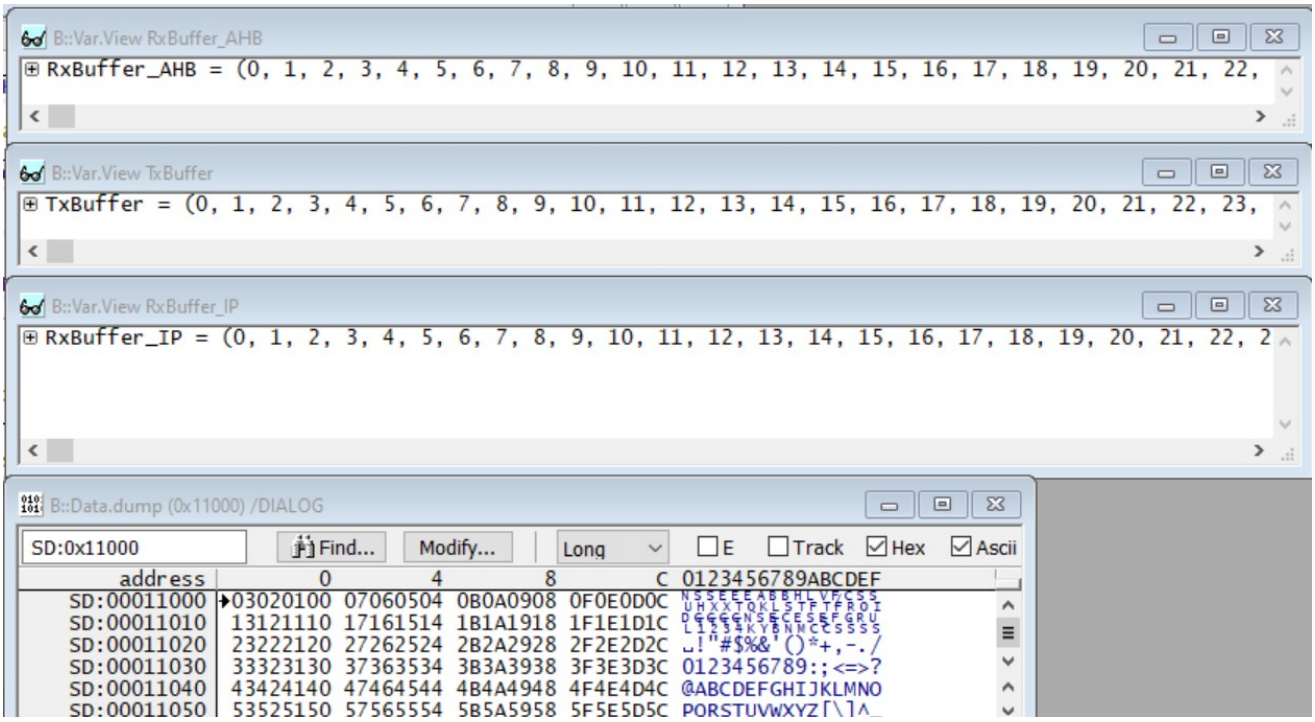
5.4 测试结果

使用 Lauterbach 加载脚本：C:\NXP\SW32G_RTD_4.4_4.0.2\eclipse\plugins\Fls_TS_T40D11M40I2R0\examples\EBT\S32G3\Fls_Example_S32G399A_M7\debug\run.cmm，停止在 main 函数入口外。使用 Lauterbach 检查 Fls_Init，Fls_Write，Fls_Read，的执行结果，在：

/* 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。
```

以及 Fls_GetAhbData 的执行结果。然后检查变量和地址(#define PHYSICAL_START_ADDR 0x11000U /* The HW start address corresponding to the logical address 0 */)：



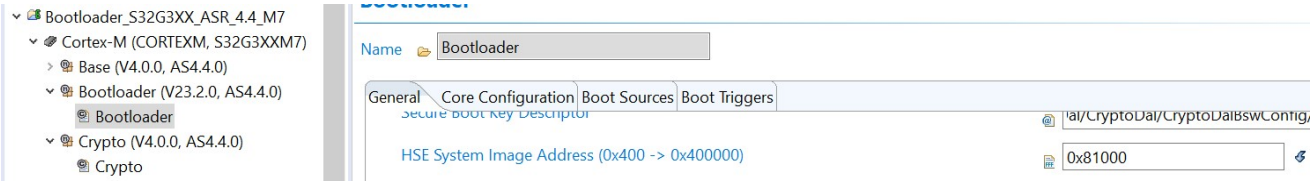
表明写，IP 读与 AHB 读数据均一致，证明驱动工作正常。

6 开发 Bootloader 工程中 Fls 驱动

6.1 Bootloader 工程说明

以版本 Integration_Reference_Examples_S32G3_2023_02 为例，根据文档《S32G_Bootloader_V*.pdf》创建和修改 Bootloader 工程，注意：

- 因为 Bootloader 使用 QSPI NOR DMA 驱动来搬运启动模块，而使用 IP 驱动来操作与 secure 相关 QSPI NOR 操作，所以我们选择去掉 XRDC 和 eMMC，但是保留 secure boot 功能。
- 关掉软件调试点。
- 去掉 eMMC 后，在 Bootloader 的 Boot Sources 中，将相关 eMMC boot Sources 删除掉
- 生成代码前将 C:\EB\tresos\workspace\Bootloader_S32G3XX_ASR_4.4_M7\output\下的 include, output, src 先删除再生成，以防止之前的生成留下的旧文件。
- 编译时遇到的 GPT 的问题，《S32G_Bootloader_V*.pdf》文档有说明如何 fix。
- 由于 IVT 配置 SYS-IMG 地址为 0x81000，所以相应修改：



- 编译脚本修改如下：

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 ::注意此版本 Bootloader 默认对应原 RTD 版本是 4.4_4.0.0
:: 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
```

- 注意 secure boot 在执行 publish sys-img 时，会烧写内部的 anti-rollback counter fuse，所以 secure boot 功能测试会减少 counter 的 fuse 资源，由于本文主要关注 Qspi nor flash 的操作，所以我们修改代码：

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 用于将 fls erase 操作圆整到 4KB
```

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

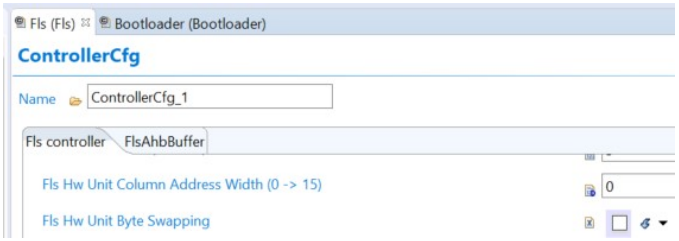
```
Bl_ConfigureSecureBoot
|-> //注释掉 sys-img publish 的操作
/*
    if (E_OK == status)
    {
        status = CryptoDal_GetSysImage(
            &Bl_HseSysImage[0], &u32SysImageOffSet, &u32SysImageSize);
    }
*/

volatile int debug_erase;
volatile int debug_write;
|  |->Bl_SaveConfiguration
uint32_t u32SysImageSizeAligned = BL_ALIGN_4096B(u32SysImageSize); //johnli change from 1024 修改代码防止 erase 数据不是 4KB 对齐。
    while(debug_erase); //此处停下代码，方便使用 Lauterbach 来检查 Fls IP 驱动的操作情况
|  |  |->Fls_Erase(u32SysImageStorageAddr + u32SysImageOffset,
        u32SysImageSizeAligned);
while(debug_write);
|  |  |->Fls_Write(u32SysImageStorageAddr + u32SysImageOffset,
        (const uint8 *) pSysImage, u32SysImageSizeAligned);
```

- Bootloader 工程的 Debug 方法请参考文档《S32G_Bootloader_V*.pdf》。
- Bootloader 工程相应的修改细节参考随文档发布的工程源文件 。

6.2 Bootloader 与 MCAL Fls 驱动的不同点

MCAL Fls 驱动中对 QSPI NOR 配置方面的修改，Bootloader 仍需要相应修改，此外，两者还有以下的不同之外：

	Bootloader Fls	MCAL Fls	说明
	MemCfg_0 配置页		
1	initResetSettings 关闭	initResetSettings 打开	由于 Bootloader 是

2	initConfiguration 关闭 FlsLUT 相同，将 MCAL FlsLUT 的配置移植过来	initConfiguration 打开 FlsLUT 相同	使用 IVT QSPI NOR 参数来初始化 QSPI NOR 的，所以驱动中不再考虑做 InitReset 及 Init reconfiguration。FlsLUT 中的 FastRead/Write 没有引用
	FlsController 配置页		
3	ControllerCfg_1	ControllerCfg_SDR=0	MCAL 初始化为 SDR 模式，然后在 initConfiguration 中，初始化 QSPI NOR 后，再切换为 DDR 模式，Bootloader 不需要，直接初始化为 DDR 模式
4	ControllerCfg_1. Fls Hw Unit Byte Swapping=unchecked 	ControllerCfg_1. Fls Hw Unit Byte Swapping=unchecked	注意 Bootloader 时，IVT 会初始化为 swapping=0，而默认 Bootloader 中是配置为 Macronix=1 的，所以 Bootloader 拷贝后，运行到 Fls_init 后，则之后对 AHB 地址的访问会导致数据颠倒，从而失败，所以这儿必须要修改，注意。
	ControllerCfg 配置页		
5	Cfg_0 设置为 DDR Loopback， autoupdate 模式,但此 Cfg 没有引用	Cfg_0 设置为 SDR Loopback， bypass 模	Cfg_1 为 DDR, external DQS,

		式,此 Cfg 为初始化时的模式	autoupdate 模式 Cfg_1 相同，Bootloader 中初始化为此模式，MCAL Fls 在 initConfiguration 中重新配置为此模式
	General 配置页		
6			由于 Bootloader 和 MCAL 时钟配置不同，所以相应 Fls 驱动中时钟相关配置不同
	FlsSector 配置页		
7			Bootloader 中 Sector 配置，一般和其要储存的如 Bootloader 镜像，M7 镜像及 A53 ATF 镜像是地址 align 的，而 Mcal 中为一示例

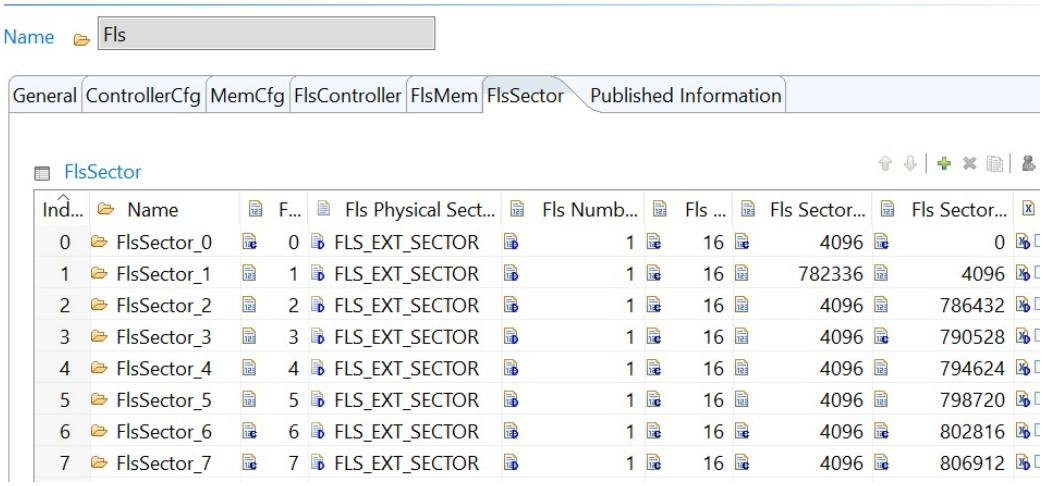
6.3 镜像打包

关于镜像打包和烧写的情况请参考文档《S32G_Bootloader_V*.pdf》，其中注意：

- IVT 头需要之前第 4 章开发出来的版本。
- SRAM 初始化 DCD 脚本选择 G3 的版本：
  C:\NXP\Integration_Reference_Examples_S32G3_2023_02\code\framework\realtime\swc\bootloader\platforms\S32G3XX\res\flash\ S32G3XX_DCD_InitSRAM.bin。
- 另外打开 IVT 工具要预告选择一个创建好的 G3 的工程，这样才会使用 G3 20MB 的 SRAM 内存越界检查。
- 由于我们选择了使用 secure boot，所以需要增加 HSE 镜像：
  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

- 然后还需要参考 Flash 驱动 sector 的配置来，调节 IVT 镜像分布，避免重叠：

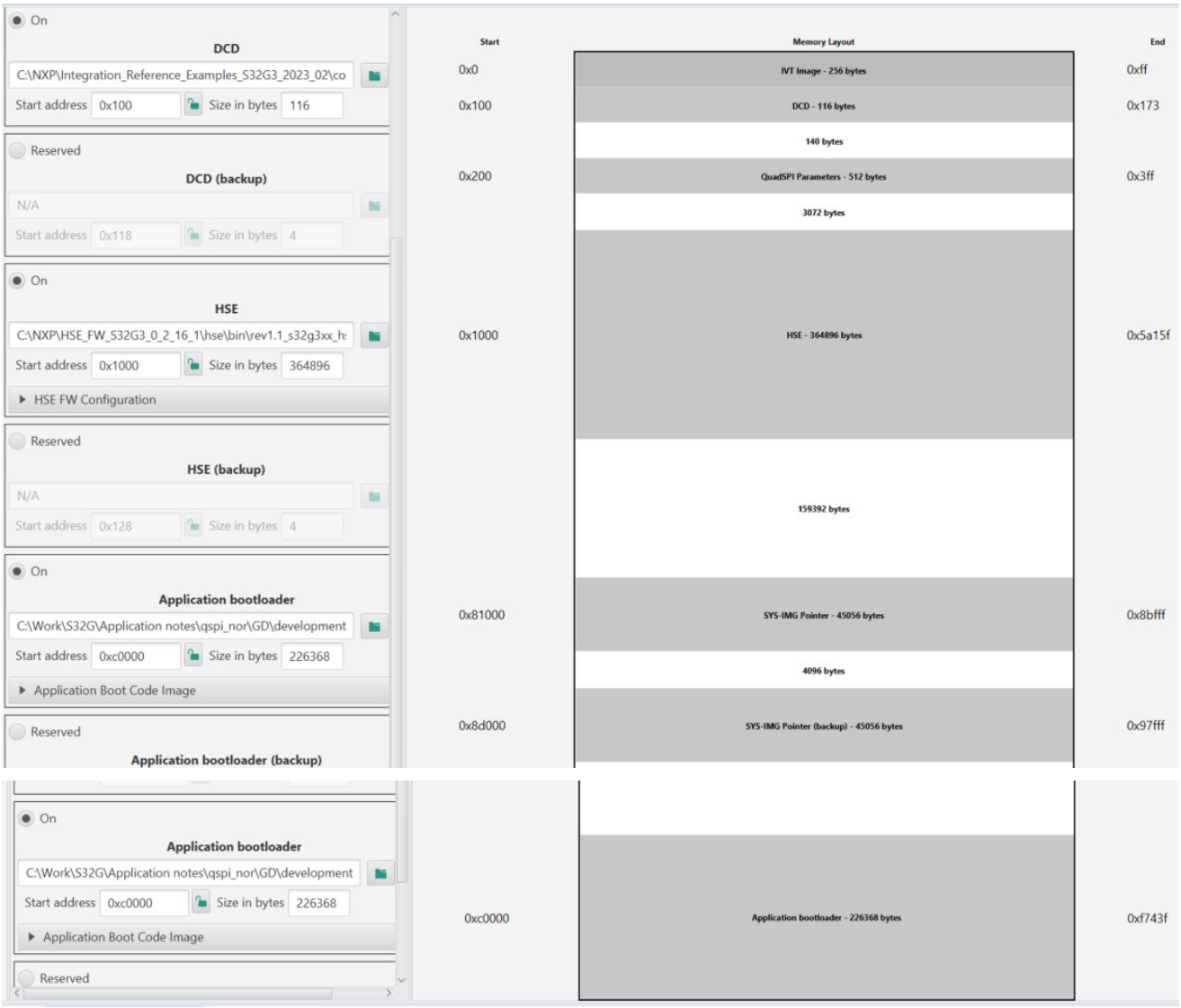
  Sector 配置为：



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

所以：

1. 使用 0x0 开头，大小为 0x1000=4096 来储存 DCD(偏移 0x100)，QSPI NOR 头(偏移 0x200)。
2. 使用 0x1000 开头，大小为 0xBF000=782336 来储存 HSE FW 及 SYS-IMG。
3. 使用 0xc000 开头的部分来储存 bootloader 镜像。



可以看出需要写操作的镜像都是对齐到 4KB 的 Sector size 的。

6.4 测试结果

将打包好的 Bootloader 镜像使用 Flash tool 烧写到 QSPI NOR flash 中，切换启动模式为 QSPI NOR flash 后，再次上电启动：

使用Lauterbach运行脚本：C:\NXP\Integration_Reference_Examples_S32G3_2023_02\code\framework\realtime\swc\bootloader\platforms\S32G3XX\build\cmm\connect_s32g3xx_m7.cmm，则Bootloader 代码会停留在最开始的地方，这个时候运行脚本连接调试器，即可以调试：

```
    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);
```

当代码运行到 Bl_SaveConfiguration 中的 while(debug_erase);处，将 debug_erase 改成 0 后，就可以继续运行，检查函数 Fls_Erase 的执行情况，同理检查 Fls_Write 的执行情况。

7 开发 Linux 驱动(可选)

参考文档《S32G_QSPINOR_定制_*.pdf》，第 9 章：软件定制 Linux Kernel 了解 Linux 驱动的定制方法，其中已经有 Micron MT35XU256ABA 的示例，参考如下添加 GD GD25LX256E。以 BSP37 为例。

ATF 与 Uboot 的 QSPI NOR 驱动与内核类似，本文不再详述。

7.1 Linux GD 驱动支持情况

\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，目前还比较原始，由于 Micron 和 GD flash 有兼容的情况，所以可以利用 Micron 的源代码，则可以参考《S32G_QSPINOR_定制_*.pdf》中的 Micron MT35XU256ABA 示例，注意 BSP37 的代码中已经支持 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 时钟相关的修改

Linux 驱动架构设计为：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];
```

所以是先从外部 QSPI NOR flash 中读出 ID 值，然后使用 ID 值去匹配 QSPI NOR 相关信息的数据结构数组。

而读 ID 的 JEDEC 时钟要求不高，如下：MACRONIX MX25UW51245G flash 说明为：

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 说明为：

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

而目前 S32G Linux BSP 初始化的 QSPI NOR 时钟为 200Mhz，所以是存在读 ID 不正确的风险的，(实测使用 200Mhz 读 GD25LX256E 的 ID 值不正确)。由于目前 Linux SPI NOR 驱动架构中没有读 ID 后没有升频的 API，所以考虑直接在 Linux 中降频使用，如下 Linux 中的 QSPI NOR 时钟树：

```
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,而除 3=266Mhz。

所以在不修改根时钟的情况下，将频率由原来的 200Mhz 降为 133Mhz 使用，修改为：

7.2.1 ATF 中修改

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>; 此 DTS 值用于 uboot 驱动设置
```

所以 Uboot 中无修改。

7.2.2 Linux DTS 的修改

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>; \\主要是此处修改，此值用于 Linux 驱动时钟设置，这样时钟就修改为 133Mhz 了。
```

7.3 在 DTS 中增加 GD flash 的支持

\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 模式
            spi-rx-bus-width = <8>;
            reg = <0>;
            force-soft-reset;
            inverted-cmd-ext; //DDR 模式命令第二字节用补码方式
            memory-default-octal-dtr; //支持 8bit DDR 模式
```

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

```
//&mcronix_memory {
```

```
&gigadevice_memory {
```

然后修改掉编译错误：

```
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 修改源代码增加 flash 信息结构体

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

```
static const struct flash_info micron_parts[] = {
{ "mt35xu512aba"…,
    //增加一个 gd25lx256e，注意名字要和 DTS 里对应。
```

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 参考前文，每个 sector 是 4K，一共 8192 个 sector

// gd25lx256e 不需要操作 flag register,所以去掉 USE_FSR，OCTAL_DTR 的读写操作在 fixup 中重置，所以不需要配置 SPI_NOR_OCTAL_DTR_READ 和 SPI_NOR_OCTAL_DTR_PP

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 修改源代码中 flash 的 fixup 支持 DTR 模式

Driver/mtd/spi-nor/Macronix.c 中的

```
static struct spi_nor_fixups mx25uw51245g_fixups = {
    .default_init = mx25uw51245g_default_init, //用于配置 QSPI 进入 DTR 模式
    .post_bfpt = mx25uw51245g_post_bfpt_fixup, //用于配置写操作的 DTR 模式
    .post_sfdp = mx25uw51245g_post_sfdp_fixup,//用于配置写和读操作的 DTR 模式
};
```

所以参考 mx25uw51245g 的 fixup 配置，修改：

```
static struct spi_nor_fixups mt35xu512aba_fixups = {
    .default_init = mt35xu512aba_default_init,
    .post_sfdp = mt35xu512aba_post_sfdp_fixup, // bfpt 是重复操作，不需要
};
static int spi_nor_micron_octal_dtr_enable(struct spi_nor *nor, bool enable)
{
#if 0
//保持 configure 寄存器中 16 dummy cycles 的配置，不需要修改
    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 //验证代码，不需要
        /* Read flash ID to make sure the switch was successful. */
    …
    #endif
        return 0;
    }

    static void mt35xu512aba_post_sfdp_fixup(struct spi_nor *nor)
    {
      //设置写操作为 octal dtr 模式
        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. */ //设置读操作主 cotal dtr 模式
    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;
}
```

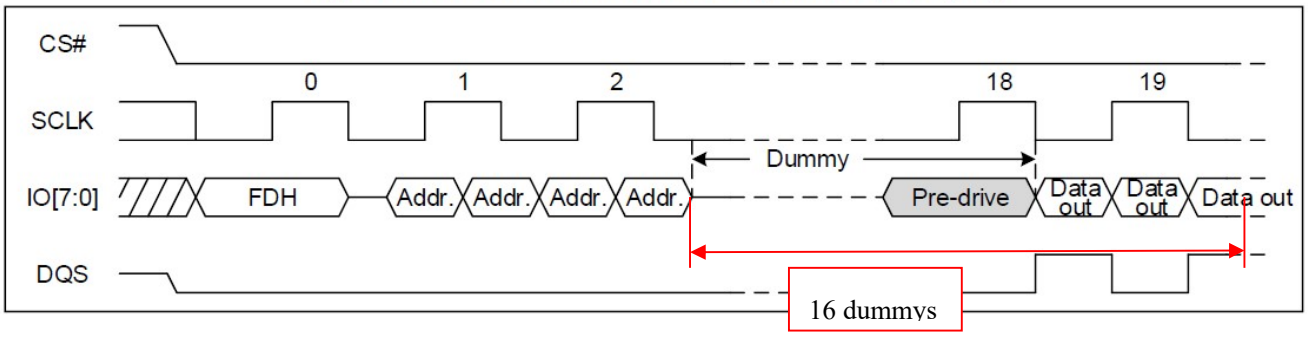
7.6 Turning dummy 值解决读错位的问题

函数 mt35xu512aba_post_sfdp_fixup 中 DTR 读的 dummy 配置为 16，与 QSPI NOR 中配置寄存器的默认配置相同，测试下来：

```
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
```

所以数据向后偏移了 2X15 个字节，按照 8bit DTR 模式计算，一个 clock 传输两字节，所以 dummy 值应该向前提前 15 个 clock。

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



所以应该设置为 1，修改：

```
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);
```

测试正确：

```
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
```

此处 controller 与 QSPI NOR flash 中 dummy 设置不匹配才能工作的原因需要 QSPI NOR flash 厂家说明。

7.7 测试结果

将fsl-image-auto-s32g399ardb3.sdcard使用Linux Host PC DD到TFcard上，替换掉ATF，Image，DTB后插入RDB3板子，启动模式设置为SDcard启动，USDHC拨码设置为SD，连接好串口与电源，上电从eMMC上启动linux，进入shell：

- 启动信息：

```
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
```

- 设备文件：

```
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
```

- 时钟

```
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 设备要先擦除，后使用，因为只能从1 写成0，擦除后可以读出来看全是0xff;

```
root@s32g399ardb3:~# mtd_debug erase /dev/mtd0 0 0x10000 //擦除操作以4K字节 sector size为地址对齐和数据对齐Erased 65536 bytes from address 0x00000000 in flash
root@s32g399ardb3:~# hexdump -v -n 0x100 /dev/mtd0 //读写操作以256字节 page size为地址对齐和数据对齐

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

- 写入数据，多读几次来互相对比，及和原文件对比，如果一致，则驱动工作正确。

```
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
```