Flash XIP Parameter Setting User Guide

Calterah Semiconductor

Date	Version	Description	Author
2020.06.04	V0.1.0	First version	Xudong Ran
2020.07.29	V0.2.0	Add Static Parameter Example	Xudong Ran

Table of Content

CHAPTER

ONE

INSTRUCTIONS

The purpose of this document is to help customer modify flash command parameters in radar firmware to implement XIP operation on different flash vendors. You are required modify these parameters only when using a flash chip other than Cypress S25FL128SAGNFI000. Please note that this document only applies to Alps and Rhine series.

Calterah provide QSPI flash command parameter entry for different vendors in **radar-sensor-firmware/***embarc_osp/device/peripheral/nor_flash/vendor/.* Please follow the instructions below to modify flash parameters that conform to your flash datasheet of your flash vendor.

Vendor.h

1. Find the **command to enable Quad SPI operation** on flash datasheet.
2. Find the number of **Mode Cycle** and **Dummy Cycle** required in **Quad I/O Fast Read (0xEB)** operation.
3. Based on 1, modify the following **FLASH_DEV_CMD** to enable Quad SPI operation in *embarc_osp/device/peripheral/nor_flash/vendor/[vendor]/vendor.h*

```
   #define FLASH_DEV_CMD0      DEF_FLASH_CMD(CMD_WRITE_ENABLE, 0, 32, 0
, 0, 0, 0)
   #define FLASH_DEV_CMD1      DEF_FLASH_CMD(CMD_WRITE_STS1_CFG1, 0, 32
, 0xff02, 0, 0, 0)
   #define FLASH_DEV_CMD2      DEF_FLASH_CMD(0, 0, 0, 0, 0, 0, 0)
   #define FLASH_DEV_CMD3      DEF_FLASH_CMD(0, 0, 0, 0, 0, 0, 0)
```

4. Based on 2, modify **FLASH_DEV_DUMMY_CYCLE** and **FLASH_DEV_MODE_CYCLE** according to datasheet.

```
    #define FLASH_DEV_DUMMY_CYCLE    (4)
    #define FLASH_DEV_MODE_CYCLE     (2)
```

[Vendor].c

1. Go to *embarc_osp/device/peripheral/nor_flash/vendor/[vendor]/[vendor].c*.
2. Type in the flash parameters according to flash datasheet
   total_size: total size of flash in byte
   page_size: page size of flash in byte
   m_region: DEF_FLASH_REGION (region base address, region size, erase sector size, sector erase command).
   Note: DEF_FLASH_REGION is a flexible design for those flash chips whose sectors are organized as a hybrid combination of two different sector sizes, such as 4-KB and 64-KB

sectors. This hybrid combination of flash memory with different sector size is called region. In different region, sector size, sector count, and sector erase command are different. If flash chips with this type of flash memory mapping, users are required to specify the region addresses with specific sector size and erase command for two separate regions.

Here is the example of S25FL128S. Total size is 128Mbit. Page size is 256KB. Region 1 starts from 0x00000 to 0x1FFFF with 4KB sector size and 0x20 sector erase command. Region 2 starts from 0x20000 to 0xFFFFFF with 64KB sector size and 0xd8 sector erase command.

```
static flash_device_t giga_dev = {
    .total_size = 128000000 >> 3,
    .page_size = 256,
    .m_region = {
        DEF_FLASH_REGION(0x0, 0x20000, 0x1000, 0x20),
        DEF_FLASH_REGION(0x20000, 0xFE0000, 0x10000, 0xd8)
    },
    .ops = &giga_ops
};
```

3. By default, *vendor_quad_entry()* is empty. You can refer to *s25fl_quad_entry()* in *s25fls.c* and change the command (labelled in red below) to enable QSPI mode. Change **FLASH_WEL** command in **vendor_status_read()** function according to flash datasheet.

```
static int32_t s25fl_quad_entry(dw_ssi_t *dw_ssi)
{
    int32_t result = E_OK;
    uint32_t cpu_status = 0;
    uint8_t wdata[2] = {0x02, 0x02};


    dw_ssi_xfer_t xfer = DW_SSI_XFER_INIT(CMD_WRITE_STS1_CFG1, 0, 0,
0);
    dw_ssi_xfer_t we_xfer = DW_SSI_XFER_INIT(CMD_WRITE_ENABLE, 0, 0,
0);

    if ((NULL == dw_ssi) || (NULL != dw_ssi->ops)) {
        dw_ssi_ops_t *ssi_ops = (dw_ssi_ops_t *)dw_ssi->ops;
        cpu_status = arc_lock_save();
        result = ssi_ops->writeb(dw_ssi, &we_xfer);
        arc_unlock_restore(cpu_status);

        while (E_OK == result) {
            chip_hw_mdelay(1);
            result = s25fl_status_read(dw_ssi, FLASH_WEL);
```

```
            }

            xfer.buf = (void *)wdata;
            xfer.len = 2;
            result = ssi_ops->writeb(dw_ssi, &xfer);
            if (E_OK == result) {
                chip_hw_mdelay(1);
            }
        }

        return result;
    }
```

4. To activate the XIP settings, add macro "*FLASH_TYPE=vendor FLASH_XIP=1 LOAD_XIP_TEXT_EN=1*" while compiling. Please note that if any other macros are used during compiling, just add these three macros to the end.

```
make FLASH_TYPE=vendor FLASH_XIP=1 LOAD_XIP_TEXT_EN=1 bin
```

5. Use the following command to generate header.bin and firmware.bin

```
python post.py
obj_alpsMP_ref_design_v1_sensor/gnu_arcem6/sensor_gnu_arcem6.bin
firmware.bin xip
```

CHAPTER

TWO

EXAMPLE

Here is the example of NOR Flash GD25Q32C from GigaDevice.

On GD25Q32C, the way to enable Quad SPI is to set Status Register QE bit to 1.

Quad SPI

The GD25Q32C supports Quad SPI operation when using the "Quad Output Fast Read" (6BH), "Quad I/O Fast Read"(EBH), "Quad I/O Word Fast Read" (E7H), "Read Manufacture ID/ Device ID Quad I/O" (94H) and "Quad Page Program" (32H) commands. These commands allow data to be transferred to or from the device at four times the rate of the standard SPI. When using the Quad SPI command the SI and SO pins become bidirectional I/O pins: IO0 and IO1, and WP# and HOLD# pins become IO2 and IO3. Quad SPI commands require the non-volatile Quad Enable bit (QE) in Status Register to be set.

7.5. Write Status Register (WRSR) (01H or 31H or 11H)

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).

First you need to **send a Write Enable (0x06) command**.

7.1. Write Enable (WREN) (06H)

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) and Erase/Program Security Registers command. The Write Enable (WREN) command sequence: CS# goes low → sending the Write Enable command → CS# goes high.

And then send 0x31H command to set QE bit (S9 bit of Write Status Register) to 1.

Write Status Register-1	01H	S7-S0					
Write Status Register-2	31H	S15-S8					
Write Status Register-3	11H	S23-S16					

6. STATUS REGISTER

S23	S22	S21	S20	S19	S18	S17	S16
Reserved	DRV1	DRV0	HPF	Reserved	Reserved	Reserved	Reserved

S15	S14	S13	S12	S11	S10	S9	S8
SUS1	CMP	LB3	LB2	LB1	SUS2	QE	SRP1

S7	S6	S5	S4	S3	S2	S1	S0
SRP0	BP4	BP3	BP2	BP1	BP0	WEL	WIP

In Quad I/O Fast Read operation, the 3-Byte address (A23-0) is followed by **a Continuous Read mode Byte**, and **4-dummy clock** 4-bit per clock by IO0, IO1, IO2, and IO3. It means that the **Mode Cycle is 2**, and **Dummy Cycle is 4**.

7.11. Quad I/O Fast Read (EBH)

The Quad I/O Fast Read command is similar to the Dual I/O Fast Read command but with the capability to input the 3-Byte address (A23-0) and a "Continuous Read Mode" Byte and 4-dummy clock 4-bit per clock by IO0, IO1, IO2, IO3, each bit being latched in during the rising edge of SCLK, then the memory contents are shifted out 4-bit per clock cycle from IO0, IO1, IO2, IO3. The command sequence is shown in followed Figure13. The first Byte addressed can be at any location. The address is automatically incremented to the next higher address after each Byte of data is shifted out. The Quad Enable bit (QE) of Status Register (S9) must be set to enable for the Quad I/O Fast read command.

According to datasheet, GD25Q32C has uniform sector. Total size is **32Mbit**. Page size is **256 Bytes**. Uniform region starts **from 0x0 to 0x400000** with **4KB sector size** and **0x20 sector erase command**.

3. MEMORY ORGANIZATION

GD25Q32C

Each device has	Each block has	Each sector has	Each page has	
4M	64/32K	4K	256	Bytes
16K	256/128	16	-	pages
1024	16/8	-	-	sectors
64/128	-	-	-	blocks

Page Program	02H	A23-A16	A15-A8	A7-A0	D7-D0	Next Byte	continuous
Quad Page Program	32H	A23-A16	A15-A8	A7-A0	D7-D0(3)	Next Byte	continuous
Fast Page Program	F2H	A23-A16	A15-A8	A7-A0	D7-D0	Next Byte	continuous
Sector Erase	20H	A23-A16	A15-A8	A7-A0			
Block Erase(32K)	52H	A23-A16	A15-A8	A7-A0			
Block Erase(64K)	D8H	A23-A16	A15-A8	A7-A0			

embarc_osp/device/peripheral/nor_flash/vendor/vendor.h

1. Therefore, **vendor.h** should be modified as follows.

```
#define CMD_RD_JEDEC_WC     (8)

#define FLASH_DEV_SAMPLE_DELAY  (0)
#define FLASH_DEV_DUMMY_CYCLE   (4)
#define FLASH_DEV_MODE_CYCLE    (2)


/* Add for boot split feature: */
#define FLASH_DEV_CMD0_INS  (0x06)
#define FLASH_DEV_CMD1_INS  (0x31)
#define FLASH_DEV_CMD2_INS  (0)
#define FLASH_DEV_CMD3_INS  (0)
/* 31 --- 24 23 --- 0 */
/*  valid   *  addr * */
#define FLASH_DEV_CMD0_ADDR (0)
#define FLASH_DEV_CMD1_ADDR (0)
#define FLASH_DEV_CMD2_ADDR (0)
#define FLASH_DEV_CMD3_ADDR (0)
/* 31 --- 8 7 --- 0 */
/*  valid  * value */
#define FLASH_DEV_CMD0_VAL0 (0)
#define FLASH_DEV_CMD0_VAL1 (0)
#define FLASH_DEV_CMD0_VAL2 (0)
#define FLASH_DEV_CMD0_VAL3 (0)
#define FLASH_DEV_CMD1_VAL0 (0xff02)
#define FLASH_DEV_CMD1_VAL1 (0)
#define FLASH_DEV_CMD1_VAL2 (0)
#define FLASH_DEV_CMD1_VAL3 (0)
#define FLASH_DEV_CMD2_VAL0 (0)
#define FLASH_DEV_CMD2_VAL1 (0)
#define FLASH_DEV_CMD2_VAL2 (0)
#define FLASH_DEV_CMD2_VAL3 (0)
#define FLASH_DEV_CMD3_VAL0 (0)
#define FLASH_DEV_CMD3_VAL1 (0)
#define FLASH_DEV_CMD3_VAL2 (0)
#define FLASH_DEV_CMD3_VAL3 (0)

/* (ins, addr, delay(ms), val0-
val3(lowest byte, others as valid flag)). */
```

```
#define FLASH_DEV_CMD0      DEF_FLASH_CMD(FLASH_DEV_CMD0_INS, 0, 32, FL
ASH_DEV_CMD0_VAL0, FLASH_DEV_CMD0_VAL1, FLASH_DEV_CMD0_VAL2, FLASH_DEV_
CMD0_VAL3)
#define FLASH_DEV_CMD1      DEF_FLASH_CMD(FLASH_DEV_CMD1_INS, 0, 32, FL
ASH_DEV_CMD1_VAL0, FLASH_DEV_CMD1_VAL1, FLASH_DEV_CMD1_VAL2, FLASH_DEV_
CMD1_VAL3)
#define FLASH_DEV_CMD2      DEF_FLASH_CMD_INV()
#define FLASH_DEV_CMD3      DEF_FLASH_CMD_INV()
```

embarc_osp/device/peripheral/nor_flash/vendor/giga.c

2. Therefore, in ***giga.c***, change static flash parameter as below.

```
static flash_device_t giga_dev = {
    .total_size = 32000000 >> 3,
    .page_size = 256,
    .m_region = {
        DEF_FLASH_REGION(0x0, 0x400000, 0x1000, 0x20)
    },
    .ops = &giga_ops
};
```

3. Change command to enter Q mode in *giga_quad_entry()* as follows.

```
static int32_t giga_quad_entry(dw_ssi_t *dw_ssi)
{
    int32_t result = E_OK;
    uint32_t cpu_status = 0;
    uint8_t wdata[2] = {0x02, 0x02};


    dw_ssi_xfer_t xfer = DW_SSI_XFER_INIT(0x31, 0, 0, 0);
    dw_ssi_xfer_t we_xfer = DW_SSI_XFER_INIT(0x06, 0, 0, 0);

    if ((NULL == dw_ssi) || (NULL != dw_ssi->ops)) {
        dw_ssi_ops_t *ssi_ops = (dw_ssi_ops_t *)dw_ssi->ops;

        cpu_status = arc_lock_save();
        result = ssi_ops->writeb(dw_ssi, &we_xfer);
        arc_unlock_restore(cpu_status);

        while (E_OK == result) {
            chip_hw_mdelay(1);
            result = giga_status_read(dw_ssi, FLASH_WEL);
        }
```

```
        xfer.buf = (void *)wdata;
        xfer.len = 2;
        result = ssi_ops->writeb(dw_ssi, &xfer);
        if (E_OK == result) {
            chip_hw_mdelay(1);
        }
    }
    return result;
}
```

Add FLASH_WEL(0x05) command to giga_status_read() functions.

```
static int32_t giga_status_read(dw_ssi_t *dw_ssi, uint32_t status)
{
    int32_t result = E_OK;
    uint8_t dev_status = 0;
    uint8_t sts_pos = 0;

    dw_ssi_ops_t *ssi_ops = NULL;
    dw_ssi_xfer_t xfer = DW_SSI_XFER_INIT(0, 0, 0, 0);

    if ((NULL == dw_ssi) || (NULL == dw_ssi->ops)) {
        result = E_PAR;
    } else {
        ssi_ops = (dw_ssi_ops_t *)dw_ssi->ops;

        xfer.buf = (void *)&dev_status;
        xfer.len = 1;
        switch (status) {
            case FLASH_WIP:
            case FLASH_WEL:
                xfer.ins = 0x05;
                sts_pos = 1 << status;
                break;
            case FLASH_BP:
            case FLASH_E_ERR:
            case FLASH_P_ERR:
            case FLASH_STSR_WD:
                xfer.ins = CMD_READ_RSTS1;
                sts_pos = 1 << status;
                break;
            case FLASH_P_SUSPEND:
            case FLASH_E_SUSPEND:
                xfer.ins = CMD_READ_RSTS2;
                sts_pos = 1 << (status - 8);
                break;
```

```
            default:
                result = E_PAR;
        }

        if (E_OK == result) {
            result = ssi_ops->readb(dw_ssi, &xfer);
            if (E_OK == result) {
                if (sts_pos & dev_status) {
                    result = 1;
                }
            }
        }
    }
```

4. Add macro to the compiling command.

```
make FLASH_TYPE=giga FLASH_XIP=1 LOAD_XIP_TEXT_EN=1 bin
python post.py
obj_alpsMP_ref_design_v1_sensor/gnu_arcem6/sensor_gnu_arcem6.bin
firmware.bin xip
```