GigaDevice Semiconductor Inc.

GD32G5x3 系列 QSPI 高速模式时钟调整方法

应用笔记

AN210

1.0 版本

（2024 年 6 月）

目录

表索引

1. 简介

QSPI 是一种专用于和 flash 存储器通信的接口，可以支持单线，双线，四线 SPI flash。可以工作在普通模式、读轮询模式和内存映射模式。GD32G5x3 QSPI 支持 SDR 模式和 DDR 模式，时钟高达 200M。在读数据时，支持配置采样时钟为内部 QSPI SCK 或外部设备提供的 DQS 信号。高速外设 DDR 模式时，对时钟采样点的精度要求较高，QSPI 模块提供了一些功能来调整时钟采样点。本文介绍了在高速模式时如何调整时钟采样点的方法。

2. 时钟采样点调整功能位介绍

QSPI 模块提供了一些寄存器比特位，可用来调整时钟采样点。请参考*表 2-1. 时钟采样点调整功能位*。

表 2-1. 时钟采样点调整功能位

寄存器	位域名	功能描述
QSPI_CTL	OCKDV[3:0]	不分频时，输出时钟延时值，使用时需先置位OCKEN
	OCKDEN	输出时钟延时使能
	SSAMPLE	采样延时，允许配置QSPI在flash存储器驱动数据后二分之一个SCK时钟周期采样
QSPI_DCFG	CSNCKM	选择CSN在第一个SCK有效上升沿的之前一个还是两个SCK时钟周期拉低和在最后一个SCK有效上升沿之后一个还是两个SCK时钟周期拉高。
	DLYSCEN	延迟扫描功能使能，使用CPDM模块对时钟进行精细调整。
	RCKSEL	选择接收时钟源为QSPI内部产生的SCK还是外部设备提供的DQS。
	RXSFT[2:0]	接收移位步长，数据延长超过0.5个周期时，该位域可和SSAMPLE位一起调整接收采样点。
QSPI_TCFG	DDRHEN	DDR输出保持使能，在 DDR 模式下，分频时，延迟1/4个QSPI输出时钟周期再输出数据

3. 时钟采样点调整软件实现方法

本文提供了一些调整数据采样点的方法，在高速模式或外部环境恶劣的情况下，可以将以下几种方法结合一起使用，以达到高速稳定读写数据的目的。

3.1. 调整输出时钟

配置 OCKDEN 位和 OCKDV 位域使能输出时钟延迟功能，调整输出时钟采样点。

使用举例：

先配置 QSPI 时钟为 25M，擦除并编程数据至目标 flash。读回验证，flash 中已正确写入预期数据。

配置 flash 为 DTR 模式。

配置 QSPI 时钟为 200M，使用 DDR 四线读命令（0xED）读回验证，发现读出的数据不是预期数据。

使能输出时钟延迟功能，逐步累加时钟延迟值（0~15），使用 DDR 四线读命令（0xED）读回验证，直至读出的数据是预期数据，说明已经找到准确的采样点。为了提高代码的容忍度，将此时的延迟值记为 down_ockdv，继续逐步累加时钟延迟值（0~15），使用 DDR 四线读命令（0xED）读回验证，直至读出的数据不等于预期数据，将此时的延迟值记为 up_ockdv。将下限值和上限值取平均，作为最终的配置参数。可以使用当前配置进行后续操作。代码请参考*表 3-1. 输出时钟调整代码*。

注意：

1. QSPI 仅支持不分频时使用输出延迟功能。分频时，DDR 模式下，可通过配置 DDRHEN 位，延迟 1/4 个 QSPI 输出时钟周期再输出数据。

2. DDR 模式时，推荐 CSNCKM 位置 1，即选择 CSN 在第一个 SCK 有效上升沿的之前两个 SCK 时钟周期拉低和在最后一个 SCK 有效上升沿之后两个 SCK 时钟周期拉高。

表 3-1. 输出时钟调整代码

```
ockdv   = 0;
count1 = 0;
count2 = 0;
count3 = 0;
flag1 = 0;
flag2 = 0;

/* enable QSPI output clock delay */
qspi_output_clock_delay_enable();
/* select CSN falls and rises 2 sck cycles */
qspi_csn_edge_cycle(QSPI_CSN_2_CYCLE);
```

```
while(ockdv <= 15){
    /* configure the delay value */
    QSPI_CTL &= ~QSPI_CTL_OCKDV;
    QSPI_CTL |= ockdv << 12U;

    qspi_enable();

    qspi_cmd.instruction          = 0xED;
    qspi_cmd.instruction_mode = QSPI_INSTRUCTION_4_LINES;
    qspi_cmd.addr                    = 0x1000;
    qspi_cmd.addr_mode             = QSPI_ADDR_4_LINES;
    qspi_cmd.addr_size             = QSPI_ADDR_24_BITS;
    qspi_cmd.altebytes             = 0xFE;
    qspi_cmd.altebytes_mode     = QSPI_ALTE_BYTES_4_LINES;
    qspi_cmd.altebytes_size     = QSPI_ALTE_BYTES_8_BITS;
    qspi_cmd.data_mode             = QSPI_DATA_4_LINES;
    qspi_cmd.data_length          = 1;
    qspi_cmd.dummycycles          = 15;
    qspi_cmd.sioo_mode             = QSPI_SIOO_INST_EVERY_CMD;
    qspi_cmd.trans_rate           = QSPI_TCFG_DDREN;
    qspi_cmd.trans_delay          = 0;
    qspi_command_config(&qspi_cmd);
    QSPI_DTLEN = buf_size - 1;

    qspi_data_receive(rx_buffer);
    /* wait for the data receive completed */
    while(0U == qspi_flag_get(QSPI_FLAG_TC)){
    }
    /* clear the TC flag */
    qspi_flag_clear(QSPI_FLAG_TC);

    qspi_disable();

    /*   compare the data written and read out until they are equal */
    if(memory_compare(rx_buffer, tx_buffer, buf_size)){
        if(count1 == 0){
            /* the first time read-back verification is correct, record the down limit */
            down_ockdv = ockdv;
        }
        /* read back verification succeeds once, count1 is incremented by one */
        count1++;
```

```
            if(ockdv == 15U){
                /* record the up limit.
                    If the value is 15 and the read-back verification is correct,
                    the up limit is recorded as 15 */
                up_ockdv = ockdv;
                break;
            }
        }
        else{
            /* read back verification fails once, count2 is incremented by one */
            count2++;
        }
        /* read back and verify once, count3 is incremented by one */
        count3++;

        if((count3 == 1) && (count2 == 1)){
            flag1 = 1;
        } else if((count3 == 1) && (count1 == 1)){
            flag2 = 1;
        }

        if((flag1 == 1) && (count1 == 1)){
            flag2 = 1;
            count1 += count2;
        } else if((flag2 == 1) && (count3 != count1)){
            /* read-back verification runs from success to failure,
                record the up limit */
            up_ockdv = ockdv;
            break;
        }

        ockdv++;
    }

    /* use the calculated delay value to configure */
    ockdv = (up_ockdv + down_ockdv)/2;
    QSPI_CTL &= ~QSPI_CTL_OCKDV;
    QSPI_CTL |= ockdv << 12U;

    qspi_enable();
```

3.2. 粗调接收时钟

配置 RXSFT 位域和 SSAMPLE 位可实现步长为 0.5 个周期的接收采样点调整。

使用举例：

先配置 QSPI 时钟为 25M，擦除并编程数据至目标 flash。读回验证，flash 中已正确写入预期数据。

配置 QSPI 时钟为 200M，使用 SDR 四线快速读命令（0xEB）读回验证，发现读出的数据不是预期数据。

配置 RXSFT 位域和 SSAMPLE 位，逐步累加，使用 SDR 四线快速读命令（0xEB）读回验证，直至读出的数据是预期数据，说明已经找到准确的采样点。可以使用当前配置进行后续操作。代码请参考*表 3-2. 粗调接收时钟代码*。

表 3-2. 粗调接收时钟代码

```
i = 0;
j = 0;

while(i < 16){
    /* quad fast read */
    qspi_cmd.instruction         = 0xEB;
    qspi_cmd.instruction_mode = QSPI_INSTRUCTION_1_LINE;
    qspi_cmd.addr                  = 0x1000;
    qspi_cmd.addr_mode           = QSPI_ADDR_4_LINES;
    qspi_cmd.addr_size           = QSPI_ADDR_24_BITS;
    qspi_cmd.altebytes           = 0xFE;
    qspi_cmd.altebytes_mode    = QSPI_ALTE_BYTES_4_LINES;
    qspi_cmd.altebytes_size    = QSPI_ALTE_BYTES_8_BITS;
    qspi_cmd.data_mode           = QSPI_DATA_4_LINES;
    qspi_cmd.data_length         = 1;
    qspi_cmd.dummycycles         = 14;
    qspi_cmd.sioo_mode           = QSPI_SIOO_INST_EVERY_CMD;
    qspi_cmd.trans_rate          = 0;
    qspi_cmd.trans_delay         = 0;
    qspi_command_config(&qspi_cmd);
    QSPI_DTLEN = buf_size - 1;

    qspi_data_receive(rx_buffer);
    /* wait for the data receive completed */
    while(0U == qspi_flag_get(QSPI_FLAG_TC)){
    }
    /* clear the TC flag */
    qspi_flag_clear(QSPI_FLAG_TC);
```

```
        /* compare the data written and read out until they are equal.
           At this point, it indicates that the appropriate sampling point has been adjusted. */
        if(memory_compare(rx_buffer, tx_buffer, buf_size)){
            break;
        } else {
            if(j == 0){
                QSPI_CTL |= QSPI_SAMPLE_SHIFTING_HALFCYCLE;
                j = 1;
                i++;
            }
            else{
                QSPI_CTL &= ~QSPI_SAMPLE_SHIFTING_HALFCYCLE;
                j = 0;
                i++;
            }

            if((i%2 == 0) && (i != 0)){
                qspi_dcfg = QSPI_DCFG;
                qspi_dcfg &= ~QSPI_SHIFTING_7_CYCLE;
                qspi_dcfg |= DCFG_SSAMPLE(i/2);
                QSPI_DCFG = qspi_dcfg;
            }
        }
    }
```

3.3. 细调接收时钟

配置 DLYSCEN 位使能 CPDM 功能，调整接收采样点。

使用举例：

先配置 QSPI 时钟为 25M，擦除并编程数据至目标 flash。读回验证，flash 中已正确写入预期数据。

配置 QSPI 时钟为 200M，使用 DDR 四线读命令（0xED）读回验证，发现读出的数据不是预期数据。

使能延迟线采样模块及 CPDM，保持 CPDM 输出时钟相位等于输入时钟，逐步累加延迟步长计数值（0~127），使用 DDR 四线读命令（0xED）读回验证，直至读出的数据是预期数据，说明已经找到准确的采样点。为了提高代码的容忍度，将此时的延迟步长计数值记为 down_dlstcnt，继续逐步累加延迟步长计数值（0~127），使用 DDR 四线读命令（0xED）读回验证，直至读出的数据不等于预期数据，将此时的延迟步长计数值记为 up_dlstcnt。将下限值和上限值取平均，作为最终的配置参数。可以使用当前配置进行后续操作。代码请参考*表 3-3. 细调接收时钟代码（RCKSEL = 0）*及*表 3-4. 细调接收时钟代码（RCKSEL = 1）*。

注意：

1. 可以配置接收数据时钟为 SCK 或外部 flash 提供的 DQS 信号，均可使用 CPDM 调整。
2. 当配置接收数据时钟为 SCK 时，每次调整时，需先置位 DLYSCEN 位，使能 SCK 连续输出到 CPDM 模块，CPDM 使用配置的参数进行调整，调整完成后，清除 DLYSCEN 位，关闭 SCK 连续输出到 CPDM。
3. 当配置接收数据时钟为 DQS 时，不要配置 DLYSCEN 位，可直接使用 CPDM 时钟调整功能。但读数据时需要执行 2 次操作，确保 CPDM 状态已稳定。

表 3-3. 细调接收时钟代码（RCKSEL = 0）

```
count1 = 0;
count2 = 0;
count3 = 0;
flag1 = 0;
flag2 = 0;
temp_dlstcnt = 0;
k = 0;

while(temp_dlstcnt <= 127){
    /* configure the delay step count value */
    CPDM_CTL = CPDM_CTL_CPDMEN | CPDM_CTL_DLSEN;
    CPDM_CFG = (temp_dlstcnt << 8U) | 1;
    CPDM_CTL = CPDM_CTL_CPDMEN;

    /* enable QSPI clock phase delay function */
    qspi_disable();
    qspi_delay_scan_enable();
    for(k = 0; k < 20; k++);
    qspi_delay_scan_disable();
    qspi_enable();

    qspi_cmd.instruction          = 0xED;
    qspi_cmd.instruction_mode = QSPI_INSTRUCTION_4_LINES;
    qspi_cmd.addr                    = 0x1000;
    qspi_cmd.addr_mode             = QSPI_ADDR_4_LINES;
    qspi_cmd.addr_size             = QSPI_ADDR_24_BITS;
    qspi_cmd.altebytes             = 0xFE;
    qspi_cmd.altebytes_mode     = QSPI_ALTE_BYTES_4_LINES;
    qspi_cmd.altebytes_size     = QSPI_ALTE_BYTES_8_BITS;
    qspi_cmd.data_mode             = QSPI_DATA_4_LINES;
    qspi_cmd.data_length          = 1;
    qspi_cmd.dummycycles          = 15;
    qspi_cmd.sioo_mode             = QSPI_SIOO_INST_EVERY_CMD;
```

```
qspi_cmd.trans_rate           = QSPI_TCFG_DDREN;
qspi_cmd.trans_delay          = 0;
qspi_command_config(&qspi_cmd);
QSPI_DTLEN = buf_size - 1;

qspi_data_receive(rx_buffer);
/* wait for the data receive completed */
while(0U == qspi_flag_get(QSPI_FLAG_TC)){
}
/* clear the TC flag */
qspi_flag_clear(QSPI_FLAG_TC);

/*    compare the data written and read out until they are equal */
if(memory_compare(rx_buffer, tx_buffer, buf_size)){
    if(count1 == 0){
        /* the first time read-back verification is correct,record the down limit */
        down_dlstcnt = temp_dlstcnt;
    }
    /* read back verification succeeds once, count1 is incremented by one */
    count1++;

    if(temp_dlstcnt == 127){
        /* record the up limit.
           If the count is 127 and the read-back verification is correct,
           the up limit is recorded as 127 */
        up_dlstcnt = temp_dlstcnt;
        break;
    }
}
else{
    /* read back verification fails once, count2 is incremented by one */
    count2++;
}
/* read back and verify once, count3 is incremented by one */
count3++;

if((count3 == 1) && (count2 == 1)){
    flag1 = 1;
} else if((count3 == 1) && (count1 == 1)){
    flag2 = 1;
}

if((flag1 == 1) && (count1 == 1)){
```

```
            flag2 = 1;
            count1 += count2;
        } else if((flag2 == 1) && (count3 != count1)){
            /* read-back verification runs from success to failure,
               record the up limit */
            up_dlstcnt = temp_dlstcnt;
            break;
        }

        temp_dlstcnt++;
    }


    temp_dlstcnt = (down_dlstcnt + up_dlstcnt)/2;

    /* use the calculated delay line step count value to configure*/
    CPDM_CTL = CPDM_CTL_CPDMEN | CPDM_CTL_DLSEN;
    CPDM_CFG = (temp_dlstcnt << 8U) | 1;
    CPDM_CTL = CPDM_CTL_CPDMEN;

    qspi_disable();
    qspi_delay_scan_enable();
    for(k = 0; k < 20; k++);
    qspi_delay_scan_disable();
    qspi_enable();
```

表 3-4. 细调接收时钟代码（RCKSEL = 1）

```
count1 = 0;
count2 = 0;
count3 = 0;
flag1 = 0;
flag2 = 0;
temp_dlstcnt = 0;
k = 0;

/* select DQS as receive clock */
qspi_receive_clock_sel(QSPI_RECEIVE_CLOCK_DQS);

while(temp_dlstcnt <= 127){
    /* configure the delay step count value */
    CPDM_CTL = CPDM_CTL_CPDMEN | CPDM_CTL_DLSEN;
    CPDM_CFG = (temp_dlstcnt << 8U) | 1;
    CPDM_CTL = CPDM_CTL_CPDMEN;

    qspi_cmd.instruction         = 0xED;
    qspi_cmd.instruction_mode = QSPI_INSTRUCTION_4_LINES;
    qspi_cmd.addr                = 0x1000;
    qspi_cmd.addr_mode           = QSPI_ADDR_4_LINES;
    qspi_cmd.addr_size           = QSPI_ADDR_24_BITS;
    qspi_cmd.altebytes           = 0xFE;
    qspi_cmd.altebytes_mode      = QSPI_ALTE_BYTES_4_LINES;
    qspi_cmd.altebytes_size      = QSPI_ALTE_BYTES_8_BITS;
    qspi_cmd.data_mode           = QSPI_DATA_4_LINES;
    qspi_cmd.data_length         = 1;
    qspi_cmd.dummycycles         = 15;
    qspi_cmd.sioo_mode           = QSPI_SIOO_INST_EVERY_CMD;
    qspi_cmd.trans_rate          = QSPI_TCFG_DDREN;
    qspi_cmd.trans_delay         = 0;
    qspi_command_config(&qspi_cmd);
    QSPI_DTLEN = buf_size - 1;

    qspi_data_receive(rx_buffer);
    /* wait for the data receive completed */
    while(0U == qspi_flag_get(QSPI_FLAG_TC)){
    }
    /* clear the TC flag */
    qspi_flag_clear(QSPI_FLAG_TC);

    qspi_disable();
```

```
qspi_enable();

qspi_cmd.instruction          = 0xED;
qspi_cmd.instruction_mode = QSPI_INSTRUCTION_4_LINES;
qspi_cmd.addr                    = 0x1000;
qspi_cmd.addr_mode             = QSPI_ADDR_4_LINES;
qspi_cmd.addr_size             = QSPI_ADDR_24_BITS;
qspi_cmd.altebytes             = 0xFE;
qspi_cmd.altebytes_mode     = QSPI_ALTE_BYTES_4_LINES;
qspi_cmd.altebytes_size     = QSPI_ALTE_BYTES_8_BITS;
qspi_cmd.data_mode             = QSPI_DATA_4_LINES;
qspi_cmd.data_length          = 1;
qspi_cmd.dummycycles          = 15;
qspi_cmd.sioo_mode             = QSPI_SIOO_INST_EVERY_CMD;
qspi_cmd.trans_rate           = QSPI_TCFG_DDREN;
qspi_cmd.trans_delay          = 0;
qspi_command_config(&qspi_cmd);
QSPI_DTLEN = buf_size - 1;

qspi_data_receive(rx_buffer);
/* wait for the data receive completed */
while(0U == qspi_flag_get(QSPI_FLAG_TC)){
}
/* clear the TC flag */
qspi_flag_clear(QSPI_FLAG_TC);

/*    compare the data written and read out until they are equal */
if(memory_compare(rx_buffer, tx_buffer, buf_size)){
    if(count1 == 0){
        /* the first time read-back verification is correct,record the down limit */
        down_dlstcnt = temp_dlstcnt;
    }
    /* read back verification succeeds once, count1 is incremented by one */
    count1++;

    if(temp_dlstcnt == 127){
        /* record the up limit.
            If the count is 127 and the read-back verification is correct,
            the up limit is recorded as 127 */
        up_dlstcnt = temp_dlstcnt;
        break;
    }
}
```

```
        else{
            /* read back verification fails once, count2 is incremented by one */
            count2++;
        }
        /* read back and verify once, count3 is incremented by one */
        count3++;

        if((count3 == 1) && (count2 == 1)){
            flag1 = 1;
        } else if((count3 == 1) && (count1 == 1)){
            flag2 = 1;
        }

        if((flag1 == 1) && (count1 == 1)){
            flag2 = 1;
            count1 += count2;
        } else if((flag2 == 1) && (count3 != count1)){
            /* read-back verification runs from success to failure,
               record the up limit */
            up_dlstcnt = temp_dlstcnt;
            break;
        }

        temp_dlstcnt++;
    }


    temp_dlstcnt = (down_dlstcnt + up_dlstcnt)/2;

    /* use the calculated delay line step count value to configure*/
    CPDM_CTL = CPDM_CTL_CPDMEN | CPDM_CTL_DLSEN;
    CPDM_CFG = (temp_dlstcnt << 8U) | 1;
    CPDM_CTL = CPDM_CTL_CPDMEN;
```

4. 版本历史

表 4-1. 版本历史

版本号.	说明	日期
1.0	首次发布	2024 年 06 月 30 日

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the “Product”), is owned by the Company under the intellectual property laws and treaties of the People’s Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as it’s suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as it’s suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.