GigaDevice Semiconductor Inc.

QSPI Clock Adjustment Method in High-speed Mode for GD32G5x3 Series

Application Note

AN210

Revision 1.0

(June. 2024)

Table of Contents

List of Tables

1. Introduction

The QSPI is a specialized interface that can communicate with flash memories. This interface supports single, dual or quad SPI Flash. It can operate in normal mode, read polling mode and memory map mode. The QSPI of GD32G5x3 supports SDR mode and DDR mode with a clock of up to 200M. When reading data, it supports configuring the sampling clock to be the DQS signal provided by the external device or the internal QSPI SCK. In DDR mode, high-speed peripherals require high accuracy of clock sampling points. The QSPI module provides some functions to adjust the clock sampling point. This application note introduces the method of how to adjust the clock sampling point in high-speed mode.

2. Clock sampling point adjustment function bit introduction

The QSPI module provides a number of register bits that can be used to adjust the clock sample points. Please refer to ***Table 2-1. Clock sampling point adjustment function bit***.

Table 2-1. Clock sampling point adjustment function bit

Register	Bit name	Function descriptions
QSPI_CTL	OCKDV[3:0]	Output clock delay value when clock is not divided. These bits only useful when OCKDEN is enabled.
	OCKDEN	Output clock delay enable
	SSAMPLE	Sample delay, allows QSPI to be configured to sample data at 1/2 SCK clock cycle after flash memory drive.
QSPI_DCFG	CSNCKM	Select whether the CSN falls and rises one or two SCK clock cycles before the first SCK effective rising edge and after the last SCK effective rising edge.
	DLYSCEN	Delay scan enable. The CPDM module is used to fine-tune the clock.
	RCKSEL	Select the receiving clock source is the SCK generated inside the QSPI or the DQS provided by an external device.
	RXSFT[2:0]	Shift receive step. This bit field can be used to adjust the received sampling point together with the SSAMPLE bit when the rececive data delay more than 0.5 cycle.
QSPI_TCFG	DDRHEN	DDR output hold enable. In DDR mode, when the clock is divided, the QSPI output clock cycle is delayed by 1/4 before the data is output.

3. Clock sampling point adjustment function software implementation

This application note provides some methods to adjust the data sampling point in high-speed mode or in the case of bad external environment. The following methods can be used together to achieve the purpose of reading and writing data stably in high-speed mode.

3.1. Adjust the output clock

Configure the OCKDEN bit and OCKDV bit field to enable the output clock delay function and adjust the output clock sample point.

Examples of usage:

First, configure the QSPI clock to 25M, erase and program the data to the target flash. Read back and verify that the expected data has been correctly written to flash.

Configure flash to DTR mode.

Configure the QSPI clock to 200M, and use the DDR four-wire read command (0xED) to read back and verify that the read out data is not the expected data.

Enable the output clock delay function and accumulate the clock delay value (0~15) gradually. Use the DDR four-line read command (0xED) to read back and verify, until the data read out is the expected data. It indicates that the accurate sampling point has been found. In order to improve the tolerance of the code, the delay value at this time is denoted as down_ockdv, and the clock delay value (0~15) is accumulated gradually, and use the DDR four-line read command (0xED) to read back and verify, until the data read out is not equal to the expected data, and the delay value at this time is denoted as up_ockdv. The lower and upper limits are averaged and used as the final configuration parameters. The current configuration can be used for subsequent operations. Please refer to ***Table 3-1. Output clock adjustment code*** for the code.

Note:

1. QSPI only supports the use of output delay function when clock is not divided. In DDR mode, the DDRHEN bit can be configured to delay 1/4 QSPI output clock cycle before output data.

2. In DDR mode, it is recommended to set the CSNCKM bit to 1, that is, CSN is selected to pull down two SCK clock cycles before the first effective rising edge of SCK and to pull up two SCK clock cycles after the last effective rising edge of SCK.

Table 3-1. Output clock adjustment code

```
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. Coarse adjustment of the receiving clock

Configure the RXSFT bit field and the SSAMPLE bit to realize the adjustment of the received sampling point with a step size of 0.5 cycle.

Examples of usage:

First, configure the QSPI clock to 25M, erase and program the data to the target flash. Read back and verify that the expected data has been correctly written to flash.

Configure the QSPI clock to 200M, and use the SDR four-wire fast read command (0xEB) to read back and verify that the read out data is not the expected data.

Configure the RXSFT bit field and SSAMPLE bit and accumulate gradually. use the SDR four-wire fast read command (0xEB) to read back and verify, until the data read out is the expected data. It indicates that the accurate sampling point has been found. The current configuration can be used for subsequent operations. Please refer to ***Table 3-2. Coarse adjustment of the receiving clock code*** for the code.

Table 3-2. Coarse adjustment of the receiving clock code

```
    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. Fine adjustment of the receiving clock

Configure the DLYSCEN bit to enable the CPDM function and adjust the received sampling point.

Examples of usage:

First, configure the QSPI clock to 25M, erase and program the data to the target flash. Read back and verify that the expected data has been correctly written to flash.

Configure flash to DTR mode.

Configure the QSPI clock to 200M, and use the DDR four-wire read command (0xED) to read back and verify that the read out data is not the expected data.

Enable the delay line sampling module and CPDM and keep the CPDM output clock phase equal to the input clock. Accumulate the delay step count value (0~127) gradually. Use the DDR four-line read command (0xED) to read back and verify, until the data read out is the expected data. It indicates that the accurate sampling point has been found. In order to improve the tolerance of the code, the delay step count value at this time is denoted as down_dlstcnt, and the delay step count value (0~127) is accumulated gradually, and use the DDR four-line read command (0xED) to read back and verify, until the data read out is not equal to the expected data, and the delay step count value at this time is denoted as up_dlstcnt. The lower and upper limits are averaged and used as the final configuration parameters. The current configuration can be used for subsequent operations. Please refer to ***Table 3-3. Fine adjustment of the receiving clock code (RCKSEL = 0)*** and ***Table 3-4. Fine adjustment of the receiving clock code (RCKSEL = 1)*** for the code.

Note:

1. The receiving data clock can be configured to the DQS signal provided by external flash or SCK, which both can be adjusted by CPDM.

2. When the receiving data clock is configured as SCK, the DLYSCEN bit should be set first to enable SCK to be output to the CPDM module continuously during each adjustment, and the CPDM should adjust it using the configured parameters. After adjustment, the DLYSCEN bit should be cleared to turn off SCK continuous output to CPDM.

3. When the received data clock is configured as DQS, user not need to configure the DLYSCEN bit at this time, and can directly use the CPDM clock adjustment function. However, it needs to perform reading operations twice to ensure that the CPDM state is stable.

Table 3-3. Fine adjustment of the receiving clock code (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();
```

Table 3-4. Fine adjustment of the receiving clock code (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. Revision history

Table 4-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	June.30, 2024