Cascade DCO Configuration

Application Note

Contents

Figures

Tables

1 SiT9514x DCO configuration

1.1 General description

This application note details the SiT9514x DCO configuration procedures. DCO configuration to control frequency through pins and register updates is described along with the example codes to illustrate the programming sequence in detail.

1.2 Key features

- DCO in sync mode configuration
- DCO in free-run mode configuration

1.3 Introduction

Digitally controlled oscillator (DCO) mode of operation is used for changing the output frequency of a PLL using either software control on the serial interface or pin control. A predefined change in frequency can be programmed for each PLL. After that, an increase (FINC) or decrease (FDEC) command can be given to the PLL to make the change in output frequency effective. Alternatively, appropriate GPIOs are chosen for the trigger of the DCO function. A low-to-high transition (as an edge detect) is used to trigger the DCO increment or decrement. Any relative change in frequency from as fine as 5 ppt to as coarse as 100 ppm is available in DCO mode.

DCO mode is available in both free-run and synchronized (sync) modes of operation. It can be accessed from both pin control and from register map. This application note provides the detailed procedures for DCO sync mode and free-run mode from pin control and register map control.

1.4 Applicability

This document and references within to SiT9514x apply to the SiT95141, SIT95145, SiT95147, and SIT95148 devices, except where noted otherwise.

2 DCO code calculation for predefined frequency change

The DCO code should be calculated and written into the appropriate registers to achieve the predefined change in frequency while doing a DCO increment or decrement as desired by the user.

The calculation is largely dependent on whether the PLL is running in sync mode or free-run mode. The following sections describe the DCO code calculation in both sync mode and free-run mode.

2.1 Calculation of DCO fractional and integer code in sync mode

The DCO control word has fractional and integer code components that are computed based on the desired DCO frequency step (ppm) and the feedback divider value DIVN2.

Step 1: DIVN2 calculation

The `DIVN2` can be calculated by obtaining the values of `PLL_DIVN2_INT` (integer part), `PLL_DIVN2_FRACNUM` (fractional part – numerator), and `PLL_DIVN2_FRACDEN` (fractional part – denominator) from the register mentioned in the table PLL DIVN2 divider configuration.

`DIVN2 = PLL_DIVN2_INT + PLL_DIVN2_FRACNUM / PLL_DIVN2_FRACDEN` (refer this value from the `.nvm` file)

Note:

- `PLL_DIVN2_INT`: 21-bit integer
- `PLL_DIVN2_FRACNUM`: signed 32-bit (two's complement)
- `PLL_DIVN2_FRACDEN`: unsigned 32-bit

Table 1: PLL DIVN2 divider configuration

Page number	Register address	Bit number	Description
PLL page (Any one of the four pages A, B, C, or D)	0x1d	7:0	PLL_DIVN2_INT [7:0]
	0x1e	7:0	PLL_DIVN2_INT [15:8]
	0x1f	4:0	PLL_DIVN2_INT [20:16]
	0x20	7:0	PLL_DIVN2_FRACNUM [7:0]
	0x21	7:0	PLL_DIVN2_FRACNUM [15:8]
	0x22	7:0	PLL_DIVN2_FRACNUM [23:16]
	0x23	7:0	PLL_DIVN2_FRACNUM [31:24]
	0x24	7:0	PLL_DIVN2_FRACDEN [7:0]
	0x25	7:0	PLL_DIVN2_FRACDEN [15:8]
	0x26	7:0	PLL_DIVN2_FRACDEN [23:16]
	0x27	7:0	PLL_DIVN2_FRACDEN [31:24]

Step 2: DCO code calculation

The `DCO code` value can be calculated from the predefined frequency change (`dco_ppm`) and the `DIVN2` value calculated in the previous step.

`DCO code = DIVN2 * dco_ppm * 1e-6`

`PLL_DCO_INT` = Integer part of `DCO code`

`PLL_DCO_FRAC = (dco_code - PLL_DCO_INT) * PLL_DIVN2_FRACDEN` # Fractional part of dco_code

Note:

- `PLL_DCO_INT`: 10-bit number
- `PLL_DCO_FRAC`: 32-bit number

Example calculation:

```
dco_ppm = 10 ppm,
PLL_DIVN2_INT = 860,
PLL_DIVN2_FRACNUM = 343597380,
PLL_DIVN2_FRACDEN = 2147483625 (from .nvm file)

PLL_DIVN2_FRACNUM_SIGNED = twos_comp_to_signed_int (PLL_DIVN2_FRACNUM, 32) =
343597380

DIVN2 = PLL_DIVN2_INT + PLL_DIVN2_FRACNUM_SIGNED / PLL_DIVN2_FRACDEN

= 860 + 343597380/2147483625

dco_code = DIVN2 * dco_ppm * 1e-6

= (860 + 343597380/2147483625) * 10 * 1e-6

PLL_DCO_INT = floor (dco_code+0.5) = 0

PLL_DCO_FRAC = floor ((dco_code - PLL_DCO_INT) * PLL_DIVN2_FRACDEN)

= floor ((dco_code - 0) * 2147483625) = 18471795
```

Note:

- `twos_comp_to_signed_int` (Number, Word length) – A Python function use to convert two's complement to signed integer.
- `floor (number)` – A Python function used to return the closest integer value which is less than or equal to the given numeric value.

2.2 Calculation of DCO fractional code in free-run mode

The DCO control word has fractional and integer code components that are computed based on the desired DCO frequency step (ppm) and the feedback divider value DIVN.

Step 1: DIVN calculation

The `DIVN` value can be calculated by obtaining the values of `PLL_DIVN_INT (Integer part)`, `PLL_DIVN_FRAC_SIGNED (Fractional part – Numerator)` from the register mentioned in the table below.

`DIVN = PLL_DIVN_IN + PLL_DIVN_FRAC_SIGNED/2^32` refer this value (from .nvm file)

Note:

- `PLL_DIVN_INT`: 9-bit
- `PLL_DIVN_FRAC`: Signed 32-bit (two's complement, LSB 8 bits are 0)

Table 2: PLL DIVN divider register configuration

Page number	Register address	Bit number	Description
PLL page (Any one of 4 pages A, B, C, or D)	0x17	6:0	PLL_DIVN_INT[6:0]
	0x18	4	PLL_DIVN_INT[7]
	0x1a	7:0	PLL_DIVN_FRAC[15:8]
	0X1b	7:0	PLL_DIVN_FRAC[23:16]
	0x1c	7:0	PLL_DIVN_FRAC[31:25]

The DCO code can be calculated from the predefined frequency change (dco_ppm) and the DIVN value calculated in the previous step.

```
dco_code = DIVN * dco_ppm * 1e-6 * 2^32
PLL_DCO_FRAC = floor (dco_code)
```

Note:

- PLL_DCO_FRAC: 32-bit number

Example calculation

```
DCO step required, dco_ppm = 10 ppm, PLL_DIVN_INT=67, DIVN_FRAC = 553573376
(from .nvm file)
PLL_DIVN_FRAC_SIGNED = twos_comp_to_signed_int (DIVN_FRAC, WL_frac) = 553573376
DIVN = PLL_DIVN_INT + PLL_DIVN_FRAC_SIGNED/2^32
= 67+553573376/2^32
= 67.128888845443725585937
dco_code = DIVN * 10 * 1e-6 * 2^32
= 67.128888845443725585937* 10 * 1e-6 *2^32
= 2883163.822080
PLL_DCO_FRAC = floor (dco_code)
PLL_DCO_FRAC = 2883163
```

Note:

1. twos_comp_to_signed_int (Number, Word length) – Python function use to convert 2's compliment to signed integer.
2. floor(number) – Python function used to return the closest integer value which is less than or equal to given numeric value.

3 Steps to perform DCO

3.1 DCO increment and decrement using registers

- Calculate PLL_DCO_INT and PLL_DCO_FRAC values as described in section Calculation of DCO fractional and integer code in sync mode
- Configure the device to do DCO from the register map (Set page 0, 0x23 to 0)
- Set the direction of PLL_SEL0 and PLL_SEL1 pins as input (page 0, 0x15) – Only for the SiT95147 variant
- Select the PLL to be incremented or decremented with the DCO step:
  - Page 0, 0x05 register for SiT95141/SiT95145/SiT95148
  - Page 0, 0x05 register or PLL_SEL0 and PLL_SEL1 pins for SiT95147
- Do small trigger update (update the device configuration):
  - Write 0 to 0x0f[2] on page 0
  - Write 1 to 0x0f[2] on page 0
  - Write 0 to 0x0f[2] on page 0
- Write the PLL_DCO_INT and PLL_DCO_FRAC values to the respective registers (PLL page, 0x31 to 0x37)
- Enable DCO sync mode/free-run mode as required (PLL page, 0x37/ 0x35):
  - To increment:
    - Write 0 to 0x35[3] on the respective PLL page
    - Write 1 to 0x35[3] on the respective PLL page
    - **Note:** Add a delay with the minimum pulse width required as described in section Minimum pulse width, maximum update rate, DCO range and step size.
    - Write 0 to 0x35[3] on the respective PLL page
  - To decrement:
    - Trigger DCO update:
    - Write 1 to 0x35[2] on the respective PLL page
    - **Note:** Add a delay with the minimum pulse width required as described in section Minimum pulse width, maximum update rate, DCO range and step size.
    - Write 0 to 0x35[2] on the respective PLL page

Note:

- To perform the DCO increment and decrement operation via register write, FINC (Increment) and FDEC (Decrement) pins should be driven to logic 0.
- For DCO in free-run, only PLL_DCO_FRAC needs to be calculated and written to the device.

3.2 DCO increment and decrement using FINC and FDEC pins

- Calculate PLL_DCO_INT and PLL_DCO_FRAC values as described in section Calculation of DCO fractional and integer code in sync mode.
- Configure the device to do DCO using FINC and FDEC pins (Page 0, 0x23)
- Configure the FINC and FDEC pins as input:
  - Page 0, 0x14 and 0x17 registers for SiT95141/SiT95145/SiT95148 variants
  - Page 0, 0x16 and 0x17 registers for SiT95147 variant
- Set the direction of PLL_SEL0 and PLL_SEL1 pins as input (page 0, 0x15) – Only for SiT95147 variant
- Select the PLL to be incremented or decremented with the DCO step:
  - Page 0, 0x05 register for SiT95141/SiT95145/SiT95148
  - Page 0, 0x05 register or PLL_SEL0 and PLL_SEL1 pins for SiT95147
- Do small trigger update (update the device configuration):
  - Write 0 to 0x0f[2] on page 0
  - Write 1 to 0x0f[2] on page 0
  - Write 0 to 0x0f[2] on page 0
- Mask DCO operation of the respective PLL, if that PLL frequency needs to remain unaffected during DCO increment/decrement operation (PLL page, 0x35)
- Write the PLL_DCO_INT and PLL_DCO_FRAC values to the respective registers (PLL page, 0x31 to 0x37)
- Enable DCO sync/free-run mode as required (PLL page, 0x37/ 0x35)
- To increment toggle FINC as below:
  - FINC = 0 (logic 0);
  - FINC = 1 (logic 1); Perform DCO increment operation from pin
  - **Note:** Add a delay with the minimum pulse width required, as described in section Minimum pulse width, maximum update rate, DCO range and step size.
  - FINC = 0 (logic 0);
- To decrement toggle FDEC as below:
  - FDEC = 0 (logic 0);
  - FDEC = 1 (logic 1); Perform DCO increment operation from pin
  - **Note:** Add a delay with the minimum pulse width required as described in section Minimum pulse width, maximum update rate, DCO range and step size.
  - FDEC = 0 (logic 0);

Note:

- To perform the DCO increment and decrement operations via FINC and FDEC pins, PLL page register bits 0x35[3](Increment) and 0x35[2](Decrement) should be 0.
- For DCO in free-run, only PLL_DCO_FRAC needs to be calculated and written to the device.

Table 3: Registers required for DCO – SIT95141/SiT95145/SiT95148

Page number	Register address	Bit number	Description
Page 0	0x23	6	If reg23[6] = 0; DCO from register map If reg23[6] = 1; DCO from pin
	0x14	7	If reg14[7] = 0; set direction of FDEC as input If reg14[7] = 1; set direction of FDEC as output
	0x17	3	If reg17[3] = 0; set direction of FINC as input If reg17[3] = 1; set direction of FINC as output
	0x05	3:2	If reg05[3:2] = 0; DCO_PLLA is selected If reg05[3:2] = 1; DCO_PLLB is selected If reg05[3:2] = 2; DCO_PLLC is selected If reg05[3:2] = 3; DCO_PLLD is selected
	0x0f	2	Command directive bit to update NVM bank

Page number	Register address	Bit number	Description
Continuous			
PLL page (Any one of 4 pages A, B, C, or D)	0x31	7:0	PLL_DCO_FRAC[7:0]
	0x32	7:0	PLL_DCO_FRAC[15:8]
	0x33	7:0	PLL_DCO_FRAC[23:16]
	0x34	7:0	PLL_DCO_FRAC[31:24]
	0x36	7:0	PLL_DCO_INT[9:2]
	0x37	7:6	PLL_DCO_INT[1:0]
	0x37	5	1: DCO sync mode enabled 0: DCO sync mode disable
	0x35	1	1: DCO free-run mode enabled 0: DCO free-run mode disable
	0x35	0	1: DCO Mask is enable 0: DCO Mask is disable
	0x35	2	1: DCO Decrement bit is enable 0: DCO Decrement bit is disable
	0x35	3	1: DCO Increment bit is enable 0: DCO Decrement bit is disable

Table 4: Registers required for DCO – SiT95147

Page number	Register address	Bit number	Description
Page 0	0x23	6	If reg23[6] = 0; DCO from register map If reg23[6] = 1; DCO from pin
	0x16	3	If reg16[3] = 0; set direction of FDEC as input If reg16[3] = 1; set direction of FDEC as output
	0x17	3	If reg17[3] = 0; set direction of FINC as input If reg17[3] = 1; set direction of FINC as output
	0x05	3:2	If reg05[3:2] = 0; DCO_PLLA is selected If reg05[3:2] = 1; DCO_PLLB is selected If reg05[3:2] = 2; DCO_PLLC is selected If reg05[3:2] = 3; DCO_PLLD is selected
	0x15	7	If reg15[7] = 0; it will set direction of PLL_SEL0 as input If reg15[7] = 1; it will set direction of PLL_SEL0 as output
	0x15	3	If reg15[3] = 0; it will set direction of PLL_SEL1 as input If reg15[3] = 1; it will set direction of PLL_SEL1 as output
	0x0f	2	Command directive bit to update NVM bank

Page number	Register address	Bit number	Description
Continuous			
PLL Page (Any one of 4 pages A, B, C, or D)	0x31	7:0	PLL_DCO_FRAC[7:0]
	0x32	7:0	PLL_DCO_FRAC[15:8]
	0x33	7:0	PLL_DCO_FRAC[23:16]
	0x34	7:0	PLL_DCO_FRAC[31:24]
	0x36	7:0	PLL_DCO_INT[9:2]
	0x37	7:6	PLL_DCO_INT[1:0]
	0x37	5	1: DCO sync mode enabled 0: DCO sync mode disable
	0x35	1	1: DCO free-run mode enabled 0: DCO free-run mode disable
	0x35	0	1: DCO Mask is enable 0: DCO Mask is disable
	0x35	2	1: DCO Decrement bit is enable 0: DCO Decrement bit is disable
	0x35	3	1: DCO Increment bit is enable 0: DCO Decrement bit is disable

4 Minimum pulse width, maximum update rate, DCO range and step size

4.1 DCO in sync mode

For DCO in sync mode, the minimum pulse width and maximum update rate depends on the PLL input clock frequency (`finA, finB, finC, finD`):

- Minimum pulse width: `(tDCO_PW_MIN ) = 2/finA` (e.g. `finA = 2 MHz, tDCO_PW_MIN = 2/2 MHz = 1us`)
- Maximum update rate frequency: `(fDCO_max) = finA/4` (e.g. `finA = 2 MHz, fDCO_max = 500 kHz`)
- Settling time: `≈ 5/(2*pi*BW)`
- DCO range: `+/−500 ppm`
- Minimum DCO step size: `5 ppt`

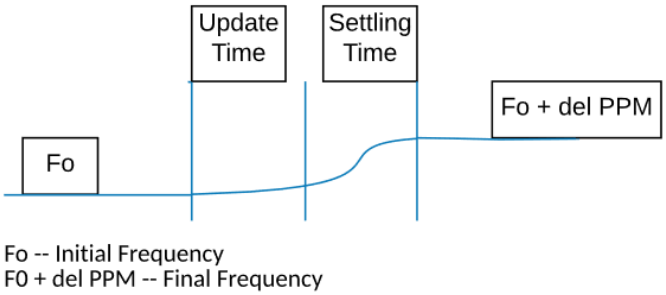

Figure 1: Output frequency update in sync mode

Note: This pulse width fulfills both pin and register control operation requirements.

4.2 DCO in free-run mode

- Minimum pulse width (tDCO_PW_MIN) = 100 nS
- Maximum update rate frequency (fDCO_max) = 1 MHz
- Settling time ≈ 5 / (2*pi*500 Khz) ≈ 1.59 uS
- DCO Range : +/-500 ppm
- Minimum DCO step size: 5 ppt

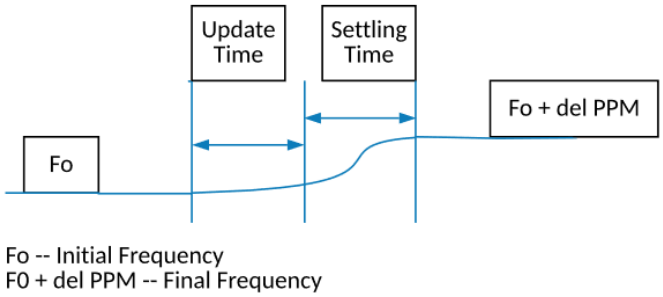

Figure 2: Output frequency update in free-run mode

Note: This pulse width fulfills both pin and register control operation requirements.

5 DCO output readback procedure

5.1 Free-run mode

The following steps can be used to readback the DCO debug registers and to confirm the DCO operation has been done in free-run mode.

Step 1: Readback the integer and fractional parts of DIVN before DCO operation by performing the DCO free-run debug readback sequence to get the `Pre_dco_divn_int` and `Pre_dco_divn_frac` values.

Step 2: Carry out the desired DCO increment/decrement operation.

Step 3: Readback the integer and fractional parts of DIVN after DCO operation by performing the 'DCO free-run debug readback sequence' to get the `Post_dco_divn_int` and `Post_dco_divn_frac` values.

Step 4: Compare the DIVN values before and after DCO operation using the equation:

- `Post_dco_divn_int +Post_dco_divn_frac = Pre_dco_divn_int + Pre_dco_divn_frac +/- PLL_DCO_FRAC` (consider + for increment operation, – for decrement operation).
- RHS should be equal to LHS if a DCO operation has been performed.

Note: For `PLL_DCO_FRAC`, refer to section Calculation of DCO fractional code in free-run.

5.1.1 DCO free-run debug readback sequence

- Program debug select address to read the `Pre_dco_divn_int`
  - `i2c.i2cw(0x69,0xff,0x0x0a)` – Select PLL-A (page number can change as per the PLL selection)
  - `i2c.i2cw(0x69,0xb3,0x1b)`
- Read the debug data (`dco_divn_int`)
  - `dco_divn_int [7:0] = i2c.i2cr(0x69,0xb9)`
  - `dco_divn_int [15:8] = i2c.i2cr(0x69,0xba)`
  - `dco_divn_int [23:16] = i2c.i2cr(0x69,0xbb)`
  - `dco_divn_int [31:24] = i2c.i2cr(0x69,0xbc)`

Table 5: Readback registers comparison with DIVN_INT_COMPARE

Readback register	BIT range	DIVN_INT	Comment
0xB9	0:6	0x17[0:6]	MSB 24-bits are zero.
0xBA	7	0x18[4]	
0xBA	9:15		
0xBB	16:23		
0XBC	24:31		

Note that readback registers value can be different from `DIVN_INT`.

Readback register values are used for calculating the current value of `DIVN_INT` after the DCO update.

- Program debug select address to read the `dco_divn_frac`:
  - `I2c.i2cw(0x69,0xb3,0x1c)`
- Read the debug data (`dco_divn_frac`):
  - `dco_divn_frac [7:0] = i2c.i2cr(0x69,0xb9)`
  - `dco_divn_frac [15:8] = i2c.i2cr(0x69,0xba)`
  - `dco_divn_frac [23:16] = i2c.i2cr(0x69,0xbb)`
  - `dco_divn_frac [31:24] = i2c.i2cr(0x69,0xbc)`

Table 6: Read back registers comparison with DIVN_FRAC_COMPARE

Readback register	BIT range	DIVN_FRAC	Comment
0xB9	0:7	8d'0	LSB 8 bits are zero in the DIVN_FRAC registers.
0xBA	8:15	0x1a[8:15]	
0xBB	16:23	0x1b[16:23]	
0XBC	24:31	0x1c[24:31]	

Note: If the SPI interface is used for debug read operation, `Spi_clock_frequency` should be less than or equal to `xo_frequency/10`.

Note that readback register values can be different from `DIVN_FRAC`.

Readback register values are used for calculating the current value of `DIVN_INT` after `DCO` update.

DIVN calculation

The DIVN value can be calculated by obtaining the values of `PLL_DIVN_INT (Integer part)`, `PLL_DIVN_FRAC_SIGNED` (fractional part – numerator) from the register mentioned in Table 2.

`DIVN = PLL_DIVN_IN + PLL_DIVN_FRAC_SIGNED/2^32` refer to this value (from .nvm file)

Note:

1. `PLL_DIVN_INT`: 9-bit
2. `PLL_DIVN_FRAC`: Signed 32-bit (two's complement, LSB 8 bits are 0)

5.2 Sync mode

The following steps can be used to readback the DCO debug registers and confirm DCO operation has been done in sync mode.

Step 1: Readback the integer and fractional parts of DIVN2 before DCO operation by performing the 'DCO sync mode debug readback sequence' to get `Pre_dco_divn2_int`, `Pre_dco_divn2_frac_num` and `Pre_dco_divn2_frac_den` values.

Step 2: Carry out the desired DCO increment/decrement operation.

Step 3: Readback the integer and fractional parts of DIVN2 after DCO operation by performing the 'DCO sync mode debug readback sequence' to get `Post_dco_divn2_int_dsm` and `Post_dco_divn2_frac_num_dsm` and `Post_dco_divn2_frac_den_dsm` values.

Step 4: Compare the DIVN2 values before and after DCO operation using the equation:
`Pre_dco_divn2_frac_den_dsm*Post_dco_divn2_int_dsm+Post_dco_divn2_frac_num_dsm=Pre_dco_divn2_frac_den_dsm*Pre_dco_divn2_int_dsm+Pre_dco_divn2_frac_num_dsm +/- (Pre_dco_divn2_frac_den_dsm*PLL_DCO_INT + PLL_DCO_FRAC)` (consider + for increment operation or consider – for decrement operation)

RHS should be equal to LHS if DCO operation has been performed

Note: For PLL_DCO_FRAC and PLL_DCO_INT, refer to section Calculation of DCO fractional and integer code in sync mode.

5.2.1 DCO sync mode debug readback sequence

- Program debug select address to read the dco_divn2_int_dsm:
  - `I2c.i2cw(0x69,0xff,0x0x0a)` # select PLL-A (page number can change as per the PLL selection)
  - `I2c.i2cw(0x69,0xb3,0x12)`
- Read the debug data (`Pre_dco_divn2_int_dsm`):
  - `dco_divn2_int_dsm[7:0] = i2c.i2cr(0x69,0xb9)`
  - `dco_divn2_int_dsm[15:8] = i2c.i2cr(0x69,0xba)`
  - `dco_divn2_int_dsm[23:16] = i2c.i2cr(0x69,0xbb)`
  - `dco_divn2_int_dsm[31:24] = i2c.i2cr(0x69,0xbc)`

Table 7: Readback registers comparison with DIVN2 INT COMPARE

Readback register	BIT range	DIVN2_INT	Comment
0xB9	0:7	0x1d[0:7]	11 MSB's will be zero
0xBA	8:15	0x1e[8:15]	
0xBB	16:23	0x1f[16:20]	
0XBC	24:31		

Note that readback registers value can be different from `DIVN2_INT`, Table 7 can be used for calculating the current value of `DIVN2_INT` after readback.

- Program debug select address to read the `dco_divn2_frac_num_dsm`:
  - `I2c.i2cw(0x69,0xb3,0x13)`
- Read the debug data (`dco_divn2_frac_num_dsm`):
  - `dco_divn2_frac_num_dsm[7:0] = i2c.i2cr(0x69,0xb9)`
  - `dco_divn2_ frac_num _dsm[15:8] = i2c.i2cr(0x69,0xba)`
  - `dco_divn2_ frac_num _dsm[23:16] = i2c.i2cr(0x69,0xbb)`
  - `dco_divn2_ frac_num _dsm[31:24] = i2c.i2cr(0x69,0xbc)`

Table 8: Readback registers comparison with DIVN2_FRAC_NUM_COMPARE

Readback register	BIT range	DIVN2_FRAC_NUM
0xB9	0:7	0x20[0:7]
0xBA	8:15	0x21[8:15]
0xBB	16:23	0x22[16:23]
0XBC	24:31	0x23[24:31]

Note that readback registers value can be different from `DIVN2_FRAC_NUM`.

Readback register values are used for calculating the current value of `DIVN2_FRAC_NUM` after DCO update.

- Program debug select address to read the `Pre_dco_divn2_frac_den_dsm`:
  - `I2c.i2cw(0x69,0xb3,0x14)`
- Read the debug data (`Pre_dco_divn2_frac_den_dsm`):
  - `dco_divn2_frac_den_dsm[7:0] = i2c.i2cr(0x69,0xb9)`
  - `dco_divn2_ frac_ den _dsm[15:8] = i2c.i2cr(0x69,0xba)`
  - `dco_divn2_ frac_ den _dsm[23:16] = i2c.i2cr(0x69,0xbb)`
  - `dco_divn2_ frac_ den _dsm[31:24] = i2c.i2cr(0x69,0xbc)`

Table 9: Readback registers compare with DIVN2_FRAC_DEN_COMPARE

Readback register	BIT range	DIVN2_FRAC_DEN
0xB9	0:7	0x24[0:7]
0xBA	8:15	0x25[8:15]
0xBB	16:23	0x26[16:23]
0XBC	24:31	0x27[24:31]

Note that readback registers can be different from DIVN2_FRAC_DEN.

Readback register values are used for calculating the current value of DIVN2_FRAC_NUM after DCO update.

Note:

- If SPI interface is used for debug read operations, `Spi_clock_frequency` should be less than or equal to `fin_frequency/10`.

DIVN2 readback calculation

The `DIVN2` can be calculated by obtaining the values of `PLL_DIVN2_INT`(Integer part), `PLL_DIVN2_FRACNUM`(fractional part – numerator) and `PLL_DIVN2_FRACDEN` (fractional part – denominator) from the register mentioned in the table PLL DIVN2 divider configuration.

```
DIVN2 = PLL_DIVN2_INT + PLL_DIVN2_FRACNUM / PLL_DIVN2_FRACDEN
```

Note:

- `PLL_DIVN2_INT`: 21-bit integer
- `PLL_DIVN2_FRACNUM`: Signed 32-bit (two's complement)
- `PLL_DIVN2_FRACDEN`: Unsigned 32-bit

6 Example code

6.1 DCO through register map (SiT9514x, sync mode profile)

Do a 10 ppm increment and decrement on PLL-A using the register map:

Step 1: DCO code calculation

Obtain the PLL_DIVN2_INT, PLL_DIVN2_FRACNUM_SIGNED, PLL_DIVN2_FRACDEN (from .nvm file):

```
DIVN2 = PLL_DIVN2_INT + (PLL_DIVN2_FRACNUM_SIGNED / PLL_DIVN2_FRACDEN
        = (860 + 343597380/2147483625)
dco_code = DIVN2 * PPM Step Size *1e-6
        = (860 + 343597380/2147483625) * 10 * 1e-6
        = 0.0086016
PLL_DCO_INT = floor (dco_code+0.5) = 0
PLL_DCO_FRAC = floor ((dco_code - PLL_DCO_INT) * PLL_DIVN2_FRACDEN)
        = floor ((0.0086016- 0) * 2147483625)
        = 18471795
```

Step 2: Device configuration

```
i2c.i2cw(0x69,0xff,0x00)       # Go to generic page
i2c.i2crmw(0x69,0x23,6,1,0)    # Configure the device to do DCO through register map
i2c.i2crmw(0x69,0x05,3,2,0)    # Select PLL-A for DCO
i2c.i2crmw(0x69,0x0f,2,1,0)    # Small trigger update
i2c.i2crmw(0x69,0x0f,2,1,1)
i2c.i2crmw(0x69,0x0f,2,1,0)
i2c.i2cw(0x69,0xff,0x0A)       # Go to the PLL-A page
i2c.i2cw(0x69,0x31,0x73)       # Update PLL_DCO_FRAC[7:0]
i2c.i2cw(0x69,0x32,0xdb)       # Update PLL_DCO_FRAC[8:15]
i2c.i2cw(0x69,0x33,0x19)       # Update PLL_DCO_FRAC[16:24]
i2c.i2cw(0x69,0x34,0x1)        # Update PLL_DCO_FRAC[25:32]
i2c.i2cw(0x69,0x36,0x0)        # Update PLL_DCO_INT[9:2]
i2c.i2crmw(0x69,0x37,7,2,0)    # Update PLL_DCO_INT[1:0]
i2c.i2crmw(0x69,0x37,5,1,1)    # Enable DCO for sync mode – Set the bit 0x37[5] on PLL page
```

Step 3: DCO increment and decrement

DCO increment operation

```
I2c.i2cw(0x69,0x35,0x00)
I2c.i2cw(0x69,0x35,0x08)        # DCO Increment operation
```

Add a delay with the minimum pulse width required as described in section Minimum pulse width, maximum update rate, DCO range and step size.

```
I2c.i2cw(0x69,0x35,0x00)
```

DCO decrement operation

```
I2c.i2cw(0x69,0x35,0x00)
I2c.i2cw(0x69,0x35,0x04)        # DCO decrement operation
```

Add a delay with the minimum pulse width required as described in section Minimum pulse width, maximum update rate, DCO range and step size.

```
I2c.i2cw(0x69,0x35,0x00)
```

Note:

- `i2c.i2cw` (device address, register address, data) – Python function used to write data to a register
- `i2c.i2crmw` (dev_addr,reg_addr,bit_loc,no_of_bits,hex_data) - Python function used to do a Read-Modify-Write in register
- See section Calculation of DCO fractional and integer code in sync mode for delay calculation.

6.2 DCO through pins (SiT9514x, sync mode profile)

Do a 10 ppm increment and decrement on PLL-A using the pin:

Step 1: DCO code calculation

Obtain the `PLL_DIVN2_INT, PLL_DIVN2_FRACNUM_SIGNED, PLL_DIVN2_FRACDEN` (from .nvm file):

```
DIVN2 = PLL_DIVN2_INT + (PLL_DIVN2_FRACNUM_SIGNED / PLL_DIVN2_FRACDEN
        = (860 + 343597380/2147483625)
dco_code = DIVN2 * PPM Step Size *1e-6
        = (860 + 343597380/2147483625) * 10 * 1e-6
        = 0.0086016
PLL_DCO_INT = floor (dco_code+0.5) = 0
PLL_DCO_FRAC = floor ((dco_code - PLL_DCO_INT) * PLL_DIVN2_FRACDEN)
        = floor ((0.0086016- 0) * 2147483625)
        = 18471795
```

Step 2: Device configuration

```
i2c.i2cw(0x69,0xff,0x00)          # Go to generic page
i2c.i2crmw(0x69,0x23,6,1,1)       # Configure the device to do DCO through pin
```

```
i2c.i2crmw(0x69,0x05,3,2,0)        # Select PLL-A for DCO
i2c.i2crmw(0x69,0x14,7,1,0)        # Set FDEC pin as input – Set bit 0x14[7] on generic page to 0
i2c.i2crmw(0x69,0x17,3,1,0)        # Set FINC pin as input – Set bit 0x17[3] on generic page to 0
i2c.i2crmw(0x69,0x0f,2,1,0)        # Small trigger update
i2c.i2crmw(0x69,0x0f,2,1,1)
i2c.i2crmw(0x69,0x0f,2,1,0)
i2c.i2cw(0x69,0xff,0x0A)           # Go to the PLL-A page
i2c.i2cw(0x69,0x31,0x73)           # Update PLL_DCO_FRAC [7:0]
i2c.i2cw(0x69,0x32,0xdb)           # Update PLL_DCO_FRAC[8:15]
i2c.i2cw(0x69,0x33,0x19)           # Update PLL_DCO_FRAC[16:24]
i2c.i2cw(0x69,0x34,0x1)            # Update PLL_DCO_FRAC[25:32]
i2c.i2cw(0x69,0x36,0x0)            # Update PLL_DCO_INT[9:2]
i2c.i2crmw(0x69,0x37,7,2,0)        # Update PLL_DCO_INT[1:0]
i2c.i2crmw(0x69,0x37,5,1,1)        # Enable DCO for sync mode – Set the bit 0x37[5] on PLL page
```

Step 3: DCO increment and decrement

- For DCO INCR, Toggle FINC pin in the following sequence:

  FINC = 0 (logic 0)

  FINC = 1 (logic 1) # Perform DCO increment operation from pin

  Add a delay with the minimum pulse width required as described in section Minimum pulse width, maximum update rate, DCO range and step size.

  FINC = 0 (logic 0)

- For DCO DECR, Toggle FDEC pin in the following sequence:

  FDEC = 0 (logic 0)

  FDEC = 1 (logic 1) # Perform DCO decrement operation from pin

  Add a delay with the minimum pulse width required, as described in section Minimum pulse width, maximum update rate, DCO range and step size.

  FDEC = 0 (logic 0)

Note:

- `i2c.i2cw(device address, register address, data)` – Python function used to write data to a register
- `i2c.i2crmw(dev_addr,reg_addr,bit_loc,no_of_bits,hex_data)` – Python function used to do a Read-Modify-Write in register

6.3 DCO through register map (SiT9514x, free-run mode profile)

Do a 10 ppm increment and decrement on PLL-A using the register map:

Step 1: DCO code calculation

Obtain the DIVN = PLL_DIVN_INT + PLL_DIVN_FRAC_SIGNED/2^32 from the nvm file

= 67+553573376/2^32

= 67.1288888454437255859375

```
dco_code = DIVN * 10 * 1e-6 * 2^32
    = 67.1288888454437255859375* 10 * 1e-6 *2^32
    = 2883163.822080
PLL_DCO_FRAC = floor (dco_code)
PLL_DCO_FRAC = 2883163
```

Step 2: Device configuration

```
i2c.i2cw(0x69,0xff,0x00)       # Go to generic page
i2c.i2crmw(0x69,0x23,6,1,0)    # Configure the device to do DCO through register map
i2c.i2crmw(0x69,0x05,3,2,0)    # Select PLL-A for DCO
i2c.i2crmw(0x69,0x0f,2,1,0)    # Small trigger update
i2c.i2crmw(0x69,0x0f,2,1,1)
i2c.i2crmw(0x69,0x0f,2,1,0)
i2c.i2cw(0x69,0xff,0x0A)       # Go to the PLL-A page
i2c.i2cw(0x69,0x31,0x5b)       # Update PLL_DCO_FRAC[7:0]
i2c.i2cw(0x69,0x32,0xfe)       # Update PLL_DCO_FRAC[8:15]
i2c.i2cw(0x69,0x33,0x2b)       # Update PLL_DCO_FRAC[16:24]
i2c.i2cw(0x69,0x34,0x00)       # Update PLL_DCO_FRAC[25:32]
i2c.i2crmw(0x69,0x35,1,1,1)    # Enable DCO for free-run mode – Set the bit 0x35[1] on PLL page
```

Step 3: DCO increment and decrement

DCO increment operation

```
I2c.i2crmw(0x69,0x35,3,1,0)
I2c.i2crmw(0x69,0x35,3,1,1)    # DCO increment operation
```

Add a delay with the minimum pulse width required as described in section Minimum pulse width, maximum update rate, DCO range and step size.

```
I2c.i2crmw(0x69,0x35,3,1,0)
```

DCO decrement operation

```
I2c.i2crmw(0x69,0x35,2,1,0)
```

```
I2c.i2crmw(0x69,0x35,2,1,1)      # DCO decrement operation
```

Add a delay with the minimum pulse width required as as described in section Minimum pulse width, maximum update rate, DCO range and step size.

```
I2c.i2crmw(0x69,0x35,2,1,0)
```

Note:

- `i2c.i2cw(device address, register address, data)` – Python function used to write data to a register
- `i2c.i2crmw(dev_addr,reg_addr,bit_loc,no_of_bits,hex_data)` – Python function used to do a read-modify-write in register
- Refer to the delay calculation instructions, as described in section Minimum pulse width, maximum update rate, DCO range and step size.

6.4 DCO through pins (SiT9514x, free-run mode profile)

Do a 10 ppm increment and decrement on PLL-A using the pin:

Step 1: DCO code calculation

Obtain the `DIVN = PLL_DIVN_INT + PLL_DIVN_FRAC_SIGNED/2^32` from the `nvm` file

```
        = 67+553573376/2^32
        = 67.128888845443725585937
dco_code = DIVN * 10 * 1e-6 * 2^32
        = 67.128888845443725585937* 10 * 1e-6 *2^32
        = 2883163.822080
PLL_DCO_FRAC = floor (dco_code)
PLL_DCO_FRAC = 2883163
```

Step 2: Device configuration

```
i2c.i2cw(0x69,0xff,0x00)        # Go to generic page
i2c.i2crmw(0x69,0x23,6,1,1)     # Configure the device to do DCO through pin
i2c.i2crmw(0x69,0x05,3,2,0)     # Select PLL-A for DCO
i2c.i2crmw(0x69,0x14,7,1,0)     # Set FDEC pin as input – Set bit 0x14[7] on generic page to 0
i2c.i2crmw(0x69,0x17,3,1,0)     # Set FINC pin as input – Set bit 0x17[3] on generic page to 0
i2c.i2crmw(0x69,0x0f,2,1,0)     # Small trigger update
i2c.i2crmw(0x69,0x0f,2,1,1)
i2c.i2crmw(0x69,0x0f,2,1,0)
i2c.i2cw(0x69,0xff,0x0A)        # Go to the PLL-A page
i2c.i2cw(0x69,0x31,0x5b)        # Update PLL_DCO_FRAC[7:0]
i2c.i2cw(0x69,0x32,0xfe)        # Update PLL_DCO_FRAC[8:15]
```

```
i2c.i2cw(0x69,0x33,0x2b)          # Update PLL_DCO_FRAC[16:24]
i2c.i2cw(0x69,0x34,0x00)          # Update PLL_DCO_FRAC[25:32]
i2c.i2crmw(0x69,0x35,1,1,1)       # Enable DCO for free-run mode – Set the bit 0x35[1] on PLL page
```

Step 3: DCO increment and decrement

For `DCO INCR`, Toggle FINC pin in the following sequence:

`FINC` = 0 (logic 0)

`FINC` = 1 (logic 1); # Perform DCO increment operation from pin

Add a delay with the minimum pulse width required as described in section Minimum pulse width, maximum update rate, DCO range and step size.

`FINC` = 0 (logic 0)

For `DCO DECR`, Toggle FDEC pin in the following sequence:

`FDEC` = 0 (logic 0)

`FDEC` = 1 (logic 1); # Perform DCO decrement operation from pin

Add a delay with the minimum pulse width required as described in section Minimum pulse width, maximum update rate, DCO range and step size.

`FDEC` = 0 (logic 0)

Note:

- `i2c.i2cw(device address, register address, data)` – Python function used to write data to a register
- `i2c.i2crmw(dev_addr, reg_addr,bit_loc,no_of_bits,hex_data)` – Python function used to do a read-modify-write in register

7 Annexure

Python functions:

- Function used to convert two's complement to signed integer:

```
def twos_comp_to_signed_int (DIVN_FRAC, WL_frac) # Convert from two's complement to signed integer for negative numbers
if (DIVN_FRAC >= 0.5*pow (2, WL_frac)): # Negative fraction
    divn_fraction = -(pow (2, WL_frac) - DIVN_FRAC)
elif (DIVN_FRAC < 0.5*pow (2, WL_frac)): # Positive fraction
    divn_fraction = DIVN_FRAC
return(divn_fraction)
```

- Function used to return the closest integer value which is less than or equal to given numeric value:
  `math. floor(number)` # python math library should be imported to include this function

8 Revision history

Table 10: Revision history of this document

Revision	Date	Description
1.0	07 Jul 2021	Initial release
1.1	20 Jan 2022	Minor layout changes only, no technical content was changed.

SiTime Corporation, 5451 Patrick Henry Drive, Santa Clara, CA 95054, USA | Phone: +1-408-328-4400 | Fax: +1-408-328-4439