I2C_MAS

REVISION HISTORY

Revision Number	Date	Description of Change	Author
V0.0	9/20/2022	Draft version	Su Aixue

Table of Contents

I2C_MAS

Introduction

The I2C bus is a simple, bidirectional two-wire synchronous serial bus. It requires only two wires to transfer information between devices connected to the bus.
The I2C master device is used to start the bus to transmit data and generate a clock to open the device for transmission. At this time, any addressed device is regarded as a slave device. The relationship between master and slave, sending and receiving on the bus is not constant, but depends on the direction of data transfer at this time. If the master device wants to send data to the slave device, the master device first addresses the slave device, then actively sends the data to the slave device, and finally the master device terminates the data transfer; if the master device wants to receive data from the slave device, the master device first addresses the slave device first, then the master device receives the data sent from the device, and finally the master device terminates the receiving process. In this case, the master device is responsible for generating the timing clock and terminating the data transfer.

The I2C_MAS module has the following features:
• Bidirectional two-wire synchronous serial bus;
• Support i2c master writing command;
• Output SCL and SDA_OUT according to I2C_MAS_EN and I2C_CTRL; (HWR001_I2C_MAS)
• Output RD_DATA and ACK_BIT to DS_REG according to SDA_IN. (HWR002_I2C_MAS)

Register Definition

Register Map

Table 11 I2C_MAS Register Map

ADDRESS	NAME	DESCRIPTION	RESET VALUE
I2C_MAS			
0x2200	**I2C_MAS_CTRL**	I2C_MAS control register	0x00
0x2201	**I2C_TR**	I2C send register	0x00
0x2202	**I2C_RD**	I2C receive register	0x00

I2C_MAS_CTRL

Register 1. I2C_MAS_CTRL (I2C_MAS control register, offset 0x000)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7	**STOP**	R/W	1'b0	Stop Command 0: Ready 1: Execute
6	**RX**	R/W	1'b0	Receive Command 0: Ready 1: Execute

5	**SR**	R/W	1'b0	Restart Command 0: Ready 1: Execute
4	**ACK**	R	1'b0	Acknowledge from Slave device 0: NACK 1: ACK
3:1	**REV**	R	3'b0	Reserved
0	**TX**	R/W	1'b0	Send Command 0: Ready 1: Execute

I2C_TR

Register 2. I2C TR (I2C send register, offset 0x001)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:0	**DATA**	R/W	8'h00	Data Sent

I2C_RD

Register 3. I2C_RD (I2C receive register, offset 0x002)

BIT	NAME	ACCESS	RESET	DESCRIPTION
7:0	**DATA**	R/W	8'h00	Data Received

Function Details

Block Diagram

The main elements of I2C_MAS and their interactions are shown in Fig 1.

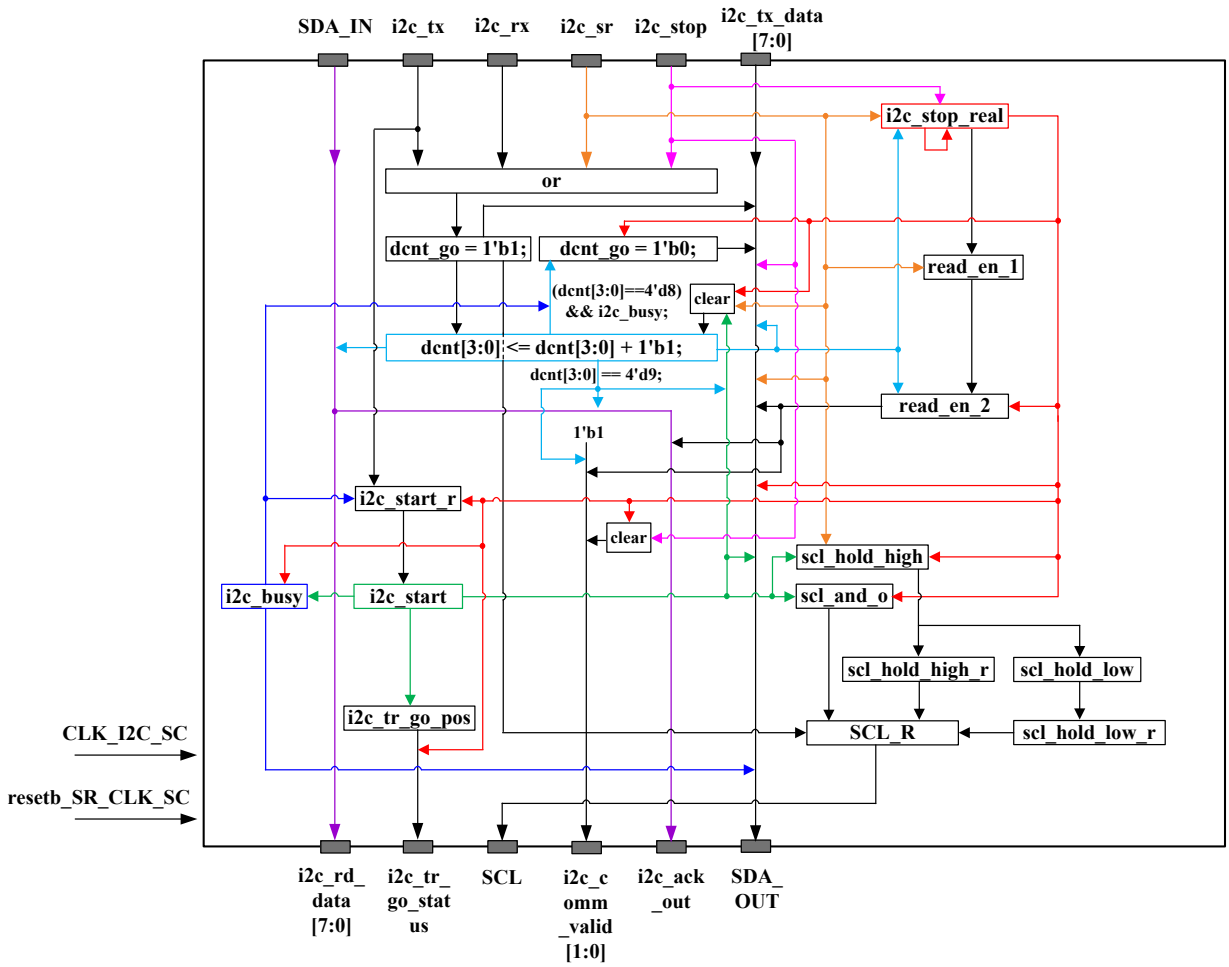

Fig 1. I2C_MAS Block Diagram

I2C_MAS IO Descriptions

This section provides the I2C_MAS IO descriptions.

Table 2 I2C_MAS IO descriptions

Signal	Width	Duration	I/O	Default Value	Register	Description
CLK_I2C_SC	1	--	I	--	--	400kHz
resetb_SR_CLK_I2C	1	--	I	--	--	--
SCL	1	--	O	1'b0	--	--
SDA_OUT	1	--	O	1'b0	--	--
SDA_IN	1	--	I	--	--	--
i2c_tx	1	1~2 CLK_I2C_SC_DIV4	I	--	TX	8us pulse
i2c_rx	1	1~2 CLK_I2C_SC_DIV4	I	--	RX	8us pulse
i2c_sr	1	1~2 CLK_I2C_SC_DIV4	I	--	SR	8us pulse
i2c_stop	1	1~2 CLK_I2C_SC_DIV4	I	--	STOP	8us pulse
i2c_tx_data	8	--	I	--	I2C TR	--
i2c_tr_go_status	1	--	O	1'b1	--	--

i2c_comm_valid	2	2036 CLK_I2C_SC	O	2'b0	--	--
i2c_ack_out	1	--	O	1'b0	ACK	--
i2c_rd_data	8	--	O	8'b0	I2C_RD	--

I2C Key Signal Descriptions

Table 3 I2C key signal descriptions

Signal	Width	Duration	Default Value	Description
div_cnt	4	1 CLK_I2C_SC	4'd0	It is used to count the posedge of CLK_I2C_SC from 0 to 3 circularly. The encoding format is gray code.
pulse_i2c_div4_0	1	1 CLK_I2C_SC	1'b1	It is equal to "(div_cnt == 'h0)".
pulse_i2c_div4_1	1	1 CLK_I2C_SC	1'b0	It is equal to "(div_cnt == 'h1)".
pulse_i2c_div4_2	1	1 CLK_I2C_SC	1'b0	It is equal to "(div_cnt == 'h3)".
pulse_i2c_div4_3	1	1 CLK_I2C_SC	1'b0	It is equal to "(div_cnt == 'h2)".
CLK_I2C_SC_DIV4	1	2 CLK_I2C_SC	1'b0	It is high level when "(div_cnt == 'h1) || (div_cnt == 'h3)"; and it is low level when "(div_cnt == 'h0) || (div_cnt == 'h2)". Thus, the period of this signal is two times the period of CLK_I2C_SC.
i2c_stop_r	1	52 CLK_I2C_SC	1'b0	It is a level signal between i2c_stop_sync (i2c_stop after synchronization) and "i2c_stop_rr_ddd & i2c_stop_real_dd". In last words, i2c_stop_rr_ddd is generated by delaying i2c_stop_rr three clock cycles, i2c_stop_real_dd is generated by delaying i2c_stop two clock cycles.
i2c_stop_rr	1	12 CLK_I2C_SC	1'b0	It is a level signal between "i2c_stop_r & (dcnt == 9)" and "i2c_stop_rr_ddd & i2c_stop_real_dd". In last words, i2c_stop_rr_ddd is generated by delaying i2c_stop_rr three clock cycles, i2c_stop_real_dd is generated by delaying i2c_stop two clock cycles.
i2c_stop_real	1	8 CLK_I2C_SC/ 5 CLK_I2C_SC	1'b0	The value of this signal is equal to "(i2c_stop_rr_d & (~i2c_stop_rr_ddd) | (i2c_stop_sync & ~i2c_sr_r))", in which, the i2c_stop_rr_d is generated by delaying i2c_stop_rr one clock cycle, the i2c_stop_rr_ddd is generated by delaying i2c_stop_rr three clock cycles, i2c_stop_sync is after synchronization of i2c_stop. This signal is used to note the stop information after the last read data that behind the i2c_stop pulse.
i2c_stop_real_d	1	8 CLK_I2C_SC/ 4 CLK_I2C_SC	1'b0	It is the signal obtained by signal i2c_stop_real through a trigger. Clock pins of the trigger is connected to CLK_I2C_SC_DIV4.
i2c_stop_real_dd	1	8 CLK_I2C_SC/ 4 CLK_I2C_SC	1'b0	It is the signal obtained by signal i2c_stop_real through two triggers. Clock pins of two triggers are connected to CLK_I2C_SC_DIV4.
i2c_sr_r	1	--	1'b0	It is a level signal between i2c_sr_sync (i2c_sr after synchronization) and "i2c_stop_rr_ddd & i2c_stop_real_dd". In last words, i2c_stop_rr_ddd is generated by delaying i2c_stop_rr three clock cycles, i2c_stop_real_dd is generated by delaying i2c_stop two clock cycles.
i2c_sr_sync	1	5 CLK_I2C_SC	1'b0	It is the signal obtained by signal i2c_sr through a trigger. Clock pins of the trigger is connected to

				CLK_I2C_SC.
i2c_sr_d[0]	1	--	1'b0	It is the signal obtained by signal i2c_sr_sync through a trigger. Clock pins of the trigger is connected to CLK_I2C_SC_DIV4.
i2c_sr_d[1]	1	--	1'b0	It is the signal obtained by signal i2c_sr_sync through two triggers. Clock pins of two triggers are connected to CLK_I2C_SC_DIV4.
i2c_sr_d[2]	1	--	1'b0	It is the signal obtained by signal i2c_sr_sync through three triggers. Clock pins of three triggers are connected to CLK_I2C_SC_DIV4.
i2c_start_r	1	812 CLK_I2C_SC/ 800 CLK_I2C_SC	1'b0	It is a level signal between "(~i2c_busy) & i2c_tx_sync" and i2c_stop_real, in which, i2c_tx_sync is after synchronization of i2c_tx.
i2c_start	1	8 CLK_I2C_SC	1'b0	It is a pulse signal generated by detecting the rising edge of i2c_start_r. It is used to start the communication.
i2c_start_d	1	8 CLK_I2C_SC	1'b0	It is the signal obtained by signal i2c_start through a trigger. Clock pins of the trigger is connected to CLK_I2C_SC_DIV4.
i2c_start_dd	1	8 CLK_I2C_SC	1'b0	It is the signal obtained by signal i2c_start through two triggers. Clock pins of two triggers are connected to CLK_I2C_SC_DIV4.
i2c_busy	1	804 CLK_I2C_SC/ 792 CLK_I2C_SC	1'b0	It is a level signal between i2c_start_d and i2c_stop_real. In last words, i2c_start_d is generated by delaying i2c_start one clock cycle.
i2c_rx_ex	1	5 CLK_I2C_SC	1'b0	The value of this signal is equal to "i2c_rx_sync | i2c_stop_sync". In last words, the i2c_rx_sync is after synchronization of i2c_rx, i2c_stop_sync is after synchronization of i2c_stop.
dcnt_go	1	36 CLK_I2C_SC/ 44 CLK_I2C_SC/ 48 CLK_I2C_SC	1'b0	It is a level signal between "i2c_tx_sync || i2c_sr_sync || i2c_rx_ex" and "(dcnt == 8) & i2c_busy". In last words, the i2c_tx_sync is after synchronization of i2c_tx, i2c_sr_sync is after synchronization of i2c_sr. It is used to enable the internal counter.
dcnt_go_d	1	36 CLK_I2C_SC/ 44 CLK_I2C_SC/ 48 CLK_I2C_SC	1'b0	It is the signal obtained by signal dcnt_go through a trigger. The clock pin of the trigger is connected to CLK_I2C_SC_DIV4.
dcnt	4	4 CLK_I2C_SC	4'h0	This signal is set to 0 when "i2c_start || i2c_stop_real || i2c_sr_sync || i2c_sr_d[0] || i2c_sr_d[1] = 1" or "dcnt == 9". In last words, the i2c_sr_sync is after synchronization of i2c_sr, the i2c_sr_d[0] is generated by delaying i2c_sr one clock cycle. I2c_sr_d[1] is generated by delaying i2c_sr two clock cycles. If the above condition is not established and dcnt_go = 1, this signal is increased by 1 every clock. After sampling the SDA_IN (input signal), the value of this signal is used as the note of clock cycles for putting the sample values into corresponding bit of i2c_rx_data (output signal) and i2c_ack_out (output signal).
read_en_1	1	--	1'b0	It is a level signal between i2c_sr_d[0] and i2c_stop_real. In last words, i2c_sr_d[0] is generated by delaying i2c_sr one clock cycle.
read_en_2	1	448 CLK_I2C_SC	1'b0	It is a level signal between "read_en_1 & (dcnt == 8)" and i2c_stop_real. It is used to enable the operation of reading data.
read_en_2_dddd	1	448 CLK_I2C_SC	1'b0	It is the signal obtained by signal read_en_2 through four triggers. The clock pins of four triggers are

				connected to CLK_I2C_SC_DIV4.

I2C Communication Formats

The I2C communication timing diagram is shown in Fig 2.

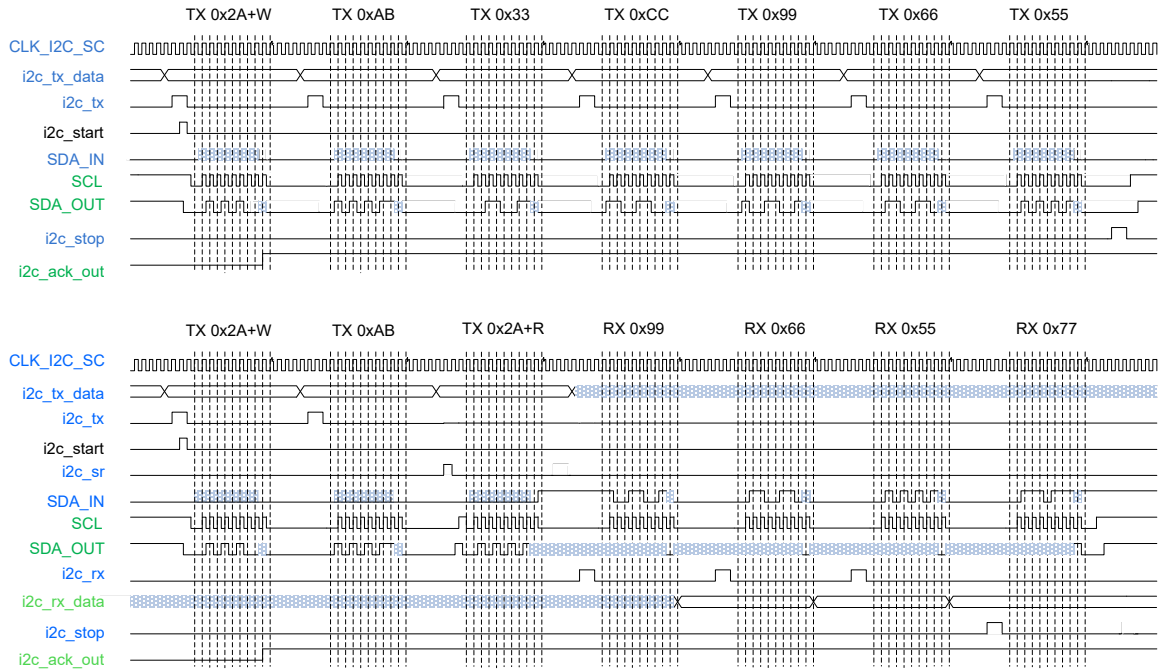

Fig 2. I2C Communication Timing Diagram

The I2C data format is shown in Fig 3.

Data writed from master to slave

S	SA	W	A	D	A	D	A	…	D	A/$\overline{A}$	P
1 bit	9 bits			9 bits		9 bits		…	9 bits		1 bit

Data read from salve to master

S	SA	$\overline{W}$	A	D	A	Sr	SA	R	A	D	A	D	A	…	D	$\overline{A}$	P
1 bit	9 bits			9 bits		1 bit	9 bits			9 bits		9 bits		…	9 bits		1 bit

Fig 3. I2C Data Format

NOTE:
S – Start bit. (1 bit)
SA – Address of slave device. (7 bits)
$\overline{W}$ – The flag bit of writing, 1'b0. (1 bit)
R – The flag bit of reading, 1'b1. (1 bit)
A – Response bit, 1'b0. (1 bit)
$\overline{A}$ – Non-response bit, 1'b1. (1 bit)
D – Data bits. (8 bits)
P – The flag bit of stopping. (1 bit)

– The data direction is from master device to slave device.

– The data direction is from slave device to master device.

I2C_MAS Function Descriptions

The I2C_MAS module has two functions:
• Output SCL and SDA_OUT according to I2C_MAS_EN and I2C_CTRL; (Func 1 & Func 2) (HWR001_I2C_MAS)
• Output RD_DATA and ACK_BIT to DS_REG according to SDA_IN. (Func 3 & Func4) (HWR002_I2C_MAS)

Above functions can be found in the following timing diagrams.
Func 1: Signal SCL is the signal obtained by signal SCL_R through a trigger. The clock pin of the trigger is connected to CLK_I2C_SC. Signal SCL_R is equal to "(((scl_and_o | scl_and_o_d) & dcnt_go_d & CLK_I2C_SC_DIV4) | scl_hold_high_r) & (~scl_hold_low_r)". The detailed description is as follows:

(1) Signal dcnt_go_d, CLK_I2C_SC_DIV4 can refer to Table 3.
(2) Signal scl_and_o will become to high level when the high level of signal i2c_start_dd is detected using the posedge of CLK_I2C_SC_DIV4. If the above condition is not satisfied, signal scl_and_o will become to low level when the high level of signal i2c_stop_real is sampled using the posedge of CLK_I2C_SC_DIV4. If the above two conditions are not satisfied, the scl_and_o remains unchanged.
Note: Signal i2c_start_dd and i2c_stop_real can refer to Table 3.
(3) Signal scl_and_o_d is the signal obtained by signal scl_and_o through a trigger. The clock pin of the trigger is connected to CLK_I2C_SC_DIV4.
(4) Signal scl_hold_high_r is the signal obtained by signal scl_hold_high through a trigger. The clock pin of the trigger is connected to CLK_I2C_SC. Signal scl_hold_high will become to high level when the high level of signal "i2c_stop_real_d || i2c_sr_d[0]" is detected using CLK_I2C_SC. If the above condition is not satisfied, signal scl_hold_high will become to low level when the high level of signal "pulse_i2c_div4_2 && (i2c_start_d || i2c_sr_d[1])". If the above two conditions are not satisfied, the scl_hold_high remains unchanged.
Note: Signal i2c_stop_real_d, i2c_sr_d, pulse_i2c_div4_2 and i2c_start_d can refer to Table 3.
(5) Signal scl_hold_low_r is the signal obtained by signal scl_hold_low through a trigger. The clock pin of the trigger is connected to CLK_I2C_SC. Signal scl_hold_low is equal to "(~scl_hold_high) & scl_hold_high_d". Signal scl_hold_high_d is the signal obtained by signal "scl_hold_high & read_en_1" through a trigger. The clock pin of the trigger is connected to "~CLK_I2C_SC_DIV4".
Note: Signal read_en_1 can refer to Table 3.

Func 2: Signal SDA_OUT is the signal obtained by signal sda_out_reg through two triggers. The clock pins of two triggers are connected to CLK_I2C_SC. The generation of signal sda_out_reg can refer to Fig 4. And the signal in Fig 4 can refer to Table 3.

```
always@(posedge CLK_I2C_SC or negedge resetb_SR_CLK_I2C)
begin
    if(!resetb_SR_CLK_I2C)
        sda_out_reg <= 1'b1;
    else if(pulse_i2c_div4_2 & i2c_stop_real_dd)
        sda_out_reg <= 1'b1;
    else if(i2c_sr_d[1])
        sda_out_reg <= 1'b0;
    else if(i2c_sr_sync)
        sda_out_reg <= 1'b1;
    else if(pulse_i2c_div4_2 & i2c_start /*|| i2c_start_d*/)
        sda_out_reg <= 1'b0;
    else if(pulse_i2c_div4_2 & ((~read_en_2_dddd) & i2c_busy & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h0)))
        sda_out_reg <= send_data[7];
    else if(pulse_i2c_div4_2 & ((~read_en_2_dddd) & i2c_busy & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h1)))
        sda_out_reg <= send_data[6];
    else if(pulse_i2c_div4_2 & ((~read_en_2_dddd) & i2c_busy & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h2)))
        sda_out_reg <= send_data[5];
    else if(pulse_i2c_div4_2 & ((~read_en_2_dddd) & i2c_busy & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h3)))
        sda_out_reg <= send_data[4];
    else if(pulse_i2c_div4_2 & ((~read_en_2_dddd) & i2c_busy & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h4)))
        sda_out_reg <= send_data[3];
    else if(pulse_i2c_div4_2 & ((~read_en_2_dddd) & i2c_busy & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h5)))
        sda_out_reg <= send_data[2];
    else if(pulse_i2c_div4_2 & ((~read_en_2_dddd) & i2c_busy & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h6)))
        sda_out_reg <= send_data[1];
    else if(pulse_i2c_div4_2 & ((~read_en_2_dddd) & i2c_busy & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h7)))
        sda_out_reg <= send_data[0];
    else if(pulse_i2c_div4_2 & ((~read_en_2_dddd) & i2c_busy & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h8)))
        sda_out_reg <= 1'b1;
    else if(pulse_i2c_div4_2 & ((~read_en_2_dddd) & i2c_busy & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h9)))
        sda_out_reg <= 1'b0;
    else if(pulse_i2c_div4_2 & (read_en_2_dddd & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h0)))
        sda_out_reg <= 1'b1;
    else if(pulse_i2c_div4_2 & (read_en_2_dddd & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h1)))
        sda_out_reg <= 1'b1;
    else if(pulse_i2c_div4_2 & (read_en_2_dddd & (dcnt_qo | dcnt_qo_d) & (dcnt[3.0]==4'h2)))
        sda_out_reg <= 1'b1;
    else if(pulse_i2c_div4_2 & (read_en_2_dddd & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h3)))
        sda_out_reg <= 1'b1;
    else if(pulse_i2c_div4_2 & (read_en_2_dddd & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h4)))
        sda_out_reg <= 1'b1;
    else if(pulse_i2c_div4_2 & (read_en_2_dddd & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h5)))
        sda_out_reg <= 1'b1;
    else if(pulse_i2c_div4_2 & (read_en_2_dddd & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h6)))
        sda_out_reg <= 1'b1;
    else if(pulse_i2c_div4_2 & (read_en_2_dddd & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h7)))
        sda_out_reg <= 1'b1;
    else if(pulse_i2c_div4_2 & (read_en_2_dddd & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h8)))
            if(i2c_stop_r)
                sda_out_reg <= 1'b1;
            else
                sda_out_reg <= 1'b0;
    else if(pulse_i2c_div4_2 & (read_en_2_dddd & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h9) & (~i2c_stop_r)))
        sda_out_reg <= 1'b1;
    else if(pulse_i2c_div4_2 & (read_en_2_dddd & (dcnt_qo | dcnt_qo_d) & (dcnt[3:0]==4'h9) & i2c_stop_r))
        sda_out_reg <= 1'b0;
    else if(pulse_i2c_div4_2 & i2c_stop_rr_ddd)
        sda_out_reg <= 1'b1;
end
```

Fig 4. Generation of signal sda_out_reg

Func 3: i2c_rx_data[i] (RD_DATA[i], i=0, 1, 2, …, 7) is the signal obtained by signal i2c_rx_data_r[i] through a trigger. The clock pins of the triggers are connected to CLK_I2C_SC. While, the generation of i2c_rx_data_r can refer to Fig 5. And the signal in Fig 5 can refer to Table 3.

```
always@(posedge CLK_I2C_SC or negedge resetb_SR_CLK_I2C)
begin
    if(!resetb_SR_CLK_I2C)
        i2c_rx_data_r[7:0] <= 8'h0;//zz;
    else if(pulse_i2c_div4_1 & (read_en_2_dddd & dcnt_qo))
        begin
            if(dcnt[3:0] == 4'd1)
                i2c_rx_data_r[7] <= SDA_IN;
            else if(dcnt[3:0] == 4'd2)
                i2c_rx_data_r[6] <= SDA_IN;
            else if(dcnt[3:0] == 4'd3)
                i2c_rx_data_r[5] <= SDA_IN;
            else if(dcnt[3:0] == 4'd4)
                i2c_rx_data_r[4] <= SDA_IN;
            else if(dcnt[3:0] == 4'd5)
                i2c_rx_data_r[3] <= SDA_IN;
            else if(dcnt[3:0] == 4'd6)
                i2c_rx_data_r[2] <= SDA_IN;
            else if(dcnt[3:0] == 4'd7)
                i2c_rx_data_r[1] <= SDA_IN;
            else if(dcnt[3:0] == 4'd8)
                i2c_rx_data_r[0] <= SDA_IN;
          //else
          //    i2c_rx_data_r[7:0] <= 'h0;//zz;
        end
end
```

Fig 5. Generation of signal i2c_rx_data_r

Func 4: i2c_ack_out (ACK_BIT) is the signal obtained by signal i2c_ack_out_r through a trigger. The clock pins of the trigger is connected to CLK_I2C_SC. While, i2c_ack_out_r will become to "~SDA_IN"

when the high level of "(~read_en_2_dddd) && (dcnt[3:0] == 4'h9)" is detected using the posedge of CLK_I2C_SC_DIV4. And SDA_IN is an input signal, signal read_en_2_dddd and dcnt can refer to Table 3.

The result of implement is as following figures.

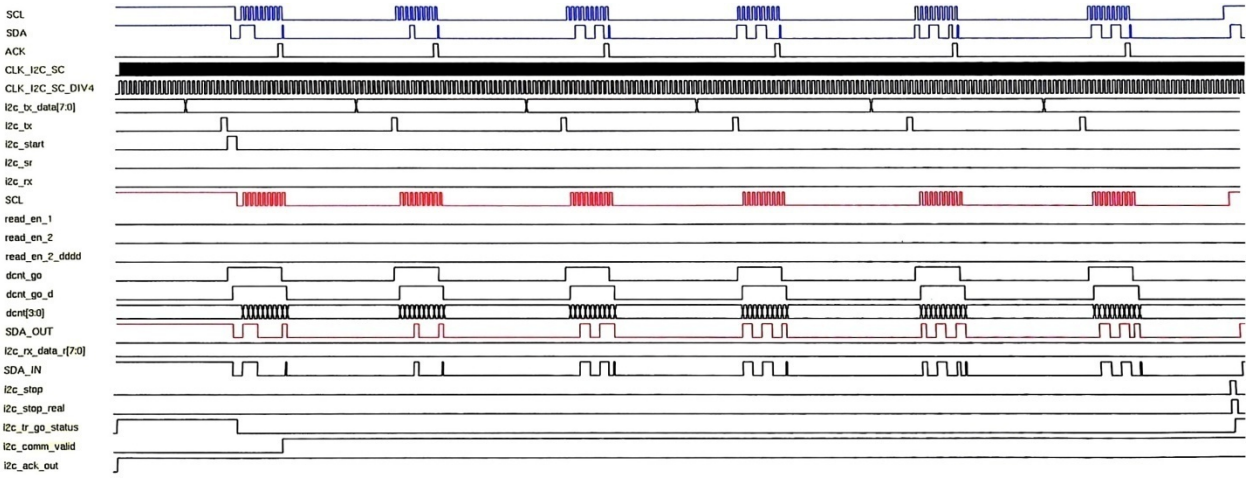

Fig 6. OVUV_OTUT_CMP Timing Diagram 1

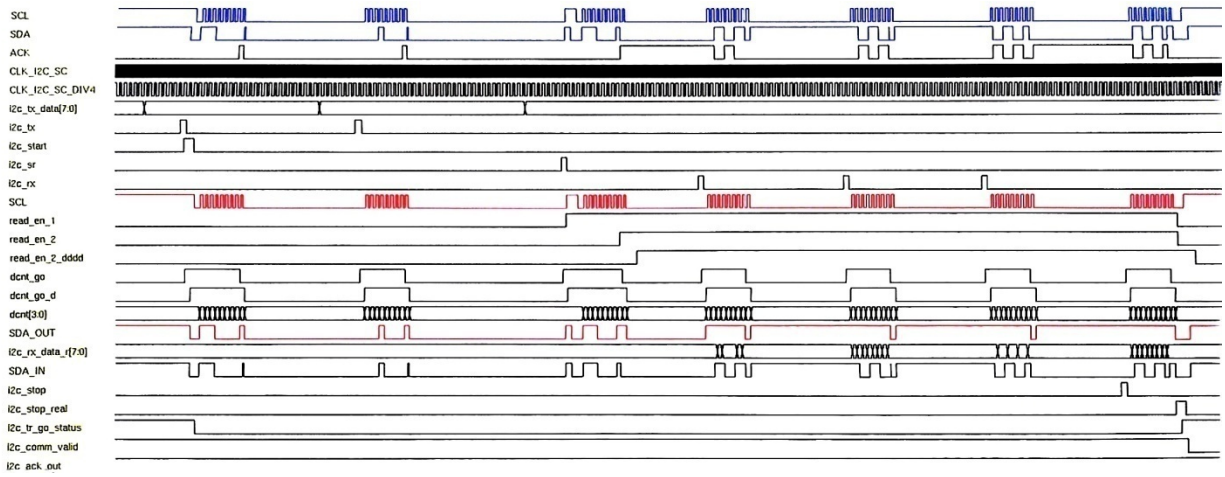

Fig 7. OVUV_OTUT_CMP Timing Diagram 2

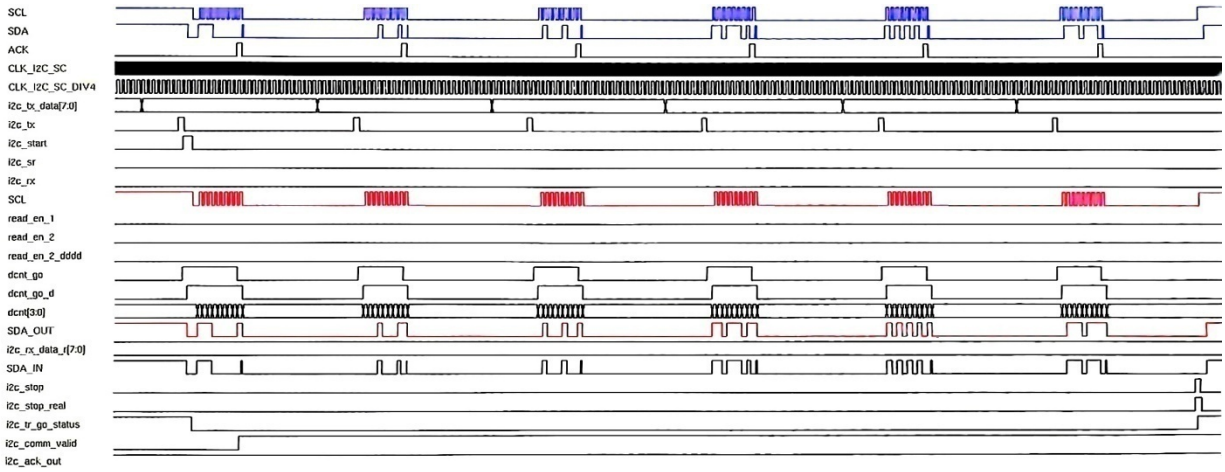

Fig 8. OVUV_OTUT_CMP Timing Diagram 3

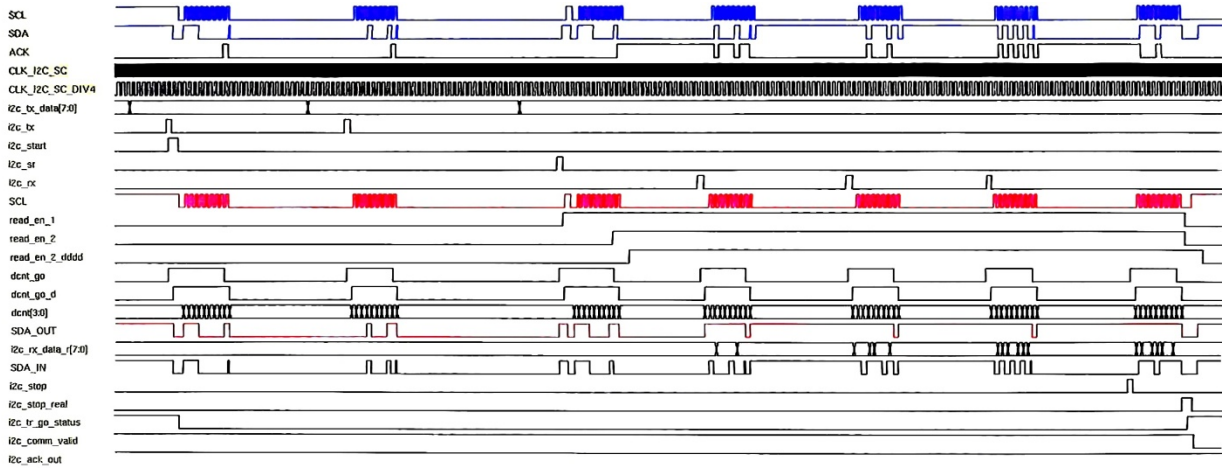

Fig 9. OVUV_OTUT_CMP Timing Diagram 4