RTL9310

I2C

Application Note

Rev. 1.0
27 Apr 2018

Realtek Semiconductor Corp.
No. 2, Innovation Road II, Hsinchu Science Park, Hsinchu 300, Taiwan
Tel.: +886-3-578-0211 Fax: +886-3-577-6047
www.realtek.com

USING THIS DOCUMENT
This document is intended for use by the system engineer when integrating with Realtek switch products. Though every effort has been made to assure that this document is current and accurate, more information may have become available subsequent to the production of this guide. In that event, please contact your Realtek representative for additional information that may help in the development process.

Revision	Release Date	Summary
1.0	2018/4/27	Initial

Contents

Table List

Figure List

1 Overview

RTL9310 supports two I2C masters:

I2C masters	Each master has one SCL pin respectively;
	The two masters share with 12 SDA pins.

2 I2C Master

RTL9310 supports I2C master mode to access the other I2C slave devices such as PoE controller and fiber module. For I2C master mode, it needs to configure GPIO pins as I2C interface first.

GPIO13~GPIO26 can be reused as master I2C interface. When GPIO13~GPIO26 are reused as master I2C interface, it needs to configure related registers for I2C usage.

Figure 2-1 and Figure 2-2 explain the multiplex relation.

Figure 2-1 The Multiplex of Master I2C(1)

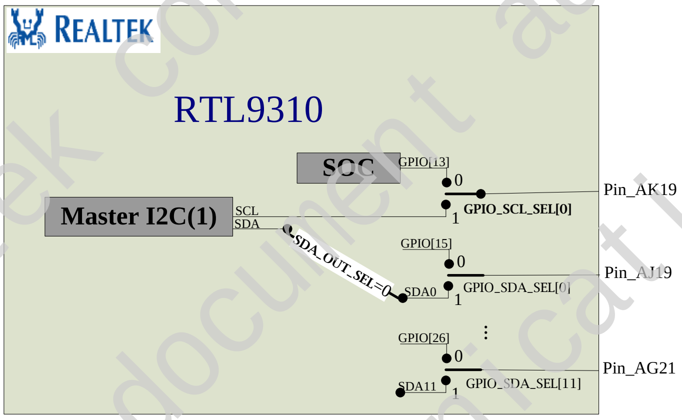
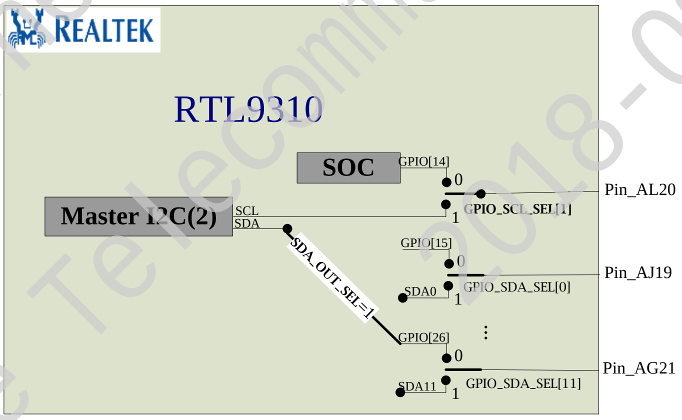


Figure 2-2 The Multiplex of Master I2C(2)



According to the actual requirement, users can choose their own solution.

For one example, in the 48G+6*10G model:

- I2C Master1 to control 6 fiber module(GPIO13 as SCL, GPIO15-20 as SDA0-5);
- I2C Master2 to connect 6 PoE controllers to control 48 downlink ports(GPIO14 as SCL, GPIO21 as SDA6);

For another example, in the 12*10G model:

- I2C Master1 to control 12 fiber module(GPIO13 as SCL, GPIO15-26 as SDA0-11);

ASIC provides lots of register to control the I2C master behavior which is introduced below.

GPIO Mode Select

The SCL0 pin is reused with GPIO13, the SCL1 pin is reused with GPIO14, and the SDA0~SDA11 pins are reused with GPIO15~GPIO26.

SDA Out Select

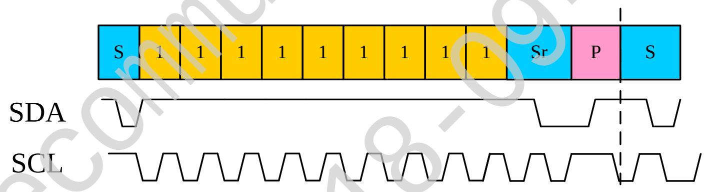
SDA_OUT_SEL is used to select which one of the 12 SDA pins to output data. Please note that master I2C controller 1 has higher priority. That means, if *SDA_OUT_SEL* of master I2C controller 1 is equal to *SDA_OUT_SEL* of master I2C controller 2, the SDA pin will be occupied by master I2C controller 1 after arbitration.

I2C Reset

I2C_RESET can configure whether adding I2C reset command before each read/write operation or not. It is usually used to access EEPROM, which supports reset command.

- I2C Reset Steps:
  - Create a start bit condition
  - Clock 9 cycles
  - Sr…
  - Create another start bit followed by stop bit condition as shown below.

Figure 2-3 Protocol Reset Condition



Please refer to I2C slave device datasheet to decide whether adding I2C Reset or not.

Signal Delay Configure

DRIVE_ACK_DELAY can set the delay time of master I2C controller driving ACK signal.
CHECK_ACK_DELAY can set the delay time of master I2C controller checking ACK signal.

I2C Read Standard/Old Mode Support

I2C_RD_MODE can select in standard mode or old mode.

There are two differences between standard mode and old mode. (Refer to Figure 2-3 and Figure 2-4.)

- Read operation
  - Standard mode
    - write command, memory address, read command and read data
  - Old mode
    - read command, memory address, read data.
- Sequence of memory address
  - Standard mode
    - *MEM_ADDR* [23:16], *MEM_ADDR* [15:8], and *MEM_ADDR* [7:0]
  - Old mode
    - *MEM_ADDR* [7:0], *MEM_ADDR* [15:8], and *MEM_ADDR* [23:16]

Figure 2-3 Master I2C Read Data Timing Waveform (Standard Mode)

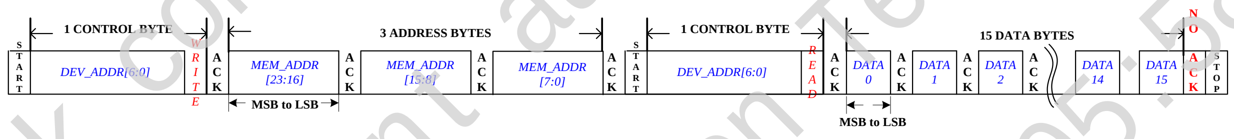
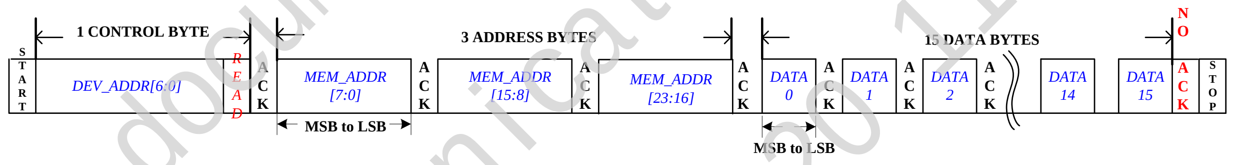


Figure 2-4 Master I2C Read Data Timing Waveform (Old Mode)



Clock Frequency Select

Master I2C controllers support four frequencies (50KHz ,100KHz, 400KHz and 2.5MHz) of SCL decided by *SCL_FREQ*.

SCL&SDA Output Mode Select

I2C_OPEN_DRN_SCL and *I2C_OPEN_DRN_SDA* can set the SCL pin and SDA pin in drive mode or open drain mode.

SDA Hold Time Configure

CFG_DATA_HOLD_TIME can set the hold time of SDA when master I2C controller output data.

SCL Input Delay Configure

CFG_SCL_I_DLY can set the delay time of data sampling when master I2C controller input data.

Waiting SCL Mode Select

CFG_WAIT_SCL_MODE is used to select which mode the controller can support, and the mode is to judge whether the SCL is released by slave device. Please refer to Table 2-1(*I2C_MST_IF_CTRL*).

2.1 Register Setting

Table 2-1 I2C_MST_IF_CTRL Register

Bits	Field	Description	Type	Default
31:28	RESERVED			
27	CFG_DATA_HOLD_TIME_2	Select data hold time for I2C Master#2 controller. 0b0: 100ns 0b1: T/4 (T is SCL period)	RW	0x0
26	CFG_DATA_HOLD_TIME_1	Select data hold time for I2C Master#1 controller. 0b0: 100ns 0b1: T/4 (T is SCL period)	RW	0x0
25:22	CFG_SCL_I_DLY_2	Configure SCL input delay for I2C Master#2 controller. Use 250MHz to delay, 1 unit = 4ns. 0x0: no delay 0x1_0xF: 1_15 unit delay	RW	0x0
21:18	CFG_SCL_I_DLY_1	Configure SCL input delay for I2C Master#1 controller. Use 250MHz to delay, 1 unit = 4ns. 0x0: no delay 0x1_0xF: 1_15 unit delay	RW	0x0
17:16	CFG_WAIT_SCL_MODE_2	For I2C Master2 controller. 0x0: do not wait for SCL=1 or not 0x1: wait for SCL=1 after ACK, and then launch the next command 0x2: wait for SCL=1 after each SCL cycle, and then launch the next command 0x3: reserved	RW	0x0
15:14	CFG_WAIT_SCL_MODE_1	For I2C Master1 controller. 0x0: do not wait for SCL=1 or not 0x1: wait for SCL=1 after ACK, and then launch the next command 0x2: wait for SCL=1 after each SCL cycle, and then launch the next command 0x3: reserved	RW	0x0
13	I2C_OPEN_DRN_SCL_2	For SCL of I2C Maseter2 controller 0b0: always output enable 0b1: use open-drain style	RW	0x0

12	I2C_OPEN_DRN_SCL_1	For SCL of I2C Maseter1 controller 0b0: always output enable 0b1: use open-drain style	RW	0x0
11:0	I2C_OPEN_DRN_SDA	BitMask control for SDA11_SDA0 0b0: always output enable 0b1: use open-drain style	RW	0x0

Table 2-2 I2C_MST_IF_SEL Register

Bits	Field	Description	Type	Default
31:14	RESERVED			
13:12	GPIO_SCL_SEL	Bit mask[1:0] for selecting GPIO14_GPIO13 works as GPIO or SCL. 0b0: GPIO 0b1: SCL Note: 1. GPIO13 is mapping to SCL0 of controller#1. 2. GPIO14 is mapping to SCL1 of controller#2.	RW	0x0
11:0	GPIO_SDA_SEL	Bit mask[11:0] for selecting GPIO26_GPIO15 work as GPIO or SDA. 0b0: GPIO 0b1: SDA Note: GPIO15_GPIO26 is mapping to SDA0_SDA11.	RW	0x0

Table 2-3 I2C_MST1_CTRL/ I2C_MST2_CTRL Register

Bits	Field	Description	Type	Default
31:30	SCL_FREQ	SCL clock rate selection for I2C master mode. 0x0: 50KHz 0x1: 100KHz 0x2: 400KHz 0x3: 2.5MHz	RW	0x0
29:26	CHK_ACK_DLY	Delay time for I2C Master to check the replying ACK of slave device. ACK check delay = Check_ACK_Delay*unit 1 unit = 1/(SCL_FREQ) 0 means no delay, max delay time is 15 clock cycle.	RW	0x0
25:22	DRV_ACK_DLY	Delay time for I2C Master to driving ACK bit. Delay time = Drive_ACK_Delay *unit 1 unit = 1/(SCL_FREQ) 0 means no delay, max delay time is 15 clock cycle.	RW	0x0

21:18	SDA_OUT_SEL	Select SDA output from SDA0_SDA11. 0x0: select SDA0 as SDA output. 0x1: select SDA1 as SDA output. 0x2: select SDA2 as SDA output 0x3: select SDA3 as SDA output 0x4: select SDA4 as SDA output 0x5: select SDA5 as SDA output 0x6: select SDA6 as SDA output 0x7: select SDA7 as SDA output 0x8: select SDA8 as SDA output 0x9: select SDA9 as SDA output 0xA: select SDA10 as SDA output 0xB: select SDA11 as SDA output Others: Reserved. Note: before setting SDA_OUT_SEL, CPU should first set GPIO pin as SDA function; otherwise, SDA_OUT_SEL will not work as expected; if SDA[x] is not selected as SDA output, it should output a high-impedance to avoid any mistake.	RW	0x0
17:11	DEV_ADDR	Select device type id & address of external I2C slave device. Note: SFP/SFP+ device address is fixed at 7'b1010000	RW	0x50
10:9	MEM_ADDR_WIDTH	Select memory address width of external I2C slave device. 0b0: 0-byte 0b1: 1-byte 0b2: 2-byte 0b3: 3-byte	RW	0x1
8:5	DATA_WIDTH	Configure I2C read/write data width=DATA_WIDTH + 1. 0x0: 1-byte 0x1: 2-byte 0x2: 3-byte : 0xF: 16-byte	RW	0x0
4	READ_MODE	read command mode. 0b0: standard mode. 0b1: old mode. Note: 1. Standard mode read command is Write + Address + Read+ Data 2. Old mode read command is Read + Address + Data	RW	0x0
3	RST	I2C Reset Command before each start command. 0b0: do not add I2C reset command before each start command 0b1: add I2C reset command before each start command	RW	0x0
2	RWOP	Read/write operation. 0b0: read 0b1: write	RW	0x0

1	FAIL	I2C indirect access fail. 0b0: normal 0b1: fail, self cleared at each I2C trigger operation. Note: 1. Access fail means the slave device has no ack response while access.	RO	0x0
0	TRIG	Trigger I2C master command. 0b0: complete access. 0b1: execute access. Note: self clear to 0 after complete access.	RWAC	0x0

Table 2-4 I2C_MST1_MEMADDR_CTRL/ I2C_MST2_MEMADDR_CTRL Register

Bits	Field	Description	Type	Default
31:24	RESERVED			
23:0	MEM_ADDR	Select memory address of external I2C slave device to access. Note: If I2C_MemAddr_Width selectin is 1-byte, only Mem_Addr [7:0] is valid, bit[23:8] is invalid.	RW	0x0

Table 2-5 I2C_MST1_DATA_CTRL/ I2C_MST2_DATA_CTRL Register

Bits	Field	Description	Type	Default
127:0	DATA	Data for read or write access.	RW	0x0

Some examples with detailed configuration steps as follows.

Example 1:

GPIO13, GPIO15 are reused as master I2C(1) and GPIO14, GPIO16 are reused as master I2C(2).

GPIO_SCL_SEL[0] = 1; GPIO13 as master I2C(1) SCL

GPIO_SCL_SEL[1] = 1; GPIO14 as master I2C(2) SCL

GPIO_SDA_SEL[0] = 1; GPIO15 as SDA0

GPIO_SDA_SEL[1] = 1; GPIO16 as SDA1

I2C_MST1_CTRL .SDA_OUT_SEL = 0; I2C(1) select SDA0(GPIO15) as SDA output

I2C_MST2_CTRL .SDA_OUT_SEL = 1; I2C(2) select SDA1(GPIO16) as SDA output

Example 2:

Write slave device register 0xc608 to value 0x12345678 by master I2C controller 1 with SDA6.(Slave device address is 0x54, address width is 2byte, data width is 4byte.)

GPIO_SCL_SEL[0] = 1; GPIO13 as master I2C(1)

GPIO_SDA_SEL[6] = 1; GPIO21 as SDA6

I2C_MST1_CTRL .SDA_OUT_SEL = 0x6; I2C(1) select SDA6 as SDA output

I2C_MST1_CTRL .DEV_ADDR = 0x54;

I2C_MST1_CTRL.MEM_ADDR_WIDTH = 0x2;

I2C_MST1_MEMADDR_CTRL.MEM_ADDR = 0xc608;

I2C_MST1_CTRL.DATA_WIDTH = 0x3;

I2C_MST1_DATA_CTRL.DATA[31:0] = 0x12345678;

I2C_MST1_CTRL.RWOP = 0x1;

I2C_MST1_CTRL.TRIG = 0x1;

Wait for I2C_MST1_CTRL.TRIG = 0 and I2C_MST1_CTRL.FAIL = 0, then write OK.

Example 3:

Read slave device register 0xc608 by master I2C controller 2 with SDA1.(Slave device address is 0x54, address width is 2byte, data width is 16byte.)

GPIO_SCL_SEL[1] = 1; GPIO14 as master I2C(2)

GPIO_SDA_SEL[1] = 1; GPIO16 as SDA1

I2C_MST2_CTRL .SDA_OUT_SEL = 0x1; I2C(2) select SDA1 as SDA output

I2C_MST2_CTRL .DEV_ADDR = 0x54;

I2C_MST2_CTRL.MEM_ADDR_WIDTH = 0x2;

I2C_MST2_MEMADDR_CTRL.MEM_ADDR = 0xc608;

I2C_MST2_CTRL.DATA_WIDTH = 0xf;

I2C_MST2_CTRL.RWOP = 0x0;

I2C_MST2_CTRL.TRIG = 0x1;

Wait for I2C_MST2_CTRL.TRIG = 0, and I2C_MST2_CTRL.FAIL = 0,

Read value from I2C_MST2_DATA_CTRL.DATA.

2.2 SDK Setting

SDK3

None.

rtk API

int32 drv_i2c_init(uint32 unit);

int32 drv_i2c_dev_init(uint32 unit, i2c_devConf_t *i2c_dev);

int32 drv_i2c_write(uint32 unit, drv_i2c_devId_t i2c_dev_id, uint32 reg_idx, uint8 *pBuff);

int32drv_i2c_read(uint32 unit, drv_i2c_devId_t i2c_dev_id, uint32 reg_idx, uint8 *pBuff);

I2C operation flow:

1. Initialize I2C

2. Initialize I2C dev

3. Read/Write I2C Data

Diag Shell Example:

Read slave device register 0xc608 by master I2C controller 2 with SDA1 (Slave device address is 0x54, address width is 2byte, data width is 16byte.)

```
RTK.0>i2c init unit 0 dev 1 intf 1 sck-gpiodev 0 sck-pin 0 sda-gpiodev 0 sda-pin 1 addr-width 2
data-width 16 scl-freq 2 i2c-delay 4000 chip_addr 0x54
RTK.0> i2c get unit 0 dev 1 reg 0xc608
//intf 0 => GPIO13 as SCL
//intf 1 => GPIO14 as SCL
//sda-pin=0~11 => GPIO15~GPIO26
//addr-width => Memory address width
                    0x0: 0-Byte (I2C Command does not have address byte)
                    0x1: 1-Byte
                    0x2: 2-Byte
                    0x3: 3-Byte
//data-width => Data width
//scl-freq    =>    0x0: 100KHz
                    0x1: 400KHz
                    0x2: 50KHz
                    0x3: 2.5MHz

//chip_addr => I2C slave device address
```