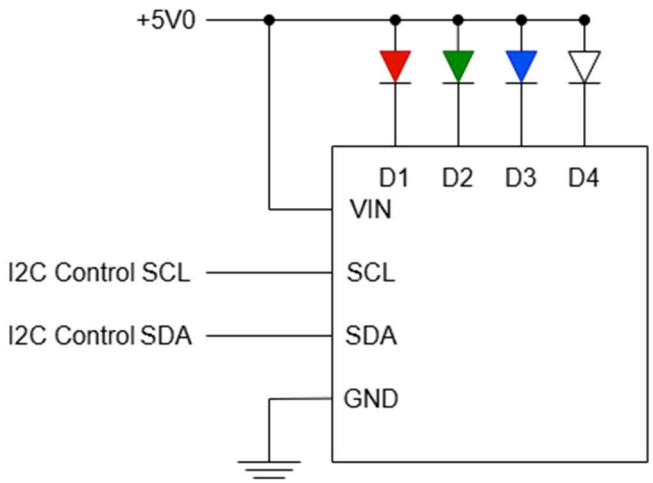
Выдержка из описания микросхемы управления RGB светодиодами

Typical Application



I2C-Compatible Timing Specifications (SCL, SDA), Figure 1

Symbol	Description	Conditions	Min	Typ	Max	Units
t_1	SCL (Clock Period)		2.5			µs
t_2	Data In Setup Time to SCL High		100			ns
t_3	Data Out Stable After SCL Low		0			ns
t_4	SDA Low Setup Time to SCL Low (Start)		100			ns
t_5	Setup time for STOP condition		600			ns

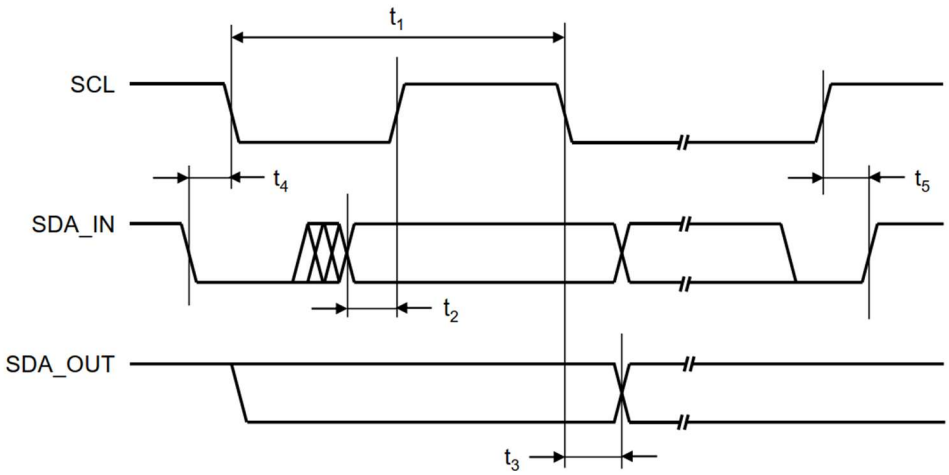


Figure 1. I2C Compatible Interface Timing

Register Map:

Register #	Description	Reset Value
0	Reset/Control	00h
1	Reserved	
2	Reserved	
3	Reserved	
4	Channel Enable	00h
5	Reserved	
6	Iout D1	4Fh
7	Iout D2	4Fh
8	Iout D3	4Fh
9	Iout D4	4Fh

Register 0:

Bits	Function
2…0	0b100 - Do Nothing (bit cleared) 0b101 - Reset Registers only 0b111 - Reset Complete Chip
4…3	0b00 - Device Enters Shutdown either SCL or SDA goes low 0b11 - Device always ON
7…5	Reserved

After power-up or VIN dropping below 2.7V, the device should be reset by writing register 0 value 0b111 binary. All registers are then restored to their default reset values. After sending the command for complete chip reset register 0[2:0] value 0b111, a 200µs delay is recommended before the next command to allow the device to execute the complete reset

Register 4:

Register 4 sets the mode of each D channel to either always ON/OFF.
For example, register 4: 0b00000001(binary), sets D1 ON and other channels OFF.

Channel Enable (1/2/3/4)	
Bit value	Function
0b00	Always OFF
0b01	Always ON

Register 6, Register 7, Register 8, Register 9:

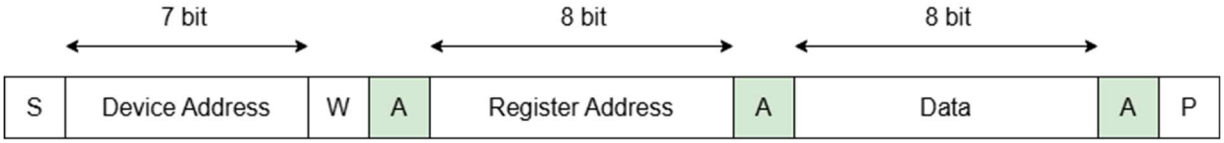
Registers 6 to 9 define the LED current setting for the channels D1 to D4 respectively. The LED current can be programmed to 192 levels between 0.125mA minimum and 24mA maximum with 0.125mA step.
For example, 24mA is set by the code BF hexadecimal (191 decimal, 1011 1111 binary) or any higher code value. 10mA current is set by the code 4F hexadecimal (79 decimal, 0100 1111 binary)

Iout (mA)	Data Dec	Data Hex	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 17	Bit 07
0.125	0	00h	0	0	0	0	0	0	0	0
0.25	1	01h	0	0	0	0	0	0	0	1
0.38	2	02h	0	0	0	0	0	0	1	0
0.50	3	03h	0	0	0	0	0	0	1	1
10.00	79	4Fh	0	1	0	0	1	1	1	1
10.13	80	50h	0	1	0	1	0	0	0	0
20.00	159	9Fh	1	0	0	1	1	1	1	1
20.13	160	A0h	1	0	1	0	0	0	0	0
23.88	190	BEh	1	0	1	1	1	1	1	0
24.00	191	BFh	1	0	1	1	1	1	1	1
24.00	192	C0h	1	1	0	0	0	0	0	0
24.00	254	FEh	1	1	1	1	1	1	1	0
24.00	255	FFh	1	1	1	1	1	1	1	1

I2C interface Protocol

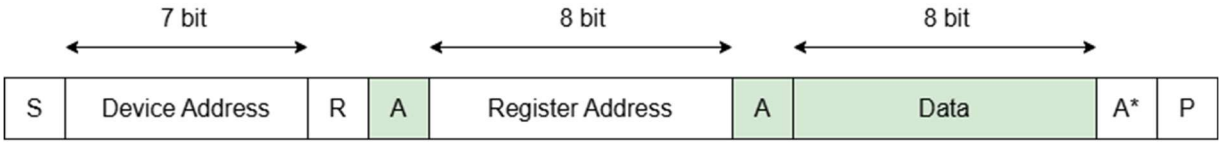
The internal registers can be accessed via the I2C interface. The I2C device address is 0x30 hexadecimal or 0b110000 binary. The read and write commands allow to modify the content of each register. For further details on the I2C protocol, please refer to the I2C-Bus Specification, document number 9398 393 40011, from Philips Semiconductors.
The protocol for Write and Read is the following.

Write

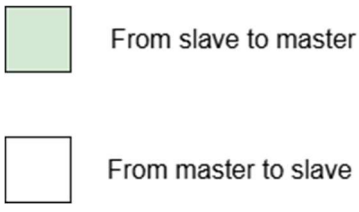


Note: For the I2C Reset commands (“Reset Register only” and “Reset Complete Chip”), the last byte is followed by a “not acknowledge” (SDA high). For these two commands, the lack of acknowledge at the end of the command is to be ignored.

Read



Where:
S = START condition
P = STOP condition
Device Address = 0110000 (7 bits, MSB first)
Register Address = Reg0 - Reg9 address (8 bits)
Data = data to read or write (8 bits)
1 = Read command bit
0 = Write command bit
A = acknowledge (SDA low)
A* = not acknowledge (SDA high)



API драйвера I2C интерфейса микроконтроллера

```
#ifndef I2CHW_H
#define I2CHW_H

#ifdef __cplusplus
extern "C" {
#endif

/**************************************************************************
 * INCLUDES
 **************************************************************************/

#include <stddef.h>
#include <stdint.h>
#include <stdbool.h>

/**************************************************************************
 * PUBLIC TYPES
 **************************************************************************/

typedef enum
{
    I2CHW_SUCCESS,
    I2CHW_ERR_INVALID_PARAMS,
    I2CHW_ERR_TIMEOUT,
    I2CHW_ERR_NACK,
    I2CHW_ERR_HW,
} i2chw_error_t;

typedef enum
{
    I2CHW_BUS_I2C0,
    I2CHW_BUS_I2C1,
    I2CHW_BUS_I2C2,
    I2CHW_BUS_I2C3,
} i2chw_bus_t;

typedef enum
{
```

```
    I2CHW_SLAVE_MODE,
    I2CHW_MASTER_MODE,
} i2chw_dir_mode_t;

typedef enum
{
    I2CHW_ADDR_WIDTH_7BIT,
    I2CHW_ADDR_WIDTH_8BIT,
    I2CHW_ADDR_WIDTH_10BIT,
} i2chw_addr_t;

typedef enum
{
    I2CHW_10_KHZ,
    I2CHW_100_KHZ,
    I2CHW_400_KHZ,
    I2CHW_1000_KHZ,
} i2chw_bus_freq_t;

typedef struct
{
    i2chw_bus_t  bus_num;       // I2C bus number
    uint16_t     dev_addr;      // I2C device address on I2C bus
    i2chw_addr_t addr_width;    // I2C device address width
} i2chw_dev_t;

typedef struct
{
    i2chw_bus_freq_t bus_freq;  // I2C frequency
    i2chw_dir_mode_t dir_mode;  // I2C slave/master mode
} i2chw_cfg_t;

/*****************************************************************************
 * PUBLIC FUNCTION PROTOTYPES
 *****************************************************************************/

/**
 * @brief Initialize I2C bus. Shall be called after I2CHW_Configure
 *
```

```
 * @param bus_num - number of I2C bus
 * @return hal_error_t - initialization result
 *        I2CHW_SUCCESS - the bus was initialized successfully
 *        I2CHW_ERR_INVALID_PARAMS - invalid parameters are given of the bus was not configured before
 *        I2CHW_ERR_HW - the bus was not initialized because of HW issues
 *
 */
i2chw_error_t I2CHW_Init(const i2chw_bus_t bus_num);



/**
 * @brief Configure I2C bus before initialization
 *
 * @param bus_num - number of I2C bus
 * @param p_config - pointer to I2C configuration
 * @return i2chw_error_t - configuration result
 *        I2CHW_SUCCESS - the bus was configured successfully
 *        I2CHW_ERR_INVALID_PARAMS - invalid parameters are given
 */
i2chw_error_t I2CHW_Configure(const i2chw_bus_t bus_num,
                              const i2chw_cfg_t* p_config);


/**
 * @brief Transmit data to device in master mode synchronously
 *
 * @param p_dev - pointer to I2C device configuration
 * @param p_tx_data - pointer to data to be transmitted
 * @param tx_data_len - length of transmitted data
 * @return i2chw_error_t - result of transmission
 *        I2CHW_SUCCESS - all data has been transmitted
 *        I2CHW_ERR_INVALID_PARAMS - invalid parameters are given
 *        I2CHW_ERR_TIMEOUT - transmission was not finished because of execution timeout
 *        I2CHW_ERR_NACK - the device did't set ACK bit
 *        I2CHW_ERR_HW - the bus has hardware issues
 */
i2chw_error_t I2CHW_WriteSync(const i2chw_dev_t* p_dev,
                              const uint8_t* p_tx_data,
                              const uint16_t tx_data_len);
```

```
/**
 * @brie Transmit data to I2C device and then receive data from the device in master mode synchronously
 *
 * @param p_dev - pointer to I2C device configuration
 * @param p_tx_data - pointer to data to be transmitted
 * @param tx_data_len - length of transmitted data
 * @param p_rx_data - pointer to buffer to put received data
 * @param rx_data_len - length of received data
 * @return i2chw_error_t - result of exchange
 *      I2CHW_SUCCESS - all data has been transmitted and received
 *      I2CHW_ERR_INVALID_PARAMS - invalid parameters are given
 *      I2CHW_ERR_TIMEOUT - exchange was not finished because of execution timeout
 *      I2CHW_ERR_NACK - the device did't set ACK bit
 *      I2CHW_ERR_HW - the bus has hardware issues
 *
 */
i2chw_error_t I2CHW_WriteReadSync(const i2chw_dev_t* p_dev,
                                  const uint8_t* p_tx_data,
                                  const uint16_t tx_data_len,
                                  uint8_t* p_rx_data,
                                  const uint16_t rx_data_len);

/**
 * @brief Receive data from device in master mode synchronously
 *
 * @param p_dev - pointer to I2C device configuration
 * @param p_rx_data - pointer to buffer to put received data
 * @param rx_data_len - length of received data
 * @return i2chw_error_t - result of receiving
 *      I2CHW_SUCCESS - all data has received
 *      I2CHW_ERR_INVALID_PARAMS - invalid parameters are given
 *      I2CHW_ERR_TIMEOUT - exchange was not finished because of execution timeout
 *      I2CHW_ERR_NACK - the device did't set ACK bit
 *      I2CHW_ERR_HW - the bus has hardware issues
 */
i2chw_error_t I2CHW_ReadSync(const i2chw_dev_t* p_dev,
                             uint8_t* p_rx_data,
                             const uint16_t rx_data_len);
```

```
#ifdef __cplusplus
}
#endif

#endif // I2CHW_H
```