I2C-Master IP Core
User Guide

张泽
Rev. V1.0
2017 年 12 月 13 日

1. 简介

I2C 是一种简单的串行通讯总线，由飞利浦公司在 1980 年代为让主板、嵌入式系统或手机用以连接低速周边设备而发展。自 2006 年 11 月起，I2C 协议是可以被免费使用的，但是芯片厂商仍需要付费以获得 I2C 从属设备的地址。

I2C 只使用两条双向漏极开路（串行数据 SDA 与串行时钟 SCL）并利用电阻进行上拉，I2C 允许相当大的工作电压范围，但典型电压等级为+3.3V 或者 5V。其设备地址包含 7bit 长度与 10bit 长度。I2C 传输速率有不同的模式：

·标准模式：100Kbit/s
·低速模式：10Kbit/s
·快速模式：400Kbit/s
·高速模式：3.4Mbit/s

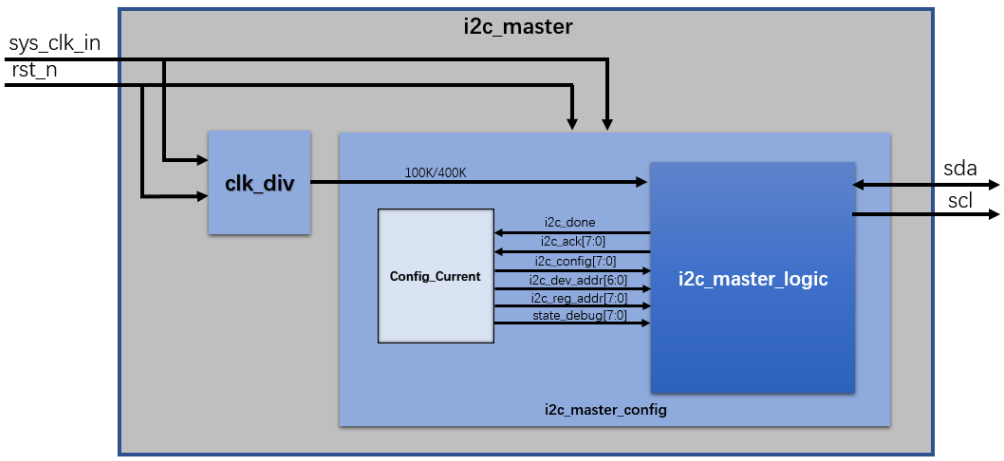
本 IP 的设计旨在使用户能够高效进行开发，减少开发周期。支持 7 位设备地址与标准/快速/低速模式，暂不支持 10 位地址操作模式，若相关器件性能支持，该 IP 也可运行在 3.4Mbit/s(所使用的 FPGA IO 建立时间需满足时序要求)。

注：本 IP 仅限学习交流使用，禁止商业用途。

2. I2C Master Core 特性

·符合 Philips I2C 标准
·支持主动挂起 I2C 总线进入等待状态
·支持多种传输速率
·支持重复读/写操作
·支持非标准 I2C 设备（如无寄存器地址操作）
·提供 ACK 触发信号
·提供状态调试寄存器

3. I2C Master Core 结构



I2C_Master_Core 由 4 部分组成，分别为 i2c_master，i2c_master_config，i2c_master_logic 与 clk_div，功能分别如下：

·i2c_master：为内核的 top module，调用所有 module 并建立连接。

·i2c_master_config：为内核配置 module，用户可修改该 module 实现 i2c 通信。

·i2c_master_logic：为内部逻辑实现 module，若预设操作模式没有涵盖到某种 I2C 设备，你可以自行修改该 module 中的内容，建立一种新的状态跳变模式。

·clk_civ：时钟分频 module，可进行任意整数分频，点击详细了解。

4. Signal & I/O Ports

4.1 i2c_master_logic Module

I2C_Master_Logic Module 是 I2C Master Core 逻辑实现部分，所包含接口如下所示：

Port	Width	Direction	Description
clk_in	1	Input	时钟输入，典型值为 100Kbit/s
rst_n	1	Input	复位信号输入，低电平有效
scl	1	output	I2C 总线时钟信号 Serial Clock
sda	1	Inout	I2C 总线数据信号 Serial Data
i2c_read_data	8	Output	I2C 总线读出数据
i2c_device_address	7	Input	I2C 从机设备地址，7bits
i2c_reg_address	8	Input	I2C 目标寄存器地址
i2c_write_reg_data	8	Input	I2C 写入目标寄存器数据
i2c_config	8	Input	配置 I2C 操作模式信号
write_done	1	Output	写完成标志位
read_done	1	Output	读完成标志位
state_debug	8	Output	状态指示寄存器
i2c_ack	8	Output	从机响应/主机响应位

4.1.1 clk_in

时钟输入信号，该信号直接影响 I2C 总线工作频率，典型输入频率为 100Kbit/s，可使 I2C 总线工作在 100K 标准模式下，如果从机设备支持快速或高速模式，该时钟频率可响应输入更高频率，如 400Kbit/s 或者 3.4Mbit/s。

4.1.2 rst_n

复位输入信号，该信号为被拉低时，电路进入复位状态。

4.1.3 scl & sda

I2C 通讯总线，硬件电路需配置为上拉，管脚约束建议同样调整为为上拉。

4.1.4 i2c_read_data

该寄存器用于存储主机在从机中读取到的数据。

4.1.5 i2c_device_address

该信号为从机设备地址，暂只支持 7 位的设备地址。

4.1.6 i2c_reg_address

该信号为读/写操作目标寄存器的地址，由外部输入。

4.1.7 i2c_write_reg_data

该信号为 I2C 写入目标寄存器数据，由外部输入。

4.1.8 i2c_config

该信号为配置 I2C 工作模式的外部输入信号，本 I2C Master IP 核支持如下工作模式，分别对应输入信号为：

·i2c_config= 8`h00

I2C 挂起，进入等待模式(WAIT)，等待状态下主机将 SCL 拉高并释放 SDA 总线。

·i2c_config= 8`h01

I2C 单次写入模式(I2C_Single_Write_Byte)，标准 1Byte 数据写入模式。

·i2c_config= 8`h02

I2C 连续写入模式(I2C_Continuous_Write_Byte)，主机对从机目标寄存器进行连续写入 1Byte 的数据，当主机发送到从机的 1Byte 数据并接受到 ACK 信号时，不会停止 I2C 总线，而是继续写入 1Byte 数据。该模式不会主动停止。

·i2c_config= 8`h03

I2C 直接写入状态(I2C_Write_Directly)，主机直接对从机设备进行 1Byte 的数据写入，即成功访问到设备并接受到 ACK 信号之后，直接写入 8bit 的数据即可。

·i2c_config= 8`h04

I2C 单次读取状态(I2C_Single_Read_Byte)，标准 1Byte 数据读取模式。

·i2c_config= 8`h05

I2C 连续读取状态(I2C_Continuous_Read_Byte)，主机对从机目标寄存器进行连续的数据读取操作，即在通讯的过程中，主机成功读取从机目标寄存器数据后，不会发送 NACK 信号，而是发送 ACK 信号并再次读取从机目标寄存器的数据。

·i2c_config= 8`h06

I2C 直接读取状态(I2C_Read_Directly)，主机对从机设备进行直接读取数据操作，即成功访问到设备并接受到 ACK 信号后，直接再次进行 START 模式，进行数据的读取。

4.1.9 write_done

写入完成并接收到从机响应后，该信号发出低电平脉冲，该信号在连续写入模式下无效。

4.1.10 read_done

读取完成并接受到从机响应后，该信号发出低电平脉冲，该信号在连续读

取模式下无效。

4.1.11 state_debug

该信号为程序运行状态指示寄存器。

4.1.12 i2c_ack

该信号为响应指示信号，对应关系如下：

i2c_ack[0]：写入设备地址从机响应位
i2c_ack[1]：写入寄存器地址从机响应位
i2c_ack[2]：写入寄存器数据从机响应位
i2c_ack[3]：读取寄存器地址从机响应位
i2c_ack[4]：成功读取寄存数据后主机发送 ACK
i2c_ack[5]：成功读取寄存器数据后主机发送 NACK
i2c_ack[4]：预留，默认值为 0
i2c_ack[5]：预留，默认值为 0

4.2 i2c_master_config module

i2c_master_config module 是主要功能为对 I2C 运行模式及寄存器地址、数据进行配置。相关寄存器与 i2c_master_logic module 相同，在此不在赘述。

4.3 clk_div module

本 module 为时钟分频模块，通过对系统时钟信号进行分频可生成 I2C 所需时钟，你可以点击此处了解该 clk_div module 的详细说明。

5. I2C 运行模式

本 I2C_Master IP 可以配置为多种操作模式（详见 3.1.8 小节），对应状态跳变如下所示。你可以在 i2c_master_config module 中拉取 i2c_done 与 i2c_ack 标志位信号实现状态跳变；此外，你也可以更改 i2c_master_logic 中关于状态跳变部分的代码，实现其它变种 I2C 的通讯模式。

5.1 I2C 单次写入模式(I2C_Single_Write_Byte)

起始信号	设备地址+写	ACK	寄存器地址	ACK	寄存器数据	ACK	STOP

5.2 I2C 单次读取模式(I2C_Single_Write_Byte)

起始信号	设备地址+写	ACK	寄存器地址	ACK	START2	设备地址+读	ACK
读取数据	NACK	STOP					

5.3 I2C 连续写入模式(I2C_Continuous_Write_Byte)

起始信号	设备地址+写	ACK	寄存器地址	ACK	寄存器数据	ACK	寄存器数据
ACK	寄存器数据	ACK	.				

5.4 I2C 连续读取模式(I2C_Continuous_Read_Byte)

起始信号	设备地址+写	ACK	寄存器地址	ACK	START2	设备地址+读	ACK
读取数据	ACK	读取数据	ACK	读取数据	ACK	读取数据	·

5.5 I2C 直接写入模式(I2C_Write_Directly)

起始信号	设备地址+写	ACK	寄存器数据	ACK	STOP

5.6 I2C 直接读取模式(I2C_Read_Directly)

起始信号	设备地址+写	ACK	START2	设备地址+读	ACK	读取数据
NACK	STOP					

5.7 等待状态（WAIT）

等待状态下，SCL 置 1，SDA 被释放。

注：

1. 当访问从机设备地址没有响应时，程序将重新访问从机设备地址，直至设备响应。
2. 除访问设备地址的 ACK 信号外，从机没有反馈 ACK 信号，程序将停止运行，此时可调取 state_dubug 或 i2c_ack 信号进行调试。

6. 应用示例&相关信号说明

6.1 配置流程说明

I2C Master Core 的配置流程主要分为 3 步：

(1) 调用通讯过程中涉及到的 i2c_ack 信号并进行逻辑处理，作为状态跳变的触发条件；

(2) 根据实际需求，描述状态跳变；

(3) 在不同的状态中，配置 i2c 设备地址、设备寄存器地址与所写入的数据/存储读取的数据。

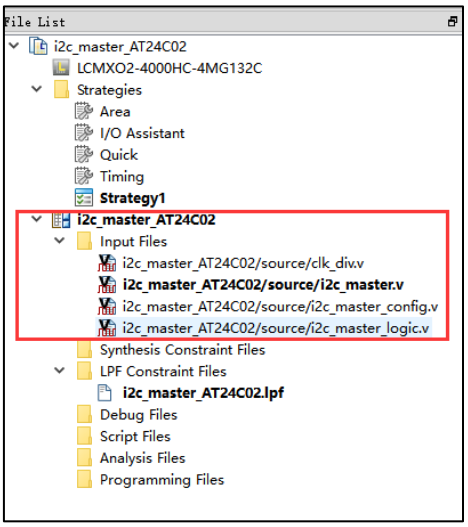
在本小节中，会以操作 AT24C02 为例，详细介绍的配置方法。详细实现操作 AT24C02 操作如下：

·在 AT24C02 的 00 寄存器中写入 8`hBB。

·读取 AT24C02 中 00 寄存器的数据。

实现流程:

首先，我们创建一个新工程，添加 I2C_Master IP 的 Verilog 文件进入工程之中：



根据需求，需要进行一次写操作、一次读操作、之后停止 I2C 通讯，所以我们需要一个写完成的信号(可用写入完成标志位 i2c_ack[2]上升沿作为触发，详见 6.2 小节)，一个单次读完成的信号(i2c_ack[5]，主机发送 NACK 标志位)，作为状态变化的触发。

打开 i2c_master_config，找到 i2c_ack 信号：

```
input           clk_12m;            //系统时钟 = 12M
input           rst_n;              //复位信号，低电平有效
output          scl;                //I2C时钟信号
inout           sda;                //I2C数据信号
input           i2c_clk;            //时钟输入管脚，100K or 400K;
output  [7:0]   i2c_read_data;      //I2C读出数据
output  [7:0]   i2c_config;         //I2C配置输入管脚
output  [6:0]   i2c_dev_addr;       //I2C从机设备地址
output  [7:0]   i2c_reg_addr;       //I2C写入目标寄存器地址
output  [7:0]   i2c_reg_data;       //I2C写入目标寄存器数据
output  [7:0]   state_debug;        //I2C状态机指示
output  [7:0]   i2c_ack;            //I2C响应位

i2c_master_logic i2c_master_logic_inst(
    .clk_in(i2c_clk),
    .rst_n(rst_n),
    .scl(scl),
    .sda(sda),
    .i2c_read_data(i2c_read_data),
    .i2c_device_address(i2c_dev_addr),
    .i2c_reg_address(i2c_reg_addr),
    .i2c_write_reg_data(i2c_reg_data),
    .state_debug(state_debug),
    .i2c_config(i2c_config),
    .i2c_ack(i2c_ack)
    );
```

新建一个名为 i2c_flag 的线网型变量，该信号等于（i2c_ack[2]|i2c_ack[5]）：

```
wire    i2c_flag;
assign  i2c_flag = i2c_ack[2] | i2c_ack[5];
```

这样，我们就实现了一个触发信号，之后再利用这个触发信号实现状态跳变，根据需求，程序可以分位三个状态，第一个状态为写入状态，第二个状态为读取状态，第三个状态为总线挂起。实现方式如下;

新建一个寄存器，位宽要足够容纳下总状态数:

```
reg [1:0]   i2c_state;      //I2C运行状态
```

描述三个状态的转移关系:

```
always@(posedge i2c_flag or negedge rst_n) begin
    if(!rst_n)
        i2c_state <= 8'h00;
    else if(i2c_state == 8'h02)
        i2c_state <= 8'h02;
    else
        i2c_state <= i2c_state+ 1'b1;
end
```

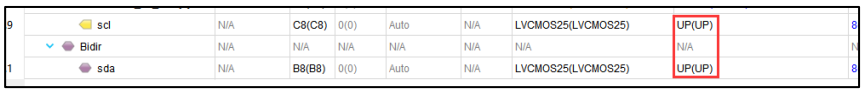
这样我们就实现了写入-读取-挂起的状态跳变，之后，我们建立一个 8 位的寄存器，用于存储读取出的数据，寄存器命名为 at24c02_00_data;

```
reg     [7:0]   at24c02_00_data;
```

接着， 我们在对应的状态中配置 i2c 的运行模式，设备地址，寄存器地址，写入的数据等参数，并保存读取的数据至 at24c02_00_data 寄存器中：

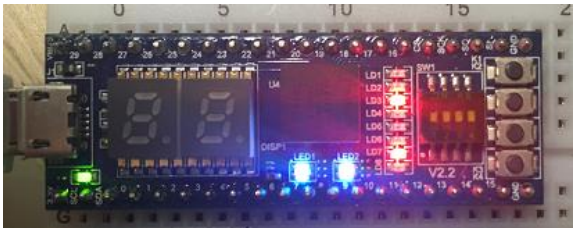
```
reg     [6:0]   i2c_dev_addr;   //设备地址
reg     [7:0]   i2c_reg_addr;   //寄存器地址
reg     [7:0]   i2c_reg_data;   //寄存器数据(写入操作有效)
reg     [7:0]   i2c_config;     //I2C工作模式配置寄存器
always@(posedge clk_12m or negedge rst_n) begin
    if(!rst_n) begin
        i2c_config  <=  8'd0;
        i2c_dev_addr<=  7'd0;
        i2c_reg_addr<=  8'd0;
        i2c_reg_data<=  8'd0; end
    else begin
        case(i2c_state)
            //状态0，单次写入模式，在AT24C02中的00地址写入BB
            0   :   begin   i2c_config      <= I2C_Single_Write_Byte;
                            i2c_dev_addr    <= 8'h50;
                            i2c_reg_addr    <= 8'h00;
                            i2c_reg_data    <= 8'hBB;   end
            //状态1，单次读取模式，读取AT24C02中的00地址的数据并将数据存储在寄存器中
            1   :   begin   i2c_config      <= I2C_Single_Read_Byte;
                            i2c_dev_addr    <= 8'h50;
                            i2c_reg_addr    <= 8'h00;
                            at24c02_00_data <= i2c_read_data;   end
            //状态2，通讯完成，总线挂起
            2   :   begin   i2c_config      <= I2C_Wait;    end
        default :   begin   i2c_config      <= 8'd0;    end
        endcase
    end
end
```

这样，我们就完成了对 AT24C02 的写入、读取、和 I2C 的挂起操作，之后，还要将 I2C 总线配置为内部上拉模式：



综合工程并下载，抓取 I2C 通讯的波形如下:

将 at24c02_00_data 寄存器输出至 LED 上，读出内容为“10111011”，即写入的“BB”。



你可以在“i2c_master/source/i2c_master_AT24C02”下找到该工程。

6.2 连续读写(需要器件支持)配置方式应用示例

连续读写通常需要从机支持，对于连续读写操作来说，在连续读或者连续写的过程中，不会返回 i2c_done 的完成脉冲信号，你需要拉取 i2c_ack 寄存器对应位的信号作为标志，来控制状态跳变，本小节以 PCF8591 为例，实现将存储在 Rom 中的数据转化为电压值进行输出，相关参数如下：

测试平台：STEP Baseboard

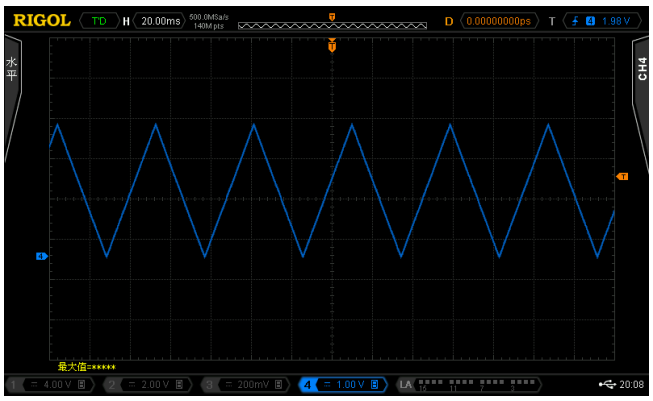
目标设备地址(i2c_dev_addr) = 8`h48；

目标寄存器地址(i2c_reg_addr) = 8`h00;

写入数据(i2c_reg_data) = rom_data

你可以在“i2c_master/source/i2c_master_PCF8591_DAC”目录中找到该工程。

运行该程序，量取板卡 DAC 输出管脚，可查看到输出波形如下：



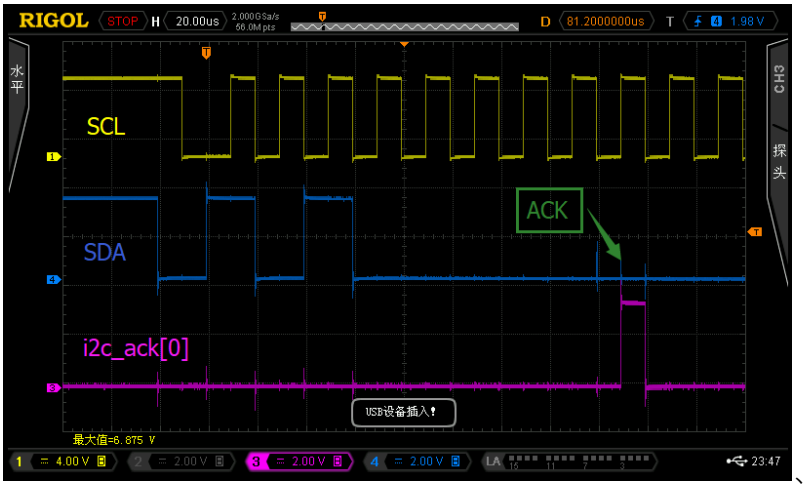
6.3 直接读写(需要器件支持)配置方式应用示例

某些从机设备需要在访问设备地址后直接对寄存器进行读写操作，你可以在“i2c_master/source/i2c_master_SHT20”目录中找到该工程。

6.4 i2c_ack 信号

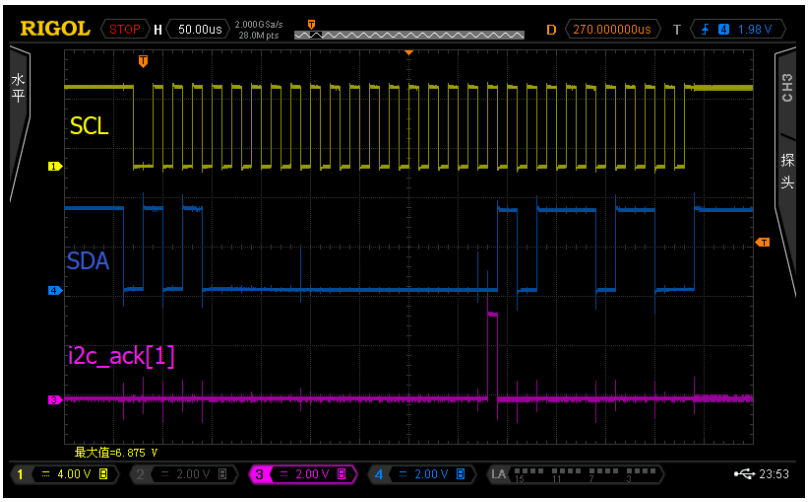
6.4.1 i2c_ack[0] - 写入设备地址从机响应位

i2c_ack[0]在设备响应之后，会发送一个高电平脉冲，信号如下：



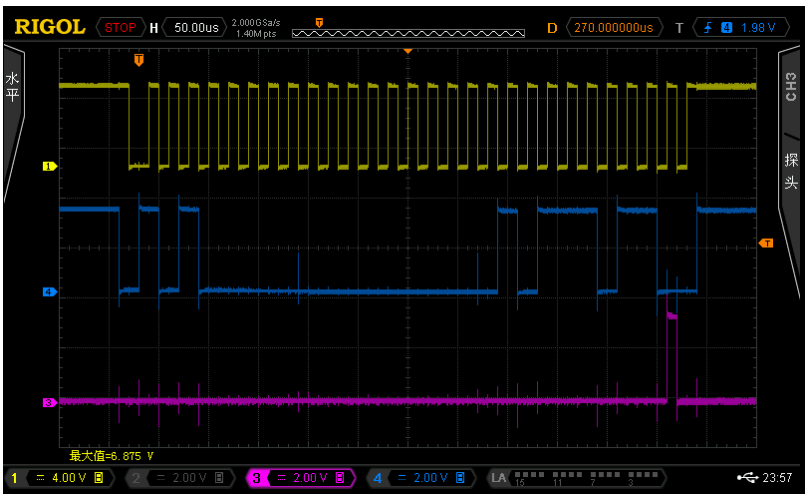
6.4.2 i2c_ack[1] - 写入寄存器地址从机响应位

i2c_ack[1]在写入从机设备的寄存器地址并收到响应之后，会发送一个高电平脉冲，信号如下：



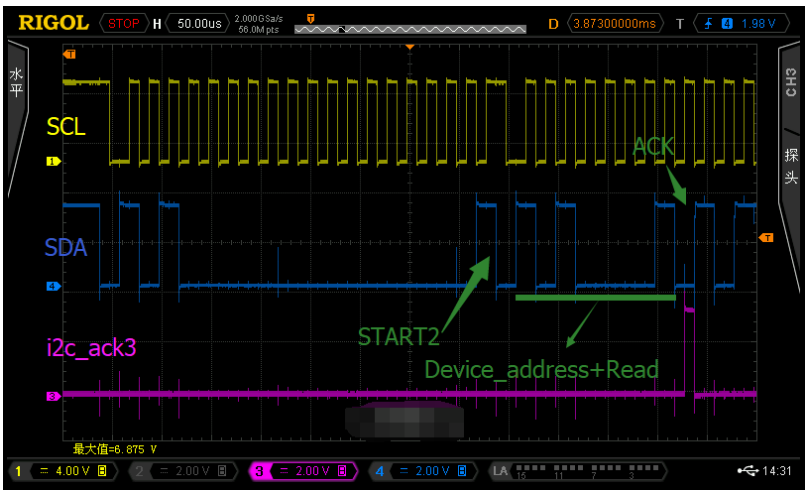
6.4.3 i2c_ack[2] – 写入寄存器数据从机响应位

i2c_ack[2]在写入从机设备的寄存器数据并收到响应之后，会发送一个高电平脉冲，信号如下：



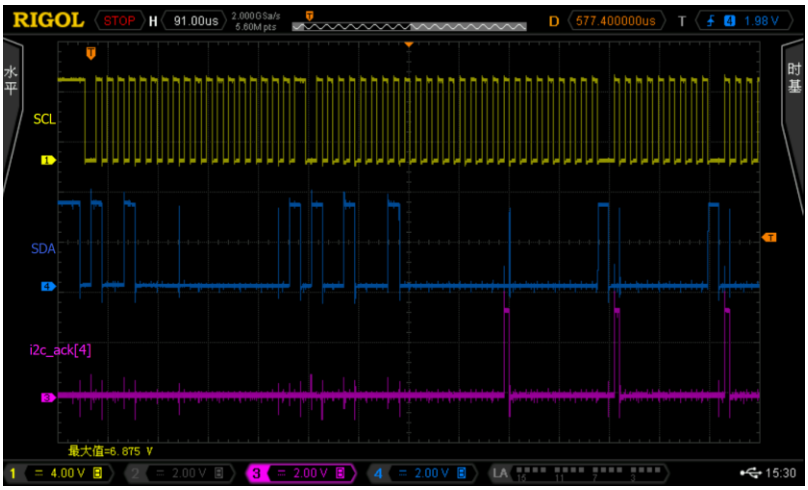
6.4.4 i2c_ack[3] – 设备地址+读从机响应位

i2c_ack[3]在读取操作过程中，写入从机设备地址+读并收到响应信号后，会发送一个高电平脉冲，信号如下：



6.4.5 i2c_ack[4] – **主机发送 ACK 标志位**

i2c_ack[4]在连续读取操作过程中，，每次成功读取数据后，会发送一个高电平脉冲，信号如下：



6.4.6 i2c_ack[5] – **主机发送 NACK 标志位**

i2c_ack[5]在单词读取操作过程中，成功读取数据后，主机发送 NACK 信号，i2c_ack[5]会发送一个高电平脉冲，信号如下：

7. 版本说明

日期	版本号	修订人	改动
2017.12.16	V0.0	张泽	最初版本