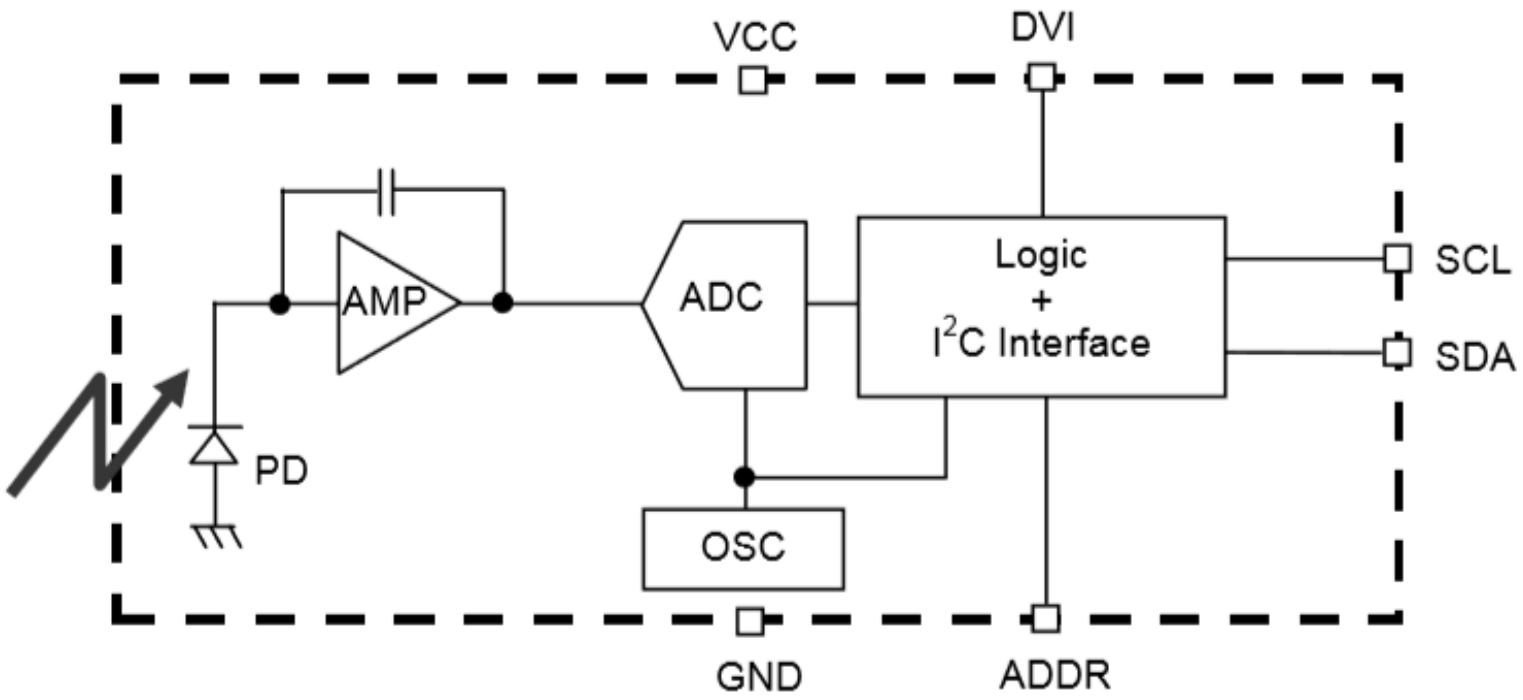
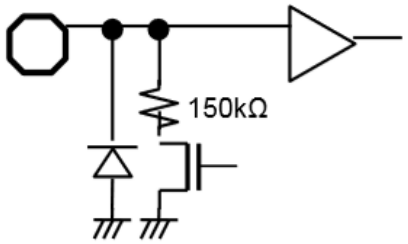
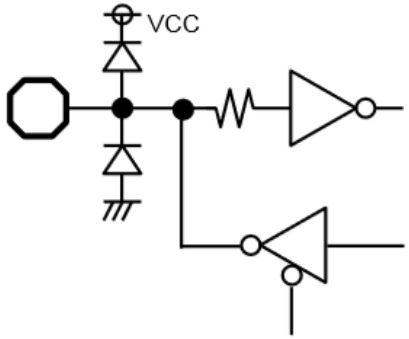
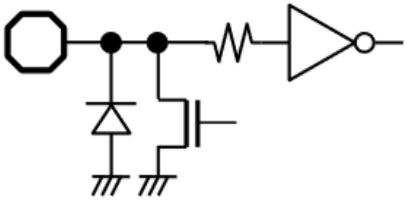
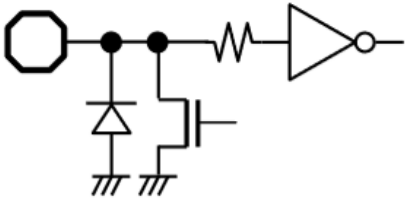
环境光传感器驱动

王安然
STEP FPGA

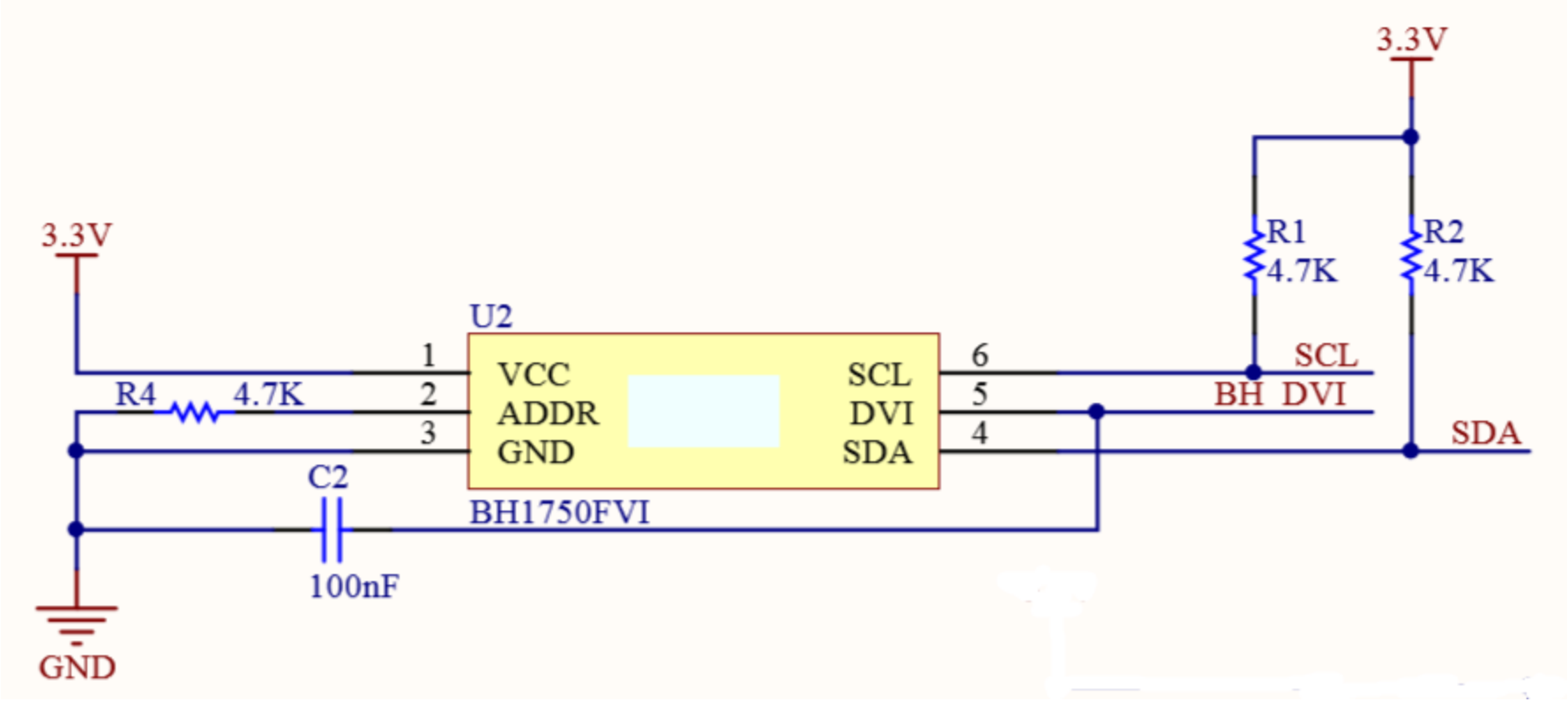
BH1750



BH1750引脚介绍

5	DVI		SDA, SCL Reference Voltage Terminal And DVI Terminal is also asynchronous Reset for internal registers.So that please set to 'L' (at least 1μs, DVI <= 0.4V) after Vcc is supplied. BH1750FVI is pulled down by 150kOhm while DVI = 'L'.
2	ADDR		I²C Slave-address Terminal ADDR = 'H' (ADDR ≧ 0.7Vcc) "1011100" ADDR = 'L' (ADDR ≦ 0.3Vcc) "0100011" ADDR Terminal is designed as 3 state buffer for internal test. So that please take care of Vcc and DVI supply procedure.Please see P6.
6	SCL		I²C bus Interface SCL Terminal
4	SDA		I²C bus Interface SDA Terminal

BH1750硬件连接

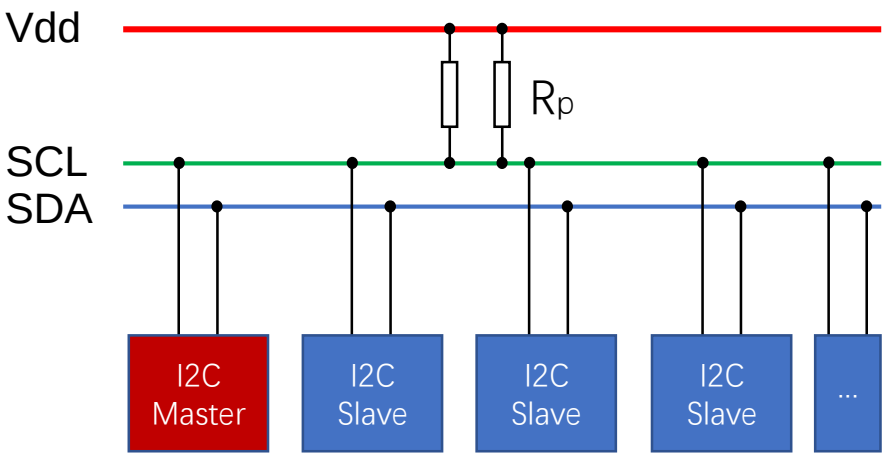


ADDR管脚接下拉电阻，I2C设备BH1750从机地址为0100011，7'h23
DVI管脚连接FPGA管脚，FPGA控制异步复位操作

I2C总线介绍

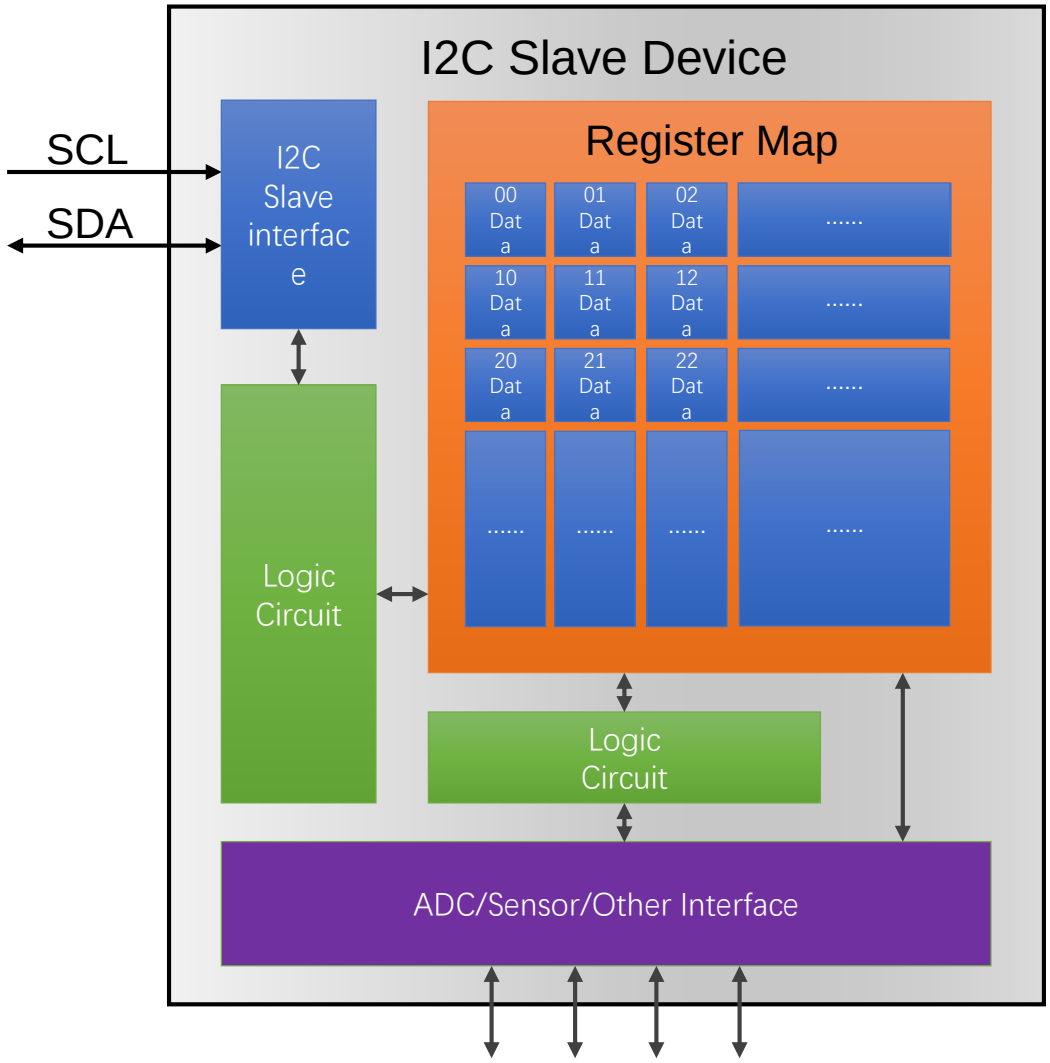
- 由飞利浦开发并获得专利(现属NXP)，将低速外围设备连接至主板、嵌入式系统或其它设备。

- 特性
  - 是一种支持多主机的串行总线
  - 一条数据线（SDA）
    一条时钟线（SCL）
    均为双向开漏需加上拉电阻
  - 每个连接入总线的从属设备
    均有唯一的7位/10位设备地址
  - 主机控制通讯的时钟信号

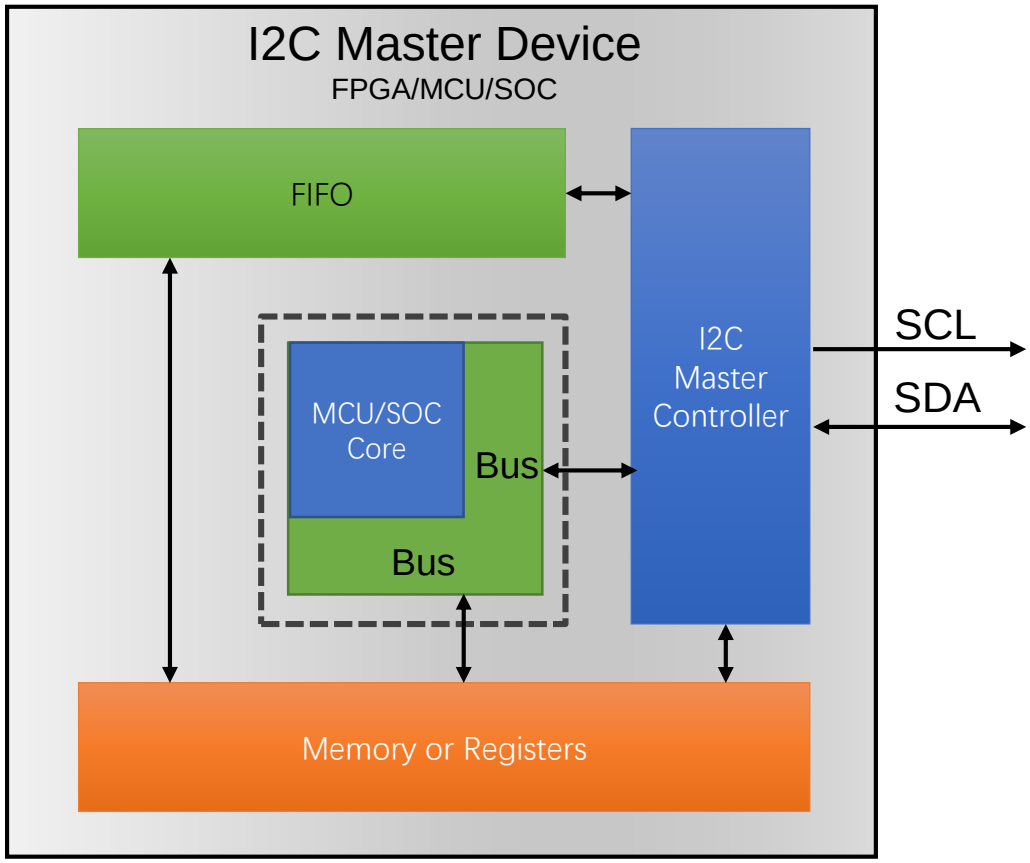


I2C总线 从机

- I2C从机接口无法操作时钟线
- 根据通信流程发送ACK信号
- 对I2C从机进行操作实际是对其内部寄存器读取和写入的过程
- 从机根据其寄存器数据变化而执行对应操作
- Slave设备通常都包含一个“Device ID”寄存器，内部存储着该I2C Slave的设备码（非设备地址）。这个设备码是需要向NXP申请并付费才能够获得。

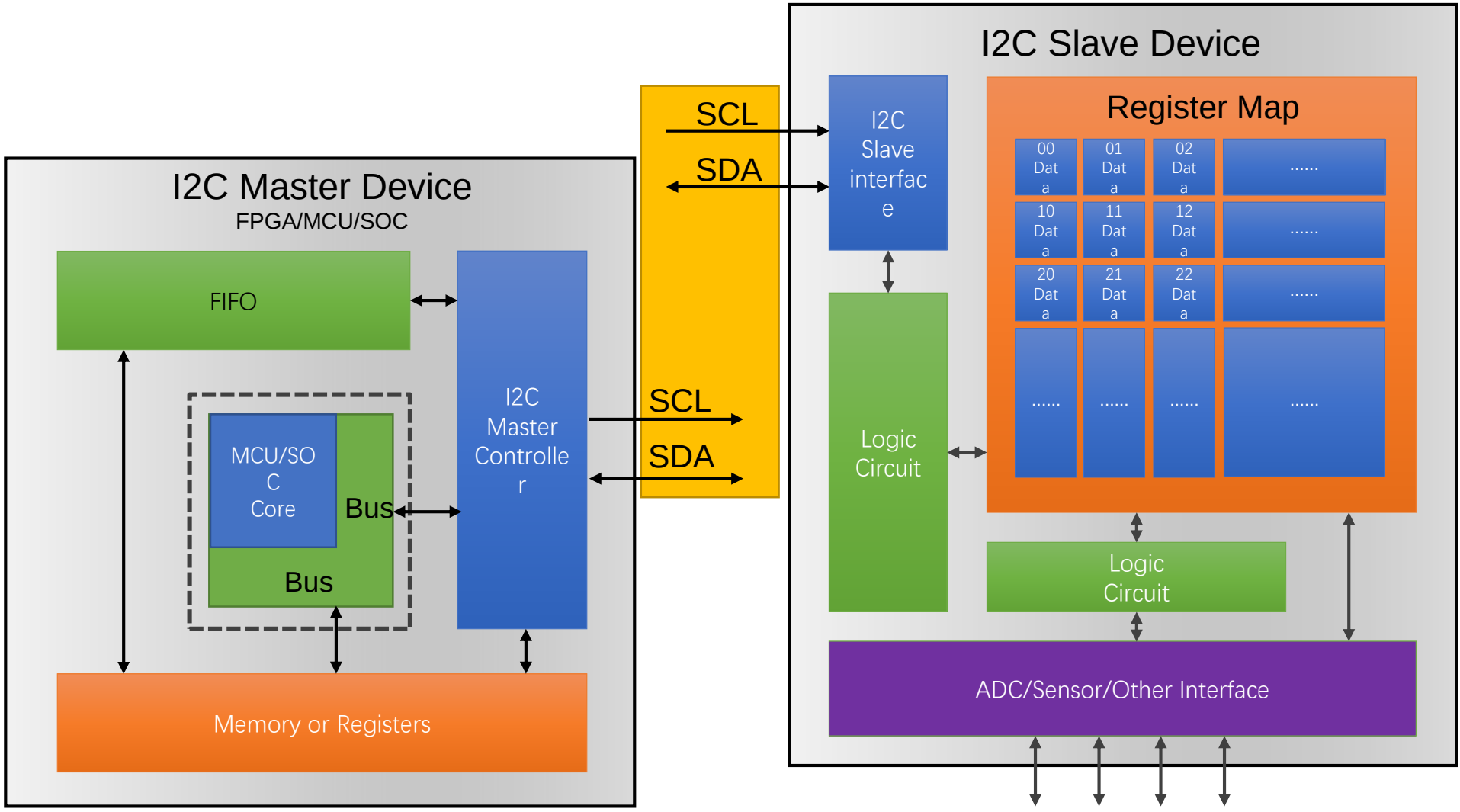


I2C总线-主机



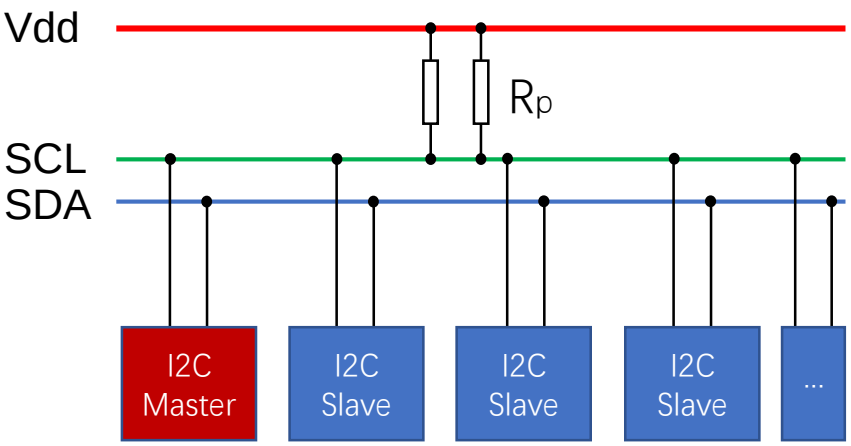
- I2C主机接口操作时钟线
- 配置I2C总线的工作流程
- 配置I2C总线的地址与数据
- ~~多主机时总线忙的判定~~

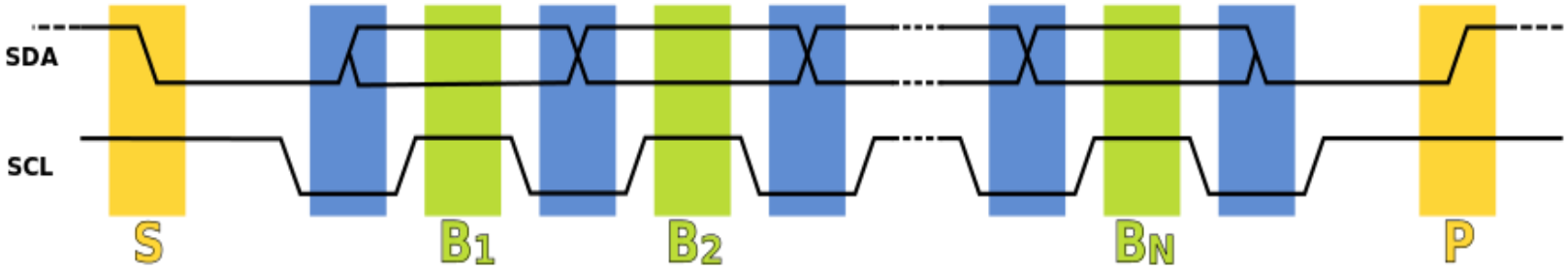
I2C总线连接


I2C Slave Device
Register Map
00 Data
01 Data
02 Data
.....
10 Data
11 Data
12 Data
20 Data
21 Data
22 Data
I2C Slave interface
Logic Circuit
Logic Circuit
ADC/Sensor/Other Interface
SCL
SDA
SCL
SDA
I2C Master Device
FPGA/MCU/SOC
FIFO
I2C Master Controller
MCU/SOC Core
Bus
Bus
Memory or Registers

I2C总线原理

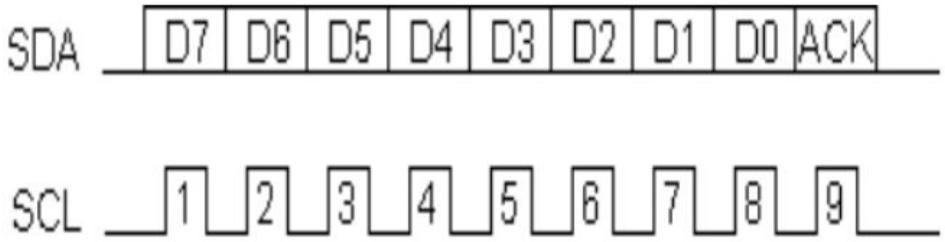
- 主器件用于启动总线传送数据，并产生时钟以开放传送的器件，此时任何被寻址的器件均被认为是从器件。
- 如果主机要发送数据给从器件，则主机首先寻址从器件，然后主动发送数据至从器件，最后由主机终止数据传送；
- 如果主机要接收从器件的数据，首先由主器件寻址从器件．然后主机接收从器件发送的数据，最后由主机终止接收过程。





I2C总线字节格式

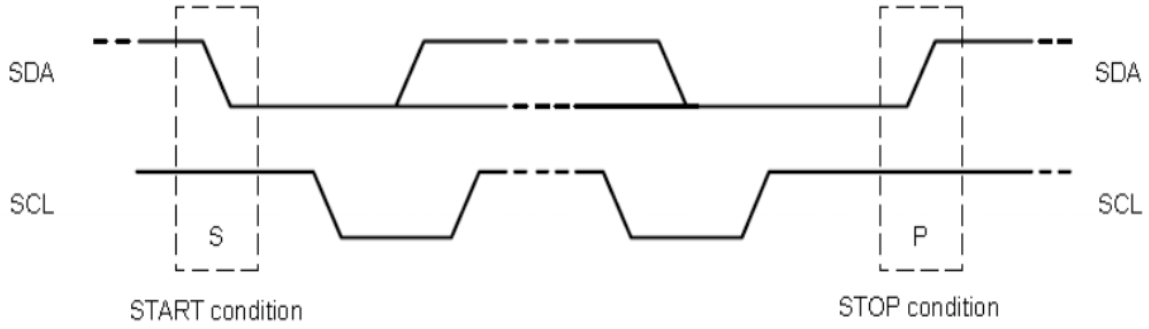
发送到SDA 线上的每个字节必须为8 位，每次传输可以发送的字节数量不受限制。每个字节后必须跟一个响应位。首先传输的是数据的最高位（MSB），如果从机要完成一些其他功能后（例如一个内部中断服务程序）才能接收或发送下一个完整的数据字节，可以使时钟线SCL 保持低电平，迫使主机进入等待状态，当从机准备好接收下一个数据字节并释放时钟线SCL 后数据传输继续。



I2C总线启动和停止

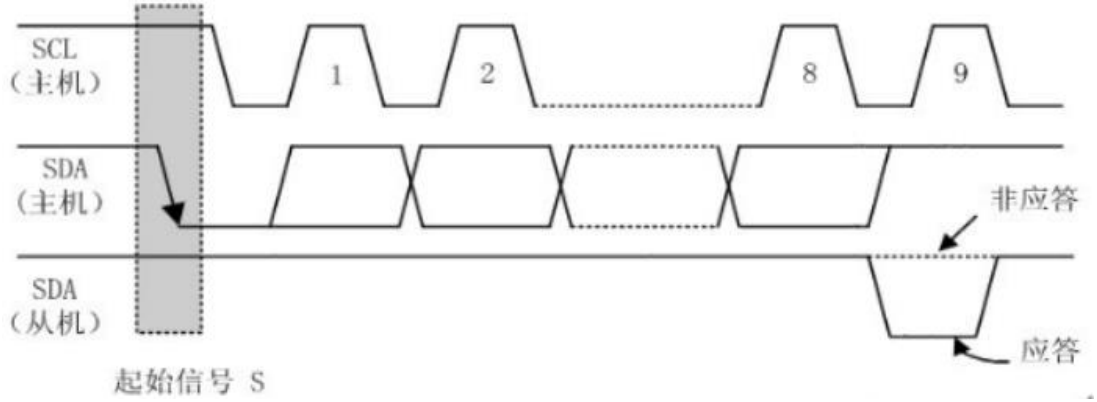
在时钟线SCL保持高电平期间，数据线SDA上的电平被拉低（即负跳变），定义为I2C总线总线的启动信号，它标志着一次数据传输的开始。启动信号是一种电平跳变时序信号，而不是一个电平信号。启动信号是由主控器主动建立的，在建立该信号之前I2C总线必须处于空闲状态。

在时钟线SCL保持高电平期间，数据线SDA被释放，使得SDA返回高电平（即正跳变），称为I2C总线的停止信号，它标志着一次数据传输的终止。停止信号也是一种电平跳变时序信号，而不是一个电平信号，停止信号也是由主控器主动建立的，建立该信号之后，I2C总线将返回空闲状态。



I2C总线应答响应

数据传输必须带响应，相关的响应时钟脉冲由主机产生。在响应的时钟脉冲期间，发送器释放SDA 线（上拉电阻拉高），接收器必须将SDA 线拉低，使它在这个时钟脉冲的高电平期间保持稳定的低电平，这种情况下是应答，如果在这个时钟脉冲的高电平期间SDA线没有被拉低则表示没有应答。通常被寻址的接收器在接收到的每个字节后，必须产生一个应答。当从机接收器不应答时，主机产生一个停止或重复起始条件。



I2C总线通信速率

常见的I²C总线依传输速率的不同而有不同的模式：标准模式（100 Kbit/s）、低速模式（10 Kbit/s），但时钟频率可被允许下降至零，这代表可以暂停通信。

而新一代的I2C总线可以和更多的节点（支持10比特长度的地址空间）以更快的速率通信：快速模式（400 Kbit/s）、高速模式（3.4 Mbit/s）。

常见写时序

假设主机向从机写命令，所需要经过的流程如下：

（1）主机发送起始信号
（2）主机访问设备地址+写信号
（3）从机应答
（4）主机发送指令数据
（5）从机应答
（6）主机发送停止信号

假设主机向从机发送数据，所需要经过的流程如下：

（1）主机发送起始信号
（2）主机访问设备地址+写信号
（3）从机应答
（4）主机发送寄存器地址
（5）从机应答
（6）主机发送寄存器数据
（7）从机应答
（8）主机发送停止信号

常见读数据时序

假设主机读取从机寄存器的数据，所需要经过的流程如下：

（1）主机发送起始信号

（2）主机访问设备地址+写信号

（3）从机应答

（4）主机发送寄存器地址

（5）从机应答

（6）主机再次发送起始信号

（7）主机访问设备地址+读信号

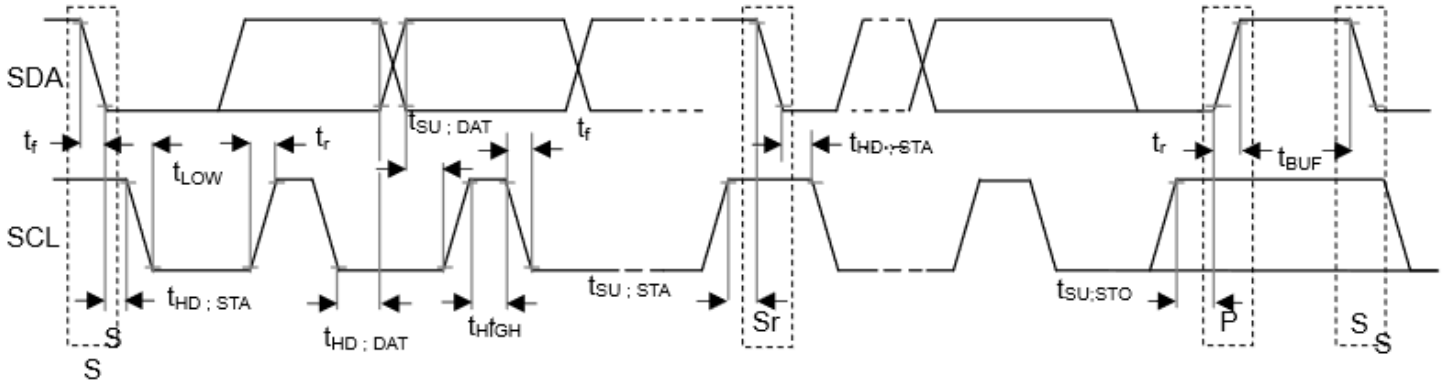
（8）从机应答

（9）主机读取数据

（10）若主机只读取1次，则发送NACK信号，跳转至(11)，若连续读取，则发送ACK信号并跳转至(9)继续读取。

（11）主机发送停止信号，停止通讯。

BH1750通信速率

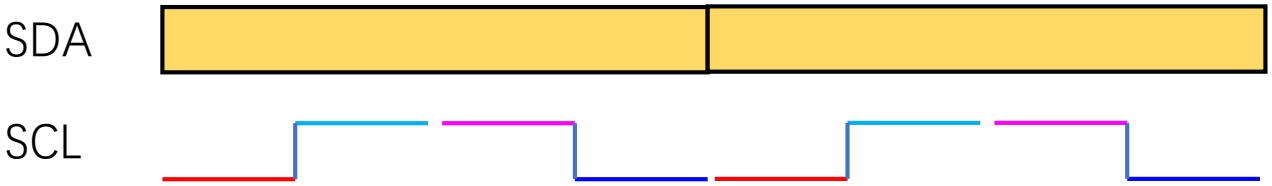


I²C SCL Clock Frequency	fSCL	–	–	400	kHz	
I²C Bus Free Time	tBUF	1.3	–	–	µs	
I²C Hold Time (repeated) START Condition	tHDSTA	0.6	–	–	µs	
I²C Set up time for a Repeated START Condition	tSUSTA	0.6	–	–	µs	
I²C Set up time for a Repeated STOP Condition	tSUSTD	0.6	–	–	µs	
I²C Data Hold Time	tHDDAT	0	–	0.9	µs	
I²C Data Setup Time	tSUDAT	100	–	–	ns	
I²C 'L' Period of the SCL Clock	tLOW	1.3	–	–	µs	
I²C 'H' Period of the SCL Clock	tHIGH	0.6	–	–	µs	

支持高速模式
400KHz

兼容正常模式
100KHz

I2C设计时钟




```
//使用计数器分频产生400KHz时钟信号clk_400khz
reg clk_400khz;
reg [9:0] cnt_400khz;
always@(posedge clk or negedge rst_n) begin
  if(!rst_n) begin
    cnt_400khz <= 10'd0; clk_400khz <= 1'b0;
  end else if(cnt_400khz >= CNT_NUM-1) begin
    cnt_400khz <= 10'd0;
    clk_400khz <= ~clk_400khz;
  end else begin
    cnt_400khz <= cnt_400khz + 1'b1;
  end
end
```

I2C状态机分析

I2C时序可以分解成基本单元（启动、停止、发送、接收、发应答、读应答），整个I2C通信都是由这些单元按照不同的顺序组合，我们设计一个状态机，将这些基本单元做成状态，控制状态机的跳转就能实现I2C通信时序。主机每次发送数据都要接收判断从机的响应，每次接收数据也要向从机发送响应，所以发送单元和读应答单元可以合并，接收单元和写应答单元可以合并。

- 启动状态
- 发送状态
- 接收状态
- 停止状态

启动和停止状态实现

```
START:begin //I2C通信时序中的起始START
        if(cnt_start >= 3'd5) cnt_start <= 1'b0;     //对START中的子状态执行控制cnt_start
        else cnt_start <= cnt_start + 1'b1;
        case(cnt_start)
            3'd0:   begin sda <= 1'b1; scl <= 1'b1; end //将SCL和SDA拉高，保持4.7us以上
            3'd1:   begin sda <= 1'b1; scl <= 1'b1; end //每个周期2.5us，需要两个周期
            3'd2:   begin sda <= 1'b0; end  //SDA拉低到SCL拉低，保持4.0us以上
            3'd3:   begin sda <= 1'b0; end  //clk_400khz每个周期2.5us，需要两个周期
            3'd4:   begin scl <= 1'b0; end  //SCL拉低，保持4.7us以上
            3'd5:   begin scl <= 1'b0; state <= state_back; end //每个周期2.5us，两个周期
            default: state <= IDLE; //如果程序失控，进入IDLE自复位状态
        endcase
    end

STOP:begin  //I2C通信时序中的结束STOP
        if(cnt_stop >= 3'd5) cnt_stop <= 1'b0;  //对STOP中的子状态执行控制cnt_stop
        else cnt_stop <= cnt_stop + 1'b1;
        case(cnt_stop)
            3'd0:   begin sda <= 1'b0; end  //SDA拉低，准备STOP
            3'd1:   begin sda <= 1'b0; end  //SDA拉低，准备STOP
            3'd2:   begin scl <= 1'b1; end  //SCL提前SDA拉高4.0us
            3'd3:   begin scl <= 1'b1; end  //SCL提前SDA拉高4.0us
            3'd4:   begin sda <= 1'b1; end  //SDA拉高
            3'd5:   begin sda <= 1'b1; state <= state_back; end //完成STOP操作
            default: state <= IDLE; //如果程序失控，进入IDLE自复位状态
        endcase
    end
```

写数据状态实现

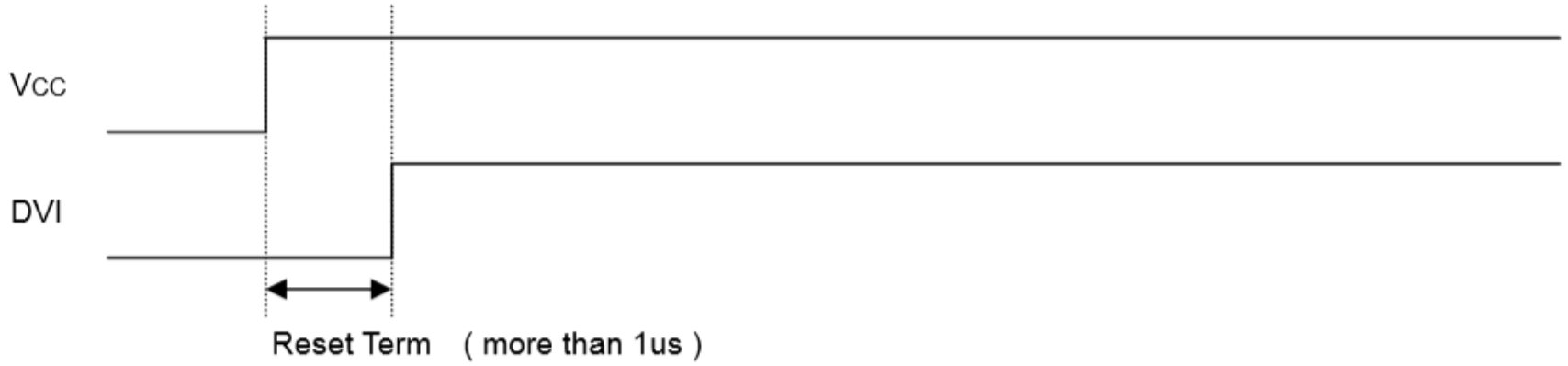
```
WRITE:begin //I2C通信时序中的写操作WRITE和相应判断操作ACK
        if(cnt <= 3'd6) begin   //共需要发送8bit的数据，这里控制循环的次数
            if(cnt_write >= 3'd3) begin cnt_write <= 1'b0; cnt <= cnt + 1'b1; end
            else begin cnt_write <= cnt_write + 1'b1; cnt <= cnt; end
        end else begin
            if(cnt_write >= 3'd7) begin cnt_write <= 1'b0; cnt <= 1'b0; end //复位变量
            else begin cnt_write <= cnt_write + 1'b1; cnt <= cnt; end
        end
        case(cnt_write)
            //按照I2C的时序传输数据
            3'd0:   begin scl <= 1'b0; sda <= data_wr[7-cnt]; end   //SCL拉低，SDA输出
            3'd1:   begin scl <= 1'b1; end  //SCL拉高，保持4.0us以上
            3'd2:   begin scl <= 1'b1; end  //clk_400khz每个周期2.5us，需要两个周期
            3'd3:   begin scl <= 1'b0; end  //SCL拉低，准备发送下1bit的数据
            //获取从设备的响应信号并判断
            3'd4:   begin sda <= 1'bz; end  //释放SDA线，准备接收从设备的响应信号
            3'd5:   begin scl <= 1'b1; end  //SCL拉高，保持4.0us以上
            3'd6:   begin ack_flag <= i2c_sda; end  //获取从设备的响应信号
            3'd7:   begin scl <= 1'b0;
                if(ack_flag)state <= state;
                else state <= state_back; end //SCL拉低，如果不应答循环写
            default: state <= IDLE; //如果程序失控，进入IDLE自复位状态
        endcase
    end
```

读数据状态实现

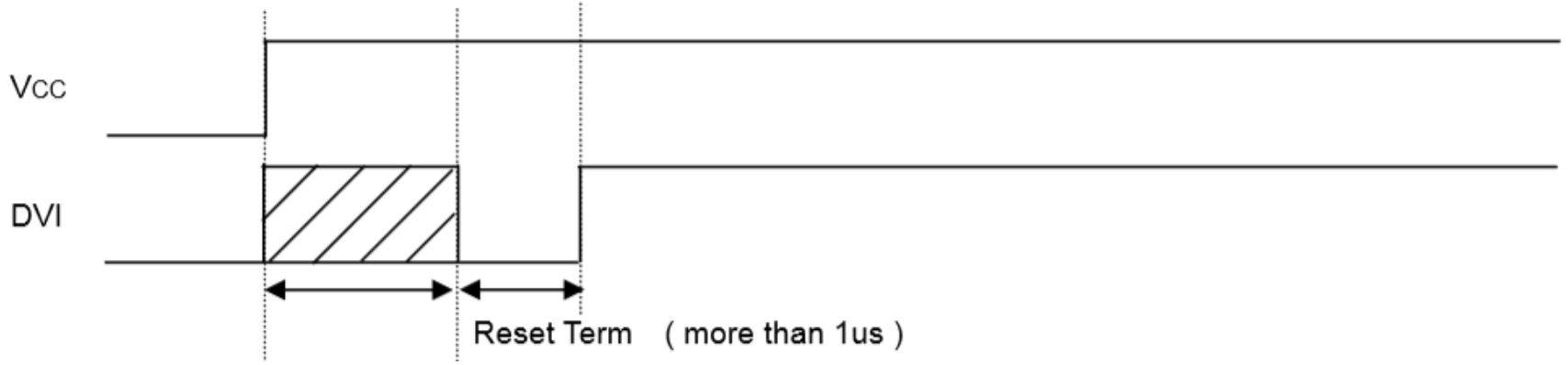
```
READ:begin  //I2C通信时序中的读操作READ和返回ACK的操作
        if(cnt <= 3'd6) begin    //共需要接收8bit的数据，这里控制循环的次数
            if(cnt_read >= 3'd3) begin cnt_read <= 1'b0; cnt <= cnt + 1'b1; end
            else begin cnt_read <= cnt_read + 1'b1; cnt <= cnt; end
        end else begin
            if(cnt_read >= 3'd7) begin cnt_read <= 1'b0; cnt <= 1'b0; end    //复位变量值
            else begin cnt_read <= cnt_read + 1'b1; cnt <= cnt; end
        end
        case(cnt_read)
            //按照I2C的时序接收数据
            3'd0:   begin scl <= 1'b0; sda <= 1'bz; end //SCL拉低，释放SDA线
            3'd1:   begin scl <= 1'b1; end  //SCL拉高，保持4.0us以上
            3'd2:   begin data_r[7-cnt] <= i2c_sda; end //读取从设备返回的数据
            3'd3:   begin scl <= 1'b0; end  //SCL拉低，准备接收下1bit的数据
            //向从设备发送响应信号
            3'd4:   begin sda <= ack; end   //发送响应信号，将前面接收的数据锁存
            3'd5:   begin scl <= 1'b1; end  //SCL拉高，保持4.0us以上
            3'd6:   begin scl <= 1'b1; end  //SCL拉高，保持4.0us以上
            3'd7:   begin scl <= 1'b0; state <= state_back; end //SCL拉低
            default: state <= IDLE; //如果程序失控，进入IDLE自复位状态
        endcase
    end
```

BH1750异步复位

1) Recommended Timing chart1 for VCC and DVI supply.

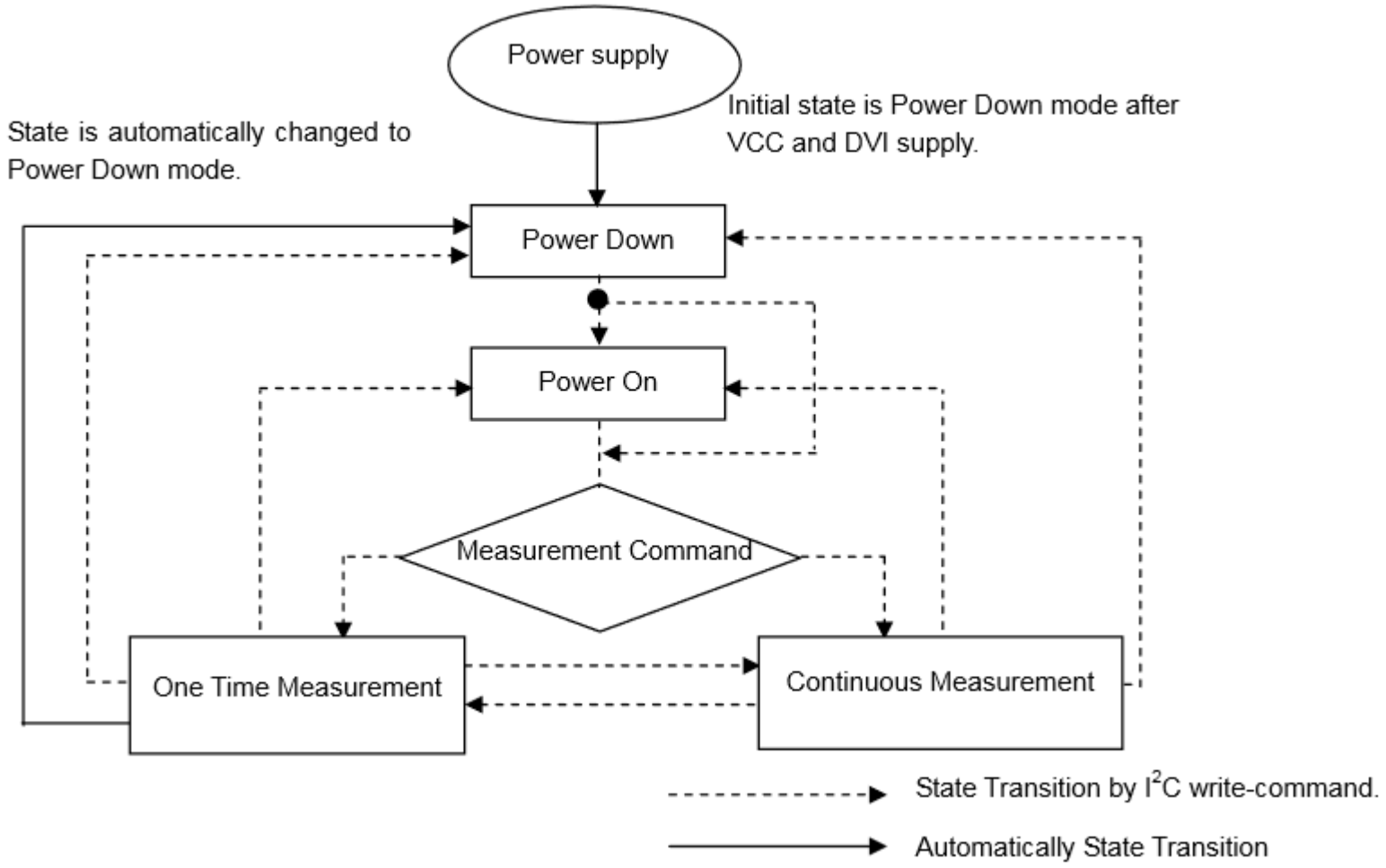


2) Timing chart2 for VCC and DVI supply.
(If DVI rises within 1μs after VCC supply)



BH1750工作流程

●Measurement Procedure



BH1750指令

Instruction	Opecode	Comments
Power Down	0000_0000	No active state.
Power On	0000_0001	Waiting for measurement command.
Reset	0000_0111	Reset Data register value. Reset command is not acceptable in Power Down mode.
Continuously H-Resolution Mode	0001_0000	Start measurement at 1lx resolution. Measurement Time is typically 120ms.
Continuously H-Resolution Mode2	0001_0001	Start measurement at 0.5lx resolution. Measurement Time is typically 120ms.
Continuously L-Resolution Mode	0001_0011	Start measurement at 4lx resolution. Measurement Time is typically 16ms.
One Time H-Resolution Mode	0010_0000	Start measurement at 1lx resolution. Measurement Time is typically 120ms. It is automatically set to Power Down mode after measurement.
One Time H-Resolution Mode2	0010_0001	Start measurement at 0.5lx resolution. Measurement Time is typically 120ms. It is automatically set to Power Down mode after measurement.
One Time L-Resolution Mode	0010_0011	Start measurement at 4lx resolution. Measurement Time is typically 16ms. It is automatically set to Power Down mode after measurement.
Change Measurement time (High bit)	01000_MT[7,6,5]	Change measurement time. ※ Please refer "adjust measurement result for influence of optical window."
Change Masurement time (Low bit)	011_MT[4,3,2,1,0]	Change measurement time. ※ Please refer "adjust measurement result for influence of optical window."

BH1750写指令时序

3) Write Format

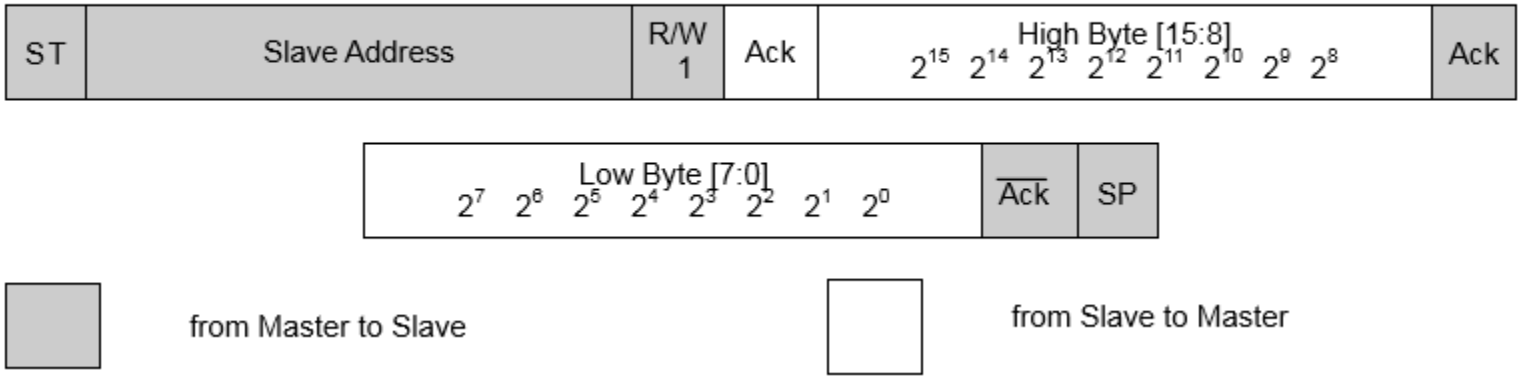
BH1750FVI is not able to accept plural command without stop condition. Please insert SP every 1 Opecode.

ST	Slave Address	R/W 0	Ack	Opecode	Ack	SP

```
MODE1:begin //单次写操作
        if(cnt_mode1 >= 4'd4) cnt_mode1 <= 1'b0;     //对START中的子状态执行控制cnt_start
        else cnt_mode1 <= cnt_mode1 + 1'b1;
        state_back <= MODE1;
        case(cnt_mode1)
            4'd0:   begin state <= START; end    //I2C通信时序中的START
            4'd1:   begin data_wr <= dev_addr<<1; state <= WRITE; end    //设备地址
            4'd2:   begin data_wr <= cmd_data; state <= WRITE; end  //寄存器地址
            4'd3:   begin state <= STOP; end     //I2C通信时序中的STOP
            4'd4:   begin state <= MAIN; end     //返回MAIN
            default: state <= IDLE; //如果程序失控，进入IDLE自复位状态
        endcase
    end
```

BH1750读数据时序

4) Read Format




```
MODE2:begin //两次读操作
        if(cnt_mode2 >= 4'd7) cnt_mode2 <= 4'd0;     //对START中的子状态执行控制cnt_start
        else cnt_mode2 <= cnt_mode2 + 1'b1;
        state_back <= MODE2;
        case(cnt_mode2)
            4'd0:   begin state <= START; end   //I2C通信时序中的START
            4'd1:   begin data_wr <= (dev_addr<<1)|8'h01; state <= WRITE; end   //设备地址
            4'd2:   begin ack <= ACK; state <= READ; end    //读寄存器数据
            4'd3:   begin dat_h <= data_r; end
            4'd4:   begin ack <= NACK; state <= READ; end   //读寄存器数据
            4'd5:   begin dat_l <= data_r; end
            4'd6:   begin state <= STOP; end    //I2C通信时序中的STOP
            4'd7:   begin state <= MAIN; end    //返回MAIN
           default: state <= IDLE; //如果程序失控，进入IDLE自复位状态
        endcase
    end
```

BH1750驱动流程

ex1) Continuously H-resolution mode (ADDR = 'L')

from Master to Slave

from Slave to Master

① Send "Continuously H-resolution mode " instruction

ST	0100011	0	Ack	00010000	Ack	SP

② Wait to complete 1st H-resolution mode measurement.(max. 180ms.)

③ Read measurement result.

ST	0100011	1	Ack	High Byte [15:8]	Ack

Low Byte [7:0]	$\overline{\text{Ack}}$	SP

How to calculate when the data High Byte is "10000011" and Low Byte is "10010000"

$(2^{15} + 2^{9} + 2^{8} + 2^{7} + 2^{4}) / 1.2 \fallingdotseq 28067$ [lx]

BH1750驱动过程

```
MAIN:begin
        if(cnt_main >= 4'd8) cnt_main <= 4'd3;       //写完控制指令后循环读数据
        else cnt_main <= cnt_main + 1'b1;
        case(cnt_main)
            4'd0:   bh_dvi <= 1'b1; //DVI输出拉高
            4'd1:   begin dev_addr <= 7'h23; cmd_data <= 8'h00; state <= MODE1; end //POWER OFF
            4'd2:   begin dev_addr <= 7'h23; cmd_data <= 8'h01; state <= MODE1; end //POWER ON

            4'd3:   begin dev_addr <= 7'h23; cmd_data <= 8'h10; state <= MODE1; end //测量
            4'd4:   begin num_delay <= 24'd72000; state <= DELAY; end   //180ms延时
            4'd5:   begin dev_addr <= 7'h23; state <= MODE2; end    //读取数据操作

            4'd6:   begin als_code <= {dat_h,dat_l}; end    //环境光数据输出
            4'd7:   als_pulse <= 1'b1;
            4'd8:   als_pulse <= 1'b0;
           default: state <= IDLE; //如果程序失控，进入IDLE自复位状态
        endcase
    end
```

除以1.2数据处理

```
reg [19:0] als_data1;
//光强数据编码运算 除以1.2  先乘以5，然后除以6
always @(posedge als_pulse or negedge rst_n)
    if(!rst_n) als_data1 <= 1'b0;
    else als_data1 <= als_code * 4'd5;

wire [3:0] remain;
wire [19:0] als_data2;
lpm_div u2  //例化除法器IP核，实现除以6运算
(
.numer              (als_data1  ), //分子
.denom              (4'd6       ), //分母
.quotient           (als_data2  ), //商
.remain             (remain     )  //余数
);
```

BCD转码 滚动显示

```
wire [19:0] als_data3;
bin_to_bcd u3
(
.rst_n              (rst_n      ),  //系统复位，低有效
.bin_code           (als_data2[15:0]),  //需要进行BCD转码的二进制数据
.bcd_code           (als_data3  )   //转码后的BCD码型数据输出
);

wire [19:0] als_data4;
assign als_data4[19:16] = als_data3[19:16]? als_data3[19:16]:4'ha;  //高位为0不显示
assign als_data4[15:12] = als_data3[19:12]? als_data3[15:12]:4'ha;  //高位为0不显示
assign als_data4[11: 8] = als_data3[19: 8]? als_data3[11: 8]:4'ha;  //高位为0不显示
assign als_data4[ 7: 4] = als_data3[19: 4]? als_data3[ 7: 4]:4'ha;  //高位为0不显示
assign als_data4[ 3: 0] = als_data3[ 3: 0]; //

dot_array_driver u4
(
.clk                (clk        ),
.rst_n              (rst_n      ),
.data               (als_data4  ),
.row                (row        ),
.col                (col        )
);
```

Thanks

扫描二维码
关注小脚丫微信公众号
了解更多FPGA知识