

语法

逻辑值

逻辑值

逻辑 0：表示低电平，也就对应我们电路 GND；

逻辑 1：表示高电平，也就对应我们电路的 VCC；

逻辑 X：表示未知，有可能是高电平，也有可能是低电平；

逻辑 Z：表示高阻态，外部没有激励信号，是一个悬空状态。

The diagram illustrates four logic values and their circuit representations:

- 0**: Low, False, Logic Low, Ground, VSS, Negative, Assertion. Represented by a buffer symbol with a 0 at the output.
- 1**: High, True, Logic High, Power, VDD, VCC, Positive Assertion. Represented by a buffer symbol with a 1 at the output.
- X**: Unknown Occurs at Logical Which Cannot be Resolved Conflict. Represented by a buffer symbol with an X at the output.
- Z**: High Impedance, Tri-Stated, Disabled Driver (Unknown). Represented by a buffer symbol with a Z at the output.

数字进制格式

数字进制格式

Verilog数字进制格式包括二进制、八进制、十进制和十六进制。
一般常用的为二进制、十进制和十六进制。

二进制表示如下：4'b0101 表示4位二进制数字0101

十进制表示如下：4'd2 表示4位十进制数字2（二进制0010）

十六进制表示如下：4'ha 表示4位十六进制数字a（二进制1010）

32'd

16'b1001_1010_1010_1001 = 16'h5AA5

100

标识符

标识符

标识符 (identifier) 用于定义模块名、端口名、信号名等。

标识符可以是任意一组字母、数字、\$符号和_(下划线)符号的组合；

但标识符的第一个字符必须是字母或者下划线；

标识符是区分大小写的；

S s

数据类型

数据类型

寄存器类型：

寄存器表示一个抽象的数据存储单元，通过赋值语句可以改变寄存器储存的值。
寄存器数据类型的关键字是 reg，reg 类型数据的默认初始值为不定值 x。

```
// reg define  
reg [31:0] delay_cnt; // 延时计数  
reg key_reg;
```

reg 类型的数据只能在 always 语句和 initial 语句中被赋值。

如果该过程语句描述的是时序逻辑，即 always 语句带有时钟信号，则该寄存器变量对应为触发器；

如果该过程语句描述的是组合逻辑，即 always 语句不带有时钟信号，则该寄存器变量对应为硬件连线；

数据类型

正点原子

线网类型：

线网数据类型表示结构实体（例如门）之间的物理连线。

线网类型的变量不能储存值，它的值是由驱动它的元件所决定的。

驱动线网类型变量的元件有门、连续赋值语句、assign等。

如果没有驱动元件连接到线网类型的变量上，则该变量就是高阻的，即其值为z。

线网数据类型包括wire型和tri型，其中最常用的就是wire类型。

```
11 // wire define
12 wire key_flag;
```

[1:0]

Verilog基础语法—数据类型

正点原子

参数类型：

参数其实就是一个常量，在 Verilog HDL 中用 parameter 定义常量。

我们可以一次定义多个参数，参数与参数之间需要用逗号隔开。

每个参数定义的右边必须是一个常数表达式。

```
//parameter define 1.3'RGB_LCD
parameter H_SYNC = 11'd41; //行同步
parameter H_BACK = 11'd2; //行显示后沿
parameter H_DISP = 11'd480; //行有效数据
parameter H_FRONT = 11'd2; //行显示前沿
parameter H_TOTAL = 11'd525; //行扫描周期
```

参数型数据常用于定义状态机的状态、数据位宽和延迟大小等。

采用标识符来代表一个常量可以提高程序的可读性和可维护性。

在模块调用时，可通过参数传递来改变被调用模块中已定义的参数。

^

$a \wedge b$

将 a 的每个位与 b 相同的位进行异或

运算符

移位运算符：

符号	使用方法	说明
<<	$a \ll b$	将 a 左移 b 位
>>	$a \gg b$	将 a 右移 b 位

两种移位运算都用0来填补移出的空位。

左移时，位宽增加；右移时，位宽不变。

$4'b1001 \ll 2 = 6'b100100;$

$4'b1001 \gg 1 = 4'b0100;$

拼接运算符：

符号	使用方法	说明
{}	{a,b}	将 a 和 b 拼接起来，作为一个新信号

$c = \{ a, b[3:0] \};$

$c[7:0] = \{ a[7:0], b[3:0] \}$

运算符的优先级：

运算符	优先级
!, ~	最高
*, /, %	次高
+, -	
<<, >>	
<, <=, >, >=	
==, !=, ===, !==	
&	
^, ^~	
&&	
	次低
?	最低

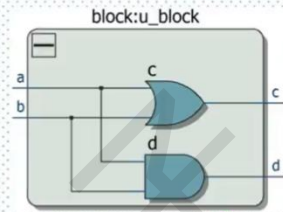
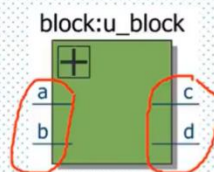
模块结构

模块的结构

Verilog 的基本设计单元是“模块” (block)。

一个模块是由两部分组成的，一部分描述接口，另一部分描述逻辑功能。

```
1 module block(a,b,c,d);  
2   input a,b;  
3   output c,d;  
4  
5   assign c = a | b;  
6   assign d = a & b;  
7  
8 endmodule
```



每个Verilog程序包括4个主要的部分：

端口定义、IO说明、内部信号声明、功能定义。

语句

结构语句

initial 和 always

initial 语句它在模块中只执行一次。

它常用于测试文件的编写，用来产生仿真测试信号（激励信号），或者用于对存储器变量赋初值。

always 语句一直在不断地重复活动。

但是只有和一定的时间控制结合在一起才有作用。

```
//给输入信号初始值  
initial begin  
  sys_clk      <= 1'b0;  
  sys_rst_n    <= 1'b0;  
  touch_key    <= 1'b0;  
  #20 sys_rst_n <= 1'b1;  
  #10 touch_key <= 1'b1;  
  #30 touch_key <= 1'b0;  
  #110 touch_key <= 1'b1;  
  #30 touch_key <= 1'b0;  
end  
  
//产生50MHz的时钟，周期为20ns  
always #10 sys_clk <= ~sys_clk;
```


结构语句

always 的时间控制可以是沿触发，也可以是电平触发：

可以是单个信号，也可以是多个信号，多个信号中间要用关键字 **or** 连接。

always 语句后紧跟的过程块是否运行，要看它的触发条件是否满足。

```
//计数器对系统时钟计数，计时0.2秒
always @(posedge sys_clk or negedge sys_rst_n) begin
    if (!sys_rst_n)
        counter <= 24'd0;
    else if (counter < 24'd1000_0000)
        counter <= counter + 1'b1;
    else
        counter <= 24'd0;
end
```

沿触发的 **always** 块常常描述时序逻辑行为。

由关键词 **or** 连接的多个事件名或信号名组成的列表称为“敏感列表”。

电平触发的 **always** 块常常描述组合逻辑行为。

```
always @(a or b or c or d or e or f or g or h or p or m) begin
    out1 = a ? (b + c) : (d + e);
    out2 = f ? (g + h) : (p + m);
end
```

如果组合逻辑块语句的输入变量很多，那么编写敏感列表会很烦琐并且容易出错。

```
always @(*) begin
    out1 = a ? (b + c) : (d + e);
    out2 = f ? (g + h) : (p + m);
end
```

@(*)表示对后面语句块中所有输入变量的变化都是敏感的。

赋值语句

Verilog HDL 语言中，信号有两种赋值方式

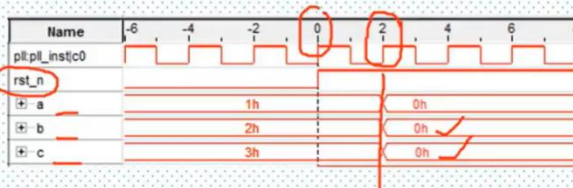
- 1、阻塞赋值 (blocking) , 如 $b = a;$
- 2、非阻塞赋值 (Non_Blocking) , 如 $b <= a;$

阻塞赋值可以认为只有一个步骤的操作：

即计算 RHS 并更新 LHS。

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        a = 1;
        b = 2;
        c = 3;
    end
    else begin
        a = 0;
        b = a;
        c = b;
    end
end
```

所谓阻塞的概念是指，在同一个always块中，后面的赋值语句是在前一句赋值语句结束后才开始赋值的。



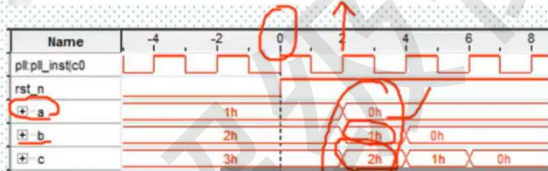
非阻塞赋值的操作过程可以看作两个步骤：

(1) 赋值开始的时候，计算 RHS；

(2) 赋值结束的时候，更新 LHS。

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        a <= 1;
        b <= 2;
        c <= 3;
    end
    else begin
        a <= 0;
        b <= a;
        c <= b;
    end
end
```

所谓非阻塞的概念是指，在计算非阻塞赋值的 RHS 以及更新 LHS 期间，允许其他的非阻塞赋值语句同时计算 RHS 和更新 LHS。



非阻塞赋值只能用于对寄存器类型的变量进行赋值，因此只能用在 initial 块和 always 块等过程块中。

在描述组合逻辑的 always 块中用阻塞赋值 =，综合成组合逻辑的电路结构；

这种电路结构只与输入电平的变化有关系。

在描述时序逻辑的 always 块中用非阻塞赋值 <=，综合成时序逻辑的电路结构；

这种电路结构往往与触发沿有关系，只有在触发沿时才可能发生赋值的变化。

注意：在同一个 always 块中不要既用非阻塞赋值又用阻塞赋值，不允许在多个 always 块中对同一个变量进行赋值！

条件语句

- 1、允许一定形式的简写，如：

if(a) 等同于 if(a == 1)

if(!a) 等同于 if(a != 1)

- 2、if语句对表达式的值进行判断，若为0, x, z, 则按假处理；若为1，按真处理。

- 3、if和else后面的操作语句可以用begin和end包含多个语句。

- 4、允许if语句的嵌套。

```
if(a) begin
    语句1;
    语句2;
end
else begin
    语句3;
    if(!b)
        语句4;
    else
        语句5;
end
```

case语句（多分支选择语句）

- 1、分支表达式的值互不相同；

- 2、所有表达式的位宽必须相等；

不能用 'bx 来代替 n'bx

- 3、casez

比较时，不考虑表达式中的高阻值z

- 4、casex

不考虑高阻值z 和 不定值x

```
reg [7:0] sel; //1100_0011
casez(sel)
    8' b1100_zzzz: 语句1;
    8' b1100_xxzz: 语句2;
endcase
```

```
//根据数码管显示的数值，控制段选信号
always @ (posedge clk or negedge rst_n) begin
    if (!rst_n)
        seg_led <= 8'b0;
    else begin
        case (num)
            4'h0 : seg_led <= 8'b1100_0000;
            4'h1 : seg_led <= 8'b1111_1001;
            4'h2 : seg_led <= 8'b1010_0100;
            4'h3 : seg_led <= 8'b1011_0000;
            4'h4 : seg_led <= 8'b1001_1001;
            4'h5 : seg_led <= 8'b1001_0010;
            4'h6 : seg_led <= 8'b1000_0010;
            4'h7 : seg_led <= 8'b1111_1000;
            4'h8 : seg_led <= 8'b1000_0000;
            4'h9 : seg_led <= 8'b1001_0000;
            4'ha : seg_led <= 8'b1000_1000;
            4'hb : seg_led <= 8'b1000_0011;
            4'hc : seg_led <= 8'b1100_0110;
            4'h4 : seg_led <= 8'b1010_0001;
            4'he : seg_led <= 8'b1000_0110;
            4'hf : seg_led <= 8'b1000_1110;
            default : seg_led <= 8'b1100_0000;
        endcase
    end
end
```


状态机

状态机设计

1 状态空间定义

```
//define state space
parameter SLEEP = 2'b00;
parameter STUDY = 2'b01;
parameter EAT = 2'b10;
parameter AMUSE = 2'b11;

// internal variable
reg [1:0] current_state;
reg [1:0] next_state;
```

```
//define state space
parameter SLEEP = 4'b1000;
parameter STUDY = 4'b0100;
parameter EAT = 4'b0010;
parameter AMUSE = 4'b0001;

// internal variable
reg [3:0] current_state;
reg [3:0] next_state;
```

独热码：每个状态只有一个寄存器置位，译码逻辑简单

2 状态跳转 (时序逻辑)

```
// transition
always @(posedge clk or negedge rst_n) begin
    if(!rst_n)
        current_state <= SLEEP;
    else
        current_state <= next_state;
end
```

敏感列表：
时钟信号以及复位信号边沿的组合

使用非阻塞赋值

3 下个状态判断 (组合逻辑)

```
// next state decision
always @(current_state or input signals) begin
    case (current_state)
        SLEEP: begin
            if(clock_alarm)
                next_state = STUDY;
            else
                next_state = SLEEP;
            end
        STUDY: begin
            if(lunch_time)
                next_state = EAT;
            else
                next_state = STUDY;
            end
        EAT: ... ;
        AMUSE: ... ;
        default: ... ;
    endcase
end
```

敏感信号表：
所有的右边表达式中的变量以及if、case条件中的变量

使用阻塞赋值

If/else要配对以避免latch的产生

■ 4 各个状态下的动作

```
// action
wire read_book;
assign read_book = (currentn_state == STUDY) ? 1'b1 : 1'b0;
```

```
@always @ (currentn_state) begin
    if (currentn_state == STUDY)
        read_book = 1;
    else
        read_book = 0;
end
```

使用阻塞赋值

Tb 文件

``timescale 1ns/1ps`: 时间刻度。 10

`#`: 等待。

`$stop`: 系统任务暂停。

`$stop(n)`: n 可取0、1、2。

`$finish`: 退出仿真任务。

`Initial`: 初始化语句

``timescale 1ns/1ps`: 时间刻度。 10

`#`: 等待。

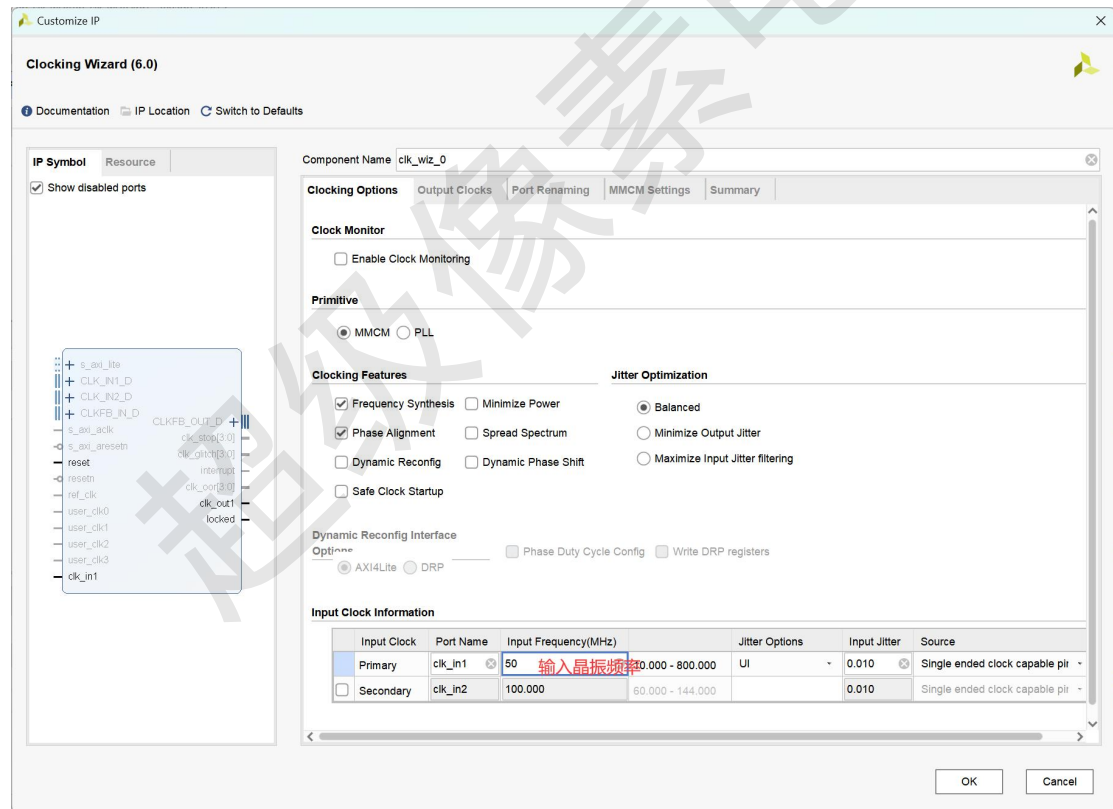
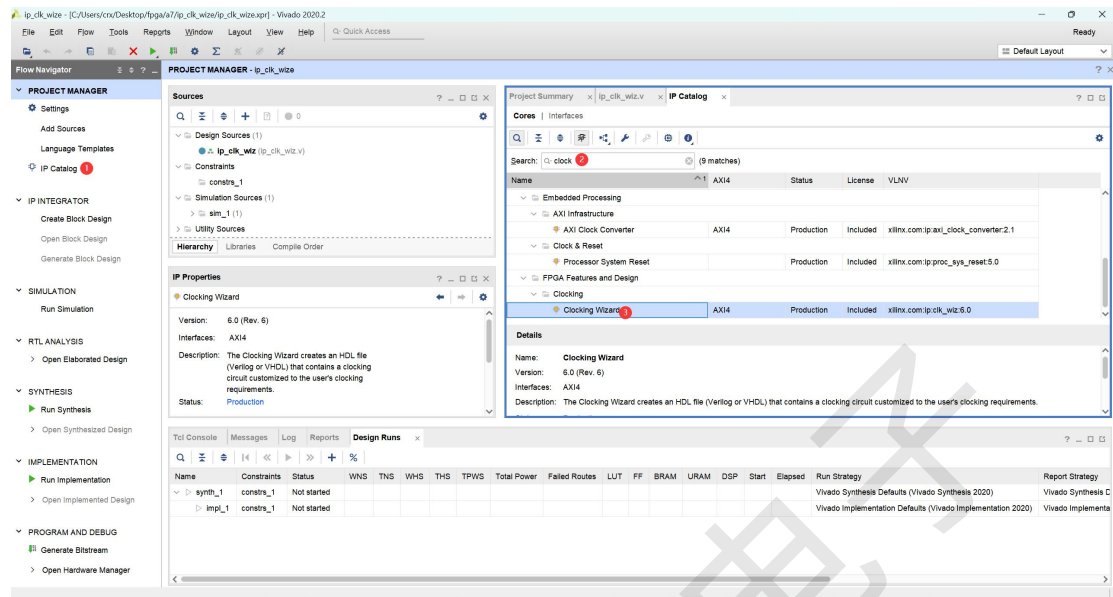
`$stop`: 系统任务暂停。

`$stop(n)`: n 可取0、1、2。

`$finish`: 退出仿真任务。

`Initial`: 初始化语句

时钟 IP 核



Customize IP

Clocking Wizard (6.0)

Documentation
IP Location
Switch to Defaults

IP Symbol

Resource

☒ Show disabled ports

+

s_axi_lite

+

CLK_IN1_D

+

CLK_IN2_D

+

CLKFB_IN_D

+

s_axi_ack

+

s_axi_arsteth

+

reset

+

ref_clk

+

user_clk0

+

user_clk1

+

user_clk2

+

user_clk3

+

clk_in1

+

CLKFB_OUT_D

+

clk_stop(3 0)

+

clk_glitch(3 0)

+

interrupt

+

clk_oor(3 0)

+

clk_out1

+

clk_out2

+

clk_out3

+

clk_out4

+

locked

Component Name

clk_wiz_0

Clocking Options

Output Clocks

Port Renaming

MMCM Settings

Summary

The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)		Drives
		Requested	Actual	Requested	Actual	Requested	Actual	
<input checked="" type="checkbox"/> clk_out1	clk_out1	100.000	100.00000	0.000	0.000	50.000	50.0	BUFG
<input checked="" type="checkbox"/> clk_out2	clk_out2	100.000	100.00000	180	180.000	50.000	50.0	BUFG
<input checked="" type="checkbox"/> clk_out3	clk_out3	50	50.00000	0.000	0.000	50.000	50.0	BUFG
<input checked="" type="checkbox"/> clk_out4	clk_out4	25	25.00000	0.000	0.000	50.000	50.0	BUFG
<input type="checkbox"/> clk_out5	clk_out5	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
<input type="checkbox"/> clk_out6	clk_out6	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
<input type="checkbox"/> clk_out7	clk_out7	100.000	N/A	0.000	N/A	50.000	N/A	BUFG

USE CLOCK SEQUENCING

Output Clock

Sequence Number

clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1
clk_out7	1

Clocking Feedback

Source

Signaling

☒ Automatic Control On-Chip
☐ Automatic Control Off-Chip
☐ User-Controlled On-Chip
☐ User-Controlled Off-Chip

☒ Single-ended
☐ Differential

OK

Cancel

Customize IP

Clocking Wizard (6.0)

Documentation
IP Location
Switch to Defaults

IP Symbol

Resource

☒ Show disabled ports

+

s_axi_lite

+

CLK_IN1_D

+

CLK_IN2_D

+

CLKFB_IN_D

+

s_axi_ack

+

s_axi_arsteth

+

reset

+

ref_clk

+

user_clk0

+

user_clk1

+

user_clk2

+

user_clk3

+

clk_in1

+

CLKFB_OUT_D

+

clk_stop(3 0)

+

clk_glitch(3 0)

+

interrupt

+

clk_oor(3 0)

+

clk_out1

+

clk_out2

+

clk_out3

+

clk_out4

+

locked

Component Name

clk_wiz_0

Clocking Options

Output Clocks

Port Renaming

MMCM Settings

Summary

<input checked="" type="checkbox"/> clk_out3	clk_out3	50	50.00000	0.000	0.000	50.000	50.0	BUFG
<input checked="" type="checkbox"/> clk_out4	clk_out4	25	25.00000	0.000	0.000	50.000	50.0	BUFG
<input type="checkbox"/> clk_out5	clk_out5	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
<input type="checkbox"/> clk_out6	clk_out6	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
<input type="checkbox"/> clk_out7	clk_out7	100.000	N/A	0.000	N/A	50.000	N/A	BUFG

USE CLOCK SEQUENCING

Output Clock

Sequence Number

clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1
clk_out7	1

Clocking Feedback

Source

Signaling

☒ Automatic Control On-Chip
☐ Automatic Control Off-Chip
☐ User-Controlled On-Chip
☐ User-Controlled Off-Chip

☒ Single-ended
☐ Differential

Enable Optional Inputs / Outputs for MMCM/PLL

Reset Type

时钟复位信号

☒ reset
☐ power_down
☐ input_clk_stopped

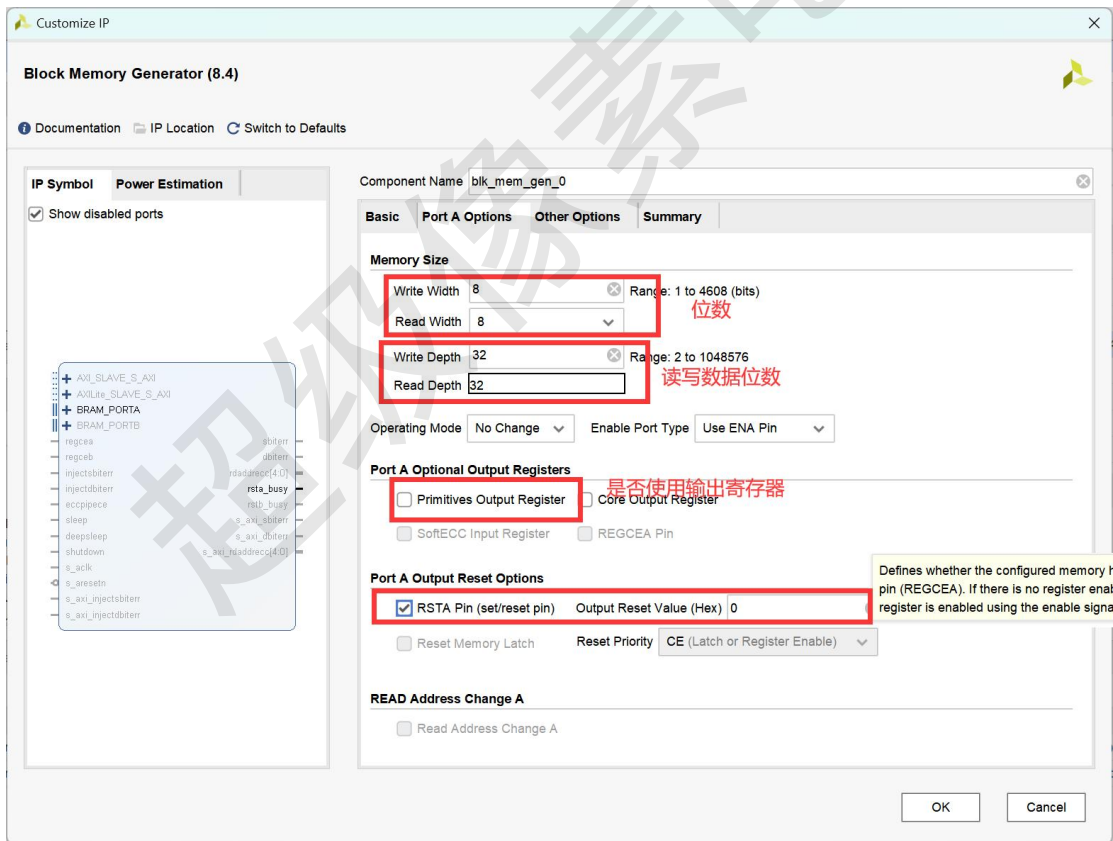
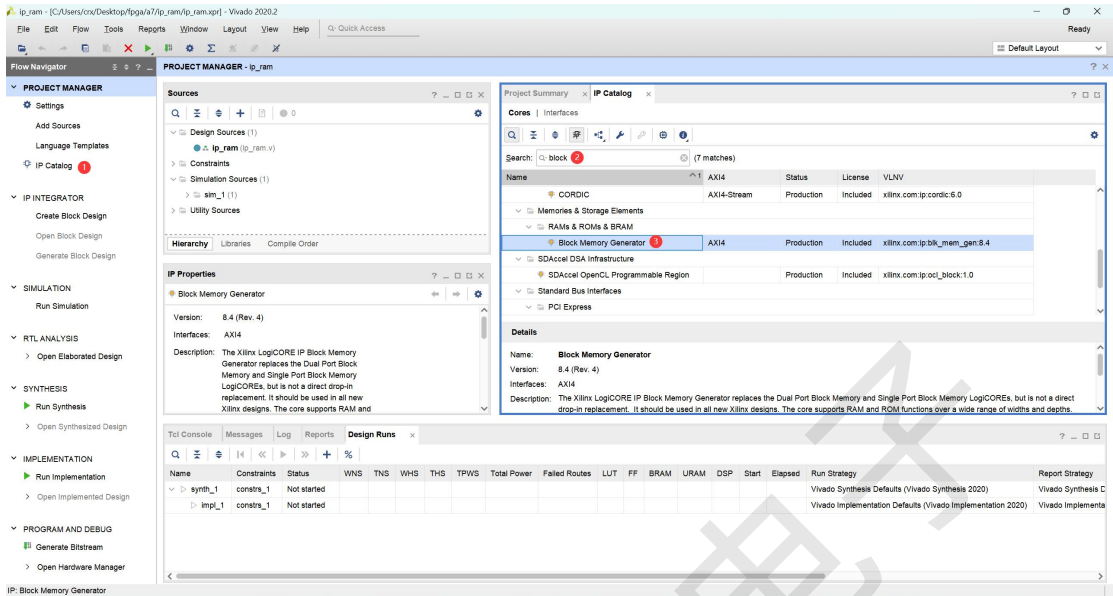
☒ locked
☐ clkfbstopped

☐ Active High
☒ Active Low

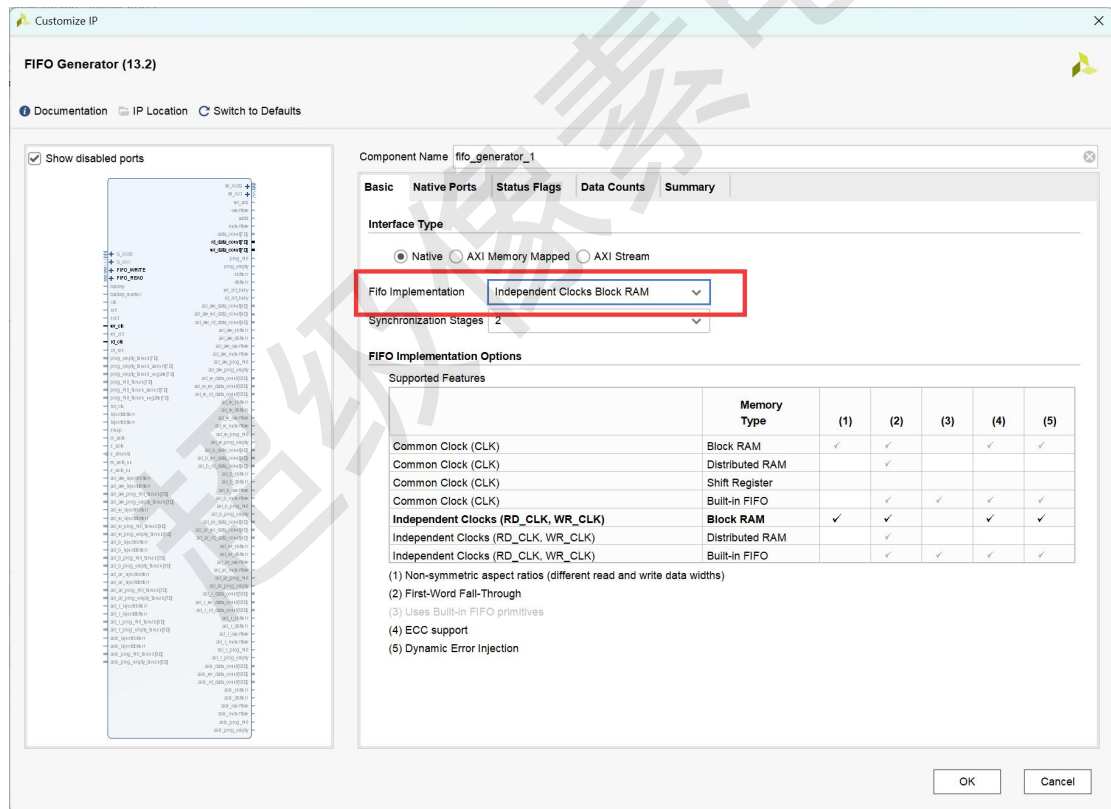
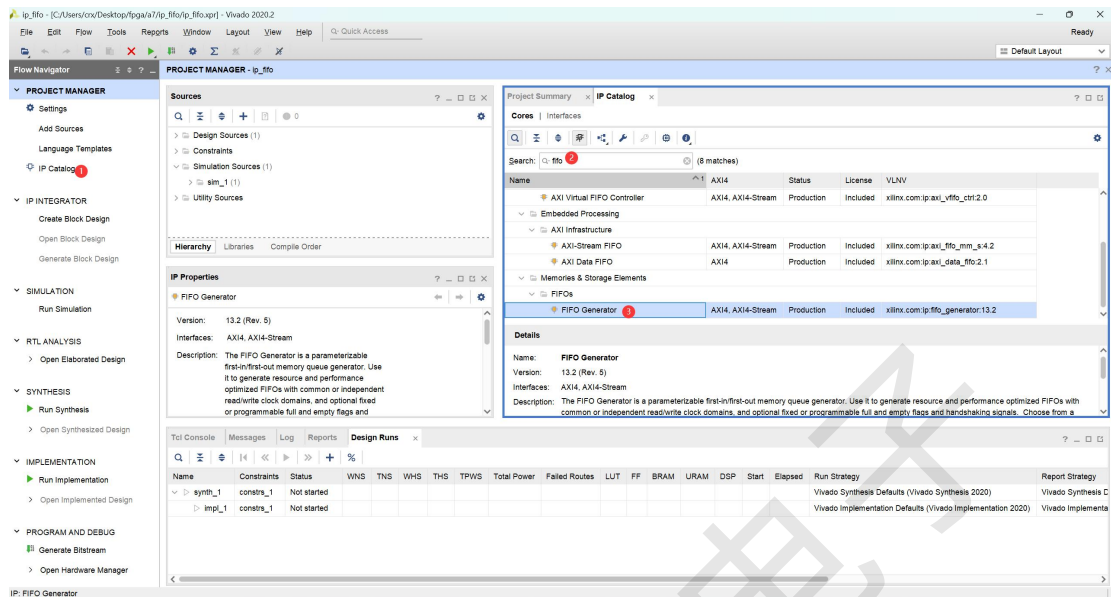
OK

Cancel

RAM IP 核



FIFO IP 核

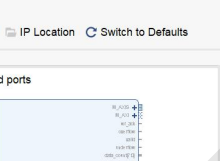


Customize IP

FIFO Generator (13.2)

Documentation
IP Location
Switch to Defaults

Show disabled ports



Component Name: ffo_generator_1

Basic

Native Ports

Status Flags

Data Counts

Summary

Output Flags

☒ Almost Full Flag
☒ Almost Empty Flag

Handshaking Options

Write Port Handshaking

☐ Write Acknowledge
Active High
☐ Overflow
Active High

Read Port Handshaking

☐ Valid Flag
Active High
☐ Underflow Flag
Active High

Programmable Flags

Programmable Full Type
No Programmable Full Threshold

Full Threshold Assert Value
254
[4 - 254]

Full Threshold Negate Value
253
[3 - 253]

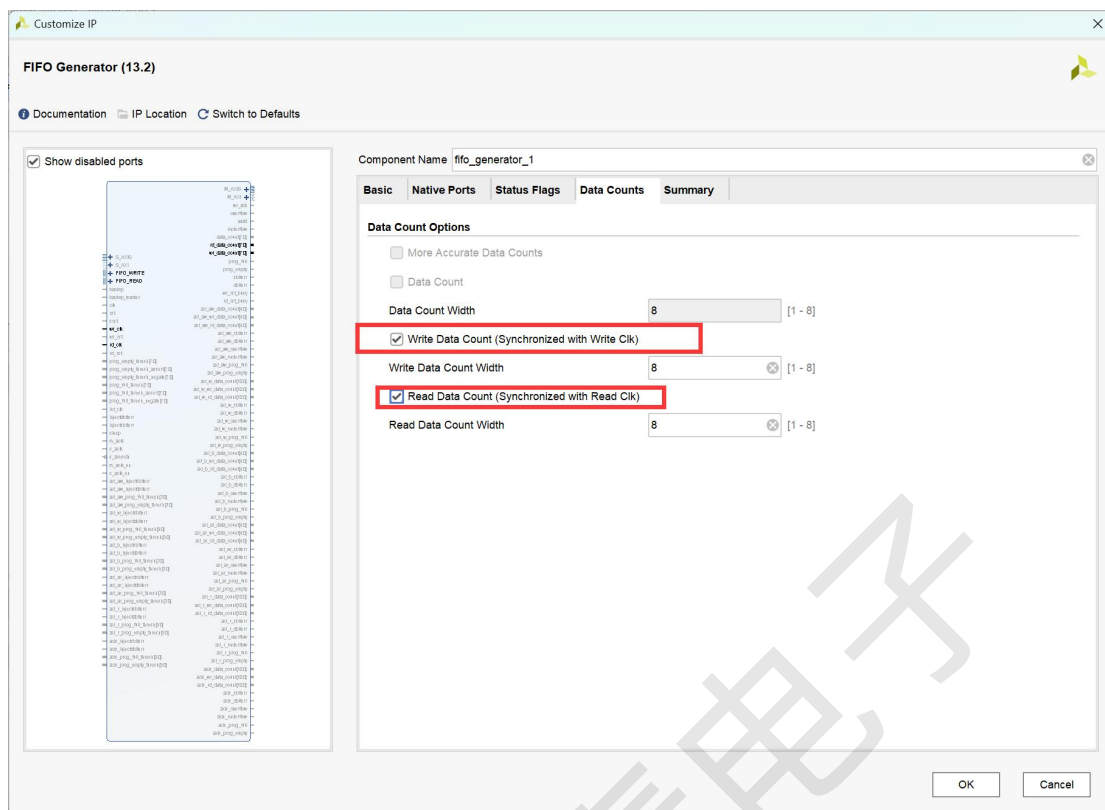
Programmable Empty Type
No Programmable Empty Threshold

Empty Threshold Assert Value
2
[2 - 252]

Empty Threshold Negate Value
3
[3 - 253]

OK

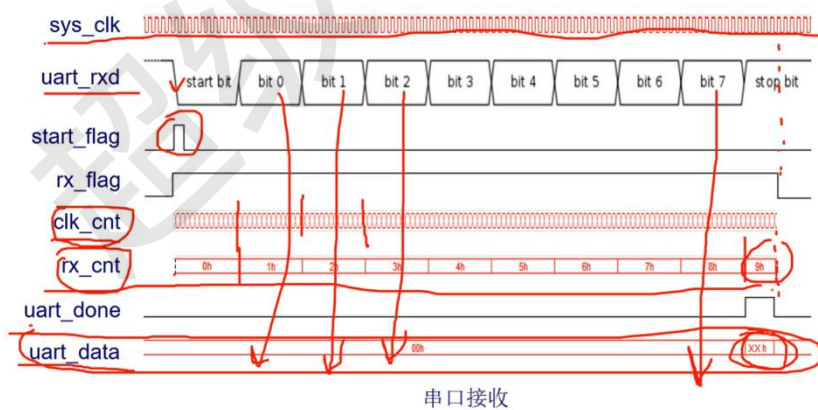
Cancel



串口

程序设计

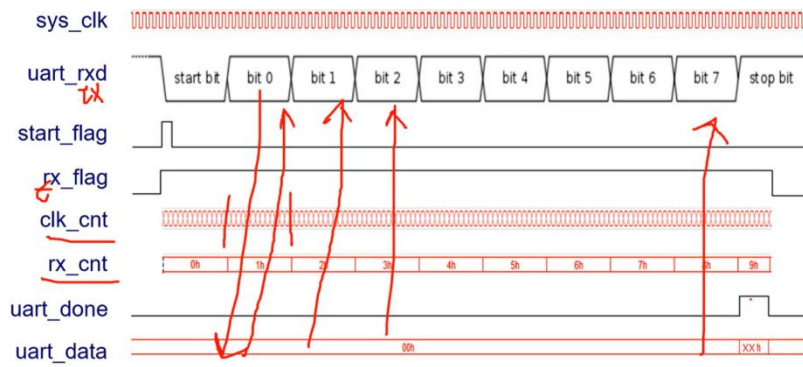
正点原子



串口接收

就是有效的一个巴比特数据

“原子哥”在线教学平台: www.yzhangzhe.com 支持论坛: www.openedv.com

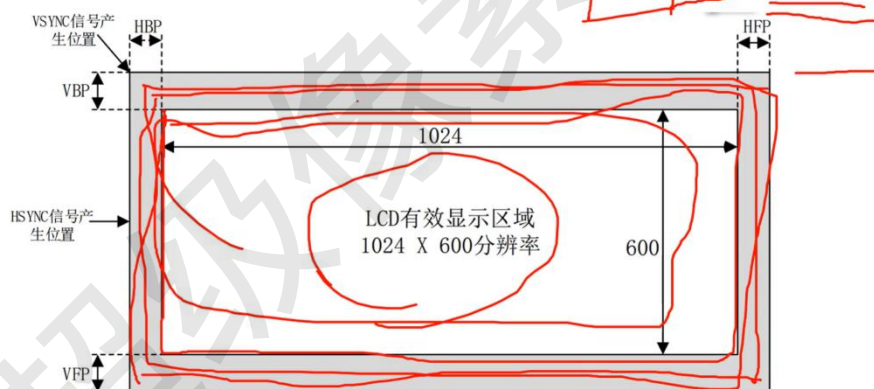


串口接收 / 发送

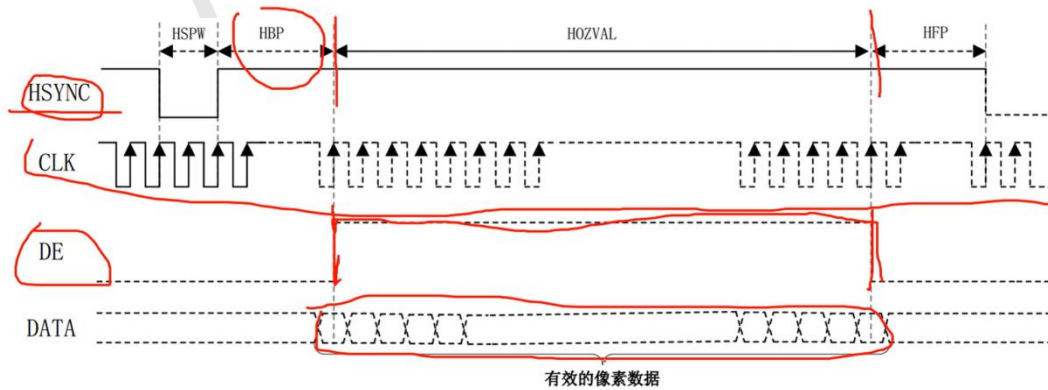
Lcd

显示彩条

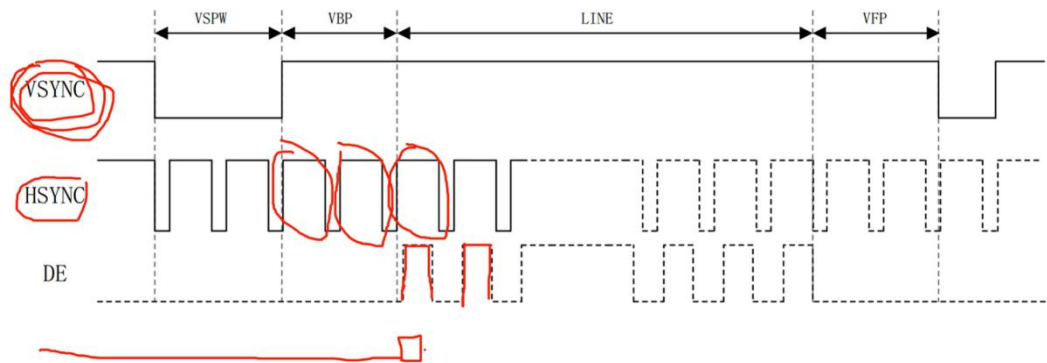
7' RGB LCD时间参数 (分辨率: 1024*600)



RGB LCD行显示时序

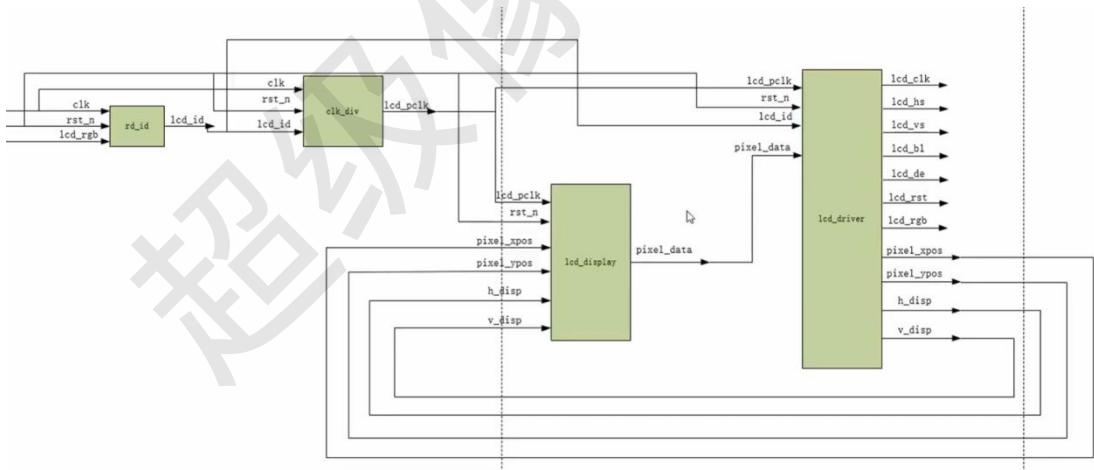


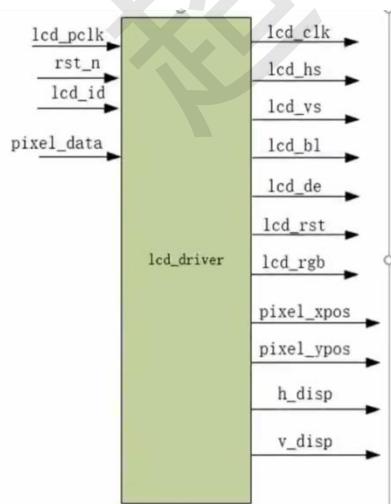
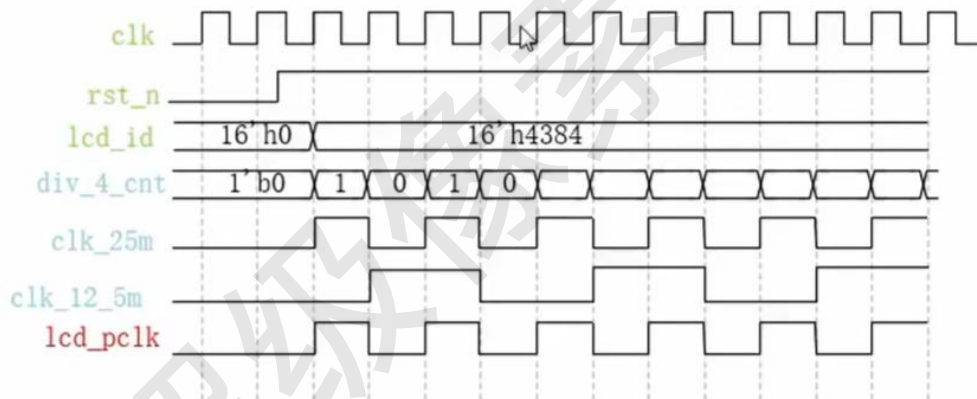
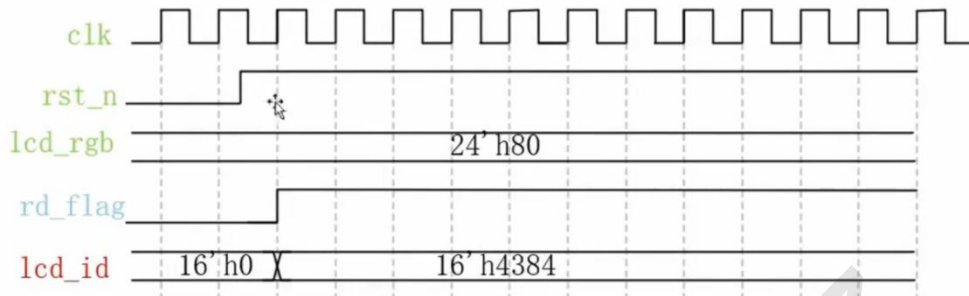
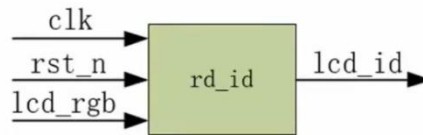
RGB LCD帧显示时序

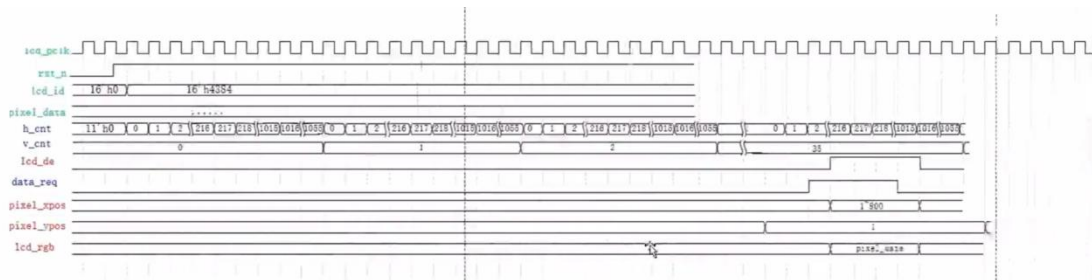


管脚说明

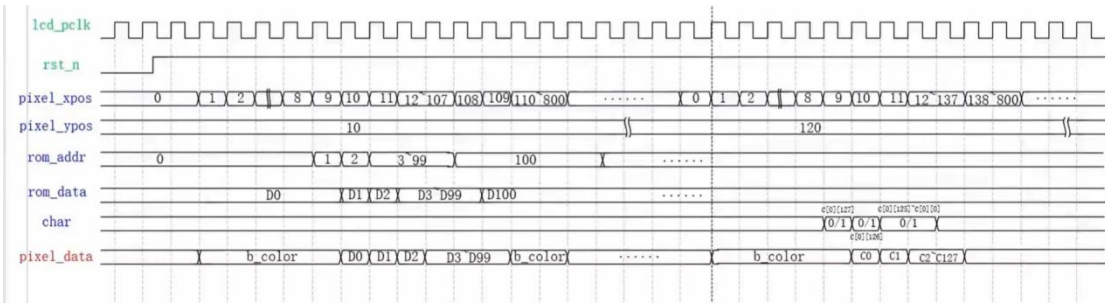
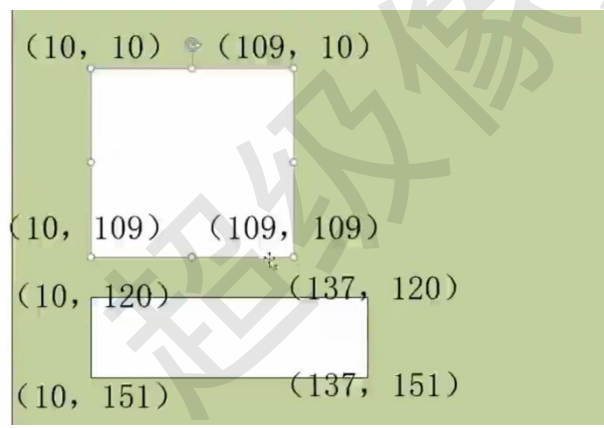
信号名	端口说明	信号名	端口说明
sys_clk	系统时钟	lcd_rgb[14]	LCD绿色
sys_rst_n	系统复位	lcd_rgb[13]	LCD绿色
lcd_de	LCD数据有效	lcd_rgb[12]	LCD绿色
lcd_hs	LCD行同步	lcd_rgb[11]	LCD绿色
lcd_vs	LCD场同步	lcd_rgb[10]	LCD绿色
lcd_clk	LCD像素时钟	lcd_rgb[9]	LCD绿色
lcd_bl	LCD背光控制	lcd_rgb[8]	LCD绿色
lcd_rgb[23]	LCD红色	lcd_rgb[7]	LCD蓝色
lcd_rgb[22]	LCD红色	lcd_rgb[6]	LCD蓝色
lcd_rgb[21]	LCD红色	lcd_rgb[5]	LCD蓝色
lcd_rgb[20]	LCD红色	lcd_rgb[4]	LCD蓝色
lcd_rgb[19]	LCD红色	lcd_rgb[3]	LCD蓝色
lcd_rgb[18]	LCD红色	lcd_rgb[2]	LCD蓝色
lcd_rgb[17]	LCD红色	lcd_rgb[1]	LCD蓝色
lcd_rgb[16]	LCD绿色	lcd_rgb[0]	LCD蓝色
lcd_rgb[15]	LCD绿色		

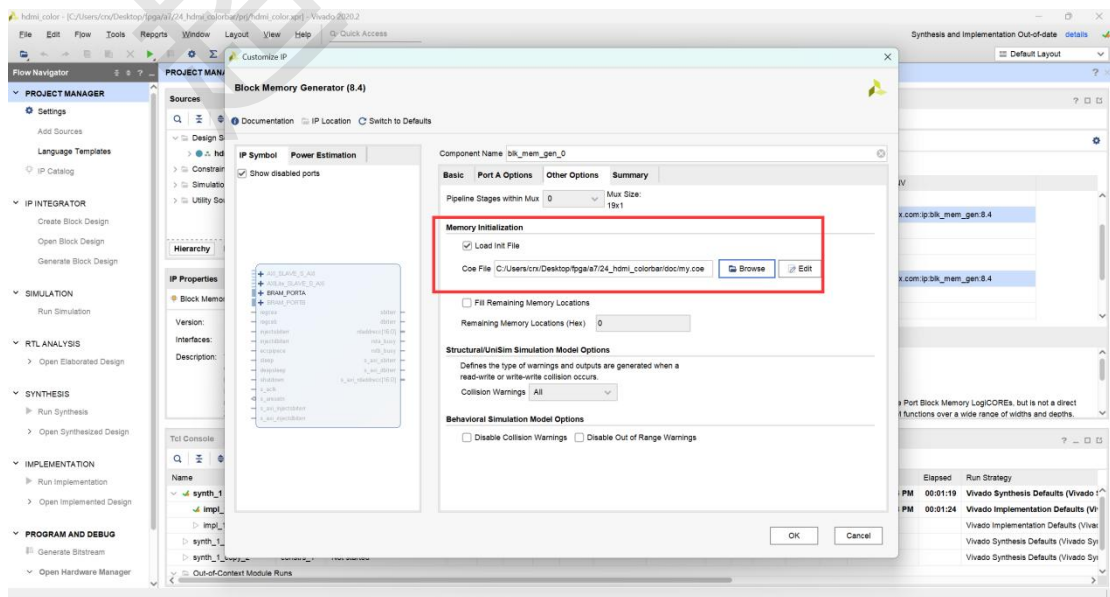
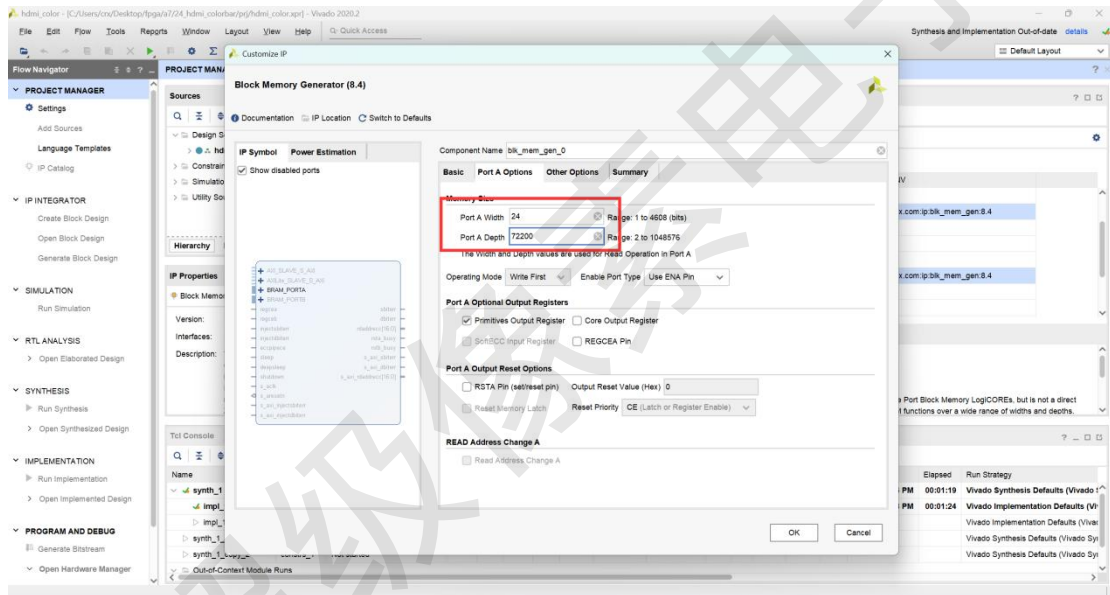
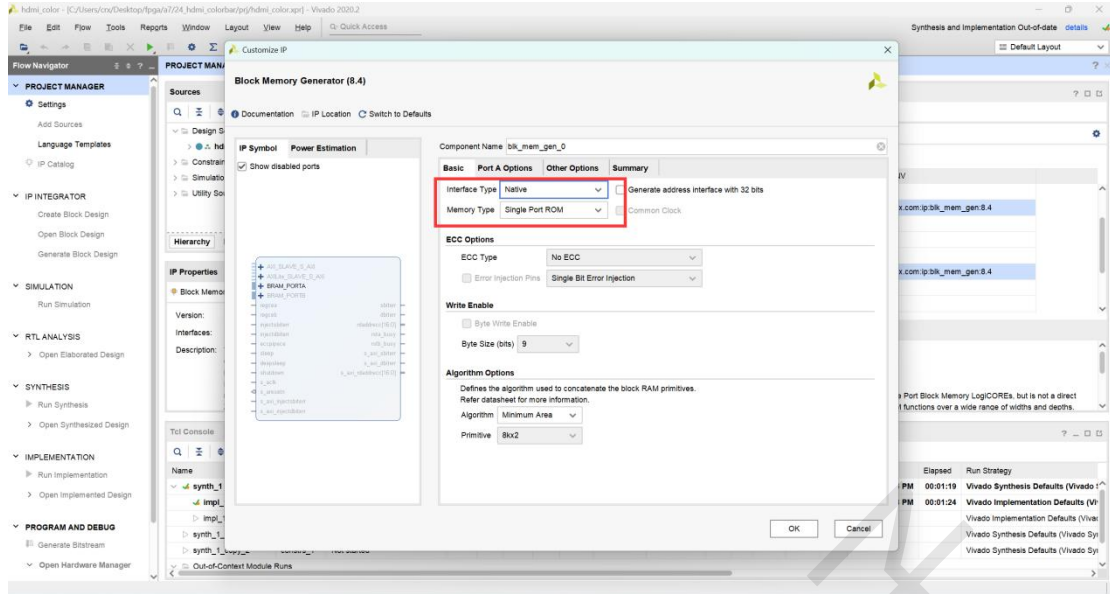






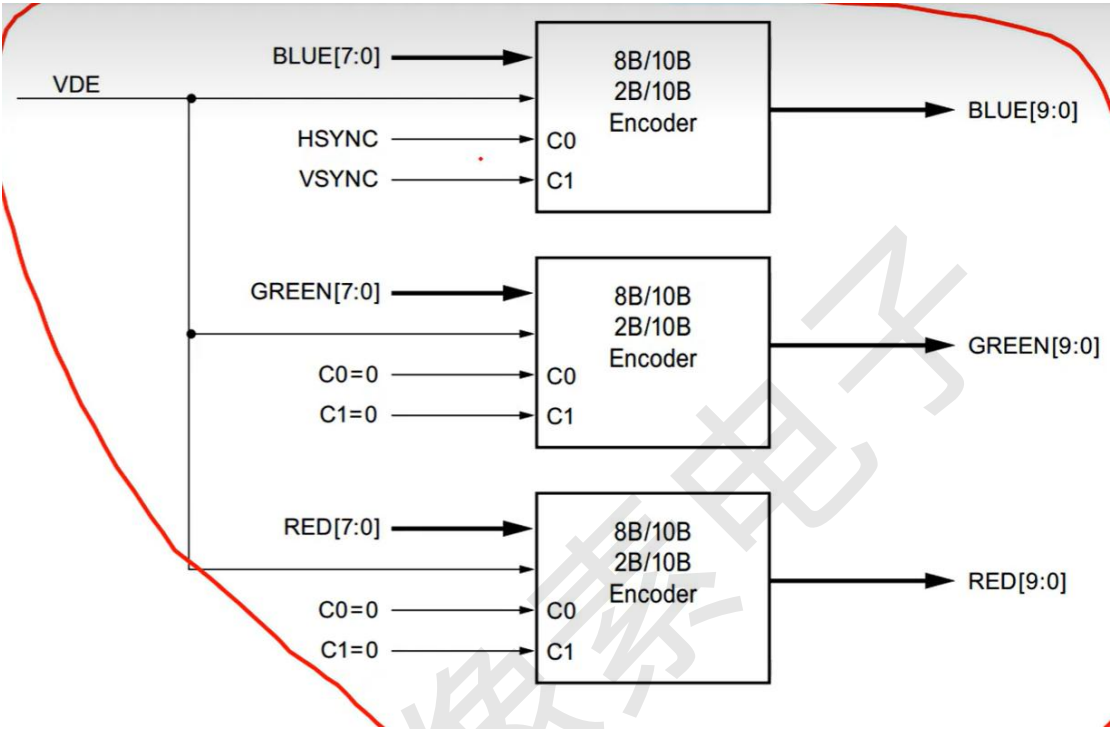
显示自定义图片



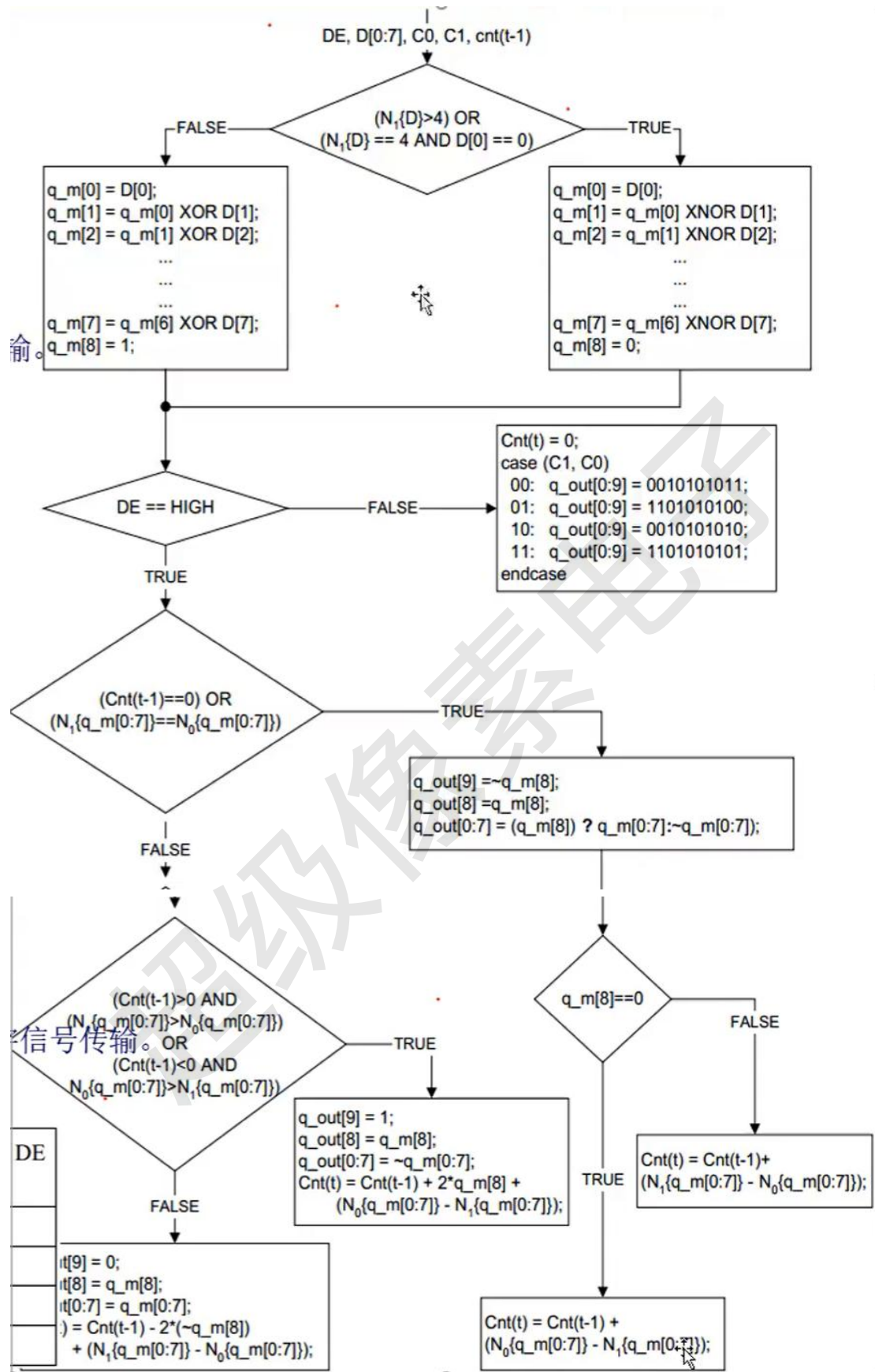


HDMI

TMDS 编码



D C1 C0 DE	D 为视频信号，C 为控制信号，DE 为使能信号
Cnt	Cnt 为寄存器参数
N1{X}	输入视频信号“1”的个数
N0{X}	输入视频信号“0”的个数
q_out	编码输出



OSERDESE2 串并转换

OSERDESE2结构示意图

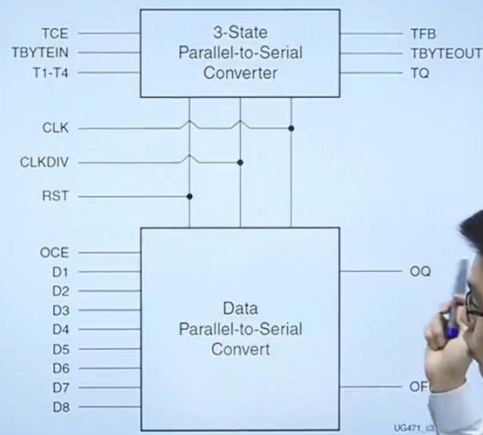
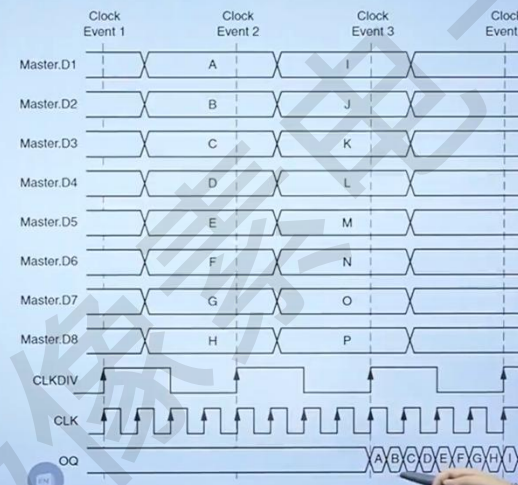


Figure 3-13: OSERDESE2 Block Diagram

OSERDESE2工作时序图



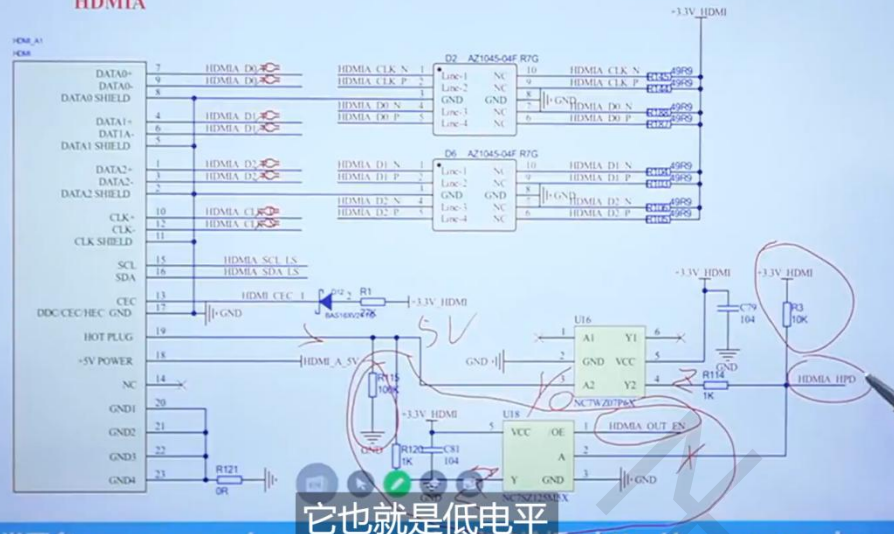
OSERDESE2三态时序图



Figure 3-14: OSERDESE2 Data Flow and Latency in

HDMI A接口 (达芬奇)

HDMI A

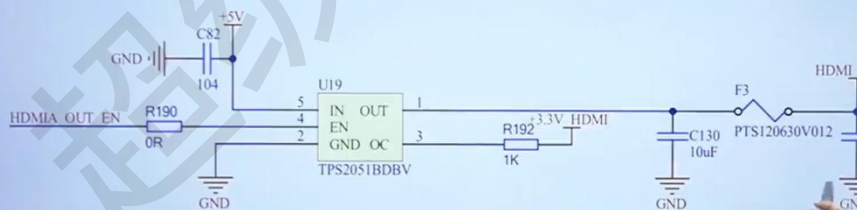


NC7WZ07P6

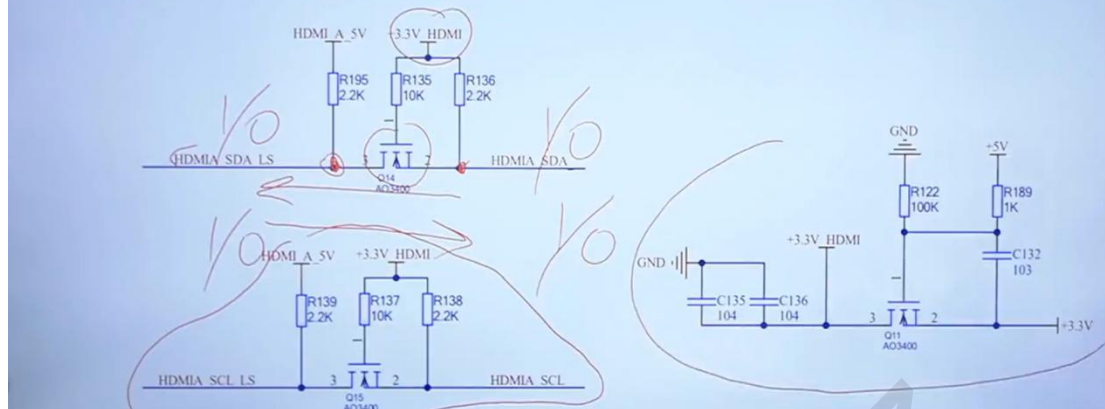
NC7SZ125M5X

A(INPUT)	Y(OUTPUT)	OE(INPUT)	A(INPUT)	Y(OUTPUT)
0	0	0	0	0
1	Z	0	1	1
		H	X	Z

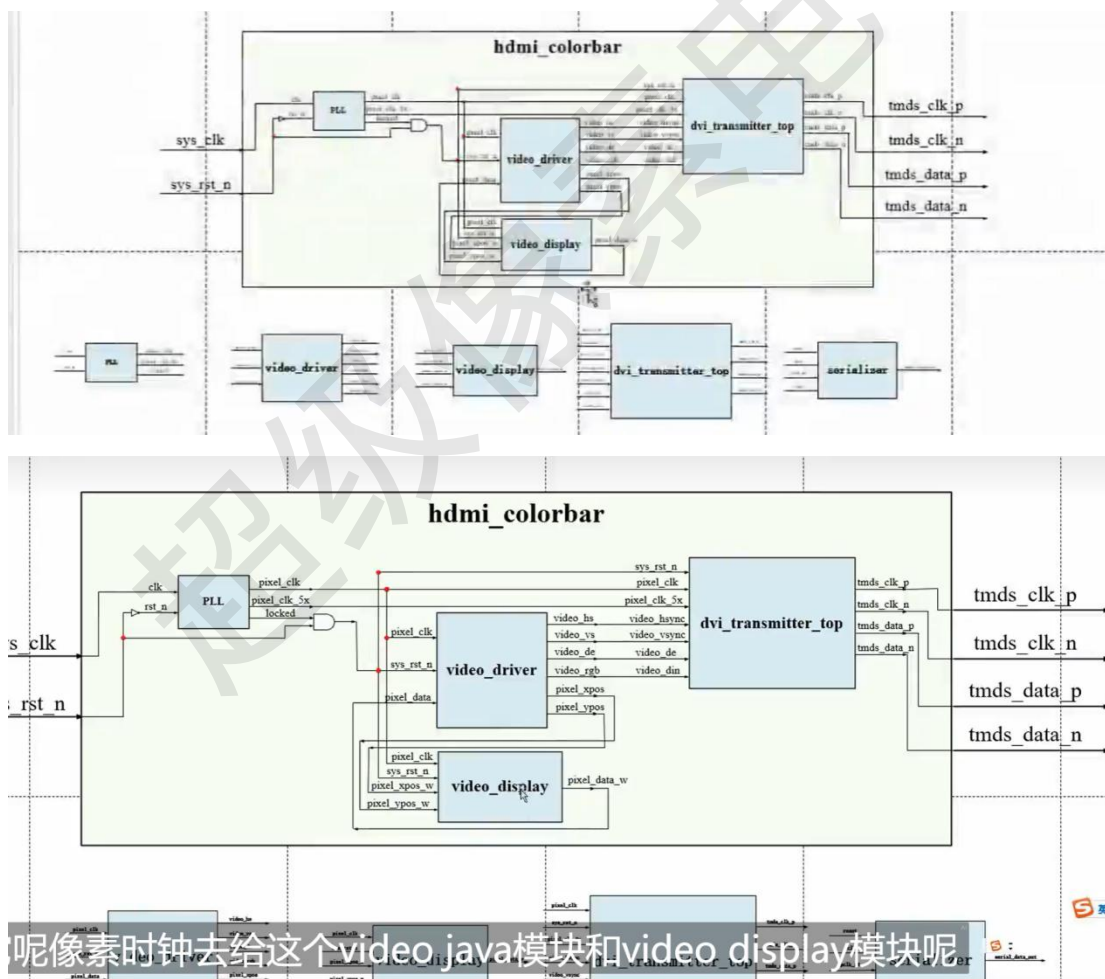
5V电源控制 (达芬奇)



电压转换电路 (达芬奇)



程序



引脚	功能	引脚	功能
A0	可编程地址引脚	VCC	电源引脚
A1	可编程地址引脚	WP	写保护高电平有效
A2	可编程地址引脚	SCL	IIC时钟引脚
GND	接地引脚	SDA	IIC数据引脚

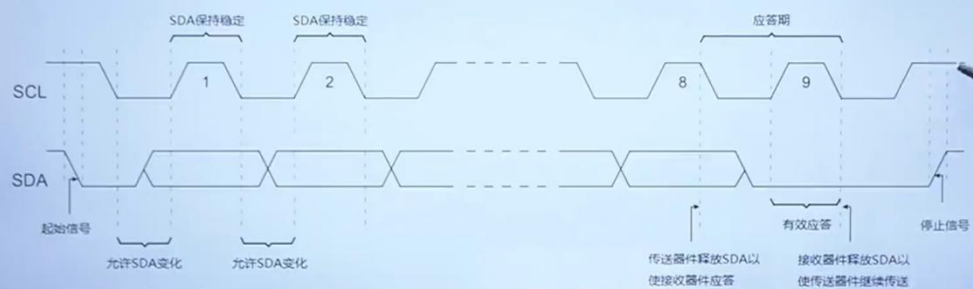
我们是不可以对这个

模式	描述
字节写入模式	按字节写入
页写入模式	按页写入
当前地址读模式	按照当前默认地址读数据
随机地址读模式	从指定的地址读数据
连续地址读模式	从起始地址连续读数据

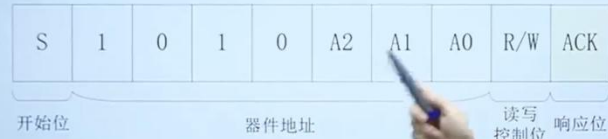
那么了解了读写模式之后

IIC

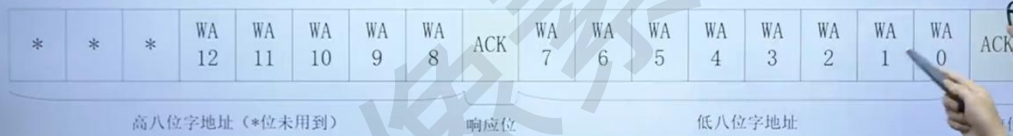
IIC协议



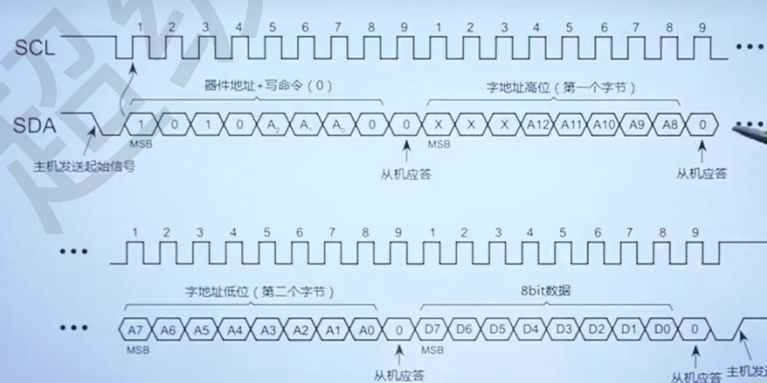
器件地址



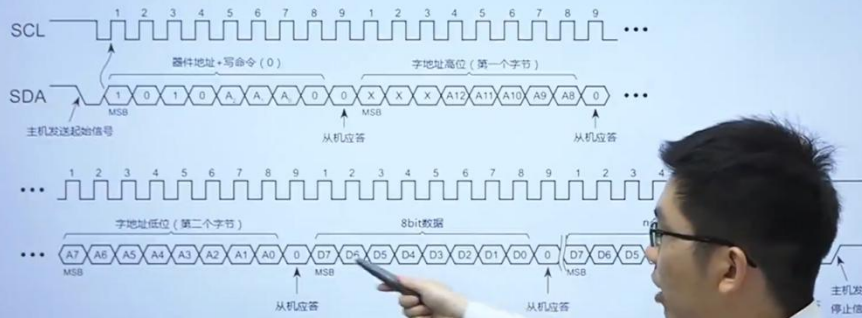
字（寄存器）地址



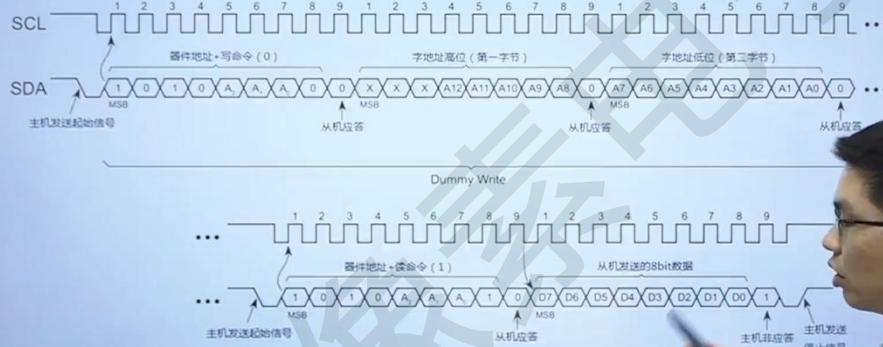
单次写



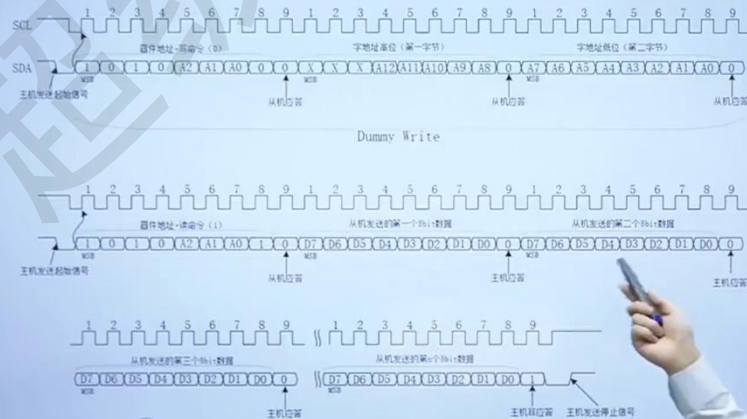
连续写



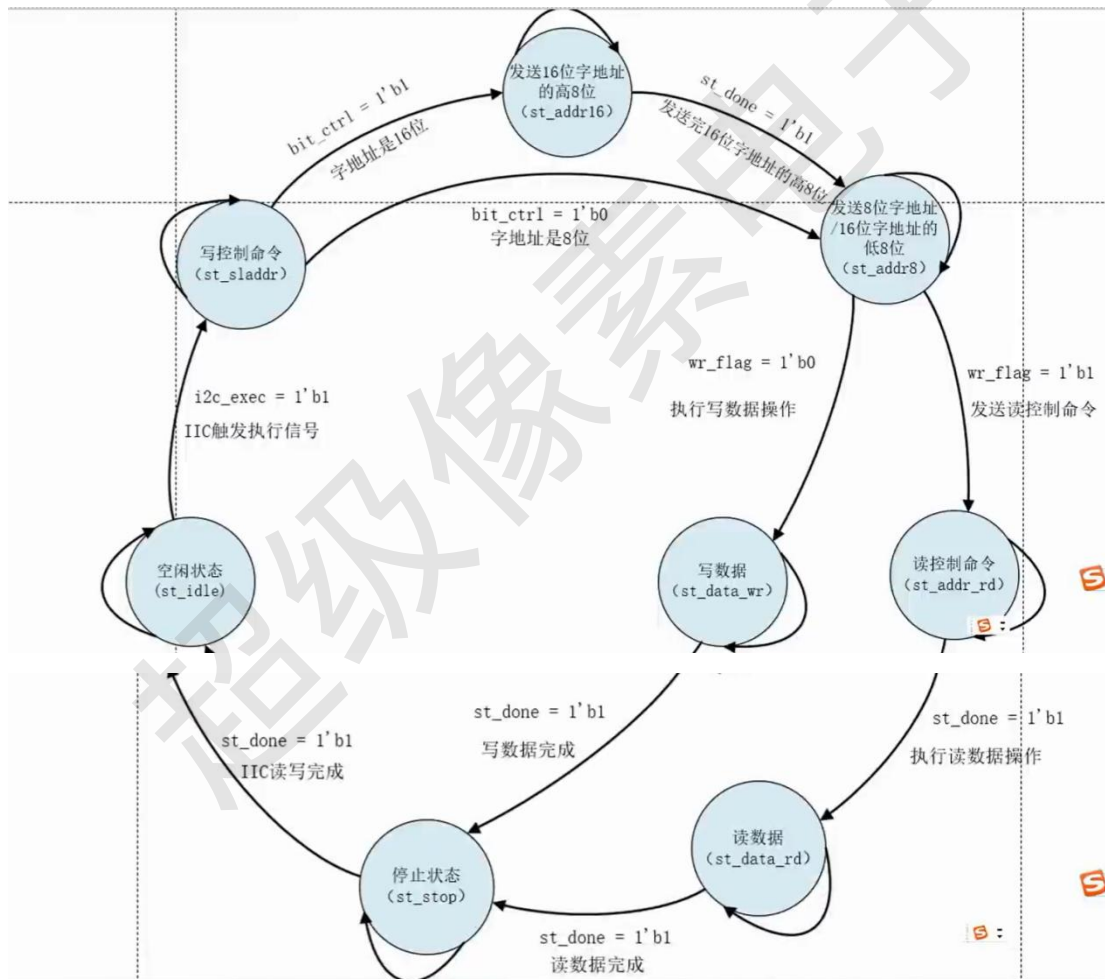
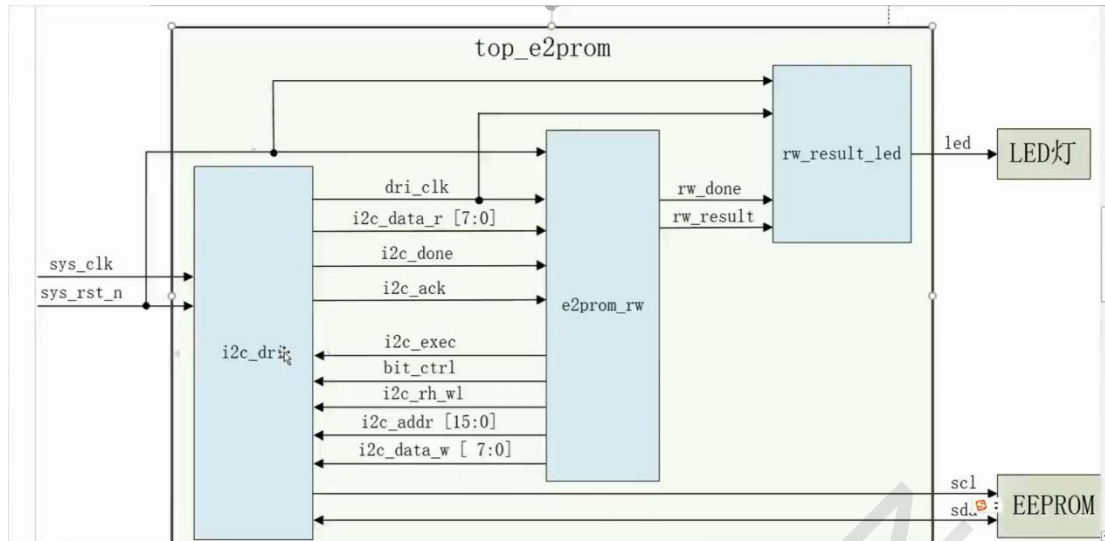
任意地址读

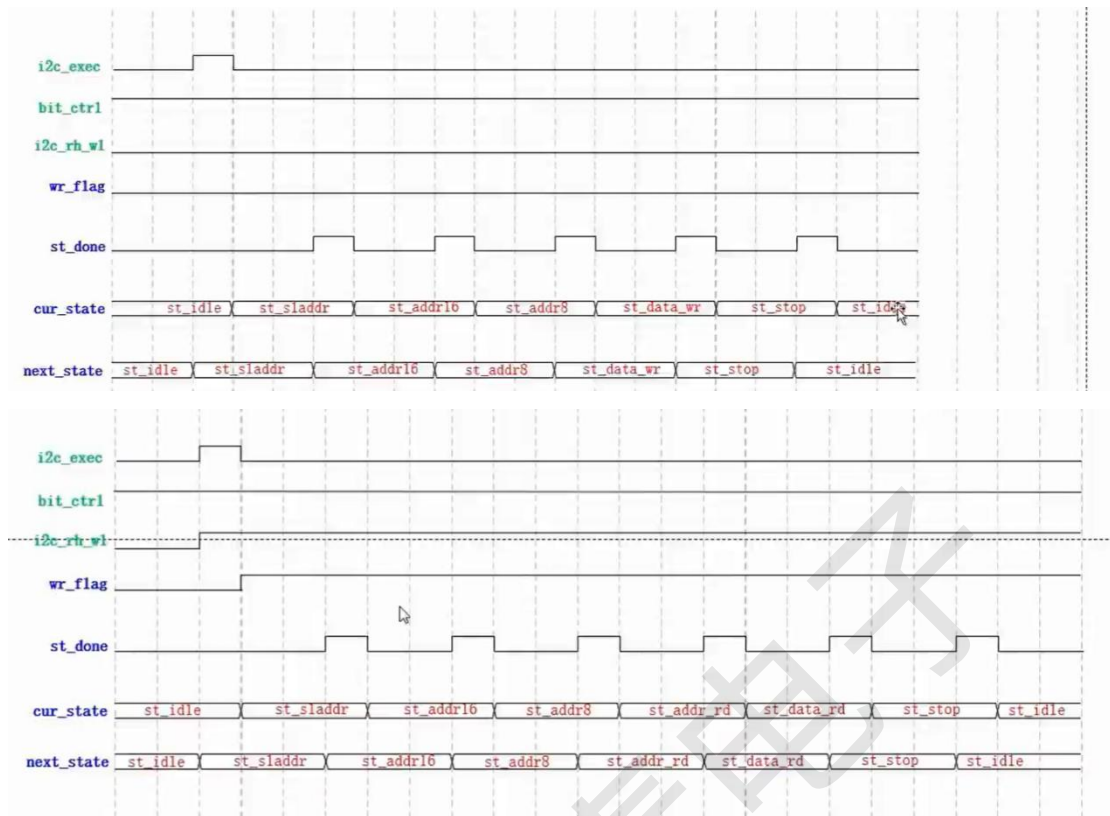


从xxx地址开始连续读

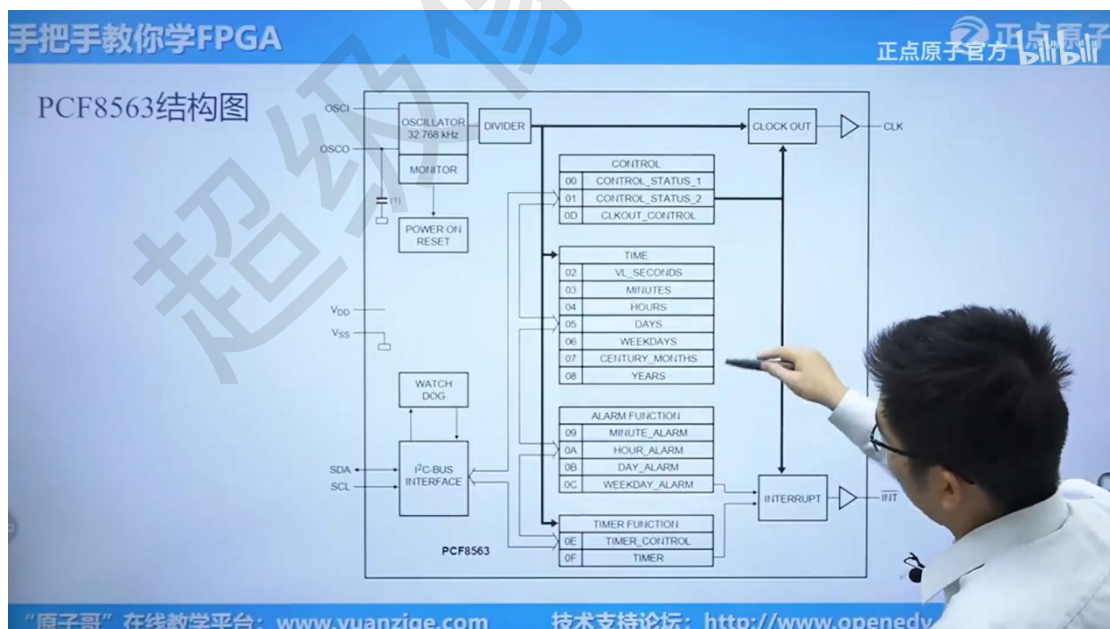


程序





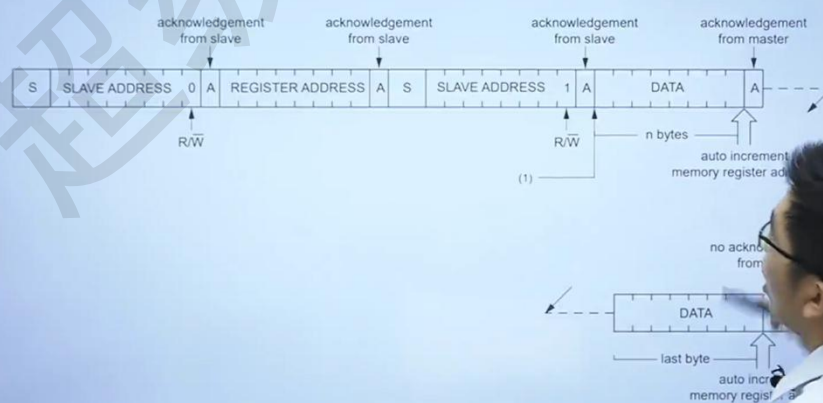
时钟模块

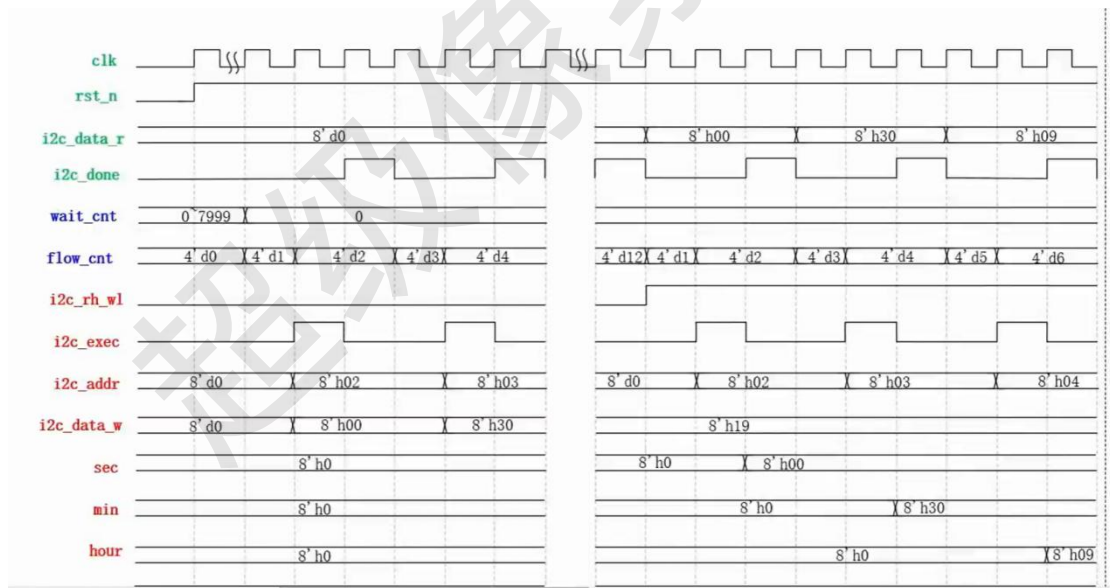
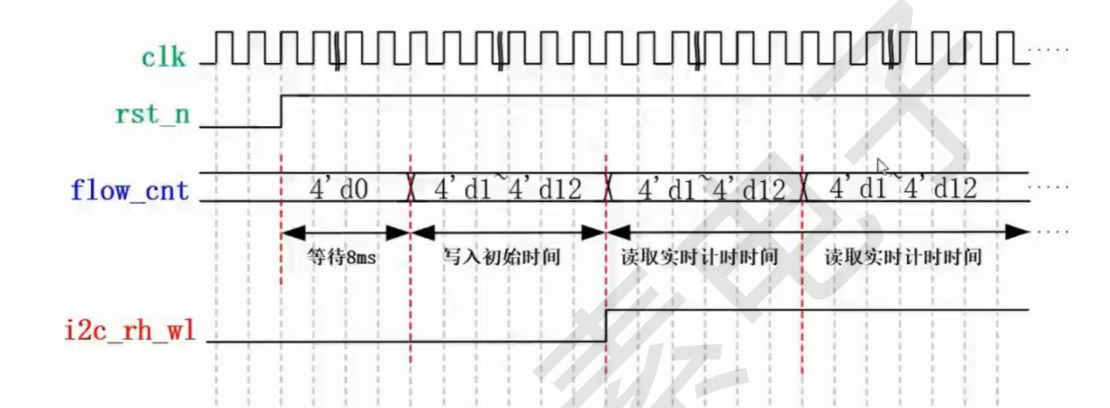
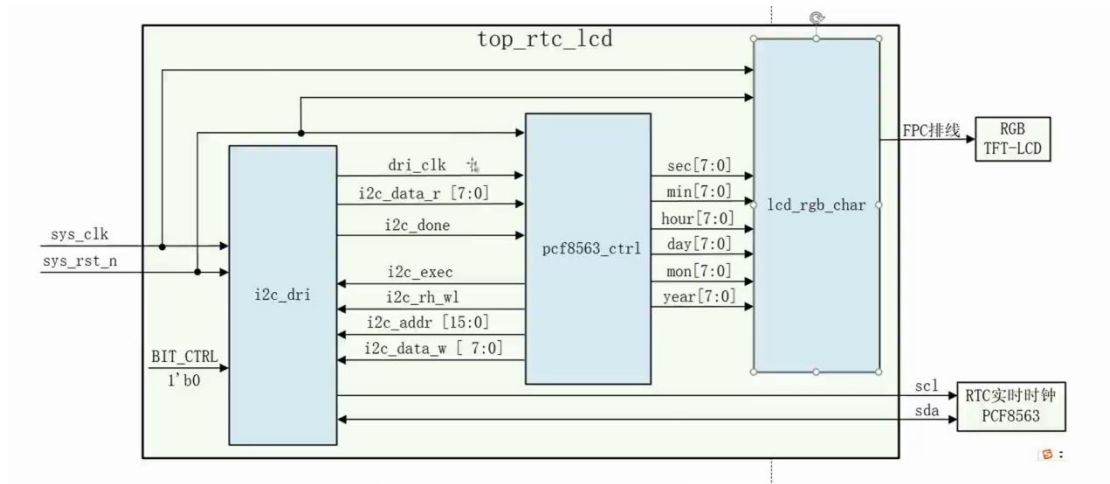


寄存器	BIT	符号	描述
02h	7	VL	VL=0保证准确的时钟/日历数据 VL=1不保证准确的时钟/日历数据
	6~0	秒	用BCD格式表示的秒数值
03h	7	-	无效
	6~0	分	用BCD格式表示的分钟值
04h	7~6	-	无效
	5~0	时	用BCD格式表示的小时值
05h	7~6	-	无效
	5~0	天	用BCD格式表示的天数值

寄存器	BIT	符号	描述
06h	7~3	-	无效
	2~0	星期	一周的数值0~6
07h	7	C	当C为0时表明是当前世纪，为1时表明是
	6~5	-	未用
	4~0	月	用BCD格式表示的月数值
08h	7~0	年	用BCD格式表示的当前年数值，值

读时序






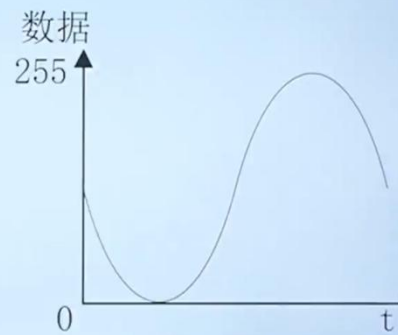
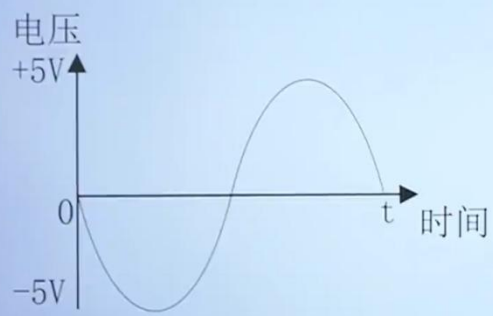
ADC/DAC

ADC

AD9280模拟数字转换关系



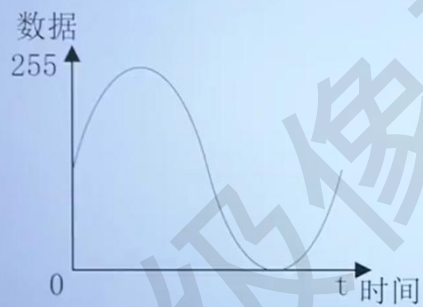
The figure consists of two side-by-side graphs. The left graph plots '电压' (Voltage) on the y-axis against '时间' (Time) on the x-axis. The y-axis has labels for +5V, 0, and -5V. A sine wave starts at 0, goes down to -5V, crosses 0, and goes up to +5V. The x-axis is labeled 't' at the end. The right graph plots '数据' (Data) on the y-axis against '时间' (Time) on the x-axis. The y-axis has labels for 255 and 0. A curve starts at a value between 0 and 255, dips to 0, then rises to a peak near 255 before dipping again. The x-axis is labeled 't' at the end.



DAC

AD9708数字模拟转换关系

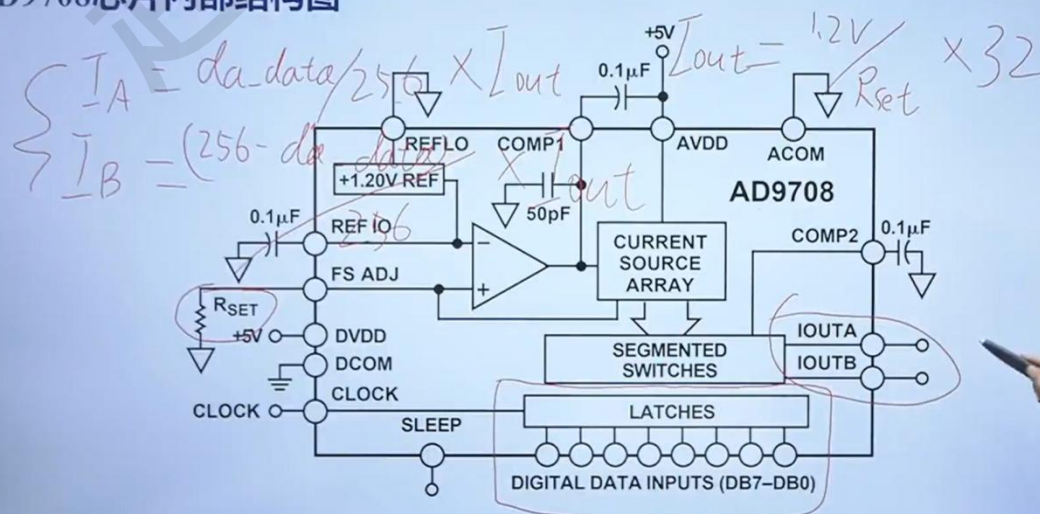
The image contains two side-by-side graphs. The left graph has a vertical axis labeled '数据' (Data) with tick marks at 0 and 255, and a horizontal axis labeled '时间' (Time) with a tick mark at 't'. It shows a sine wave starting at a positive value, peaking, crossing zero, reaching a minimum, and returning towards zero. The right graph has a vertical axis labeled '电压' (Voltage) with tick marks at +5V, 0, and -5V, and a horizontal axis labeled '时间' (Time) with a tick mark at 't'. It shows a sine wave starting at zero, reaching a minimum, crossing zero, reaching a maximum, and returning towards zero.

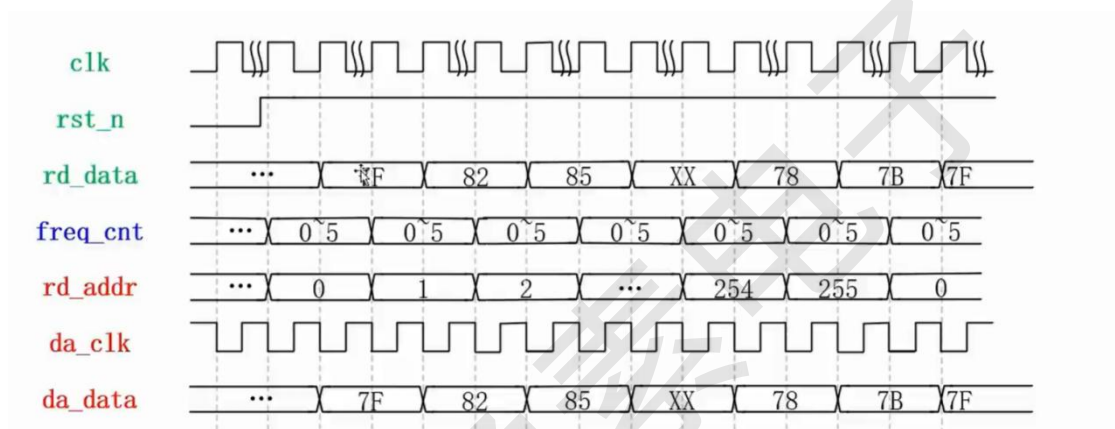
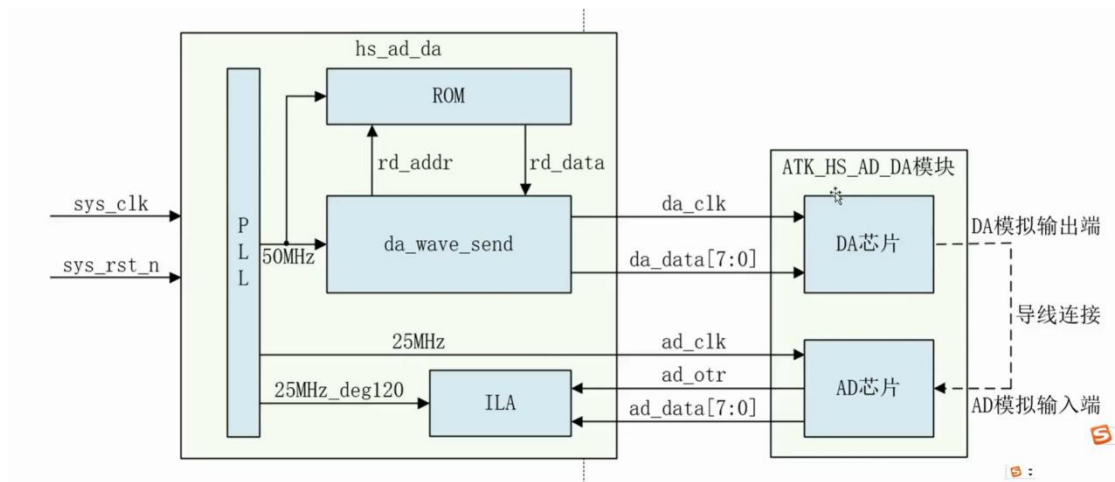


AD9708芯片内部结构图

Handwritten notes on the diagram:

- $I_A = \frac{\text{data}}{256} \times I_{out}$
- $I_B = \frac{(256 - \text{data})}{256} \times I_{out}$
- $I_{out} = \frac{1.2V}{R_{set}} \times 32$

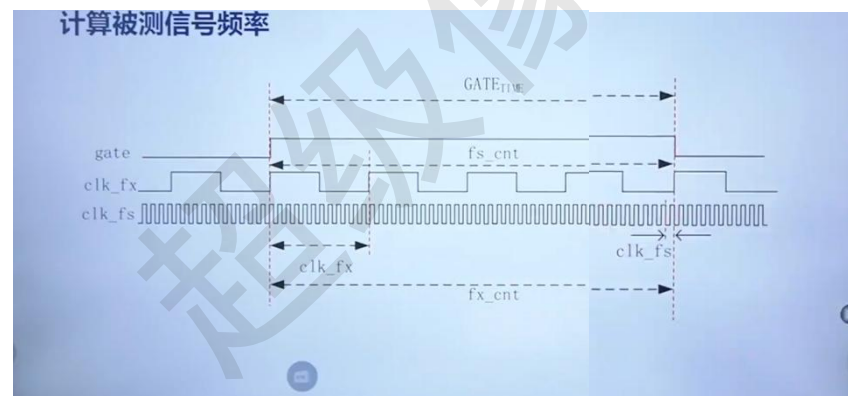






频率测量

原理



等精度计算公式

$$GATE_{TIME} = \frac{fs_cnt}{clk_{fs}} = \frac{fx_cnt}{clk_{fx}}$$

$$clk_{fx} = \frac{fx_cnt}{fs_cnt} clk_{fs}$$

误差分析

$$\delta = \frac{\overset{\text{理}}{clk_{fxe}} - \overset{\text{实}}{clk_{fx}}}{clk_{fxe}} \times 100\%$$

误差分析

$$clk_{fxe} = \frac{fx_{cnt}}{fs_{cnt} + \Delta fs_{cnt}} \times CLK_{FS}$$

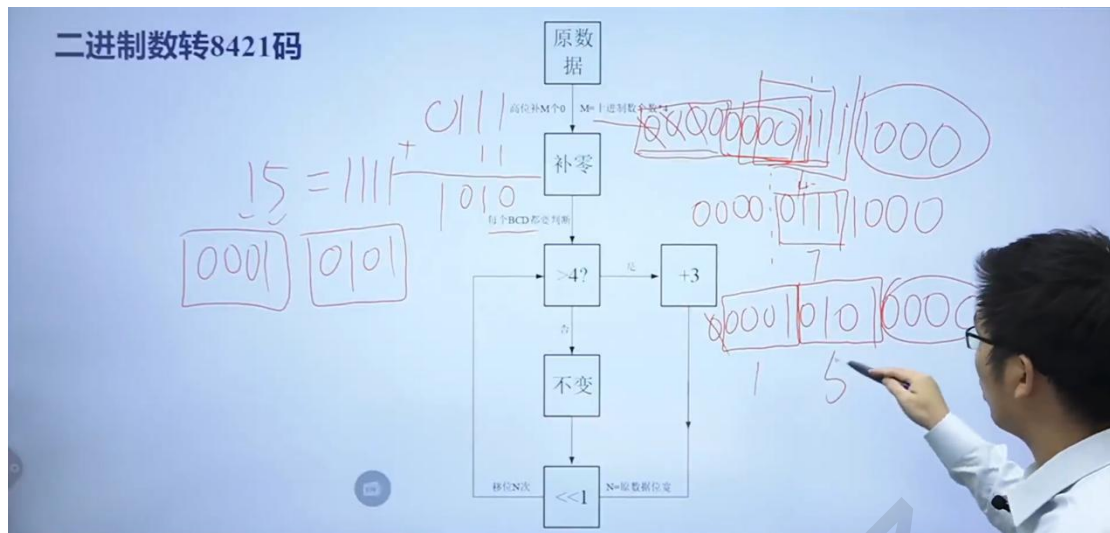
$$\delta = \left| \frac{\Delta fs_{cnt}}{fs_{cnt}} \right| \leq \frac{1}{fs_{cnt}} = \frac{1}{GATE_{TIME} \times CLK_{FS}}$$

误差分析

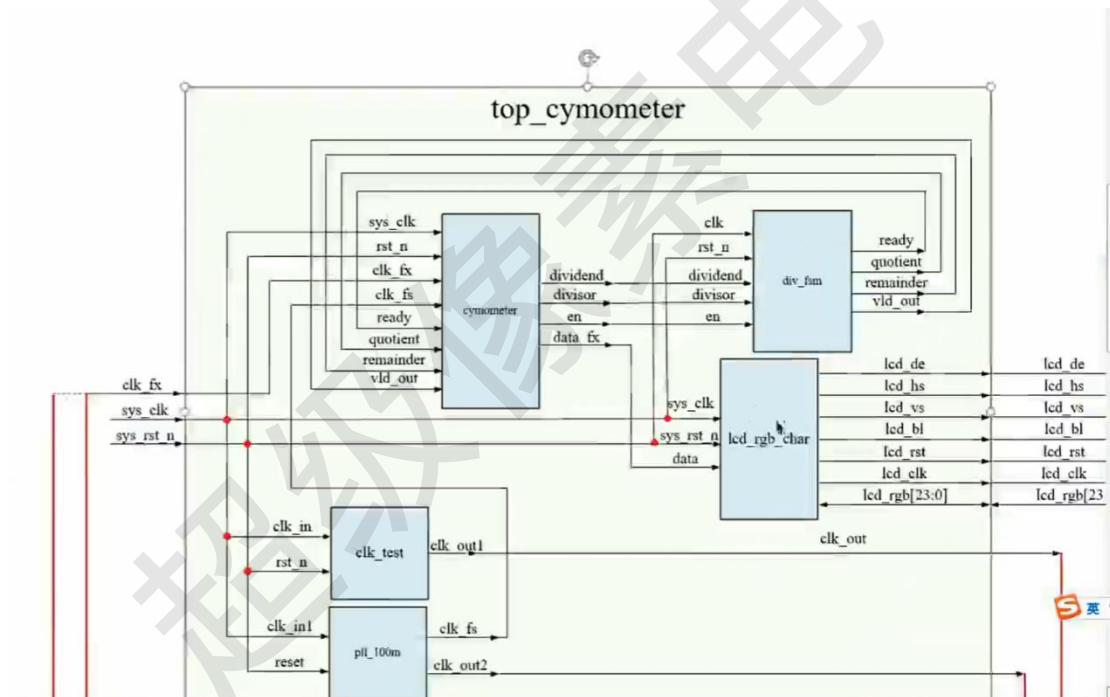
$$clk_{fxe} = \frac{fx_{cnt}}{fs_{cnt} + \Delta fs_{cnt}} \times CLK_{FS}$$

$$\delta = \left| \frac{\Delta fs_{cnt}}{fs_{cnt}} \right| \leq \frac{1}{fs_{cnt}} = \frac{1}{GATE_{TIME} \times CLK_{FS}}$$

BCD 码转换



程序



Sdram

初识SDRAM

正点原子

物理 Bank: 传统内存系统为了保证 CPU 的正常工作, 必须一次传输完 CPU 在一个传输周期内所需要的数据。而 CPU 在一个传输周期能接受的数据容量就是 CPU 数据总线的位宽, 单位是 bit (位)。当时控制内存与 CPU 之间数据交换的北桥芯片也因此将内存总线的数据位宽等同于 CPU 数据总线的位宽, 而这个位宽就称之为物理 Bank (Physical Bank) 的位宽。

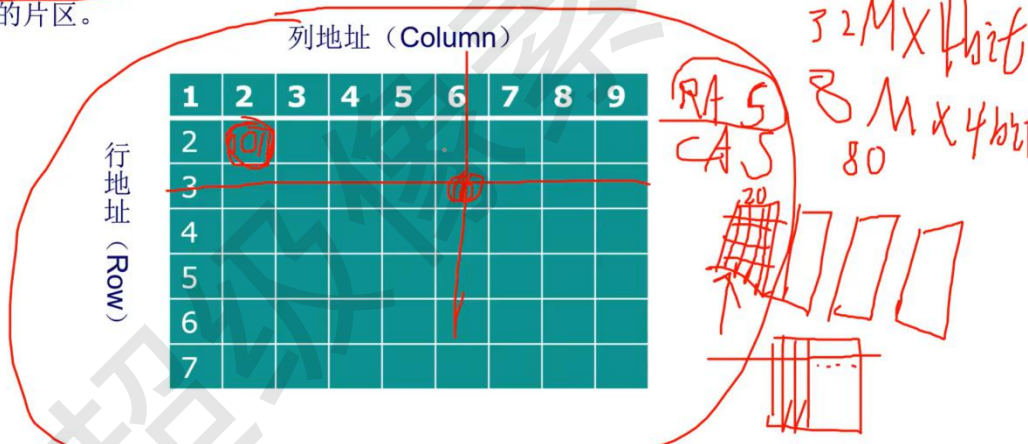
芯片位宽: 每一片 SDRAM 缓存芯片本身的位宽。

64 bit p.b
16 bit SDRAM

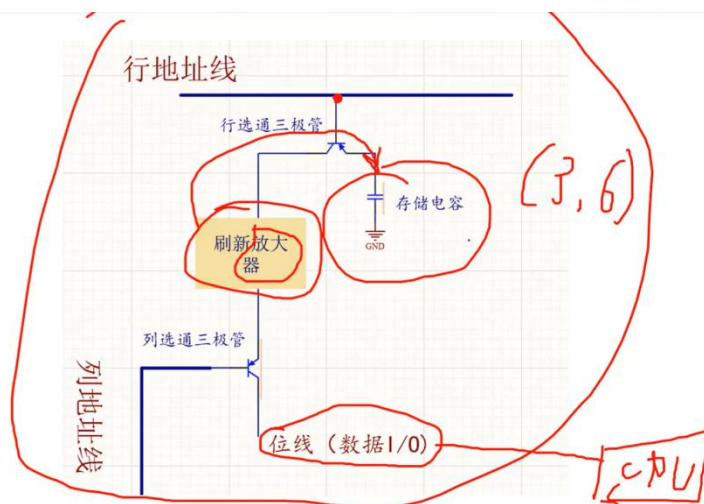
“原子哥”在线教学平台: www.yuanzige.com

技术支持论坛: www.openedv.com

逻辑 Bank: (Logical Bank, 下文简称 L-Bank) SDRAM 内部存储空间划分的片区。

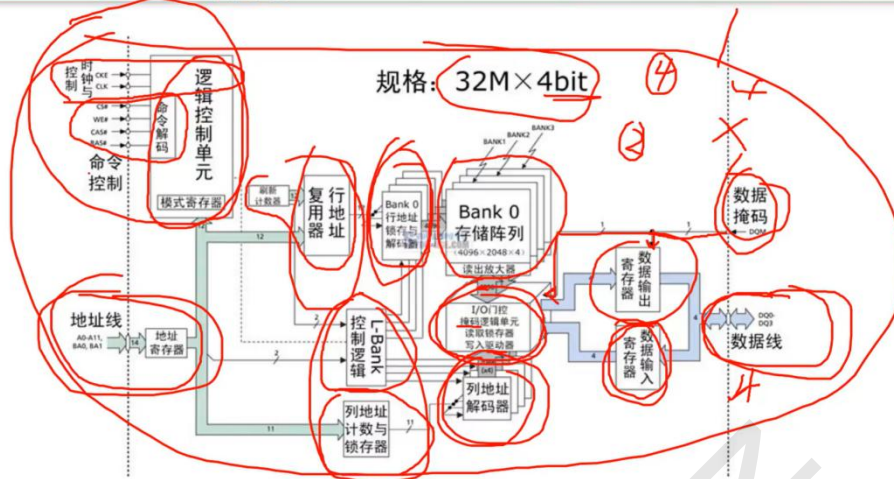


存储原理示意图
(注: 示意作用不代表内部实际电路)



SDRAM基本结构

正点原子

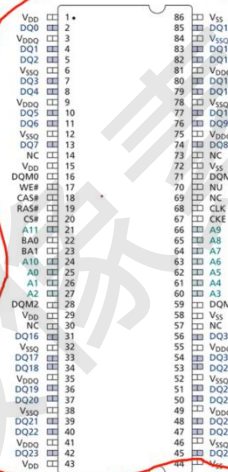


“原子哥”在线教学平台: www.yuanzige.com

技术支持论坛: www.openedv.com

SDRAM操作时序

正点原子



“原子哥”在线教学平台: www.yuanzige.com

技术支持论坛: www.openedv.com

Symbol	Type	Description
<u>CLK</u>	Input	Clock: CLK is driven by the system clock. All SDRAM input signals are sampled on the positive edge of CLK. CLK also increments the internal burst counter and controls the output registers.
<u>CKE</u>	Input	Clock enable: CKE activates (HIGH) and deactivates (LOW) the CLK signal. Deactivating the clock provides precharge power-down and SELF REFRESH operation (all banks idle), active power-down (row active in any bank), or CLOCK SUSPEND operation (burst/access in progress). CKE is synchronous except after the device enters power-down and self refresh modes, where CKE becomes asynchronous until after exiting the same mode. The input buffers, including CLK, are disabled during power-down and self refresh modes, providing low standby power. CKE may be tied HIGH.
CS#	Input	Chip select: CS# enables (registered LOW) and disables (registered HIGH) the command decoder. All commands are masked when CS# is registered HIGH, but READ/WRITE bursts already in progress will continue, and DQM operation will retain its DQ mask capability while CS# is HIGH. CS# provides for external bank selection on systems with multiple banks. CS# is considered part of the command code.
<u>CAS#, RAS#, WE#</u>	Input	Command inputs: CAS#, RAS#, and WE# (along with CS#) define the command being entered.
<u>DQM[3:0]</u>	Input	Input/output mask: DQM is sampled HIGH and is an input mask signal for write accesses and an output enable signal for read accesses. Input data is masked during a WRITE cycle. The output buffers are High-Z (two-clock latency) during a READ cycle. DQM0 corresponds to DQ[7:0], DQM1 corresponds to DQ[15:8], DQM2 corresponds to DQ[23:16], and DQM3 corresponds to DQ[31:24]. DQM[3:0] are considered the same state when referenced as DQM.
<u>BA[1:0]</u>	Input	Bank address inputs: BA[1:0] define to which bank the ACTIVE, READ, WRITE, or PRECHARGE command is being applied.
<u>A[11:0]</u>	Input	Address inputs: A[11:0] are sampled during the ACTIVE command (row address A[10:0]) and READ or WRITE command (column address A[7:0] with A10 defining auto precharge) to select one location out of the memory array in the respective bank. A10 is sampled during a PRECHARGE command to determine if all banks are to be precharged (A10 HIGH) or bank selected by BA[1:0] (LOW). The address inputs also provide the op-code during a LOAD MODE REGISTER command.
<u>DQ[31:0]</u>	Input/Output	Data input/output: Data bus.
NC	-	No connect: These pins should be left unconnected. Pin 70 is reserved for SSTL reference voltage supply.
<u>V_{DDQ}</u>	Supply	DQ power supply: Isolated on the die for improved noise immunity.
<u>V_{SSQ}</u>	Supply	DQ ground: Provides isolated ground to DQs for improved noise immunity.
<u>V_{DD}</u>	Supply	Power supply: 3.3V \pm 0.3V.
<u>V_{SS}</u>	Supply	Ground.

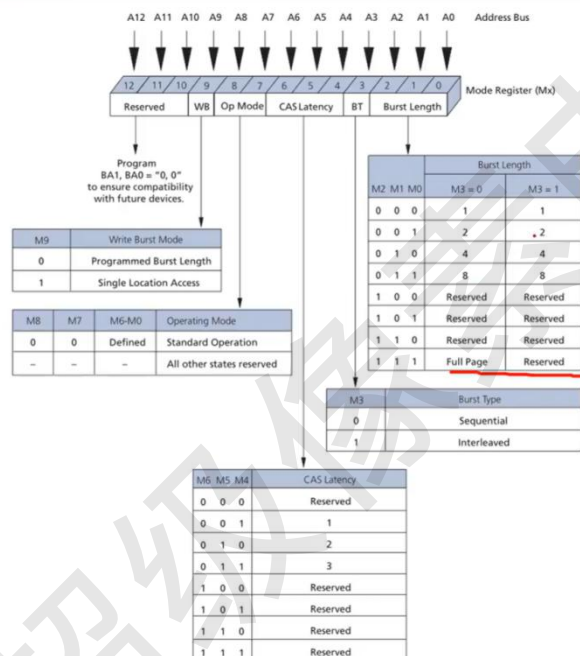
Table 14: Truth Table – Commands and DQM Operation

Note 1 applies to all parameters and conditions

Name (Function)	CS#	RAS#	CAS#	WE#	DQM	ADDR	DQ	Notes
<u>COMMAND INHIBIT (NOP)</u>	H	X	X	X	X	X	X	
<u>NO OPERATION (NOP)</u>	L	H	H	H	X	X	X	
<u>ACTIVE (select bank and activate row)</u>	L	L	H	H	X	Bank/row	X	2
<u>READ (select bank and column, and start READ burst)</u>	L	H	L	H	L/H	Bank/col	X	3
<u>WRITE (select bank and column, and start WRITE burst)</u>	L	H	L	L	L/H	Bank/col	Valid	3
<u>BURST TERMINATE</u>	L	H	H	L	X	X	Active	4
<u>PRECHARGE (Deactivate row in bank or banks)</u>	L	L	H	L	X	Code	X	(5)
<u>AUTO REFRESH or SELF REFRESH (enter self refresh mode)</u>	L	L	L	H	X	X	X	6, 7
<u>LOAD MODE REGISTER</u>	L	L	L	L	X	Op-code	X	8
<u>Write enable/output enable</u>	X	X	X	X	L	X	Active	9
<u>Write inhibit/output High-Z</u>	X	X	X	X	H	X	High-Z	9

70% * 60 ms

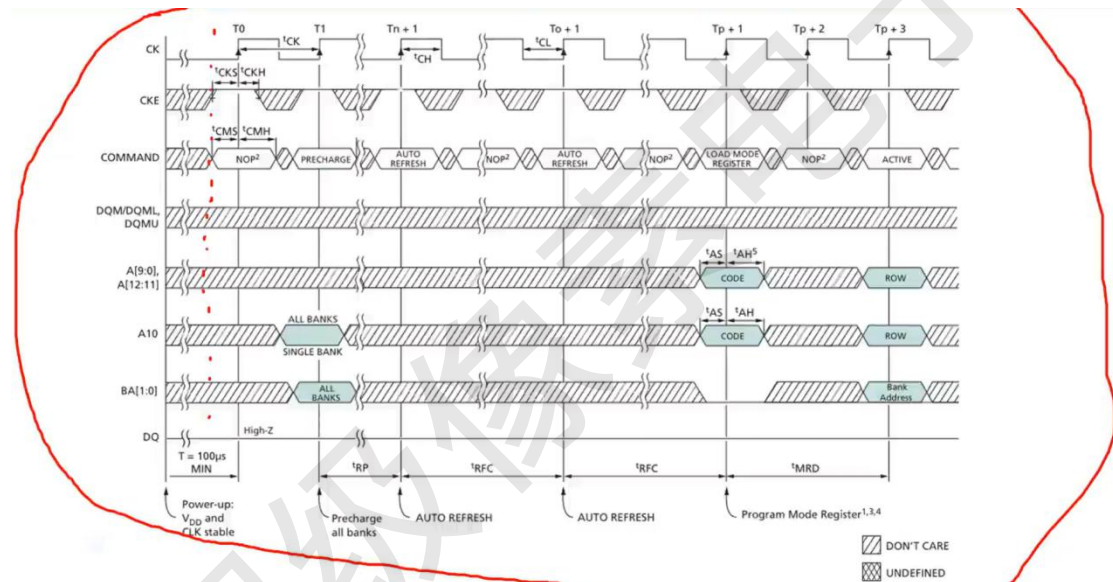
- Notes:
1. CKE is HIGH for all commands shown except SELF REFRESH.
 2. A[0:n] provide row address (where A_n is the most significant address bit), BA0 and BA1 determine which bank is made active.
 3. A[0:i] provide column address (where i = the most significant column address for a given device configuration). A10 HIGH enables the auto precharge feature (nonpersistent), while A10 LOW disables the auto precharge feature. BA0 and BA1 determine which bank is being read from or written to.
 4. The purpose of the BURST TERMINATE command is to stop a data burst, thus the command could coincide with data on the bus. However, the DQ column reads a "Don't Care" state to illustrate that the BURST TERMINATE command can occur when there is no data present.
 5. A10 LOW: BA0, BA1 determine the bank being precharged. A10 HIGH: all banks precharged and BA0, BA1 are "Don't Care."
 6. This command is AUTO REFRESH if CKE is HIGH, SELF REFRESH if CKE is LOW.
 7. Internal refresh counter controls row addressing; all inputs and I/Os are "Don't Care" except for CKE.
 8. A[11:0] define the op-code written to the mode register.
 9. Activates or deactivates the DQ during WRITES (zero-clock delay) and READs (two-clock delay).



Burst Length	Starting Column Address		Order of Accesses Within a Burst	
			Type = Sequential	Type = Interleaved
<u>2</u>		A0		
		0	0-1	0-1
		1	1-0	1-0
<u>4</u>	A1	A0		
		0	0-1-2-3	0-1-2-3
		0	1-2-3-0	1-0-3-2
		1	2-3-0-1	2-3-0-1
		1	3-0-1-2	3-2-1-0
<u>8</u>	A2	A1	A0	
		0	0	0-1-2-3-4-5-6-7
		0	0	1-2-3-4-5-6-7-0
		0	1	2-3-4-5-6-7-0-1
		0	1	3-4-5-6-7-0-1-2
		1	0	4-5-6-7-0-1-2-3
		1	0	5-6-7-0-1-2-3-4
		1	1	6-7-0-1-2-3-4-5
		1	1	7-0-1-2-3-4-5-6
Continuous	n = A0-An/9/8 (location 0-y)		Cn, Cn + 1, Cn + 2, Cn + 3...Cn - 1, Cn...	Not supported

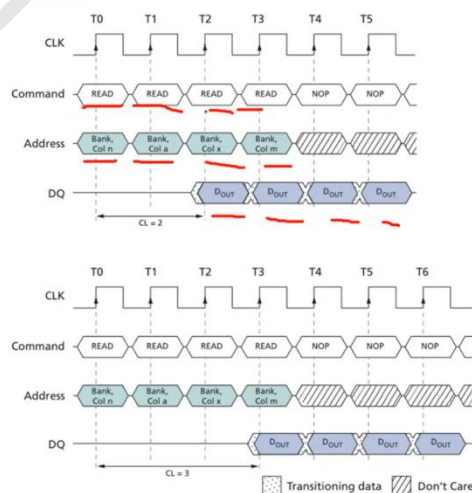
0 1 2 3

初始化时序



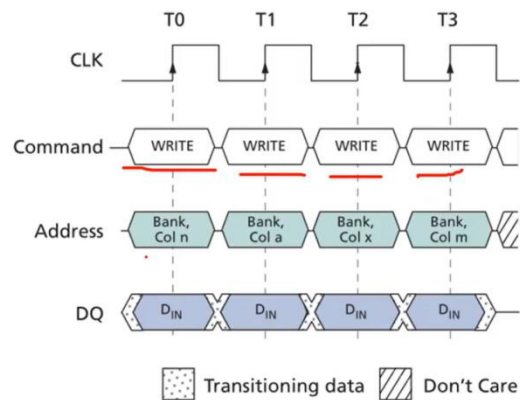
随机读写

Figure 17: Random READ Accesses



Note: 1. Each READ command can be issued to any bank. DQM is LOW.

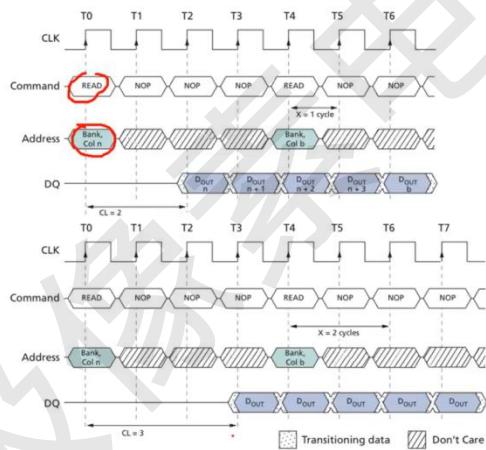
Figure 27: Random WRITE Cycles



Note: 1. Each WRITE command can be issued to any bank. DQM is LOW.

顺序读写

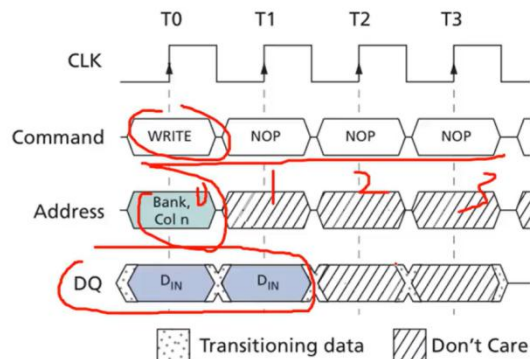
Figure 16: Consecutive READ Bursts



Note: 1. Each READ command can be issued to any bank. DQM is LOW.

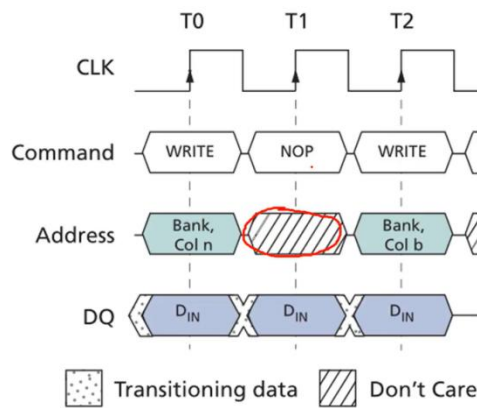
写一半时的处理

Figure 25: WRITE Burst



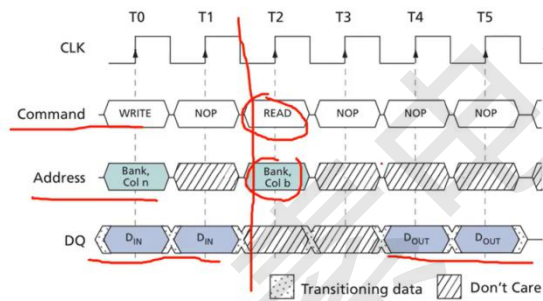
Note: 1. BL = 2. DQM is LOW.

Figure 26: WRITE-to-WRITE



Note: 1. DQM is LOW. Each WRITE command may be issued to any bank.

Figure 28: WRITE-to-READ



Note: 1. The WRITE command can be issued to any bank, and the READ command can be to any bank. DQM is LOW. CL = 2 for illustration.

Figure 29: WRITE-to-PRECHARGE

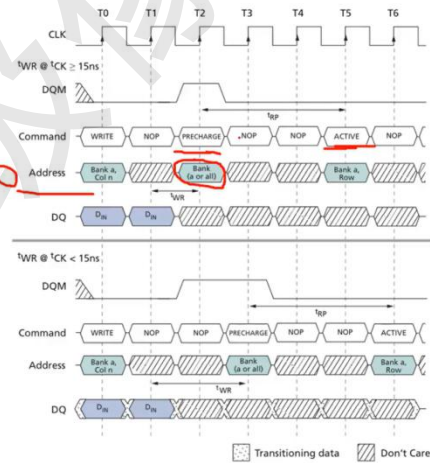
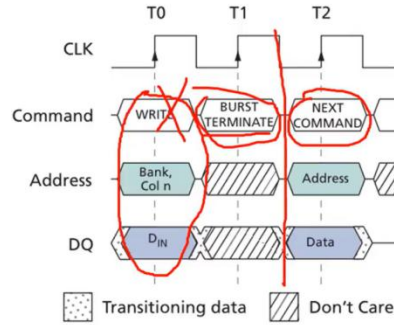
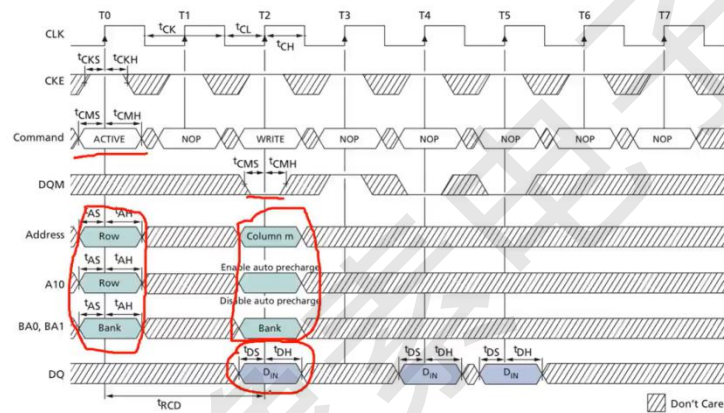


Figure 30: Terminating a WRITE Burst



Note: 1. DQM is LOW.

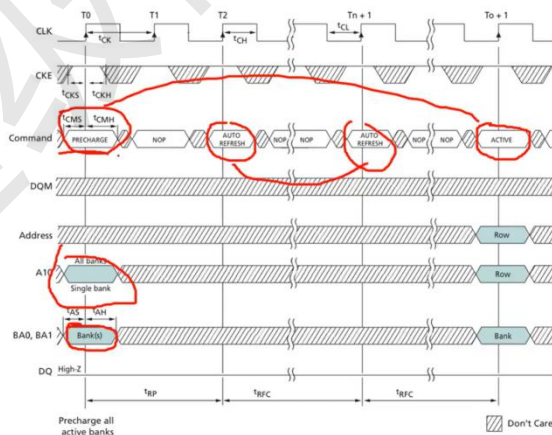
Figure 33: WRITE - DQM Operation



Note: 1. For this example, BL = 4.

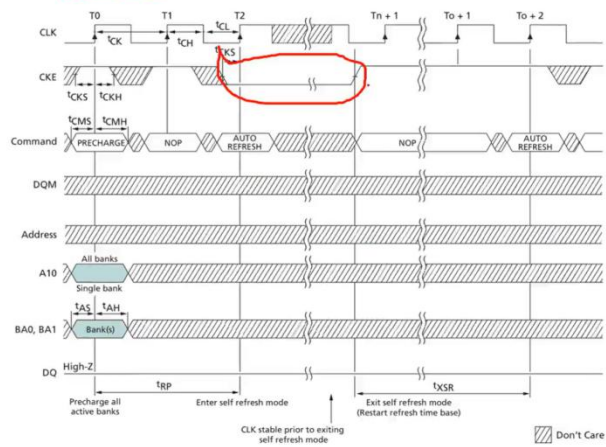
刷新处理

Figure 46: Auto Refresh Mode



Note: 1. Back-to-back AUTO REFRESH commands are not required.

Figure 47 Self Refresh Mode

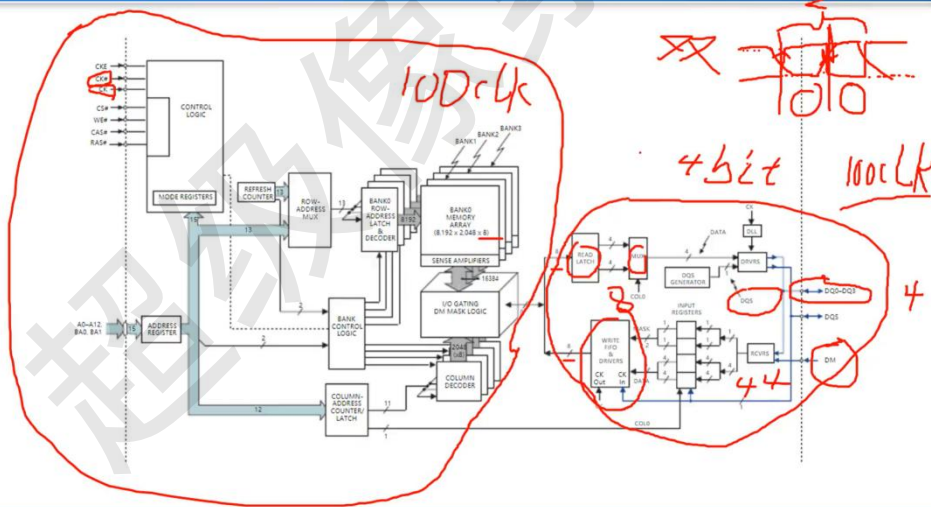


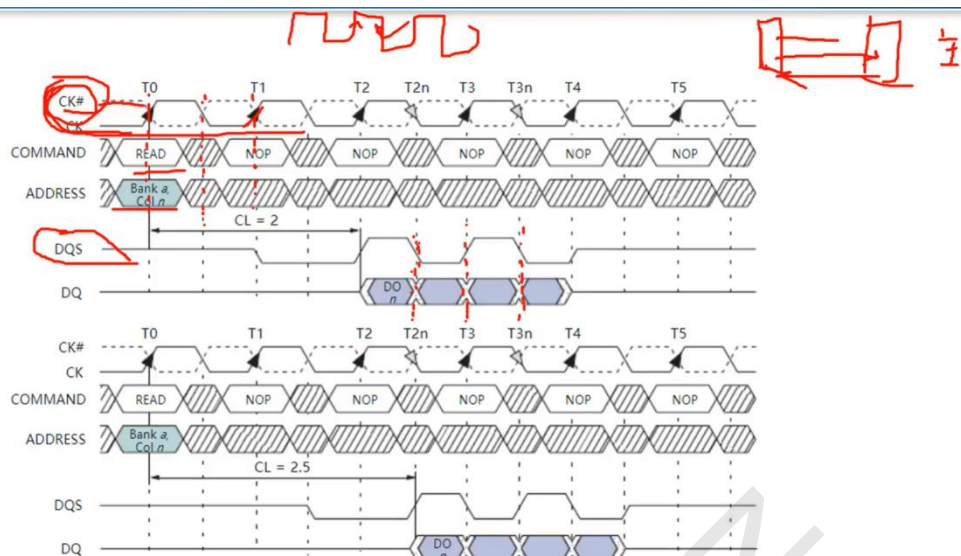
Note: 1. Each AUTO REFRESH command performs a REFRESH cycle. Back-to-back commands are not required.

Ddr

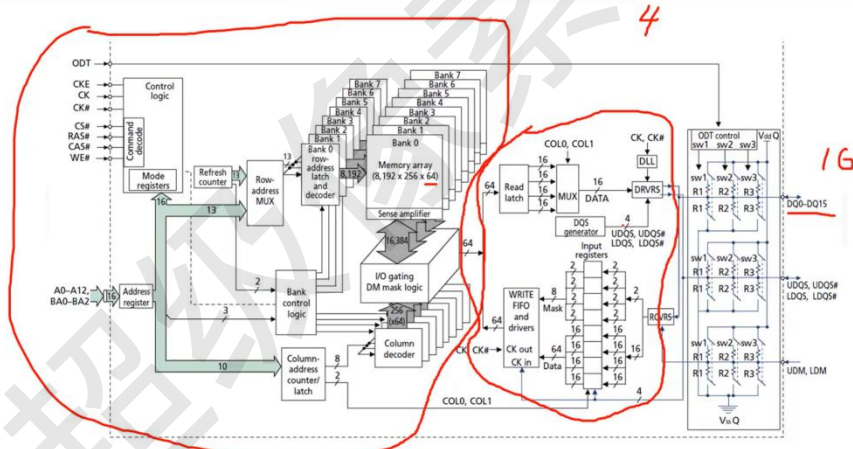
Ddr1

正点原子





Ddr2



这也是为什么我们ddr2的数据的传输速度比ddr3要快一倍的原因

DDR IP 核时序

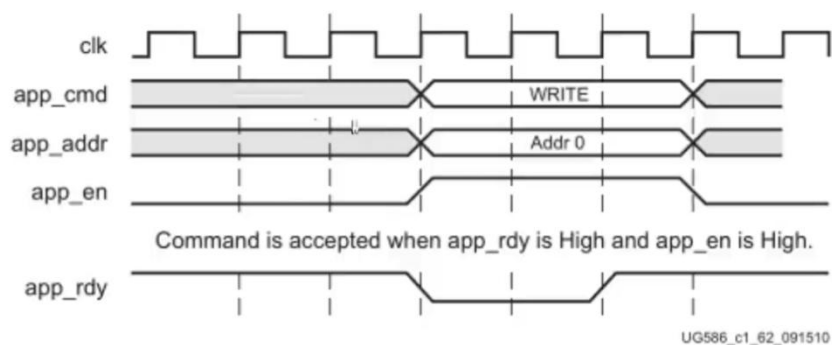


图 30.1.3 写命令时序

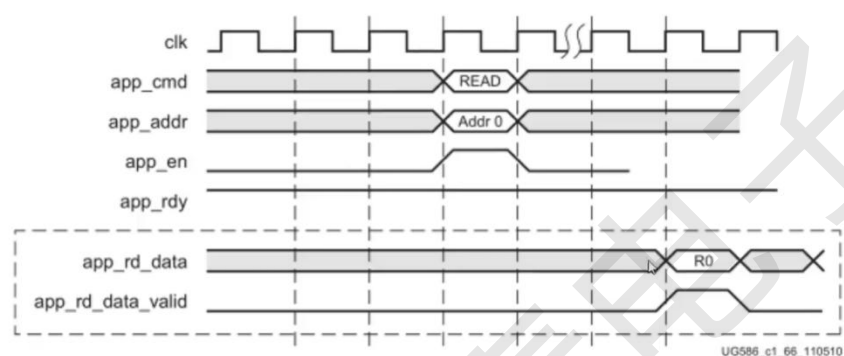


图 30.1.6 读时序

Ddr3

引脚

控制组

名称	功能	描述
CK_P/CK_N	差分时钟输入	所有的地址和控制输入信号都在CK_P的上升沿和CK_N的下降沿相交处采样
CKE	时钟使能	内部时钟使能信号
CS#	片选	CS的参考电压是VREFCA
RAS#	行地址指令	行地址标识信号，其参考电压是VREFCA
CAS#	列地址指令	列地址标识信号，其参考电压是VREFCA
WE	读写指令	低电平表示写，高电平表示读操作，其参考电压是VREFCA
RESET#	复位	芯片复位信号，参考电压VDD
ODT	片上终端使能	消除DQ、DQS、DM信号的反射，ODT的参考电压是VREFCA
ZQ#	外部校准	ZQ校准的参考引脚，这个引脚应该连接240欧姆电阻到地。

地址组

名称	功能	描述
<u>A[14:0]</u>	行列地址总线	为 ACTIVATE命令提供行地址，为 READ/WRITE 命令提供列地址和自动预充电位(A10)，以便从某个Bank的内存阵列里选出一个位置，还提供模式寄存器配置期间的操作码。地址输入的参考电压是 VREFCA。
<u>BA[2:0]</u>	Bank地址总线	确定要操作的Bank，还决定在MRS周期中访问哪种模式寄存器。参考电压是VREFCA

数据组

名称	功能	描述
<u>DQ</u>	数据总线	双向数据总线，参考电压是VREFDQ。
<u>DQS_P/DQS_N</u>	数据选通	用于数据同步，读时是输出，交叉点与读出的数据边缘对齐。写时是输入，交叉点与写数据中心对齐。参考电压是VREFDQ。
<u>DM</u>	数据屏蔽	DM是写数据的输入屏蔽信号，在写期间，当DM信号被采样为高时，输入数据被屏蔽。参考电压是VREFDQ。

时钟频率与带宽

core freq: ^{100m}核心频率，用于DDR内部cell(存储单元)的时钟；
clock freq: ^{400m}时钟频率，用于DDR的IO buffer的时钟，同时也是IO接口时钟，是通过核心频率倍频4倍得到的；
^{800m}data rate: ⁸⁰⁰数据速率，单根数据线的数据传送次数；
bandwidth: ¹⁶芯片带宽，数据速率*芯片的数据位宽；
 $800 \times 16 = 12.8G$

地址及容量计算

Parameter	512 Meg x 4	256 Meg x 8	128 Meg x 16
Configuration	64 Meg x 4 x 8 banks	32 Meg x 8 x 8 banks	16 Meg x 16 x 8 banks
Refresh count	8K	8K	8K
Row addressing	32K (A[14:0])	32K (A[14:0])	16K (A[13:0]) ^{2¹⁴}
Bank addressing	8 (BA[2:0])	8 (BA[2:0])	8 (BA[2:0]) ^{2³}
Column addressing	2K (A[11, 9:0])	1K (A[9:0])	1K (A[9:0]) ^{2¹⁰}
Page size	1KB	1KB	2KB ^{2¹¹}

总存储容量 = L-Bank 的数量 × 行数 × 列数 × 芯片位宽

突发长度 (Burst Length, BL)

由于DDR3的预取为8n，所以突发传输周期 (Burst Length, BL) 也固定为8，而对于DDR2和早期的DDR架构系统，BL=4也是常用的，DDR3为此增加了一个4bit Burst Chop (突发突变) 模式，届时可通过地址线A12来控制这一突发模式。

突发类型 (Burst Type)

Burst Length	READ/ WRITE	Starting Column Address (A[2, 1, 0])	Burst Type = Sequential (Decimal)	Burst Type = Interleaved (Decimal)
4	READ	0 0 0	0, 1, 2, 3, Z, Z, Z, Z	0, 1, 2, 3, Z, Z, Z, Z
		0 0 1	1, 2, 3, 0, Z, Z, Z, Z	1, 0, 3, 2, Z, Z, Z, Z
		0 1 0	2, 3, 0, 1, Z, Z, Z, Z	2, 3, 0, 1, Z, Z, Z, Z
		0 1 1	3, 0, 1, 2, Z, Z, Z, Z	3, 2, 1, 0, Z, Z, Z, Z
		1 0 0	4, 5, 6, 7, Z, Z, Z, Z	4, 5, 6, 7, Z, Z, Z, Z
		1 0 1	5, 6, 7, 4, Z, Z, Z, Z	5, 4, 7, 6, Z, Z, Z, Z
		1 1 0	6, 7, 4, 5, Z, Z, Z, Z	6, 7, 4, 5, Z, Z, Z, Z
		1 1 1	7, 4, 5, 6, Z, Z, Z, Z	7, 6, 5, 4, Z, Z, Z, Z
	WRITE	0 V V	0, 1, 2, 3, X, X, X, X	0, 1, 2, 3, X, X, X, X
		1 V V	4, 5, 6, 7, X, X, X, X	4, 5, 6, 7, X, X, X, X
8	READ	0 0 0	0, 1, 2, 3, 4, 5, 6, 7	0, 1, 2, 3, 4, 5, 6, 7
		0 0 1	1, 2, 3, 0, 5, 6, 7, 4	1, 0, 3, 2, 5, 4, 7, 6
		0 1 0	2, 3, 0, 1, 6, 7, 4, 5	2, 3, 0, 1, 6, 7, 4, 5
		0 1 1	3, 0, 1, 2, 7, 4, 5, 6	3, 2, 1, 0, 7, 6, 5, 4
		1 0 0	4, 5, 6, 7, 0, 1, 2, 3	4, 5, 6, 7, 0, 1, 2, 3
		1 0 1	5, 6, 7, 4, 1, 2, 3, 0	5, 4, 7, 6, 1, 0, 3, 2
		1 1 0	6, 7, 4, 5, 2, 3, 0, 1	6, 7, 4, 5, 2, 3, 0, 1
		1 1 1	7, 4, 5, 6, 3, 0, 1, 2	7, 6, 5, 4, 3, 2, 1, 0
	WRITE	V V V	0, 1, 2, 3, 4, 5, 6, 7	0, 1, 2, 3, 4, 5, 6, 7

Mig IP 核

MIG IP 核信号定义

信号名	端口方向	描述
ui_clk	output	用户时钟，由MIG IP核输出
ui_clk_sync_rst	output	复位，由MIG IP核提供，高电平有效
init_calib_complete	output	高电平有效，表明DDR3芯片初始化完成
app_addr [ADDR_WIDTH-1:0]	input	用户地址输入，地址的位宽ADDR_WIDTH等于RANK位宽+BANK位宽+ROW位宽+COL位宽
app_en	input	MIG IP核命令写入使能，高有效。写命令时需要拉高该信号
app_cmd[2:0]	input	读写控制命令读：001；写：000。其他值保留。

MIG IP 核信号定义

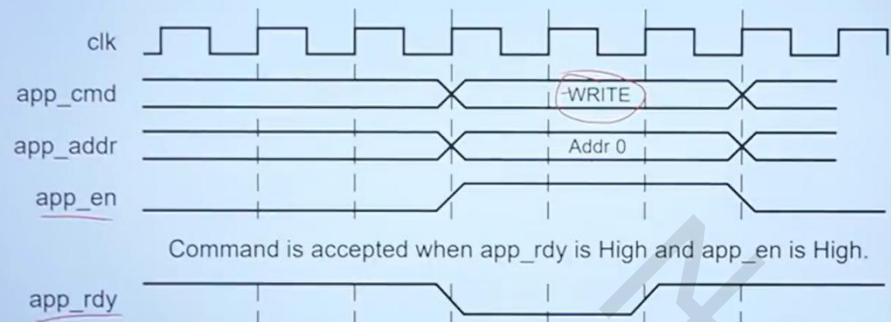
信号名	端口方向	描述
app_rdy	output	MIG IP核读写命令接收准备完成，高电平有效。
app_wdf_wren	input	MIG IP核数据写使能，高电平有效。写数据时需要拉高该信号
app_wdf_rdy	output	MIG IP核数据接收准备完成，高电平有效。
app_wdf_end	input	当前时钟是突发写过程的最后一个时钟，高电平有效。
app_wdf_data [APP_DATA_WIDTH-1:0]	input	该信号为用户写数据输入，位宽为芯片位宽*8
app_rd_data [APP_DATA_WIDTH-1:0]	output	该信号用于用户读取IP核发来的数据，位宽为芯片位宽*8
app_rd_data_valid	output	读数据有效信号，高电平有效。

MIG IP 核信号定义

信号名	端口方向	描述
app_sr_req	Input	保留位，设置为0
app_ref_req	Input	高有效，表示DRAM需要进行刷新指令
app_zq_req	input	高有效，表示DRAM需要进行校准
app_sr_active	Output	输出保留位
app_ref_ack	Output	高有效，表示控制器已经向PHY接口发送了刷新命令
app_zq_ack	Output	高有效，表示控制器已经向PHY接口发送了校准命令
app_wdf_mask [APP_MASK_WIDTH - 1:0]	input	数据掩码，高有效，APP_MASK_WIDTH = APP_DATA_WIDTH/8

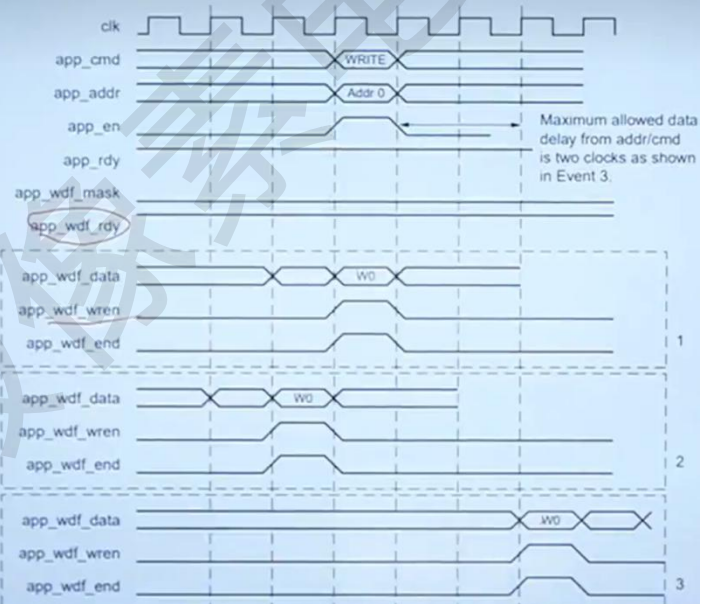
时序

写命令时序

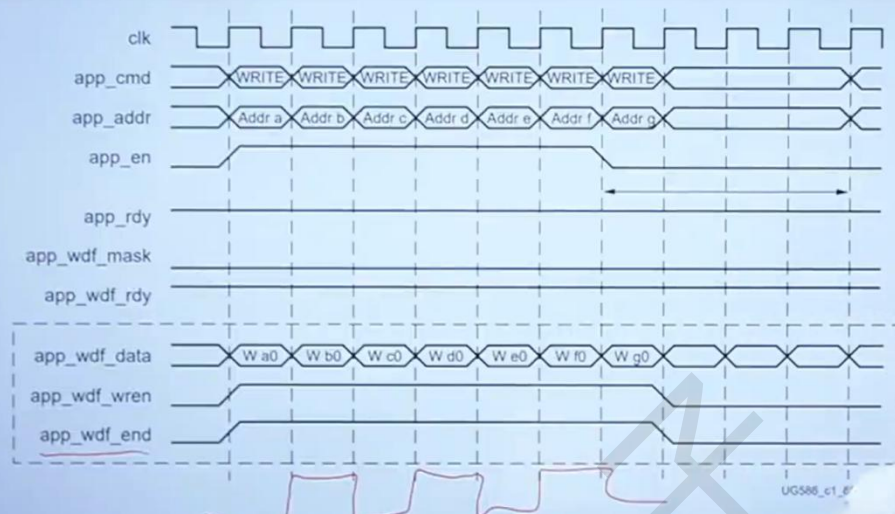


UG586_c1_62_091510

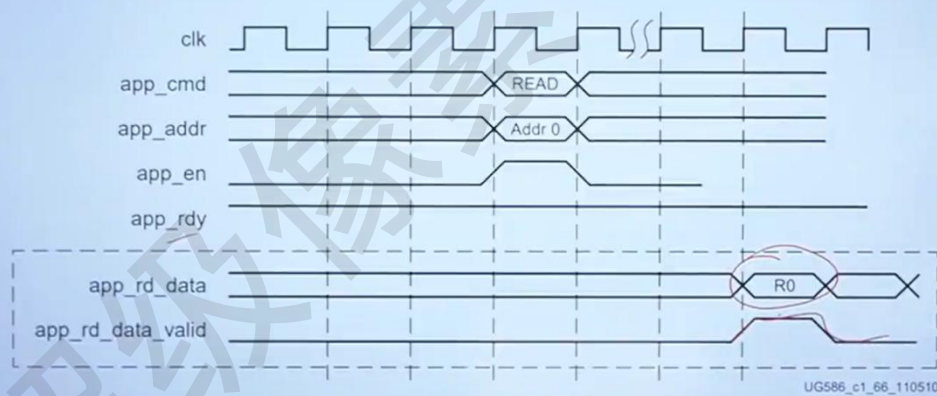
非背靠背写数据时序



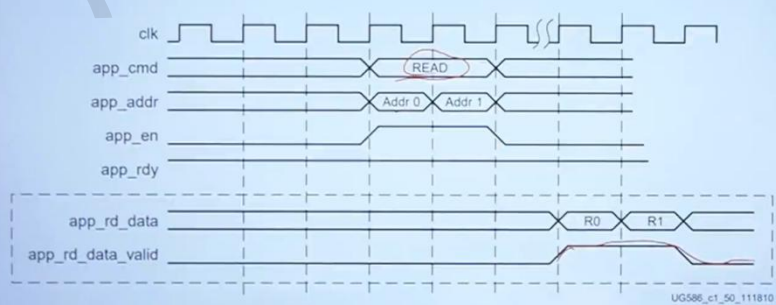
背靠背写数据时序



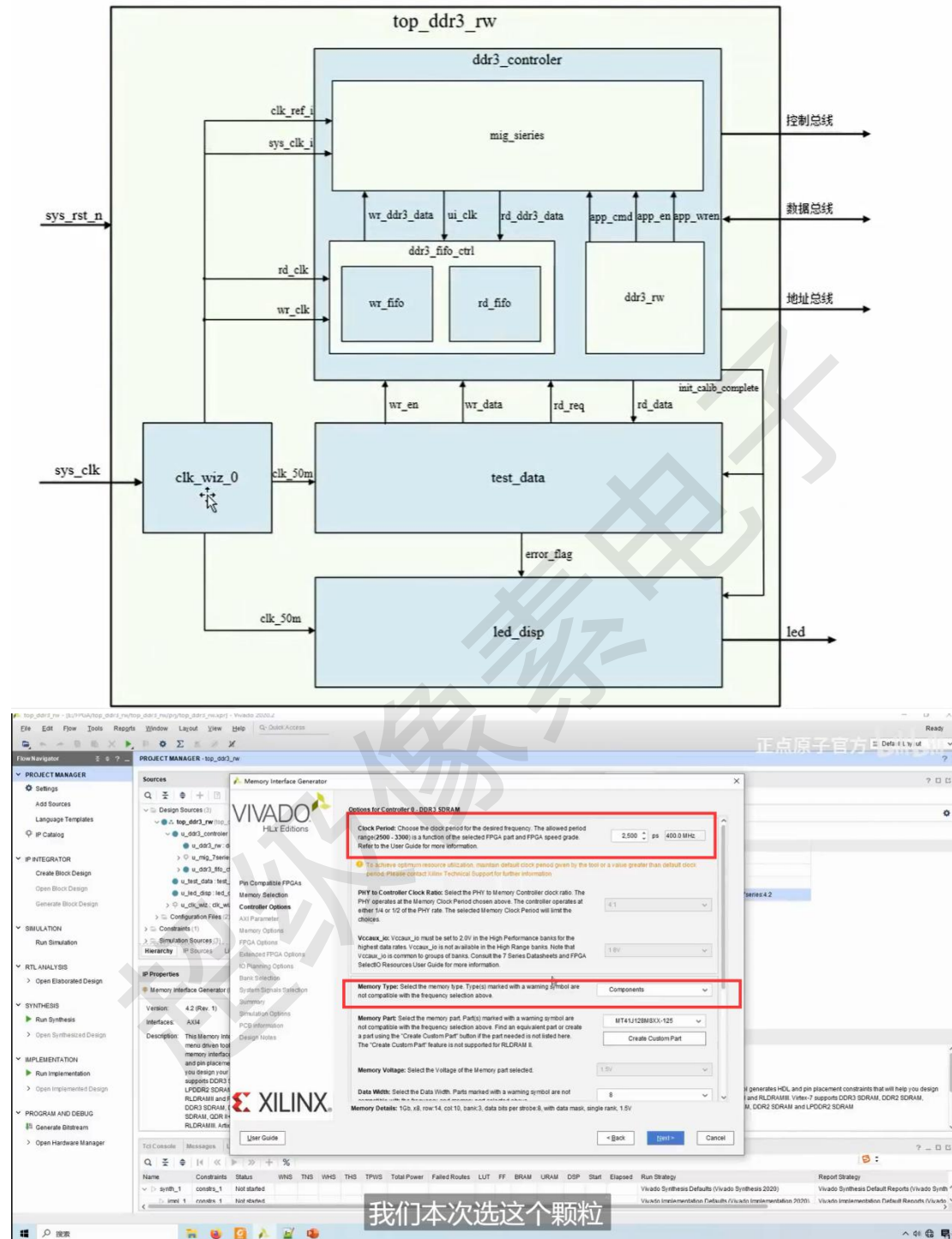
非背靠背读数据时序

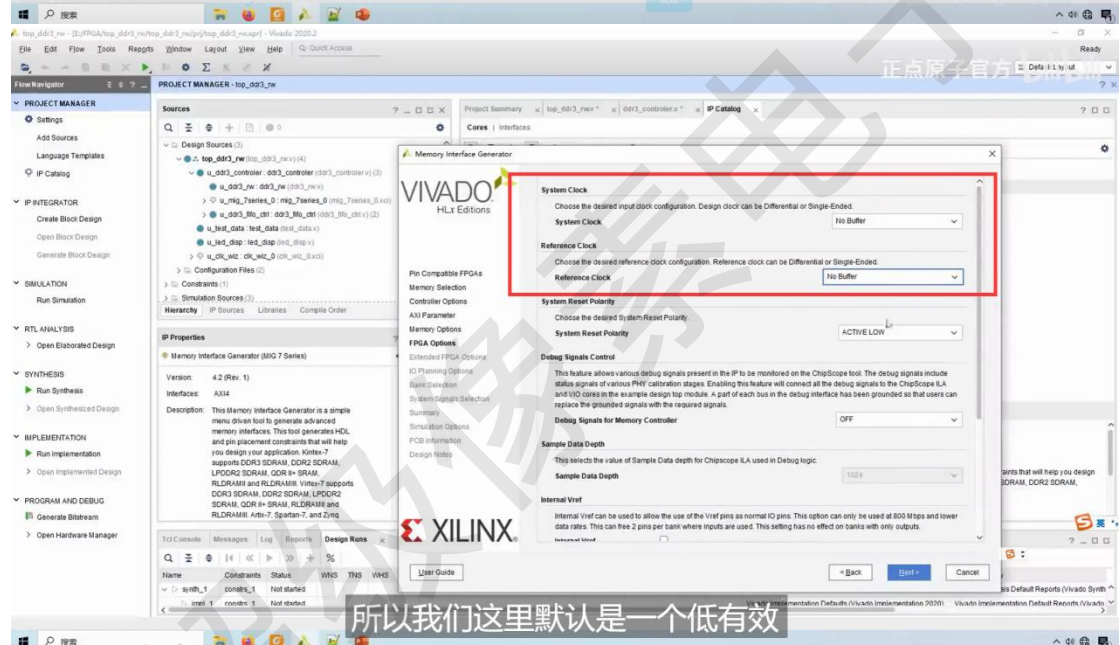
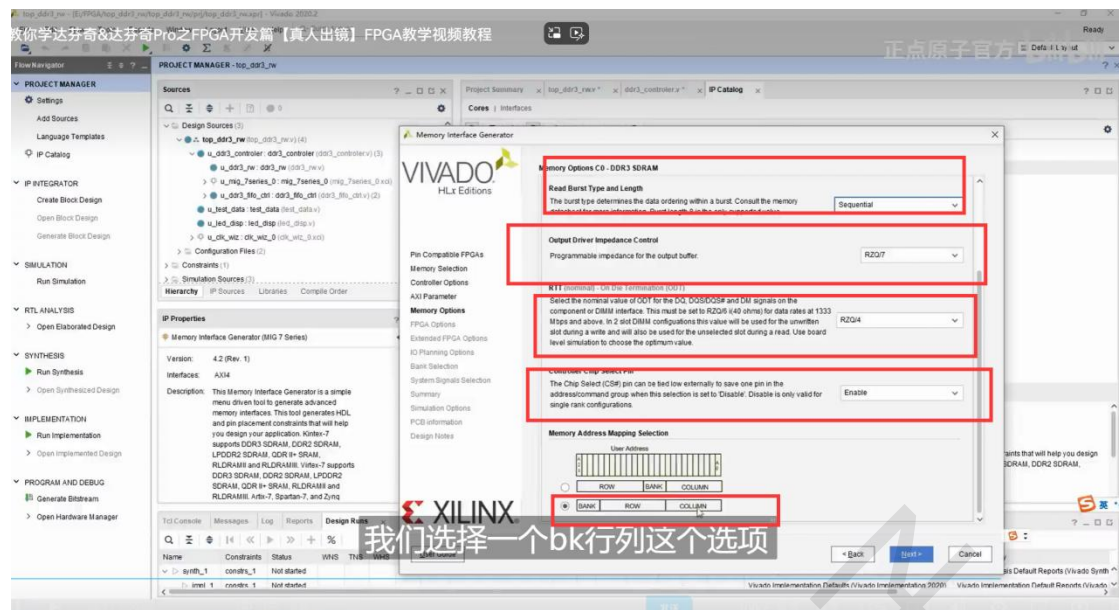


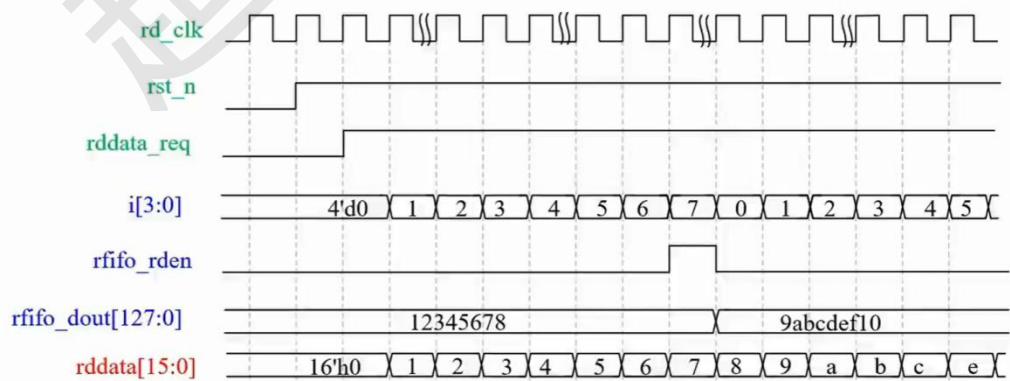
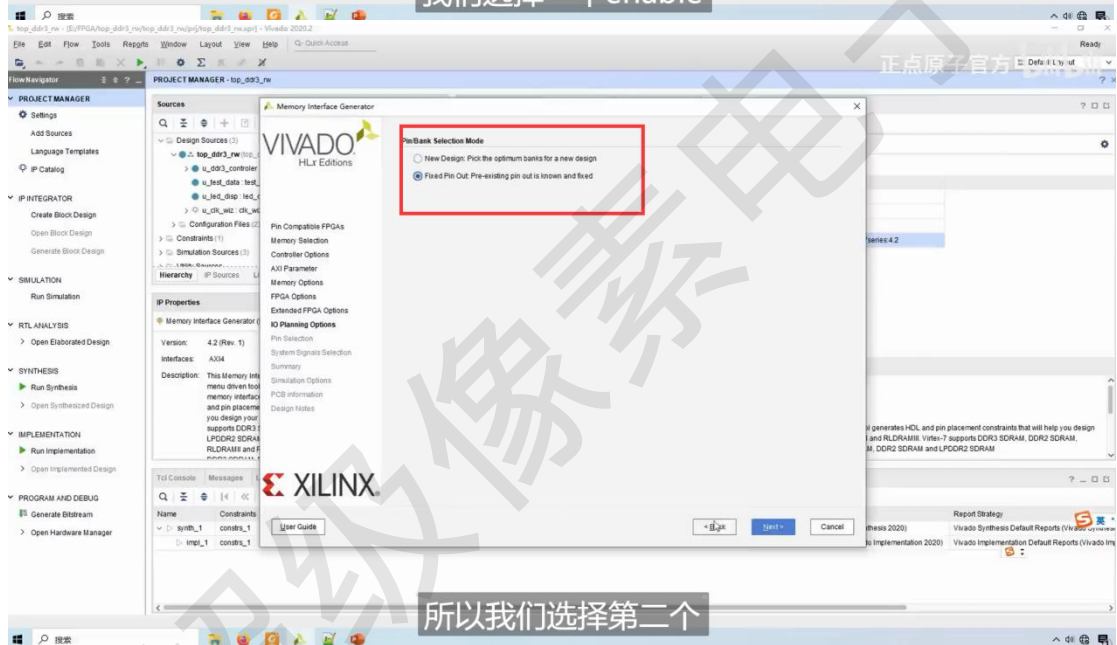
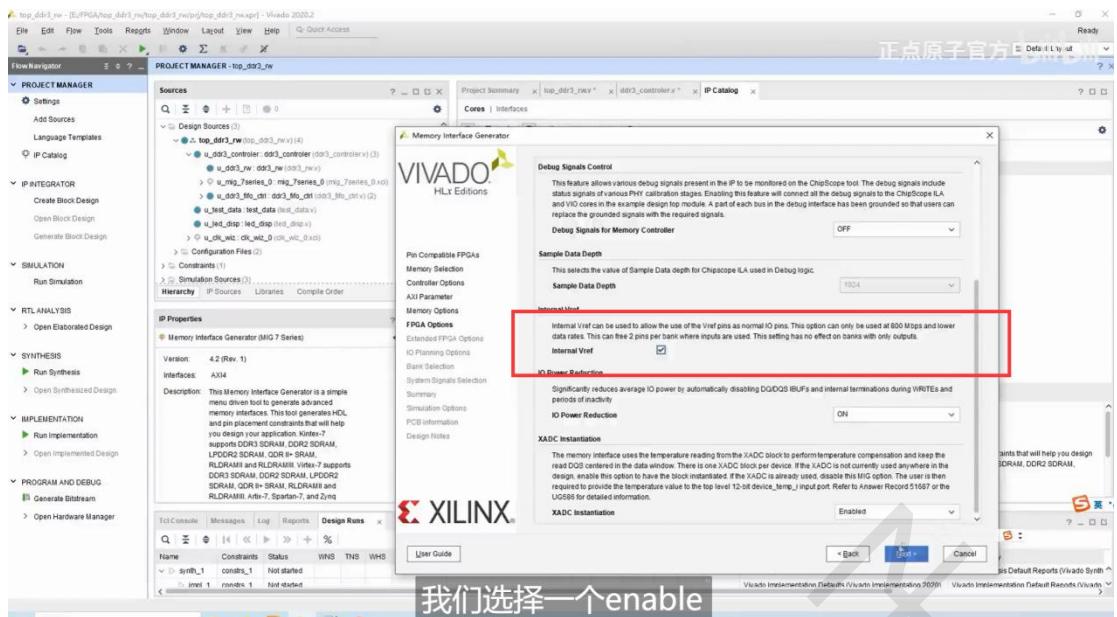
背靠背读数据时序

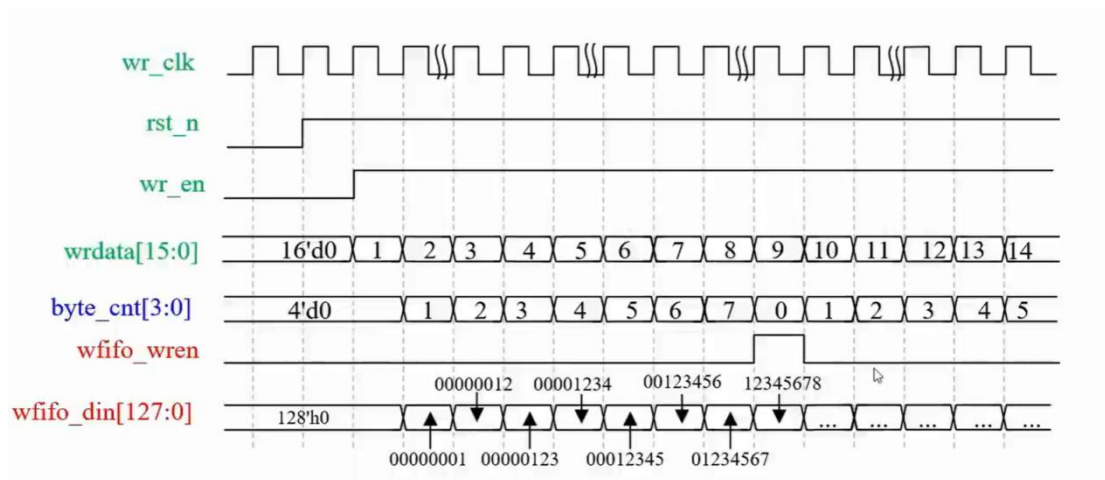


程序









MDIO

手把手教你学FPGA

正点原子官方Bilibili

RJ45接口定义(10/100M)

引脚编号	引脚名称	功能说明
Pin 1	TX+	Tranceive Data+ (发送数据+)
Pin 2	TX-	Tranceive Data- (发送数据-)
Pin 3	RX+	Receive Data+ (接收数据+)
Pin 4	NC	Not connected (未使用)
Pin 5	NC	Not connected (未使用)
Pin 6	RX-	Receive Data- (接收数据-)
Pin 7	NC	Not connected (未使用)
Pin 8	NC	Not connected (未使用)

“原子哥” 在线教学平台: www.yuanzige.com 技术支持论坛: <http://www.openedv.com/>

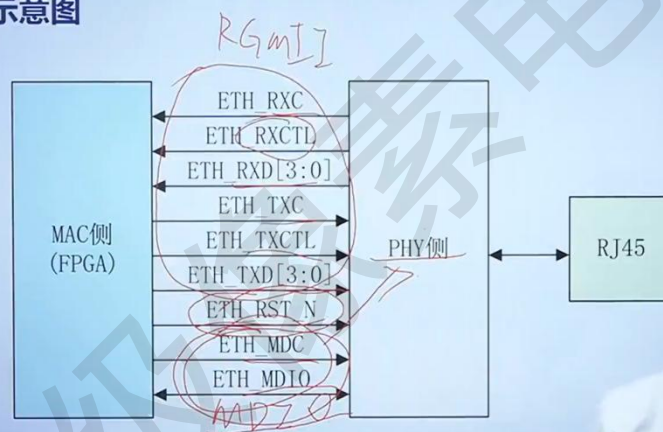
RJ45接口定义(1000M)



引脚编号	引脚名称	功能说明
Pin 1	MDI0+	双向数据Data0+
Pin 2	MDI0-	双向数据Data0-
Pin 3	MDI1+	双向数据Data1+
Pin 4	MDI2+	双向数据Data2+
Pin 5	MDI2-	双向数据Data2-
Pin 6	MDI1-	双向数据Data1-
Pin 7	MDI3+	双向数据Data3+
Pin 8	MDI3-	双向数据Data3-

“原子哥”在线教学平台: www.yuanzige.com技术支持论坛: <http://www.openedv.com>

以太网连接示意图

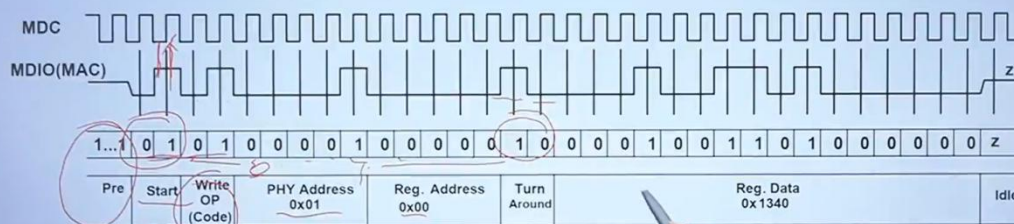
“原子哥”在线教学平台: www.yuanzige.com技术支持论坛: <http://www.openedv.com>

MDIO接口的帧格式

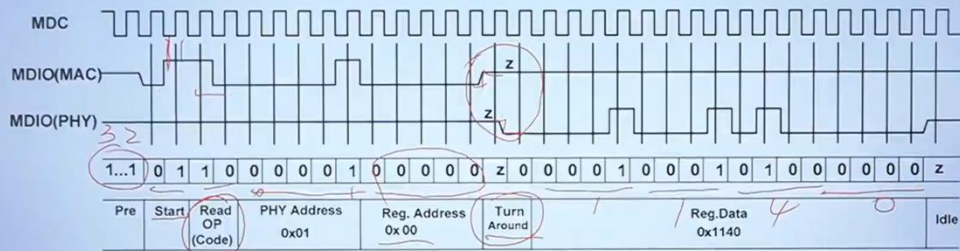
Z:高阻态

	前端码 (32个1)	帧头	读写操作位	PHY地址 (后三位为0)	寄存器地址	控制位 (传输方向改变)	数据	空闲位
	Preamble	ST	OP	PHYAD	REGAD	Turn	DATA	IDLE
Read	1...1	01	10	AAAAA	RRRRR	Z0	DDDDDDDDDDDDDDDDDD	Z
Write	1...1	01	01	AAAAA	RRRRR	10	DDDDDDDDDDDDDDDDDD	Z

MDIO接口写时序图



MDIO接口读时序图



基本控制寄存器（Basic Control Register）（地址：0x00）

Bit位	功能	访问类型	功能描述
15	复位	RW SC	软件复位, 1: PHY复位 0: 正常模式; 当复位完成后, 该位自动置0
14	环回	RW SWC	是否开启芯片内部自环回, 1: 内部环回模式 0: 正常模式;
13	通信速度选择的低位	RW	只有在自动协商使能不开启的情况下有效, 与bit6一起组成2bit数据, 用来选择通信速度 10: 1000Mb/s 01: 100Mb/s 00: 10Mb/s;

基本控制寄存器（Basic Control Register）（地址：0x00）

Bit位	功能	访问类型	功能描述
12	自协商使能	RW	自动协商使能, 1: 打开自动协商使能 0: 关闭自动协商使能
11	休眠模式	RW SWC	是否开启休眠模式, 1: 休眠模式 0: 正常模式

基本控制寄存器（Basic Control Register）（地址：0x00）

Bit位	功能	访问类型	功能描述
9	重启自协商	RW SWS SC	重启自动协商进程, 1: 重启自动协商进程 0: 正常模式
8	双工模式	RW	双工模式选择, 1: 全双工模式 0: 半双工模式; 只用在自动协商被禁止的时候可用
6	通信速度选择的高位	RW	只有在自动协商使能不开启的情况下有效, 与bit13一起组成2bit数据, 用来选择通信速度 10: 1000Mb/s 01: 100Mb/s 00: 10Mb/s;

基本状态寄存器（Basic Status Register）（地址：0x01）

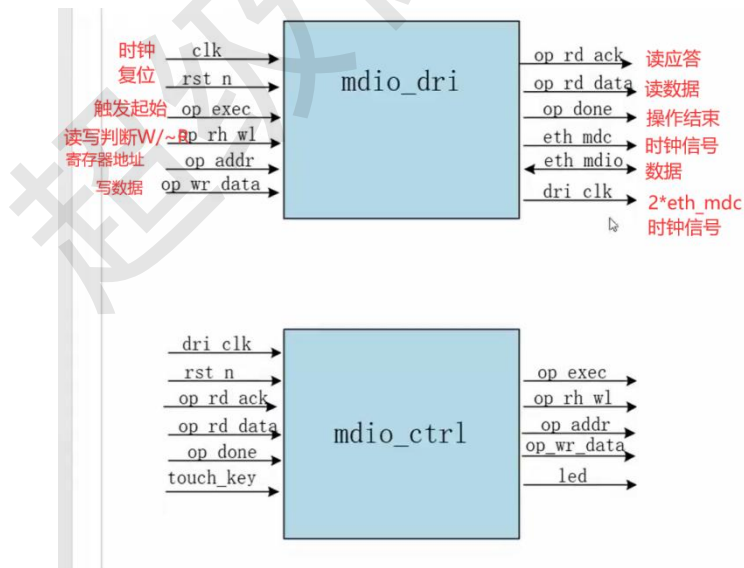
Bit位	功能	访问类型	功能描述
7	单向传输能力	RO	0: 只有当PHY确定已经建立了有效的链路时，PHY才能从MII传输 1: 无论PHY是否确定已经建立了有效的链路，PHY都能够从MII传输
6	前导码模式	RO	1: PHY能够接受管理帧，而不管它们前面是否有前导码 0: PHY不能接受管理帧，除非它们前面有前导码模式
5	自协商完成标志	RO SWC	1: 自协商完成 0: 自协商未完成

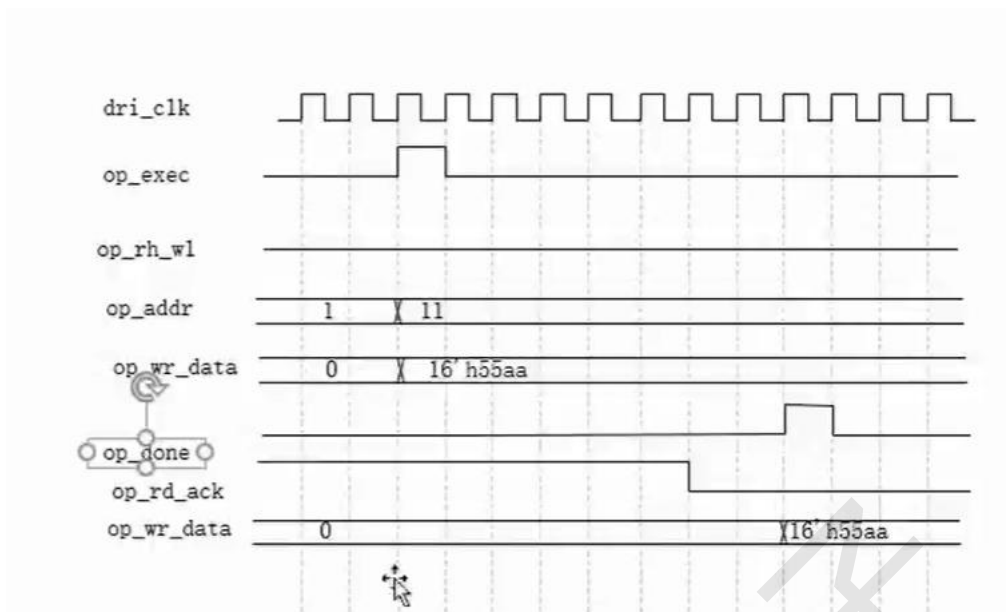
基本状态寄存器（Basic Status Register）（地址：0x01）

Bit位	功能	访问类型	功能描述
4	远端错误指示位	RO RC SWC	0: 远端设备没有出错 1: 远端设备出错
3	自协商能力	RO	1: PHY芯片具有自协商能力 0: PHY不具有自协商能力
2	链接状态	RO SWC	1: 链接已建立 0: 链接未建立

PHY芯片特定状态寄存器（PHY Specific Status Register）（地址：0x11）

Bit位	功能	访问类型	功能描述
15~14	传输速度等级检测	RO	11: 保留 10: 1000Mbps 01: 100Mbps 00: 10Mbps
13	双工模式检测	RO	1: 全双工模式 0: 半双工模式



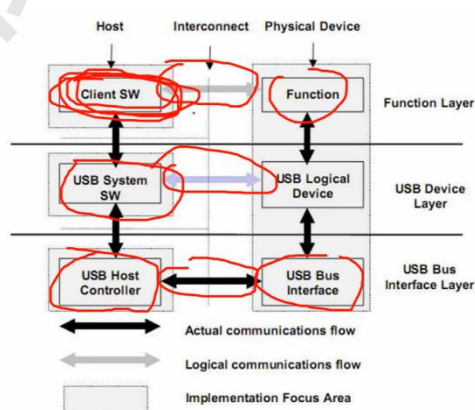


USB

协议

域与数位

在 HSOT 端，应用软件（Client SW）不能直接访问 USB 总线，而必须通过 USB 系统软件和 USB 主机控制器来访问 USB 总线，在 USB 总线上和 USB 设备进行通讯。从逻辑上可以分为功能层、设备层和总线接口层三个层次。其中功能层完成功能级的描述、定义和行为；设备层则完成从功能级到传输级的转换，把一次功能级的行为转换为一次一次的基本传输；USB 总线接口层则处理总线上的 Bit 流，完成数据传输的物理层实现和总线管理。途中黑色箭头代表真实的数据流，灰色箭头代表逻辑上的通讯。



域：是USB数据最小的单位，由若干位组成（至于是多少位由具体的域决定），域可分为七个类型：

- 1、同步域（SYNC），八位，值固定为0000 0001，用于本地时钟与输入同步；
- 2、标识域（PID），由四位标识符+四位标识符反码构成，表明包的类型和格式，这是一个很重要的部分，这里可以计算出，USB的标识码有16种；
- 3、地址域（ADDR）：七位地址，代表了设备在主机上的地址，地址000 0000被命名为零地址，是任何一个设备第一次连接到主机时，在被主机配置、枚举前的默认地址，由此可以知道为什么一个USB主机只能接127个设备的原因。
- 4、端点域（ENDP），四位，由此可知一个USB设备有的端点数量最大为16个。
- 5、帧号域（FRAM），11位，每一个帧都有一个特定的帧号，帧号域最大容量0x800，对于同步传输有重要意义（同步传输为四种传输类型之一）。
- 6、数据域（DATA）：长度为0~1023字节，在不同的传输类型中，数据域的长度各不相同，但必须为整数个字节的长度
- 7、校验域（CRC）：对令牌包和数据包（对于包的分类请看下面）中非PID域进行校验的一种方法，CRC校验在通讯中应用很泛，是一种很好的校验方法，至于具体的校验方法这里就不多说，请查阅相关资料，只须注意CRC码的除法是模2运算，不同于10进制中的除法。

USB 数据包的格式

Field	PID	ADDR	ENDP	DATA	CRC
		FrameNumber			
Bits	4+4	7	4	N*8(n=0,1...,1024)	5/16
		11			

采用 NRZI 编码

包

包：由域构成的包有四种类型，分别是令牌包、数据包、握手包和特殊包，前面三种是重要的包，不同的包的域结构不同，介绍如下

1、令牌包：可分为输入包、输出包、设置包和帧起始包（注意这里的输入包是用于设置输入命令的，输出包是用来设置输出命令的，而不是放数据的）其中输入包、输出包和设置包的格式都是一样的：SYNC+PID+ADDR+ENDP+CRC5（五位的校验码）

帧起始包的格式：SYNC+PID+11位FRAM+CRC5（五位的校验码）

2、数据包：分为DATA0包和DATA1包，当USB发送数据的时候，当一次发送的数据长度大于相应端点的容量时，就需要把数据包分为好几个包，分批发送，DATA0包和DATA1包交替发送，即如果第一个数据包是DATA0，那第二个数据包就是DATA1。但也有例外情况，在同步传输中（四类传输类型中之一），所有的数据包都是为DATA0，格式如下

：SYNC+PID+0~1023字节+CRC16

3、握手包：包括ACK、NAK、STALL以及NYET四种，其中ACK表示肯定的应答，成功的数据传输；NAK表示否定的应答，失败的数据传输，要求重新传输；STALL表示功能错误或端点被设置了STALL属性；NYET表示尚未准备好，要求等待。

格式如下 SYNC+PID

事务

事务：分别有IN事务、OUT事务和SETUP事务三大事务，每一种事务都由令牌包、数据包、握手包三个阶段构成，这里用阶段的意思是因为这些包的发送是有一定的时间先后顺序的，事务的三个阶段如下：

- 1、令牌包阶段：启动一个输入、输出或设置的事务
- 2、数据包阶段：按输入、输出发送相应的数据
- 3、握手包阶段：返回数据接收情况，在同步传输的IN和OUT事务中没有这个阶段，这是比较特殊的。

事务的三种类型如下（以下按三个阶段来说明一个事务）：

IN事务：

令牌包阶段——主机发送一个PID为IN的输入包给设备，通知设备要往主机发送数据；

数据包阶段——设备根据情况会作出三种反应（要注意：数据包阶段也不总是传送数据的，根据传输情况还会提前进入握手包阶段）

- 1) 设备端点正常，设备往主机里面发出数据包（DATA0与DATA1交替）；
- 2) 设备正在忙，无法往主机发出数据包就发送NAK无效包，IN事务提前结束，到了下一个IN事务才继续；
- 3) 相应设备端点被禁止，发送错误包STALL包，事务也就提前结束了，总线进入空闲状态。

握手包阶段——主机正确接收到数据之后就会向设备发送ACK包。

OUT事务：

令牌包阶段——主机发送一个PID为OUT的输出包给设备，通知设备要接收数据；

数据包阶段——比较简单，就是主机会给设备送数据，DATA0与DATA1交替

握手包阶段——设备根据情况会作出三种反应

- 1) 设备端点接收正确，设备往主机返回ACK，通知主机可以发送新的数据，如果数据包发生了CRC校验错误，将不返回任何握手信息；
- 2) 设备正在忙，无法接收主机发出数据包就发送NAK无效包，通知主机再次发送数据；
- 3) 相应设备端点被禁止，发送错误包STALL包，事务提前结束，总线直接进入空闲状态。

SETUP事务：

令牌包阶段——主机发送一个PID为SETUP的输出包给设备，通知设备要接收数据；

数据包阶段——比较简单，就是主机会给设备送数据，注意，这里只有一个固定为8个字节的DATA0包，这8个字节的内容就是标准的USB设备请求命令（共有11条）

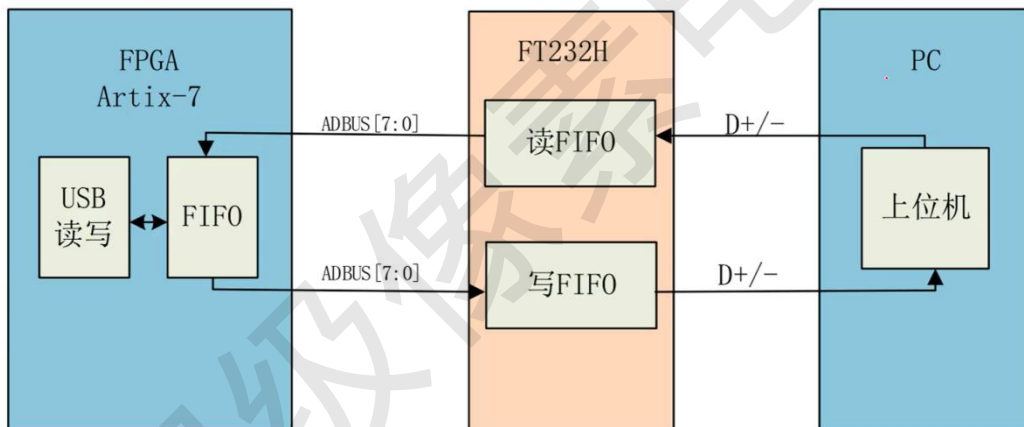
握手包阶段——设备接收到主机的命令信息后，返回ACK，此后总线进入空闲状态，并准备下一个传输（在SETUP事务后通常是一个IN或OUT事务构成的传输）

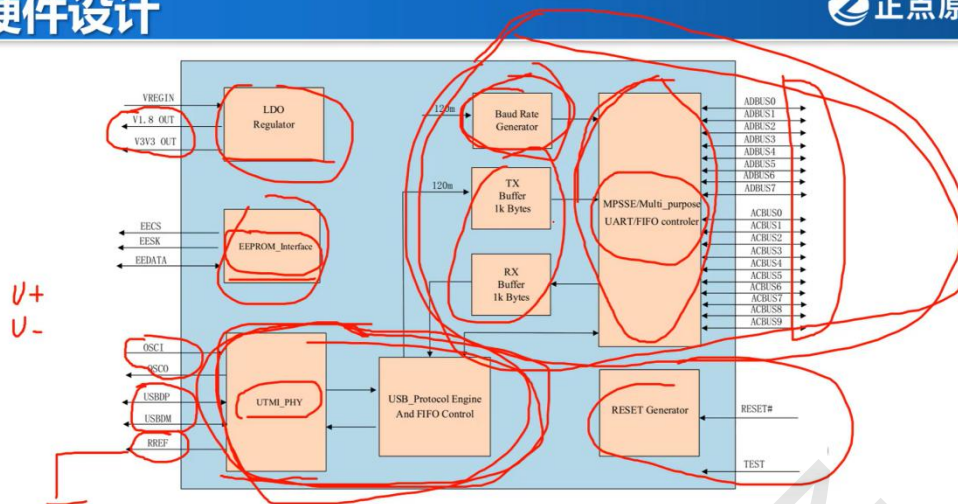
传输

传输：传输由OUT、IN、SETUP事务其中的事务构成，传输有四种类型，中断传输、批量传输、同步传输、控制传输，其中中断传输和批量传输的结构一样，同步传输有最简单的结构，而控制传输是最重要的也是最复杂的传输。

- 1、中断传输：由OUT事务和IN事务构成，用于键盘、鼠标等HID设备的数据传输中
- 2、批量传输：由OUT事务和IN事务构成，用于大容量数据传输，没有固定的传输速率，也不占用带宽，当总线忙时，USB会优先进行其他类型的数据传输，而暂时停止批量传输。
- 3、同步传输：由OUT事务和IN事务构成，有两个特殊地方，第一，在同步传输的IN和OUT事务中是没有返回包阶段的；第二，在数据包阶段所有的数据包都为DATA0
- 4、控制传输：最重要的也是最复杂的传输，控制传输由三个阶段构成（初始设置阶段、可选数据阶段、状态信息步骤），每一个阶段可以看成是一个的传输，也就是说控制传输其实是由三个传输构成的，用于USB设备初次加接到主机之后，主机通过控制传输来交换信息，设备地址和读取设备的描述符，使得主机识别设备，并安装相应的驱动程序，这是每一个USB开发者都要关心的问题。

FT232





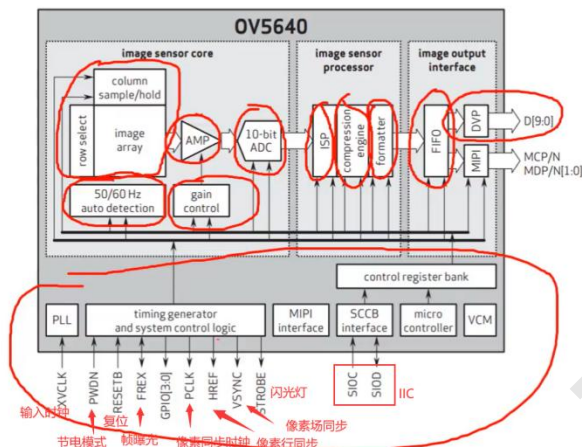
所以我们到这里就将整个芯片的各个模块简单的跟大家介绍了一下

Pin No.	Name	Type	FT245 Configuration Description
13,14,15,16,17,18,19,20	ADBUS[7:0]	I/O	D7 to D0 bidirectional FIFO data. This bus is normally input unless OE# is low.
21	RXF#	OUTPUT	When high, do not read data from the FIFO. When low, there is data available in the FIFO which can be read by driving RD# low. When in synchronous mode, data is transferred on every clock that RXF# and RD# are both low. Note that the OE# pin must be driven low at least 1 clock period before asserting RD# low.
25	TXE#	OUTPUT	When high, do not write data into the FIFO. When low, data can be written into the FIFO by driving WR# low. When in synchronous mode, data is transferred on every clock that TXE# and WR# are both low.
26	RD#	INPUT	Enables the current FIFO data byte to be driven onto D0...D7 when RD# goes low. The next FIFO data byte (if available) is fetched from the receive FIFO buffer each CLKOUT cycle until RD# goes high.
27	WR#	INPUT	Enables the data byte on the D0...D7 pins to be written into the transmit FIFO buffer when WR# is low. The next FIFO data byte is written to the transmit FIFO buffer each CLKOUT cycle until WR# goes high.
28	SIWU#	INPUT	The Send Immediate / WakeUp signal combines two functions on a single pin. If USB is in suspend mode (PWREN# = 1) and remote wakeup is enabled in the EEPROM, strobing this pin low will cause the device to request a resume on the USB Bus. Normally, this can be used to wake up the Host PC. During normal operation (PWREN# = 0), if this pin is strobed low any data in the device RX buffer will be sent out over USB on the next Bulk-IN request from the drivers regardless of the pending packet size. This can be used to optimize USB transfer speed for some applications. Tie this pin to VCCIO if not used.
29	CLKOUT	OUTPUT	60 MHz Clock driven from the chip. All signals should be synchronized to this clock.
30	OE#	INPUT	Output enable when low to drive data onto D0-7. This should be driven low at least 1 clock period before driving RD# low to allow for data buffer turn-around.



OV5640

Functional Block Diagram



SCCB 时序

SCL SPA

SCCB协议:

SCCB协议有两线也有三线，两线为SIO_C与SIO_D,三线为SIO_E、SIO_C与SIO_D。

2线的SCCB总线只能是一个主器件对一个从器件控制，但3线SCCB接口可以对多个从器件控制，因此当只有一个从机（slave device）时用两线，有多个从机时用三线。

其中SIO_C只能由主机配置（FPGA），SIO_D是一个三态门，双向数据线，既可以由主机控制，也可以由从机控制。

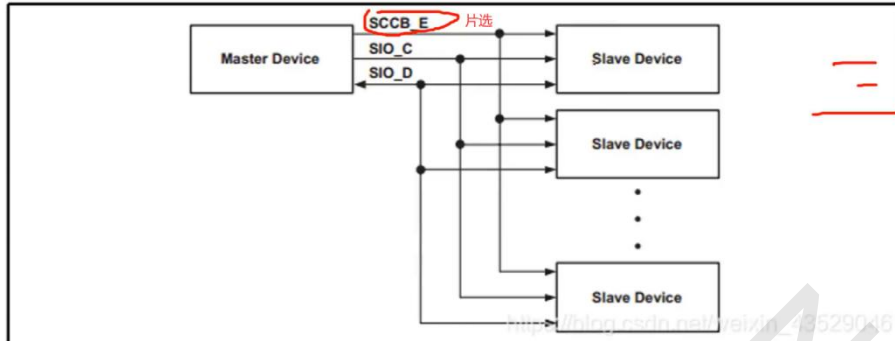
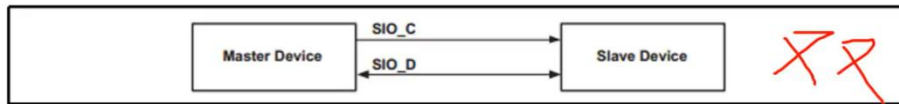
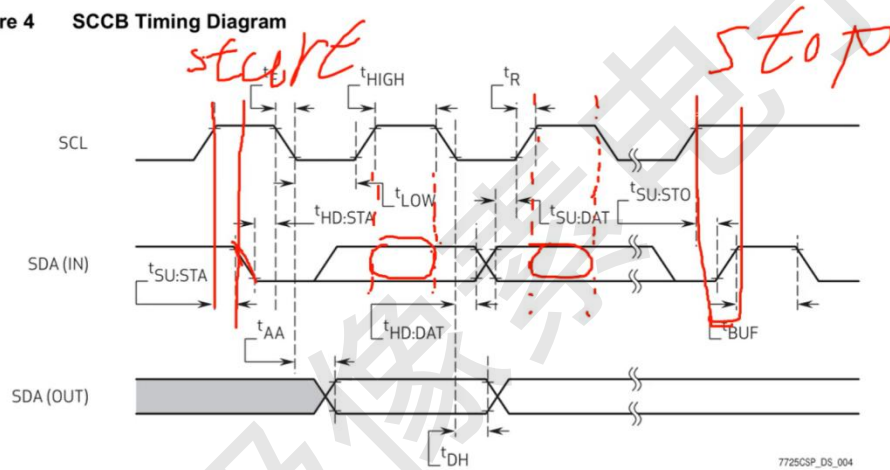
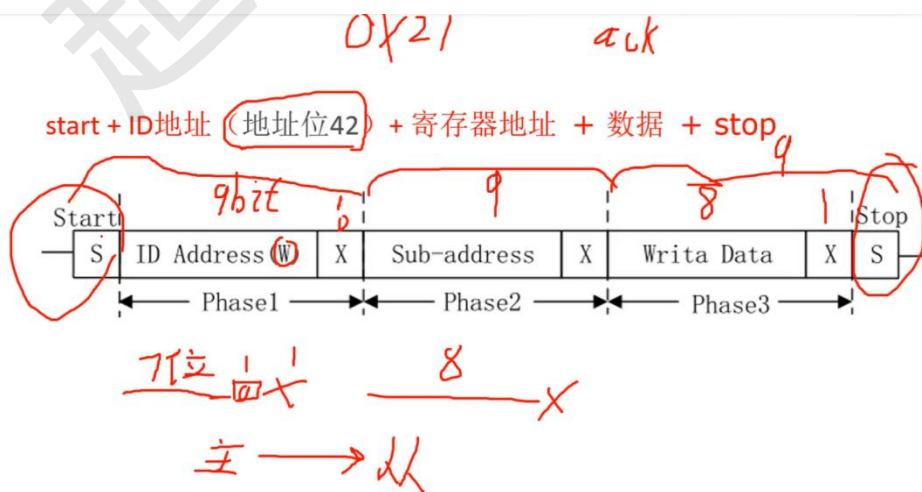


Figure 4 SCCB Timing Diagram

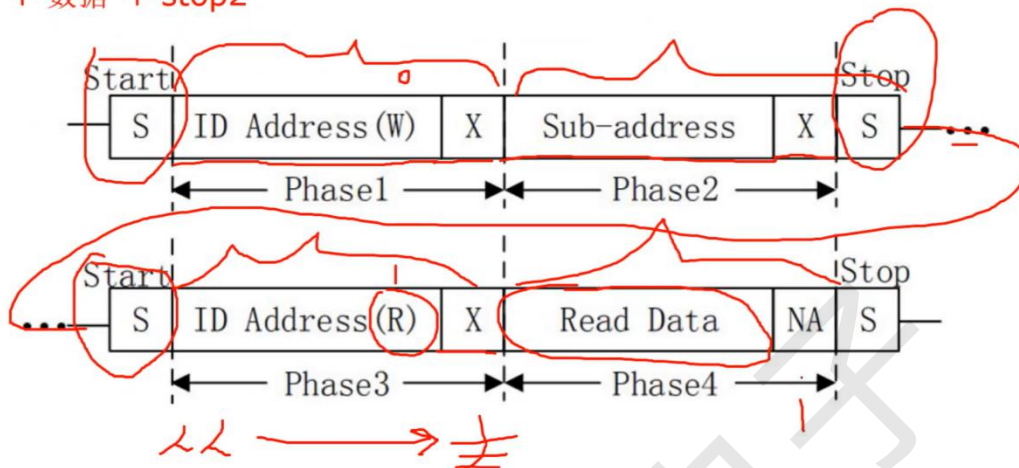


写数据



读数据

start1+ ID地址 (42) + 寄存器地址 + stop1+start 2 +ID地址 (43)
+ 数据 + stop2



摄像头数据时序

行时序

t_{Pclk} : 一个时钟周期。

t_p : 一个像素点的周期, 在 RGB565 和 YUV422 输出格式下, $t_p=2*t_{Pclk}$; Raw 输出格式下, $t_p=t_{Pclk}$ 。

下图为 OV5640 输出图像数据的行时序图。

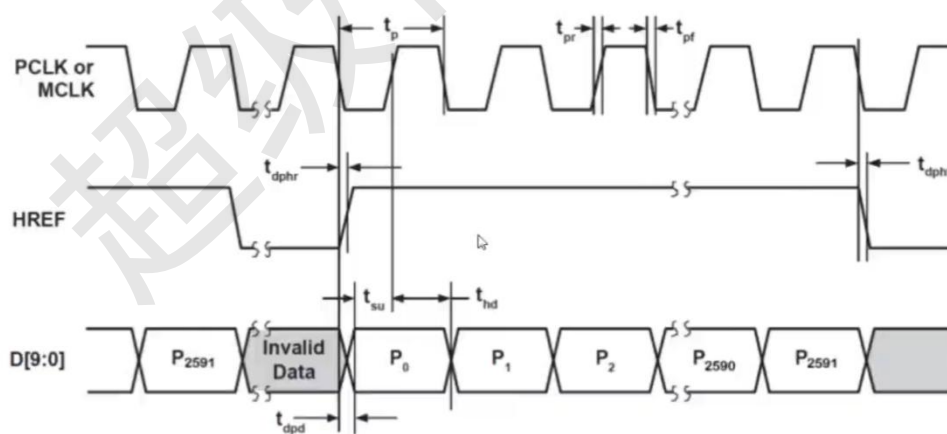


图 33.1.5 OV5640 行时序图

场时序

再来看看帧时序（QSXGA 模式，分辨率 2592*1944），如下图所示：

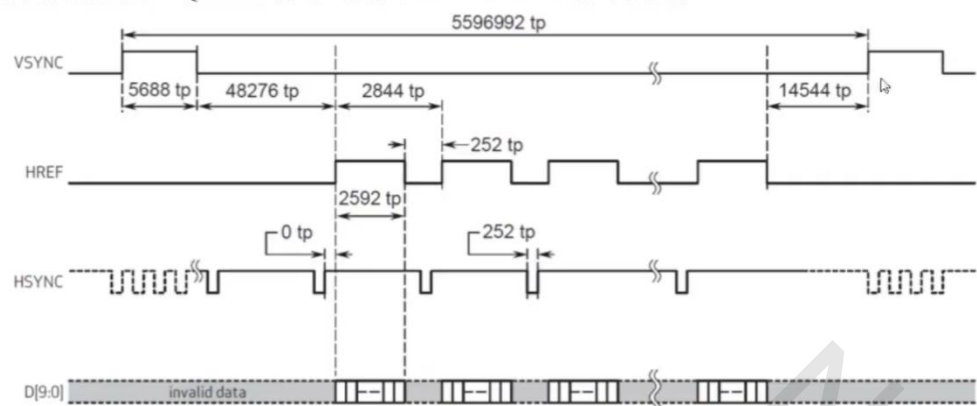


图 33.1.6 OV5640 OSXGA 帧时序

程序

