

《图像采集与传输系统的设计》

出　　品：北京智芯融科技有限公司
主　　讲：智芯融团队
实验平台：智芯融系列FPGA开发板
淘宝店铺：https://shop296232497.taobao.com/
官方网站：http://www.icfusion.cn/

内容：

- 回顾DDR MEMORY读写使用；
- 回顾UART串口协议的读写使用；
- 回顾VESA相关知识
- 回顾I2C协议的相关知识
- 回顾100T开发板HDMI接口芯片ADV7511芯片的配置
- 学习系统设计，包括硬件设计， fpga 模块设计，较大项目如何组织和调试。

目录

1 整体图像采集工程设计
2 图像采集模块OV5640简介配置
3 采集图像数据存储
4 HDMI接口的配置与现实
5 习题

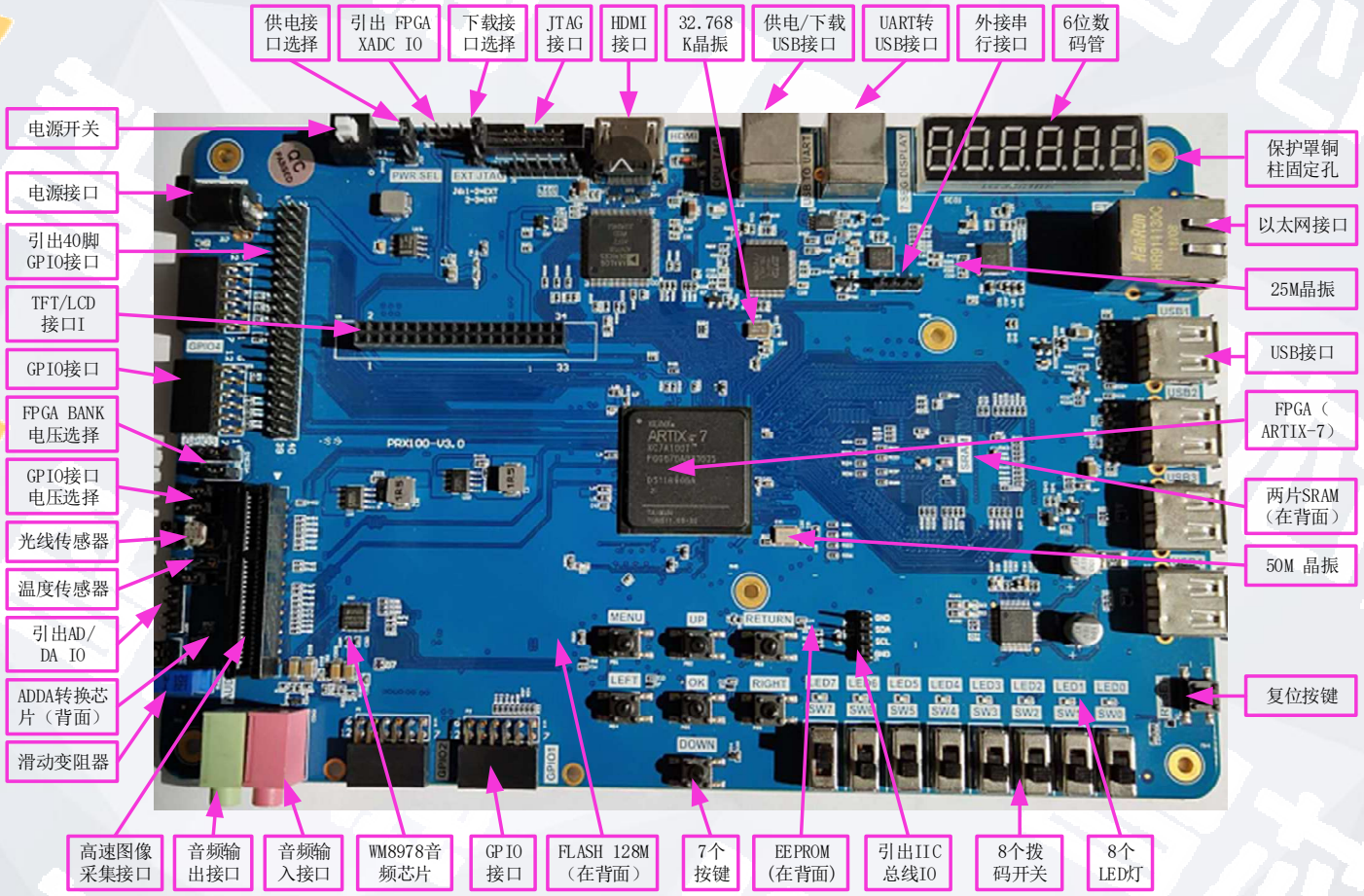
1 整体图像采集工程设计

2 图像采集模块OV5640简介配置

3 采集图像数据存储

4 HDMI接口的配置与现实

5 习题



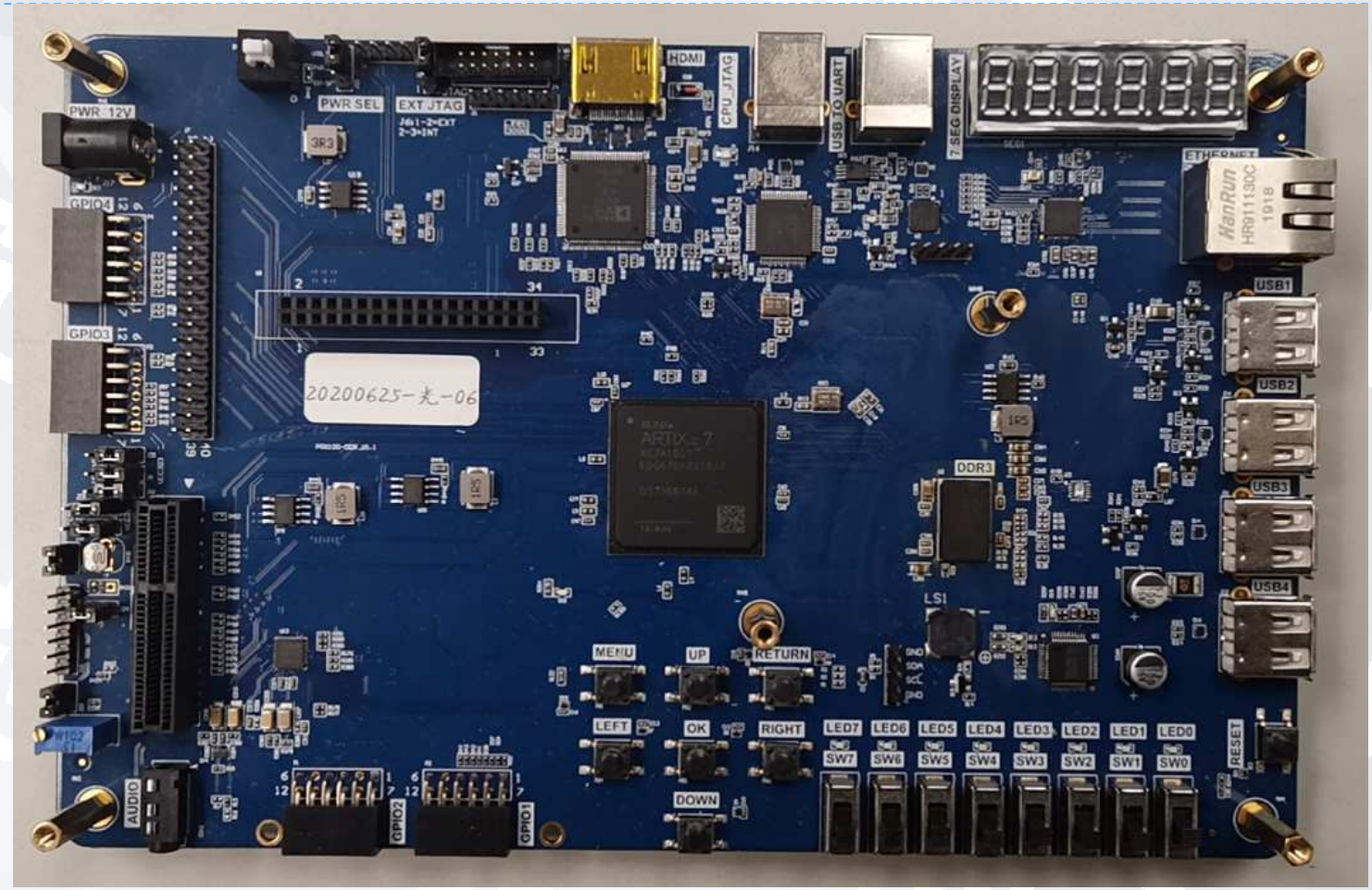
ICF_PRX100T-SRAM资源介绍图

PRX-100T-DDR 与早期的开发板PRX-100T-SRAM不同：

1）将sram 芯片替换为ddr3 芯片

2）audio接口替换为mic，speaker 统一接口

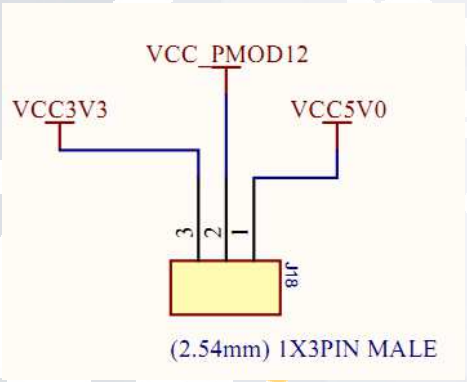
3）增加buzzer（蜂鸣器）

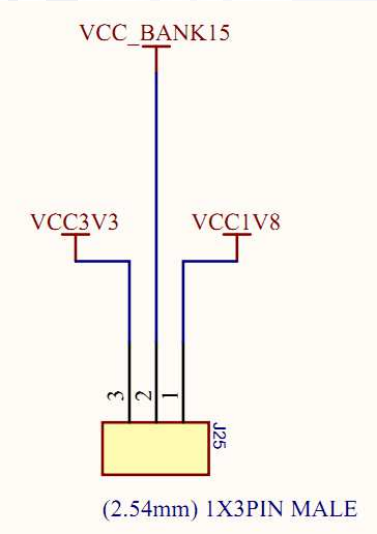


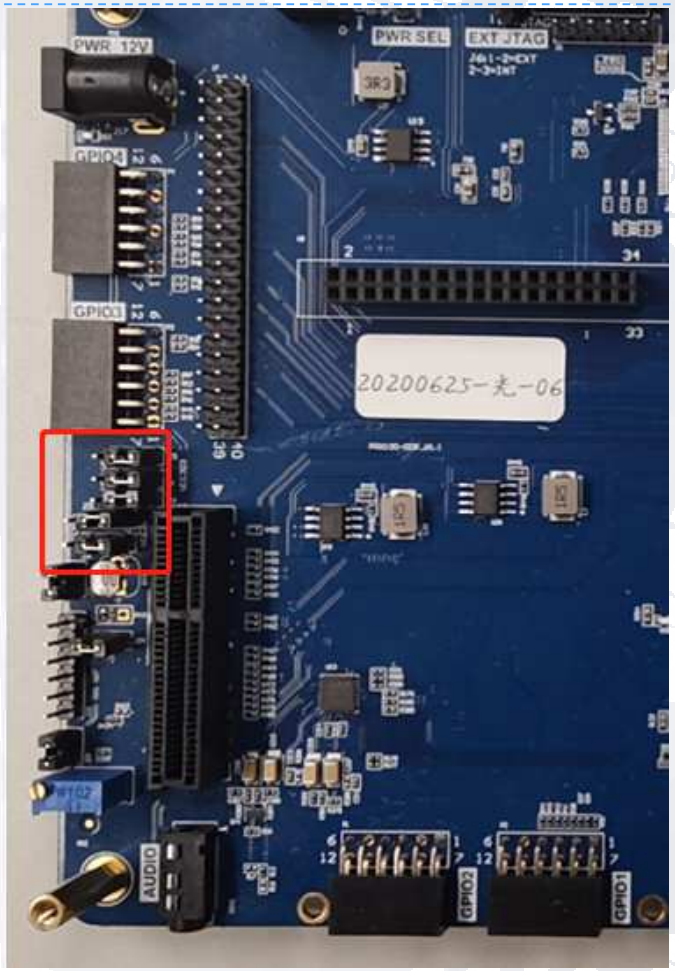
ICF_PRX100T - DDR资源介绍图

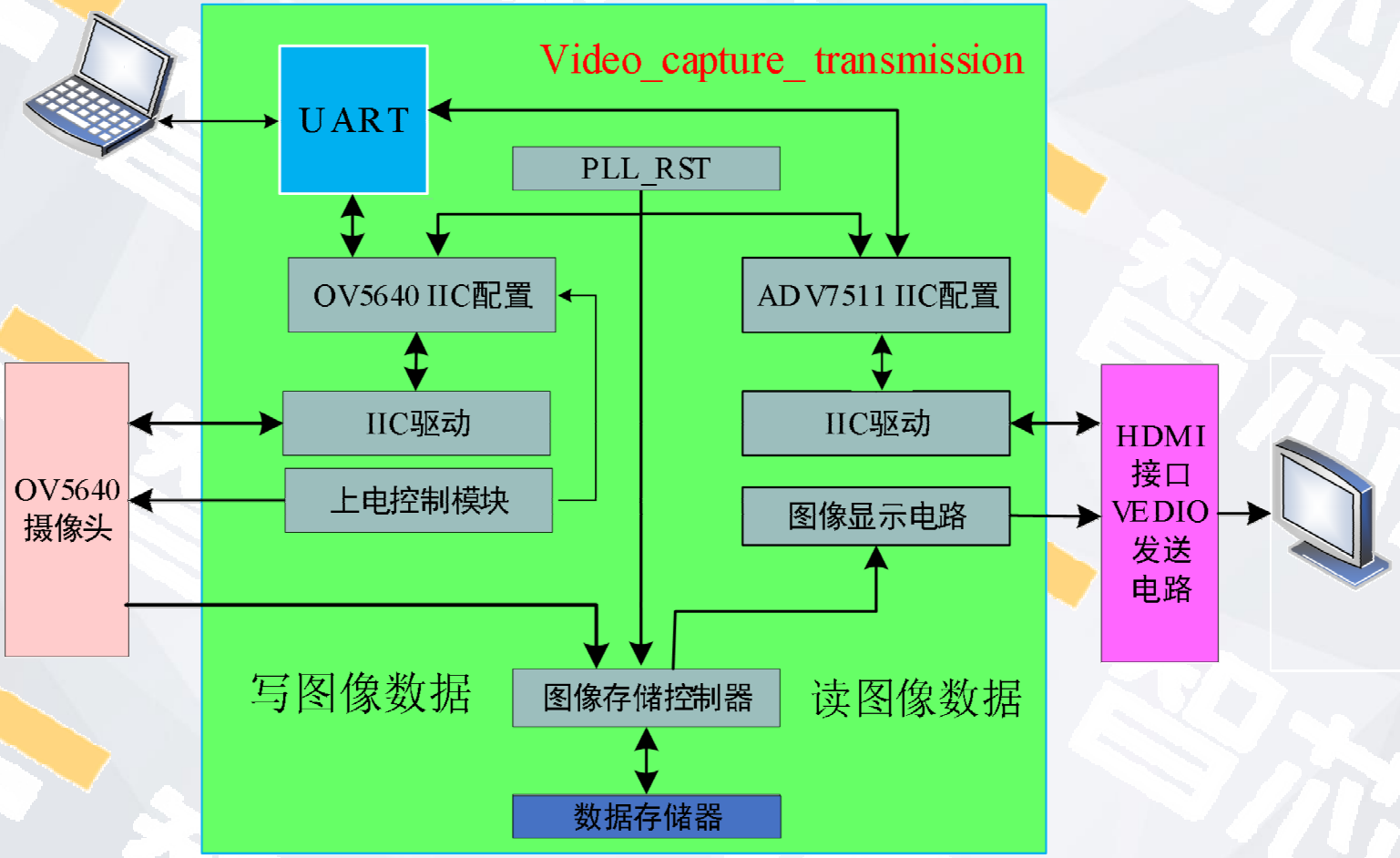
1）J18跳线到 3.3v上 ，将fpga io 口设置为 3.3v ； (右侧）

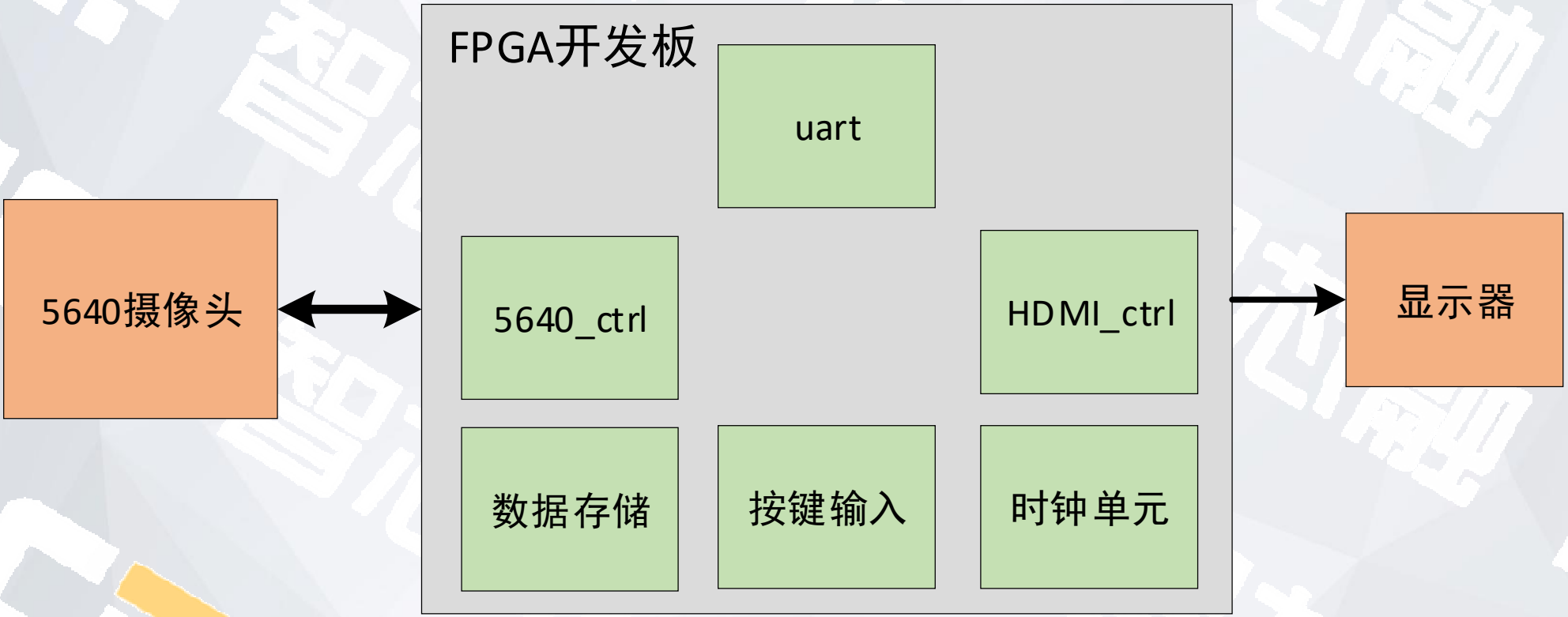
2）J25 跳线到3.3v上， pmod 供电电压为 3.3v （右侧）





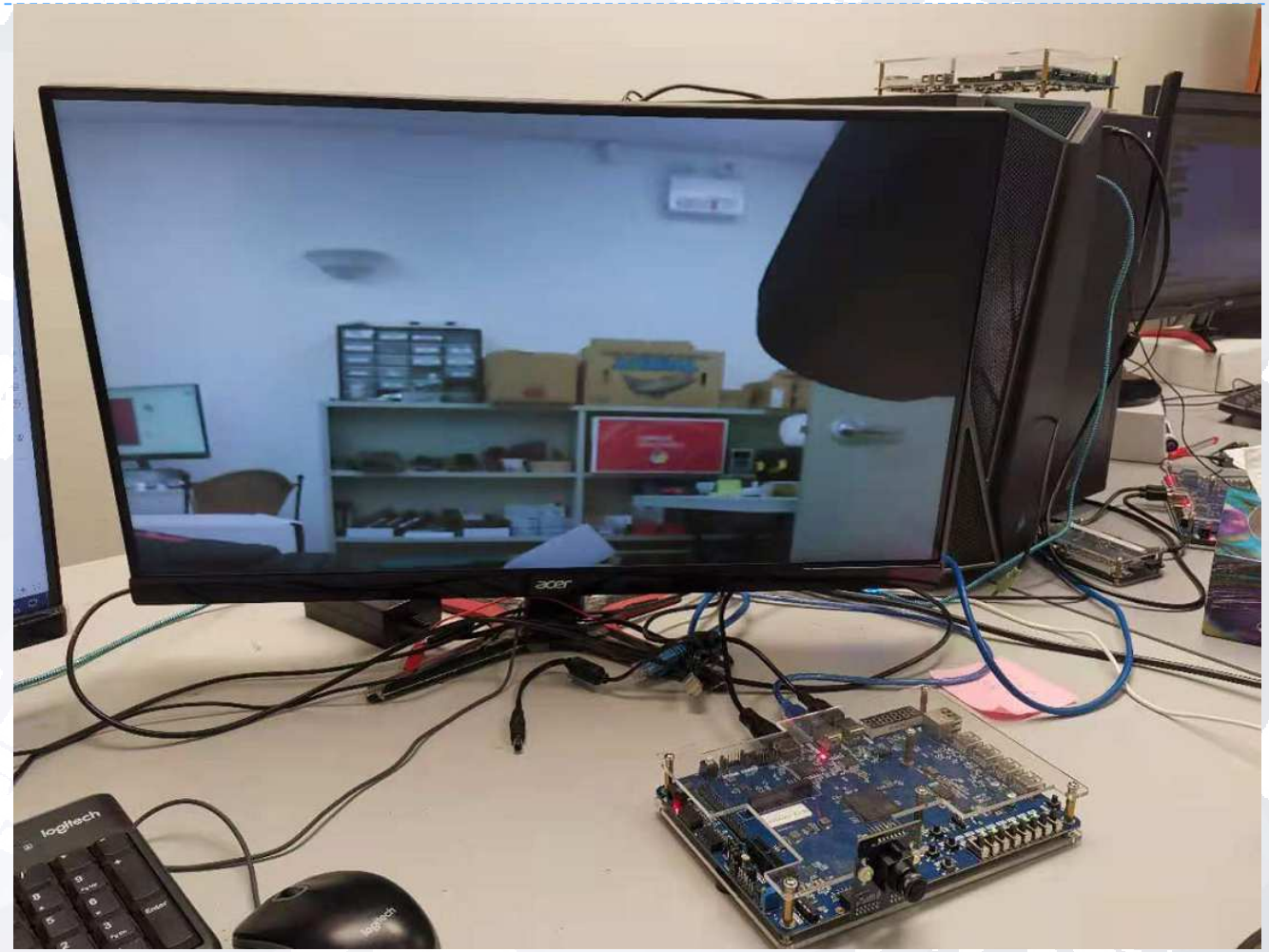



Video_capture_ transmission
UART
PLL_RST
OV5640 IIC配置
ADV7511 IIC配置
IIC驱动
IIC驱动
上电控制模块
图像显示电路
OV5640
摄像头
HDMI
接口
VEDIO
发送
电路
写图像数据
图像存储控制器
读图像数据
数据存储器

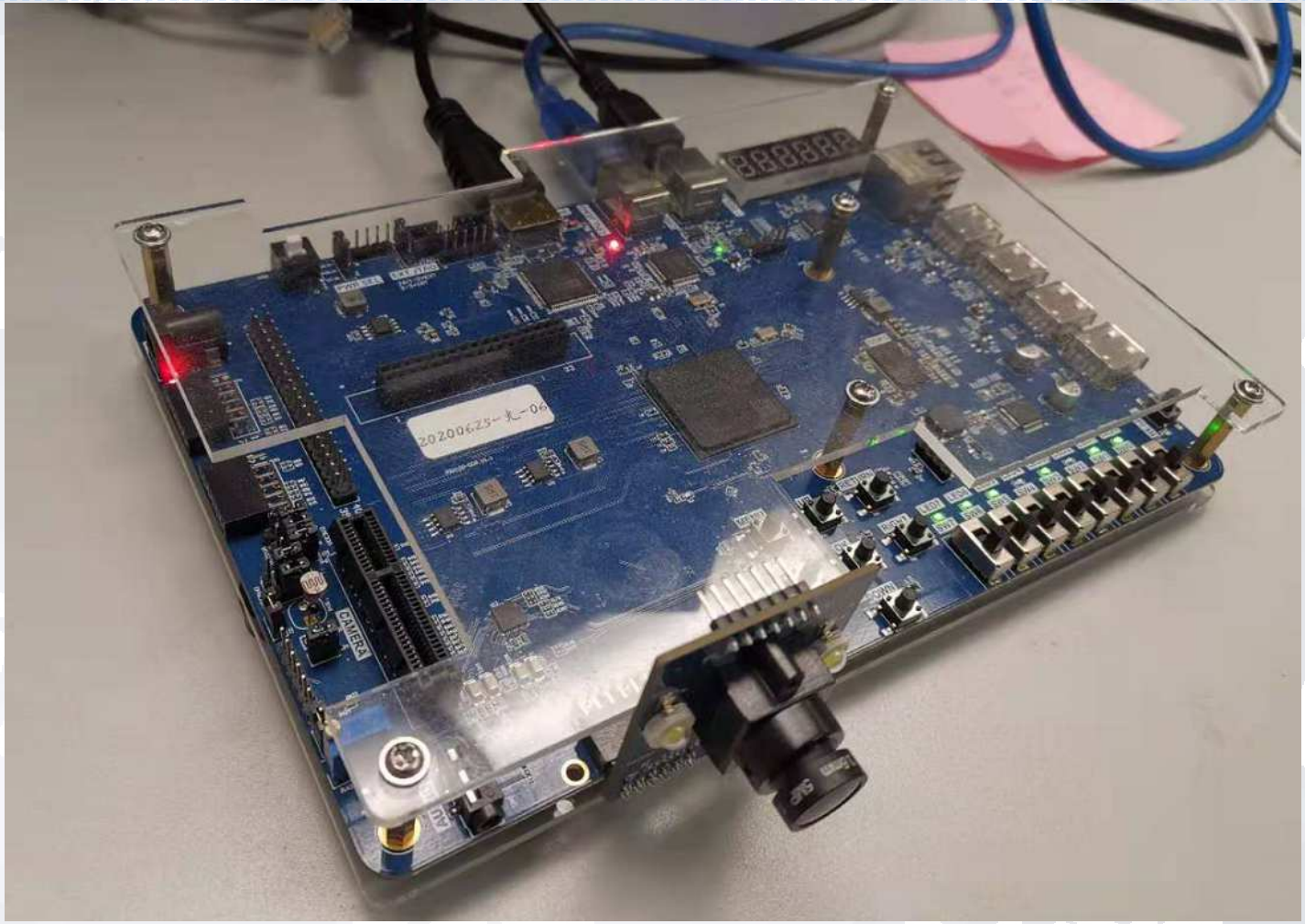


整体工程划分图

硬件连接： 通过hdmi 线连接显示器和开发板； 摄像头模块连接在pmod 端口上



连接fpga jtag 线 和 usb 转 uart 连线。这两根 usb 线既有相应的功能（jtag，uart），同时也为开发板供电使用



1 整体图像采集工程设计

2 图像采集模块OV5640简介配置

3 采集图像数据存储

4 HDMI接口的配置与现实

5 习题



OV5640摄像头实物

OV5640是OmniVision(豪威科技)公司生产的CMOS图像传感器，该传感器功耗低，可靠性高以及采集速度快，主要应用在玩具，安防监控电脑等多媒体领域

OV5640感光阵列达到2624*1964，能实现最快15fps QSXVGA（2592*1944）或者90fps VGA（640*480）分辨率的图像采集。

table 2-1 format and frame rate

format	resolution	frame rate	scaling method	pixel clock
5 Mpixel	2592x1944	15 fps	full resolution (dummy 16 pixel horizontal, 8 lines) 2608x1952 with dummy	96/192 MHz
1280x960	1280x960	45 fps	subsampling in vertical and horizontal 1296x968 supports 2x2 binning	96/192 MHz
1080p	1920x1080	30 fps	cropping from full resolution 1936x1088 with dummy pixels	96/192 MHz
720p	1280x720	60 fps	cropping 2592x1944 to 2560x1440 subsampling in vertical and horizontal 1296x728 with dummy supports 2x2 binning	96/192 MHz
VGA	640x480	90 fps	subsampling from 1280x960 648x484 with dummy supports 2x2 binning	48/96 MHz
QVGA	320x240	120 fps	subsampling from 1280x960 324x242 with dummy supports 2x2 binning	24/48 MHz

ov5640功能：

自动图像控制：

- 自动曝光控制
- 自动白平衡控制
- 自动带滤波
- 50Hz/60Hz 流明检测

可编程帧率设置

图像质量控制：

- 色彩饱和度
- 边沿增强
- gamma 设置
- 噪声消除

ov5640功能：

输出格式：

- RGB565/555/444
- YUV422/420
- YCbCr422
- 压缩格式

支持video or 拍照模式

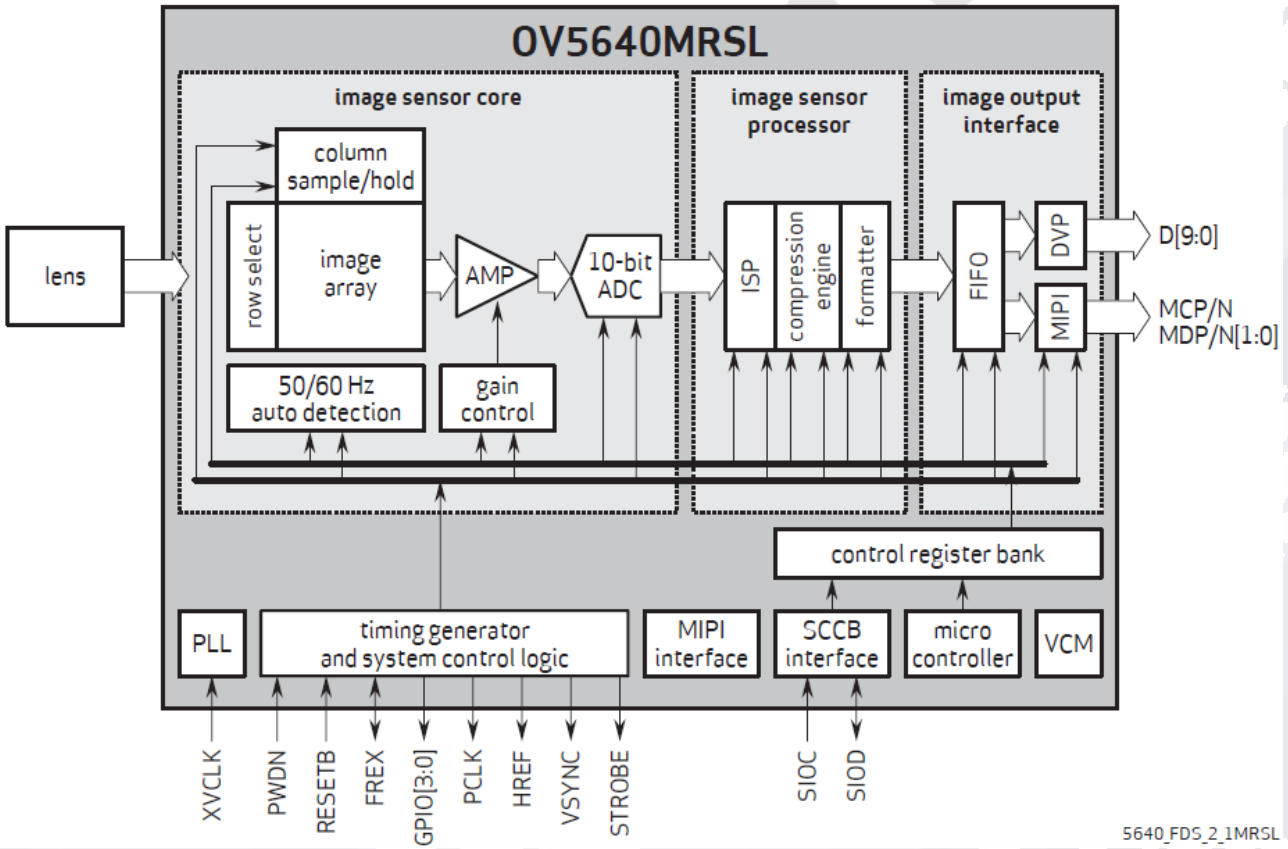
支持LED 和 flash strobe 模式

DVP （digital video port）格式输出

MIPI 格式输出

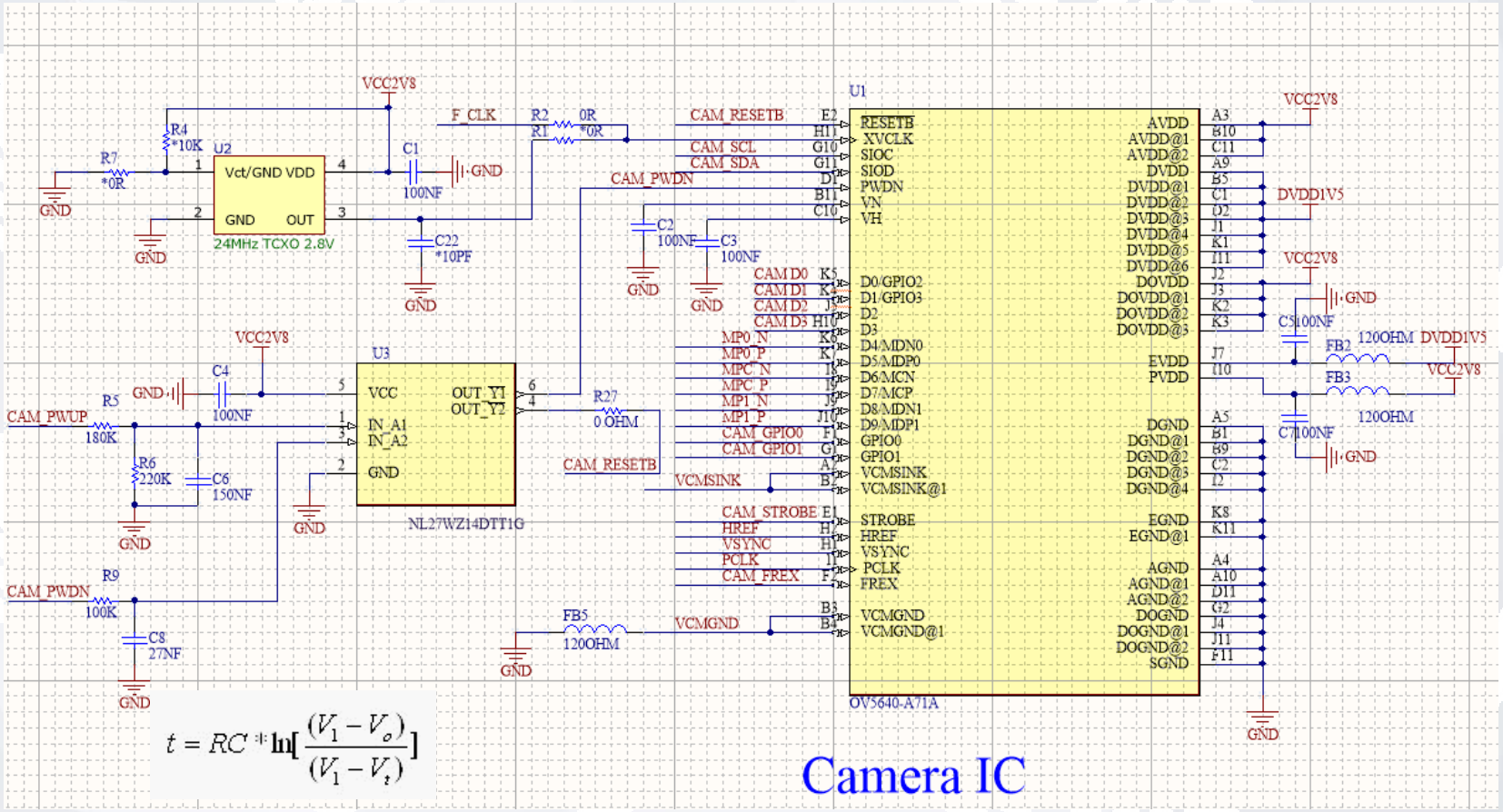
OV5640支持LED补光、MIPI（移动产业处理器接口）输出接口和DVP（数字视频并行）输出接口选择、ISP（图像信号处理）以及AFC（自动聚焦控制）等功能。

OV5640的功能框图

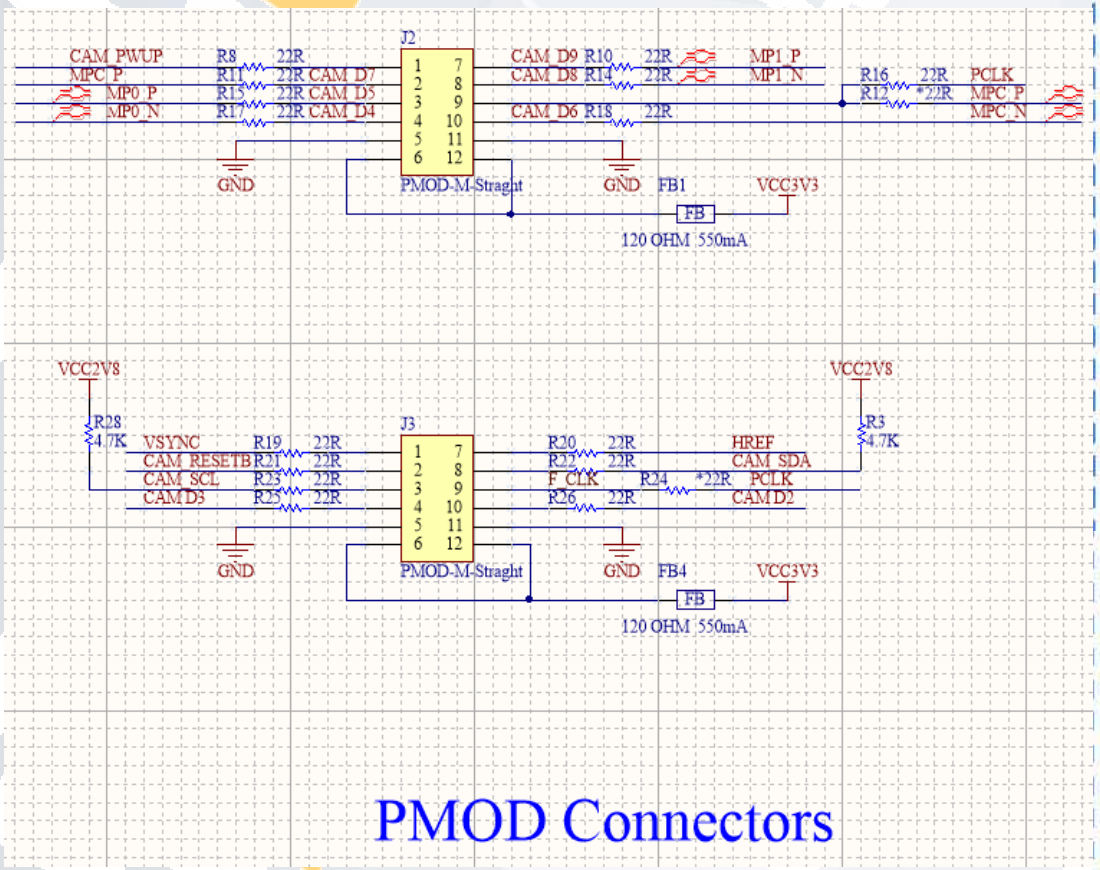


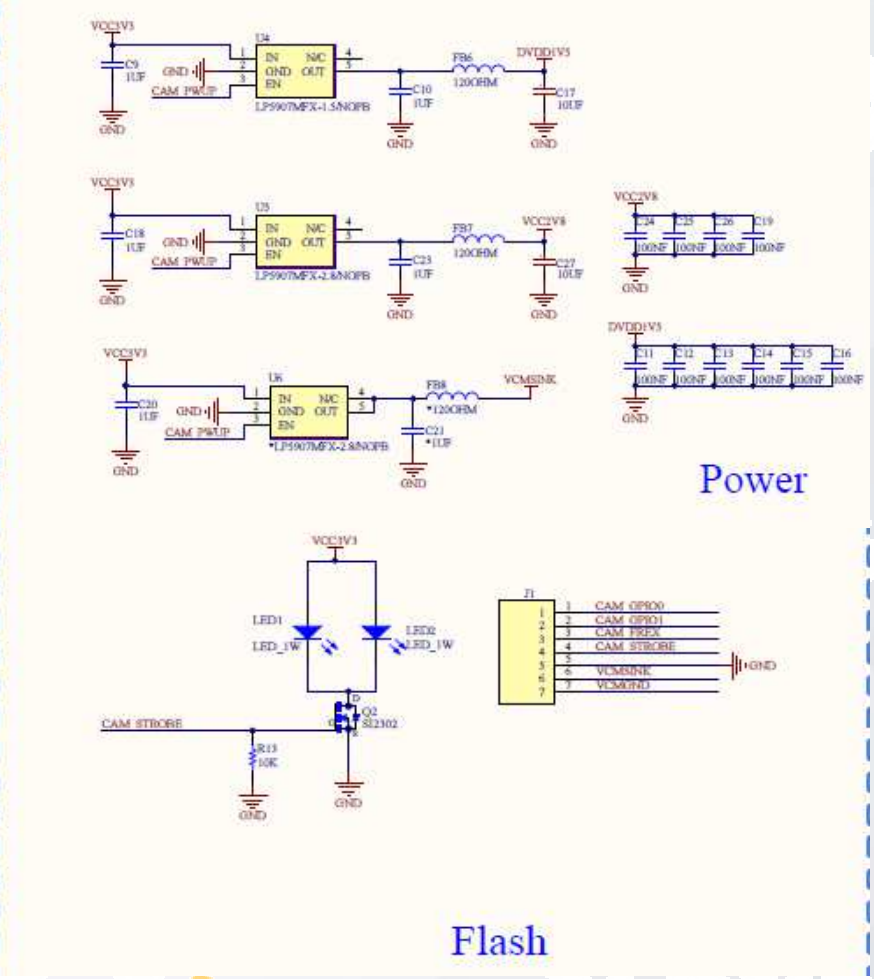
在C上串联一个电阻，当电路给C充电时，R起到了一个限流的作用，可见R和C越大，充电的时间就越长，延时的时间就是这个充电过程。

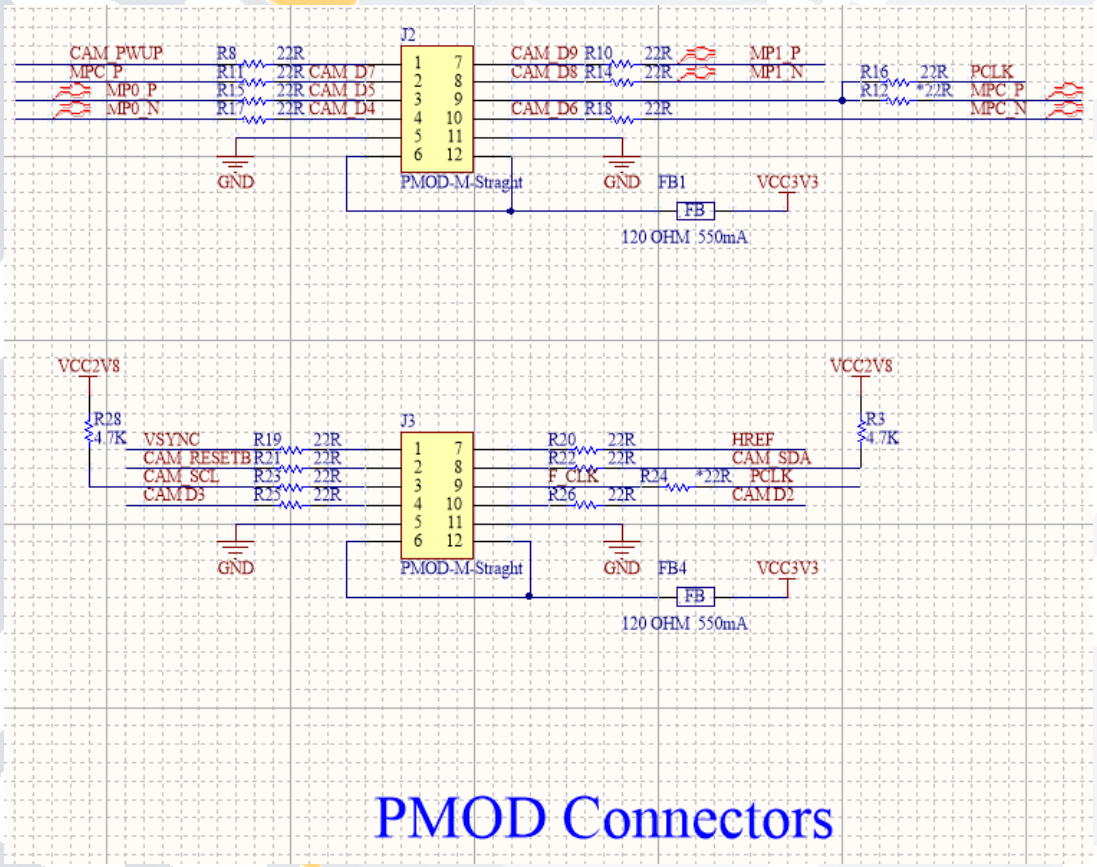
还有一个放电回路与它并联起到置零作用



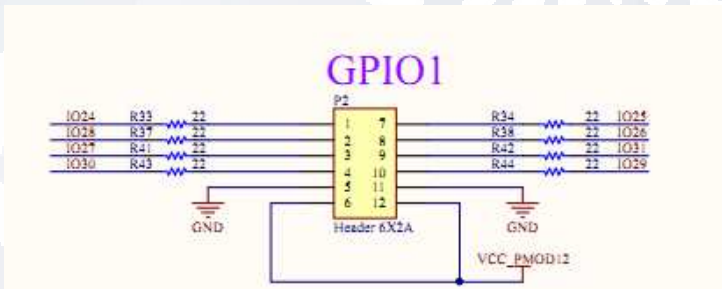
$$t = RC * \ln\left[\frac{(V_1 - V_o)}{(V_1 - V_t)}\right]$$


J2
CAM_PWUP
MPC_P
MP0_P
MP0_N
R8 22R
R11 22R CAM_D7
R15 22R CAM_D5
R17 22R CAM_D4
CAM_D9 R10 22R
CAM_D8 R14 22R
CAM_D6 R18 22R
MP1_P
MP1_N
R16 22R PCLK
R12 *22R MPC_P
MPC_N
GND
PMOD-M-Straght
GND FB1 VCC3V3
FB
120 OHM 550mA
VCC2V8
R28 4.7K
VSYNC R19 22R
CAM_RESETB R21 22R
CAM_SCL R23 22R
CAM_D3 R25 22R
J3
R20 22R HREF
R22 22R CAM_SDA
P_CLK R24 *22R PCLK
R26 22R CAM_D2
VCC2V8
R3 4.7K
GND
PMOD-M-Straght
GND FB4 VCC3V3
FB
120 OHM 550mA
PMOD Connectors


VCC3V3
U4
C9 1UF
GND
CAM_PWUP
LP5907MFX-1.5/NOPB
C10 1UF
FB6 120OHM
DVDD1V5
C17 10UF
VCC3V3
U5
C18 1UF
CAM_PWUP
LP5907MFX-2.8/NOPB
C23 1UF
FB7 120OHM
VCC2V8
C27 10UF
VCC2V8
C24 100NF
C25 100NF
C26 100NF
C19 100NF
DVDD1V5
C11 100NF
C12 100NF
C13 100NF
C14 100NF
C15 100NF
C16 100NF
VCC3V3
U6
C20 1UF
CAM_PWUP
*LP5907MFX-2.8/NOPB
C21 *1UF
FB8 *120OHM
VCMSINK
Power
VCC3V3
LED1
LED_1W
LED2
LED_1W
CAM_STROBE
R13 10K
J1
CAM_OPRO0
CAM_OPRO1
CAM_FREX
CAM_STROBE
VCMSINK
VCMGND
GND
Flash



PMOD Connectors



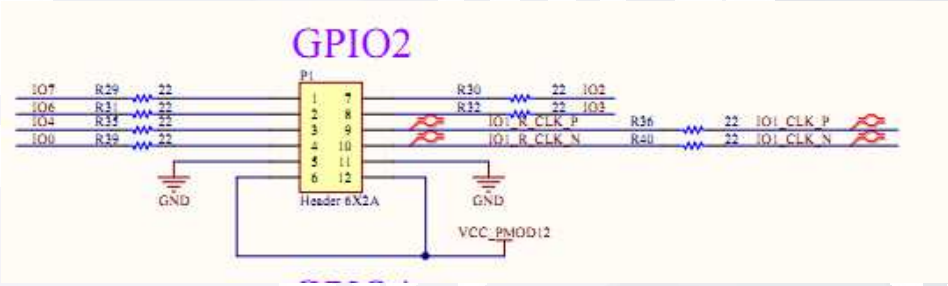
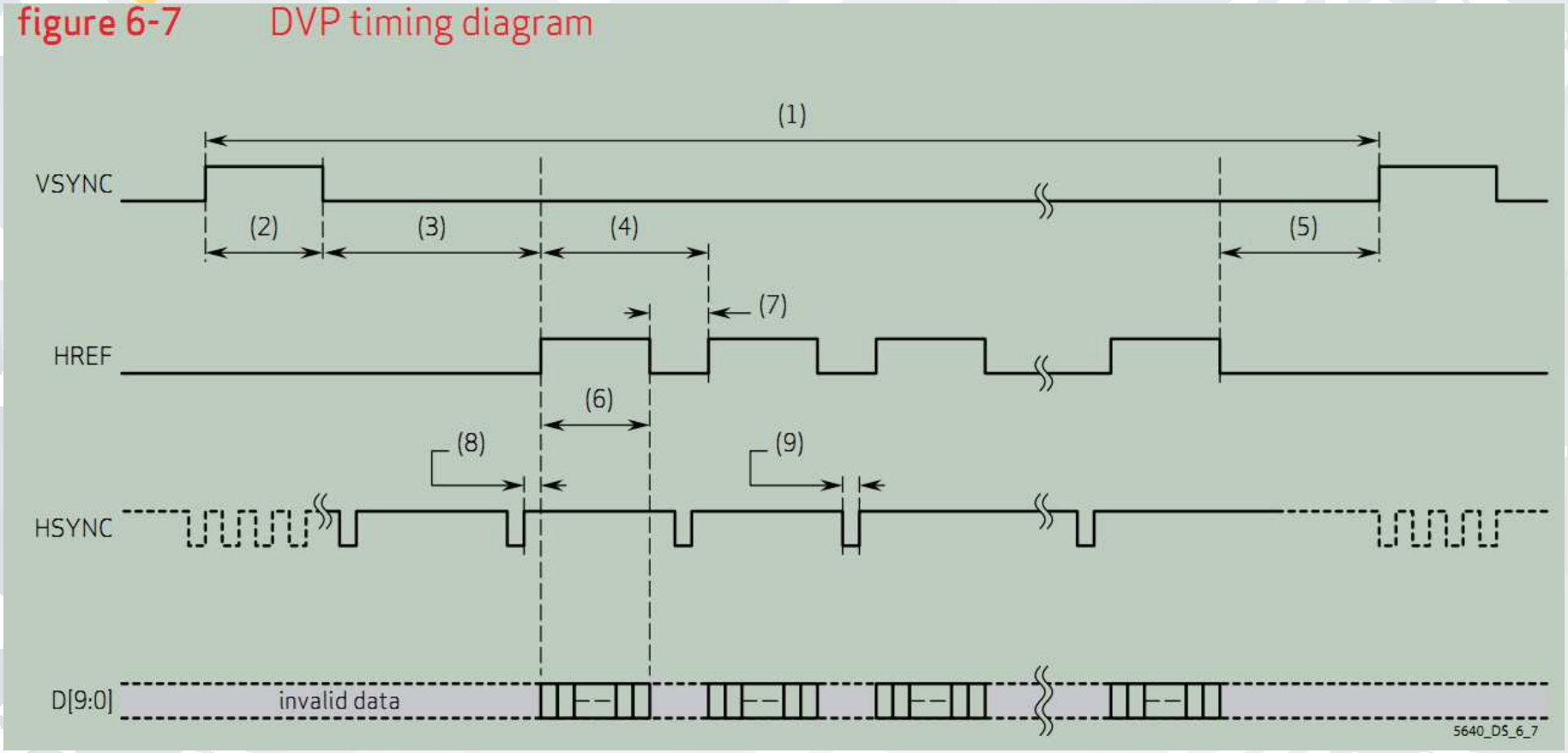
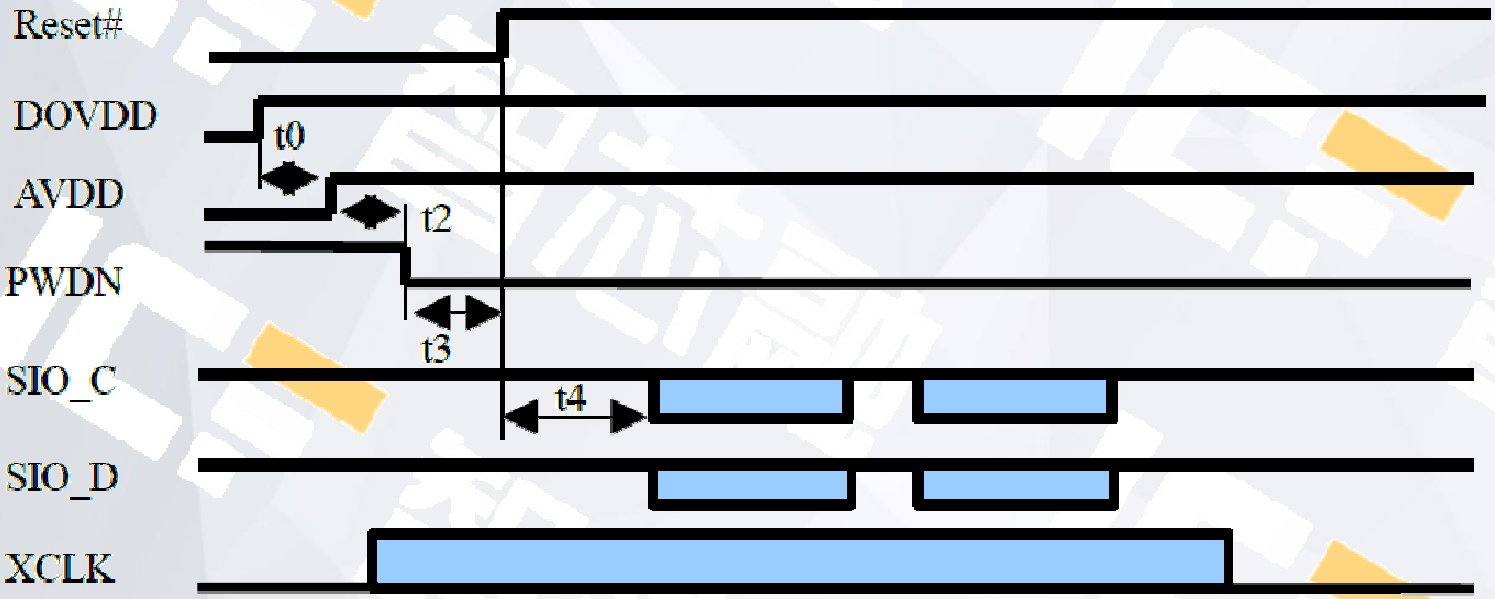


figure 6-7　DVP timing diagram





t0: >= 0ms. Delay from DOVDD stable to AVDD stable.
t2: >= 5ms. Delay from AVDD stable to sensor power up stable.
t3: >= 1ms. Delay from sensor power up stable to Reset# pull high.
t4: >=20ms. Delay from Reset pull high to SCCB initialization.

步骤1：ResetB拉低，复位OV5640。PWDN引脚拉高。

步骤2：DOVDD 和 AVDD两路最好同时上电，这在模组的电源设计中实现。

步骤3：等电源稳定5毫秒后，拉低PWDN。

步骤4：PWDN 置低1毫秒后，拉高 Reset

步骤5：20毫秒后，初始化OV5640的SCCB寄存器设置



SCCB总线介绍

SCCB（Serial Camera Control Bus，串行摄像头控制总线）是由OV公司定义和发展的三线式串行总线。

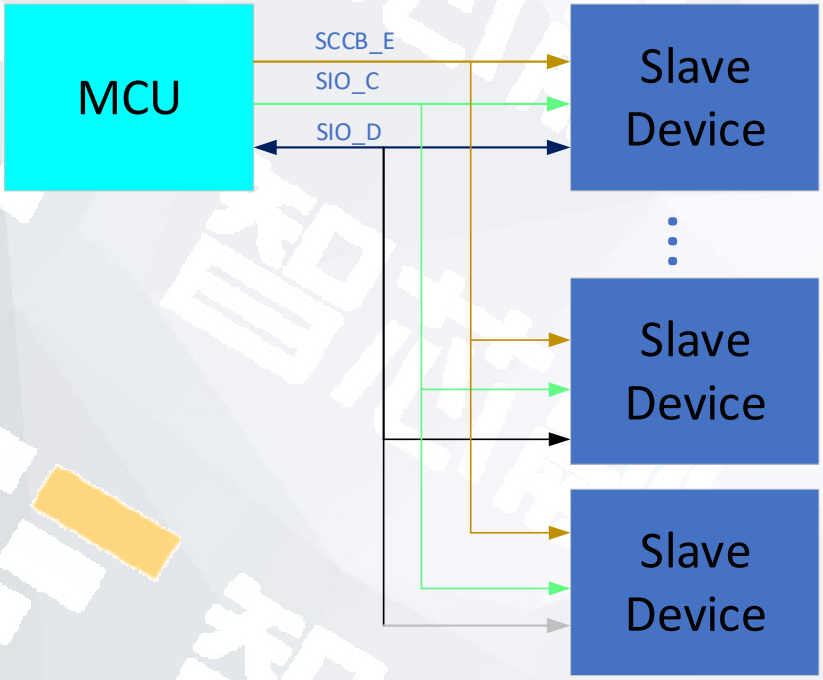
该总线控制着OV系列摄像头大部分的功能，包括图像数据格式、分辨率以及图像处理参数等。

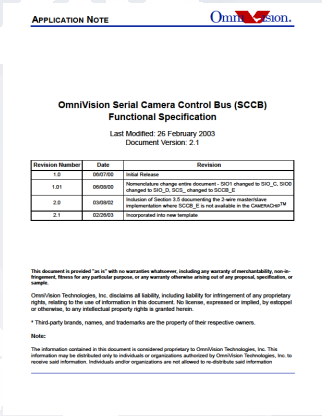
OV公司为了减少传感器引脚的封装，现在SCCB总线大多采用两线式接口总线。（类似IIC总线）

SCCB总线介绍

SCCB有两种工作模式，一主多从，一主一从模式

一主机多从机 也即3线操作：（通过控制使能端SCCB_E控制选中的从机）




APPLICATION NOTE

OmniVision Serial Camera Control Bus (SCCB) Functional Specification

Last Modified: 26 February 2003
Document Version: 2.1

Revision Number	Date	Revision
1.0	06/07/00	Initial Release
1.01	06/09/00	Nomenclature change entire document. SIO1 changed to SIO_C, SIO0 changed to SIO_D, SCL changed to SCCB_E
2.0	03/09/02	Inclusion of Section 3.0 documenting the 3-wire master/slave implementation where SCCB_E is not available in the OmniVision™
2.1	02/26/03	Incorporated into new template

This document is provided "as is" with no warranties whatsoever, including any warranty of merchantability, non-infringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample.

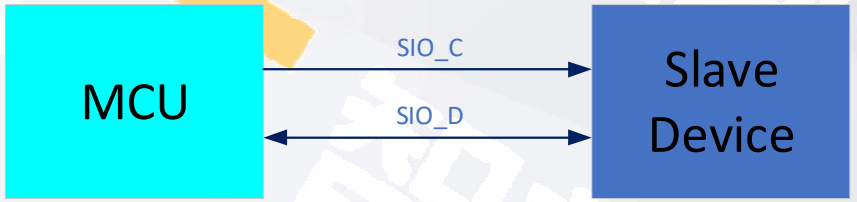
OmniVision Technologies, Inc. disclaims all liability, including liability for infringement of any proprietary rights, relating to the use of information in this document. No license, expressed or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

* Third-party brands, names, and trademarks are the property of their respective owners.

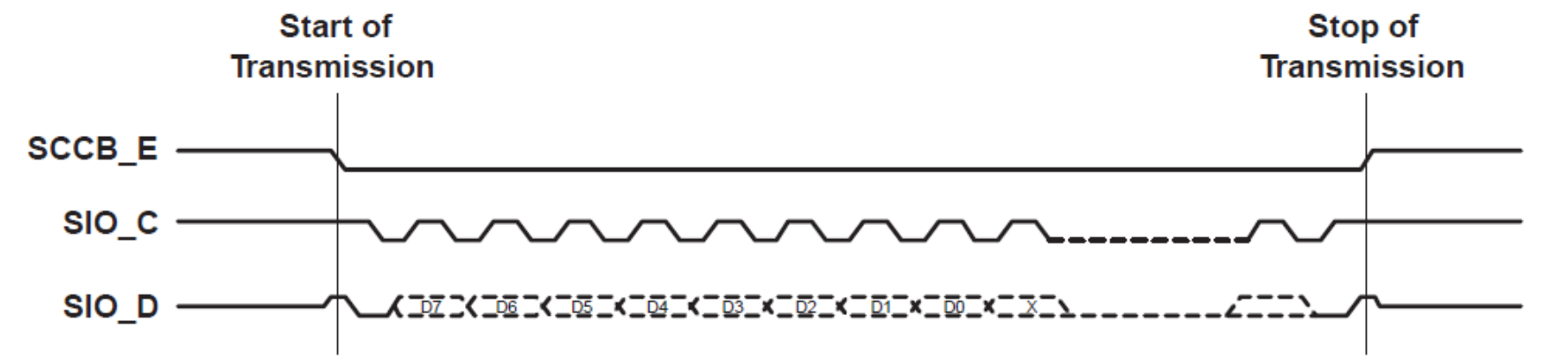
Note:

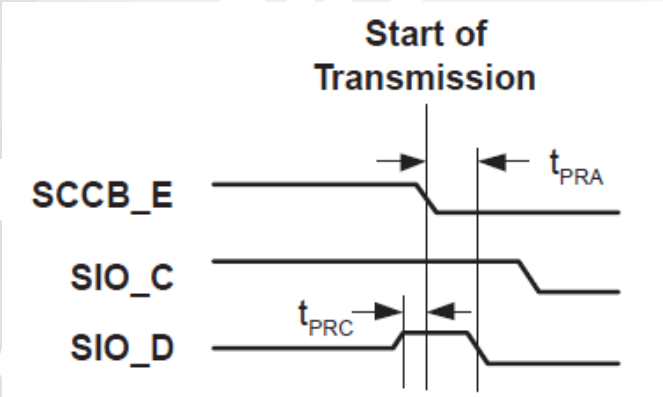
The information contained in this document is considered proprietary to OmniVision Technologies, Inc. This information may be distributed only to individuals or organizations authorized by OmniVision Technologies, Inc. to receive said information. Individuals and/or organizations are not allowed to re-distribute said information

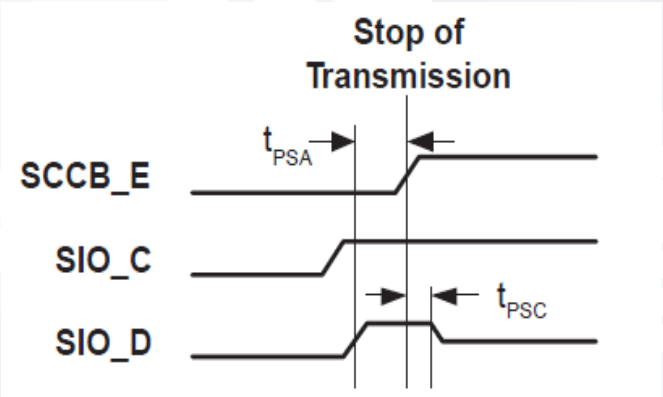
一主一从 也即2线操作：（默认SCCB_E被拉低）

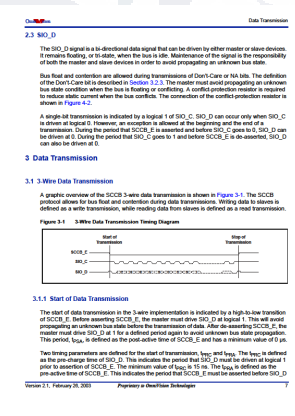


SCCB总线介绍









2.3 SIO_D

The SIO_D signal is a bi-directional data signal that can be driven by either master or slave devices. It remains floating, or tri-state, when the bus is idle. Maintenance of the signal is the responsibility of both the master and slave devices in order to avoid propagating an unknown bus state.

Bus float and contention are allowed during transmissions of Don't-Care or NA bits. The definition of the Don't-Care bit is described in Section 3.2.3. The master must avoid propagating an unknown bus state condition when the bus is floating or conflicting. A conflict-protection resistor is required to reduce static current when the bus conflicts. The connection of the conflict-protection resistor is shown in Figure 4-2.

A single-bit transmission is indicated by a logical 1 of SIO_C. SIO_D can occur only when SIO_C is driven at logical 0. However, an exception is allowed at the beginning and the end of a transmission. During the period that SCCB_E is asserted and before SIO_C goes to 0, SIO_D can be driven at 0. During the period that SIO_C goes to 1 and before SCCB_E is de-asserted, SIO_D can also be driven at 0.

3 Data Transmission

3.1 3-Wire Data Transmission

A graphic overview of the SCCB 3-wire data transmission is shown in Figure 3-1. The SCCB protocol allows for bus float and contention during data transmissions. Writing data to slaves is defined as a write transmission, while reading data from slaves is defined as a read transmission.

Figure 3-1　3-Wire Data Transmission Timing Diagram

3.1.1 Start of Data Transmission

The start of data transmission in the 3-wire implementation is indicated by a high-to-low transition of SCCB_E. Before asserting SCCB_E, the master must drive SIO_D at logical 1. This will avoid propagating an unknown bus state before the transmission of data. After de-asserting SCCB_E, the master must drive SIO_D at 1 for a defined period again to avoid unknown bus state propagation. This period, t_{PSA}, is defined as the post-active time of SCCB_E and has a minimum value of 0 µs.

Two timing parameters are defined for the start of transmission, t_{PRC} and t_{PRA}. The t_{PRC} is defined as the pre-charge time of SIO_D. This indicates the period that SIO_D must be driven at logical 1 prior to assertion of SCCB_E. The minimum value of t_{PRC} is 15 ns. The t_{PRA} is defined as the pre-active time of SCCB_E. This indicates the period that SCCB_E must be asserted before SIO_D

Version 2.1, February 26, 2003　Proprietary to OmniVision Technologies　7

IIC总线知识回顾

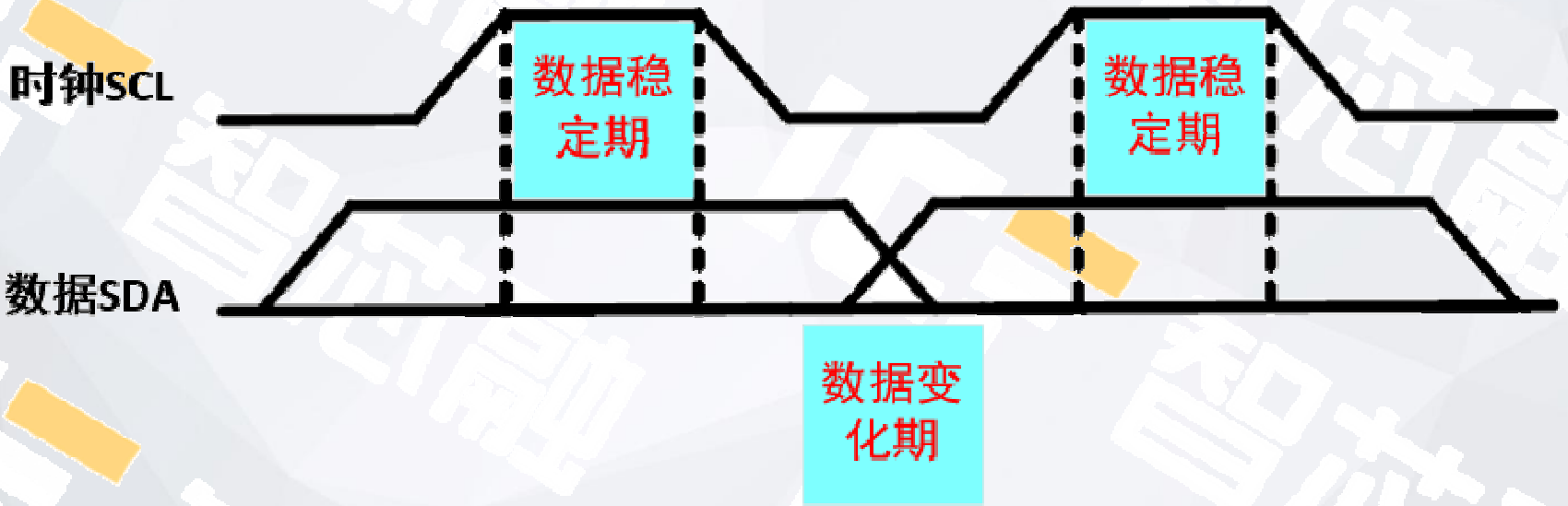
起始信号：在时钟（SCL）为高电平的时候，数据总线（SDA）由高到低的跳变为总线

停止信号：在时钟（SCL）为高电平的时候，数据总线（SDA）由低到高的跳变为总线



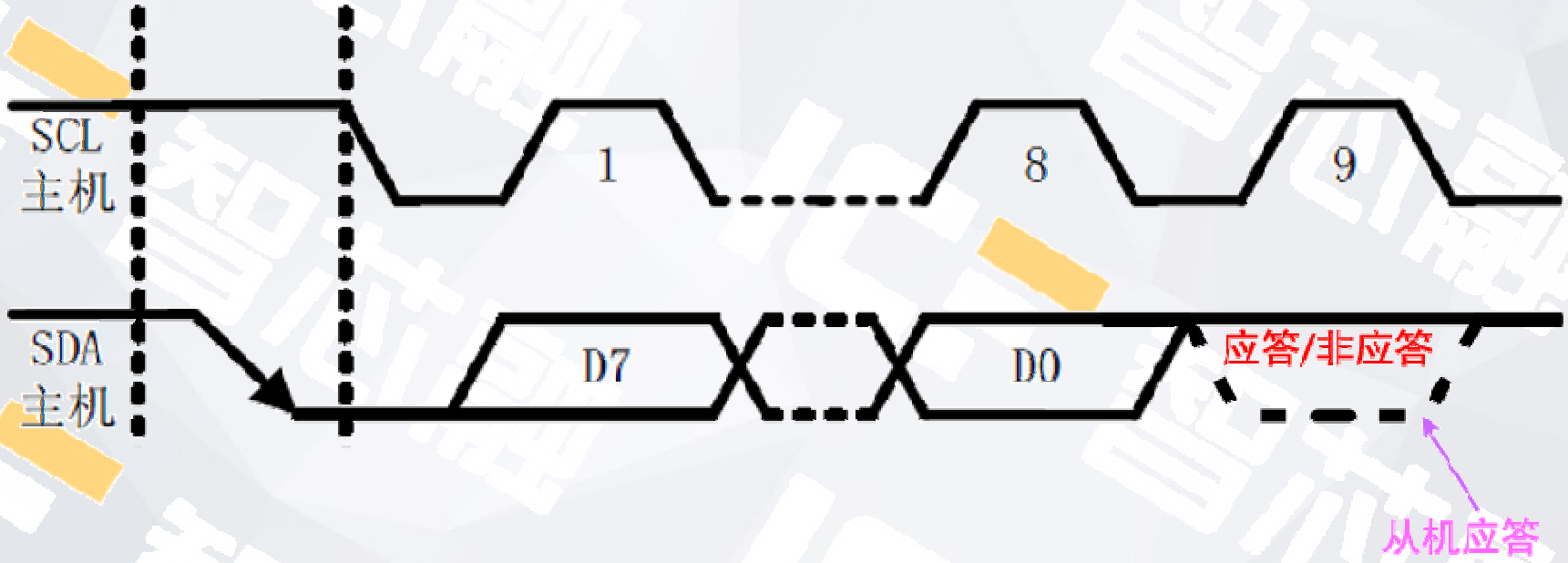
IIC总线知识回顾

在时钟（SCL）为高电平的时候，数据总线（SDA）必须保持稳定，所以数据总线（SDA）在时钟（SCL）为低电平的时候才能改变。



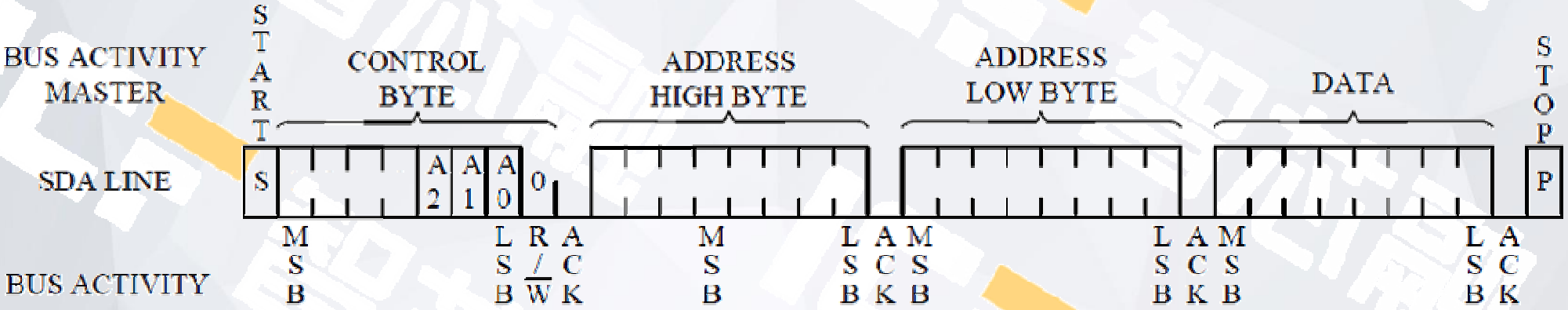
IIC总线知识回顾

应答：当IIC 主机将8 位数据传出后，会将数据总线（SDA）释放，等待从机应答（低电平0 表示应答，1 表示非应答）



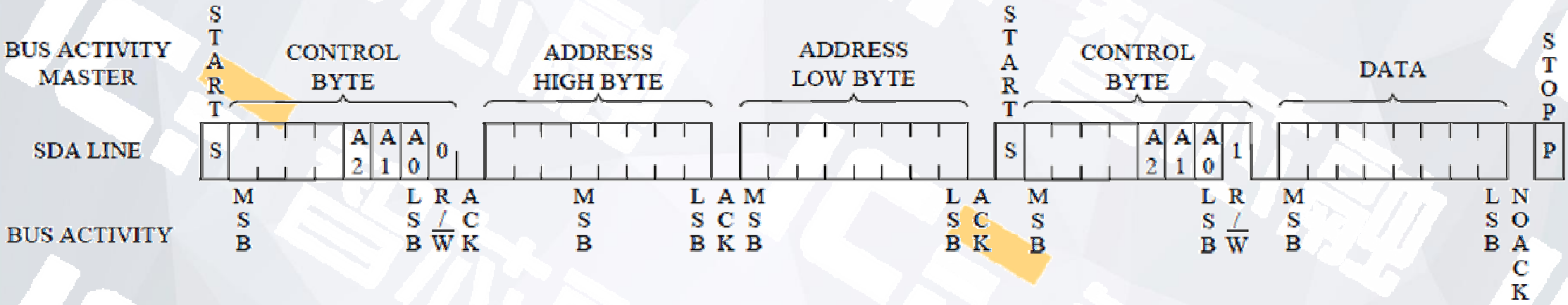
IIC总线知识回顾

双/单字节写时序



IIC总线知识回顾

双/单字节读时序

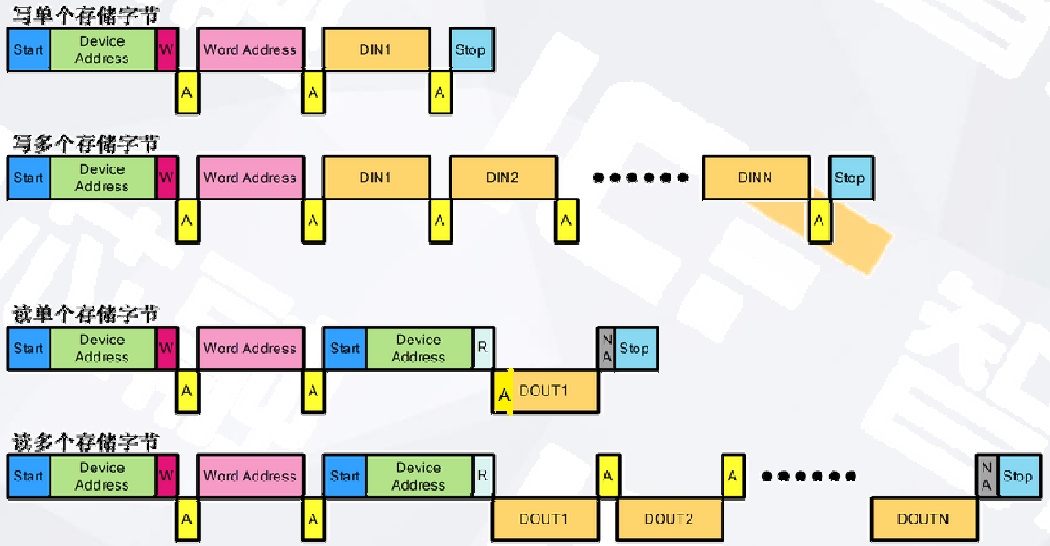


SCCB总线与IIC总线异同

1、SCCB总线写协议中，第9位传输数据为占位数据，而IIC写传输协议为应答位。

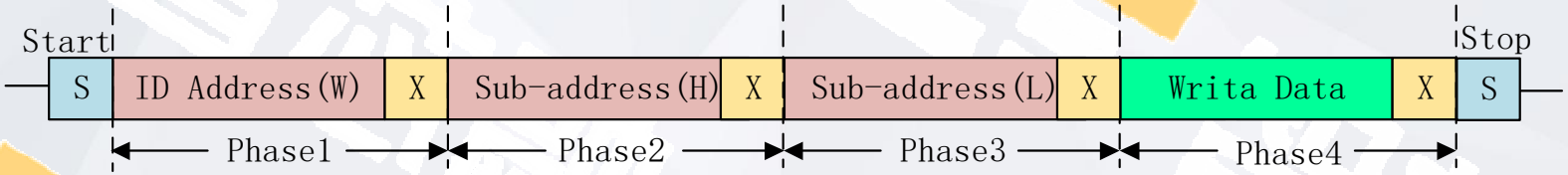
2、SCCB总线写协议每次传输过程只能写入slave一个byte数据。

3、SCCB总线读协议中没有重复开始的概念，在写完寄存器地址后，然后发起停止信号。

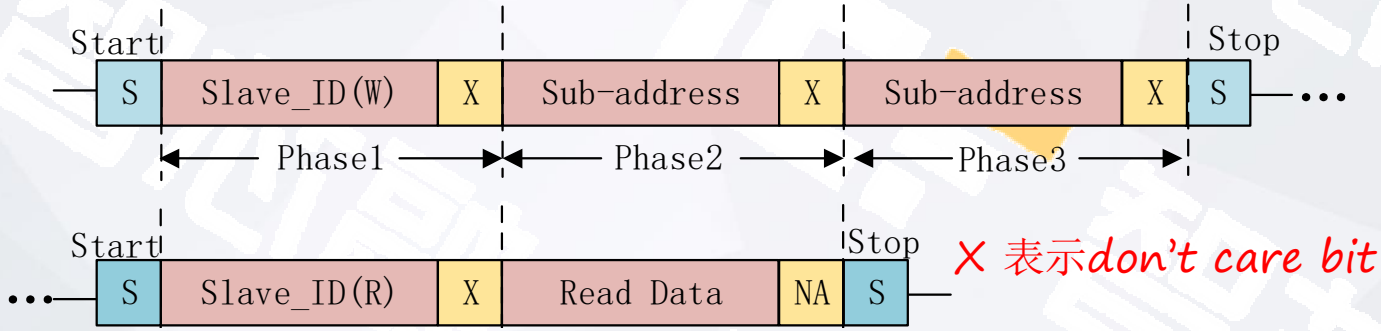


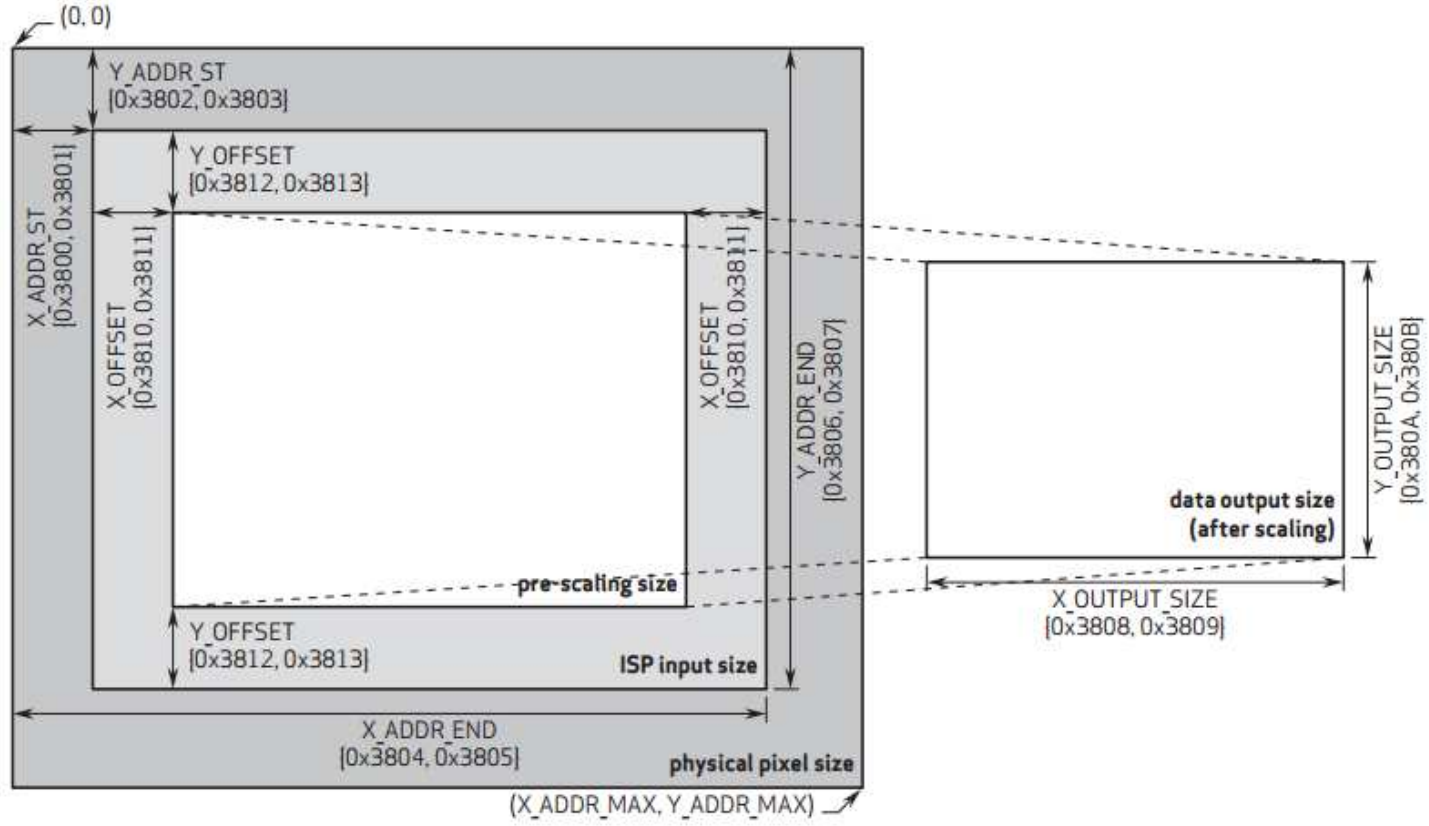
SCCB总线读写传输协议

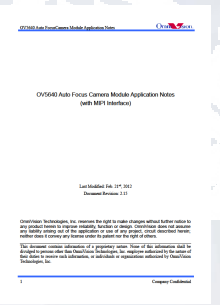
SCCB总线写数据传输协议（5640的寄存器地址为2个字节） X 表示don't care bit



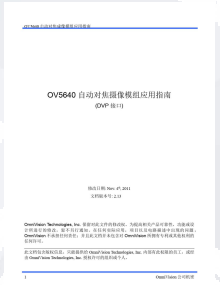
SCCB总线读数据传输协议



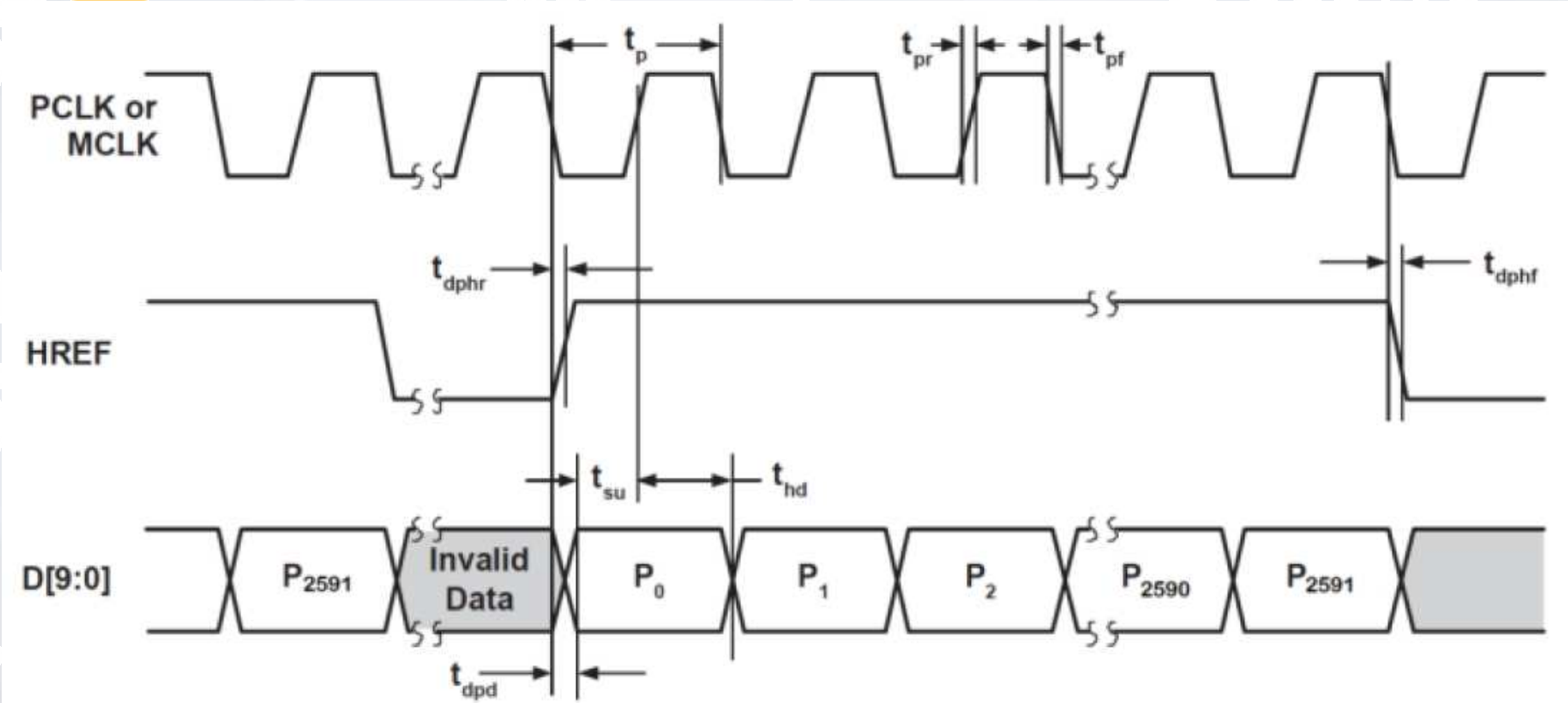

(0, 0)
Y_ADDR_ST
[0x3802, 0x3803]
Y_OFFSET
[0x3812, 0x3813]
X_ADDR_ST
[0x3800, 0x3801]
X_OFFSET
[0x3810, 0x3811]
X_OFFSET
[0x3810, 0x3811]
Y_ADDR_END
[0x3806, 0x3807]
pre-scaling size
Y_OFFSET
[0x3812, 0x3813]
ISP input size
X_ADDR_END
[0x3804, 0x3805]
physical pixel size
(X_ADDR_MAX, Y_ADDR_MAX)
data output size
(after scaling)
Y_OUTPUT_SIZE
[0x380A, 0x380B]
X_OUTPUT_SIZE
[0x3808, 0x3809]

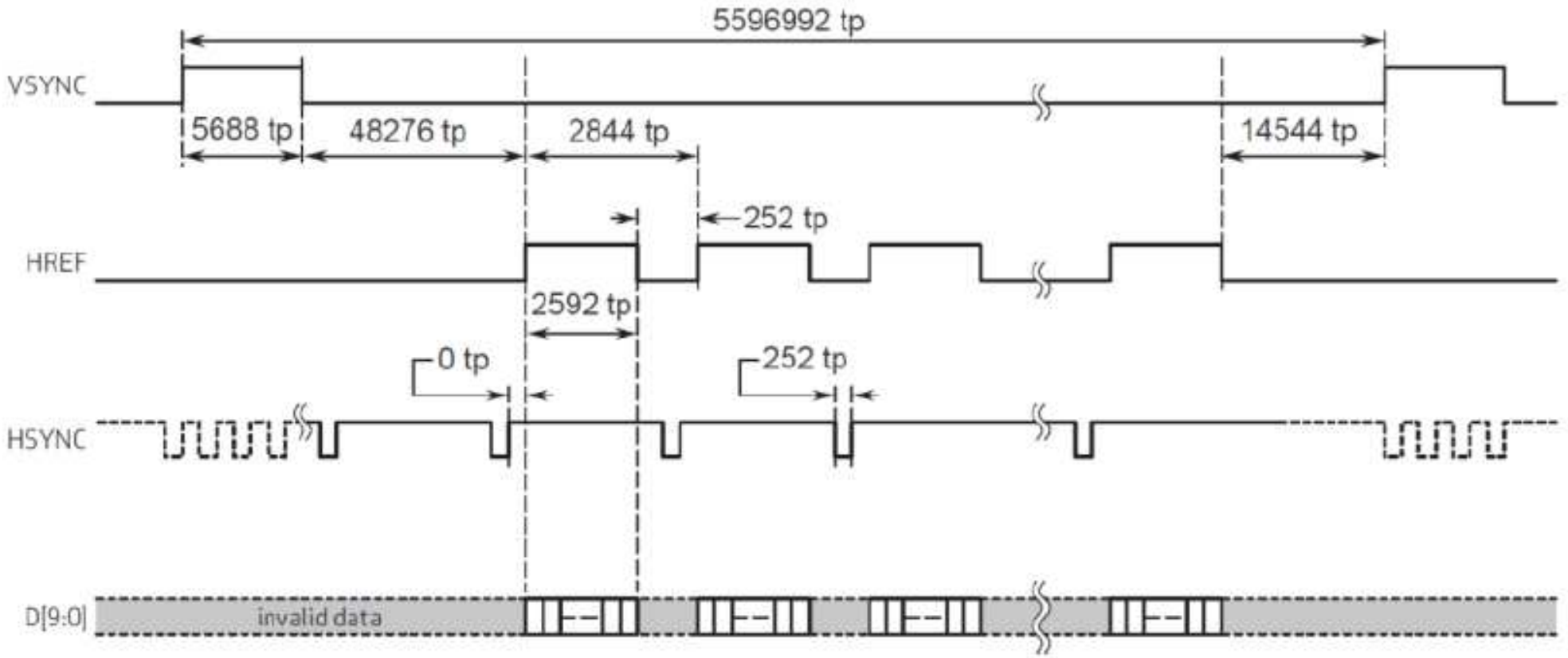

OV5640 Auto Focus Camera Module Application Notes
(with MIPI Interface)


datasheet
OV5640

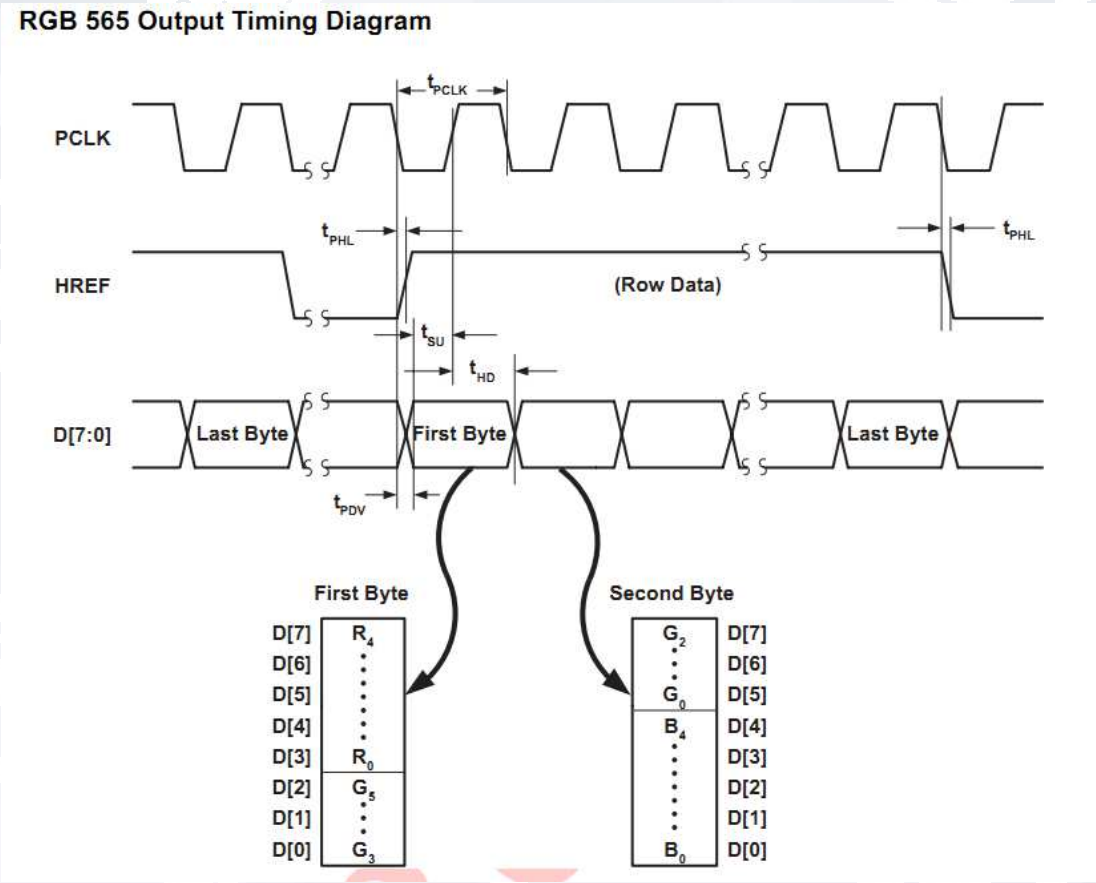

OV5640 自动对焦摄像模组应用指南

图像窗口设置


PCLK or MCLK
HREF
D[9:0]
t_p
t_{pr}
t_{pf}
t_{dphr}
t_{dphf}
t_{su}
t_{hd}
t_{dpd}
P_{2591}
Invalid Data
P_0
P_1
P_2
P_{2590}
P_{2591}



VGA帧模式输出时序图（QSXGA模式，分辨率2592*1944）



RGB565模式时序图


系统时钟域
uart调试接口
I2C总线
flash_switch
闪光灯按键
reg_conf_done
摄像头配置完成
reg_config_camera_ov5640_phy
摄像头控制模块
power_on_delay_ov5640_phy
电源控制模块
initial_en
camera_resetb
camera_pwup
camera 控制模块


系统时钟域
uart调试接口
I2C总线
flash_switch
闪光灯按键
reg_conf_done
摄像头配置完成
reg_config_camera_ov5640_phy
摄像头控制模块
power_on_delay_ov5640_phy
电源控制模块
initial_en
camera_resetb
camera_pwup
camera 控制模块

camera电源控制模块

在50m 系统时钟下，产生一个计数器，以后就可以根据这个计数器来实现 camer_pwup， cam_resetb，和initial_en 等信号的输出。

注： 目前并没有使用**cam_resetb**信号，这个信号是由硬件实现的。

```
// 定义一个多位计数器，通过计数器的自加1，建立一个时间轴，
// 在此时间轴上不同时间点，给出5640正确上电的驱动信号即可。

reg     [31:0]  cam_pwr_cnt             =       0   ;
always@ (   posedge clk_50m     )
if( rst_n == 1'b0 )
    cam_pwr_cnt <= 0;
else
begin
    if (cam_pwr_cnt < 200_000_000)
        cam_pwr_cnt <= cam_pwr_cnt + 1 ;
    else
        cam_pwr_cnt <= cam_pwr_cnt ;
end
```

camera_pwup 输出

当**rst_n == 0** 时，**camera_pwup** 清零，

当计数器 等于 **20_000_000**时，输出**camera_pwup**信号，这个信号从**fpga** 输出，连接到**camera** 硬件模块上，实现**camera** 硬件的上电

```
//  camera_pwup signal  generate

always  @   (   posedge clk_50m     )
if(rst_n == 1'b0)
    camera_pwup <= 0;
else
begin
    if (cam_pwr_cnt == 20_000_000)
        camera_pwup  <= 1;
    else
        camera_pwup  <= camera_pwup  ;
end
```

initial_en输出

当**rst_n == 0** 时，**initial_en**清零，

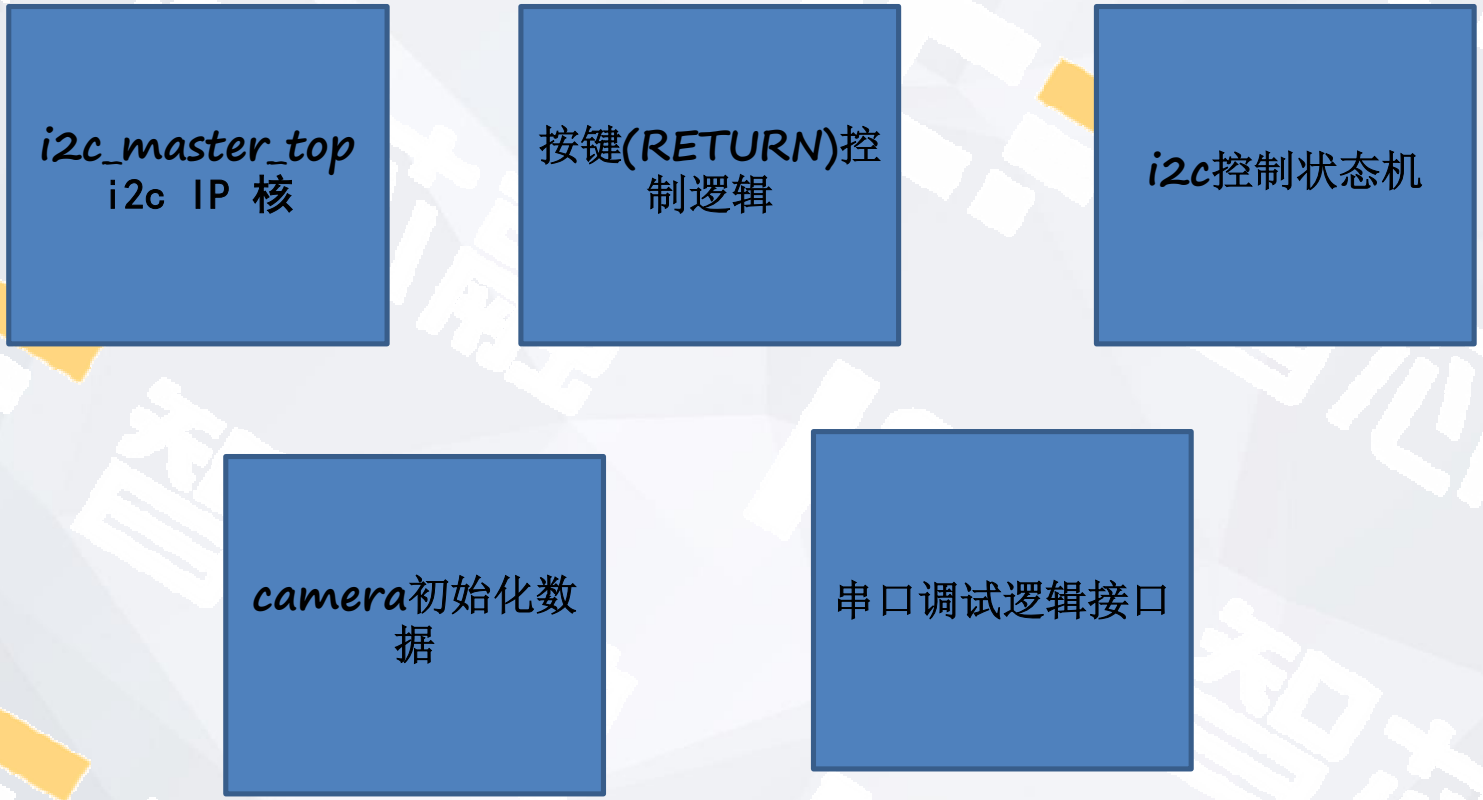
当计数器 等于 **100_000_000**时，输出**initial_en**信号，这个信号用于**fpga** 其他模块。 当**initial_en**为高时, 通知其他**fpga** 模块，**camera** 硬件已经稳定， 可以进行后续的初始化，和其他的正常操作了

```
//  initial_en signal  generate

always  @   (   posedge clk_50m     )
if(rst_n == 1'b0)
    initial_en <= 0;
else
begin
    if (cam_pwr_cnt == 100_000_000)
        initial_en <= 1;
    else
        initial_en <= initial_en;
end
```

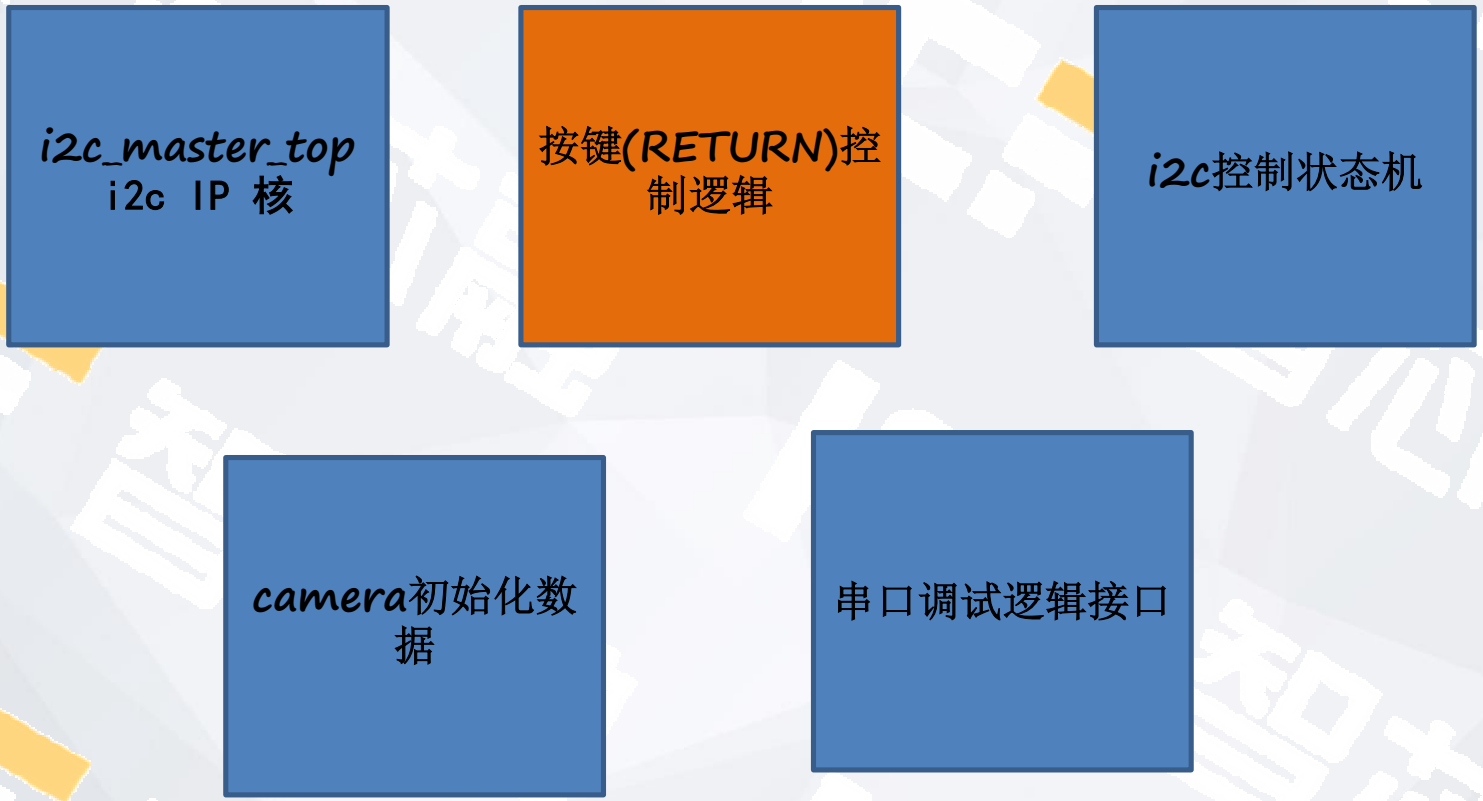

系统时钟域
uart调试接口
I2C总线
flash_switch
闪光灯按键
reg_conf_done
摄像头配置完成
reg_config_camera_ov5640_phy
摄像头控制模块
power_on_delay_ov5640_phy
电源控制模块
initial_en
camera_resetb
camera_pwup
camera 控制模块

系统时钟域



camera 控制模块

系统时钟域



camera 控制模块

按键控制flash led

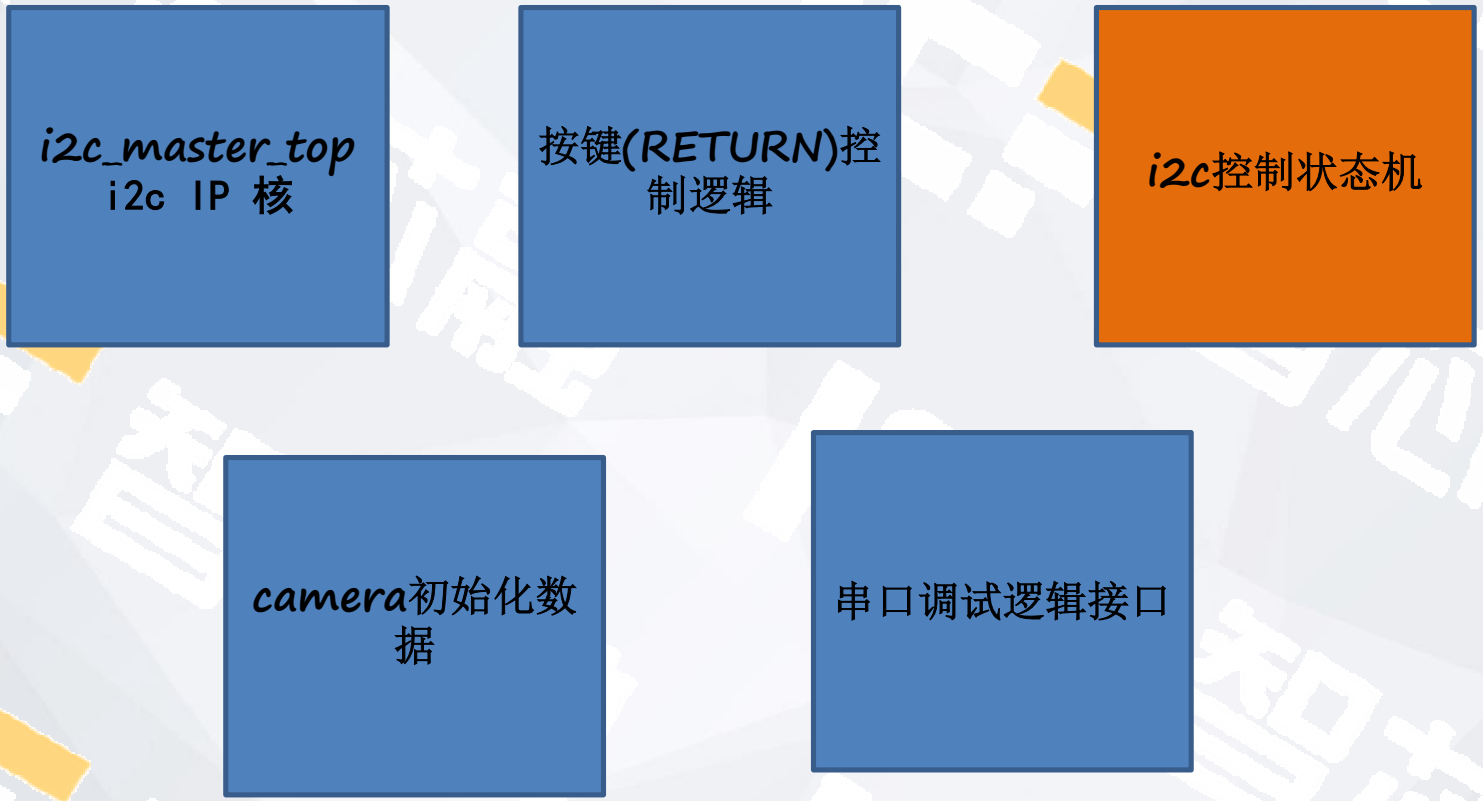
实现功能，每当用户按下return键，st_mux信号就反转一次，根据st_mux的值（0/1），相应的产生sign_Strobe 输出2'b01 或者2'b10。 这个限号用于i2c 控制逻辑读取相应的i2c 指令，写入到ov5640芯片中去，实现闪光灯 亮/灭。

```
assign reg_conf_done    =  reg_conf_done_reg  ;

/*
闪光灯开关信号st_Strobe的产生，按键每按一次（改变st_mux），
sign_Strobe的值在2'b01或者2'b10间进行一次赋值。（sign_Strobe为2'b01，
打开闪光灯。sign_Strobe为2'b10，则关闭闪光灯）
*/
reg [1:0] sign_Strobe  = 2'b00  ;
reg       st_mux       = 0      ;

always @ ( posedge clk_50m)
if (!initial_en)
begin
    sign_Strobe <= 2'b00 ;
    st_mux      <= 0     ;
end
else
begin
    if ( flash_switch  )
    begin
        sign_Strobe <= st_mux ? 2'b10 : 2'b01   ;
        st_mux      <= st_mux + 1               ;
    end
    else  sign_Strobe <= 2'b00   ;
end
```

系统时钟域



camera 控制模块

i2c 控制状态机

```
// 首先配置基本的253个寄存器，ov5640摄像头可以正常工作，采集图像。寄存器253,254的作用分别是打开闪光灯，关闭闪光灯。

reg     [1:0] i2c_st  = 0;
always@(posedge clk_50m )
begin
    if(!initial_en)      // ov5640摄像头上电未完成时，配置寄存器的电路模块处于复位状态
    begin
        i2c_st              <=  0       ;
        reg_wr_i2c          <=  0       ;
        reg_index           <=  0       ;
        reg_conf_done_reg   <=  0       ;
    end
```

当**initail_en** 为零，状态机回到初始状态，相关的寄存器清零， **reg_conf_done_reg** 输出为零，这个信号用于通知**fpga** 其他模块，告诉**camera** 芯片是否已经初始化完毕，**fpga** 其他相关模块， 得到初始化完毕的消息后， 可以进行其他操作了（比如数据写入**ddr**，就需要等待**camera** 稳定）

i2c 控制状态机

当**reg_conf_done_reg** 这个信号为低时（**camera** 还没有初始化完毕），进行**camera** 芯片的初始化操作.

这个逻辑核 **i2c master IP** 核通讯，告诉**i2c IP** ： **i2c** 地址， **i2c** 读/写，**i2c** 寄存器，**i2c** 数值等

当完成所有（**252**）条操作后，将**reg_conf_done_reg**拉高

```
if(reg_conf_done_reg == 1'b0)
begin
    if( reg_index <= 252  )
    begin
        // 调用IIC总线配置（0~252）寄存器的值，配置完成后，将reg_conf_done_reg值拉高
        case(i2c_st)
        0:
        begin
            i2c_data   <= {ov5640_devide_id, 1'b0, reg_data}  ;
            reg_wr_i2c <=  1                                  ;
            i2c_st     <=  1                                  ;
        end
        1:
        begin
            if(wr_done_i2c)
            begin
                reg_wr_i2c  <=  0   ;
                i2c_st <=  2   ;
            end
            else   ;
        end
        2:
        begin
            reg_index <= reg_index + 1'b1;
            i2c_st <= 0;
        end
        endcase
    end
    else
        reg_conf_done_reg   <=  1'b1;
```

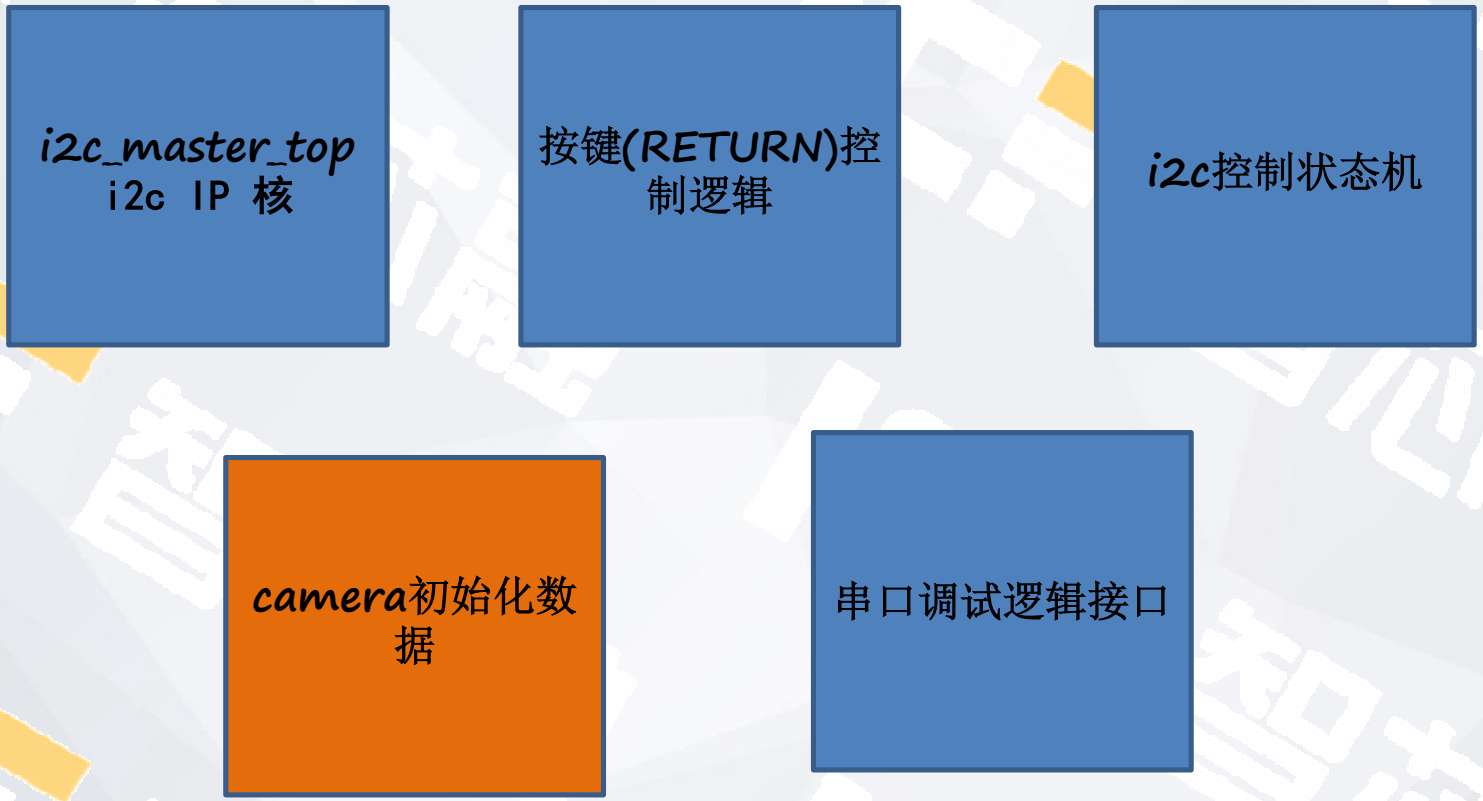
i2c 控制状态机

当**reg_conf_done_reg** 这个信号为高时（**camera** 初始化完毕），可以对**camera** 芯片做其他的操作了.

当按键（**return**）有操作时，就会把相应的操作，发送给**i2c master** 模块，实现人机交互

```
case(i2c_st)
0   :
begin
    if( sign_Strobe == 2'b01 )
    begin
        i2c_st     <= 1      ;
        reg_index  <= 253    ;
    end
    else
    begin
        if (sign_Strobe == 2'b10)
        begin             //
            i2c_st     <=  1    ;
            reg_index  <=  254  ;
        end
        else   ;
    end
end
1   :
begin
    i2c_data    <= {ov5640_devide_id, 1'b0, reg_data }  ;       /
    reg_wr_i2c  <= 1                    ;
    i2c_st      <= 2                    ;
end
2   :
begin
    if(wr_done_i2c)
    begin                           //IIC发送结束
        reg_wr_i2c <= 0;
        i2c_st <= 3;
    end
    else  ;
end
3   :
begin
    i2c_st <= 0;
    reg_index <= 253;
end
endcase
```

系统时钟域



camera 控制模块

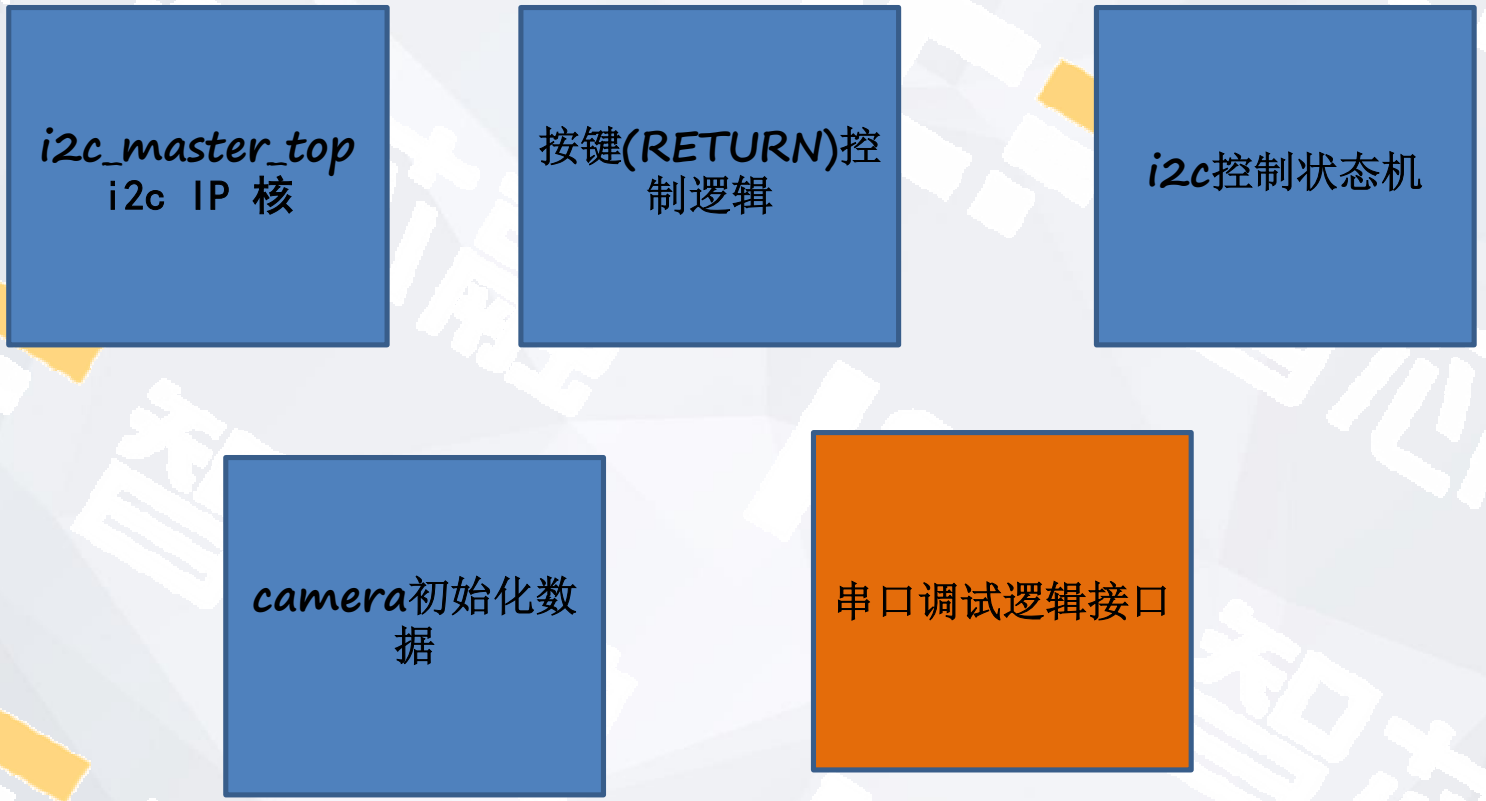
camera 初始化数据

camera 初始化数据，用于初始化**camera**芯片，完成相关的配置工作，包括**camera** 芯片的时钟设置，分辨率的设置，图像采样设置，输出模式设置，图像位置设置，图像颜色调整等， 很多操作。

这里我们选择的时 **640x480@30Hz** ，**RGB565, DVP** 模式，将图像数据输出给**fpga**。

很多具体的操作，需要仔细阅读**ov5640**的相关寄存器手册。

系统时钟域



camera 控制模块

uart调试接口

当reg_conf_done_reg 为零时：

uart调试状态机 将相关的数据清零。

uart调试不是camera项目必须的， 但却是强烈推荐的。调试工具可以用来验证i2c 的正确性。同时以可以用它来快速调试相关的I2C 寄存器，这种调试方法是非常高效的。

```
//************** 串口在线配置ov5640寄存器

reg [2:0] uart_st = 0;
always @ (posedge clk_50m)
if(reg_conf_done_reg == 0)
begin
    uart_wr_i2c <= 0    ;
    uart_rd_i2c <= 0    ;
    uart_st     <= 0    ;
    uart_sel    <= 0    ;
end
```

uart调试接口

状态0： 等待uart 是否有数据被接收到， 同时查看一下接收到的第一个数据是否为01, 如果数据为01，表示uart 准备调试camera； 如果为02，表示uart 准备调试adv7511.

状态1： 等待uart 接收第二个数据，这个数据为I2C的地址，并且从这个数据的最低位还能判断出对I2C的读写操作。同时uart_sel 切换I2C master IP 到uart上。

状态2：接收数据为寄存器的高8位数据

状态3：接收数据为寄存器的低8位数据，同时判断uart的操作： I2C 读， I2C写，根据不同的操作，跳转到不同的状态机。

状态4：写操作，将下一个uart 接收到的数据写入I2C (给寄存器赋值）

状态5：写操作完成， 同时将uart_sel 切换到数据初始化状态机上。

状态6：对I2C master IP 进行读操作

状态7：将I2C 读取到的数据，发送到uart 上。

uart调试

uart 的调试方法在“图像采集 uart 在线调试使用文档说明”中有详细描述。

使用方法：使用计算机uart工具，举例sscom。

举例：

01 78 4741 07

01 78 4741 00

01 78 503D 80

01 78 503D 00



uart调试

调试读命令： id（1byte） + i2c 地址（1byte） + 寄存器（2bytes）

调试写命令： id（1byte） + i2c 地址（1byte） + 寄存器（2bytes）+ 寄存器值（1byte）

id ： 8bit ， 01 = ov5640； 02 = adv7511. 用来指示当前的调试命令和哪个模块通讯（ov5640，adv7511）

i2c地址： 8bit， 在读命令中 这个地址的最低位是‘1’，在写命令中这个地址的最低为是‘0’。

寄存器：ov5640 的寄存器是16bit的， 对于adv7511来说， 寄存器是由8bit，所以，在读写adv7511时：都是00xx，将高位地址 设置位‘00’。

寄存器值：8bit， 写命令才有。 写入寄存器的值。

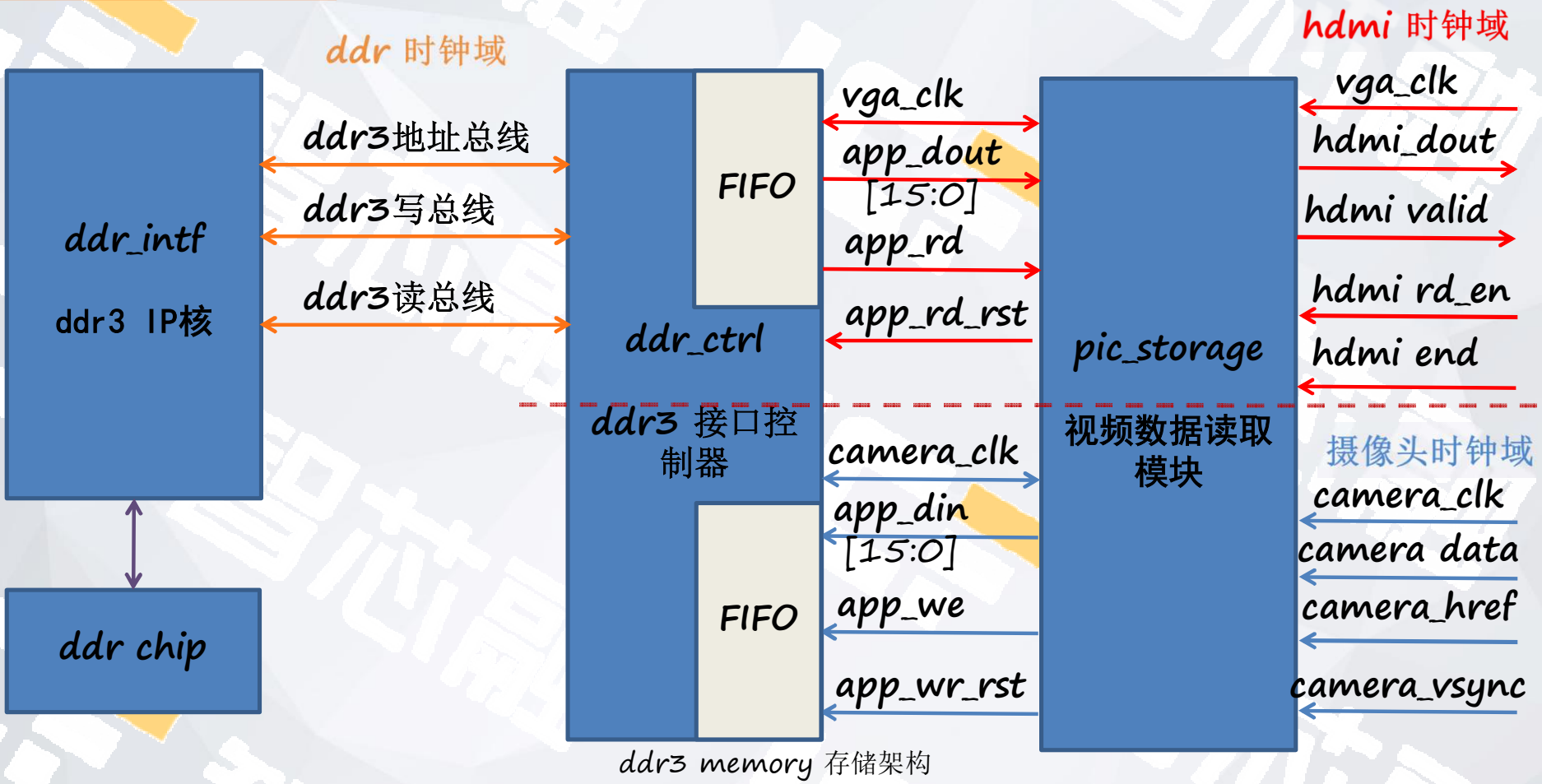
1 整体图像采集工程设计

2 图像采集模块OV5640简介配置

3 采集图像数据存储

4 HDMI接口的配置与现实

5 习题



ddr3 memory 存储架构

图像采集系统数据存储架构

ddr_intf: ddr IP 核 例化接口，包括地址总线， 写数据总线， 读数据总线

ddr_ctrl：

读取fifo数据（读fifo），将这些数据写入到指定的ddr 存储空间上，

根据用户指定的ddr 地址， 将ddr存储空间里的数据读出， 写入到fifo 中去 （写fifo）

pic_storage：

接收camera 来的数据，将这些数据写入到fifo中去（ddr 读fifo）；根据camera 的href信号，确定当前数据时哪一行（写入ddr 存储空间）；根据camera 的vsync 信号， 确定那一场的数据（ddr page 空间）

根据hdmi_end信号， 确定要读取那一场（page）； 根据hdmi_rd_en信号，确定从ddr存储空间中读取哪一行的数据

```
input               ddr_ready           ,
input               hdmi_reg_done       ,   // HDMI 寄存器配置完成标识信号
input               ov5640_reg_done     ,   // OV5640摄像头寄存器配置完成标识信号
input               camera_switch       ,   // 拍照按键经过按键消抖之后生成的拍照命令电平信号，低有效
```

ddr 校准， adv7511 初始化， ov5640初始化。 拍照按键。

```
// camera data input channel
    input               camera_clk          ,   // 图像数据伴随时钟
    input               camera_href         ,   // 图像数据行信号，高（此时传输的是有效图像数据）有效
    input               camera_vsync        ,   // 图像数据场信号
    input       [7:0]   camera_data         ,   // 图像数据
```

从camera 模块传来的信号接口

```
output reg [15:0] app_din = 0           ,   // 应用程序 写数据到 ddr
output reg        app_we = 0            ,   // 用用程序 写数据 enable
input             ddr_wr_ack            ,   // ddr 通知 应用程序 fifo 里的数据已经写完了
output reg [15:0] app_wr_line_num = 0   ,   // 通知ddr ，将fifo里的数据写到 ddr 的什么位置上（哪一行）
output            app_wr_start          ,   // 配合 app_wr_line_num 一起使用
output reg        app_wr_rst = 0        ,   // 应用程序写fifo 的 reset 信号
```

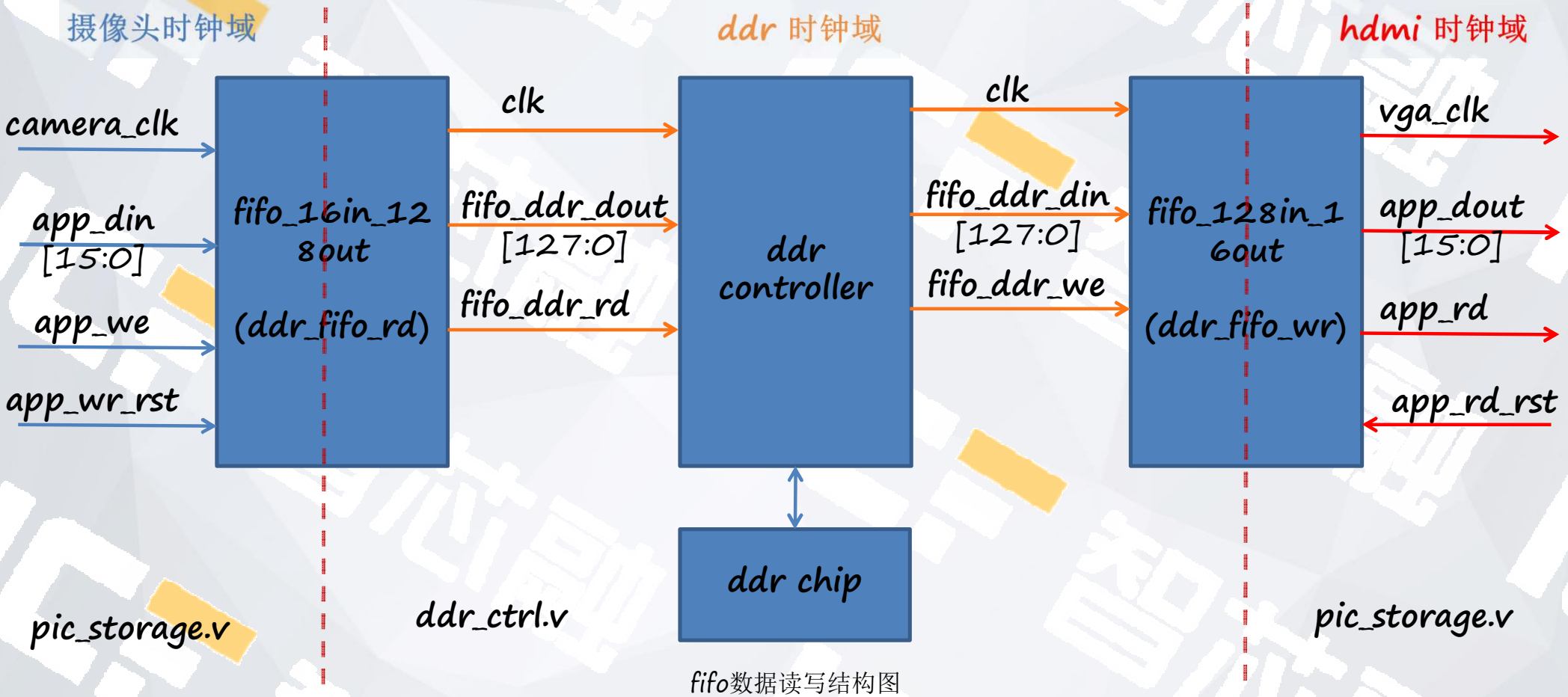
camera模块的信号，经过状态机逻辑， 写入的fifo 的接口

```
input  [15:0]      app_dout                ,  // fifo 读出 ddr 的内容,
output reg         app_rd = 0              ,  // FIFO 读信号 ,  读ddr 里的内容
input              ddr_rd_ack              ,  // 当 ddr 将数据写入到fifo后,  通知用用程序端,  fifo 准备好
output reg [15:0]  app_rd_line_num = 0     ,  // 应用程序通知ddr,  读取ddr里哪一行的内容
output             app_rd_start            ,  // 配合 app_rd_line_num ,  通知ddr,  读取那一行的内容
output reg         app_rd_rst = 0          ,  // 应用程序读fifo reset 信号
```

从**ddr**读出数据，这些数据最好会传送到**hdmi** 接口上

```
// hdmi data output channel
    input              vga_clk             ,   // HDMI显示图像的时钟
    output             hdmi_valid          ,   // HDMI模块从FIFO缓冲读出的图像数据有效标志
    output [15:0]      fifo_hdmi_dout      ,   // HDMI模块从FIFO缓冲读出的图像数据
    input              hdmi_rd_en          ,   // HDMI模块从FIFO缓冲取图像数据的读使能
(* mark_debug = "true" *)    input          hdmi_end            ,   // HDMI图像显示一场有效图像显示完毕的标识信号
(* mark_debug = "true" *)    input          hdmi_req            ,   // HDMI模块从SRAM预读图像的请求信号
```

hdmi 数据接口，这个接口连接**adv7511** 芯片，将图像数据传送到显示器上



fifo数据读写结构图

图像采集系统数据存储设计

区分时钟域：

camera 时钟域（从camera 硬件模块而来，包括时钟， 数据，数据enable，v信号等）

ddr 时钟域 （控制ddr 芯片，包括地址总线，写数据总线，读数据总线等）

hdmi 时钟域 （fpga 发送数据到adv7511， 包括 数据，h，v，de，clk 等）

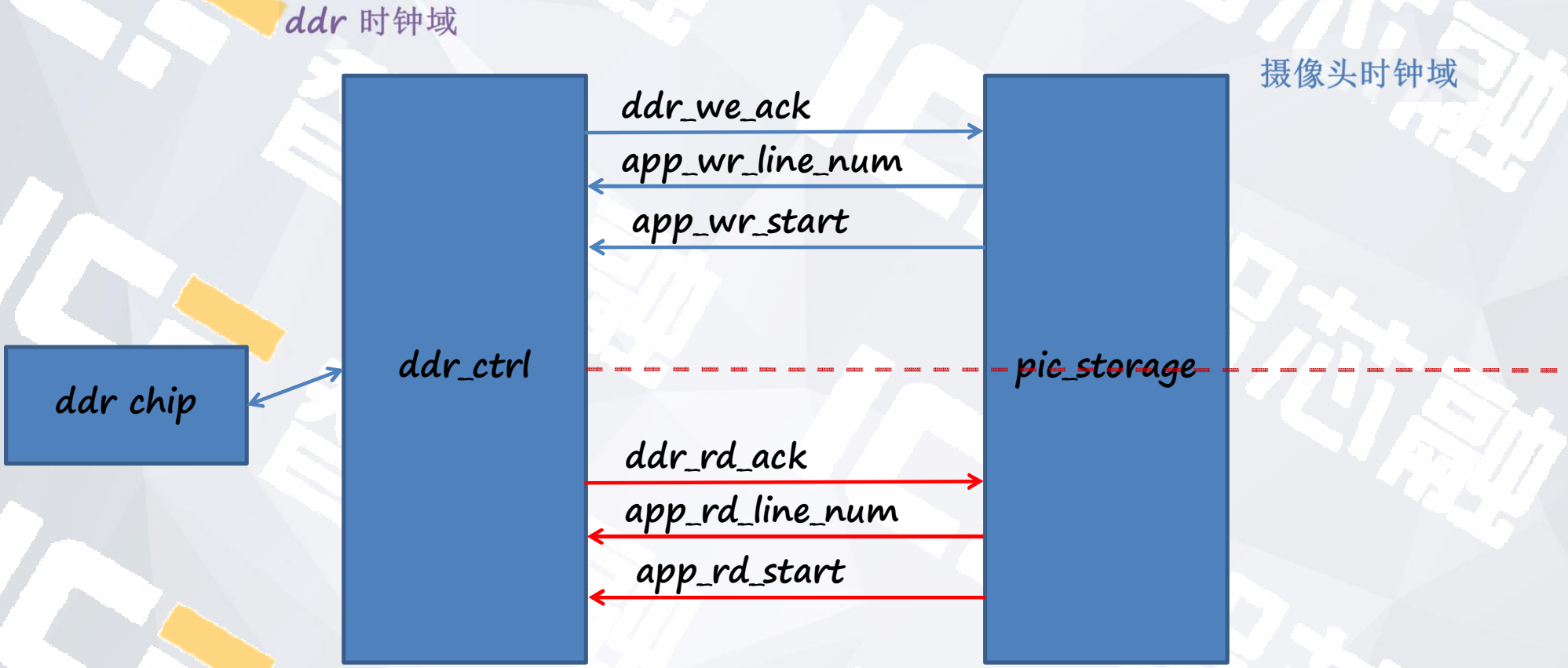
camera 数据输入fifo：

16bit in， 128bit out fifo， 但camera 来的数据是8bit,需要每两个数据写入fifo一次。

ddr时钟域端读取fifo数据，存储到ddr 芯片中去

hdmi 数据输出fifo：

16bit out，128bit in fifo， ddr 写入数据到这个fifo， hdmi 时钟域端读取数据，并且发送给adv7511 芯片



fifo数据读写结构图

ddr 地址选择接口（异步信号同步方案）

摄像头时钟域：

发送app_wr_start 信号 和 app_wr_line_num 通知ddr模块 将camera的数据写入到ddr存储空间中的什么位置（ddr 地址空间）；由于时异步时钟域，所以app_wr_start, app_wr_line_num 都是持续多个camera_clk的时钟周期，用于ddr时钟域采集这个信号

当ddr 将fifo 里的数据写入到app_wr_line_num 指定的ddr 地址空间后，发送ddr_we_ack，通知应用程序，ddr 操作完成。 由于是异步时钟域，ddr_we_ack 也是多个时钟周期（ddr_clk)的信号，用于摄像头时钟域来采集。

ddr 地址选择接口（异步信号同步方案）

hmdi 时钟域：

发送app_rd_start 信号 和 app_rd_line_num 通知ddr模块， 通知ddr，从app_rd_line_num 指定的ddr地址空间中，将数据读出，写入到fifo中去；由于时异步时钟域，所以app_rd_start, app_rd_line_num 都是持续多个camera_clk的时钟周期，用于ddr时钟域采集这个信号

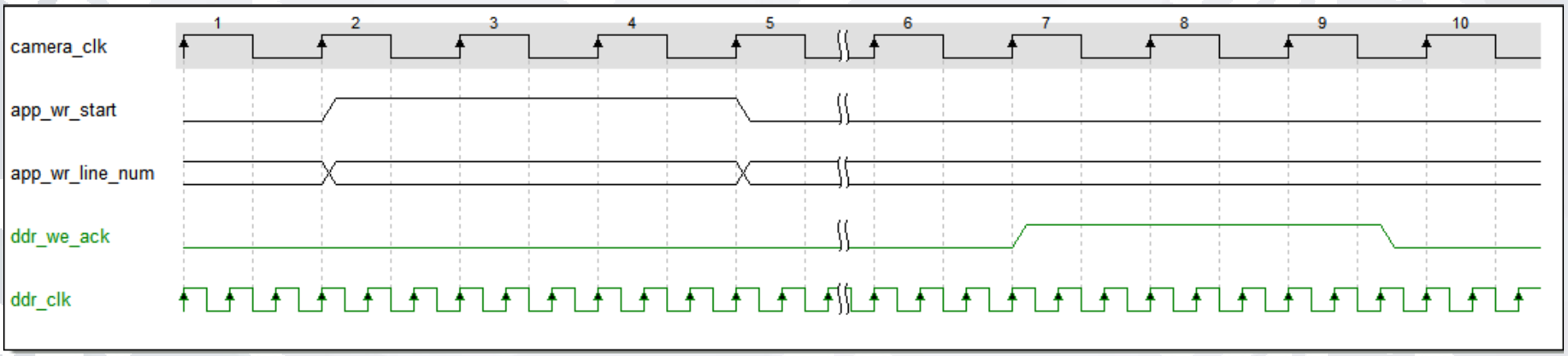
当ddr 将数据写入fifo后，发送ddr_rd_ack，通知应用程序，ddr 操作完成。应用程序这时就可以读取fifo 里的数据了。 由于是异步时钟域，ddr_we_ack 也是多个时钟周期（ddr_clk)的信号，用于摄像头时钟域来采集。

ddr 地址选择接口（异步信号同步方案）

camera_clk时钟和ddr_clk时钟分数不同的时钟域

app_wr_start, app_wr_line_num 被ddr_clk 时钟域采样

ddr_we_ack 由ddr_clk 时钟域产生， 可以被camera_clk 时钟域采样

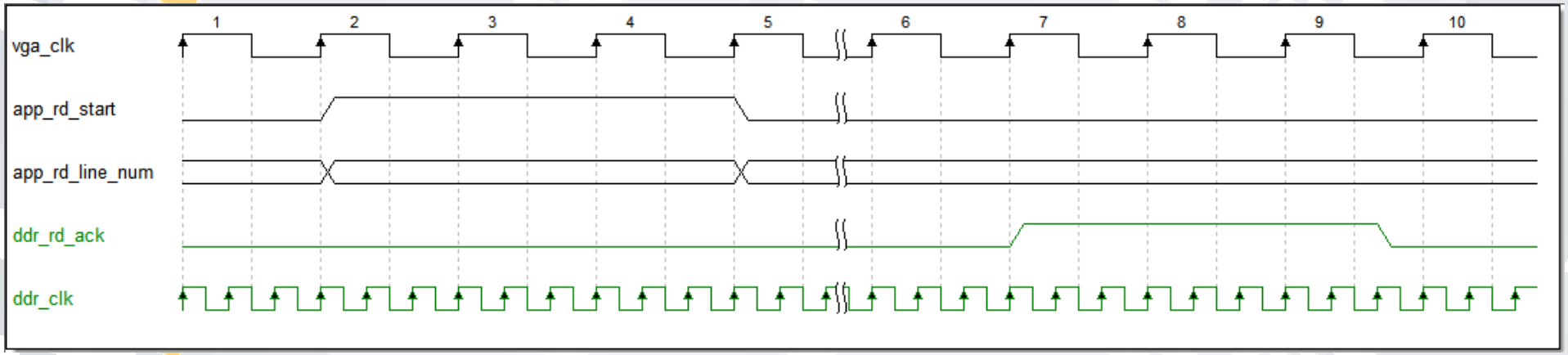


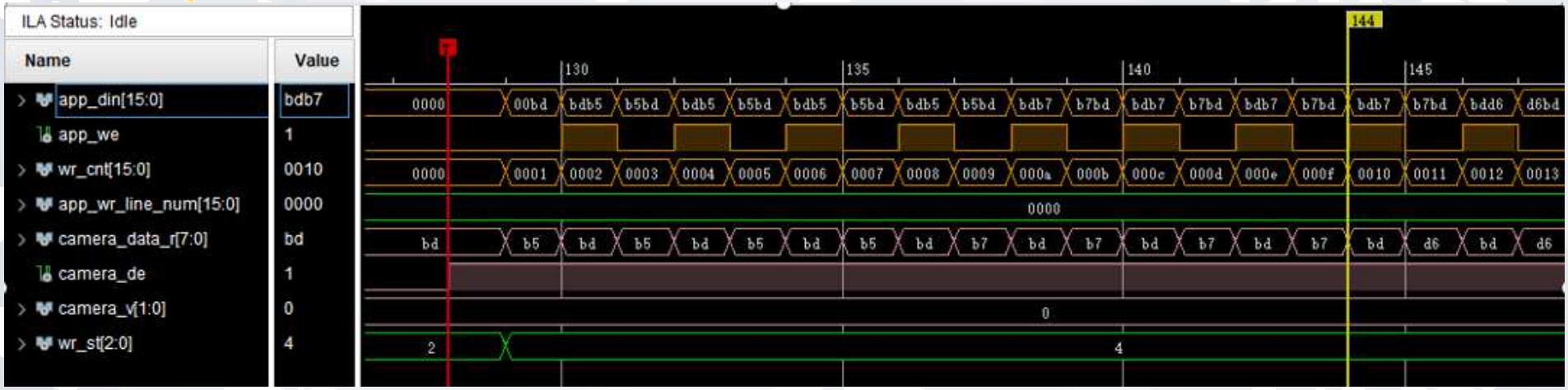
ddr 地址选择接口（异步信号同步方案）

vga_clk时钟和ddr_clk时钟分数不同的时钟域

app_rd_start, app_rd_line_num 被ddr_clk 时钟域采样

ddr_rd_ack 由ddr_clk 时钟域产生， 可以被vga_clk 时钟域采样





camera 模块数据采集

数据接口camera_data 是8bit， 而fifo 确实16bit的， 所以，每两个数据，写入fifo一次。

由于fifo在例化是，没有办法选择 8bin 128out； 所以，我们使用的fifo是 16in 128out

位拼接存储数据示意图

TEST_PIXEL 参数

在正常操作时，TEST_PIXEL = 0， 在测试模式时，利用这个参数，可以测试fifo，ddr是否正确。这在项目开发中是十分必要的。

```
localparam TEST_PIXEL = 0;  // 测试pixel ，正常情况下 TEST_PIXEL = 0
```

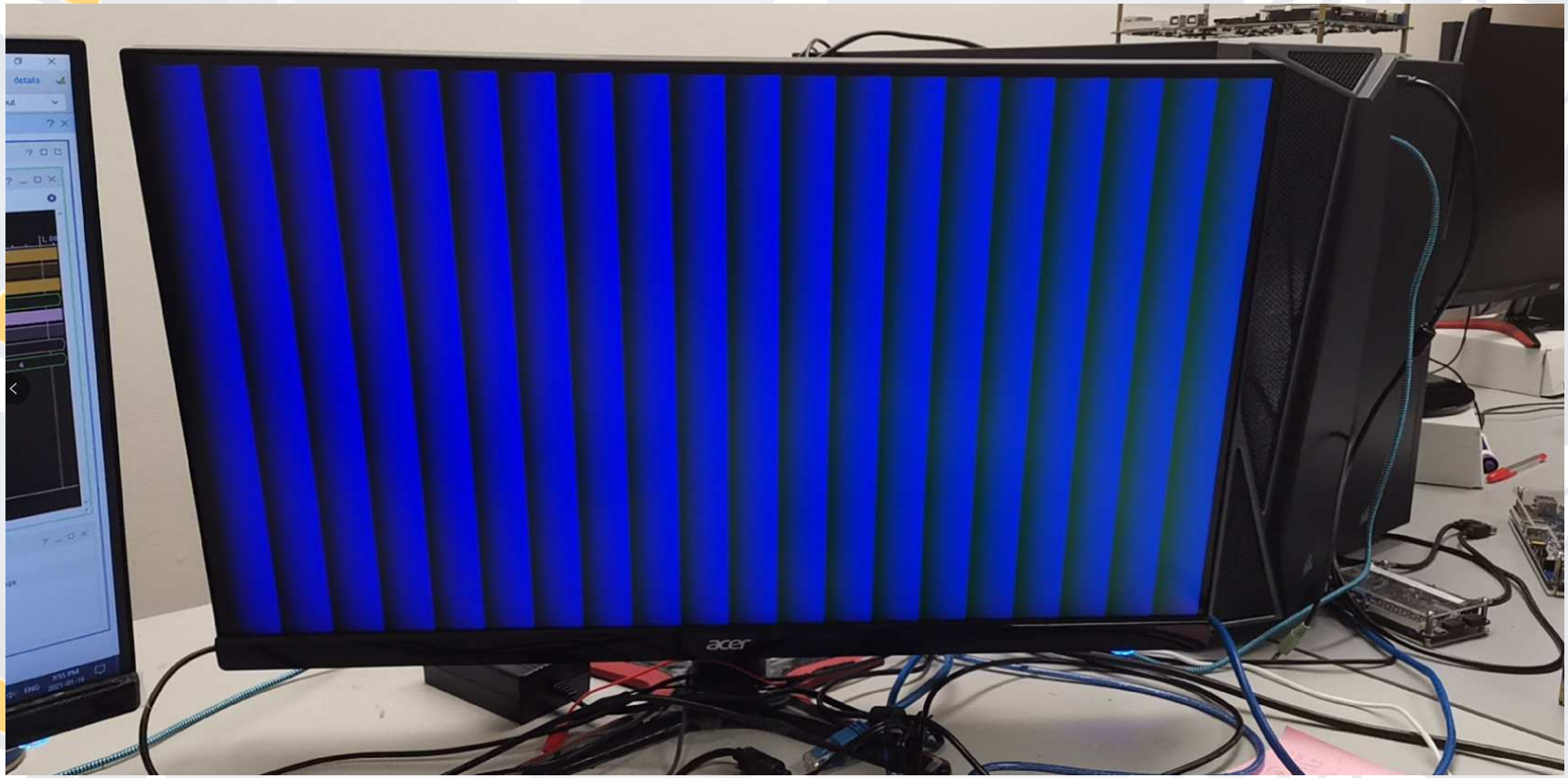
```
2:
begin
    app_we = 0;

    if(camera_v == 2'b01)  // 等待 camera 输出 v 信号，同时提取 v 信号的上升
    begin
        if(snapshot_r)     // 如果这个时候，是拍照模式，进入 拍照 ，当前图像
            wr_st <= 3;
        else  // 如果不是拍照模式，将行地址 清零，同时 更换 一页存储空间，
        begin
            app_wr_line_num[10:0] <= 0;  // 行地址
            app_wr_line_num[12:11] <= app_wr_line_num[12:11] + 1;   // 页 地址
        end
    end

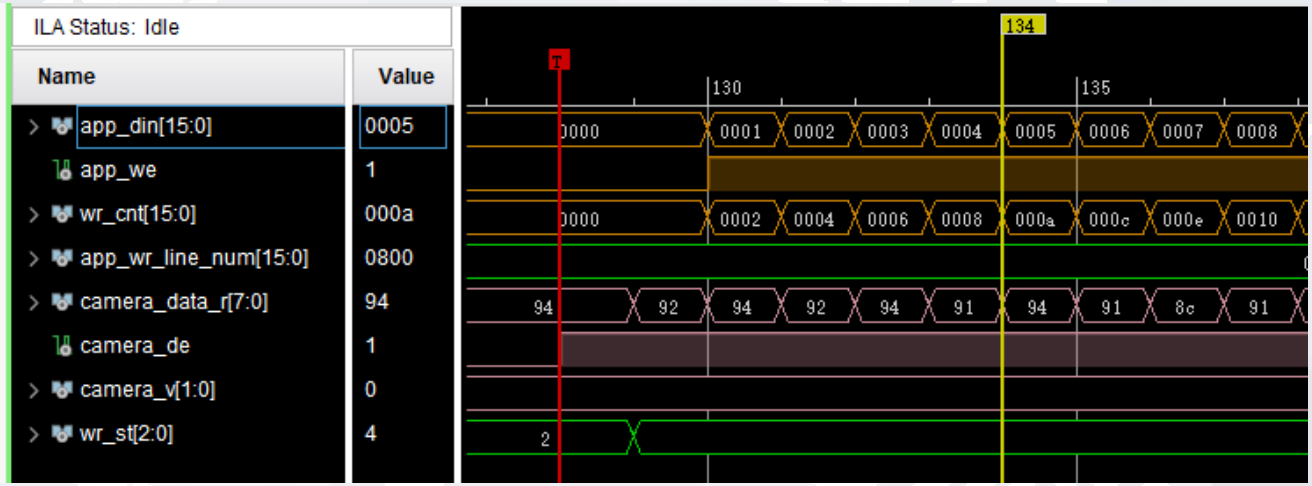
    if(camera_de)  // 如果 camera 数据enable 信号出现，开始存储数据到fifo 中
    begin
        app_din <= TEST_PIXEL ? 0 : {app_din[7:0] , camera_data_r};
        wr_cnt  <= TEST_PIXEL ? 0 : wr_cnt + 1;
        wr_st   <= 4;
    end
end
```

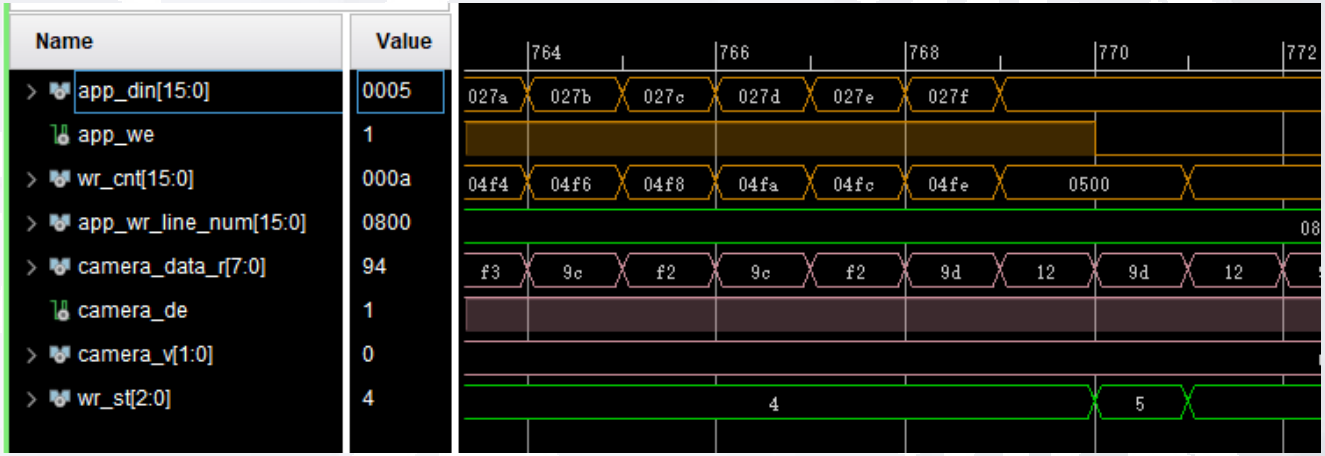
利用TEST_PIXEL这个参数，还可以实现其他的视频操作，比如测试图片等。

```
4:
begin
    if(wr_cnt == BYTE_WR_NUM) // 当数据 等于 1280 = 640pixel *2 bytes ( 一行数据
    begin
        app_we = 0;
        wr_st <= 5;
    end
    else
    begin
        if(TEST_PIXEL == 1)
        begin
            app_din <= app_din + 1;
            wr_cnt <= wr_cnt + 2;
            app_we = 1;
        end
        else
        begin
            app_din <= {app_din[7:0] , camera_data_r};
            wr_cnt <= wr_cnt + 1;
            if(wr_cnt[0])          app_we = 1;  // 每两个 bytes 写入fifo 一次
            else app_we <= 0;
        end
    end
end
```

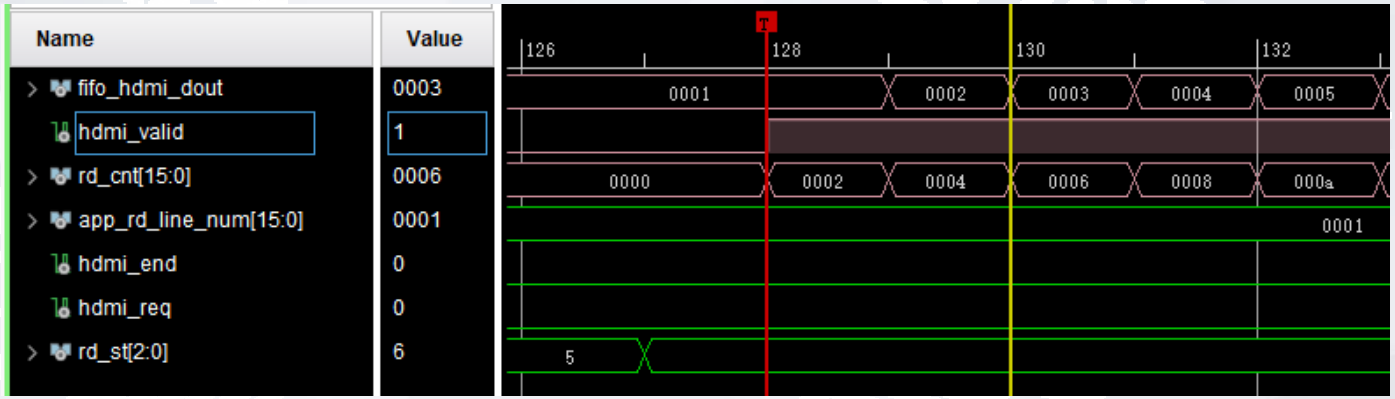

acer

在测试模式下，模拟数据输入，检查数据写入ddr的过程

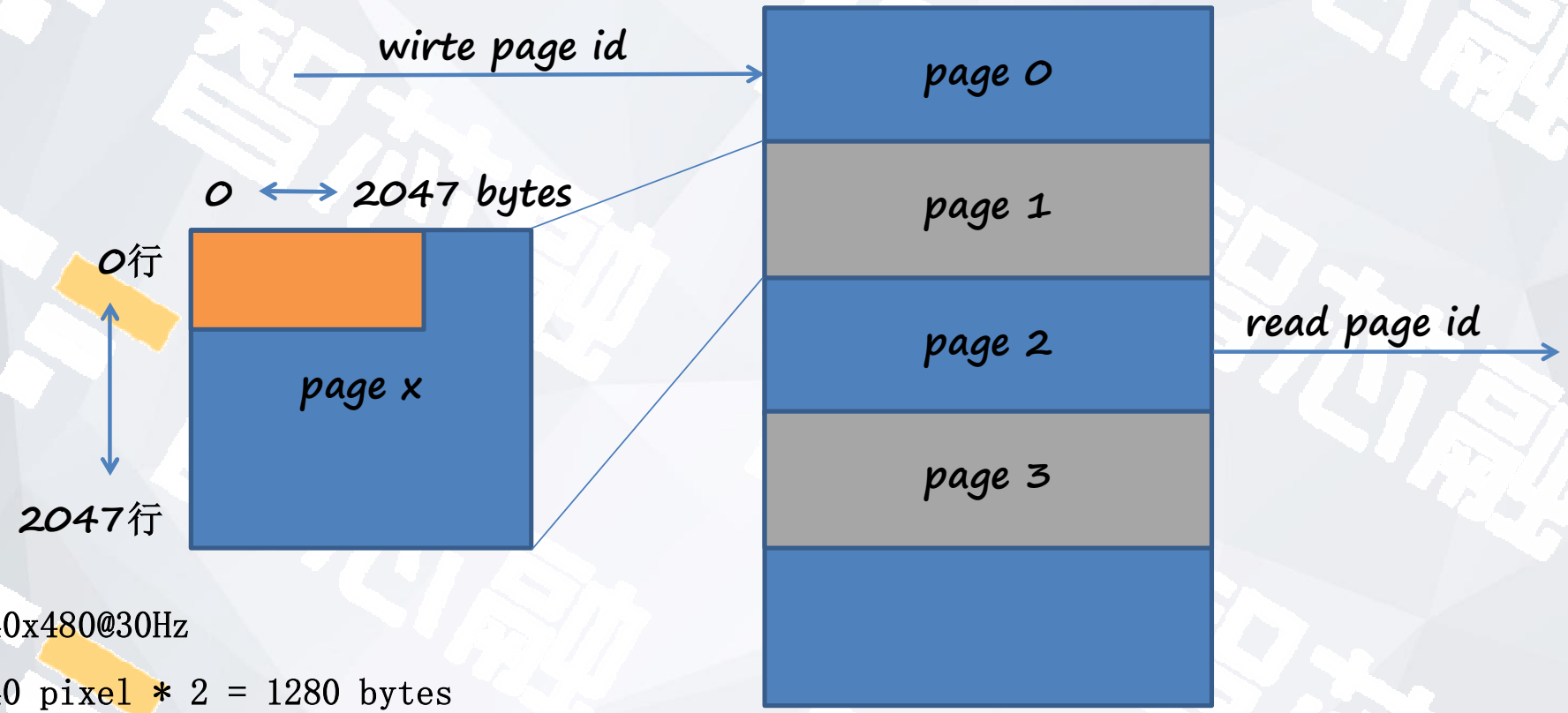




在测试模式，检查fifo读出过程，同样也可以验证ddr 的读取过程。







640x480@30Hz

640 pixel * 2 = 1280 bytes

480 行

ddr 芯片中的存储空间

page0 地址： 0x00_0000 - 0x3f_ffff

page1 地址： 0x40_0000 - 0x7f_ffff

page2 地址： 0x80_0000 - 0xbf_ffff

page3 地址： 0xc0_0000 - 0xff_ffff

在每一页地址中：

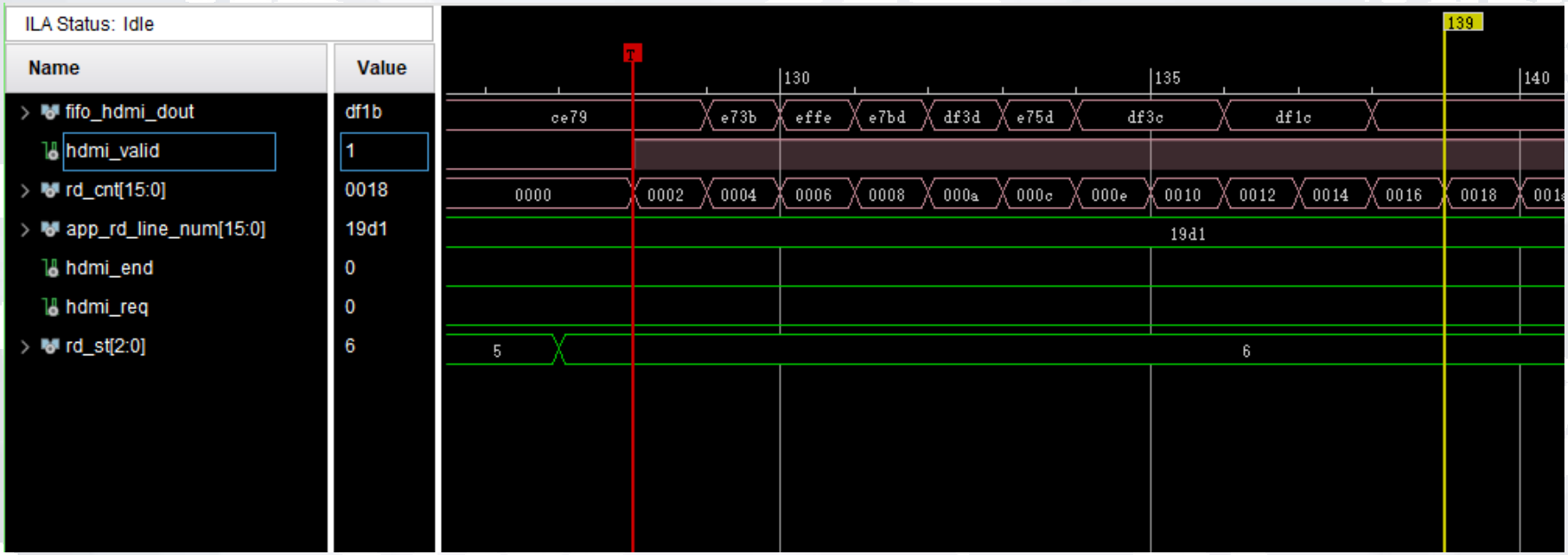
第1行地址起始地址：0x0000

第2行地址起始地址：0x0800 = 上一行起始地址 + 2048 bytes

第3行地址起始地址：0x1000 = 上一行起始地址 + 2048 bytes

第4行地址起始地址：0x1800 = 上一行起始地址 + 2048 bytes

.....


ILA Status: Idle
Name
Value
fifo_hdmi_dout
df1b
hdmi_valid
1
rd_cnt[15:0]
0018
app_rd_line_num[15:0]
19d1
hdmi_end
0
hdmi_req
0
rd_st[2:0]
6
130
135
140
139
ce79
e73b
effe
e7bd
df3d
e75d
df3c
df1c
0000
0002
0004
0006
0008
000a
000c
000e
0010
0012
0014
0016
0018
19d1
5
6

位拼接存储数据示意图

1 整体图像采集工程设计

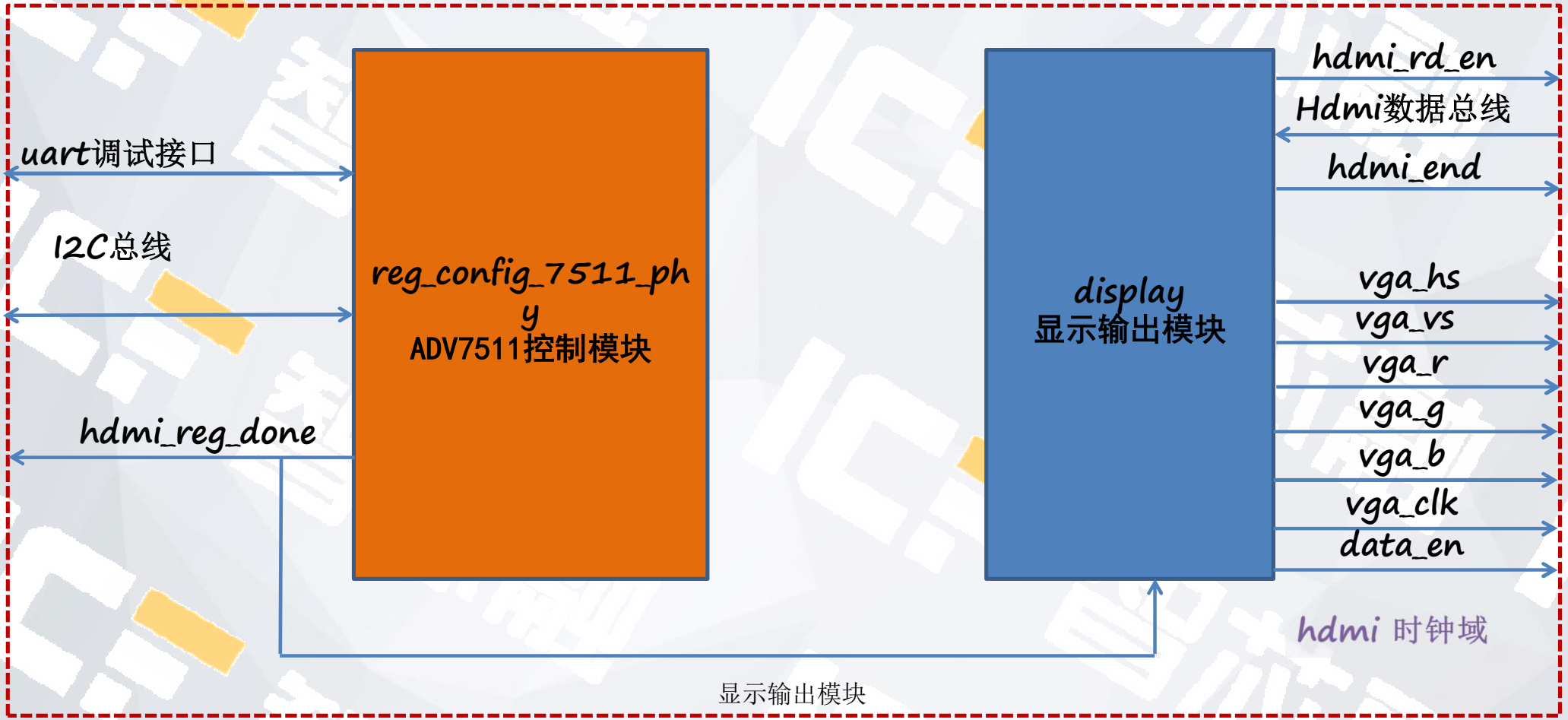
2 图像采集模块OV5640简介配置

3 采集图像数据存储

4 HDMI接口的配置

5 习题


uart调试接口
I2C总线
hdmi_reg_done
reg_config_7511_phy
ADV7511控制模块
display
显示输出模块
hdmi_rd_en
Hdmi数据总线
hdmi_end
vga_hs
vga_vs
vga_r
vga_g
vga_b
vga_clk
data_en
hdmi 时钟域
显示输出模块


uart调试接口
I2C总线
hdmi_reg_done
reg_config_7511_phy
ADV7511控制模块
display
显示输出模块
hdmi_rd_en
Hdmi数据总线
hdmi_end
vga_hs
vga_vs
vga_r
vga_g
vga_b
vga_clk
data_en
hdmi 时钟域
显示输出模块

系统时钟域

i2c_master_top
i2c IP 核

i2c控制状态机

adv7511初始化数据

串口调试逻辑接口

hdmi 控制模块，reg_config_adv_7511_phy

系统时钟域



hdmi 控制模块，reg_config_adv_7511_phy

初始化状态机

reset 信号需要同步，这是一个很好的习惯， 虽然不同步，也可以使用，但会给整个系统带来不必要的时序问题。如果是其他信号，而不是reset 信号，还有可能产生一些错误。

在一段式状态机中， 是可以检测到这些错误的。（警告）

```
assign hdmi_reg_done    =  reg_conf_done_reg  ;
//**********5640 configuration register ******************
reg      [1:0] i2c_st = 0;
always@(posedge clk_50m )
if(~rst_n_50m)
begin
    i2c_st              <=  0   ;
    reg_wr_i2c          <=  0   ;
    reg_index           <=  0   ;
    reg_conf_done_reg   <=  0   ;
    cnt_time_delay      <=  0   ;
end
```

在状态机初始化时： reg_conf_done_reg信号为零，表示adv7511没有初始化，i2c_st 状态机，接下来就要对adv7511进行相关的初始化操作。

状态0：读取adv7511寄存器列表中的数据，送给i2c_master 模块。

状态1：等待i2c_master操作完成

状态2：读取adv7511寄存器的列表加一

状态3：等待一段时间， 这个不是i2c_master所必须的， 但是adv7511芯片所需要的，（并不是每个寄存器都需要，但每个寄存器加上delay也是没有问题的）

当19个寄存器全部设置完毕，**reg_conf_done_reg = 1**

```
if(reg_conf_done_reg == 1'b0)
begin
    if( reg_index <= 19  )
    begin
        case(i2c_st)
        0:
        begin
            cnt_time_delay  <=  0       ;
            i2c_data        <= {adv_7511_id, 1'b0, reg_data}  ;
            reg_wr_i2c      <=  1       ;
            i2c_st          <=  1       ;
        end
        1:
        begin
            if(wr_done_i2c)
            begin
                reg_wr_i2c  <=  0   ;
                i2c_st <=  2        ;
            end
            else                    ;
        end
        2:
        begin
            reg_index <= reg_index + 1'b1;
            i2c_st <= 3;
        end
        3:
        begin
            reg_index <= reg_index  ;
            cnt_time_delay <= cnt_time_delay + 1  ;
            if   (  cnt_time_delay >= 20'd 250000)
                i2c_st    <=  0   ;
            else i2c_st    <=  3   ;
        end
        endcase
    end
    else
        reg_conf_done_reg   <=  1'b1;
end
```

系统时钟域



hdmi 控制模块，reg_config_adv_7511_phy

adv7511寄存器列表：

为保持和camera使用相同的i2c_master IP，寄存器使用16bit，实际上adv7511芯片只需要8bit 寄存器地址就可以了。

当前的数据结构时 16bit 寄存器地址， 8bit 数据。数据的具体内容， 可以查看相关的adv7511 寄存器手册。

```
always@(reg_index)
begin
    case(reg_index)
    0 :reg_data <= 24'h004150 ;
    1 :reg_data <= 24'h004110 ;
    2 :reg_data <= 24'h009803 ;
    3 :reg_data <= 24'h009a03 ;
    4 :reg_data <= 24'h009c30 ;
    5 :reg_data <= 24'h009d01 ;
    6 :reg_data <= 24'h00a2a4 ;
    7 :reg_data <= 24'h00a3a4 ;
    8 :reg_data <= 24'h00e0d0 ;
    9 :reg_data <= 24'h00f900 ;
    10:reg_data <= 24'h001520 ;
    11:reg_data <= 24'h001630 ;
    12:reg_data <= 24'h00af02 ;
    13:reg_data <= 24'h000100 ;
    14:reg_data <= 24'h000218 ;
    15:reg_data <= 24'h000300 ;
    16:reg_data <= 24'h000a03 ;
    17:reg_data <= 24'h000b6e ;
    18:reg_data <= 24'h000cbd ;
    19:reg_data <= 24'h00d6c0 ;

    default:reg_data <= 24'h000000;
    endcase
end
```

系统时钟域



hdmi 控制模块，reg_config_adv_7511_phy

adv7511 的 uart 调试接口：

这个接口的逻辑和camera 的调试接口非常类似，只是在状态0时，接受的数据选择 02， 代表从计算机发来的调试 是指向 adv7511 芯片的（不是ov5640）。其他的状态机与camera 调试接口相同。

uart接口在工程中会经常使用，实现这个接口所需逻辑不多，但给调试带来极大的方便，在工程中会经常使用（经验）。灵活使用，将大大提高工作效率，同时在解决突发bug时，也有好处。

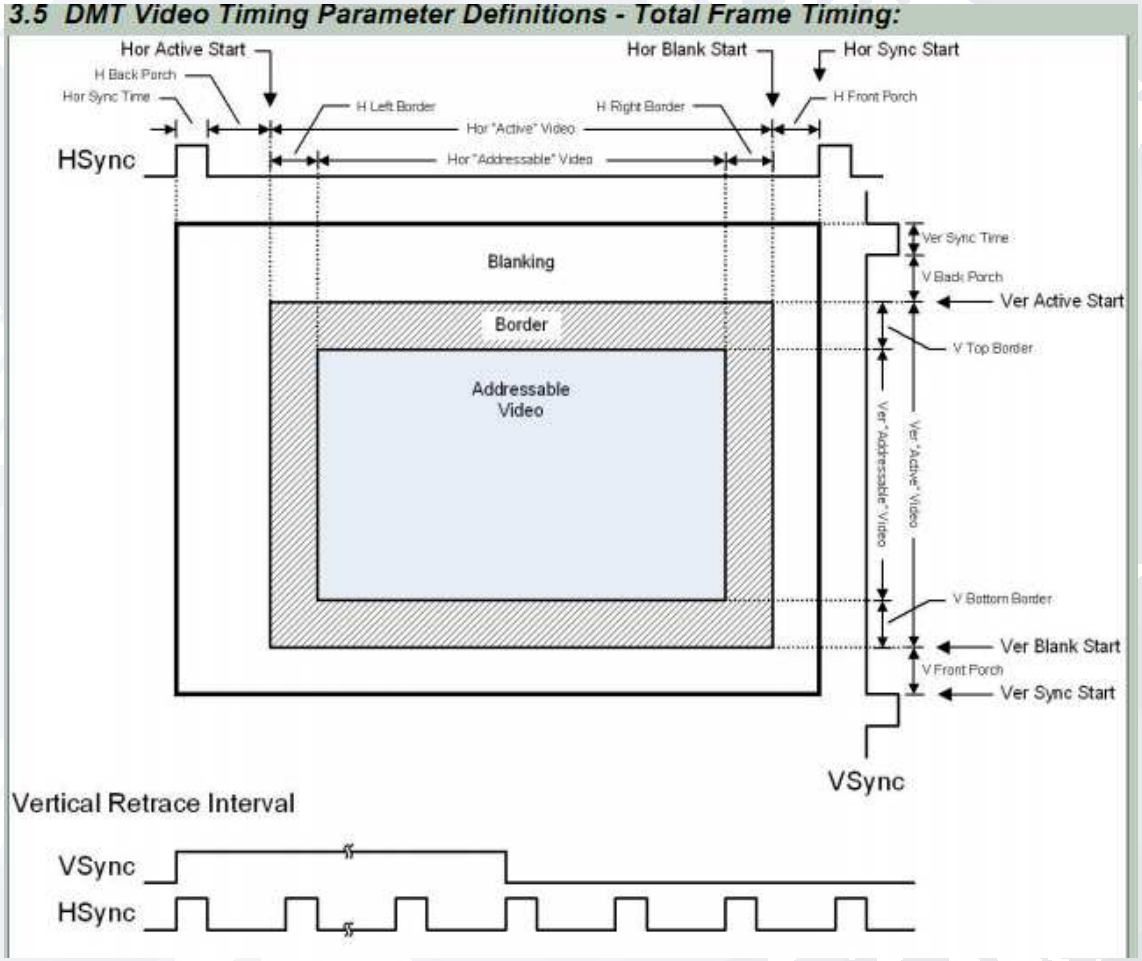

uart调试接口
I2C总线
hdmi_reg_done
reg_config_7511_phy
ADV7511控制模块
display
显示输出模块
hdmi_rd_en
Hdmi数据总线
hdmi_end
vga_hs
vga_vs
vga_r
vga_g
vga_b
vga_clk
data_en
hdmi 时钟域
显示输出模块

display 接口：

根据VESA 标准， fpga 模块输出标准的VESA信号。

在这个模块中，我们需要输出Hsync，Vsync，active de， 以及输出相关的clk时钟。

在de信号产生之前一个时钟周期，输出hdmi_rd_en信号，通知fifo，输出相关的视频数据。



```
LinePeriod    = 800 ; //     行周期数
H_SyncPulse   = 96  ; //     行同步脉冲（Sync a
H_BackPorch   = 48  ; //     显示后沿   （Back p
H_ActivePix   = 640 ; //     显示时序段（Displa
H_FrontPorch  = 16  ; //     显示前沿   （Front
Hde_start     = H_SyncPulse +  H_BackPorch;
Hde_end       = Hde_start +  H_ActivePix;

描参数的设定640*480 VGA 60FPS  25.175

FramePeriod   = 525 ; //     列周期数
V_SyncPulse   = 2   ; //     列同步脉冲（Sync
V_BackPorch   = 33  ; //     显示后沿（Back pc
V_ActivePix   = 480 ; //     显示时序段（Displ
V_FrontPorch  = 10  ; //     显示前沿（Front p
Vde_start     = V_SyncPulse + V_BackPorch  ;
Vde_end       = Vde_start + V_ActivePix ; //
```

Timing Name	= **640 x 480 @ 60Hz;**					
Hor Pixels	= **640;**	// Pixels				
Ver Pixels	= **480;**	// Lines				
Hor Frequency	= 31.469;	// kHz	=	31.8 usec	/	line
Ver Frequency	= 59.940;	// Hz	=	16.7 msec	/	frame
Pixel Clock	= **25.175;**	// MHz	=	39.7 nsec	±	0.5%
Character Width	= **8;**	// Pixels	=	317.8 nsec		
Scan Type	= **NONINTERLACED;**			// H Phase =		2.0 %
Hor Sync Polarity	= **NEGATIVE;**	// HBlank	=	18.0% of HTotal		
Ver Sync Polarity	= **NEGATIVE;**	// VBlank	=	5.5% of VTotal		
Hor Total Time	= 31.778;	// (usec)	=	100 chars	=	800 Pixels
Hor Addr Time	= 25.422;	// (usec)	=	80 chars	=	640 Pixels
Hor Blank Start	= 25.740;	// (usec)	=	81 chars	=	648 Pixels
Hor Blank Time	= 5.720;	// (usec)	=	18 chars	=	144 Pixels
Hor Sync Start	= 26.058;	// (usec)	=	82 chars	=	656 Pixels
// H Right Border	= 0.318;	// (usec)	=	**1** chars	=	8 Pixels
// H Front Porch	= 0.318;	// (usec)	=	**1** chars	=	8 Pixels
Hor Sync Time	= 3.813;	// (usec)	=	**12** chars	=	96 Pixels
// H Back Porch	= 1.589;	// (usec)	=	**5** chars	=	40 Pixels
// H Left Border	= 0.318;	// (usec)	=	**1** chars	=	8 Pixels
Ver Total Time	= 16.683;	// (msec)	=	525 lines		HT – (1.06xHA)
Ver Addr Time	= 15.253;	// (msec)	=	480 lines		= 4.83
Ver Blank Start	= 15.507;	// (msec)	=	488 lines		
Ver Blank Time	= 0.922;	// (msec)	=	29 lines		
Ver Sync Start	= 15.571;	// (msec)	=	490 lines		
// V Bottom Border	= 0.254;	// (msec)	=	**8** lines		
// V Front Porch	= 0.064;	// (msec)	=	**2** lines		
Ver Sync Time	= 0.064;	// (msec)	=	**2** lines		
// V Back Porch	= 0.794;	// (msec)	=	**25** lines		
// V Top Border	= 0.254;	// (msec)	=	**8** lines		

x_cnt 计数器：

这个计数器用于产生h，v，de 等信号。

```
//-------------------------------------------------------
////////// 水平扫描计数
//-------------------------------------------------------
always @ (posedge vga_clk)
begin
    if( (vga_rst_n == 1'b0 ) || (~hdmi_reg_done) )
    begin
        x_cnt <= 10'd1;
    end
    else
    begin
        if(x_cnt == LinePeriod)
            x_cnt <= 1;
        else
            x_cnt <= x_cnt + 1;
    end
end
```

产生h 信号：

注意，当前产生的h信号 是正电平的，而VESA 需要的是负电平的。

一般情况下在fpga 内部都是使用正电平，只有到端口输出时：选择负电平。

```
//------------------------------------------------------------------------
////////// 水平扫描信号hsync,hsync_de产生
//------------------------------------------------------------------------

// 产生 hsync  信号
always @ (posedge vga_clk)
begin
    if( (vga_rst_n == 1'b0 ) || (~hdmi_reg_done) )
        hsync_r <= 1'b0;
    else if(x_cnt == LinePeriod)
        hsync_r <= 1'b1;
    else
        if(x_cnt == H_SyncPulse)
            hsync_r <= 1'b0    ;
        else                   ;
end
```

产生h方向的 de 信号：

h方向的信号，和之后v方向的de信号组合， 最终会输出VESA 所需要的video de 信号。

de 信号，就是VESA 标准中的active video 信号，在很多芯片的定义中，使用de，或者是vde。

de 信号 一般来说都是高电平的，很少有低电平有效的信号。

```
// 产生 hsync_de 信号
always @ (posedge vga_clk)
begin
    if( (vga_rst_n == 1'b0 ) || (~hdmi_reg_done) )
        hsync_de <= 1'b0;
    else
    begin
        if(x_cnt == Hde_start)
            hsync_de <= 1'b1;
        else
        begin
            if(x_cnt == Hde_end)
                hsync_de <= 1'b0;
            else    ;
        end
    end
end
```

场计数器：

当每行的pixel 寄存器结束后，场计数器加一

场计数器用于产生vde， v 信号 。

```
//------------------------------------------------------------------------
////////// 垂直扫描计数
//------------------------------------------------------------------------
always @ (posedge vga_clk)
begin
    if( (vga_rst_n == 1'b0 ) || (~hdmi_reg_done) )
        y_cnt <= 10'd1;
    else
    begin
        if (x_cnt == LinePeriod)
        begin
            if  ( y_cnt == FramePeriod)
                y_cnt <= 1;
            else
                y_cnt <= y_cnt + 1;
        end
        else
            y_cnt <= y_cnt;
    end
end
```

v 信号：

当每行的pixel 寄存器结束后，同时场计数器开始为1，y_cnt == v_syncpulse 后 v信号为0.

```
//------------------------------------------------------------------
////////// 垂直扫描信号vsync，vsync_de产生
//------------------------------------------------------------------

//产生vsync信号
always @ (posedge vga_clk)
begin
    if( (vga_rst_n == 1'b0 ) || (~hdmi_reg_done) )
        vsync_r <= 1'b0;
    else
    begin
        if( (y_cnt == FramePeriod) & (x_cnt == LinePeriod) )
            vsync_r <= 1'b1;
        else
        begin
            if( (y_cnt == V_SyncPulse) & (x_cnt == LinePeriod) )
                vsync_r <= 1'b0;
            else  ;
        end
    end
end
```

场方向de 信号：

当每行的pixel 寄存器结束后，根据VESA标准，产生相应的 场方向de信号。

h方向的de 和 v方向的de 组合，最终输出VESA 标准de

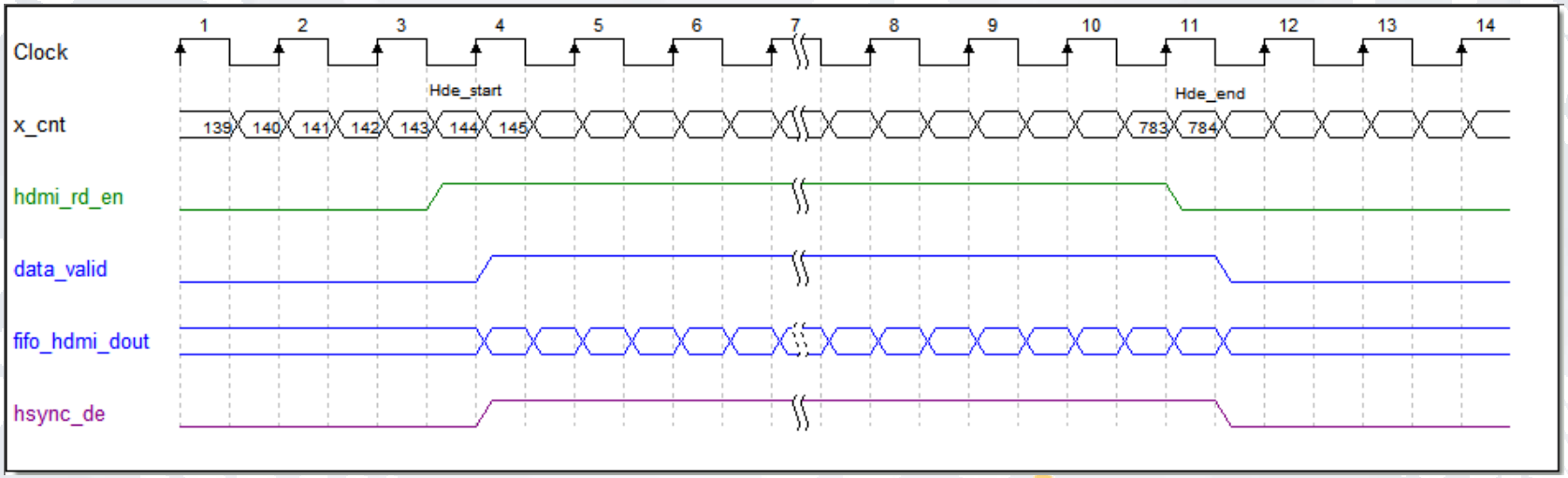
```
//产生vsync_de信号
always @ (posedge vga_clk)
begin
    if( (vga_rst_n == 1'b0 ) || (~hdmi_reg_done) )
        vsync_de <= 1'b0;
    else
    begin
        if( (y_cnt == Vde_start) & (x_cnt == LinePeriod) )
            vsync_de <= 1'b1;
        else
        begin
            if( (y_cnt == Vde_end) & (x_cnt == LinePeriod) )
                vsync_de <= 1'b0;
            else  ;
        end
    end
end
```

```
// HDMI 接口图像显示有效RGB888的图像数据的标识信号
assign data_en = hsync_de & vsync_de;
```

产生读取fifo的信号：

fifo的读取信号比de信号早一个时钟周期，结束也早一个时钟周期。

```
// 产生从 fifo_128_to_16读取图像数据的读使能
always @ (posedge vga_clk)
begin
    if ( (vga_rst_n == 1'b0 ) || (~hdmi_reg_done) )
        hdmi_rd_en <= 0 ;
    else
    begin
        if ( vsync_de && (x_cnt == Hde_start - 1) )
            hdmi_rd_en <= 1 ;
        else if (vsync_de && (x_cnt == Hde_end - 1) )
            hdmi_rd_en <= 0 ;
        else ;
    end
end
```


1
2
3
4
5
6
7
8
9
10
11
12
13
14
Clock
Hde_start
Hde_end
x_cnt
139
140
141
142
143
144
145
783
784
hdmi_rd_en
data_valid
fifo_hdmi_dout
hsync_de

最终输出的所有信号：

fifo输出的图像为RGB565，hdmi接口为RGB888，　所以rgb 信号需要补齐到8bit

de 为hde和vde 的组合

由于vesa 需要h，v 为低电平，所以最终fpga 输出取反。

```
// 当输入图像数据有效时，输出该图像数据

assign vga_r = data_valid ? {fifo_hdmi_dout [15:11],3'b000 } : 8'h00;
assign vga_g = data_valid ? {fifo_hdmi_dout [10:5], 2'b00  } : 8'h00;
assign vga_b = data_valid ? {fifo_hdmi_dout [4:0],  3'b000 } : 8'b00;


// HDMI 接口图像显示有效RGB888的图像数据的标识信号
assign data_en = hsync_de & vsync_de;

assign vga_hs = ~hsync_r;    // negative
assign vga_vs = ~vsync_r;    // negative

assign hdmi_end = vsync_r;
```

1 整体图像采集工程设计

2 图像采集模块OV5640简介配置

3 采集图像数据存储

4 HDMI接口的配置与现实

5 习题

练习：

- 掌握SCCB协议的时序关系；
- 依据老师提供的寄存器配置信息配置摄像头模块，使其正常工作采集到图像数据；
- 掌握工程模块化设计思想，搭建图像采集电路，实现拍照功能（拍照功能使用按键控制，闪光灯开关使用按键控制）；
- 实验1：将拍照的图片保存到ddr上， 增加UP 按键， 将图片读取出来显示到屏幕上，增加down 按键，返回预览模式
- 实验2：通过uart 调试模式，修改ov5640 寄存器0x3801, 0x3803， 增加UP，DOWN, LEFT, RIGHT 按键，实现摄像头上下左右移动。（可参考闪关灯逻辑）

未来的你，一定会感谢今天拼命努力的自己！

THANK YOU !

官方网站：http://www.icfusion.cn/

淘宝店铺：https://shop296232497.taobao.com/

微信客户端

淘宝客户端