

5 并行接口 LCD 和摄像头控制模块设计

如果 FPGA 开发板没有 ADV7511 芯片，也没有高速串行接口，例如 Nexys 4 DDR 开发板，那么无法直接使用 HDMI。这时可以使用并行接口的 LCD 进行显示。由于非差分的并行接口一般速度较慢，并行 LCD 通常分辨率较低，本节实验在 Nexys 4 DDR 上实现 854x480 的并行 LCD 显示。

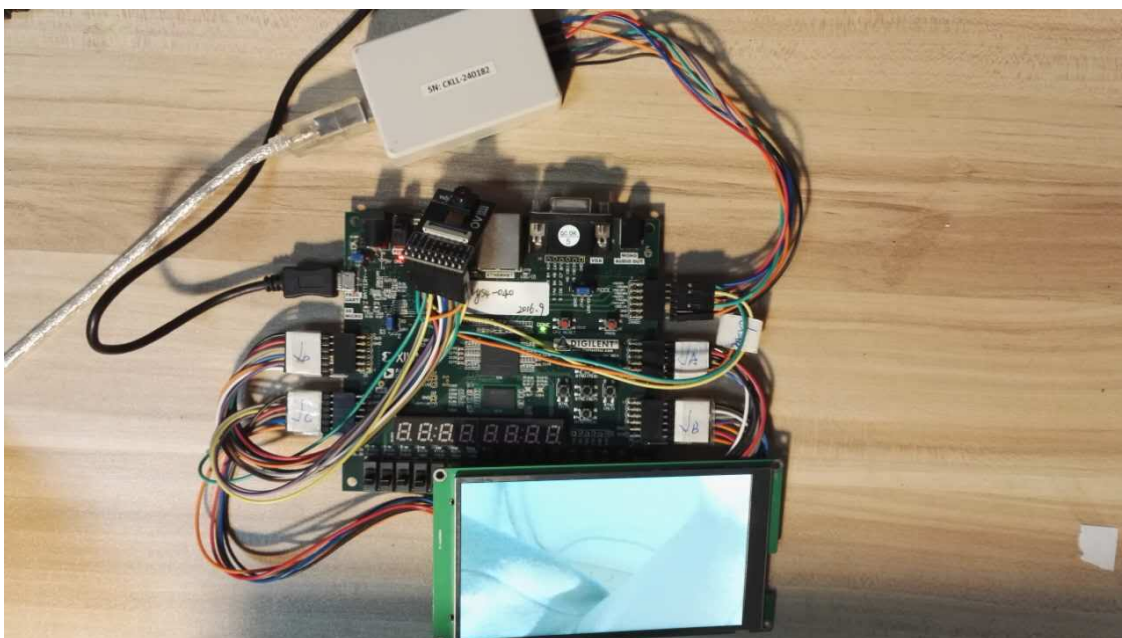


Figure 1. CK807 LCD

通常 LCD 都需要一个 LCD 驱动芯片，这个驱动芯片接收命令和要显示的内容，然后驱动 LCD 实现显示。本节实验采用 ILI9806G 驱动的并行 LCD，Figure 1 是 Nexys 4 DDR 接并行 LCD 的方法，由于没有专用接口，因此采用杜邦线直接连接。

本章的实验可以分为两部分，可以先完成 LCD 显示后再完成 OV2640 摄像头视频采集。

5.1 并行 LCD 接口

非差分的并行 LCD 接口是一种比较早期的 LCD 接口，它采用并行传输数据的方法，速度较低，只能支持较低的分辨率，目前正在被差分串行接口，如高速的 MIPI 和低速的 SPI 等替代。由于驱动简单，在一些单片机等低端领域还在继续使用。

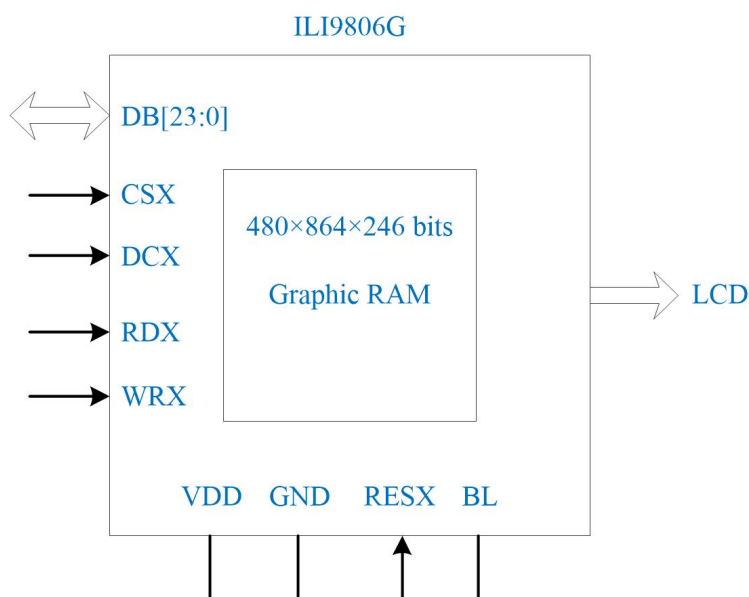


Figure 2. ILI9806G 示意图

Figure 2 是 ILI9806G 的简化示意图，它只包含了本节实验要用到的引脚。由于芯片内包含了 GRAM(Graphic RAM)，因此只需要把显示内容写入 GRAM，不需要发送任何的显示控制信号，由 ILI9806G 自己生成显示控制信号驱动 LCD。另一方面，GRAM 存储了要显示的内容，相当于一个缓冲区，因此不需要按照严格的时序发送显示内容给 ILI9806G，第四章用到的 cache-line 在这种情况下不需要使用。

并行 LCD 接口有多种形式，本节实验的 ILI9806G 使用如下引脚：

□ VDD

3.3 V，由于本节实验的 LCD 功耗不高，因此可以直接使用 Nexys 4 DDR 的 pmod 的 VDD。

☐ GND

接地端，可以使用 Nexys 4 DDR 的 pmod 的 GND。

☐ BL

背光，可以使用 Nexys4 DDR 的 pmod 的 VDD。

☐ DB[15:0]

16 位并行数据和命令接口，由 DCX 引脚决定作为数据还是命令接口。

当传输命令时，使用 DB[7:0]。

当传输数据时，使用 DB[15:0]，数据使用 RGB565 格式。

☐ CSX

片选，低电平有效。本实验可以把它接 GND。

☐ DCX

信号和命令选择引脚。

低电平，D[7:0]为命令。

高电平，D[15:0]为数据。

☐ RDX

读信号，从 LCD 读回数据，上升沿有效。

☐ WRX

写信号，上升沿有效。

☐ RESX

复位，低电平有效。

如下表所示，DCX 决定了传输的是命令还是数据，WRX 和 RDX 的上升沿决定了写入还是读出。接口中没有时钟信号，传输采用异步模式。

WRX	RDX	DCX	功能
↑	H	L	发送命令
H	↑	H	读取显示数据或参数
↑	H	H	发送显示数据或参数

为了节约引脚给 OV2640 摄像头，本节实验采用 16 bit 并行数据接口。

并行 LCD 驱动芯片需要先初始化，再发送显示数据。初始化过程和第四章的 HDMI 芯片初始化类似，需要在上电后给 LCD 芯片发送初始化命令。

发送命令需要使用四组信号，WRX、RDX、DCX 和 DB[7:0]。一般的命令发送格式为“命令代码”+“参数”，以初始化的第一条 EXTC 命令为例，命令代码是十六进制 FFh，参数为 FFh、98h 和 06h，需要 4 次发送。

	DCX	RDX	WRX	DB[7:0]
命令代码	0	1	↑	FFh
参数 1	1	1	↑	FFh
参数 2	1	1	↑	98h
参数 3	1	1	↑	06h

EXTC 是 LCD 初始化发送的第一个指令，包含了连续 4 次发送：

1. DCX=0

发送命令代码，WRX 为上升沿，把 D[7:0]的值发送给 LCD 控制芯片。

2. DCX=1

发送三个参数，WRX 为上升沿，把 D[7:0]的值发送给 LCD 控制芯片。

本节实验配置显示窗口地址的命令如下：

	DCX	RDX	WRX	DB[7:0]
命令代码	0	1	↑	2Ah
参数 1	1	1	↑	00h

参数 2	1	1	↑	00h
参数 3	1	1	↑	03h
参数 4	1	1	↑	20h
命令代码	0	1	↑	2Bh
参数 1	1	1	↑	00h
参数 2	1	1	↑	00h
参数 3	1	1	↑	01h
参数 4	1	1	↑	DFh

其中 2Ah 和 2Bh 分别是显示窗口行和列地址设置，上述命令将显示的窗口设置成 854x480。

初始化完成后，就可以对 LCD 控制芯片写入要显示的内容，命令代码是 2Ch。

	DCX	RDX	WRX	DB
命令代码	0	1	↑	2Ch
参数 1	1	1	↑	DB[15:0]
... ..	1	1	↑	DB[15:0]
参数 n	1	1	↑	DB[15:0]

每一帧开始时，发送 2Ch 命令代码，接下来发送要显示的内容，对于本实验，采用的是 16 位数据宽度，所以每次发送 16bit。因为分辨率是 854x480，因此共有 854x480 个参数。

因为 ILI9806G 芯片内部包含了存储单元 (Graphic RAM)，每个像素点被写入的数值都会被存储，如果停止写入显示数据，显示的内容就会保持原有的值不变，这一点和第四章的 HDMI 显示方法不同。如果要显示动态视频，需要不停的刷新显示存储的值。

ILI9806G 初始化代码请参考芯片手册和本节提供的 `lcd_driver.v`。

发送命令的波形示意图如 Figure 3 所示。在 WRX 的第一个上升沿，发送了命令代码 DB[7:0]，在 WEX 的第二个上升沿，发送了数据 DB[7:0]。在 LCD 初始化阶段，不断重复这个方法，发送各种初始化命令。

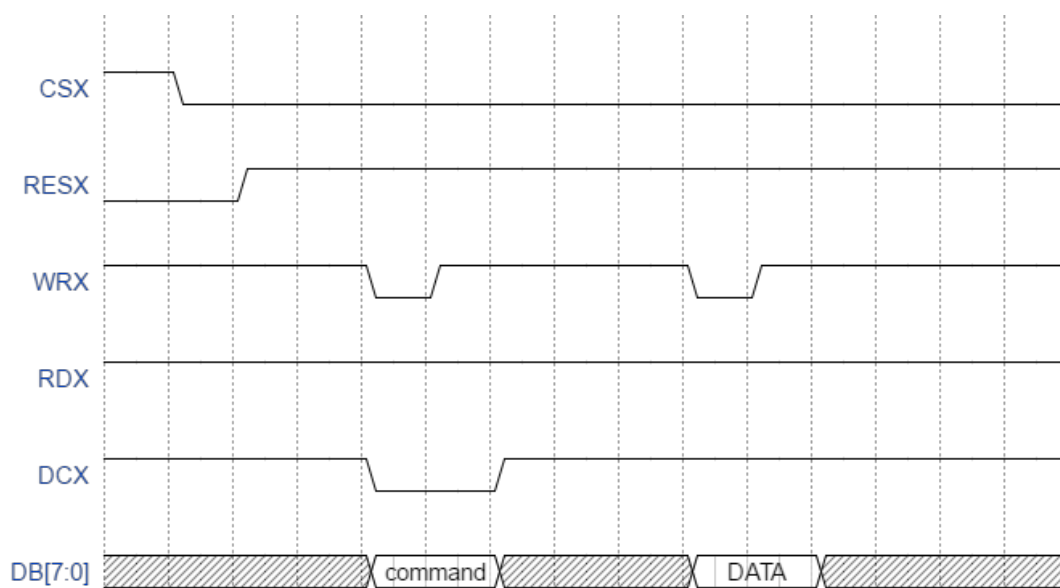


Figure 3. 发送命令

写入要显示的内容波形示意图如 Figure 4 所示，初始化完成后，在每一帧开始时发送 8 bit 的命令代码 2Ch，接着不停的发送 16 bit 的显示内容 DB[15:0]，显示内容会被写入内部的 GRAM。

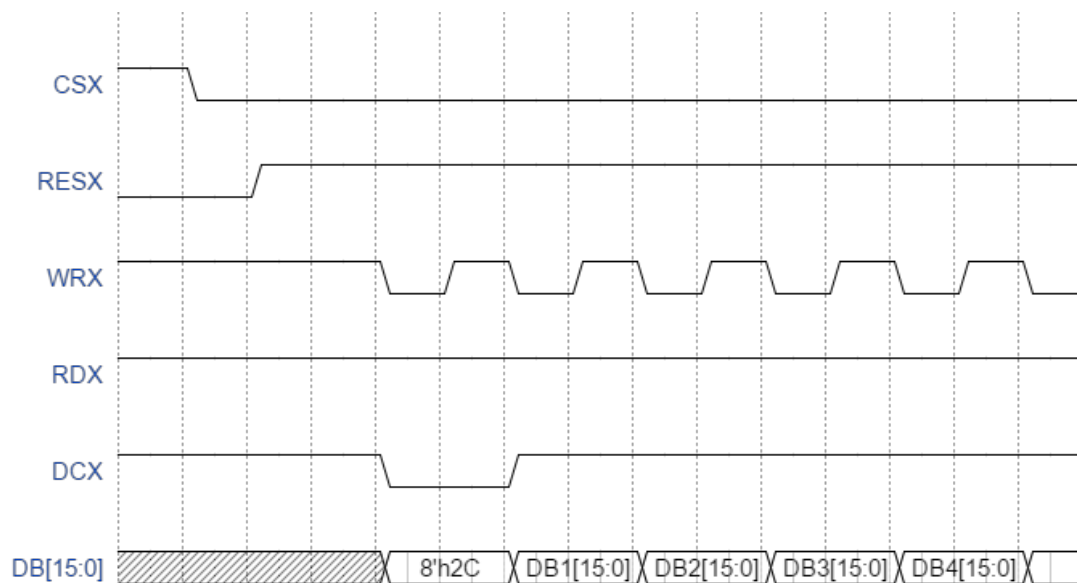


Figure 4. 写入要显示的内容

5.2 并行 LCD 控制器设计

并行 LCD 控制器设计包含三个部分，分别用于发送初始化命令、发送显示内容和 AXI 接口。

A. 初始化电路

初始化电路有两种实现方法，第一种可以参照 ADV7511 的 IIC 初始化电路，用硬件电路生成 100 多条初始化命令的波形，发送到 LCD。这种方法的好处是设计简单，用类似移位寄存器的方法可以很容易实现。从 5.1 节可以看出，初始化过程和发送显示数据的过程基本是一样的，区别在于初始化用了 DB[7:0]，发送显示数据用了 DB[15:0]。因此可以共用同一个发送电路。

但这种方法也有一些缺点，首先是修改初始化命令不方便，由于初始化命令被固化在电路里，修改初始化命令需要修改电路。对于 FPGA，需要重新综合代码，对于 ASIC，则需要重新流片，这显然是不现实的。其次，硬件存储初始化命令需要一块较大的存储器，本节这块存储器可能是 256x19，它只在初始化时使用一次，利用率不高，提高了系统成本，增加了功耗。因此，在实际的电路设计中，可

以结合驱动软件和部分硬件电路完成 LCD 控制芯片的初始化。初始化指令存储在驱动软件里，SoC 运行时这部分内容被存储在 SoC 的存储空间，由于 SoC 的 DDR 等存储相对 SoC 硬件内部的存储要便宜很多，因此可以降低系统硬件成本。驱动软件的修改不需要改动硬件，因此更加方便。

本例中 ILI9806G 初始化采用软件初始化，代码请参考芯片手册，本节提供的 `lcd_driver.v` 里包含了初始化参数，本节 C 部分对软件初始化方法也进行了详细介绍。

B. 发送显示内容

发送显示内容电路要实现的功能是把 SoC 存储的一块空间内容写入 LCD，这个存储空间在系统上电后存储的是初始化命令，在显示阶段存储的是要显示的内容。也可以用两块空间分别存储初始化命令和显示内容。

发送过程只需要控制 WRX 和 DCX，配合 DB[15:0] 产生合适的波形即可。波形可参考 Figure 4。

在发送初始化命令时，为了简化电路设计，DB[15:8] 这 8 位可以发送任意的数值，这些值会被 ILI9806G 忽略。

发送数据的频率受 ILI9806G 和 FPGA 引脚以及杜邦线的最高频率限制，理论上限为 33 MHz。对本实验，20 - 30 MHz 可以实现较流畅的视频显示效果。

LCD 显示的数据格式为 RGB565，R/G/B 的顺序如 Figure 5 所示。

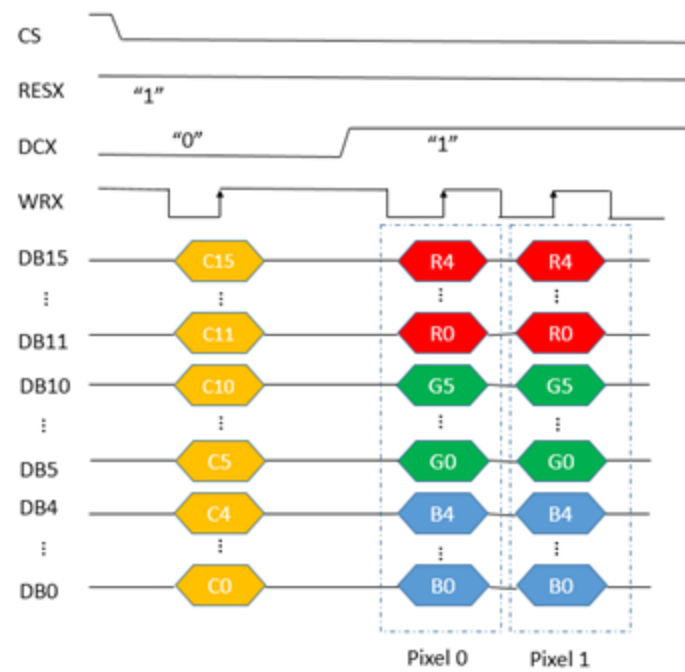


Figure 5. RGB565 传输

C. AXI 接口和软件设计

AXI 接口用于 SoC 与 LCD 控制器进行通讯。由于需要传输大量的内存内容，因此需要 AXI-Full 接口。在 SoC 的内存空间保留一块 854x480x16 的空间作为显示缓存，LCD 控制器主动的搬运内存内容到 LCD 驱动芯片，因此 AXI 接口为 master (虽然 ILI9806G 内部已经包含了 GRAM，可以不用在内存内保留缓存区域，也可以用被动的方法搬运内存，但为了接下来的摄像头实验方便，这里保留缓存并且使用 master 接口)。这个 AXI-Full master 接口的功能与第四章的 HDMI 控制器类似，不停的搬运显存内的数据到 LCD 驱动器。

除了 AXI-Full master 接口用于传输内存内的显示数据，还需要一个 AXI-lite slave 接口用于控制这个 LCD 控制器。这个接口通过写 LCD 控制器模块内的寄存器，设置模块处于初始化状态还是处于显示状态。同时可以通过写入寄存器的值，设置初始化命令内存空间的基地址和显存的基地址。LCD 控制器的状态也可以通过读取内部寄存器的值，从这个 AXI 接口返回。例如可以通过内部寄存器的值得知初始化是否完成等。由于这些操作数据量很小，因此使用 AXI-lite 即可。

AXI-lite slave 内部的寄存器可以包含如下内容:

1. 显存地址 VIDEOMEM_ADDR[31:0]
 这个寄存器组用于存储 SoC 系统内存中显存的位置, 由驱动软件通过 CPU 写入。这样显存的地址不需要固定, 可以由软件修改。

2. 显存长度 VIDEOMEM_LENGTH[31:0]
 配合 VIDEOMEM_ADDR[31:0], 告诉 LCD 控制器显存的大小。LCD 控制器从 VIDEOMEM_ADDR 开始按时主动的搬运 VIDEOMEM_LENGTH 个数据, 配合不同的初始化命令, 可以用软件实现不同的分辨率显示。

3. 初始化命令地址 LCDINITCMD_ADDR[31:0]
 这个寄存器组用于存储初始化命令在内存中的位置, 驱动软件需要将初始化命令存储在这个内存位置。任何初始化命令的改变都可以通过修改驱动软件来实现。

4. 初始化命令长度 LCDINITCMD_LENGTH[31:0]
 这个寄存器组用于存储初始化命令的长度, 驱动软件需要把这个值传给 LCD 控制器。

5. LCD 控制寄存器 LCD_CTRL[31:0]
 LCD 控制寄存器用于控制初始化等功能。可以将这 32 bit 寄存器分成多个开关位, 例如:

LCD_CTRL[0]=1	初始化
LCD_CTRL[1]=1	发送显示命令
LCD_CTRL[2]=1	关闭 LCD 控制器
... ..	

6. LCD 状态寄存器 LCD_STATUS[31:0]

LCD 状态寄存器用于用于 CPU 读取 LCD 控制器的状态，同样这 32 个寄存器可以分为多个状态位，例如：

LCD_STATUS[0]=1	初始化完成
LCD_STATUS[1]=1	LCD 控制器处于显示状态
LCD_STATUS[8:2]	LCD 分辨率
LCD_STATUS[9]=1	LCD 控制器关闭
...	...

这六个寄存器组的命名是任意的，它们通过各自的地址进行访问。对于只读寄存器，需要在 AXI 接口设计时在硬件上设计为不可写。寄存器组的写操作如下。

```
reg [31:0]    VIDEOMEM_ADDR;
reg [31:0]    VIDEOMEM_LENGTH;
reg [31:0]    LCDINITCMD_ADDR;
reg [31:0]    LCDINITCMD_LENGTH;
reg [31:0]    LCD_CTRL;
reg [31:0]    LCD_STATUS;

parameter addr_VIDEOMEM_ADDR    3'h0;
parameter addr_VIDEOMEM_LENGTH  3'h1;
parameter addr_LCDINITCMD_ADDR  3'h2;
parameter addr_LCDINITCMD_LENGTH 3'h3;
parameter addr_LCD_CTRL         3'h4;
parameter addr_LCD_STATUS       3'h5;

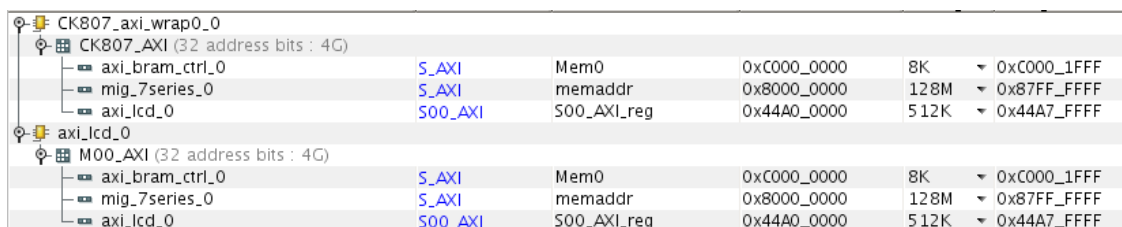
always @(posedge S_AXI_ACLK)
begin
    if ( S_AXI_ARESETN == 1'b0 )
    begin
        VIDEOMEM_ADDR    <= 0;
        VIDEOMEM_LENGTH  <= 0;
        LCDINITCMD_ADDR  <= 0;
        LCDINITCMD_LENGTH <= 0;
        LCD_CTRL         <= 0;
    end
    else
    begin
        if (slv_reg_wren)
        begin
```

```

case ( axi_awaddr[4:2] )
    addr_VIDEOMEM_ADDR      : VIDEOMEM_ADDR      <= S_AXI_WDATA ;
    addr_VIDEOMEM_LENGTH    : VIDEOMEM_LENGTH    <= S_AXI_WDATA ;
    addr_LCDINITCMD_ADDR    : LCDINITCMD_ADDR    <= S_AXI_WDATA ;
    addr_LCDINITCMD_LENGTH  : LCDINITCMD_LENGTH  <= S_AXI_WDATA ;
    addr_LCD_CTRL           : LCD_CTRL           <= S_AXI_WDATA ;
    default                 :
        begin
            VIDEOMEM_ADDR    <= VIDEOMEM_ADDR;
            VIDEOMEM_LENGTH  <= VIDEOMEM_LENGTH;
            LCDINITCMD_ADDR  <= LCDINITCMD_ADDR;
            LCDINITCMD_LENGTH <= LCDINITCMD_LENGTH;
            LCD_CTRL         <= LCD_CTRL;
        end

    endcase // case ( axi_awaddr[4:2] )
end // if (slv_reg_wren)
end // else: !if( S_AXI_ARESETN == 1'b0 )
end // always @ (posedge S_AXI_ACLK)
    
```

从上可以看出，VIDEOMEM_ADDR 的地址是这个 AXI-slave 的基地址，第二个寄存器组 VIDEOMEM_LENGTH 的地址是基地址加 4(32 bit 等于 4 byte)，其余类推。LCD_STATUS 是只读寄存器组，所以在 case 语句里，没有对它进行写操作。AXI slave 的基地址在 SoC 设计时，在硬件上实现。如 Figure 6 所示。AXI master 端口的地址空间必须包含 VIDEOMEM_ADDR 和 LCDINITCMD_ADDR 这两个存储空间。



Component	Signal	MMIO Region	Start Address	Size	End Address
CK807_axi_wrap0_0					
CK807_AXI (32 address bits : 4G)					
axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	8K	0xC000_1FFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF
axi_lcd_0	S00_AXI	S00_AXI_reg	0x44A0_0000	512K	0x44A7_FFFF
axi_lcd_0					
M00_AXI (32 address bits : 4G)					
axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	8K	0xC000_1FFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF
axi_lcd_0	S00_AXI	S00_AXI_reg	0x44A0_0000	512K	0x44A7_FFFF

Figure 6. 设置硬件地址

寄存器组的读操作如下。

```

assign slv_reg_rden = axi_arready & S_AXI_ARVALID & ~axi_rvalid;

always @(*)
    
```

```

begin
  if ( S_AXI_ARESETN == 1'b0 )
    begin
      reg_data_out <= 0;
    end
  else
    begin
      // Address decoding for reading registers
      case ( axi_araddr[4:2] )
        addr_VIDEOMEM_ADDR :
          reg_data_out <= VIDEOMEM_ADDR ;
        addr_VIDEOMEM_LENGTH :
          reg_data_out <= VIDEOMEM_LENGTH ;
        addr_LCD_CTRL :
          reg_data_out <= LCD_CTRL ;
        addr_LCD_STATUS :
          reg_data_out <= LCD_STATUS ;
        default :
          reg_data_out <= 0;
      endcase
    end
  end

  // Output register or memory read data
  always @( posedge S_AXI_ACLK )
    begin
      if ( S_AXI_ARESETN == 1'b0 )
        begin
          axi_rdata <= 0;
        end
      else
        begin
          if (slv_reg_rden)
            begin
              axi_rdata <= reg_data_out;    // register read data
            end
          end
        end
      end
    end
end

```

寄存器读操作与写类似，译码后读取对应的寄存器组。可以在硬件上设置某些寄存器组不可读，作为例子，LCDINITCMD_ADDR 和 LCDINITCMD_LENGTH 没有被读出。

LCD_STATUS 用于传递 LCD 控制器的状态给 CPU，它从 LCD 控制器的专用端口引脚获取 LCD 控制器的状态，LCD_STATUS[0]用于 LCD 的 INIT_DONE。

```
always @( posedge S_AXI_ACLK )
begin
    if ( S_AXI_ARESETN == 1'b0 )
    begin
        LCD_STATUS  <= 0;
    end
    else
    begin
        LCD_STATUS[0] <= init_done;
        ... ..
    end
end // always @ ( posedge S_AXI_ACLK )

// Add user logic here
LCD_ctrl  LCD_U1 (.clk_in(clk),
                  .CSX(cs),
                  .RDX(rd),
                  .WRX(wr),
                  .DB(lcd_data),
                  ... ..
                  .INIT_DONE(init_done)
                  );

// User logic ends
```

驱动软件需要保持这些寄存器地址与硬件地址相一致，这样驱动软件才能正确的读写这些寄存器。

```
#define LCD_AXI_MASTER_BASEADDR      0x8200_0000
#define LCD_AXI_SLAVE_BASEADDR       0x44A0_0000

#define LCD_AXIs_VIDEOMEM_ADDR        (LCD_AXI_SLAVE_BASEADDR + 0)
#define LCD_AXIs_VIDEOMEM_LENGTH      (LCD_AXI_SLAVE_BASEADDR + 4)
#define LCD_AXIs_LCDINITCMD_ADDR      (LCD_AXI_SLAVE_BASEADDR + 8)
#define LCD_AXIs_LCDINITCMD_LENGTH    (LCD_AXI_SLAVE_BASEADDR + 12)
#define LCD_AXIs_LCD_CTRL              (LCD_AXI_SLAVE_BASEADDR + 16)
#define LCD_AXIs_LCD_STATUS            (LCD_AXI_SLAVE_BASEADDR + 20)
```

LCD 控制器一共使用了 6 组 32 位寄存器，需要 6x32 个 D 触发器实现，对于较大的 SoC 系统而言，这不是很大的负担。如果要减少寄存器个数，可以减少合并寄存器组，例如 VIDEOMEM_ADDR 和 VIDEO_LENGTH 可以通过预先指定地址的位置范围等方法减少寄存器的个数。LCD_CTRL 和 LCD_STATUS 可以合并，硬件设计时只实现用到的那几个 bit。

LCD 控制器也可以采用另外的方法完成初始化命令的发送，在 AXI-slave 的接口设置一个寄存器组 LCD_INITCMD，配合 LCD_STATUS 和 LCD_CTRL，每次由 CPU 写入一条命令到 LCD_INITCMD 并设置 LCD_CTRL 的控制位，LCD 控制器由 LCD_CTRL 对应的控制位触发，发送 LCD_INITCMD 的内容，发送完成后更新 LCD_STATUS 的对应状态位。这样重复多次，就可以完成 LCD 的初始化。

包含 LCD 控制器的 SoC 原理图如 Figure 7 所示，其中 AXI master 接口用于传输显示内容，AXI slave 接口用于传输控制寄存器。

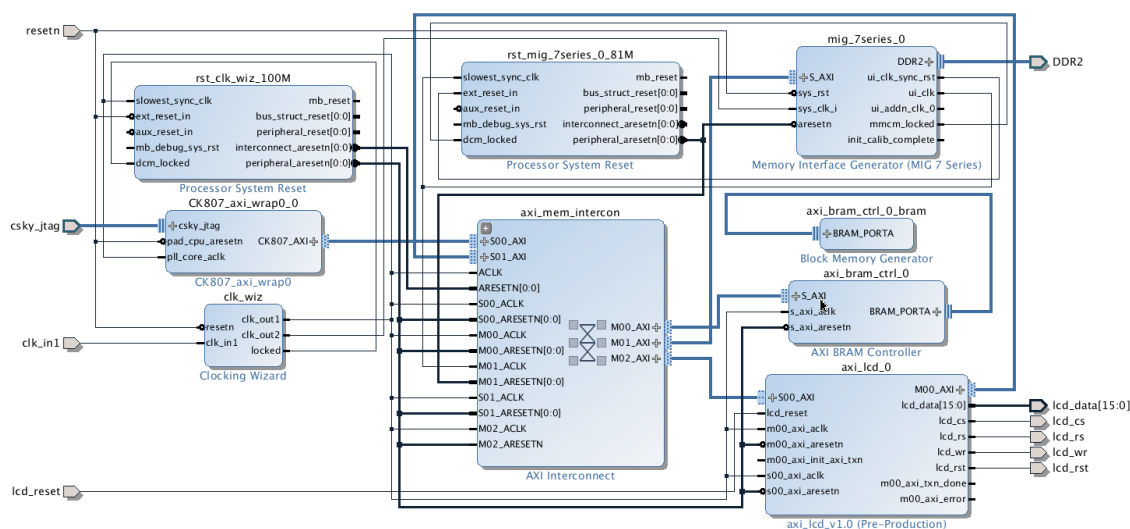


Figure 7. LCD SoC

LCD 控制器的驱动软件设计主要完成 LCD 的初始化，通过 CPU 写入显存地址等信息。驱动软件的寄存器地址与硬件地址必须一致。

```
#define LCD_AXI_MASTER_BASEADDR    0x8200_0000
#define LCD_AXI_SLAVE_BASEADDR     0x44A0_0000

#define LCD_VIDEOMEM_ADDR           (LCD_AXI_SLAVE_BASEADDR + 0)
#define LCD_VIDEOMEM_LENGTH        (LCD_AXI_SLAVE_BASEADDR + 4)
```

```

#define LCD_LCDINITCMD_ADDR      (LCD_AXI_SLAVE_BASEADDR + 8)
#define LCD_LCDINITCMD_LENGTH    (LCD_AXI_SLAVE_BASEADDR + 12)
#define LCD_LCD_CTRL              (LCD_AXI_SLAVE_BASEADDR + 16)
#define LCD_LCD_STATUS            (LCD_AXI_SLAVE_BASEADDR + 20)

```

SoC 的地址分配如下。

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
CK807_axi_wrap0_0					
CK807_AXI (32 address bits : 4G)					
axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	8K	0xC000_1FFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF
axi_lcd_0	S00_AXI	S00_AXI_reg	0x44A0_0000	512K	0x44A7_FFFF
Unmapped Slaves (1)					
axi_mem_intercon/s00_couplers/auto_pc	S_AXI	Reg			
axi_lcd_0					
M00_AXI (32 address bits : 4G)					
axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	8K	0xC000_1FFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF
axi_lcd_0	S00_AXI	S00_AXI_reg	0x44A0_0000	512K	0x44A7_FFFF
Unmapped Slaves (1)					
axi_mem_intercon/s01_couplers/auto_us	S_AXI	Reg			

LCD 控制器的测试与第四章类似，通过 CPU 向显存写入要显示的数据即可。实验步骤请参考视频 [video_5.1_lcd_demo](#)。

通过 LCD 控制器，理论上现在可以显示任意的视频等画面，但由于没有视频等输入信息，目前只能显示 CPU 算出的画面，或者存储在 ELF 文件中的画面。

5.3 并行接口摄像头控制器设计

本节介绍并行接口摄像头控制器的设计，设计完成的控制器可以将并行摄像头的视频信息读入 SoC，进而在 LCD 上显示。

并行接口摄像头是一种较早期的摄像头，一般用于 500 万以下分辨率，由于非差分的接口速度不高，因此不能实现较高的分辨率或帧率。由于接口协议简单，适合用于入门实验。手机摄像头目前广泛采用的是 MIPI (Mobile Industry Processor Interface) 接口，这是一种高速差分串行接口，可以实现 FHD 甚至 4K 以上的视频传输。将在第九章介绍 MIPI 接口摄像头控制器的设计。

本节实验采用 Omnivision 公司的 OV2640 摄像头，它是一个 200 万像素的彩色摄像头，Nexys 4 DDR 可以用杜邦线直接与 OV2640 的模块连接，如 Figure 8 所示。

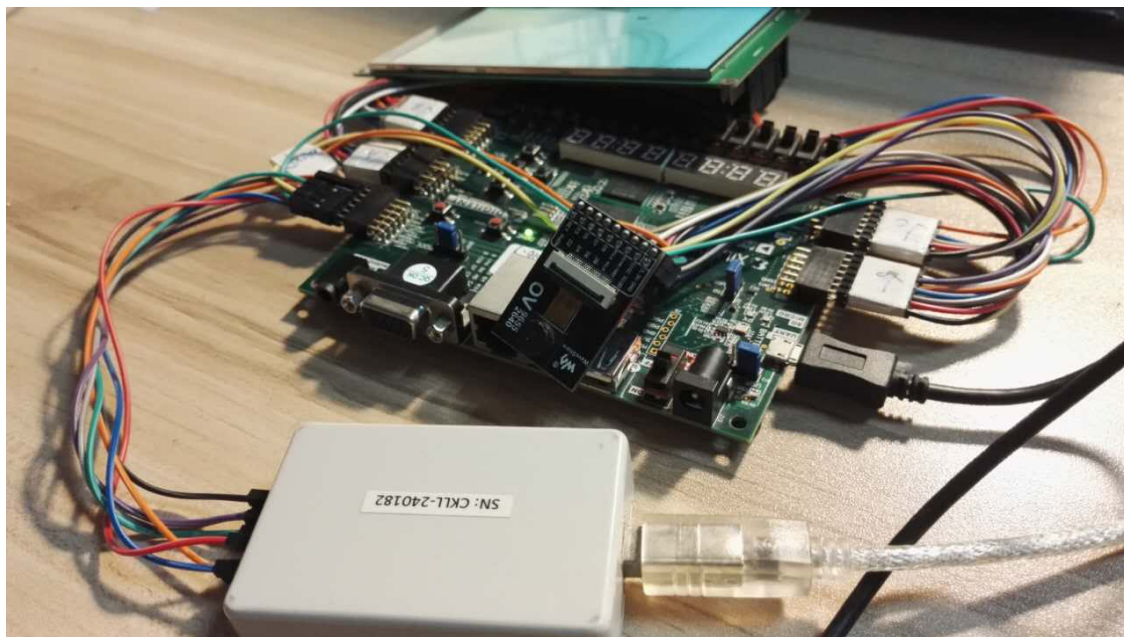


Figure 8. OV2640 摄像头连接 FPGA 开发板

OV2640 的主要功能包括：

□ 标准 SCCB 控制接口。

SCCB (Serial Camera Control Bus) 可以简单认为是 IIC 的一个改动很小的版本，对本实验，可以用类似第四章 IIC 的方法发送控制命令。

SCCB 用于发送控制命令给摄像头，初始化需要从这个端口发送很多命令和参数给摄像头，在摄像头工作时，如果需要改变摄像头的工作模式，也是通过这个端口发送命令。读取摄像头的状态等也是通过这个端口。因为这些操作对速度要求不高，所以可以使用 SCCB 接口，它的优点是只需要 2 个引脚。

- 输出格式支持 Raw RGB, RGB565, YUV422 和 JPEG 等。
RGB565 适合本实验用到的 LCD，因此不需要转化电路，可以直接使用 OV2640 的输出。本实验使用 RGB565 是因为 Nexys 4 DDR 开发板的可用引脚有限，硬件上 OV2640 摄像头和 LCD 都支持 RGB888。

JPEG 输出有助于降低数据传输所需带宽，但高档手机摄像头目前的发展趋势是不提供这个功能，由 SoC 完成 JPEG 的压缩。

- 支持自动曝光、自动增益和自动白平衡等。
对于本实验，这是很重要的功能。OV2640 的这些功能可以认为是自带的 ISP(Image Signal Processor)，有了这些功能，可以让输出的视频色彩、曝光等不需要后续处理，直接显示在 LCD 上也能有较好的效果。

ISP 的功能越来越复杂，各厂家有自己的算法以提高照片和视频质量。高档摄像头目前发展的趋势是不带 ISP 或仅提供很基本的功能。摄像头直接输出原始的图片或视频数据(Raw data)，由 SoC 提供功能更强大的 ISP 功能处理这些数据。

- 支持 UXGA、SXGA、SVGA 以及更低的分辨率
由于本实验的 LCD 只支持 854x480 分辨率，所以 SVGA(800x600)分辨率可以满足本实验的要求，在这个分辨率，OV2640 可以达到每秒 30 帧。

OV2640 的简化功能模块图如 Figure 9 所示。它包含了 1632x1232 的像素点，通过 10 bit 的 ADC 量化成数字值，由 ISP 处理成需要的格式输出。它的输出时序控制信号(HREF 和 VSYNC 等)与 VGA/HDMI 呈镜像关系。

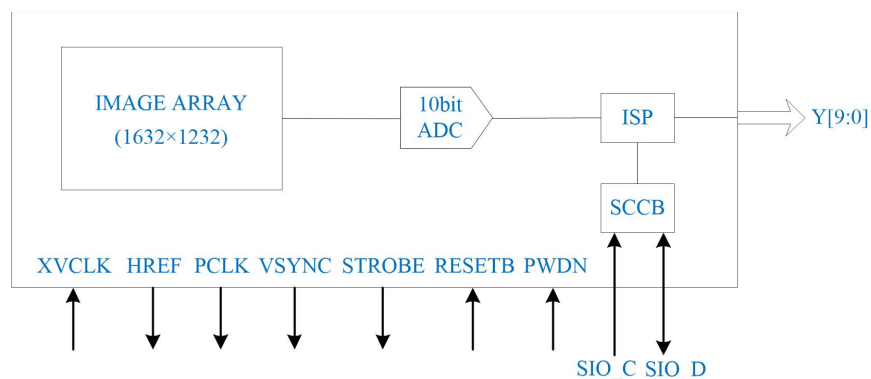


Figure 9. OV2640 功能

OV2640 主要引脚包括:

□ Y[9:0]

10 bit 数据输出, 因为片内的 ADC 为 10 bit, 所以 OV2640 提供 10 bit 高动态范围 HDR(High Dynamic Range)的输出。带有 HDR 的电视支持这个功能, 本实验的 LCD 只支持到 8 bit 动态范围(RGB888), 由于引脚个数的限制, 本实验只实现了 GRB565, 所以 Y[1:0]这两位没有使用(缺省配置), 在实验时可以将它悬空。也可以配置 OV2640 使用 Y[7:0], Y[9:8]悬空。

RGB565 每个像素点需要 16 个 bit(5+6+5), Y[9:2]端口每个时钟输出 8 个 bit, 两个时钟周期输出一个像素点。

□ XVCLK

这是摄像头系统输入时钟, 一般输入 24 MHz。这个时钟可以由 FPGA 输出一个 24 MHz 的时钟信号驱动。

□ HREF、PCLK 和 VSYNC

PCLK 是摄像头片内 PLL 生成的时钟信号, 每个时钟周期 Y[9:0]输出一组数据。PCLK 的频率与摄像头工作的分辨率和帧率匹配。通过 SCCB 配置摄像头的分辨率等模式后, PCLK 的频率会相应变化。

VSYNC 表明一帧的开始, 和 VGA/HDMI 类似。

HREF 表明数据 Y[9:0]有效，当 HREF 为高电平时，输出数据 Y[9:0]有效。如 Figure 9 所示。

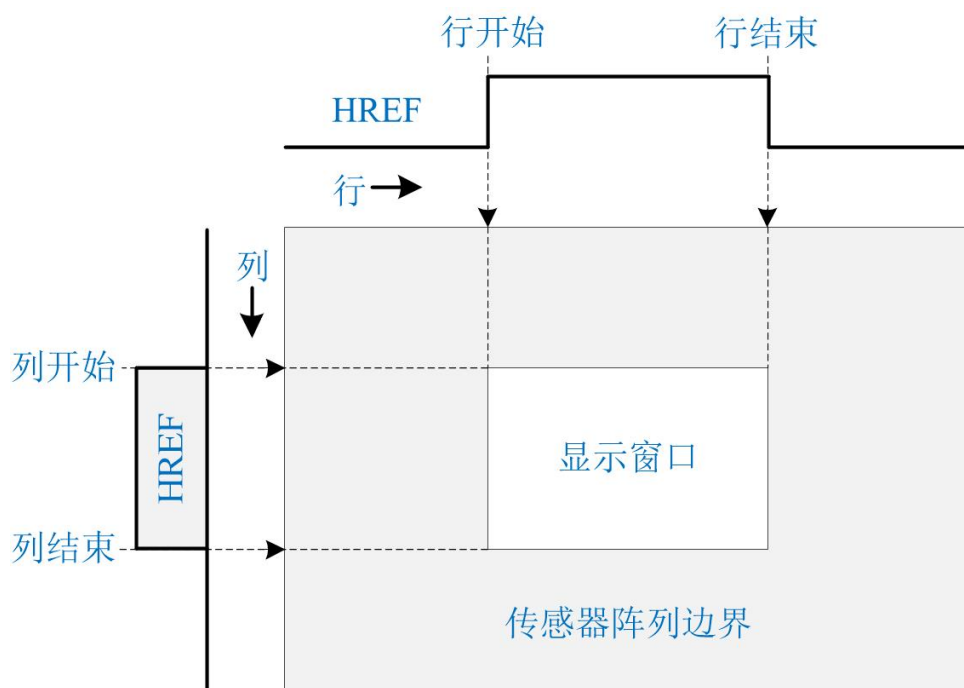


Figure 10. 显示窗口

从 Figure 10 可以看出，接收摄像头数据的时序比 HDMI 更简单一点，只要当 HREF 有效时接收 Y[9:0]即可。HREF 的上升沿表明一行的开始，下降沿表明一行的结束。

❑ RESETB

复位信号，低电平有效。OV2640 上电后，发送初始化命令之前需要先用 RESETB 复位。

❑ STROBE

这个引脚可以外接 LED 作为闪光灯或指示灯使用。也可以不接。

❑ RESETB 和 PWDN

RESETB 低电平有效。系统上电后，通过 RESETB 复位 OV2640。

PWDN 高电平有效，关闭 OV2640，降低功耗。正常使用时应接低电平。

□ SIO_C 和 SIO_D

SIO_C 是 SCCB 串行接口的时钟信号，类似 IIC 的 SCL。

SIO_D 是 SCCB 串行接口的数据信号，类似 IIC 的 SDA。

OV2640 的所有配置信息等控制命令都通过 SCCB 接口传输，在本实验里，需要通过这个接口发送 OV2640 的初始化命令。在 OV2640 工作过程中，可以通过这个接口发送命令改变它的工作模式。

SCCB 和 IIC 很相似，对本实验，可以直接用第四章的 IIC 控制方法发送初始化命令。

本节实验需要 OV2640 以 30 帧的频率输出 SVGA(800x600)分辨率 RGB565 视频。为了显示视频，只需要将接收到的 OV2640 频数据写入 LCD 的显示缓冲区。

接收 OV2640 的数据只要在 HREF 有效时用 PCLK 读取 Y[9:0]即可，每个 VSYNC 的上升沿开始新的一帧。由于 VSYNC 和 HREF 都是 PCLK 的同步信号，因此直接用 PCLK 采样这两个信号即可。为了防止杜邦线受到的干扰，也可以在 VSYNC 前加上抗抖动电路。

由 OV2640 输入的信号只有 8 bit，RGB565 分两次传输一个像素点。因此需要在电路需要将两个 8 bit 数据拼接成 RGB565，可以通过一个简单的状态机完成。由于 OV2640 输出的视频已经是 RGB565，因此可以直接发送给 LCD 用于显示。

OV2640 控制器设计包含三个部分，分别用于发送初始化命令、接收视频和 AXI 接口。

A. 初始化电路

OV2640 的初始化是通过 SCCB 端口发送命令和参数，将 OV2640 配置成输出 RGB565 格式的 800x600 视频，并设置自动曝光和自动白平衡等，OV2640 输出的视频数据不需要经过处理，直接传送给 LCD 显示。

初始化指令请参考本节提供的 `sccb.v`，OV2640 的地址是 `h60`。利用第四章的初始化方法可以完成一次性的初始化，但无法方便的发送新的指令给 OV2640。如果要实现随时发送指令，需要实现一个 AXI slave 接口的 SCCB 控制器。第六章会介绍如何实现类似协议的 AXI 接口 IIC 控制器，这样可以由 CPU 通过软件发送新指令。

初始化电路同时要控制 RESETB 和 PWDN 信号，在发送 SCCB 之前，先设置正确的 RESETB 和 PWDN 电平。

对于本实验，简单的方法是用硬件电路直接输出 RESETB、PWDN 和 SCCB 的所有波形。初始化完成后，OV2640 将稳定的输出视频，只需要用 PCLK 采样 VSYNC、HREF 和 Y[9:2]。

B. 接收显示内容

OV2640 发送的显示内容包含 Y[9:2]、VSYNC 和 HREF，这些信号由 PCLK 采样，如 Figure 11 所示。VSYNC 开始一帧数据，HREF 有效时，传输一行数据。

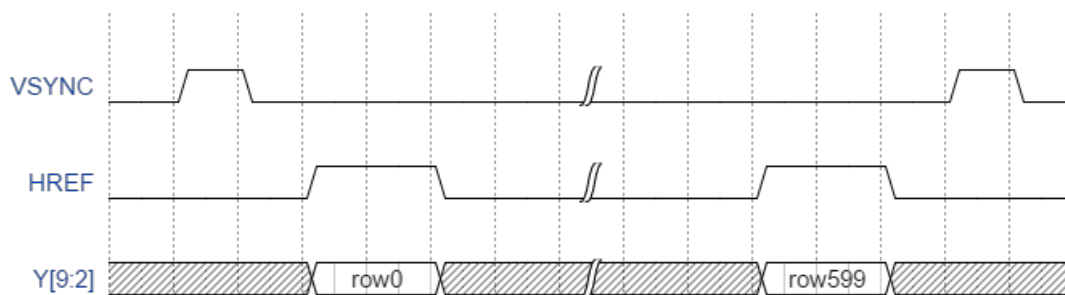


Figure 11. 视频数据传输

从 Y[9:2]得到的数据需要发送到 SoC 的 DDR 内存(LCD 的显存)，Y[9:2]是 PCLK 时钟采样，写 SoC 的 DDR 显存需要用到 AXI 时钟，这两个时钟之间没有关系，属于异步时钟。在本实验里，PCLK 频率低于 36 MHz(与分辨率/帧率有关)，而 AXI 时钟为 100 MHz 或 150 MHz，因此写 DDR 的速度快于读摄像头的速度。

为了在两个不同的时钟域之间传递数据，可以使用一个双端口的 SRAM，SRAM 的端口 A 采用摄像头的 PCLK，写入 Y[9:2] 视频数据。SRAM 的端口 B 采用 AXI 时钟，AXI 总线从端口 B 读取 SRAM 的内容，再写入 DDR 显存。双端口 SRAM 的容量为一行像素点所需要的存储空间。

C. AXI 接口设计

为了实时写入 DDR 显存，摄像头模块需要一个 AXI master 接口，由于视频信息数据量较大，因此采用 AXI full 模式。

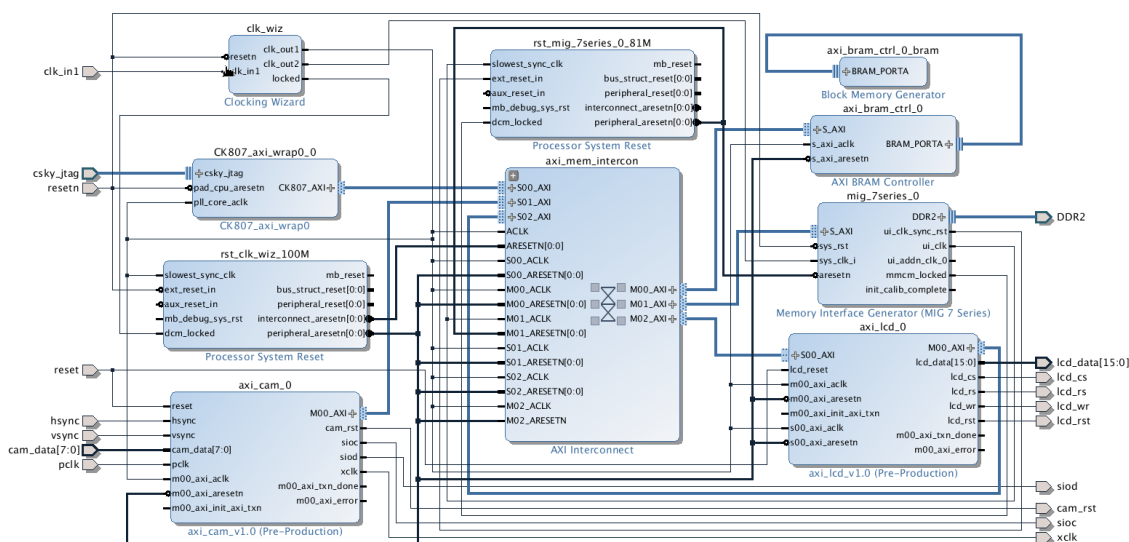


Figure 12. Camera SoC 原理图

Figure 12 是摄像头 SoC 的原理图，它包含了 OV2640 和 LCD 模块，OV2640 读取的数据通过 AXI 写入 DDR，LCD 再从 DDR 里读取显示内容，完成摄像头实时视频的显示。实验效果请参考本节视频 video 5.2 camera demo。

地址分配如下。

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
CK807_axi_wrap0_0					
CK807_AXI (32 address bits : 4G)					
axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	8K	0xC000_1FFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF
axi_lcd_0	S00_AXI	S00_AXI_reg	0x44A0_0000	512K	0x44A7_FFFF
Unmapped Slaves (1)					
axi_mem_intercon/s00_couplers/auto_pc	S_AXI	Reg			
axi_lcd_0					
M00_AXI (32 address bits : 4G)					
axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	8K	0xC000_1FFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF
axi_lcd_0	S00_AXI	S00_AXI_reg	0x44A0_0000	512K	0x44A7_FFFF
Unmapped Slaves (1)					
axi_mem_intercon/s02_couplers/auto_us	S_AXI	Reg			
axi_cam_0					
M00_AXI (32 address bits : 4G)					
axi_bram_ctrl_0	S_AXI	Mem0	0xC000_0000	8K	0xC000_1FFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF
axi_lcd_0	S00_AXI	S00_AXI_reg	0x44A0_0000	512K	0x44A7_FFFF
Unmapped Slaves (1)					
axi_mem_intercon/s01_couplers/auto_us	S_AXI	Reg			

本节实验的 LCD 和 0V2640 都采用了并行接口，占用了大量的 FPGA 引脚，Nexys 4 DDR 开发板的 5 个 Pmod 都需要使用，请参考如下引脚使用方法。


```

set_property -dict { PACKAGE_PIN E3      IOSTANDARD LVCMOS33 } [get_ports { clk }];
set_property -dict { PACKAGE_PIN J15     IOSTANDARD LVCMOS33 } [get_ports { reset }];

##Pmod Header JB

set_property -dict { PACKAGE_PIN D14     IOSTANDARD LVCMOS33 } [get_ports { lcd_data0[15] }];
set_property -dict { PACKAGE_PIN F16     IOSTANDARD LVCMOS33 } [get_ports { lcd_data0[14] }];
set_property -dict { PACKAGE_PIN G16     IOSTANDARD LVCMOS33 } [get_ports { lcd_data0[13] }];
set_property -dict { PACKAGE_PIN H14     IOSTANDARD LVCMOS33 } [get_ports { lcd_data0[12] }];
set_property -dict { PACKAGE_PIN E16     IOSTANDARD LVCMOS33 } [get_ports { lcd_data0[11] }];
set_property -dict { PACKAGE_PIN F13     IOSTANDARD LVCMOS33 } [get_ports { lcd_data0[10] }];
set_property -dict { PACKAGE_PIN G13     IOSTANDARD LVCMOS33 } [get_ports { lcd_data0[9] }];
set_property -dict { PACKAGE_PIN H16     IOSTANDARD LVCMOS33 } [get_ports { lcd_data0[8] }];

##Pmod Header JA

set_property -dict { PACKAGE_PIN C17     IOSTANDARD LVCMOS33 } [get_ports { lcd_data0[7] }];
set_property -dict { PACKAGE_PIN D18     IOSTANDARD LVCMOS33 } [get_ports { lcd_data0[6] }];
set_property -dict { PACKAGE_PIN E18     IOSTANDARD LVCMOS33 } [get_ports { lcd_data0[5] }];
set_property -dict { PACKAGE_PIN G17     IOSTANDARD LVCMOS33 } [get_ports { lcd_data0[4] }];
set_property -dict { PACKAGE_PIN D17     IOSTANDARD LVCMOS33 } [get_ports { lcd_data0[3] }];
set_property -dict { PACKAGE_PIN E17     IOSTANDARD LVCMOS33 } [get_ports { lcd_data0[2] }];
set_property -dict { PACKAGE_PIN F18     IOSTANDARD LVCMOS33 } [get_ports { lcd_data0[1] }];
set_property -dict { PACKAGE_PIN G18     IOSTANDARD LVCMOS33 } [get_ports { lcd_data0[0] }];

##Pmod Header JC

set_property -dict { PACKAGE_PIN K1      IOSTANDARD LVCMOS33 } [get_ports { lcd_rst0 }];
set_property -dict { PACKAGE_PIN F6      IOSTANDARD LVCMOS33 } [get_ports { lcd_cs0 }];
set_property -dict { PACKAGE_PIN J2      IOSTANDARD LVCMOS33 } [get_ports { lcd_rs0 }];
set_property -dict { PACKAGE_PIN G6      IOSTANDARD LVCMOS33 } [get_ports { lcd_wr0 }];
set_property -dict { PACKAGE_PIN E7      IOSTANDARD LVCMOS33 } [get_ports { cam_vsync0 }];
set_property -dict { PACKAGE_PIN J3      IOSTANDARD LVCMOS33 } [get_ports { cam_he0 }];
set_property -dict { PACKAGE_PIN J4      IOSTANDARD LVCMOS33 } [get_ports { cam_pclk0 }];
#set_property -dict { PACKAGE_PIN E6      IOSTANDARD LVCMOS33 } [get_ports { cam_xvclk }];
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets cam_pclk0_IBUF];

##Pmod Header JD

set_property -dict { PACKAGE_PIN H4      IOSTANDARD LVCMOS33 } [get_ports { cam_data0[7] }];
set_property -dict { PACKAGE_PIN H1      IOSTANDARD LVCMOS33 } [get_ports { cam_data0[6] }];
set_property -dict { PACKAGE_PIN G1      IOSTANDARD LVCMOS33 } [get_ports { cam_data0[5] }];
set_property -dict { PACKAGE_PIN G3      IOSTANDARD LVCMOS33 } [get_ports { cam_data0[4] }];
set_property -dict { PACKAGE_PIN H2      IOSTANDARD LVCMOS33 } [get_ports { cam_data0[3] }];
set_property -dict { PACKAGE_PIN G4      IOSTANDARD LVCMOS33 } [get_ports { cam_data0[2] }];
set_property -dict { PACKAGE_PIN G2      IOSTANDARD LVCMOS33 } [get_ports { cam_data0[1] }];

```

```
set_property -dict { PACKAGE_PIN F3      IOSTANDARD LVCMOS33 } [get_ports { cam_data0[0] }];

##Pmod Header JXADC

set_property -dict { PACKAGE_PIN A13      IOSTANDARD LVCMOS33      } [get_ports { soic0 }];
set_property -dict { PACKAGE_PIN A15      IOSTANDARD LVCMOS33      } [get_ports { soid0 }];
set_property -dict { PACKAGE_PIN B16      IOSTANDARD LVCMOS33      } [get_ports { rstscbb0 }];

set_property -dict {PACKAGE_PIN B18 IOSTANDARD LVCMOS33} [get_ports csky_jtag_tdi];
set_property -dict {PACKAGE_PIN A14 IOSTANDARD LVCMOS33} [get_ports csky_jtag_tdo];
set_property -dict {PACKAGE_PIN A16 IOSTANDARD LVCMOS33} [get_ports csky_jtag_tck];
set_property -dict {PACKAGE_PIN B17 IOSTANDARD LVCMOS33} [get_ports csky_jtag_tms];
set_property -dict {PACKAGE_PIN A18 IOSTANDARD LVCMOS33} [get_ports csky_jtag_trst];
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets csky_jtag_tck_IBUF];
```

大部分的 OV2640 模块采用 3.3V 单一供电，模块通过 LDO 芯片转化成 OV2640 所需的各种电压，因此可以直接利用 Nexys-4 DDR 的 Pmod 给 OV2640 模块供电，同时 OV2640 的逻辑电平与 Nexys-4 DDR 兼容，因此可以直接用杜邦线连接摄像头与 Nexys-4 DDR 开发板。

采用并行接口需要占用大量的引脚，在 PCB 上需要占用大量的布线资源。由于并行非差分接口速度较慢(一般不超过 150 MHz)，因此 LCD 和摄像头的并行接口被速度更高的差分串行接口淘汰。