开发板购买链接：

https://item.taobao.com/item.htm?spm=a1z10.1-c-s.w4004-13687301132.6.8uPVJ6&id=544830995588



Altera SOPC FrameBuffer 系统设计教程

小梅哥编写，未经授权，严禁转载或用于任何商业用途

在嵌入式系统中，LCD 屏作为最友好的人机交互方式，被大量的应用到了各个系统中。在基于 ARM 处理器的系统中，应用更是非常广泛。FPGA 作为广义嵌入式系统的一员，自然也有很多时候需要来驱动显示屏显示一些内容，例如经常有需求要用 FPGA 来做液晶测试架，做显示器驱动测试卡。很多学习了 FPGA 的朋友都知道，FPGA 驱动 VGA 显示器是比较轻松的，几乎每个板卡商提供的资料中都提供了有诸如显示彩条，显示图案，甚至显示简单的文字。但是，当希望显示更加复杂的内容的时候，往往就很难办到了。因为 FPGA 设计的是电路，是硬件，特点是头脑简单，四肢发达。而显示复杂的内容，恰恰需要一个头脑发达的控制器来执行这项任务。

在 FPGA 这些年的发展中，先后经历了简单逻辑扩展、复杂逻辑设计、SOPC 系统和 SOC 系统，逻辑设计部分就不多说了，对于 SOPC 系统设计，Xilinx 家有 MicroBlaze 软核、Altera 家有大名鼎鼎的 NIOS II 软核处理器。而到了 SOC 时代，Xilinx 家和 Altera（现已被因特尔收购，成为 Intel 的可编程事业部）都推出了内嵌双核 ARM Cortex-A9 的 SOC 芯片，Xilinx 家著名的 zynq7000 系列，以及 Altera 花了大力气在高校推广的 Cyclone V SOC。

虽然随着技术的发展以及时间的推移，SOC 的应用会越来越广泛，但是作为我们一般学习和使用来说，SOPC 技术也还是有其使用的价值的，毕竟内嵌硬核的 SOC 芯片，动辄几百块，而使用软核的方案，只要 40 元不到的 BOM 成本就能完成一个系统设计，优势还是很明显的。本例，我们就带领大家使用 Altera Cyclone IV 系列最低端的 FPGA 芯片 EP4CE6/EP4CE10 设计的能够驱动 640*480 分辨率显示屏动态显示复杂内容的系统。例如显示文字，显示图片

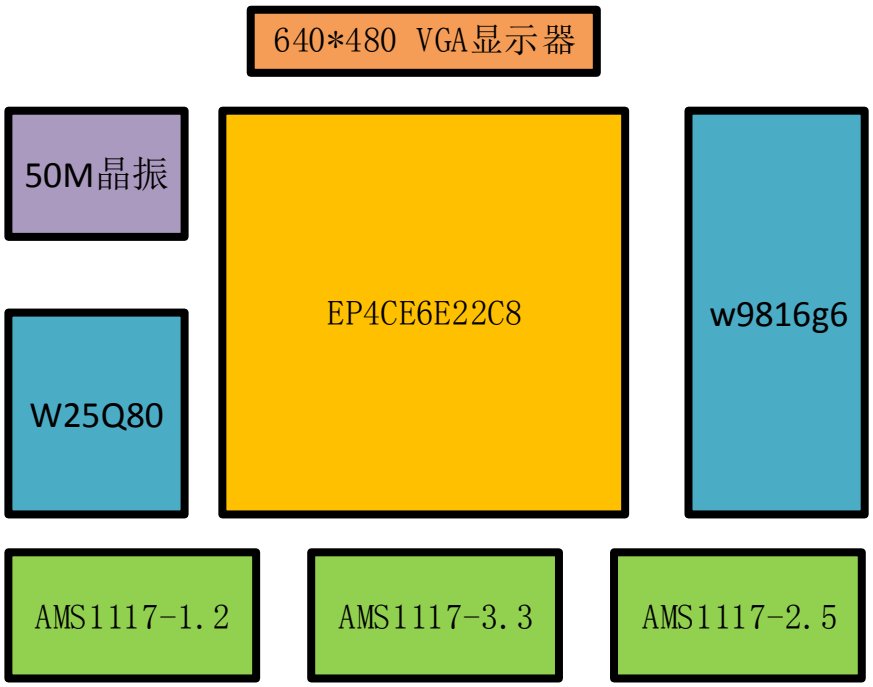
在之前我们讲解过 RGB 显示屏的驱动，RGB 显示屏驱动时序和 VGA 驱动时序几乎完全一样，只是不同的分辨率之间，其时序参数不同。说到这个驱动 480*272 分辨率显示屏动态显示复杂内容，相信很多朋友并不陌生，是的，在我们之前的 SOPC 公开课时候，就已经给大家讲解过实现这种系统的一种方式，即使用了一片独立的 SRAM 来作为显示屏的显存，显示驱动模块直接读取 SRAM 中的数据并实时刷新显示，而 NIOS II CPU 则只在需要刷新显示内容的时候，才执行对 SRAM 的写入操作。此种方式，需要一片独立的 SRAM 作为显存，以及一片 SDRAM 作为 NIOS II 的运行内存。除了 SRAM 的使用会增加系统的 BOM 成本外，SRAM 也会占用掉 FPGA 芯片较多的管脚，导致必须得使用 256 脚 BGA 方案的芯片才能够实现。因此，此种方案仅适合在对成本不太敏感高的场合使用。本节要介绍的方案，是对该系统改进优化得到的。

本节所描述的系统，仅需用到以下硬件资源

1. EP4CE6/EP4CE10 型 FPGA 芯片一片（30 元）
2. 16Mbit 或以上 SDRAM 芯片一片（3 元，如华邦 w9816g6）
3. 50M 有源晶振一片（3 元）
4. 4Mbit SPI FLASH 芯片（1.5 元，如 W25Q80）
5. 3.3V、2.5V、1.2V LDO 稳压电源（AMS117 系列，合计不到 1 元）

本节希望达到的目的：

在显示屏上显示一帧图像，并在指定位置显示字符内容



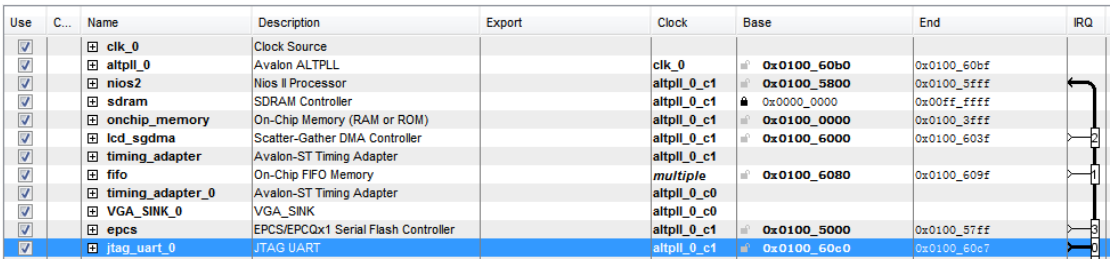
上述框架是我们经过分析，要实现本系统方案所需的最小硬件系统，本节我们将基于芯航线的 AC620 学习套件来完成 FPGA 的系统搭建和 NIOS II 软件设计。芯航线的 AC620 FPGA 开发板整体硬件配置高于上述分析，有助于我们进行原型验证。

FPGA 系统搭建

本系统主要在 Qsys 系统中完成，包含了以下 IP 核：

- clk_0：输入时钟管理单元，系统默认添加
- altpll_0：PLL 锁相环，对输入时钟进行分频和倍频，以得到两路频率为 100MHz，相位相差 180 度的时钟信号，分别提供给系统中逻辑电路工作（包含 NIOS II 软核处理器）和 SDRAM 芯片使用。以及 1 路 24M 的时钟信号，供 640*480 分辨率的 VGA 驱动电路使用。
- NIOS II CPU：实现系统的控制以及显示内容的处理。
- SDRAM：NIOS II CPU 运行内存和 TFT 显示图像帧缓存。
- onchip_memory：片上存储器，指定 SGDMA 要执行的数据传输，主要用作 SGDMA 的描述符存储器。
- lcd_sgdma：SGDMA IP，主要实现大量数据的高效搬运，支持流模式，效率比 Avalon MM 接口的 DMA 核高。
- timing_adapter：时序匹配 IP 核，主要用于流数据在两个不同模块间进行传递时，对部分信号加上一定延迟，使得数据和标志信号能够完全同步（说的简单点儿，就是用寄存器打拍的方式对一些信号进行延迟，使数据和控制信号对齐）。
- fifo：双时钟 fifo，主要完成数据流的传输速度转换。SGDMA 输出的数据流，是按照和存储器相同的时钟速度（本例中为 100M）进行的，而在数据使用端，即 VGA 显示部分，是按照 9M 的时钟取用数据的，因此使用双时钟 fifo，解决跨时钟域的问题。
- VGA_SINK：该 IP 为第三方开源 IP，由友晶科技提供，主要完成 Avalon ST 流数据转换成 VGA 行场时序并驱动 VGA 屏进行显示。

- EPCS：EPCS FLASH 存储器控制 IP，使用该 IP，使得 EPCS 芯片能够存储 FPGA 固件和 NIOS II 的程序。
- jtag_uart0：调试串口，主要用来打印一些调试信息，在设计的前期调试中很有用



Use	C...	Name	Description	Export	Clock	Base	End	IRQ
☑		clk_0	Clock Source					
☑		altpll_0	Avalon ALTPLL		clk_0	0x0100_60b0	0x0100_60bf	
☑		nios2	Nios II Processor		altpll_0_c1	0x0100_5800	0x0100_5fff	
☑		sdram	SDRAM Controller		altpll_0_c1	0x0000_0000	0x00ff_ffff	
☑		onchip_memory	On-Chip Memory (RAM or ROM)		altpll_0_c1	0x0100_0000	0x0100_3fff	
☑		lcd_sgdma	Scatter-Gather DMA Controller		altpll_0_c1	0x0100_6000	0x0100_603f	2
☑		timing_adapter	Avalon-ST Timing Adapter		altpll_0_c1			
☑		fifo	On-Chip FIFO Memory		multiple	0x0100_6080	0x0100_609f	1
☑		timing_adapter_0	Avalon-ST Timing Adapter		altpll_0_c0			
☑		VGA_SINK_0	VGA_SINK		altpll_0_c0			
☑		epcs	EPCS/EPCQx1 Serial Flash Controller		altpll_0_c1	0x0100_5000	0x0100_57ff	3
☑		jtag_uart_0	JTAG UART		altpll_0_c1	0x0100_60c0	0x0100_60c7	0

接下来，将详细讲解本系统的搭建过程。

1、 altpll_0

为了保证整个系统能够具有较高的性能，因此让其运行在 100MHz，所以需要使用 PLL 对外部晶振产生的 50MHz 输入的时钟信号进行倍频，得到两路频率为 100M，相位相差 90 度的时钟信号。另外，还需要提供一路 9MHz 时钟信号供 480*272 分辨率的 TFT 屏驱动使用。关于 PLL 的详细添加和参数修改方式这里不再多说，仅提供设置结果：

inclk0：50M

不需要 areset 信号和 locked 信号

c0：9MHz

c1：100MHz，相位为 0

c2：100MHz，相位为-180 度

2、 nios ii cpu

所有设置使用默认值，等后续添加完 sdram 和 EPCS 和，将复位向量（Reset Vector）设置为 epcs，将异常向量（Exception Vector）设置为 sdram。

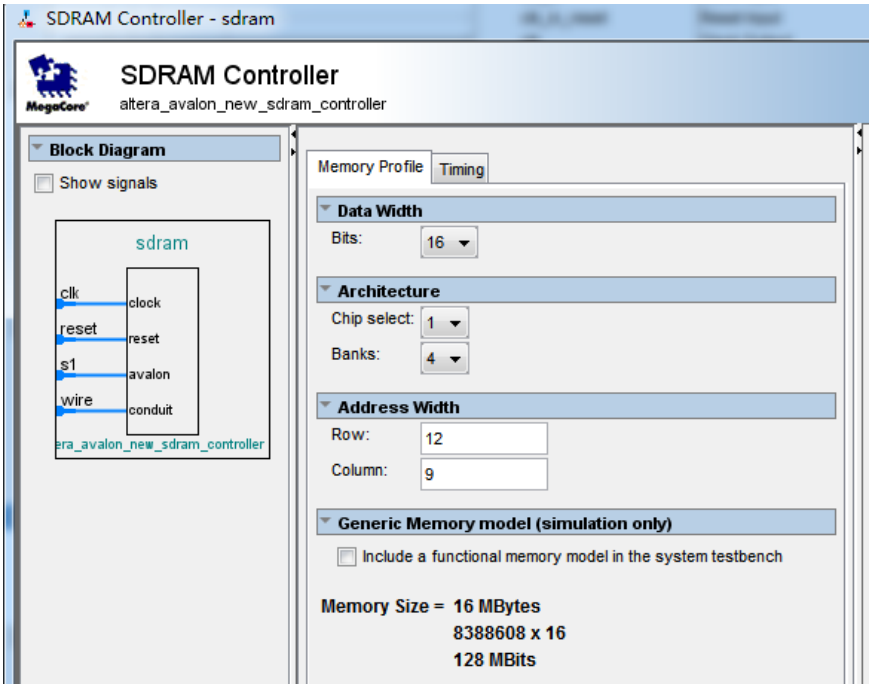
3、 sdram

数据位宽（Data Width）为 16bit。

结构（Architecture）为一个片选，4 个 bank。

地址宽度（Address Width）Row 地址为 12、Column 地址为 9。

其他默认即可。



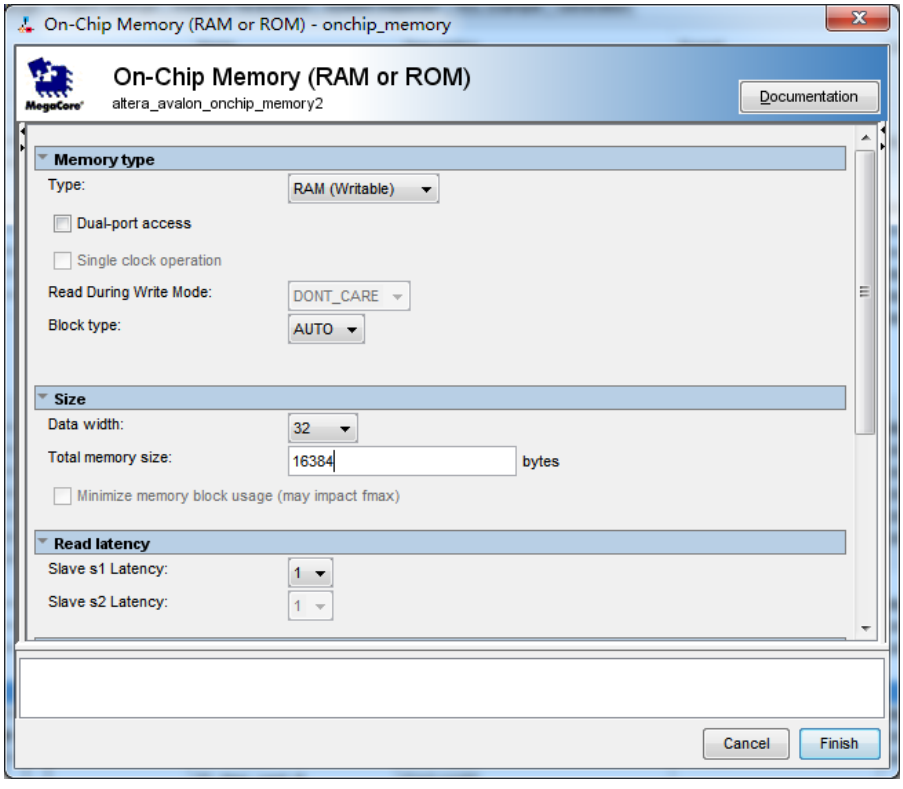
4、 onchip_memory

片上缓存 onchip_memory 作为 SGDMA 的描述符存储器，存储 SGDMA 的数据传输描述内容。

类型为 RAM；

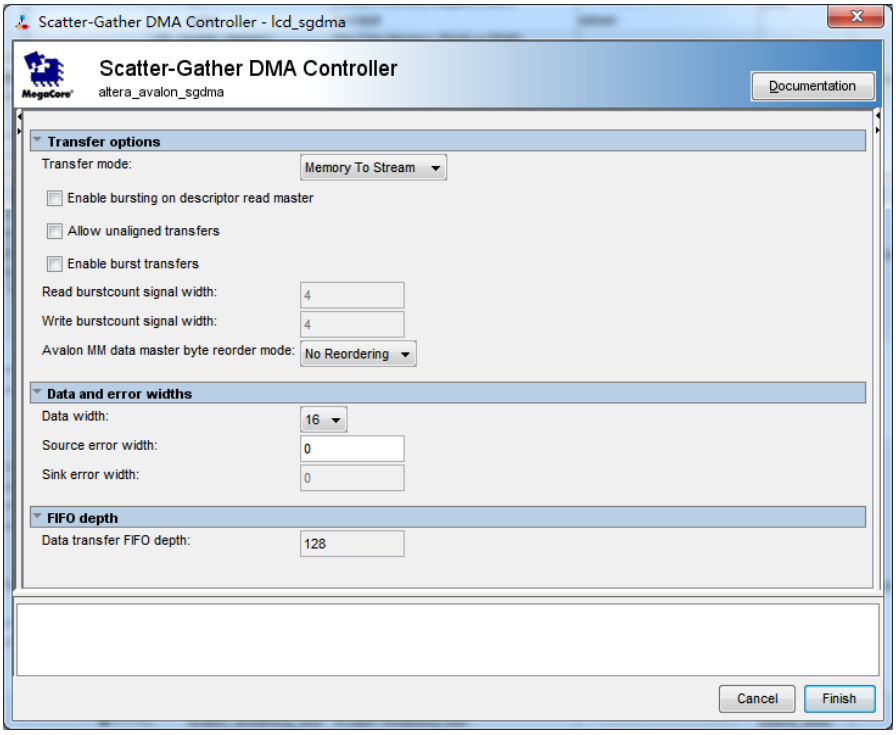
数据位宽选择 32 位；

存储器总量选择 16384Bytes（用户可以根据自己芯片的 RAM 容量适当增加或减小该值）；



5、 lcd_sgdma

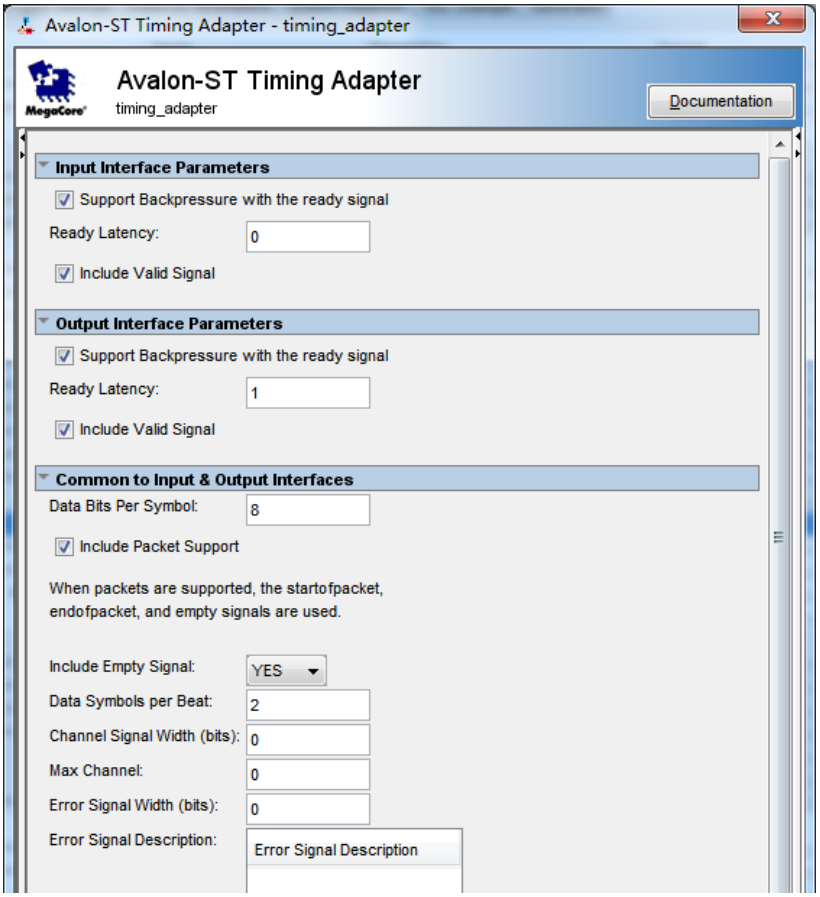
选择传输模式为存储器到数据流模式（Memory To Streram）；
数据宽度（Data width）为 16bit，该位宽与存储器（SDRAM）的 Avalon MM 总线宽度一致。
其他按照默认即可，如下图所示。



6、 timing_adapter

时序匹配 IP 核，主要用于流数据在两个不同模块间进行传递时，对部分信号加上一定延迟，使得数据和标志信号能够完全同步（说的简单点儿，就是用寄存器打拍的方式对一些信号进行延迟，使数据和控制信号对齐）。

如下图所示设置，所有勾选项都勾选上，输入 Ready Latency 设置为 0，输出 Ready Latency 设置为 1，每个数据总共有两个 symbols，每个 symbol 含 8 位数据。（根据 RGB 数据流的实际意义，这里应该是每个数据有 3 个 symbols，但是本设计使用的是 16 位色 RGB565 模式，即 RGB 总共才 16 位，所以在搭建系统的时候，假设只有 2 个 symbols，只是在最终数据输出到 VGA 的时候，将 16bit 数据拆分成 RGB565，分别送给 VGA 的 R、G、B 色通道）



以上各个模块，都是工作在 100MHz 的时钟频率下。主要完成将需要显示的数据从 SDRAM 中读取出来。而读取出来的数据最终是需要送到 TFT 显示屏上去显示的，TFT 显示部分，对于本系统，数据传输速率为 9M，因此就需要实现数据从 100M 时钟域到 9M 时钟域的跨时钟域传输。而实现数据流跨时钟域传输，较好的方式是使用双时钟 fifo。因此，接下来就需要添加一个双时钟 fifo，先将 100M 时钟域的数据缓存起来，然后再等待 9M 时钟域的读取逻辑来读取。

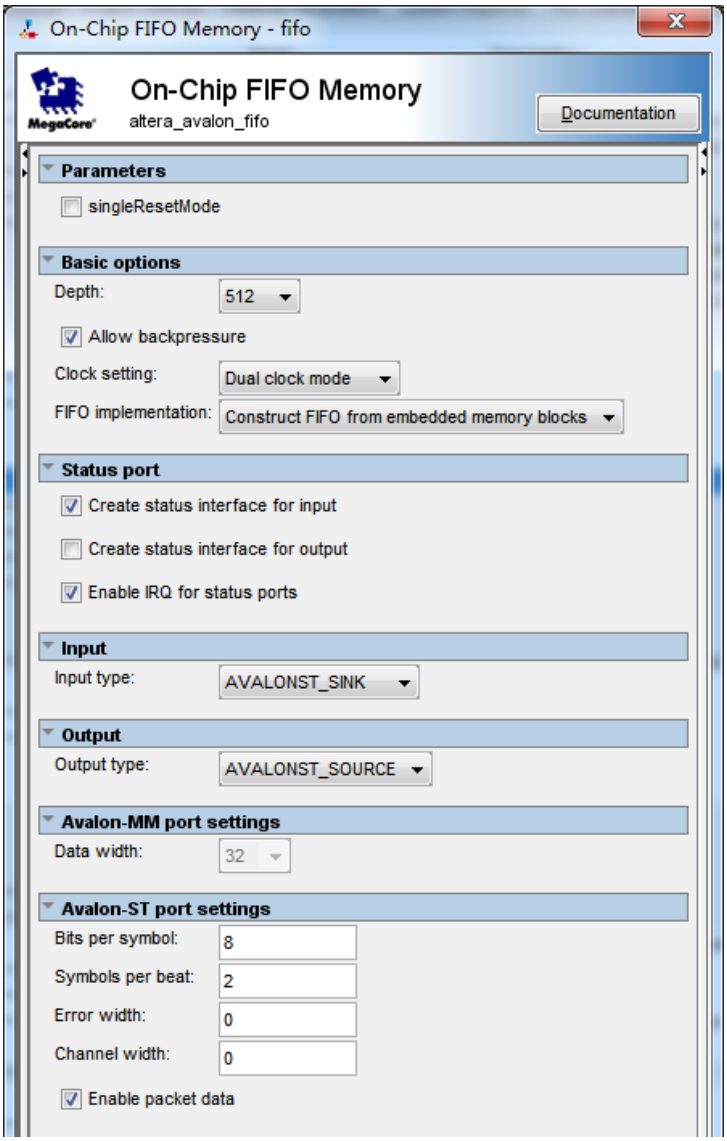
7、 fifo

设置 fifo 的深度为 512 字节，每个数据含 2 个 symbols，每个 symbols 由 8 位数据组成，即整个数据宽度为 16 位。时钟设置为双时钟模式（Dual Clock Mode），输入和输出都使用 Avalon ST 接口，使能数据包（Enable Packet Data）。这里顺带说一下，Cyclone IV E 器件中的嵌入式块 RAM 为 M9K 存储器，每个器件里都有若干个 M9K 存储器。所谓 M9K 存储器，就是一个块存储器中有 9Kb 个存储位，每个 M9K 存储器可以被配置为以下模式：

- 8192 × 1
- 4096 × 2
- 2048 × 4
- 1024 × 8
- 1024 × 9
- 512 × 16
- 512 × 18

- 256 × 32
- 256 × 36

由于本设计中，数据是 16 位的，因此，如果要想仅使用一个 M9K 存储器就实现本 Fifo，存储深度可在 1 到 512 之间任选。假如我们选择存储深度为 128，那么就只占用 1/4 个 M9K 的存储容量，但是，剩下的 3/4 的存储容量也依旧无法单独使用，在该设计中会永远的被浪费掉，因此，不如将存储深度直接设置为 512，还能保证缓存足够大，避免意外丢失数据。

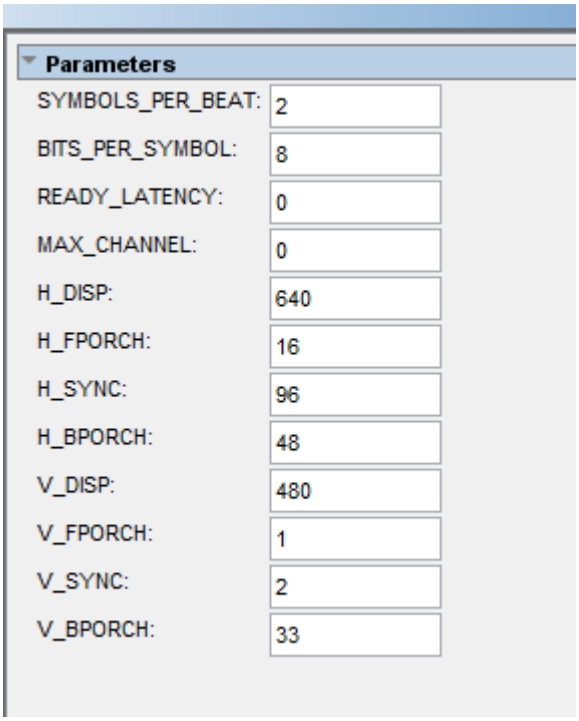


8、 VGA_SINK

VGA_SINK 模块是从友晶提供的设计中修改得到的，该 IP 原本仅支持 24 位色模式，而本系统，由于 SDRAM 为 16 位宽度，如果强行使用 24 位色模式，则必须得从 SDRAM 中读取两个数据才能拼接成一个像素，导致 SDRAM 的负载过大，从而使得系统带宽遇到瓶颈，无法正常工作，因此对该 IP 进行了修改，即降低为 16bit 的 RGB565 模式。（根据 RGB 数据流的实际意义，这里应该是每个数据有 3 个 symbols，但是本设计使用的是 16 位色 RGB565 模式，即 RGB 总共才 16 位，所以在搭建系统的时候，假设只有 2 个 symbols，只是在 VGA_SINK 模块内部，在最终数据输出到 VGA 的时候，进行了修改，将 16bit 数据拆分成 RGB565，分别送给 VGA 的 R、G、B 色通道）。这里参数分别如下设置：

H_DISP（行显示有效像素）：640
H_FPORCH：16 时钟周期
H_SYNC：96 时钟周期
H_BPROCH：48 时钟周期

V_DISP（行显示有效像素）：480
V_FPORCH：1 时钟周期
V_SYNC：2 时钟周期
V_BPROCH：33 时钟周期



这些参数都是从 VGA 标准中查到的。

9、EPCS

EPCS 的添加没有什么需要特别注意的，一切按照默认设置即可。

至此，我们已经完成了整个显示系统的组件添加工作，接下来就该进行组件间的总线连接了。

在总线连接之前，必须要先给大家讲解一下本系统的数据流向，因为本系统和我们之前在 SOPC 公开课时候讲过的系统架构还不太一样，本系统主要引入了 Avalon ST 总线。在之前公开课时候讲过的各种例子，都是基于 Avalon MM 总线的，所有外设 IP 都通过 Avalon MM 总线连接到 NIOS II CPU 上，因此总线连接是非常简单的。但是 Avalon MM 总线的数据搬运能力是比较弱的，无法支持高速大量的数据传输。而在基于 Avalon ST 总线的数据流中，NIOS II CPU 实质相当于被旁路了，只起到了状态管理的作用，所有的数据流转都是模块间通过 Avalon ST 总线进行交互的，不需要 NIOS II 去执行搬运工作，因此数据的搬运效率大大提高。

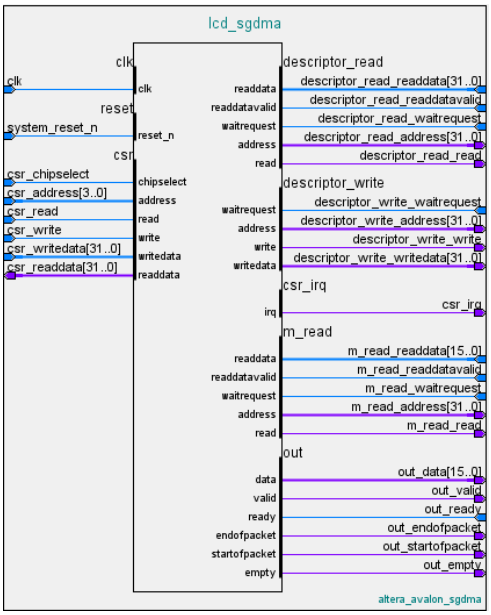
本节暂不对 Avalon ST 总线进行过多的讲解，总之，我们可以把 Avalon ST 总线想象成一

个水管，水从水管的一头可以很容易且快速的流到另一头，不需要第三方用水瓢一瓢一瓢的去舀过去。

Framebuffer 系统数据流

在整个 FrameBuffer 系统中，SGDMA 模块是最核心的部分，他实现了对 SDRAM 存储器的直接读取操作，并将读到的数据最终以数据流的形式输出。SGDMA 的使用相当的广泛，不仅是在本系统中，在 Altera 很多官方设计中也都使用了 SGDMA IP 核，比如 PCIE 示例、千兆以太网示例、VIP 示例等，关于 SGDMA 的详细介绍，可以参看 SGDMA 的用户手册，我们也希望能尽快推出 SGDMA 的专题讲解。这里仅对 SGDMA 的端口和功能进行下简述。

本系统中 SGDMA 被配置为存储映射模式到数据流模式，即数据的源头采用地址映射的方式组织，而数据的接收方则使用数据流的方式进行接收。在此种模式下，SGDMA 总共有 5 个端口，分别为 Avalon MM Slave 接口的控制总线，Avalon MM Master 接口源数据读取总线、Avalon MM Master 接口的描述符写入总线和 Avalon MM Master 接口的描述符读取总线，以及最终的 Avalon ST Source 数据输出总线。



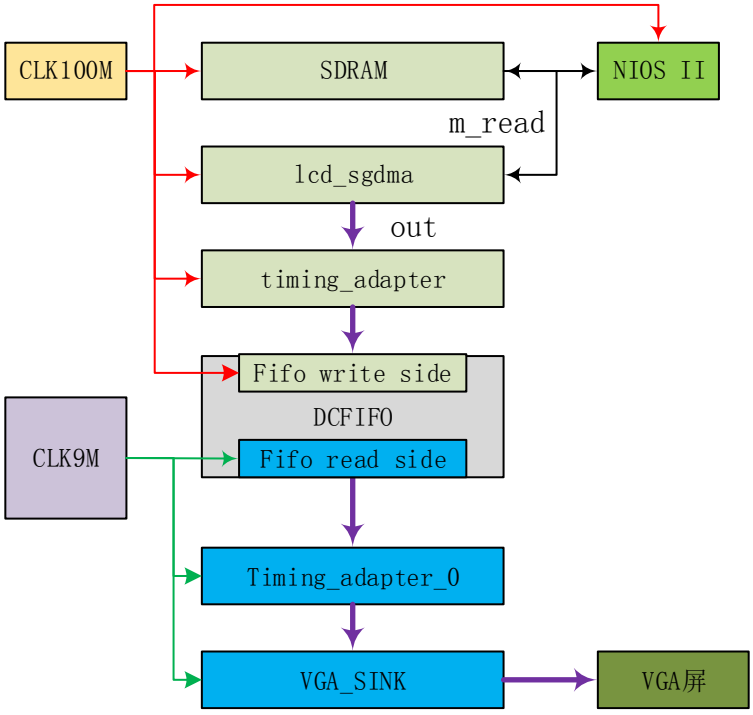
csr：Avalon MM Slave 接口的控制总线，与 NIOS II CPU 进行连接，NIOS II CPU 通过该接口读写 SGDMA 内部的控制和状态寄存器，以实现 DMA 的传输控制。这也是整个数据流传输唯一需要 NIOS II 参与的地方。NIOS II 实际就扮演了个老板的角色，安排 SGDMA 去做什么，SGDMA 接到任务后就下去干搬运的苦力活儿了，而 NIOS II 这个老板，则可以坐在办公室喝喝咖啡，看看报纸，等着 SGDMA 把苦力干完后回来报告（中断）或者隔一段时间打个电话问问工作状态和进度（查询）。

m_read：Avalon MM Master 接口的源数据读取总线，该接口实现对 Avalon MM 总线上挂载的存储需要传输的数据源存储器的读取，例如 SDRAM、DDR2 SDRAM。从这里，大家也可以看到，在 Qsys 搭建的系统中，不是只有 NIOS II 能够去读取 SDRAM、DDR2 等存储器，实际上，只要是 Avalon MM Master 接口的 IP，都可以去读取这些存储器，我们甚至可以自己编写一个 Avalon MM Master 接口的逻辑，来取代 NIOS II CPU 完成各种 IP 核的读写操作。

descriptor_write：描述符写端口，SGDMA 实现数据传输需要有一个描述符，SGDMA 的所有传输都是受描述符控制的，通过描述符存储 SGDMA 的实时传输状态。这里专门使用一个片上存储器来存储 SGDMA 的描述符，能够节省 FPGA 资源。否则，如果使用 FPGA 资源来实现描述符，将会带来非常大的资源消耗。

descriptor_read：描述符读端口，SGDMA 使用此端口读取描述符，以获得传输信息。
out：数据流输出端口，从 SDRAM 或 DDR2 中读取到的数据会经由此端口流出，以提供给数据的使用端使用。

对于我们分析数据流来说，可以暂时忽略 csr、descriptor_write 和 descriptor_read 端口，因为这些端口并不处于数据通路上，真正处于数据通路上的，是 m_read（数据流入）和 out（数据流出）端口。下图为本系统的数据流图。



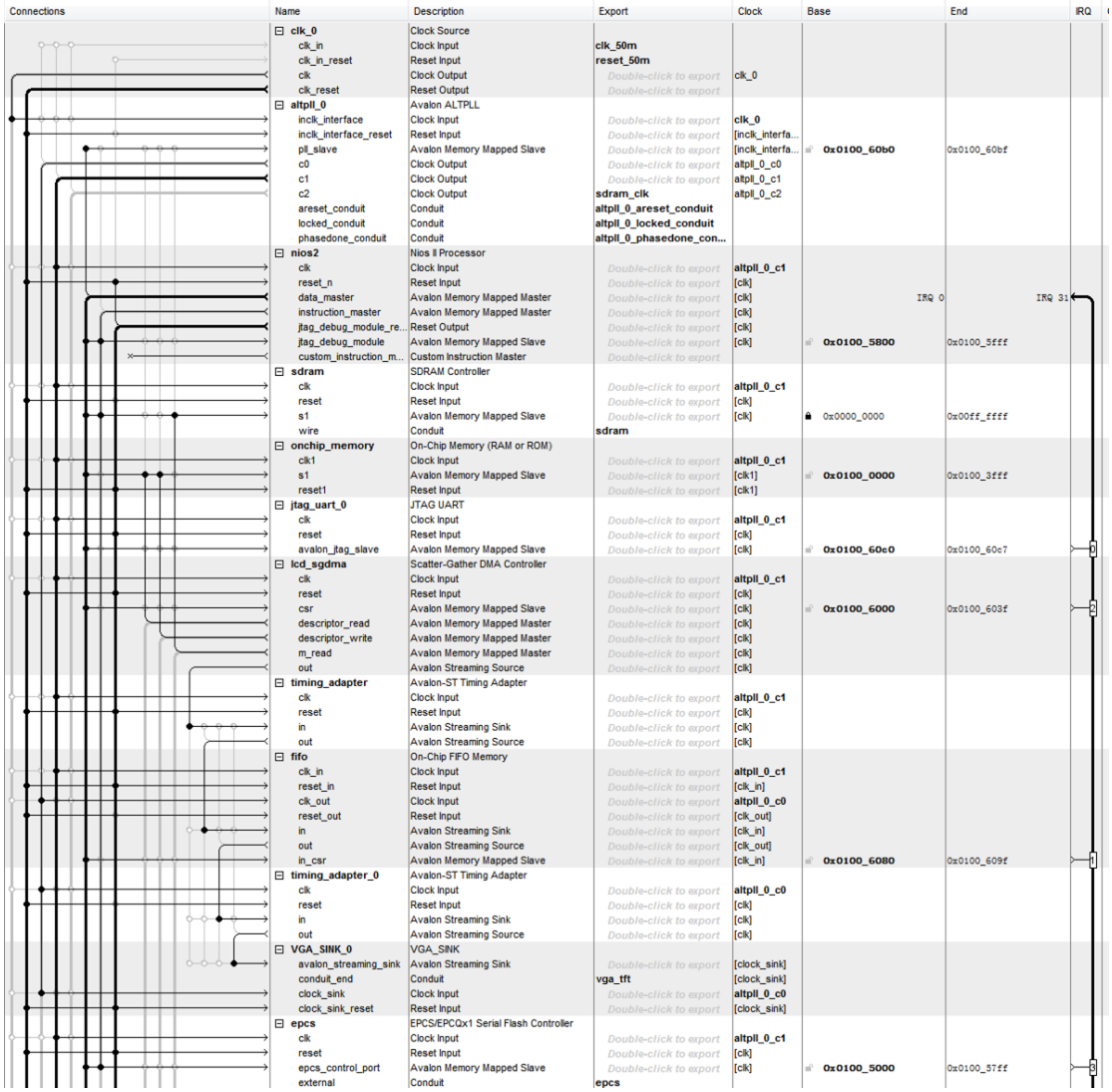
NIOS II CPU 使用 SDRAM 作为运行存储器，存储程序和数据，同时，SGDMA 从 SDRAM 中实时读取需要显示到 VGA 屏上的数据。NIOS II 和 SGDMA 之间对 SDRAM 的使用仲裁由 Avalon MM 总线的仲裁机制自动处理。

通过以上介绍，了解了系统的数据流向，就可以首先完成数据流总线的连接了。每个 st 流接口的模块都有 2 个流端口，一个是流输入端口（in: Avalon Streaming Sink），另一个是流输出端口（out: Avalon Streaming Source）。连接时，只需将上一级模块的流 out 连接到下一级模块的流 in 端口，即可实现数据流的自动衔接。在本例中每个模块端口连接如下表所示：

sdram	sdram.s1	nios2.data_master; nios2.instruction_master;	clk100m
lcd_sgdma	lcd_sgdma.csr	nios2.data_master;	
	lcd_sgdma.des_read	onchip_memory.s1	
	lcd_sgdma.des_write	onchip_memory.s1	
	lcd_sgdma.m_read	sdram.s1	
	lcd_sgdma.out	timing_adapter.in	
timing_adapter	timing_adapter.out	fifo.in	
fifo	fifo.in_csr	nios2.data_master;	
fifo	fifo.out	timing_adapter_0.in	clk9m
timing_adapter_0	timing_adapter_0.out	VGA_SINK_0.avalon_streaming_sink	

VGA_SINK_0	VGA_SINK_0.conduit_end	导出，连接到 VGA 驱动芯片引脚	

整体连接完成后，结果如下图所示：



当然，连接完成总线后，不要忘了连接复位网络、导出需要引出到系统顶层的信号以及中断网络。

- 修改 NIOS II 的复位向量为 EPCS、异常向量为 SDRAM。
- 自动分配基地址。
- 到 HDL Eaxmple 栏复制例化模版。
- 保存并开始 generate。

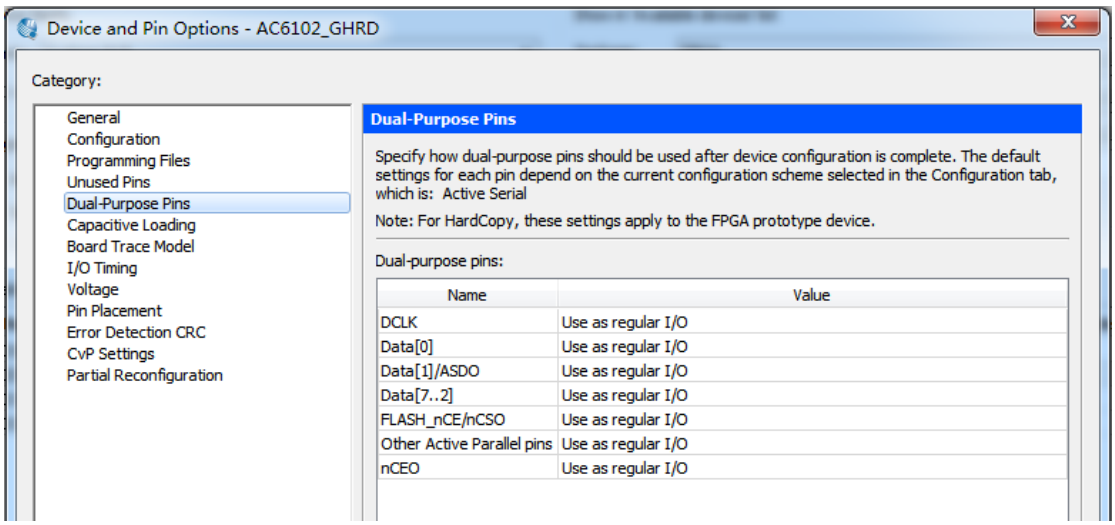
至此，整个系统就搭建完成了，下一步就是将系统加入到 Quartus II 工程中。首先是将 mysystem.qsys 文件添加到工程，然后完成工程顶层文件的编写。这些最基本的操作，这里就不在赘述，以下为工程顶层代码：

```
module Framebuffer VGA(
       input  wire        clk 50m,
       input  wire        reset n,
       output wire        vga clk,
       output wire        vga de,
```

```
        output wire [7:0]  vga r,
        output wire [7:0]  vga g,
        output wire [7:0]  vga b,
        output wire        vga hs,
        output wire        vga vs,
        output wire        vga bl,
        output wire        sdram clk,
        output wire [11:0] sdram addr,
        output wire [1:0]  sdram ba,
        output wire        sdram cas n,
        output wire        sdram cke,
        output wire        sdram cs n,
        inout  wire [15:0] sdram dq,
        output wire [1:0]  sdram dqm,
        output wire        sdram ras n,
        output wire        sdram we n,
        output wire        epcs dclk,
        output wire        epcs sce,
        output wire        epcs sdo,
        input  wire        epcs data0
);

    assign vga bl = 1;

    wire vga clk r;

    assign vga clk = ~vga clk r;    //对 VGA 时钟反相，以保证数据中心对齐

    mysystem u0 (
        .clk 50m clk                    (clk 50m),
        .reset 50m reset n               (reset n),
        .vga clk                        (vga clk r),
        .vga de                         (vga de),
        .vga r                          (vga r),
        .vga g                          (vga g),
        .vga b                          (vga b),
        .vga hs                         (vga hs),
        .vga vs                         (vga vs),
        .altpll 0 phasedone conduit export (),
        .altpll 0 locked conduit export    (),
        .altpll 0 areset conduit export    (),
        .epcs dclk                      (epcs dclk),
        .epcs sce                       (epcs sce),
        .epcs sdo                       (epcs sdo),
        .epcs data0                     (epcs data0),
        .sdram clk clk                   (sdram clk),
        .sdram addr                     (sdram addr),
        .sdram ba                       (sdram ba),
        .sdram cas n                    (sdram cas n),
        .sdram cke                      (sdram cke),
        .sdram cs n                     (sdram cs n),
        .sdram dq                       (sdram dq),
        .sdram dqm                      (sdram dqm),
        .sdram ras n                    (sdram ras n),
        .sdram we n                     (sdram we n)
    );

endmodule
```

然后分配引脚，本系统在 AC620 开发板上的引脚分配请参照 AC620 开发板引脚分配表。

分配引脚后，记得在 quartus ii 软件中设置所有双功能引脚为用户模式。

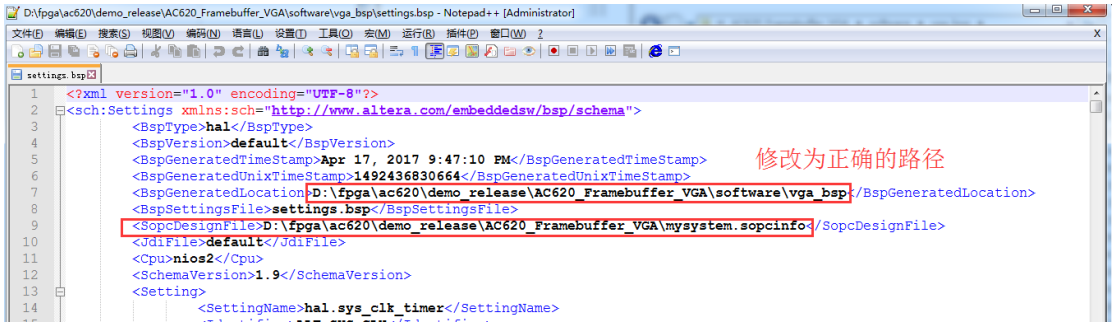


编译工程得到 sof 编程文件。

至此，整个 FrameBuffer 系统硬件设计就完成了，下一步，我们就可以编写相应的驱动程序和应用程序，在目标板上运行了。

创建 Eclipse 工程

打开 NIOS II EDS 软件，导入我们提供的 VGA 和 VGA_bsp 两个工程。修改 setting.bsp 文件中第 7 行和第 9 行两个位置为自己电脑上对应的位置。关闭文件，然后 build 工程。如果编译失败，提示无权限，请先关闭所有文件，然后选择整个文件夹右键获取管理员权限。



以下为 main 函数内容

```
#include "stdio.h"
#include "stdlib.h"
#include "io.h"
#include "sys/alt_alarm.h"
#include "altera_avalon_sgdma.h"
#include "altera_avalon_sgdma_descriptor.h"
#include "altera_avalon_sgdma_regs.h"
#include "alt_types.h"
#include "alt_video_display.h"
#include "unistd.h"
#include "pic1.h"
#include "pic2.h"
```

```
#include  "system.h"


#define  WIDTH 640
#define  HEIGHT 480
#define  NUM_FRAME 1


int main() {
    unsigned int d = 0;

    ////Initial LCD Display
    alt_video_display* display_global;
//  printf("Initializing LCD display controller\n  ");
    display_global = alt_video_display_init(LCD_SGDMA_NAME, // Name
of video controller
           WIDTH, // Width of display
           HEIGHT, // Height of display
           16, // Color depth (32 or 16)
           SDRAM_BASE + SDRAM_SPAN / 2, // Where we want our frame
buffers
           ONCHIP_MEMORY_BASE, // Where we want our descriptors
           NUM_FRAME);
//  if (display_global)
//      printf(" - LCD Initialization OK\n");
//  else
//      printf(" - LCD FAILED\n");
    alt_video_display_clear_screen(display_global, 0xff);
    show_pic(display_global, pic1);
    usleep(1000000);
    while(1){
    }
}
```

编译完成后下载 sof，并在 nios ii eds 中执行 run。连接 VGA 显示器，就能在屏幕上显示一副美女图，如下图所示：

如有其它问题，请参见小梅哥博客中关于 NIOS II 的相关内容。如“【2017 最新】解决 NIOS II 工程移动在磁盘上位置后 project 无法编译问题.pdf”和“*NIOS II 开发注意点总结.pdf*”

本文档对应设计工程名为：AC620_Framebuffer_VGA，请到 AC620 开发板网盘/光盘资料中获

取。



