杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

电子信息技术虚拟仿真实验报告

基于 Nios II 实现多类型 LCD 屏幕彩条显示

学院 卓越学院

学号 23040447

姓名 陈文轩

专业 智能硬件与系统 (电子信息工程)

指导教师 徐魁文、吴岩

2025 年 5 月 13 日

摘　要

本实验通过 Qsys 系统集成工具，利用 Altera 提供的 IP 核，搭建了一个简单的 SOPC 系统，并设计了 IP 核之间的互联逻辑。实验中将 Nios II 嵌入式软核处理器部署在 EP4CE10F17C8 正点原子开发板上，结合 SDRAM 、LCD 屏幕等硬件模块与 Avalon-MM 突发传输控制、SDRAM 控制器、SDRAM 桥控制器等软件模块，完成了软硬件协同设计。最后通过嵌入式 C 程序的开发，实验实现了兼容驱动多种类型 LCD 屏幕的功能，并能够显示指定颜色和宽度的彩条。

关键词：**LCD 屏幕，Qsys 开发，NiosII 软核**

1 引言

当今世界，电子信息技术以极快的速度在发展，各种各样的电子产品在我们的日常生活中都变得必不可少。比如手机、电脑等等。这些产品的设计开发，当然少不了最基础的 LCD 显示器。LCD 显示器作为人机交互的关键窗口，其驱动与控制技术的实现至关重要。随着可编程逻辑器件（FPGA）技术的成熟，其并行处理能力、高集成度和可重构性使其在图像处理和显示控制领域展现出巨大潜力。特别是基于 FPGA 的片上可编程系统（SOPC）技术，允许设计者将处理器核、存储器接口、外设接口等集成在单一芯片上，为复杂的嵌入式显示系统提供了高效灵活的解决方案。

在此背景下，本实验旨在探索和实践一种基于 SOPC 的 LCD 显示控制系统设计方法。实验通过 Altera 公司的 Qsys 系统集成工具，利用其提供的 IP 核，在 EP4CE10F17C8 FPGA 开发板上搭建了一个包含 Nios II 嵌入式软核处理器的 SOPC 系统。该系统整合了 SDRAM、LCD 屏幕等关键硬件模块，并利用 Avalon-MM 突发传输控制、SDRAM 控制器及桥控制器等软件模块，实现了高效的软硬件协同工作。通过开发嵌入式 C 程序，本实验成功实现了对多种类型 LCD 屏幕的兼容驱动，并能在屏幕上显示指定颜色和宽度的彩条。这一实践不仅完整展示了基于 FPGA 的嵌入式系统设计全流程，也为软硬件协同开发提供了宝贵的实践经验，为未来构建更为复杂的嵌入式系统奠定了坚实基础。

2　系统总体硬件设计

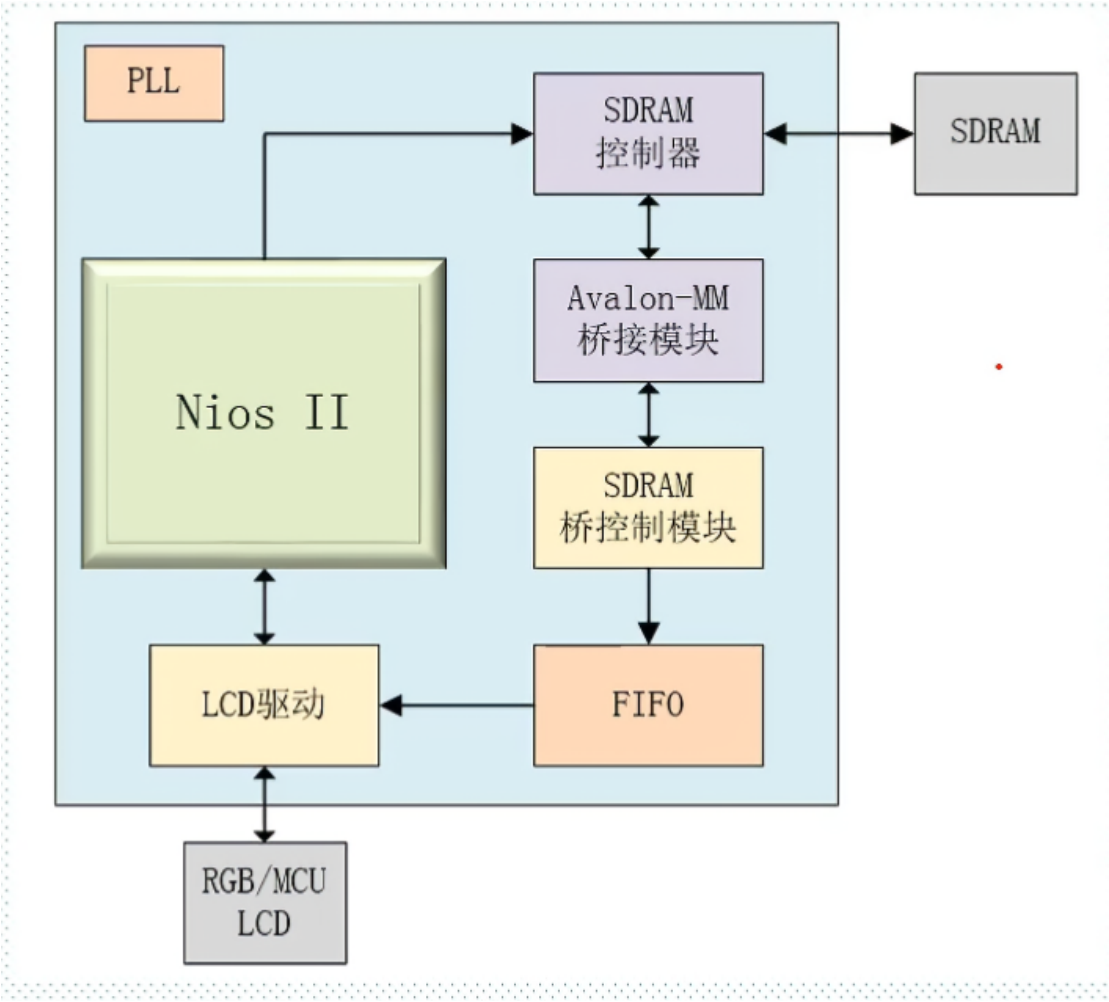

图 1 系统框图

该系统框图展示了一个基于 FPGA 的嵌入式系统，其核心目标是驱动一个 LCD 显示屏显示内容。图中浅蓝色背景框内的部分代表了在 FPGA 芯片内部实现的逻辑和组件，框外的 SDRAM 和 RGB/MCU LCD 是外部物理器件。

各模块功能及连接关系解释

1. Nios II 处理器 (Nios II)

- **功能:** 这是系统的“大脑”，一个软核嵌入式处理器。它负责运行用户编写的 C 程序，执行控制逻辑，准备要显示的数据，以及配置其他外设（如 LCD

驱动模块）。

- **连接:**
  - 连接到 **LCD 驱动模块**：Nios II 通过控制总线向 LCD 驱动模块发送命令和配置参数（例如，要显示的颜色、彩条宽度、启动显示等）。
  - 连接到 **Avalon-MM 桥接模块**：Nios II 通过 Avalon 总线访问系统中的其他 IP 核，最主要的是访问 SDRAM。它可以向 SDRAM 中写入要显示的图像数据，或者从中读取程序代码和运行时数据。
  - (隐含连接) 通常由 **PLL** 提供工作时钟。

2. PLL (Phase-Locked Loop, 锁相环)

- **功能:** 时钟管理单元。它接收外部时钟信号，并根据系统需求生成不同频率、相位的稳定时钟信号，供 FPGA 内部的各个模块使用，如 Nios II 处理器、SDRAM 控制器等。
- **连接:**
  - (隐含连接) 为 Nios II、SDRAM 控制器以及其他 FPGA 内部逻辑提供时钟。
  - 图中明确指向 **SDRAM 控制器**，表明它为 SDRAM 控制器提供精确的工作时钟。

3. SDRAM 控制器 (SDRAM Controller)

- **功能:** 这是一个 IP 核，负责管理与外部 SDRAM 芯片的物理接口和通信协议。它处理 SDRAM 的初始化、刷新、读写时序等复杂操作，将来自 Avalon 总线的简单读写请求转换为 SDRAM 能理解的命令。
- **连接:**
  - 连接到外部 **SDRAM** 芯片：进行物理层的数据交换。
  - 连接到 **Avalon-MM 桥接模块**：作为 Avalon 总线的从设备，接收来自 Nios II 或 SDRAM 桥控制模块的读写请求。
  - 接收来自 **PLL** 的时钟信号。

4. 外部 SDRAM (Synchronous Dynamic Random-Access Memory)

- **功能:** 大容量的动态随机存储器，用作系统的主内存。在本系统中，它主要用于存储 LCD 显示的帧缓冲数据（即屏幕上每个像素的颜色信息），也可能存储 Nios II 处理器的程序代码和运行时数据。
- **连接:** 与 FPGA 内部的 **SDRAM 控制器**相连。

5. Avalon-MM 桥接模块 (Avalon-MM Bridge Module)

- **功能:** Avalon Memory-Mapped (Avalon-MM) 是一种片上总线标准。这个桥接模块是系统总线的核心组件，用于连接不同的 Avalon 主设备（如 Nios II、SDRAM 桥控制模块）和从设备（如 SDRAM 控制器）。它负责地址译码、数据通路切换和总线仲裁。
- **连接:**
  - 连接到 **Nios II** (作为主设备)。
  - 连接到 **SDRAM 控制器** (作为从设备)。
  - 连接到 **SDRAM 桥控制模块** (作为主设备，用于访问 SDRAM)。

6. SDRAM 桥控制模块 (SDRAM Bridge Control Module)

- **功能:** 这个模块很可能是一个专用的数据搬运控制器，例如 DMA（Direct Memory Access）控制器，专门负责高效地从 SDRAM 中读取显示数据，并将其送往显示流水线的下一级（FIFO）。它能以突发模式高速读取 SDRAM 中的数据，减轻 Nios II 处理器的负担。
- **连接:**
  - 连接到 **Avalon-MM 桥接模块**：作为 Avalon 主设备，发起对 SDRAM 的读操作。
  - 连接到 **FIFO**：将从 SDRAM 读取到的显示数据写入 FIFO。

7. FIFO (First-In, First-Out) 缓冲器

- **功能:** 先入先出缓冲存储器。它在数据速率可能不匹配的两个模块之间起到缓冲作用。在此，它用于缓冲从 SDRAM 高速读取的显示数据，然后 LCD 驱动模块可以按照自己的固定速率从中读取数据，平滑数据流。
- **连接:**

  - 接收来自 **SDRAM 桥控制模块**的数据。
  - 向 **LCD 驱动模块**输出数据。

8. LCD 驱动模块 (LCD Driver)

- **功能:** 这是一个 IP 核，负责生成驱动 LCD 屏幕所需的特定时序信号（如行同步、场同步、像素时钟）和数据信号。它从 FIFO 中读取像素数据，按照 LCD 的时序要求将其输出到外部 LCD 屏幕。
- **连接:**
  - 接收来自 **FIFO** 的像素数据。
  - 接收来自 **Nios II** 的控制/配置信号。
  - 输出驱动信号到外部 **RGB/MCU LCD**。
  - 通过 SDRAM 桥接、Avalon-MM 桥接、SDRAM 控制器，读取 LCD 的设置信息

9. RGB/MCU LCD (外部 LCD 显示屏)

- **功能:** 物理显示设备。它接收来自 LCD 驱动模块的 RGB 数据信号和控制信号，并将图像显示出来。
- **连接:** 与 FPGA 内部的 **LCD 驱动模块**相连。

数据流总结 (以显示为例)

1. **数据准备:** Nios II 处理器计算或准备好要显示的彩条数据（像素颜色信息），并将这些数据通过 **Avalon-MM 桥接模块**写入到外部 **SDRAM** 的帧缓冲区中。

2. **数据读取与传输:**

   - **SDRAM 桥控制模块**被触发后，通过 **Avalon-MM 桥接模块**和 **SDRAM 控制器**，以突发模式从 **SDRAM** 中高速读取帧缓冲数据。
   - 读取到的数据被写入 **FIFO** 缓冲器。

3. **数据显示:**

   - **LCD 驱动模块**按照其工作时钟和 LCD 的刷新率，从 **FIFO** 中读取像素数据。

- 同时，**LCD 驱动模块**生成符合 LCD 接口规范的控制时序信号。
- 像素数据和控制信号一起被发送到外部 **RGB/MCU LCD**，最终在屏幕上显示出彩条。

控制流总结

- **Nios II** 控制整个流程的启动、配置 LCD 驱动参数（如分辨率、颜色模式、彩条宽度等）、管理 SDRAM 中的数据等。
- **PLL** 提供稳定的时钟，确保各模块同步协调工作。
- 各个**桥接模块**和**控制器**则根据 Nios II 的指令或预设逻辑，自动完成数据在不同接口和存储器之间的传输。

该系统体现了 SOPC 设计的典型特点——软硬件协同工作：Nios II 负责灵活的控制和算法处理（软件层面），而 FPGA 内部的专用硬件 IP 核（如 SDRAM 控制器、SDRAM 桥控制模块、LCD 驱动）负责高速、实时的信号处理和数据传输（硬件层面）。

3　系统总体软件设计

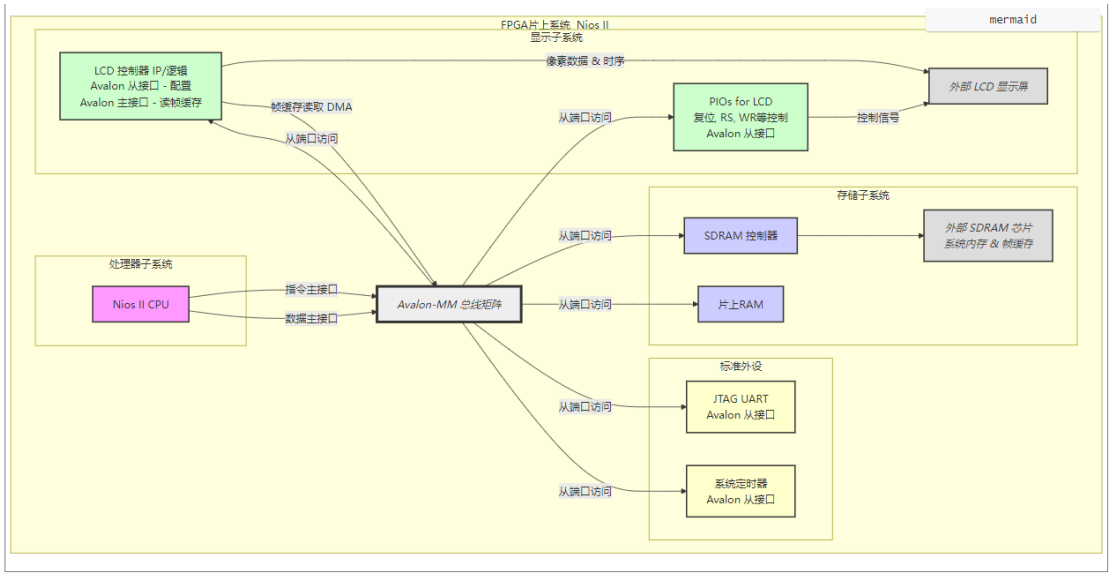

图 2 系统框图

4 系统软件设计与实现

本实验的软件设计主要围绕 Nios II 处理器的嵌入式 C 程序开发展开，核心目标是通过程序控制 SDRAM 中的显存数据，实现 LCD 屏幕的彩条显示功能。以下是软件设计的主要内容和实现过程。

4.1 软件功能概述

软件部分的主要功能包括：

- 初始化 LCD 显示屏，配置其分辨率、颜色模式等参数。
- 将彩条数据写入 SDRAM 的显存区域，按照屏幕的高度分为五个等宽区域，分别显示红色、白色、黑色、绿色和蓝色。
- 刷新数据缓存，确保 SDRAM 中的数据能够正确传输到 LCD 屏幕进行显示。

4.2 代码实现分析

以下是代码的主要实现步骤：

1. LCD 初始化

通过调用 `MY_LCD_Init()` 函数完成 LCD 的初始化，配置屏幕的分辨率和方向等参数。LCD 的宽度和高度信息存储在全局变量 `lcdgui` 中，便于后续操作。

2. 显存地址分配

显存的起始地址通过以下代码定义：

```
alt_u16 *ram_disp = (alt_u16 *)(SDRAM_BASE + SDRAM_SPAN - 2049000);
```

该地址位于 SDRAM 的末尾，用于存储 LCD 显示的像素数据。

3. 彩条数据写入

通过双重循环遍历屏幕的每个像素点，根据其所在的高度范围，将对应的颜色值写入显存：

- 红色：高度范围为屏幕的前 1/5。

- 白色：高度范围为屏幕的 1/5 到 2/5。
- 黑色：高度范围为屏幕的 2/5 到 3/5。
- 绿色：高度范围为屏幕的 3/5 到 4/5。
- 蓝色：高度范围为屏幕的最后 1/5。

颜色值通过 16 位 RGB565 格式表示，例如红色为 `0xf800`，绿色为 `0x07e0`，蓝色为 `0x001f`。

```
for(i = 0; i < lcdgui.width; i++) {
    for(j = 0; j < lcdgui.height; j++) {
        if(j < lcdgui.height / 5)
            *(ram_disp++) = 0xf800; // 红色
        else if(j < (lcdgui.height / 5 * 2))
            *(ram_disp++) = 0xffff; // 白色
        else if(j < (lcdgui.height / 5 * 3))
            *(ram_disp++) = 0x0;    // 黑色
        else if(j < (lcdgui.height / 5 * 4))
            *(ram_disp++) = 0x07e0; // 绿色
        else
            *(ram_disp++) = 0x001f; // 蓝色
    }
}
```

4. 数据缓存刷新

在数据写入 SDRAM 后，通过调用 `alt_dcache_flush_all ()` 刷新数据缓存，确保数据能够正确传输到 LCD 屏幕。

4.3 运行结果

程序运行后，LCD 屏幕按照预期显示五种颜色的彩条，每种颜色占屏幕高度的 1/5，依次为红色、白色、黑色、绿色和蓝色。该功能验证了系统软硬件协同设计的正确性和可靠性。

5　调试结果

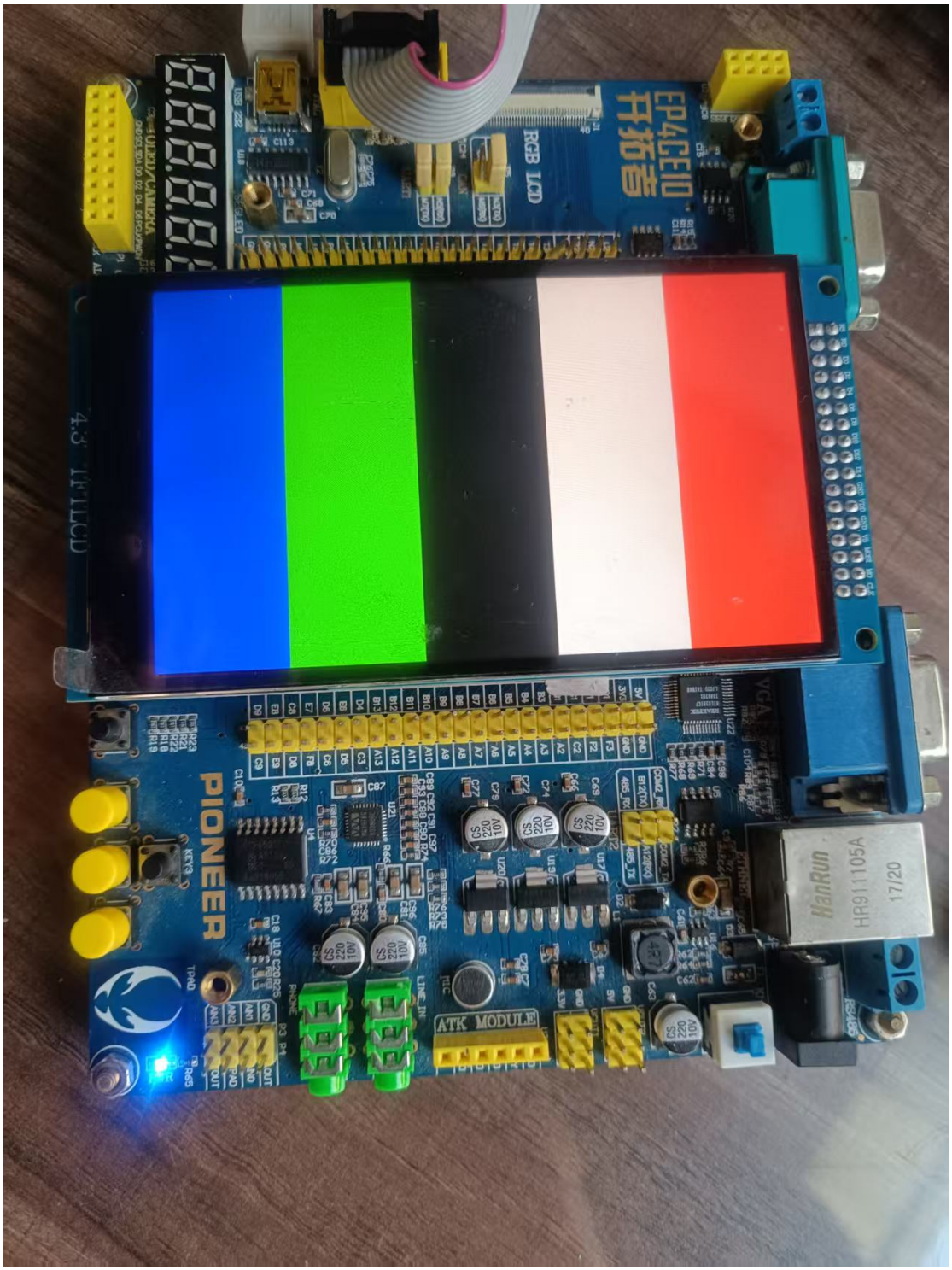

图 3 彩条测试结果

6 附录：原程序设计

附录内主要是业务程序及 Qsys 配置等，一些公有的 IP 核、NoisII 驱动等内容省略。

6.1 NoisII 软核 C 程序 main.c

```
#include <stdio.h>
#include "system.h"
#include "io.h"
#include "alt_types.h"
#include "altera_avalon_pio_regs.h"
#include "sys/alt_irq.h"
#include "unistd.h"
#include <string.h>
#include "App/mculcd.h"
#include "sys/alt_cache.h"

extern _lcd_dev lcddev; //管理LCD重要参数
_lcd_gui lcdgui;

//SDRAM显存的地址
alt_u16 *ram_disp = (alt_u16 *)(SDRAM_BASE + SDRAM_SPAN - 2049000);

int main()
{
    int i,j;
    MY_LCD_Init();              //LCD初始化
    lcdgui.width = lcddev.height;
    lcdgui.height = lcddev.width;
//向sdram中写数据,
   for(i=0;i<lcdgui.width;i++){
      for(j=0;j<lcdgui.height;j++){
        if(j<lcdgui.height/5)
            *(ram_disp++) = 0xf800;      //红色
        else if(j<(lcdgui.height/5*2))
            *(ram_disp++) = 0xffff;      //白色
        else if(j<(lcdgui.height/5*3))
```

```
32              *(ram_disp++) = 0x0;          //黑 色
33          else if(j<(lcdgui.height/5*4))
34              *(ram_disp++) = 0x07e0;       //绿 色
35          else
36              *(ram_disp++) = 0x001f;       //蓝 色
37          }
38     alt_dcache_flush_all();
39     }
40
41 return 0;
42 }
```

6.2 Qsys 配置

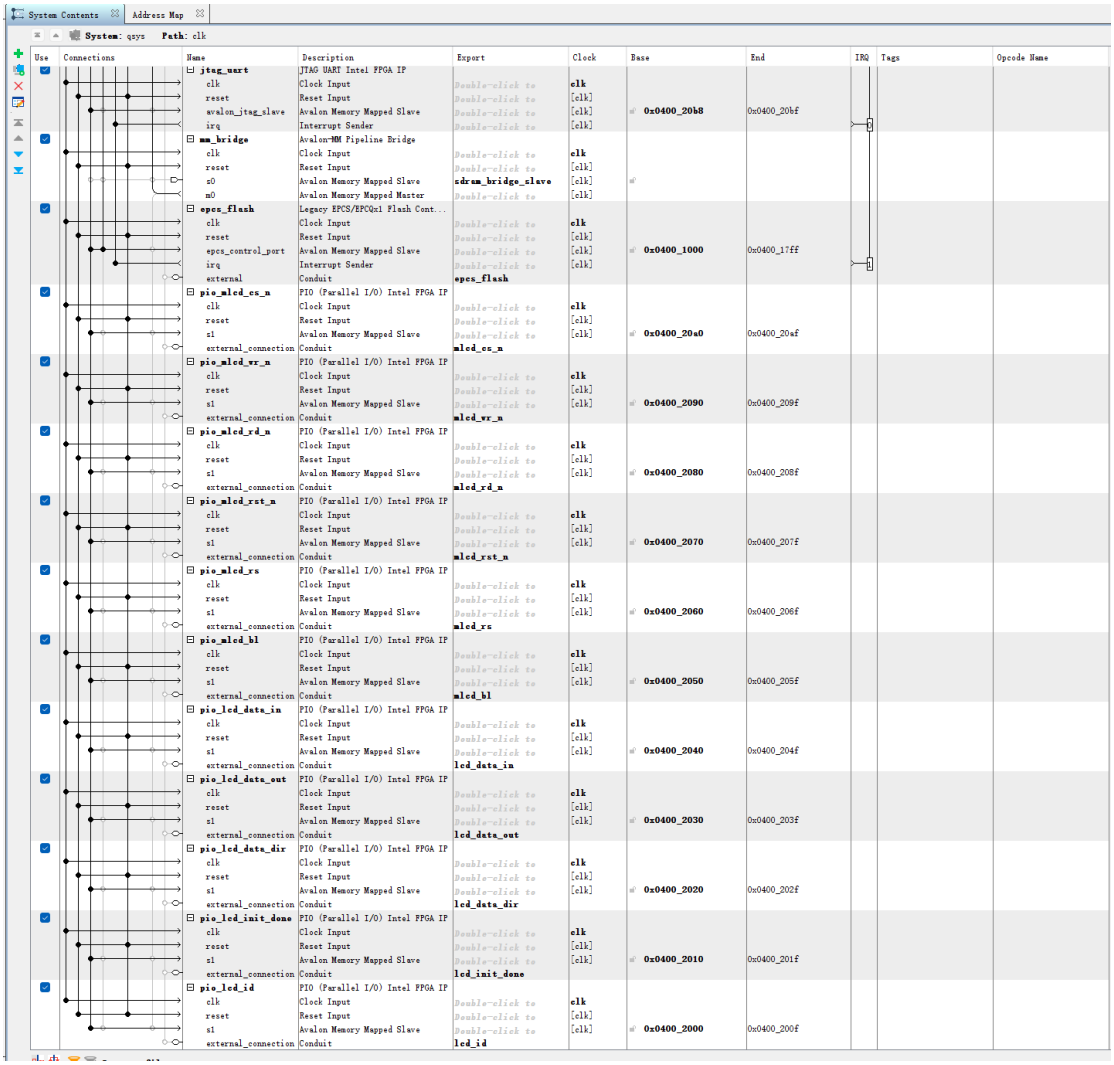

图 4 Qsys 中 IP 核配置

6.3 Verilog 顶层文件

```
module Nios_II_colorbar(

    //时钟和复位接口
    input           sys_clk,        //晶振时钟
    input           sys_rst_n,      //按键复位

    //SDRAM 接口
    output          sdram_clk,
```

```
    output [12:0]    sdram_addr,
    output [ 1:0]    sdram_ba,
    output           sdram_cas_n,
    output           sdram_cke,
    output           sdram_cs_n,
    inout  [15:0]    sdram_dq,
    output [ 1:0]    sdram_dqm,
    output           sdram_ras_n,
    output           sdram_we_n,

    //EPCS Flash 接口
    output           epcs_dclk,
    output           epcs_sce,
    output           epcs_sdo,
    input            epcs_data0,

    //LCD接口
    output           lcd_rst    ,  //LCD复位信号
    output           lcd_bl     ,  //LCD背光控制
    output           lcd_de_cs  ,  //LCD RGB:DE  MCU:CS
    output           lcd_vs_rs  ,  //LCD RGB:VS  MCU:RS
    output           lcd_hs_wr  ,  //LCD RGB:HS  MCU:WR
    output           lcd_clk_rd ,  //LCD RGB:CLK MCU:RD
    inout  [15:0]    lcd_data       //LCD DATA
);

//reg define

//wire define
wire         clk_100m_shift;
wire         sys_clk_100m;
wire         clk_50m_pll;
wire         lcd_clk;
wire         pll_locked;
wire         rst_n;

//读写 SDRAM 桥接信号
wire         bridge_write;
wire         bridge_read;
```

```
48 wire [15:0] bridge_writedata;
49 wire [15:0] bridge_readdata;
50 wire [25:0] bridge_address;
51 wire [ 9:0] bridge_burstcount;
52 wire         bridge_waitrequest;
53 wire         bridge_readdatavalid;
54
55 //source_st_fifo信号
56 wire [9:0]   source_fifo_wrusedw;
57
58 //LCD驱动模块接口信号
59 wire         lcd_data_req;
60 wire [15:0] lcd_pixel_data;
61
62 //LCD初始化完成
63 wire         lcd_init_done     ;
64 wire [15:0] lcd_id             ;
65 wire         mlcd_cs_n_init    ;
66 wire         mlcd_wr_n_init    ;
67 wire         mlcd_rd_n_init    ;
68 wire         mlcd_rst_n_init   ;
69 wire         mlcd_rs_init      ;
70 wire         mlcd_bl_init      ;
71 wire         mlcd_data_dir_init;
72 wire [15:0] mlcd_data_out_init;
73 wire [15:0] mlcd_data_in_init ;
74
75 //*****************************************************
76 //**                    main code
77 //*****************************************************
78
79 assign rst_n = sys_rst_n & pll_locked ;
80 assign sdram_clk = clk_100m_shift;
81
82 //例化锁相环模块
83 pll u_pll (
84     .inclk0                             (sys_clk    ),
85     .areset                             (~sys_rst_n),
86     .c0                                 (sys_clk_100m),
```

```
        //QSYS 系统时钟
87      .c1                                 (clk_100m_shift),
        //SDRAM 时钟
88      .c2                                 (clk_50m_pll),
        //LCD 驱动时钟
89      .locked                             (pll_locked)
90      );
91
92  //例化QSYS系统
93  qsys u_qsys(
94
95      //时钟和复位
96      .clk_clk                            (sys_clk_100m),
97      .reset_reset_n                      (rst_n),
98
99      //EPCS
100     .epcs_flash_dclk                    (epcs_dclk ),
101     .epcs_flash_sce                     (epcs_sce  ),
102     .epcs_flash_sdo                     (epcs_sdo  ),
103     .epcs_flash_data0                   (epcs_data0),
104
105     //SDRAM
106     .sdram_addr                         (sdram_addr),
107     .sdram_ba                           (sdram_ba),
108     .sdram_cas_n                        (sdram_cas_n),
109     .sdram_cke                          (sdram_cke),
110     .sdram_cs_n                         (sdram_cs_n),
111     .sdram_dq                           (sdram_dq),
112     .sdram_dqm                          (sdram_dqm),
113     .sdram_ras_n                        (sdram_ras_n),
114     .sdram_we_n                         (sdram_we_n),
115
116     //读写SDRAM的桥
117     .sdram_bridge_slave_waitrequest     (bridge_waitrequest),
118     .sdram_bridge_slave_readdata        (bridge_readdata),
119     .sdram_bridge_slave_readdatavalid   (bridge_readdatavalid),
120     .sdram_bridge_slave_burstcount      (bridge_burstcount),
121     .sdram_bridge_slave_writedata       (bridge_writedata),
122     .sdram_bridge_slave_address         (bridge_address),
```

```
123     .sdram_bridge_slave_write           (bridge_write),
124     .sdram_bridge_slave_read            (bridge_read),
125     .sdram_bridge_slave_byteenable      (2'b11),
126     .sdram_bridge_slave_debugaccess     (),
127
128     //PIO 输入输出
129     .mlcd_cs_n_export                   (mlcd_cs_n_init),
130     .mlcd_wr_n_export                   (mlcd_wr_n_init),
131     .mlcd_rd_n_export                   (mlcd_rd_n_init),
132     .mlcd_rst_n_export                  (mlcd_rst_n_init),
133     .mlcd_rs_export                     (mlcd_rs_init),
134     .mlcd_bl_export                     (mlcd_bl_init),
135     .lcd_data_in_export                 (mlcd_data_in_init),
136     .lcd_data_out_export                (mlcd_data_out_init),
137     .lcd_data_dir_export                (mlcd_data_dir_init),
138     .lcd_init_done_export               (lcd_init_done),          //
        LCD初始化完成
139     .lcd_id_export                      (lcd_id)                  //
        LCD ID
140     );
141
142 //读写 SDRAM 桥 控制模块
143 sdram_bridge_control u_bridge_ctrl(
144     .clk                                (sys_clk_100m),
145     .rst_n                              (rst_n & lcd_init_done),
146
147     .bridge_write                       (bridge_write),
148     .bridge_read                        (bridge_read),
149     .bridge_address                     (bridge_address),
150     .bridge_burstcount                  (bridge_burstcount),
151     .bridge_waitrequest                 (bridge_waitrequest),
152     .bridge_readdatavalid               (bridge_readdatavalid),
153
154     .lcd_id                             (lcd_id),
155     .source_fifo_wrusedw                (source_fifo_wrusedw)
156     );
157
158 // FIFO: 缓存SDRAM中读出的数据供LCD读取
159 fifo u_fifo(
```

```
160     .wrclk                          (sys_clk_100m),
161     .rdclk                          (lcd_clk),
162
163     .wrreq                          (bridge_readdatavalid),
164     .data                           (bridge_readdata),
165     .wrusedw                        (source_fifo_wrusedw),
166
167     .rdreq                          (lcd_data_req),
168     .q                              (lcd_pixel_data),
169     .rdempty                        (),
170
171     .aclr                           (~(rst_n & lcd_init_done))
172     );
173
174 //RGB LCD 和 MCU LCD驱动
175 lcd_top  u_lcd_top(
176     .clk                            (clk_50m_pll),
177     .rst_n                          (rst_n & lcd_init_done),
178     .pixel_data                     (lcd_pixel_data),
179     .pixel_en                       (lcd_data_req),
180     .lcd_clk                        (lcd_clk),
181
182     .lcd_rst                        (lcd_rst),
183     .lcd_bl                         (lcd_bl),
184     .lcd_de_cs                      (lcd_de_cs),
185     .lcd_vs_rs                      (lcd_vs_rs),
186     .lcd_hs_wr                      (lcd_hs_wr),
187     .lcd_clk_rd                     (lcd_clk_rd),
188     .lcd_data                       (lcd_data),
189
190     .mlcd_cs_n_init                 (mlcd_cs_n_init),
191     .mlcd_wr_n_init                 (mlcd_wr_n_init),
192     .mlcd_rd_n_init                 (mlcd_rd_n_init),
193     .mlcd_rst_n_init                (mlcd_rst_n_init),
194     .mlcd_rs_init                   (mlcd_rs_init),
195     .mlcd_bl_init                   (mlcd_bl_init),
196     .mlcd_data_dir_init             (mlcd_data_dir_init),
197     .mlcd_data_out_init             (mlcd_data_out_init),
198     .mlcd_data_in_init              (mlcd_data_in_init ),
```

```
199     .lcd_init_done                          (lcd_init_done),
200     .lcd_id                                 (lcd_id)
201     );
202
203 endmodule
```

6.4 Verilog 重要驱动 1：LCD 驱动

```
1  //****************************************Copyright (c)
       **********************************//
2  //技术支持: www.openedv.com
3  //淘宝店铺: http://openedv.taobao.com
4  //关注微信公众平台微信号: "正点原子", 免费获取FPGA & STM32资料。
5  //版权所有, 盗版必究。
6  //Copyright(C) 正点原子 2018-2028
7  //All rights reserved
8  //
       ----------------------------------------------------------------------
9  // File name:            mlcd_driver
10 // Last modified Date:   2018/1/30 11:12:36
11 // Last Version:         V1.1
12 // Descriptions:         MCU LCD驱动
13 //
       ----------------------------------------------------------------------
14 // Created by:           正点原子
15 // Created date:         2018/1/29 10:55:56
16 // Version:              V1.0
17 // Descriptions:         The original version
18 //
19 //
       ----------------------------------------------------------------------
20 // Modified by:          正点原子
21 // Modified date:        2018/8/15 14:23:12
22 // Version:              V1.1
23 // Descriptions:         Intel8080总线
```

```
24 //
25 //
       ----------------------------------------------------------------------------
26 //
       ****************************************************************************
27
28 module mlcd_driver(
29     input          clk        ,     //时钟
30     input          rst_n      ,     //复位，低电平有效
31     output         mlcd_bl    ,     //MCU LCD 背光控制信号
32     output         mlcd_cs    ,     //MCU LCD 片选信号
33     output         mlcd_rst   ,     //MCU LCD 复位信号
34     output         mlcd_wr    ,     //MCU LCD 写使能信号
35     output         mlcd_rd    ,     //MCU LCD 读使能信号
36     output         mlcd_rs    ,     //MCU LCD 指令/数据控制信号
37     output [15:0] mlcd_data ,     //MCU LCD 双向数据总线
38
39     input          lcd_init_done, //LCD初始化完成
40     input  [15:0] lcd_id     ,     //LCD ID
41     input  [15:0] pixel_data,     //从fifo中读出的数据
42     output  reg    rd_en          //fifo读使能信号
43     );
44
45 //parameter define
46 parameter  idle = 2'd0;
47 parameter  step1 = 2'd1;
48 parameter  step2 = 2'd2;
49 parameter  step3 = 2'd3;
50
51 //reg define
52 reg        lcd_done_d0;
53 reg        lcd_done_d1;
54 reg [15:0] lcd_id_r;
55
56 reg [10:0] lcd_height;
57 reg [10:0] lcd_width;
58
```

```
59 reg         wr_r;
60 reg         rd_r;
61 reg         rs_r;
62 reg [15:0]  data_r;
63
64 reg [2:0]   wr_step;
65 reg [10:0]  h_cnt;
66 reg [10:0]  v_cnt;
67 reg [10:0]  h_blank_cnt;
68 reg [10:0]  v_blank_cnt;
69
70 //wire define
71 wire        pos_lcd_done;
72
73 //*****************************************************
74 //**                    main code
75 //*****************************************************
76
77 assign mlcd_bl   = 1'b1;         //设置屏幕背光为最亮
78 assign mlcd_cs   = 1'b0;         //片选信号低电平有效
79 assign mlcd_rst  = 1'b1;         //初始化完成后，LCD不复位
80 assign mlcd_wr   = wr_r;         //LCD写信号
81 assign mlcd_rd   = rd_r;         //LCD读信号
82 assign mlcd_rs   = rs_r;         //LCD指令/数据控制信号
83 assign mlcd_data = data_r;       //LCD数据线
84
85 assign pos_lcd_done = ~lcd_done_d1 & lcd_done_d0;
86
87 //lcd_init_done上升沿
88 always@(posedge clk or negedge rst_n) begin
89     if(!rst_n) begin
90         lcd_done_d0 <= 1'b0;
91         lcd_done_d1 <= 1'b0;
92     end
93     else begin
94         lcd_done_d0 <= lcd_init_done;
95         lcd_done_d1 <= lcd_done_d0;
96     end
97 end
```

```
98
99  always@(posedge clk or negedge rst_n) begin
100     if(!rst_n)
101         lcd_id_r <= 16'd0;
102     else if(pos_lcd_done)
103         lcd_id_r <= lcd_id;
104 end
105
106 //利用状态机向LCD控制器写指令及数据
107 always@(posedge clk or negedge rst_n) begin
108     if(!rst_n) begin
109         wr_r         <= 1'b1;
110         rd_r         <= 1'b1;
111         rs_r         <= 1'b0;
112         data_r       <= 16'd0;
113         rd_en        <= 1'b0;
114         lcd_height   <= 11'b0;
115         lcd_width    <= 11'b0;
116         h_cnt        <= 11'd0;
117         v_cnt        <= 11'b0;
118         h_blank_cnt  <= 11'b0;
119         v_blank_cnt  <= 11'b0;
120         wr_step      <= idle;
121     end
122     else begin
123         case(wr_step)
124             idle: begin
125                 rd_r     <= 1'b1;
126                 wr_r     <= 1'b1;
127                 rd_en    <= 1'b0;
128                 if(lcd_done_d1) begin
129                     wr_step <= step1;
130                     case(lcd_id_r)          //根据LCD ID,选择不同的
                            寄存器指令与分辨率
131                         16'h9341 : begin
132                             lcd_width    <= 11'd320-1'b1;
133                             lcd_height   <= 11'd240-1'b1;
134                             h_blank_cnt  <= 30;
135                             v_blank_cnt  <= 10;
```

```
                    end
                    16'h5310 : begin
                        lcd_width   <= 11'd480-1'b1;
                        lcd_height  <= 11'd320-1'b1;
                        h_blank_cnt <= 80;
                        v_blank_cnt <= 45;
                    end
                    16'h5510 : begin
                        lcd_width   <= 11'd800-1'b1;
                        lcd_height  <= 11'd480-1'b1;
                        h_blank_cnt <= 11'd200;
                        v_blank_cnt <= 11'd15;
                    end
                    16'h1963 : begin
                        lcd_height  <= 11'd800-1'b1;
                        lcd_width   <= 11'd480-1'b1;
                        h_blank_cnt <= 11'd200;
                        v_blank_cnt <= 11'd15;
                    end
                    default : wr_step <= idle;
                endcase
            end
            else
                wr_step <= idle;
        end
        step1: begin                                    //发送写GRAM指
            令
            wr_r <= 1'b0;
            rs_r <= 1'b0;
            if(lcd_id_r == 16'h5510)
                data_r <= 16'h2c00;
            else
                data_r <= 16'h002c;
            if(wr_r == 1'b0) begin
                wr_r <= 1'b1;
                wr_step <= step2;
            end
        end
        step2 : begin
```

```
174                 wr_r <= 1'b1;
175                 h_cnt <= h_cnt + 1'b1;
176                 if(h_cnt == lcd_width + h_blank_cnt + 1'b1) begin
177                     h_cnt <= 11'd0;
178                     v_cnt <= v_cnt + 1'b1;
179                     if(v_cnt == lcd_height + v_blank_cnt + 1'b1)
                            begin
180                         v_cnt <= 11'd0;
181                         wr_step <= idle;
182                     end
183                 end
184                 if(v_cnt >= v_blank_cnt && v_cnt <= (lcd_height +
                        v_blank_cnt)) begin
185                     if(h_cnt >= h_blank_cnt && h_cnt <= (lcd_width +
                            h_blank_cnt))
186                         wr_step <= step3;
187                     if(h_cnt == h_blank_cnt - 1'b1)     //提前拉高fifo
                            读使能信号
188                         rd_en <= 1'b1;
189                     else
190                         rd_en <= 1'b0;
191                 end
192                 else
193                     rd_en <= 1'b0;
194             end
195             step3: begin                                    //写像素数据
196                 wr_r <= 1'b0;
197                 rs_r <= 1'b1;
198                 data_r <= pixel_data;
199                 wr_step <= step2;
200                 if(h_cnt == lcd_width + h_blank_cnt + 1'b1)
201                     rd_en <= 1'b0;
202                 else
203                     rd_en <= 1'b1;
204             end
205         endcase
206     end
207 end
208
```

```
endmodule
```

6.5 Verilog 重要驱动 2：SDRAM 控制器

```
module sdram_bridge_control(
    input                clk                    ,
    input                rst_n                  ,

    output  reg          bridge_write           ,
    output  reg          bridge_read            ,
    output  reg  [25:0]  bridge_address         ,
    output  reg  [9:0]   bridge_burstcount      ,
    input                bridge_waitrequest     ,
    input                bridge_readdatavalid   ,

    input        [15:0]  lcd_id                 ,
    input        [ 9:0]  source_fifo_wrusedw
    );

//parameter define

//SDRAM 存储容量 = 2^13(row)*2^9(col)*4(bank)*2(byte)
parameter SDRAM_SPAN = 33554432;

//LCD存储图像地址参数设置（800*480）
parameter  sdram_addr_start    = SDRAM_SPAN - 2048000 - 1000;     //
    sdram显存起始地址;
parameter  sdram_addr_end_320_240  = sdram_addr_start + 153600;  //结
    束地址（320*240）
parameter  sdram_addr_end_480_272  = sdram_addr_start + 261120;  //结
    束地址（480*272）
parameter  sdram_addr_end_480_320  = sdram_addr_start + 307200;  //结
    束地址（480*320）
parameter  sdram_addr_end_800_480  = sdram_addr_start + 768000;  //结
    束地址（800*480）
parameter  sdram_addr_end_1024_600 = sdram_addr_start + 1228800; //结
    束地址（1024*600）
parameter  sdram_addr_end_1280_800 = sdram_addr_start + 2048000; //结
```

```
    束地址（1280*800）
parameter  burstcount            = 10'd512;                    //
    SDRAM 突发长度
parameter  burst_addr            = 11'd1024 ;                  //一
    次突发的地址长度
parameter  usedw_wr              = 512;                        //读
    fifo的数据深度

//reg define
reg  [25:0] address_rd;        //读fifo端读取数据的sdram地址
reg  [9:0]  cnt_burst;         //计数一次突发读数据过程中已读取的个数
reg         step;              //step
reg         step_1;
reg  [25:0] sdram_addr_end;    //sdram显存结束地址

//wire define
wire burst_start;              //开始突发标志

//*****************************************************
//**                    main code
//*****************************************************

//采集step上升沿信号，标志着突发传输指令已发出
assign burst_start = (~step_1 ) & step;

//寄存step信号，用于边沿捕获
always @ (posedge clk  or negedge rst_n ) begin
    if(!rst_n )
        step_1 <= 1'b0;
    else
        step_1 <=  step;
end

always @ (*) begin
    case(lcd_id)
        16'h4342 : sdram_addr_end = sdram_addr_end_480_272 ;
        16'h7084 : sdram_addr_end = sdram_addr_end_800_480 ;
        16'h7016 : sdram_addr_end = sdram_addr_end_1024_600;
        16'h1018 : sdram_addr_end = sdram_addr_end_1280_800;
```

```
64          16'h9341 : sdram_addr_end = sdram_addr_end_320_240 ;
65          16'h5310 : sdram_addr_end = sdram_addr_end_480_320 ;
66          16'h5510 : sdram_addr_end = sdram_addr_end_480_320 ;
67          16'h1963 : sdram_addr_end = sdram_addr_end_480_320 ;
68      default  : sdram_addr_end = sdram_addr_end_480_272 ;
69      endcase
70  end
71
72  //读SDRAM的地址
73  always @ (posedge clk or negedge rst_n ) begin
74      if(!rst_n )
75          address_rd <= sdram_addr_start;
76      else if (address_rd == sdram_addr_end)
77              address_rd <= sdram_addr_start;
78      else if (burst_start)
79          address_rd <= address_rd + burst_addr;
80  end
81
82  //计数突发读出的数据个数
83  always @ (posedge clk or negedge rst_n) begin
84      if(!rst_n)
85          cnt_burst <= 10'b0;
86      else if (cnt_burst == burstcount)
87          cnt_burst <= 10'b0;
88      else if (bridge_readdatavalid)
89          cnt_burst <= cnt_burst + 1'b1;
90  end
91
92  //fifo中的数据量低于512时，从sdram中读数据
93  always @ (posedge clk or negedge rst_n) begin
94      if(!rst_n) begin
95          step                <= 1'b0;
96          bridge_read         <= 1'b0;
97          bridge_write        <= 1'b0;
98          bridge_address      <= 26'b0;
99          bridge_burstcount   <= 10'b0;
100     end
101     else if((! bridge_waitrequest) && (source_fifo_wrusedw < usedw_wr
            )
```

```
102             &&(cnt_burst == 10'd0) ) begin   //从sdram读数据
103         case(step)
104             1'b0: begin
105                 step                <= 1'b1;
106                 bridge_read         <= 1'b1;
107                 bridge_write        <= 1'b0;
108                 bridge_address      <= address_rd;
109                 bridge_burstcount   <= burstcount;
110             end
111             1'b1: begin
112                 bridge_read       <= 1'b0;
113                 bridge_address    <= 26'b0;
114                 bridge_burstcount <= 10'b0;
115             end
116         default: ;
117         endcase
118     end
119     else if (cnt_burst == burstcount)
120         step <= 1'b0;
121 end
122 
123 endmodule
```