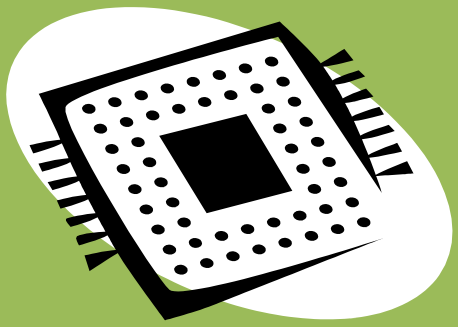
华中科技大学

2022

接口技术 实验报告

专　　业：	计算机科学与技术
班　　级：	本硕博 2101 班
学　　号：	U202115666
姓　　名：	刘文博
电　　话：	15527700612
邮　　件：	3097365481@qq.com
完成日期：	2023.11.14



计算机科学与技术学院

目 录

1 课程实验概述

1.1 实验目的

《接口技术》是一门工程性和实践性都非常强的计算机专业课程。为了充分调动学生的学习积极性，体现研究性学习，提高学习效果，在各个理论知识点讲述的过程中，设计了有针对性的实验。实验注重将各章节知识有机结合，循环展开，逐步深入，使学生更加牢固的掌握接口技术，同时树立计算机的系统观和软硬件协同观，为后续计算机系统综合能力的培养奠定坚实的基础；同时，力求通过这些硬件类课程的学习同时提升学生软件设计水平。

课程实验属于设计型实验，不仅锻炼学生简单计算机系统的设计能力，而且通过具体接口的设计与实现，进行硬件的实现、故障分析与定位，以及接口的应用和系统集成，使学生在系统调试等环节得到综合锻炼，进一步提高学生分析和解决问题的能力。

1.2 实验任务

设计一个基于 RVfpga_SoC 处理器系统的接口应用综合系统，该接口综合系统是一个具有一定实用价值的应用系统，在前面接口实验的基础上，在 RVfpga_SoC 处理器系统中至少再添加一个新的外设模块（**注：**即前面实验没有使用过的，这个外设模块可以是 Vivado 自带的、也可以是现成的免费的 IP 模块、还可以是自定制的 IP 模块），完成硬件系统的搭建并生成比特流（bitstream），完成接口驱动程序和应用软件的设计，另外综合应用系统要能够充分展示同学们的想象力和创造性。

1.3 实验要求

完成实验任务并撰写设计报告，提交能够生成综合应用系统硬件平台比特流（bitstream）的 Vivado 工程源程序、接口驱动程序和应用软件的源代码。

1.4 团队分工

刘文博：主要负责该接口综合应用系统的软件设计。

姚晨炫：主要负责该接口综合应用系统的各个硬件模块的实现。

刘兴元：主要负责该接口综合应用系统的数据可视化界面的实现

2 接口综合应用系统（VGA 界面可视化）

2.1 设计目标

构建基于 RVfpga_SoC 处理器系统的综合应用，融合 ADXL362 MEMS 加速计和 VGA 接口技术，以创造一款实用的加速计图形化应用系统。系统的核心目标在于实时采集 ADXL362 加速计的三轴高精度加速度数据，并通过我们已经设计好且连接正确的 VGA 接口，在连接的显示器上呈现清晰、实时的加速度信息。该设计旨在突显学在硬件系统搭建、外设集成、接口驱动程序编写以及应用软件设计方面的高级工程能力。通过此项目，我们能够深入了解计算机系统综合能力、软硬件协同设计，并能够展示出对现代嵌入式系统和接口技术的深刻理解。最终我们实现的速计图形化应用系统将为用户提供一种先进、可视化的加速度信息展示。

2.2 总体方案

在硬件设计层面：

我们需要在 Nexys4 DDR 开发板上整合各个模块和接口，包括相关存储模块、时钟管理单元（Clocking Wizard）以及 AXI 总线互联。通过 Vivado 设计软件的 IP Integrator，将加速计的 SPI 接口和 VGA 控制器集成到系统中。配置处理器核的地址映射，确保各模块能够正确访问。

对于加速计接口设计：我们配置接口以与 ADXL362 通信。实现对 ADXL362 寄存器的读写操作。设置相应的控制寄存器，启动测量模式，并定期读取 X、Y、Z 轴的加速度数据。设计 FIFO 缓冲区，以确保在处理器准备好接收数据时不会丢失任何采样。

对于 VGA 界面集成：我们集成 VGA 控制器 IP，并配置输出参数，包括分辨率、刷新率等。设计定制逻辑将来自 ADXL362 的加速度数据映射为图形化数字显示。通过 VGA 模块输出，将图像数据传输到连接的显示器。我们需要确保 VGA 时序和信号兼容常见显示设备，否则会出现显示异常的情况。同时，我们需要实现硬件层面的中断机制，配置 ADXL362 的中断引脚，当有新的数据可用时触发中断。中断处理

程序负责通知处理器进行数据处理和更新 VGA 界面。通过适当的同步措施，确保中断的及时响应。

在软件设计层面：

我们需要结合 lab8 开发 ADXL362 的驱动程序，使用处理器的 SPI 控制器进行通信。编写初始化例程，配置加速计的操作模式、测量范围等参数。实现中断服务例程，以处理 ADXL362 中断触发时的数据获取和处理。通过合理的缓冲机制，确保在处理器准备好时有效地接收加速度数据。同时，我们还要建构好 VGA 界面显示模块。通过软件驱动 VGA 接口控制器，将加速度数据映射为可视化图形元素，如数字像素点。以提高我们的数据可视性。

通过紧密结合硬件和软件，我们通过学习的接口技术，结合各个先前做的基于 RVfpga 的接口实验，构建了一个高度集成的加速度数据可视化系统。这一系统不仅在硬件层面实现了对 ADXL362 的高效控制和数据采集，同时通过软件实现了直观的图形化数据显示。很好的为用户提供实时加速度信息。

2.3 硬件设计与实现

2.3.1 硬件设计

硬件框架上基于 AXI4 的自定制接口，利用 FPGA 开发板的 VGA 显示接口，实现基础实验 lab8 的可视化。VGA 接口的 IP 封装如下图所示。

图 2.1 VGA IP 封装示意图

然后将 VGA 的 IP 封装借入 Vivado Block Design 中，首先在 AXI4 总线模块上添加一个新的端口，然后将自定义的 VGA IP 模块接入，，将 vga 显色，h_sync 和 v_sync 设置为外部端口，并在 FPGA 约束文件中定义相关的引脚。如下图所示

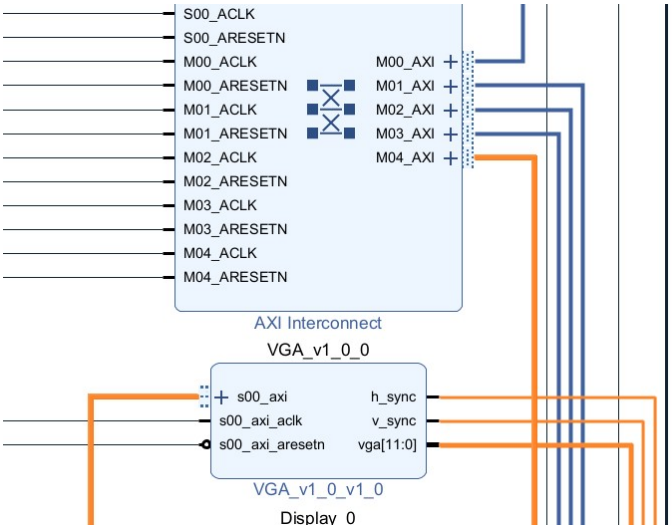


图 2.2 VGA 模块接入 AXI4 总线

在 Address Editor 中为 VGA 以及各个 AXI4 总线模块分配地址，如下图所示

/axi2wb_intcon_wrapper_0

/axi2wb_intcon_wrapper_0/o_user_axi4 (32 address bits : 4G)

/axi_gpio_0/S_AXI	S_AXI	Reg	0x8010_0000	64K	0x8010_FFFF
/axi_iic_0/S_AXI	S_AXI	Reg	0x8016_0000	64K	0x8016_FFFF
/Display_0/s00_axi	s00_axi	reg0	0x8012_0000	64K	0x8012_FFFF
/PWM_w_Int_v1_0_0/s00_axi	s00_axi	reg0	0x8014_0000	64K	0x8014_FFFF
/VGA_v1_0_0/s00_axi	s00_axi	reg0	0x8018_0000	64K	0x8018_FFFF

图 2.3 模块地址分配

对于 VGA 显示接口，运行原理为，从 AXI4 总线上获取需要显示的数据，缓存在 VGA 显存中，VGA 接口工作时不断扫描屏幕，当扫描到需要显示的像素时模块传输 vga 颜色数据和 h_sync,v_sync 数据，达到可视化的效果

实验采用的可视化模块数据源为 ADXL362 加速计数据，ADXL362 是一款 3 轴 MEMS 加速计，该加速计拥有 12 位输出分辨率，测量范围为±2g、±4g 和±8g，当 ADXL362 处于测量模式时，将连续测量加速度数据并将其存储在 X 数据、Y 数据和 Z 数据寄存器中。VGA 显示的数据来源就是这里。

ADXL362 加速计模块的连接方式与基础实验 lab8 相同，连接在 wishbone 总线上，在 Block Design 中将“o_accel_sclk”、“o_accel_cs_n”、“o_accel_mosi”和“i_accel_miso”设置为外部引脚并在约束文件中定义。ADXL362 加速计模块借入 wishbone 总线时如下图所示

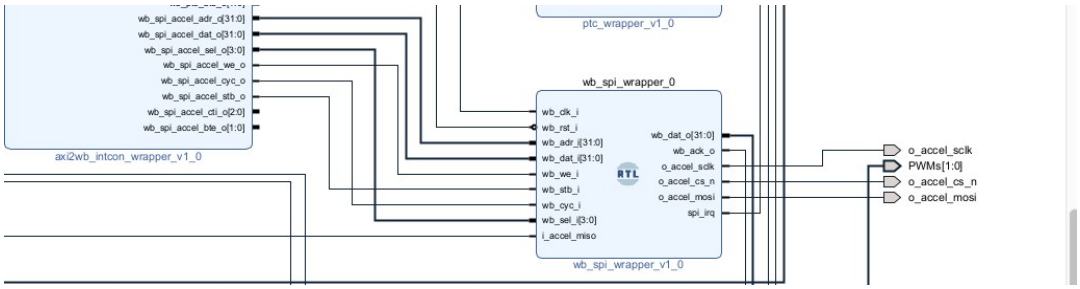


图 2.4 ADXL362 加速计模块

2.3.2 硬件实现

ADXL362 加速计模块是基础实验自带的模块，因此本节主要介绍 VGA 显示模块的设计。

首先要了解 VGA 显示的基本原理，GA 显示图像使用扫描的方式，从第一行的第一个像素开始，逐渐填充，第一行第一个,第一行第二个…第二行第一个,第二行第二个…第 n 行最后一个。通过这种方式构成一帧完整的图像，当扫描速度足够快，加之人眼的视觉暂留特性，我们会看到一幅完整的图片，而不是一个个闪烁的像素点。这就是 VGA 显示的原理。

VGA 有两个非常重要的信号，一个是行同步信号(HSYNC)，另一个是场同步信号(VSYNC)。通过这两个信号完成一帧图像的像素点扫描。在数字电路中，时序是我们进行程序设计的重要依据。行场同步信号的时序如下

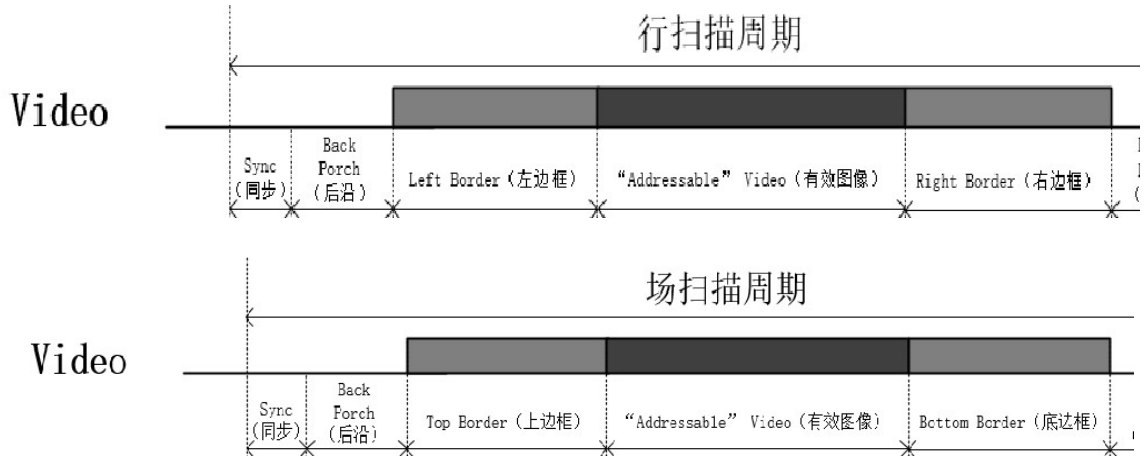


图 2.5 VGA 的行场同步信号

两个信号的配合由 VGA 的显示模式确定，VGA 可以显示不同分辨率的图像，不同分辨率及帧率对应参数如下，我们实验选择的是 1024*768@60。

显示模式	时钟	行同步信号时序(像素)							场同步信号			
	(MHz)	同步	后沿	左边框	有效图像	右边框	前沿	行扫描周期	同步	后沿	上边框	有效图像
640x480@60	25.175	96	40	8	640	8	8	800	2	25	8	480
640x480@75	31.5	64	120	0	640	0	16	840	3	16	0	480
800x600@60	40.0	128	88	0	800	0	40	1056	4	23	0	600
800x600@75	49.5	80	160	0	800	0	16	1056	3	21	0	600
1024x768@60	65	136	160	0	1024	0	24	1344	6	29	0	768

图 2.6 VGA 不同分辨率参数

然后是在 Verilog 中按上述参数实现。基本扫描代码如下

```
always @(posedge clk)

    if (x_counter == h_total_piexls-1)
        x_counter <= 0;
    else
        x_counter <= x_counter + 1;

always @(posedge clk)
    if (x_counter == h_total_piexls-1)
    begin
        if (y_counter == v_total_piexls-1)
            y_counter <= 0;
        else
            y_counter <= y_counter + 1;
    end

always @(posedge clk)
  begin
     h_sync <= (x_counter >= h_sync_width + h_back_porch && x_counter < h_active_pixels + h_sync_width + h_back_porch)
     v_sync <= (y_counter >= v_sync_width + v_back_porch && y_counter < v_active_pixels + v_sync_width + v_back_porch)
  end

assign vaddr_x = (h_sync)?(x_counter - h_sync_width - h_back_porch + 1):11'h7ff;
assign vaddr_y = (v_sync)?(y_counter - v_sync_width - v_back_porch ):10'h3ff;

assign vdata = (vaddr_x <= 11'h120)? 12'h000 :
               (vaddr_y <= 10'h90)? 12'h000 :
               (RAM[(vaddr_y-144)/24][31-((vaddr_x-287)/24)])? 12'hfff :
```

图 2.7 VGA 扫描相关代码

然后再开始时候，导入 UI 相关的初始化代码，将每一次需要显示的画面传入 RAM 中，VGA 控制模块按上述规则扫描，从而实现 RAM 中数据的可视化。

现在还剩下的问题是如何将 RAM 中的数据翻译成显示器上可以显示的数据，这需要我们给出一个类似码表的东西，将 RAM 中的数据翻译成 5*3 的形式显示在屏幕上，我们绘制了数据对应的编码，并实现了相应的模块翻译。如下图所示

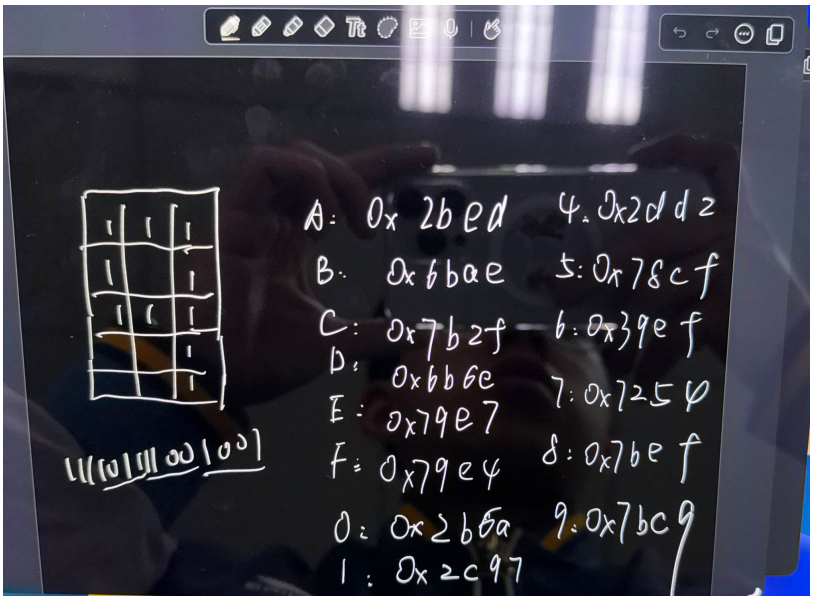


图 2.8 数字相关编码

```
case(num2)
    4'b0000: row2[14:0] <= 16'h2b6a;
    4'b0001: row2[14:0] <= 16'h2c97;
    4'b0010: row2[14:0] <= 16'h2a57;
    4'b0011: row2[14:0] <= 16'h73cf;
    4'b0100: row2[14:0] <= 16'h2dd2;
    4'b0101: row2[14:0] <= 16'h78cf;
    4'b0110: row2[14:0] <= 16'h39ef;
    4'b0111: row2[14:0] <= 16'h7254;
    4'b1000: row2[14:0] <= 16'h7bef;
    4'b1001: row2[14:0] <= 16'h7bc9;
    4'b1010: row2[14:0] <= 16'h2bed;
    4'b1011: row2[14:0] <= 16'h6bae;
    4'b1100: row2[14:0] <= 16'h7b2f;
    4'b1101: row2[14:0] <= 16'h6b6e;
    4'b1110: row2[14:0] <= 16'h79e7;
    4'b1111: row2[14:0] <= 16'h79e4;
    default: row2[14:0] <= 16'h0000;
endcase
```

图 2.8 编码相关代码

还有一个细节，就是 VGA 显示模块的 clk 时钟与 FPGA 上的不太相同，需要特化的时钟模块来分频，我这里选择的是 clk_wiz 模块，将输入的 100mhz 时钟转为 VGA 需要的 50mhz 时钟。

图 2.9 clk_wiz 模块

准备好所有的代码和仿真成功之后，我们就利用 Vivado 的 package IP 功能将自定义的 VGA 模块打包，然后接入 AXI4 总线中，生成比特流。

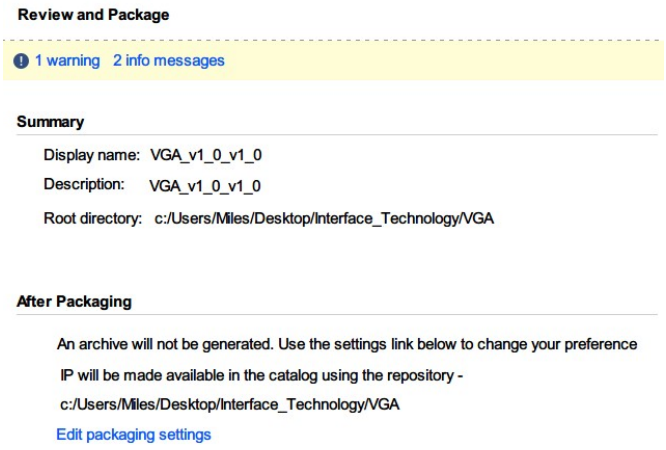


图 2.10 Package IP 示意

2.4 软件设计

软件层面的设计目标是，从 ADXL 加速度计中读出数据，写入 vga 模块的 slv_reg0。故软件要能够用 spi 与 ADXL 加速度计进行通信，同时将读到的数据写到与 vga 模块相关的正确位置。

我按照以下思路对软件进行设计：

1. 配置 spi core，设计基础的 spi 写函数，实现 spi 通信
2. 用 spi 写函数实现读写 ADXL 的函数，配置 ADXL 器件
3. 实现按次序将 ADXL 数据写入 vga 模块中的 main 函数

下面将也按此顺序对具体设计流程进行阐述：

- 首先用宏定义声明 SPI core 各寄存器的地址

```
wb_mux
  #(.num_slaves (7),
    .MATCH_ADDR ({32'h00000000, 32'h00001000, 32'h00001040, 32'h00001100, 32'h00001200, 32'h00001400, 32'h00002000}),
    .MATCH_MASK ({32'hfffff000, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hfffff000}))
```

图 2.11 从设备基地址

根据 wisbone 总线中各 slaves 地址的掩码和基址，可以指导 SPI core 组件的起始地址为 80001100，可变地址范围为地址低 6 位。

```
simple_spi spi(
    // Wishbone slave interface
    .clk_i   (wb_clk_i),
    .rst_i   (~wb_rst_i),
    .adr_i   (wb_adr_i[2]? 3'd0 : wb_adr_i[5:3]),
```

图 2.12 simple_spi 的地址输入

```
    begin
      if (adr_i == 3'b000)
        spcr <= dat_i | 8'h10; // always set master bit

      if (adr_i == 3'b011)
        sper <= dat_i;

      if (adr_i == 3'b100)
        ss_r <= dat_i[SS_WIDTH-1:0];
    end
```

图 2.13 simple_spi 的地址匹配

阅读 spi 的地址匹配逻辑，以 spcr 为例，输入到 simple_spi 模块的地址是 wb_adr_i[2]? 3'd0 : wb_adr_i[5:3]，当该地址匹配 3’b000 时就会写 spcr，故 spcr 的地址应该定义为 80001100~80001107 中的任一值，我选择的时 80001104。类似地得到其他寄存器的地址。同时阅读文档中 SPSR 各比特位的含义，定义相应的掩码。再用宏定义给出基础的读写寄存器的函数。

```
#define SPCR 0x80001104
#define SPSR 0x80001108
#define SPDR 0x80001110 //rfout
#define SPER 0x80001118
#define SS 0x80001120  //slave select
#define MASK_READ  0x1//Read FIFO Empty
#define MASK_WRITE 0x8 //Write FIFO Full
#define MASK_INT 0x80 //Interrupt flag
//The Serial Peripheral Interrupt Flag is set upon completion of a transfer block
#define DIGIT_BASE    0x80120000
#define D_UART_LSR_RBRE_BIT  (0x01)
#define M_UART_RD_REG_LSR()  (*((volatile unsigned int*)(D_UART_BASE_ADDRESS + (4*0x05) )))
#define READ(dir) (*(volatile unsigned *)dir)
#define WRITE(dir, value) { (*(volatile unsigned *)dir) = (value); }
```

图 2.13 软件中的宏定义

- 之后就可以配置 SPI core 了

阅读文档后进行如下配置：不使能中断，使能 SPI 核，让 SPI core 以模式 0（CPOL=0、且 CPHA=0）工作，因为推荐的 SPI 时钟频率范围为 1-5MHz，我们的时

钟频率为 50MHz，我们设置 SPI 的时钟为 clk 除以 16。最终得到下图所示初始化配置指令。

```
//init SPI core
WRITE(SPCR,0b01010010);
WRITE(SPER,0b00000000)
```

图 2.14 配置 SPI

- 给出 SPI 写函数

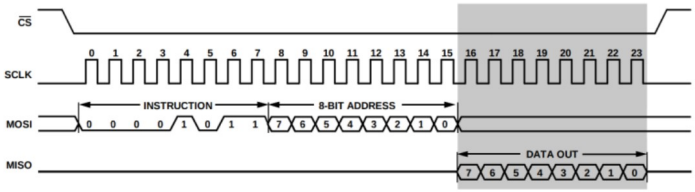
首先，将 SPSR 中的第 7 位中断标志位置 1，该位为 1 时标志着数据传输完成。之后我们可以向 SPDR 中写数据了。

```
void write_spi(unsigned char data)
{
    unsigned char recv;
    // internal clear interrupt flag
    recv = READ(SPSR);
    recv = recv | 0x80;
    WRITE(SPSR, recv);
    // send data
    WRITE(SPDR, data);
}
```

图 2.15 SPI 写函数

- 用 SPI 写函数实现 ADXL 的读写

参照实验指导中给的读写操作示意图



2 / 14

README.md 2023-10-12

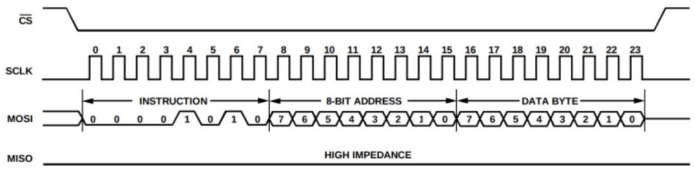


图 2.17 读写 ADXL 的过程示意图

每次通信时要将 CS 信号置为低电平，在通信结束后将 CS 信号恢复为高电平。且由于数据每个 clk 传递一位，所以要从寄存器中读到正确的数据，需要等待数据完全传输，也就是 SPSR 的中断位被置 1，写入同理。

以写函数为例

```
void write_ADXL(char addr,char value)
{
   WRITE(SS,0xFF); //CS down
   write_spi(0x0A);
   while((READ(SPSR)&MASK_INT)!= 0x80); //wait for interrupt
   write_spi(addr);
   while((READ(SPSR)&MASK_INT)!= 0x80); //wait for interrupt
   write_spi(value);
   while((READ(SPSR)&MASK_INT)!= 0x80); //wait for interrupt
   WRITE(SS,0x00); //CS up
}
```

图 2.18 ADXL 写函数

设置好 SS(slave select)信号之后，写入 0x0A 表示写操作，等待数据传输完毕，再依次写入地址和数据。

- 配置 ADXL

查看 ADXL 手册，我采用了 ADXL 三种模式中的 Measurement 模式，使其持续输出加速度数据。

```
write_ADXL(0x2D,0b00000010);
```

图 2.19 配置 ADXL 器件

- 实现按次序将 ADXL 数据写入 vga 模块中的 main 函数

查看文档中 ADXL 各数据的地址偏移量，发现 X，Y，Z 轴数据的高八位分别位于 08，09，0A，将他们循环读出，构造成 32 位的 displayData，直接写入 VGA_BASE（我们 VGA 模块分配的地址）中，即可正常显示。

```
while(1){
   xdata = read_ADXL(0x08);
   ydata = read_ADXL(0x09);
   zdata = read_ADXL(0x0A);
   printfNexys("Xdata = %d, Ydata = %d, Zdata = %d", xdata, ydata, zdata);
   displayData = xdata | (ydata << 8) | (zdata << 16);
   M_PSP_WRITE_REGISTER_32(DIGIT_BASE, displayData);
   M_PSP_WRITE_REGISTER_32(VGA_BASE, displayData);
   delay();
}
```

图 2.20 循环读加速度数据写入 VGA 显示模块

至此，软件设计完成。

2.5 综合应用系统的创新性

我们最终得到了一个能在 vga 显示屏中显示 x、y、z 轴加速度的演示系统。

虽然简单，但我们创新性地将 vga 和加速度计部件结合在一起。可以设想，如果我们将加速度计埋到地下，而能够在 vga 显示屏中实时观察加速度计的数据，对研究地底运动、测量重力、地震预警等活动都有帮助。

且该系统易于扩展，在软件层面只需要做到将加速度计中的数值数据读出并写入 vga 模块，即可显示。那么完全可以进一步，向该系统中加入显示温度传感器数据、显示光学传感器数据、显示声波传感器数据等功能。这种优秀的易于扩展性，允许应用者根据需要加入自己想要的模块，可以说蕴含了接口技术的真谛。

2.6 存在的问题及改进设想

存在的问题：

vga 显示模块还有些粗糙，我们设计的字母显示有些抽象派，且从某种程度上来说显示位置是固定的。同时显示的数据比较单一。

改进设想：

在 vga 显示模块中加入参数 parameter，使显示行号可变。

进一步优化 vga 显示的编码，使字母更加圆润。

加入更多可显示数据。

3 总结与心得

3.1 实验总结

经过对前面实验的学习，我们最终在 rvfpga_Soc 平台中加入了自己设计的 VGA 显示模块，实现了 ADXL 加速度计+VGA 显示的应用系统。

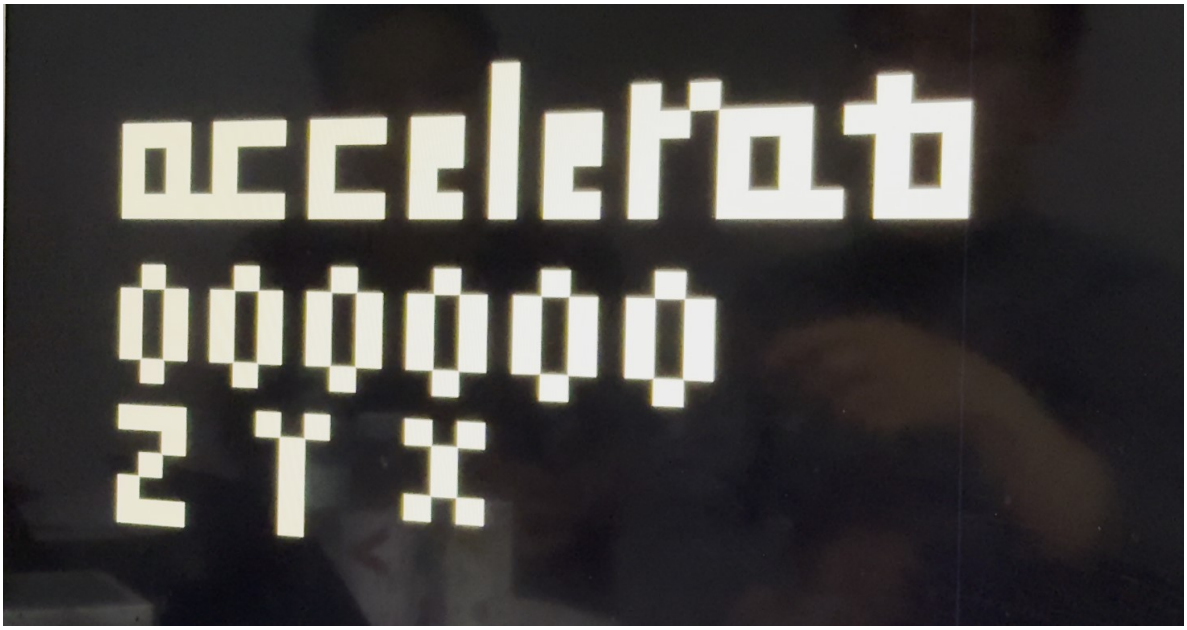


图 3.1 应用系统显示效果图

经过这次实验，我掌握了阅读 IP 文档，使用 IP，修改 IP 的步骤，能够理解复杂系统中如何和外部设备进行通信。总的来说，这门课有些晦涩难懂，但真正入门理解之后便能够用接口技术的通用方法解决各类问题。

3.2 实验心得

在 uart 串口实验中，简单的打印 hello world 第一次让我理解了接口技术的通用方法。在 uart_16550 的文档中费劲地寻找各类寄存器。参照 uartInit()函数理解 uart 串口的 start-up Routine，对于第一次接触类似实验的我来说是一次全新的挑战。这门实验，让我领会到了在实践过程中可扩展的重要性，一块小小的 rvFPGA 开发板，居然有这么多部件可以利用。

3.3 意见和建议

个人认为这门课的理论课和实验课结合的并不好，从实践中才能快速掌握解决接口问题的方法。且由于一开始有点摆，实验课上完时 lab5 还没做完，没有了老师的指导后面的实验做得很痛苦。当然像主从设备这样的核心概念还是要讲，所以我建议，适量减少理论课课时，增加实验课课时，将不是很必要的或者很难懂的先导知识放到实验文档中让同学们自学。

致谢

特别感谢 Imagination Community 对本次实验中的支持和贡献，正因为有这些业界公司无私的教学支持才能让我们这些学生在学习中走的更远。

一、原创性声明

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

特此声明！

作者签字: 刘文博

二、对课程实验的学术评语（教师填写）

三、对课程实验的评分（教师填写）

评分项目（分值）	工具应用（10 分）	实验过程（70 分）	报告撰写（20 分）	最终评定（100 分）
得分				

指导教师签字: _____