FPGA 提高实验　红外遥控系统

实验报告

姓名：赵文亮

学号：2016011452

班级：自 64

日期：2018 年 7 月 10 日

目录

1 实验内容

1.1 必做功能

设计一个以 NIOSII 软核微处理器和 Avalon 总线为核心的 SOPC 系统，实现用按键或红外遥控器输入数据，然后将结果显示在 LCD 上。要求设计红外遥控和 LCD 与 Avalon 总线接口的 IP 核。

1.2 附加功能

除了必做的功能，我还完成了一些附加功能，包括：

- 使用一个 LED 来表示电源。按下遥控器的电源键可以控制这个 LED 的亮灭。电源的开关状态可以实时地显示在 LCD 屏幕上。
- 使用一个 LED 来表示音量。调节音量上下键可以控制这个 LED 的亮度，LED 的亮度可以实时显示在 LCD 屏幕上。
- 使用一个 LED 来显示按键信号。当按下一次按键时，这个 LED 会闪烁一次。
- 使用八个 LED 来表示按键码。按下按键后，这八个 LED 的亮灭就表示了对应按键的键码。

2 设计方案

由于本实验要求构建一个基于 NIOSII 软核与 Avalon 总线的 SOPC 系统，我们需要将所有需要的外设全部挂在总线上。为此，需要设计 LCD 与红外接收模块的驱动电路以及与总线的接口电路。

将外设挂载在总线上后，便可以在 Eclipse 中使用 C 语言进行编程。通过对特定的地址进行读写操作，便通过总线向外设写入或从外设读出数据。例如，可以通过对总线上写入数据来控制 LCD 显示的位置、字符等；通过对总线读取数据可以获得 IR 模块输出的按键码等等。

红外接收模块在接收到有效按键后会发出一个 Ready 信号，我利用这个信号产生中断，并在中断服务程序里面完成对按键的读取、LCD 的显示以及其他的附加功能。

3 系统方框图

整个系统基于 NIOSII 软核与 Avalon 总线设计，系统的方框图如图 1 所示。其中，NIOSII 软核与其他的外设都挂载在总线上，并通过总线实现了相互的通信。

4 模块选择与参数设置

本系统采用的模块已经在图 1 中有所体现，下面详细介绍每个模块的功能及主要的参数

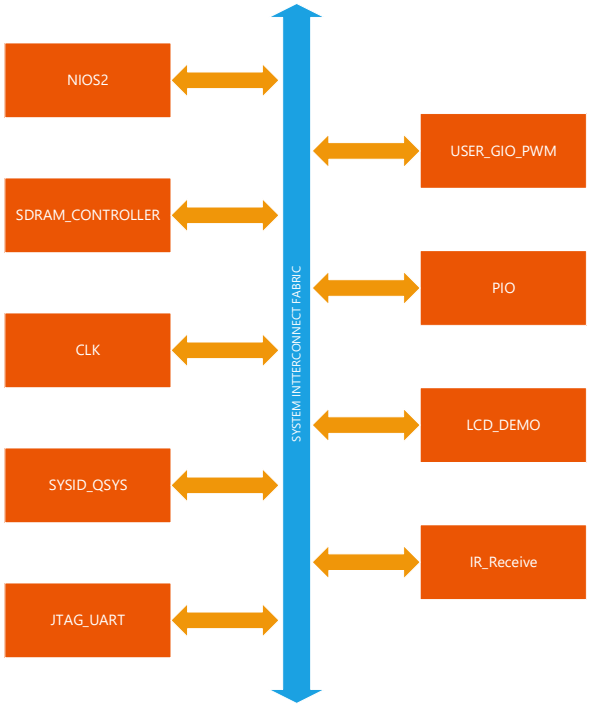

图 1: SOPC 系统方框图

名称	功能	主要参数
NIOS2	控制核心	
CLK	提供系统的时钟	$f = 100$MHz
SDRAM_CONTR	与 SDRAM 的接口，程序运行时将 SDRAM 作为内存	Data Width= 32
SYSID_QSYS	系统的标识号	ID=0x00000001
JTAG_UART	调试程序时与电脑通信	读写缓存区深度：64bytes
USER_GIO_PWM	自定义的 PWM 波发生器	
PIO	并行输入输出接口，可用来进行数据传输或产生中断	Width= 1
LCD_DEMO	与 LCD 的接口，用于在 LCD 上显示信息	
IR_RECEIVE	与红外接收模块的接口，用于接收按键信息	

5 流程图及实验代码

红外遥控系统的流程图如图 2 所示，下面将分模块分析实验的代码[1]。

相比于基础实验的 SOPC 系统，提高实验的系统多出了 LCD 和 IR 两个模块的驱动以及与总线的接口模块。

[1]完整代码见https://github.com/thu-jw/FPGA_Advanced

图 2: 红外遥控系统流程图

5.1 LCD 模块

LCD 模块的代码由老师提供，在此不再赘述，仅选取与总线接口中重要的部分加以分析。首先是通过总线的写操作：

```
            if((avs_chipselect == 1) &&(avs_write == 1))
                begin       //write data
// 2017.07.08  After verification, in Quartus15.0, the use of avs_address,
//do not need to remove the low two bit, direct use can be.
                    APB_DATA_Buf[avs_address[4:0]]<=avs_writedata[31:0];
                    //Store ARM Command to Buff .
                    if(avs_address[4:0]==5'b00000)
                            begin
                                Write_Position<=avs_writedata[31:24];
                                //Position range from 0~15
                                Write_Length<=avs_writedata[23:16];
                                Write_Row<=avs_writedata[15:8];
                                Write_Command<=avs_writedata[7:0];
                                Avalon_Write_Buff<=1'b1;
                            end
                        else
                                    Avalon_Write_Buff<=1'b0;
                end
```

可见，当地址为 0 时，写操作将位置、长度、行数、命令等信息写入；而当地址不为零时，写操作将显示内容写入缓存区。其中，命令 0x33 表示将数据写入 LCD 中：

```
                    else if(Write_Command==8'h33)  //Write Data LCD1602 .
                        mLCD_ST   <=  9;
```

再来分析读操作：

```
always @(posedge csi_CLK or negedge csi_RST_N)
 begin
        if(!csi_RST_N) begin

        end
        else
```

```
        begin
                if((avs_chipselect == 1) &&(avs_read == 1))
                begin
                if(avs_address[4:0]==5'b00000)
                        avs_readdata<={8'h00,Write_Command, 10'h000,mLCD_ST} ;
                             else
                              avs_readdata<= APB_DATA_Buf[avs_address[4:0]];
                end
        end
end
```

可见执行读操作时，会将命令、LCD 的状态读入总线。根据这些接口的代码即可知道，在 SOPC 系统搭建完毕后该如何使用 C 语言进行编程。

```
void LCD_Disp(unsigned char Row, unsigned char position , unsigned char *pData , unsigned char len)
{
    unsigned char i;
    unsigned char busy;
    unsigned char Row_Line;
    //判断写入行位置是否正确
    if(Row==1)
        Row_Line=0x00;
    else if(Row==2)
        Row_Line=0x40;
    else
        Row_Line=0x00;

    //检查写入位置是否越界
    if(position<0)
        position=0;
    else if(position>15)
        position=15;
    //检查单行写入是否会超过最大显示个数
    if((len+position)>15)
        len=15-position;
    //将待显示数据写入缓存数据
    for(i=0;i<len;i++)
    {
        *(pUser_LCD+i+1)=*(pData+i); //写入地址从1开始
    }
    //检查写入模块当前是否忙
    do{
        busy=*(pUser_LCD) & 0xf ;
      }
    while(busy!=5);
    //写入使能指令
    *(pUser_LCD)=(position<<24)+(len<<16)+(Row_Line<<8)+0x33;
}
```

其中 23-26 行即为将数据写入缓存区；28-31 行通过读操作获取当前 LCD 的状态，并检查是否正忙；33 行将位置、长度、行数等信息写入，并通过写入命令 0x33 来启动显示。

5.2 IR 模块

IR 模块中的驱动电路较为复杂，实现了将串行的红外数据转化为并行的数据。这一部分代码由老师提供，这里我们只关注它的接口：

```
1  module IR_RECEIVE(
2                        iCLK,          //clk 50MHz
3                        iRST_n,        //reset
4                        iIRDA,         //IR code input
5                        oDATA_READY,   //data ready
6                        oDATA          //decode data output
7                        );
```

根据这些接口，即可设计出将其挂载在总线上的电路。首先由于系统的时钟为 100MHz，需要将其分频至 50MHz 才能被 IR_RECEIVE 模块使用：

```
1  //CLOCK_50
2  always@(posedge csi_clk or negedge csi_reset_n)
3  begin
4      if (!csi_reset_n)
5          CLOCK_50 <= 0;
6      else
7          CLOCK_50 <= ~CLOCK_50;
8  end
```

我使用一个变量 mDataReadyDetect 来保存 DataReady 的数据，并通过它来检测 DataReady 信号的上升沿，当发现有上升沿时，就将 mData 中的数据读入缓存区：

```
1  //store data
2  always@(posedge csi_clk or negedge csi_reset_n)
3  begin
4      if (!csi_reset_n)
5          mDataReadyDetect <= 2'b00;
6      else
7      begin
8          mDataReadyDetect <= {mDataReady, mDataReadyDetect[2], mDataReadyDetect[1],};
9          if (mDataReadyDetect == 3'b110)//posedge
10             mDataBuf <= mData;
11     end
12 end
```

最后是读总线的操作，我将缓存区中的数据和 mDataReady 信号一并读入总线中。这里注意到 mDataBuf 的格式中，只有 23-16 为是有用的按键码，故将 mDataReady 信号放在总线的最高位并不会损失信息量。

```
1  //readdata
2  always@(posedge csi_clk or negedge csi_reset_n)
```

```
3  begin
4      if (!csi_reset_n)
5          avs_readdata <= 32'b0;
6      else if ((avs_chipselect == 1) && (avs_read == 1))
7      begin
8          avs_readdata <= {mDataReady, mDataBuf[30:0]};
9      end
10 end
```

5.3 C 语言部分

整个 SOPC 系统搭建完毕后，便可以在 Eclipse 中进行 C 语言的编程。我首先为红外接收莫魁岸编写了函数 char* getKeyName(unsigned int key_code);。该函数根据按键码返回按键名称。在主程序里，我首先注册了一个 PIO 中断，当 ready 信号到来时转入中断服务程序。配置中断的代码如下：

```
1  void ready_config(){
2      IOWR_ALTERA_AVALON_PIO_IRQ_MASK(PIO_0_BASE, 0x1);
3      IOWR_ALTERA_AVALON_PIO_EDGE_CAP(PIO_0_BASE, 0x0);
4      alt_ic_isr_register(PIO_0_IRQ_INTERRUPT_CONTROLLER_ID, PIO_0_IRQ, ready_handler, NULL, 0x0);
5  }
```

在中断服务程序中，我首先通过key = *pIR_RECEIVE; 通过总线对 IR 模块极进行读取。其中 pIR_RECEIVE 的定义为：unsigned int *pIR_RECEIVE = IR_RECEIVE_0_BASE;。接下来便可以通过调用 getKeyName 来获得按键名称。至此，关键的技术都已经实现，接下来的关于 LED 的亮灭（使用 PIO）、LED 的亮度调节（使用 PWM）等都可以通过简单的 C 语言编程来实现，故不再赘述。

6 实验结果

实验结果的截图如图 3 所示。图中正展示按下音量下键，将左数第二个 LED 灯的亮度调暗。最左面的 LED 显示当前电源处于开启状态。LCD 显示屏上实时显示按键的名称以及电源、音量等信息。

7 实验中遇到的问题及解决方法

本次实验主要遇到了两个问题。第一个问题是每次我修改系统后，都会重新生成一个 sopcinfo 文件，这时再重开 Eclipse 后将无法下载程序。这是因为 sopcinfo 中会保存系统的 id 号和时间戳，当检测到硬件中的信息与 sopcinfo 文件中的不符时会拒绝下载，为此需要更新 sopcinfo 文件。一开始我没有找到更新的办法，于是每次都是重新建立一个工程，十分浪费时间。后来我知道了可以在现有的工程上更新，方法如下：在 Eclipse 中选中 BSP 的工程，再进入 NIOS II->BSP Editor，重新 Generate 即可。

另一个问题是我的 IR Module 的实现问题。我第一次使用了 2 位的 mDataReadyDetect 来检测 Ready 的上升沿。完成系统后，发现每次都需要按两下按键才能够收到按键码。我于是将 IR_Driver 输出的按键码以及我自定义的 Buffer 中的按键码引出到 FPGA 上的 LED 上进行调试。接着我发现，其根本原因在于，

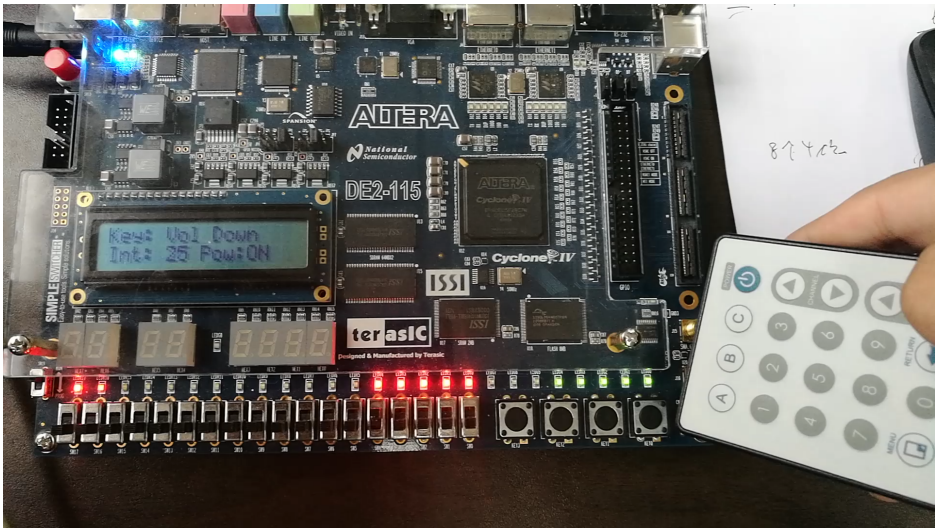

图 3: FPGA 提高实验效果图

Ready 信号到来时，新的按键并没有准备好，而是需要再等一个时钟周期。于是我将 mDataReadyDetect 修改位 3 位，便顺利第解决了这个问题。

8 体会、收获与建议

在本次实验中，我更加熟悉了基于 NIOSII 软核的 SOPC 系统的设计，在独立完成整个实验的过程中，我收获到了满满的成就感。能够顺利完成这个实验，我认为很大程度上是因为本实验的设计和参考资料的提供做的非常出色，在这里先感谢老师和助教的辛苦付出。

SOPC 系统的设计是我们之前从来没有接触过的。第一次听说要自己在 FPGA 上搭建一个系统时，我是有点担心的，尤其是看到长达 89 页的教程。我花了很长的时间照着教程一步一步地完成基础实验，生怕哪一步出错，做完基础实验之后也是似懂非懂。面对提高实验，我自然有一点不知所措。

还好我提前做好了规划，找到了一条可能通向成功的道路。由于老师给了我们 LCD 显示屏的完整的 Verilog 和 C 语言代码，以及 C 语言编程的方法。我首先按照这个步骤将 LCD 挂在总线上，通过这个过程，对 NIOSII 系统的构成更加熟悉了，也开始真正明白了在 C 语言里的指针操作与总线上的读写操作之间的对应关系。接着，我根据 IR_RECEIVE 的代码，与 LCD 和 User_Demo 中的代码对比，逐渐熟悉了变量命名和代码编写的规范，例如 avs、csi、coe 等前缀的含义以及读写的 always 块该如何编写。有了这些基础，剩下的就是举一反三。可以说我是借助老师和助教搭建的台阶，一步一步完成了实验。再次为这个实验设计的循循善诱而点赞！

一个小建议：遥控器中的电池长时间使用后会没电，建议在实验室中准备一些纽扣电池。