

华中科技大学

2018

计算机系统能力培养综合实践之 蓝牙小车 课程设计报告

题 目: _____
专 业: _____
班 级: _____
学 号: _____
姓 名: _____
同组成员: _____



计算机科学与技术学院

目 录

1 课程设计概述	1
1.1 课程设计目的	1
1.2 课程设计任务	1
1.3 课程设计的要求	1
2 系统总体设计	3
2.1 硬件部分设计	3
2.2 软件部分设计	3
3 硬件平台设计与实现	5
3.1 设计目的	5
3.2 实验步骤	5
3.3 故障与调试	6
4 操作系统设计与实现	7
4.1 设计目的	7
4.2 实验步骤	7
4.3 故障与调试	11
5 应用设计与实现	12
5.1 设计目的	12
5.2 实验步骤	12
5.3 故障与调试	18
5.4 存在的不足和缺陷	19
6 总结与心得	20
6.1 实验总结	20
6.2 实验心得	21

华中科技大学课程设计报告

6.3 意见与建议·····	21
致谢·····	22
参考文献·····	23

1 课程设计概述

1.1 课程设计目的

实现一个 MIPSfpga 处理器硬件系统与 Hos 操作系统综合的应用蓝牙小车，通过整个系统的实现，增强学生的系统能力，从底层硬件到应用的系统整体观。

学习、了解有关 AXI 总线结构的基本工作原理，设计 AXI 接口外设和可支持操作系统的 MIPSfpga 和自定义 PMOD 模块；在开发的 MIPSfpga 上运行小型的 Hos 操作系统，并在小型操作系统上进行应用层面的开发，完成相关的任务。

1.2 课程设计任务

主要分为三个部分：硬件（MIPSfpga 处理器）设计，操作系统（Hos）设计，应用（蓝牙小车）设计。

硬件设计部分要求在 Nexys4 DDR FPGA 开发板上搭建一个基于 MIPS 的处理器（软核），并在此基础上设计处理器的外围设备，包括 UART、COM 接口、蓝牙设备等；操作系统设计在清华大学的 ucore 操作系统改造简化而来，熟悉该小型操作系统，了解编译和运行的相关原理，根据源代码和相关编译的流程了解操作系统的具体实现，通过添加相关应用和系统调用实现一定的功能；应用设计基于前两个部分，通过添加相关硬件软件模块，完成一个蓝牙控制的小车，并实现其他额外的功能。

1.3 课程设计要求

- （1）基于 MIPSfpga 的硬件平台搭建及其测试
- （2）Hos 开发调试环境搭建和使用
- （3）Hos 操作系统的构建与运行

华中科技大学课程设计报告

- (4) Hos 操作系统应用开发和系统调用
- (5) 基于 MIPSfpga 平台的蓝牙小车硬件设计与实现
- (6) 蓝牙小车操作系统、驱动及应用软件设计开发
- (7) 蓝牙小车应用扩展和创新
- (8) 成果验收和检查
- (9) 总结实践过程，完成实验报告撰写

2 系统总体设计

2.1 硬件部分设计

2.1.1 MIPSfpga 处理器

利用 Vivado 平台提供的 IP 模块进行基于 AXI 接口的 MIPSfpga 处理器搭建，其中 IP 模块包括 MIPS MicroAptiv UP、AHB-Lite to AXI Bridge、AXI Interconnect、AXI GPIO、AXI BRAM Controller 和 Block Memory Generator 等，进行相应的连接和设置，可以搭建出一个时钟频率为 50MHZ、物理内存有 128MB 的简单 MIPS 处理器系统，为后续设计接口提供硬件基础。

2.1.2 蓝牙和小车马达

蓝牙外设与处理器进行数据传输是通过串口，所以需要在前述处理器系统中添加 UART 的 IP 核模块即可。

小车马达通过查阅马达驱动板 L293D 的资料与 PMOD 接口协议原理后，则采用 4 个 32 位寄存器的值进行速度和方向的控制。

2.1.3 摄像头

通过查阅摄像头 OV5640 的资料后，得知其通讯协议为 SCCB，一种非标准的 I²C 协议，则采用 Vivado 原有 I²C 的 IP 模块进行数据的通信，并通过设置直连线，打算直接利用摄像头的 RGB565 格式输出到 VGA 进行显示。

2.2 软件部分设计

2.2.1 蓝牙驱动

蓝牙驱动模块通过控制串口的读写进行蓝牙的控制，首先通过设置相应寄存器

的值初始化蓝牙，然后从寄存器中读取相应数据进行处理即可。

2.2.2 马达驱动

马达驱动则是利用蓝牙模块中获取的数据进行解码分析，然后通过每个寄存器的值控制每个正反转及其马达的 01 占空比即电压，实现驱动轮子的功能。

2.2.3 摄像头驱动

利用 I²C 协议往对应的寄存器进行初始化设置，然后根据其手册配置寄存器将其设置为 video 模式，进行实时图像的数据的输出。

2.2.4 Android APP——六角龙

安卓手机应用的开发为的是能方便地与小车的蓝牙进行通信交互，从而实现两大目标：①通过手机控制小车的四轮驱动，包括控制车轮的运行速度和方向；②用手机来接收小车蓝牙传来的摄像头数据，并将其以图片的形式显示出来。

3 硬件平台设计与实现

3.1 设计目的

Vivado 开发环境的使用，学习了解 Openocd 和其 JTAG 对 MIPSFPGA 硬件平台调试管理的工作原理，了解 MIPS 交叉编译环境提供的不同机器平台源码到 MIPS 机器的编译原理。学习、理解、实践 Vivado IP 开发的流程，掌握 IP 封装、使用、连接的基本方法。同时学习、了解有关 AXI 总线结构的基本工作原理，同时完成一个基本可运行的 MIPSfpga。

3.2 实验步骤

3.2.1 硬件平台搭建的实践准备

搭建基于 Vivado 硬件开发环境、Openocd 嵌入式调试工具、MIPS mti 交叉编译环境、GNU make 工具的开发环境、putty 串口终端的 MIPSfpga 硬件系统。将提供的 .bit 文件通过烧写到 FPGA 开发板，并观察其运行。通过搭建的交叉编译器和 make 工具编译裸机主程序，将二进制文件下载到烧写好 .bit 文件的 FPGA 开发板并观察其运行。

3.2.2 基于 MIPSfpga 的简单硬件平台搭建

通过动手构建一个由 Vivado IP 开发方式实现 FPGA 上的跑马灯，并对其进行测试。通过 Vivado IP 开发的模式搭建一个使用 AXI 总线及外设模块的 MIPSfpga 简单硬件平台，并编写 MIPS 汇编程序或 C 语言程序对其进行测试。

3.2.3 自定制接口模块的设计

编写并封装一个基于 AXI 总线接口标准的跑马灯外设模块。实现一个包含上述跑马灯外设模块的 MIPSfpga 并将其烧写到开发板上。编写 MIPS 汇编程序或

C 语言程序对这个新添加的自定义外设进行的操作演示。

3.2.4 中断处理

在简单 MIPSfpga 处理器硬件平台上实现 AXI 总线外设中断处理。在 MIPSfpga 中将 uart 和 PWM 的中断信息连接到 CPU 的中断入口，然后在 C 和 MIPS 汇编程序中编写中断服务代码，对 MIPSfpga 硬件平台的硬件中断进行测试。

3.3 故障与调试

在实验环境 Vivado2015.2 版本中，进行 Memory Interface Generator (MIG) IP 的配置时，打不开该 IP 核的配置界面，用管理员模式打开 Vivado 仍然打不开该配置界面。尝试过用管理员模式去重装 Vivado，但还是一样的结果。所以后面的实验只能基于已经配置好的 MIPS 处理器进行添加。

4 操作系统设计与实现

4.1 设计目的

学习在自己的个人电脑上安装构建（build）Hos 操作系统的环境，将所生成的镜像下载到 MIPSfpga 开发板、并将其运行起来的方法；安装VScode，能对Hos操作系统的源代码进行阅读；熟悉操作系统的原理，能够完成一定的操作系统源代码的阅读、修改，达到所要求的预期功能。

4.2 实验步骤

4.2.1 Hos 操作系统的构建与运行

安装 Hos 操作系统的构建涉及到的软件工具：Cygwin、交叉编译器（mips-sde）、Putty、Vivado、OpenOCD 等。

运行 Cygwin，并进入 Hos 源代码所在的目录，然后进行 make。如图 4.1 所示。

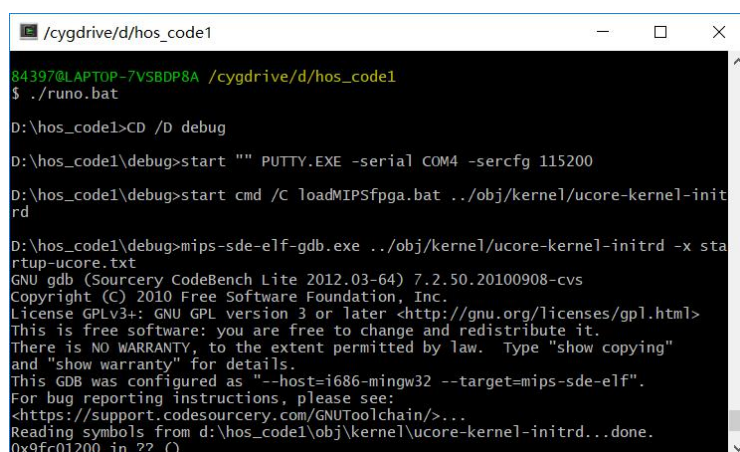


```
84397@LAPTOP-7VSBDP8A /cygdrive/d/hos_code1
$ make
mkdir -p D:/hos_code1/obj
make CC=gcc -f D:/hos_code1/tool/Makefile -C D:/hos_code1/tool all
gcc -Wall -O2 -D_FILE_OFFSET_BITS=64 -o D:/hos_code1/obj/mksfs mksfs.c
mksfs.c:33:0: 警告: "static_assert"重定义
#define static_assert(x)
\
In file included from mksfs.c:13:0:
/usr/include/assert.h:45:0: 附注: 这是先前定义的位置
# define static_assert _Static_assert
```

图 4.1 Hos 操作系统目录下 make

make 完成后，将构建好的操作系统在之前实现的 MIPSfpga 上运行。为了与调试信息的 run.bat 文件进行区别，则使用 runo.bat 文件，其已经完成了对运行操作系统所需要的所有动作的编辑，所以在当前目录下直接运行 runo.bat 文件，如图 4.2 所示。

华中科技大学课程实验报告



```
/cygdrive/d/hos_code1
84397@LAPTOP-7VSBP8A /cygdrive/d/hos_code1
$ ./runo.bat

D:\hos_code1>CD /D debug

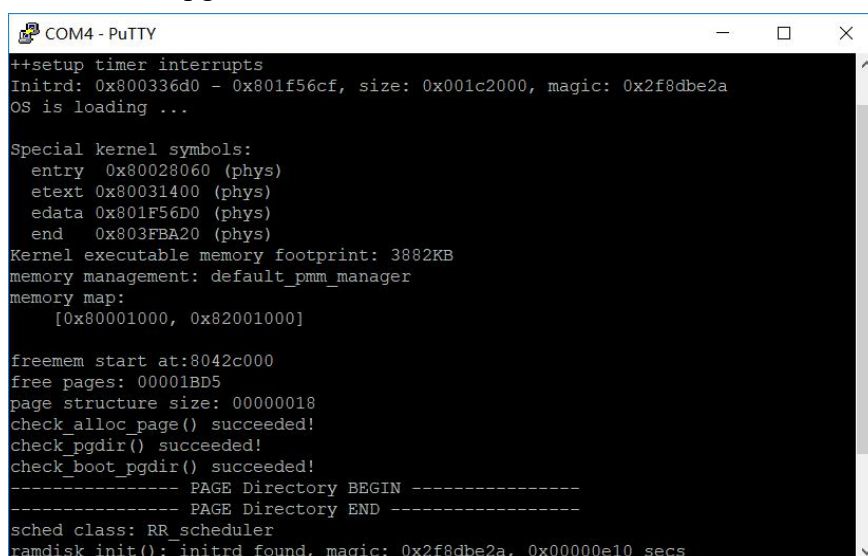
D:\hos_code1\debug>start "" PUTTY.EXE -serial COM4 -sercfg 115200

D:\hos_code1\debug>start cmd /C loadMIPSfpga.bat ../obj/kernel/ucore-kernel-initrd

D:\hos_code1\debug>mips-sde-elf-gdb.exe ../obj/kernel/ucore-kernel-initrd -x startup-ucore.txt
GNU gdb (Sourcery CodeBench Lite 2012.03-64) 7.2.50.20100908-cvs
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-mingw32 --target=mips-sde-elf".
For bug reporting instructions, please see:
<https://support.codesourcery.com/GNUToolchain/>...
Reading symbols from d:\hos_code1\obj\kernel\ucore-kernel-initrd...done.
0x9fc01200 in ?? ()
```

图 4.2 运行 runo.bat

在运行成功之后，可观察 Putty 的窗口中的输出如图 4.3 所示。此时说明 Hos 已经成功运行在 MIPSfpga 上。



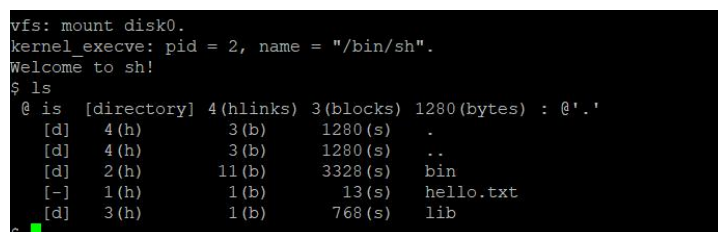
```
COM4 - PuTTY
++setup timer interrupts
Initrd: 0x800336d0 - 0x801f56cf, size: 0x001c2000, magic: 0x2f8dbe2a
OS is loading ...

Special kernel symbols:
entry 0x80028060 (phys)
etext 0x80031400 (phys)
edata 0x801f56d0 (phys)
end 0x803fba20 (phys)
Kernel executable memory footprint: 3882KB
memory management: default_pmm_manager
memory map:
[0x80001000, 0x82001000]

freemem start at:8042c000
free pages: 00001BD5
page structure size: 00000018
check_alloc_page() succeeded!
check_pgdir() succeeded!
check_boot_pgdir() succeeded!
----- PAGE Directory BEGIN -----
----- PAGE Directory END -----
sched class: RR_scheduler
ramdisk_init(): initrd found, magic: 0x2f8dbe2a, 0x00000e10 secs
```

图 4.3 运行 Hos 之后 Putty 窗口的输出

可以执行 ls 命令查看系统中的数据，如图 4.4 所示。



```
vfs: mount disk0.
kernel_execve: pid = 2, name = "/bin/sh".
Welcome to sh!
$ ls
@ is [directory] 4(hlinks) 3(blocks) 1280(bytes) : @'.'
[d] 4(h) 3(b) 1280(s) .
[d] 4(h) 3(b) 1280(s) ..
[d] 2(h) 11(b) 3328(s) bin
[-] 1(h) 1(b) 13(s) hello.txt
[d] 3(h) 1(b) 768(s) lib
```

图 4.4 系统中执行 ls

4.2.2 在 Hos 中添加应用

a) 添加系统内核调用

根据实验要求，添加系统内核调用，首先在 `kern-ucore\include\lib\unistd.h` 添加系统调用编号，如图 4.5 所示。

```
1 #define SYS_pipe      140
2 #define SYS_mkfifo    141
3
4 #define SYS_free      151
5 #define SYS_hello     152
6
7 #define SYS_mount     153
```

图 4.5 添加系统调用编号

在 `kern-ucore\syscall.c` 添加函数 `sys_hello` 和 `sys_free`，如图 4.6 和图 4.7 所示；然后在 `uint32_t(*syscalls[])(uint32_t arg[])` 函数中添加函数入口。

```
static uint32_t
sys_hello(uint32_t arg[]) {
    kprintf("kernel:hello world!\n\r");
    return 0;
}
```

图 4.6 添加 sys_hello 函数

```
static uint32_t
sys_free(uint32_t arg[]) {
    int num, res[50], i=0;
    kprintf("Free ages: ");
    num=nr_free_pages();
    while(num)
    {
        res[i++]=num%10;
        num/=10;
    }
    while(--i)
    {
        kprintf("%d", res[i]);
    }
    kprintf("%d", res[0]);
    kprintf("\n\r");
    return 0;
}
```

图 4.7 添加 sys_free 函数

其中 `sys_free` 是通过阅读系统内核代码来获取相应的空闲页数并将该数字转换成可以输出的数字。

b) 添加用户调用

华中科技大学课程实验报告

然后添加用户调用。在 user\include\unistd.h 添加系统调用编号，如图 4.8 所示。

```
1 #define SYS_pipe      140
2 #define SYS_mkfifo    141
3
4 #define SYS_free       151
5 #define SYS_hello      152
6
7 #define SYS_mount      153
```

图 4.8 添加用户调用

在 user\include\syscall.h 添加函数声明，如所示；在 user\syscall.c 中添加函数，如图 4.9 和图 4.10 所示。

```
int sys_pipe(int *fd_store);
int sys_mkfifo(const char *name, uin
int sys_hello();
int sys_free();

int sys_mount(const char *source, co
```

图 4.9 添加函数声明

```
int sys_hello()
{
    return syscall(SYS_hello);
}

int sys_free()
{
    return syscall(SYS_free);
}
```

图 4.10 添加函数 sys_free 和 sys_hello

c) 添加用户可执行程序

在 user\user-ucore 文件夹下新建文件 hello.c，代码如图 4.11 所示；新建文件 free.c，代码如图 4.12 所示。

```
#include <ulib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    sys_hello();
    return 0;
}
```

图 4.11 hello.c 文件内容

```
#include <ulib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    sys_free();
    return 0;
}
```

图 4.12 free.c 文件内容

修改文件在 user\user-ucore\Makefile，在 USER_APPLIST 中添加新的应用名称 hello 和 free，如图 4.13 所示。

```
USER_APPLIST:= pwd cat sh ls cp echo num hello free #link mkdir rename unlink lsmod insmod rmdir modprobe
ifneq ($(UCORE_TEST),)
USER_TESTLIST := $(basename $(wildcard tests/*.c))
USER_TESTLIST += $(basename $(wildcard tests/*.c))
endif
```

图 4.13 添加应用名称

4.3 故障与调试

在构建 Hos 镜像的时候遇到如图 4.14 的错误，根据错误提示，是表明缺少相应的 mips-sde-elf-gcc 命令，则经过询问老师，得到相应环境包，设置好系统变量后错误消除。

```
$ make
mkdir -p E:/Hos/hos-mips-master/hos-mips/obj
make -f E:/Hos/hos-mips-master/hos-mips/user/Makefile -C E:/Hos/hos-mips-master/hos-mips/user all
"mips-sde-elf-gcc -D__ASSEMBLY__ -I. -Iinclude -Ilib -nostdinc -nostdlib -fno-builtin -g -fno-builtin -nostdlib -nostdinc -EL -GO -Wformat -O0 -mno-mips16 -msoft-float -march=m14k -DMACH_FPGA -msoft-float -c -o E:/Hos/hos-mips-master/hos-mips/obj/user/initcode.o initcode.S
/bin/sh: mips-sde-elf-gcc: 未找到命令
make[1]: *** [E:/Hos/hos-mips-master/hos-mips/user/Makefile:39: E:/Hos/hos-mips-master/hos-mips/obj/user/initcode.o] 错误 127
make: *** [Makefile:81: userlib] 错误 2
```

图 4.14 make 出错

5 应用设计与实现

5.1 设计目的

设计并实现以 UART 串口协议为基础的蓝牙芯片总线外设 AXI 总线接口模块。设计并实现以 PMOD 接口协议为基础的 L293D 驱动板总线外设 AXI 总线接口模块。将这 2 个外设接口添加到 MIPSfpga 上,进行综合布线,并烧写到 N4 ddr 开发板上。利用 MIPS sde 交叉编译器编写一个程序下载到 MIPSfpga 中验证并测试以上两个模块的正确性。

了解关于无线蓝牙外设的基本工作原理和 L293D 驱动板基本工作原理,以便更好的实现其外设接口的设计。我在其中参与了除了安卓设计实现以外的大部分事情,其中主要负责的是马达驱动的硬件部分的改进和后面摄像头驱动的实现。

5.2 实验步骤

5.2.1 马达驱动模块改进 IP 核实现

在实现旧的 IP 核中,我们通过查阅相应的文档已经理解了马达驱动板芯片的工作原理,明白了 L293D 的控制轮子旋转方向和速度的机理,则我们打算进行“无极变速”的改进,即要想获得更多的档位变化。

首先设计整体的 IP 核,如图 5.1。

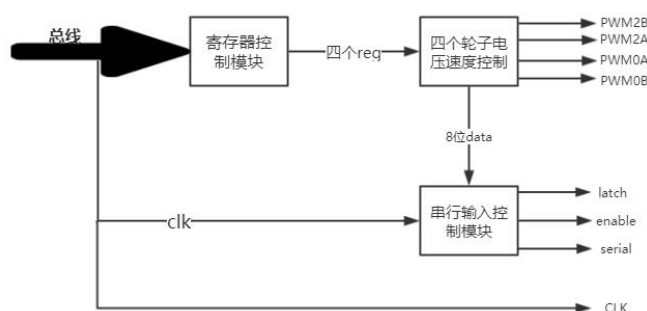


图 5.1 驱动模块图

华中科技大学课程实验报告

在上面模块图中可以看出，首先是从总线传入 4 个寄存器的值，然后经过速度方向模块的解析之后，则有 8 位控制方向的输出到串行传输的模块中输出，还有控制占空比的四个 PWM 值。

下面是解析 4 个寄存器的值，旧马达驱动 IP 核是通过一个寄存器，即 32 位 bit 的数据进行传输的，8 位方向，24 位速度。新的 IP 核采用的四个轮子的方向同样是 8 位，不同的是这 8 位根据每个寄存器的值的正负来判断前进还是后退，那么这就要知道其 PWM 值对应的轮子，通过控制不同的 PWM 的输出得出 PWM 的对应表。

表 5.1 PWM——轮子对应表

轮子	PWM
左前	PWM0A
右前	PWM0B
左后	PWM2B
右后	PWM2A

至于速度方向控制模块，输入为时钟 clk 以及四个寄存器数据。首先设置计数器，每个时钟自增 1，范围为 int，当绝对值大于 2147483647 时 counter 重新赋值 0。其某个 PWM 值根据寄存器数据判断输出，当某个轮子对应的数值大于 counter 时 PWM 值为 1，小于或者等于则为 0，这样就能在一段时间内控制其占空比，以 counter 为整体时间段，这样的想法就是有一个整型数据范围的档位。举个例子，当控制左前轮的寄存器值为 500 时，counter 从 0~500 计数时 PWM0A 输出为 1，从 501~231-1 的计数时 PWM0A 就输出 0，01 占空比就为 2147483147:500，更精细的控制了占空比。

但这就引发了其中 counter 上限值过大的问题，如果其过大时，就像上面的 0 的时间过大，提供的马达电压达不到能够启动轮子的效果，而且在某一段数值时，还会出现卡顿的小车现象。经过测试，我们确定最后 counter 的上限为 500，并且要

华中科技大学课程实验报告

最低的数值要给到 325 左右才能使得马达电压足够。则这就给到软件驱动对于速度和方向的控制要更为精细，对应的实现要更为复杂和困难。

在测试过程中，整体框架与旧 IP 基本一致，但首先是多了三个寄存器的地址值，可以根据相对地址进行赋值。再有就是解析数值和控制转向方面，则由同组的胡浩同学详细负责。

封装好新的 IP 核后，将原来的 IP 和删除，并且将新的 IP 和连接，其连接后的图，如图 5.2 所示。

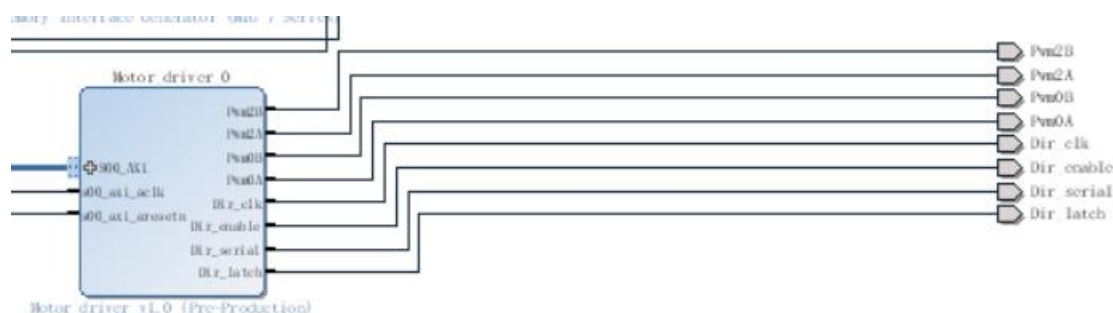


图 5.2 新 IP 核模块连接

5.2.2 摄像头模块

摄像头部分我们根据查阅到的资料，其功能框图如图 5.3 所示。

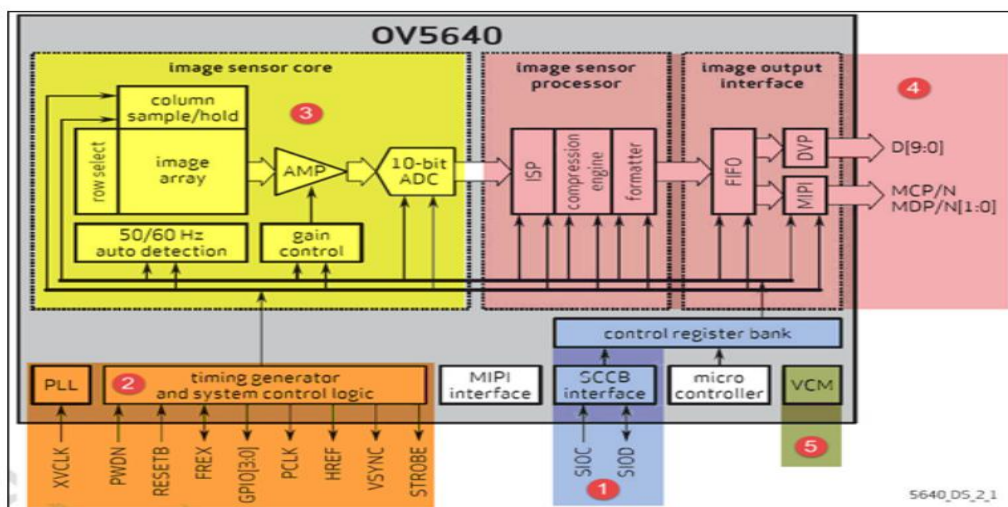


图 5.3 OV5640 功能框图

华中科技大学课程实验报告

引脚示意如表 5.2 所示。

表 5.2 OV5640 管脚

管脚名称	管脚类型	管脚描述
SIO_C	输入	SCCB总线的时钟线，可类比I2C的SCL
SIO_D	I/O	SCCB总线的数据线，可类比I2C的SDA
RESET	输入	系统复位管脚，低电平有效
PWDN	输入	掉电/省电模式，高电平有效
HREF	输出	行同步信号
VSYNC	输出	帧同步信号
PCLK	输出	像素同步时钟输出信号
XCLK	输入	外部时钟输入端口，可接外部晶振
Y2...Y9	输出	像素数据输出端口

在对应的功能框图中，标号①处的是 OV5640 的控制寄存器，它根据这些寄存器配置的参数来运行，而这些参数是由外部控制器通过 SIO_C 和 SIO_D 引脚写入的，SIO_C 与 SIO_D 使用的通讯协议跟 I2C 十分类似，详细说明在后文会出现。

标号②处包含了 OV5640 的通信、控制信号及外部时钟，其中 PCLK、HREF 及 VSYNC 分别是像素同步时钟、行同步信号以及帧同步信号，这与液晶屏控制中的信号是很类似的。RESETB 引脚为低电平时，用于复位整个传感器芯片，PWDN 用于控制芯片进入低功耗模式。注意到 XCLK 引脚，它跟 PCLK 是完全不同的，XCLK 是用于驱动整个传感器芯片的时钟信号，是外部输入到 OV5640 的信号；而 PCLK 是 OV5640 输出数据时的同步信号，它是由 OV5640 输出的信号。

标号③处的是感光矩阵，光信号在这里转化成电信号，经过各种处理，这些信号存储成由一个个像素点表示的数字图像。

华中科技大学课程实验报告

标号④处包含了 DSP 处理单元,它会根据控制寄存器的配置做一些基本的图像处理运算。这部分还包含了图像格式转换单元及压缩单元,转换出的数据最终通过 Y0-Y9 引脚输出,一般来说我们使用 8 根数据线来传输,这时仅使用 Y2-Y9 引脚,其对应的板子上的是 D0-D7。

标号⑤处为 VCM 处理单元,他会通过图像分析来实现图像的自动对焦功能。

当然其中我们主要关注的就是标号①②④的部分,因为这是关系到数据传输的,也就是实现的部分。根据上述引脚的定义,我们可以将其绑定引脚并打算将画面通过 VGA 直接输出,如图 5.4 所示。

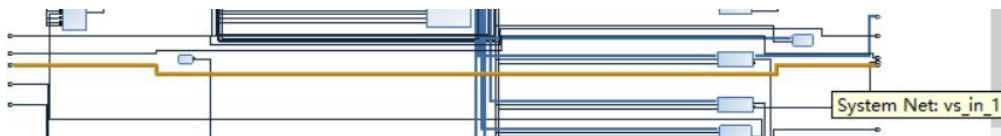


图 5.4 VS 直连输出

其连接与约束引脚绑定如图 5.5 所示。

```
set_property -dict { PACKAGE_PIN A14 IOSTANDARD LVCMOS33 } [get_ports { sencer_scl_io }]; #IO_L9N_T1_DQS_AD3N_15 Sch=xa_n[1]
set_property -dict { PACKAGE_PIN A13 IOSTANDARD LVCMOS33 } [get_ports { sencer_sda_io }]; #IO_L9P_T1_DQS_AD3P_15 Sch=xa_p[1]
set_property -dict { PACKAGE_PIN A16 IOSTANDARD LVCMOS33 } [get_ports { hs_in }]; #IO_L8N_T1_AD10N_15 Sch=xa_n[2]
set_property -dict { PACKAGE_PIN A15 IOSTANDARD LVCMOS33 } [get_ports { vs_in }]; #IO_L8P_T1_AD10P_15 Sch=xa_p[2]
set_property -dict { PACKAGE_PIN B17 IOSTANDARD LVCMOS33 } [get_ports { sxtrest[0] }]; #IO_L7N_T1_AD2N_15 Sch=xa_n[3]
set_property -dict { PACKAGE_PIN B16 IOSTANDARD LVCMOS33 } [get_ports { xclk }]; #IO_L7P_T1_AD2P_15 Sch=xa_p[3]
set_property -dict { PACKAGE_PIN A18 IOSTANDARD LVCMOS33 } [get_ports { pclk }]; #IO_L10N_T1_AD11N_15 Sch=xa_n[4]
set_property -dict { PACKAGE_PIN B18 IOSTANDARD LVDS } [get_ports { XA_P[4] }]; #IO_L10P_T1_AD11P_15 Sch=xa_p[4]

##VGA Connector

set_property -dict { PACKAGE_PIN A3 IOSTANDARD LVCMOS33 } [get_ports { VGA_R[0] }]; #IO_L8N_T1_AD14N_35 Sch=vga_r[0]
set_property -dict { PACKAGE_PIN B4 IOSTANDARD LVCMOS33 } [get_ports { vgaout[5] }]; #IO_L7N_T1_AD6N_35 Sch=vga_r[1]
set_property -dict { PACKAGE_PIN C5 IOSTANDARD LVCMOS33 } [get_ports { vgaout[6] }]; #IO_L1N_T0_AD4N_35 Sch=vga_r[2]
set_property -dict { PACKAGE_PIN A4 IOSTANDARD LVCMOS33 } [get_ports { vgaout[7] }]; #IO_L8P_T1_AD14P_35 Sch=vga_r[3]

set_property -dict { PACKAGE_PIN C6 IOSTANDARD LVCMOS33 } [get_ports { VGA_G[0] }]; #IO_L1P_T0_AD4P_35 Sch=vga_g[0]
set_property -dict { PACKAGE_PIN A5 IOSTANDARD LVCMOS33 } [get_ports { vgaout[2] }]; #IO_L3N_T0_DQS_AD5N_35 Sch=vga_g[1]
set_property -dict { PACKAGE_PIN B6 IOSTANDARD LVCMOS33 } [get_ports { vgaout[3] }]; #IO_L2N_T0_AD12N_35 Sch=vga_g[2]
set_property -dict { PACKAGE_PIN A6 IOSTANDARD LVCMOS33 } [get_ports { vgaout[4] }]; #IO_L3P_T0_DQS_AD5P_35 Sch=vga_g[3]

set_property -dict { PACKAGE_PIN B7 IOSTANDARD LVCMOS33 } [get_ports { VGA_B[0] }]; #IO_L2P_T0_AD12P_35 Sch=vga_b[0]
set_property -dict { PACKAGE_PIN C7 IOSTANDARD LVCMOS33 } [get_ports { VGA_B[1] }]; #IO_L4N_T0_35 Sch=vga_b[1]
set_property -dict { PACKAGE_PIN D7 IOSTANDARD LVCMOS33 } [get_ports { vgaout[0] }]; #IO_L6N_T0_VREF_35 Sch=vga_b[2]
set_property -dict { PACKAGE_PIN D8 IOSTANDARD LVCMOS33 } [get_ports { vgaout[1] }]; #IO_L4P_T0_35 Sch=vga_b[3]

set_property -dict { PACKAGE_PIN B11 IOSTANDARD LVCMOS33 } [get_ports { hs_out }]; #IO_L4P_T0_15 Sch=vga_hs
set_property -dict { PACKAGE_PIN B12 IOSTANDARD LVCMOS33 } [get_ports { vs_out }]; #IO_L3N_T0_DQS_AD1N_15 Sch=vga_vs
```

图 5.5 绑定引脚

华中科技大学课程实验报告

接下来是传输协议相机串行接口：SCCB (Serial Camera Control Bus)，OV 公司定义的 SCCB 是一个 3 线结构，但是，为了缩减 Sensor 的 pin 封装，SCCB 大多采用 2 线方式。SCCB 从大体上看，与 I²C 是类似的，我们对 OV5640 寄存器的配置参数是通过 SCCB 总线传输过去的，所以就可以用 I²C 进行数据的传输。

SCCB 控制模块包括 SCCB 时序控制模块和 OV5640 配置模块。SCCB 时序控制模块用于产生符合标准 SCCB 协议的 SCL 和 SDA 信号。首先明确 I²C 的时序，其如图 5.6 所示。时序控制模块每次传输 24 位数据，前 8 位是从设备地址(本系统中从设备即 OV5640，将其地址定义为 0x78，代表写 OV5640 控制寄存器)，接下来的 8 位是从设备寄存器地址，最后 8 位是对 OV5640 控制寄存器进行配置的数据。

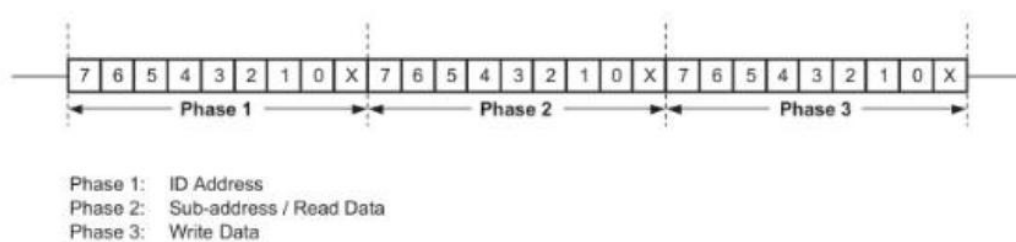


图 5.6 I²C 时序图

OV5640 配置模块对从设备地址、从设备寄存器地址及配置的寄存器值总共 24 位数据进行了定义。需要注意的是，每八位后面都应该有一个应答信号，如图 5.6 中的 X，即“Don't-Care”位，虽然这个应答信号没有 I²C 高低的定义那么严格，但是也应该对其设置。

SCCB 所定义的传输方式分为以下三种：3-phase 写传输周期，2-phase 写传输周期以及 2-phase 读传输周期。在一个传输中包含的最大 phase 数为 3 个，每个 phase 中先发送 MSB。一个 phase 包含总共 9bit，9bit 是由 8bit 有序的数据传输和第 9bit 组成。第 9bit 被称之为 Don't-Care 位或者一个 NA 位，此时就与 I²C 有所不同。具体的不同可以通过查阅 SCCB 协议资料进行对比。不注意该位有可能导致传输未知

的总线状态，然后整个总线挂起。

SCCB 的起始信号、停止信号及数据有效性与 I²C 一样。

起始信号：在 SIO_C 为高电平时，SIO_D 出现一个下降沿，则 SCCB 开始传输。

停止信号：在 SIO_C 为高电平时，SIO_D 出现一个上升沿，则 SCCB 停止传输。

数据有效性：除了开始和停止状态，在数据传输过程中，当 SIO_C 为高电平时，必须保证 SIO_D 上的数据稳定，也就是说，SIO_D 上的电平变换只能发生在 SIO_C 为低电平的时候，SIO_D 的信号在 SIO_C 为高电平时被采集。

在知道对应 I²C 可以直接用的条件下，则参考了 OV5640 资料中给出的示例驱动程序，首先将其进行初始化的配置，开启摄像头。然后将其进行设置，其中遇到的问题就是不知道该将其设置成哪一个模式，则打算采用示例中给出的 video 模式，模仿其中的程序进行操作获取相应图片先。成功的话就可以通过后面的操作系统并行处理读取并发送数据，手机通过蓝牙接收到数据之后，就可以显示图片。完成实时摄像的功能实现。

5.3 故障与调试

由于不知道是硬件的问题还是协议写的问题，那么从两个方面来尝试的话，首先是硬件方面，对于 I²C 的 IP 模块的调用进行调试，可以用硬件调试 IP 模块 ILA (Integrated Logic Analyzer) 进行信号的观察，但由于不太会使用，所以并没有很好地通过该模块进行调试。ILA 有两种模式，AXI 和 Native，一种是接总线直接观察信号，一种是接模块输出观察输出，但在可能如图 5.7 的情况下，一个 IIC 输出到 sensor 里，那如果用 Native 观察其中某个，那就会报错，说其没有能接到 sensor 的输出上去，这也是我迷惑的地方。那如果直接给到 AXI 总线的信号观察，也不太会如何在运行中进行其触发条件的设置与观察。

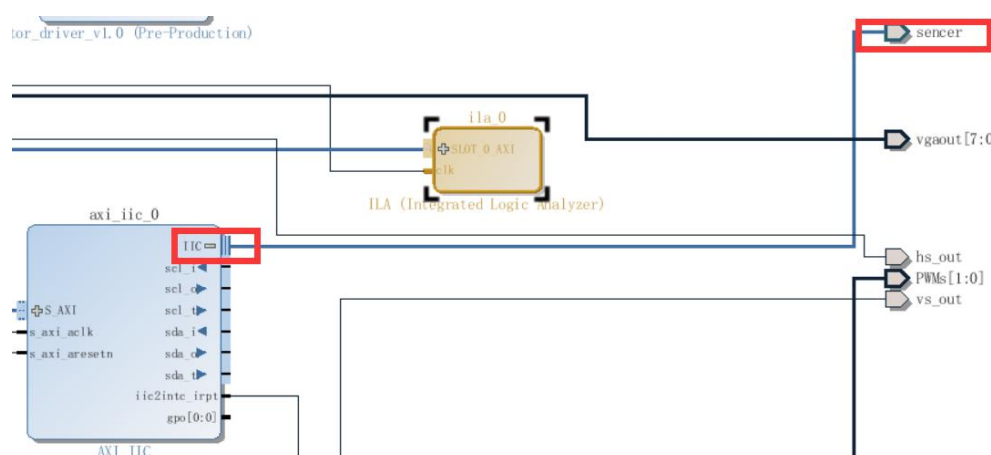


图 5.7 I²C 模块与 ILA 模块

对于 SCCB 的故障调试，其中 SCCB 协议中给出下面两段话：

主机不会对数据传输进行错误检查。从机将会记录 Don't-Care bit 的状态到内部寄存器，例如：一个从设备定义了一个 1 字节的寄存器为 Don't-Care 状态寄存器，并且该寄存器的 default 值为 0xFF，假设在数据的传输过程中没有发生错误。寄存器的值将保持不会改变。如果从机没有接受到 Don't-Care bit，寄存器的值将会变成 0xFE。（通过这种设计方法，可以检测 SCCB 传输的过程中，是不是发生了数据传输的错误，这个方法同样也适用于 I²C 总线。）主机查询 Don't-Care 状态寄存器以检测在数据传输过程中是否发生了错误。主机将向目标从设备的 Don't-Care 状态寄存器发出额外的读取传输，以检测该值。虽然在以上的讨论中，我们知道可以通过这种方式检测到数据传输过程中是否发生了错误传输。但是在实际的设计中，不推荐这样的方法。为了代码的重用性，推荐可以直接使用 I²C 控制程序来实现 SCCB 传输，一般可以将数据写入，然后再读出来以验证是否发生数据读写错误。对于本程序来说，其有时候能成功读出，有时候又不能，所以难以定位错误原因。

5.4 存在的不足和缺陷

1. 驱动通过中断处理，会导致专门化的问题，即普遍性差。
2. 摄像头并没有成功实现。

6 总结与心得

6.1 实验总结

本次蓝牙小车的设计是硬件和软件的结合,其要求我们自己基于 Vivado 平台搭建一个 MIPSfpga 处理器,然后在该处理器的基础上,运行一个微型操作系统,然后在这个操作系统基础上进行应用的设计,通过自己添加的外设接口来控制小车运转。其中涉及到了 CPU 的基本架构知识,操作系统的基本理论,接口方面的设计以及通信协议的使用。

该实验首先从搭建基本的处理器开始,再在这个基础上进行中断控制的添加,然后进行蓝牙外设的控制与操纵,其中碰到几个关键的技术难点如下:处理器与外设之间的交互,操作系统中添加小车的控制模块以及摄像头 OV5640 的数据获取。

处理器与外设之间的交互是指搭建的 CPU 与蓝牙或者马达驱动之间的数据传输。蓝牙部分是用 UART 模块,所以可以根据 UART 模块接口来进行数据传输。在马达驱动则需要用户自定义的 IP 模块封装,也较为好的是,只需要处理器向马达驱动写值,而不用从马达驱动获取数据到处理器内,也就是说单向往外写,所以该问题也不算难。但在摄像头获取数据方面,如果从摄像头中得到数据利用自定义的 IP 核封装的话,不是很清楚如何往处理器里写,当然利用 I²C 后该问题就转化为如何正确地在 I²C 协议基础上进行数据的传输。

操作系统中添加小车的控制模块,我们最终的完成品是通过中断控制的,也可以说是半成品,因为其最好的方法还是在操作系统里面添加设备驱动,这我们经过尝试,但由于对其中的原理了解的不够好,所以没有完成该技术的实现。

在摄像头 OV5640 的数据获取上,首先就是对其输出的格式不是特别了解,然后从这个输出格式到可以显示的格式之间的转换就有问题。其次就是对通信协议这

一方面了解的不是很深入，所以最后还是没有成功。

后面如果要继续完成的话，我自己有两个思路，一个是沿着我们的方案继续往下做，不过要看是硬件部分出了问题还是 IIC 协议出了问题，这样定位问题才好解决。另一个就是利用下面的技术方案：DVP 接口 -> dvp_2_axi4s -> axi vdma -> axi interconnect -> MIPS MicroAptiv UP-> DDR2，进行视频流的获取与保存，需要重新设计 DVP 接口来进行图像数据的处理与传输。

6.2 实验心得

1. 从硬件部分设计到软件接口设计，从 MIPSfpga 处理器再到 Hos 操作系统，每一步都是不可轻视的。
2. 团队之间的协作与帮助很重要，要合理安排每个人的工作。
3. 遇到问题要勇于尝试，不要轻言放弃，可以各种思路去试试。
4. 对操作系统的原理和接口的了解得还不够，多了解一些可能做起来会轻松许多。

6.3 意见与建议

本次课程给我带来的收获是不小的，而在对于本课程的建议就是，前面部分分值可以适当放低些，后面扩展部分可以适当增加分值。然后整个小组最好不超过 6 人，这样在分工上也会更加合理些。

致谢

首先感谢我的组员们，正是因为有他们跟我一起努力，才能完成我们自己的初步目标，才能让我在后面不断失败的时候没有轻言放弃。

再来就是感谢老师和助教，正是他们的辛勤指导和鼓励，我们才可以更好更快地完成我们自己任务，他们在我们困难的时候给的很多指导和建议都对项目的进程有极大的帮助。

最后特别感谢 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。

参考文献

- [1]. Xilinx. AIX IIC Bus Interface v2.0 LogiCORE IP Product Guide[EB/OL].
https://www.xilinx.com/support/documentation/ip_documentation/axi_iic/v2_0/pg090-axi-iic.pdf.
- [2]. Xilinx.AIX UART 16550 v2.0 LogiCORE IP Product Guide[EB/OL].
https://www.xilinx.com/support/documentation/ip_documentation/axi_uart16550/v2_0/pg143-axi-uart16550.pdf.
- [3]. DAVID A.PATTERSON(美).计算机组成与设计硬件/软件接口(原书第4版).北京:机械工业出版社.
- [4]. David Money Harris(美).数字设计和计算机体系结构（第二版）.机械工业出版社
- [5]. 秦磊华, 吴非, 莫正坤.计算机组成原理. 北京: 清华大学出版社, 2011 年.
- [6]. 袁春风编著. 计算机组成与系统结构. 北京: 清华大学出版社, 2011 年.
- [7]. 张晨曦, 王志英. 计算机系统结构. 高等教育出版社, 2008 年.
- [8]. 周湘贞. 操作系统原理与实践教程. 北京:清华大学出版社,2006.

华中科技大学课程实验报告

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:

二、对课程实验的学术评语（教师填写）

三、对课程设计的评分（教师填写）

评分项目 (分值)	报告撰写 (30 分)	课设过程 (70 分)	最终评定 (100 分)
得分			

指导教师签字: _____

2018-12-17