



我们

项目设计最终报告

景区自动化游戏交互机器人

组号：041

2019/6/1

姓名	职责
王秀程	组长 数字系统设计
杨锦福	外观与机械结构设计
申振	网页设计
贾竣皓	系统组装
陈云	上位机设计
熊焰波	辅组网页设计和组装
高博	秘书



目录

第一章 介绍.....	1
1.1 项目介绍	1
1.2 成就	1
1.2.1 项目需求	1
1.2.2 机械设计与制造	1
1.2.3 软件设计与实现	2
1.2.4 联合调试	2
1.2.5 低成本	2
1.3 鸣谢	3
1.4 文档概述	3
第二章 机械设计	4
2.1 方案确定	4
2.1.1 机械要求汇总	4
2.1.2 待确定的问题	4
2.1.3 初步结构	4
2.2 最终设计	8



2.2.1 落子系统	8
2.2.2 颜色传感器系统	9
2.2.3 距离传感器系统	10
2.2.4 红外传感器系统	11
2.3 最终机械展示	12
第三章 数字系统设计	18
3.1 介绍	18
3.2 STM32-L476 开发	18
3.2.1 代码结构	18
3.2.2 中断函数	29
3.2.3 数据获取	24
3.2.4 串口设置	24
3.3 FPGA 开发	25
3.3.1 代码结构	26
3.3.2 输入和输出声明以及子模块的实例化	26
3.3.3 直流电机	27
3.3.4 UART	28
3.3.5 引脚规划结果	28



第四章 软件设计	30
4.1 设计目的	30
4.1.1 服务目的	30
4.1.2 宣传目的	30
4.1.3 维护目的	30
4.2 技术信息	30
4.3 界面设计	30
4.3.1 主菜单界面	30
4.3.2 规则介绍界面	31
4.3.3 游戏界面	32
4.3.4 维护模式界面	33
4.3.5 游戏胜利界面	34
4.3.6 游戏失败界面	35
4.3.7 平局界面	36
4.4 后台逻辑设计	36
第五章 传感器系统	38
5.1 TCS3472_I2C 色彩传感器	38



5.2 VL6180 TOF 传感器.....	40
-------------------------	----



第一章 介绍

1.1 项目介绍

在这个项目中，我们的成员们通力合作，取得了与老师组签订关于研究开发景区智能互动游戏机器人合同的机会。该产品能够通过读取用户卡上的用户信息做出不同应答，并通过传感器判断用户状态做出不同种类的机械交互。在添加了软件界面开发的同时设置维护模式以便对急切状况进行较为详细的监控与调修。

在项目研发过程中，我们各部人员克服了包括产品设计、技术开发等一些列问题，设计并开发了一套采用计算机、嵌入式软件及数字技术进行有机结合的可靠系统，逐步完成了产品开发中的各阶段任务。该开发项目完成后可以出色完成老师组所要求的功能，除此之外还在部分设计上做出了改进，产品外观上也添加了较为出色的设计。与此同时我们制作了机械原型与样品机器证明了系统的有效性及合理性。

1.2 成就

1.2.1 设计需求

完成景区交互机器人的设计。

1.2.2 机械设计与制造

产品采用一体式结构，内置开发板架台，最大程度为布线提供了空间。箱体侧面开口作为检修维修通道，正面设计有安卓开发板镶嵌口，右侧设置用户卡读取槽。箱体背面设置电源口，机械结构主要集中在顶板。顶板设 3*3 的工作台阵列，代表井字棋中九个棋格。每个工作台配有料槽、打锤、电机各一个。工作台本体贴于顶板，采用 3D 打印工艺制作，方便快捷。打锤与电机相连，采用推拉掉落式的下棋方式将料槽底部棋子推入棋格，简单易制作，稳定性强。棋格底部嵌入红外传感器。用于板材连接的角铁与活页装在机箱内部，对外观无影响的同时加大了结构稳定性。产品采用压缩泡沫板材，成本降低同时保持了板材坚固性，减轻了重量，提高了产品性价比。产品使用模块化，易于替换的器件，方便运营



与维护，也便于市场推广。

1.2.3 软件设计与实现

该 Tic-Tac-Toe app 可在两种模式下运行：用户模式以及维护模式。

维护模式下是操作人员对整个机器人监控以及检测的模式，该模式下操作人员可对机器人开箱检查，检查各部分是否运转良好以及进行检修，维护模式需要开发人员用专用工程师身份验证卡进行身份验证操作。在维护模式下工程师可检测距离传感器，颜色传感器，电机的正常工作与否。

用户模式是用户通过插卡进行游戏难度选择，在阅读规则后与机器人对局，达到游戏机器人与玩家全程互动的目的。其中包括六个界面：主菜单界面、规则介绍界面、游戏模式界面以及胜利失败平局三个游戏结果界面。

另外为更好的服务于使用用户，本机器还提供了中英文国家化功能及使用过程的语音提示、规则介绍功能。

技术信息：编程语言：java、xml 布局文件语言。

编程环境：Windows 10 操作系统

编程工具：Android Studio

1.2.4 联合调试

产品开发期间，我们对系统各部分进行了多次测试与调整，成功解决了分布在机械运作、程序运行各部分中的数十处问题，保证了系统的稳定性和可靠性。

1.2.5 低成本

除 STM32、FPGA、安卓开发板与传感器外，项目使用的电机、驱动模块、材料板等材料的总成本控制在 600 元左右，是此次所有设计小组中最低的成本之一。

1.3 文档概述

本文提供了关于我们所设计的产品的具体技术细节，内容涵盖机械和软硬件



设计，传感器系统，安卓开发以及市场和项目管理等多个方面。

第一章节为概述与相关信息阐明；第二章将简要介绍机械结构的部分以及设计过程；第三章将对数字系统进行描述，包括对部分代码的解释、各种图表等；第四章为软件设计内容；营销与网络内容将被放在第五章中。



第二章 机械设计

2.1 方案确定

2.1.1 机械要求汇总

根据甲方——老师组的技术要求与实际情况，我们团队汇总各方面技术要求，综合考量各部分技术标准，最终将游戏机制作要求汇总如下：

- 游戏机的整体尺寸小于 300mm*300mm*500mm。
- 游戏机内部所包含的传感器以及电子元器件：FPGA 主板、 ATM32 主板、安卓主板、颜色传感器、距离传感器、红外传感器，电路集成主板，九个电机，必须规划出对应的空间安放在游戏机内。
- 顶板工作台每个大小：并能够被固定在顶板上。棋子每个为硬币大小。

2.1.2 待确定的问题

通过进一步核对确认，汇总待解决问题如下：

- 1) 机械加工时，由于加工板材自身特性，可能出现加工精度降低，各结构间协作不当等问题，进而影响游戏机机械工作，甚至可能与现有程序冲突，尚未有经验验证有效的改进方案。
- 2) 是否需要游戏奖励系统。

2.1.3 初步结构

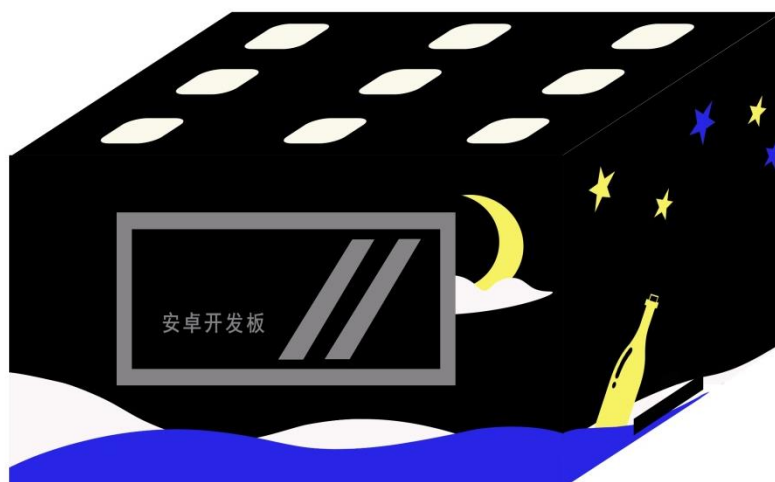
1) 文字阐述

- 采用一体式结构。
- 顶板工作台 3D 设计时留出棋格、料槽、电机、打锤与螺丝固定口。
- 有打锤将料槽底部棋子推入料槽完成机器下棋动作。
- 红外传感器镶嵌在棋格漏孔处的顶板中。
- 通过斜坡的方式将货物送至取货口。



- 将部分固定用的组建直接嵌入机壳体，增强稳定性。
- 使用金属轴承减小转动阻力，减小使用过程中对系统的损耗。
- 使用模块化，易于替换的器件，方便运营维护与市场推广。
- 使用活页角铁进行板材连接。
- 箱体开口方便检修。
- 机箱内部设置开发板置架。

2) 外观设计



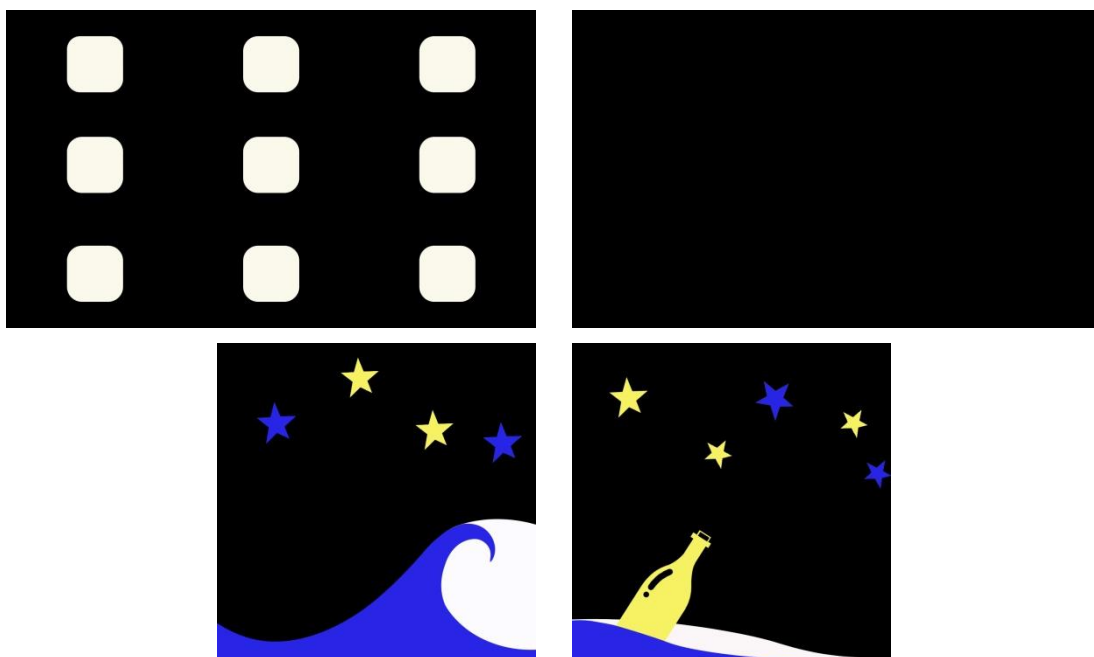


图 2.1~2.7 外观图

3) 机械设计

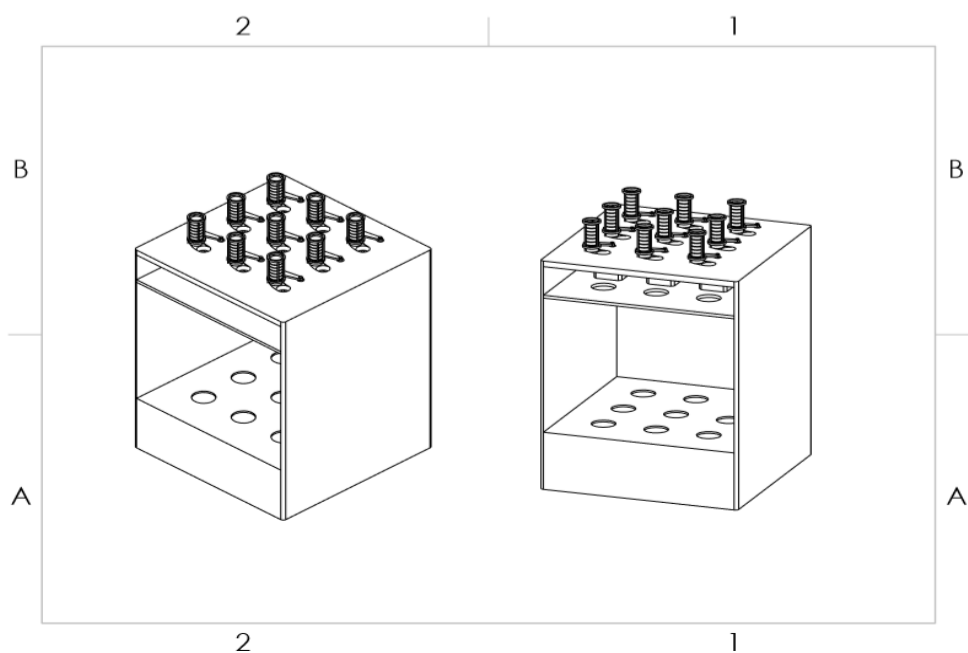


图 2.8 整体图

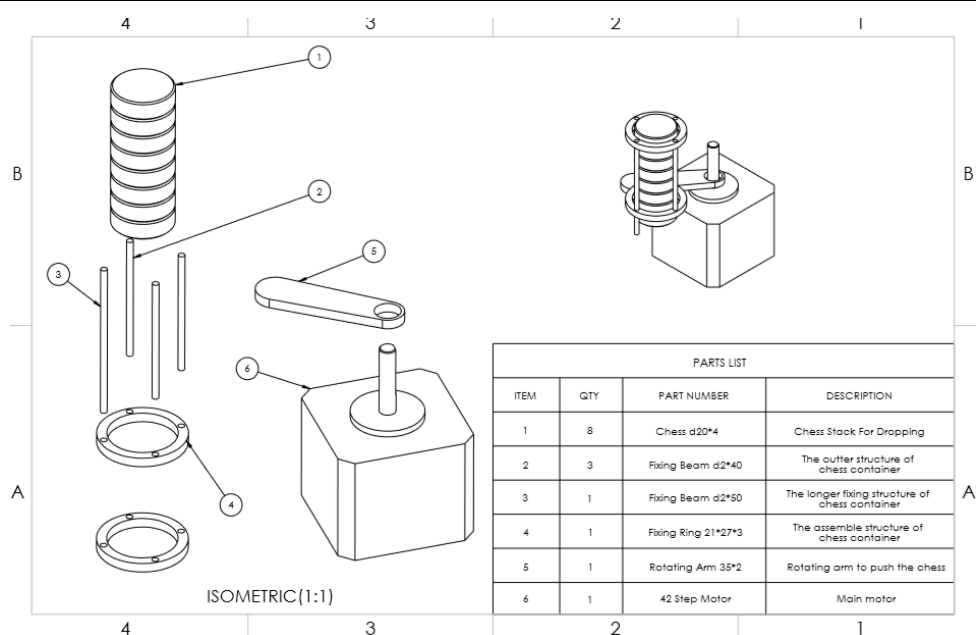


图 2.9 零件图

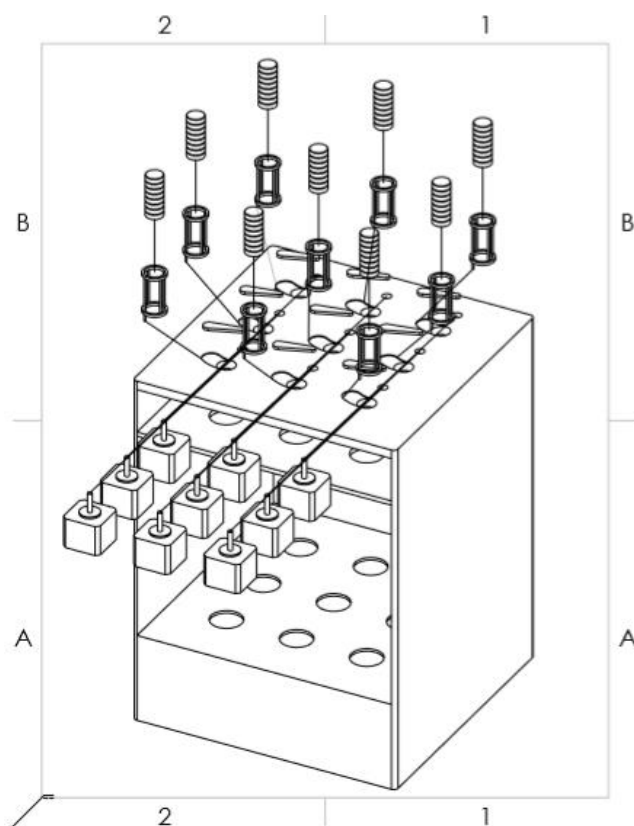


图 2.10 装配图



2.2 最终设计

2.2.1 落子系统

为节省成本与提高代码编写难度，提高产品性价比，我们对产品落子系统能够做如下设计：

- 1) 将落子单位结构扩展为九组，呈 3*3 阵列安装于箱体顶板上。
- 2) 每一组落子结构由料槽、底座、打锤、棋子组成，整体大小大约为 8*8*8（单位 cm）立方体。
- 3) 为保证落子稳定性，我们把底座设计为有落子凹槽和螺钉孔口的稳定结构，并多次试验，将料槽、电机轴心长度、棋子厚度等一系列因素考虑在内，做到了稳定落子，流畅落子。
- 4) 底座参数：80*80*5（单位：mm）



图 2.11 底座实际外观

2.2.2 颜色传感器系统

经过讨论与设计，我们决定将颜色传感器用于用户身份识别。由白色卡片代表用户，蓝色卡片代表维修工程师。

- 1) 用户执白色用户卡，在软件操作界面无法点击维修模式按钮，仅拥有游戏权限。
- 2) 维修工程师执蓝色卡，在软件界面可点击进入维修模式界面，拥有机器调修



与数据查看的权限。

- 3) 颜色传感器系统位于机箱右下方，卡槽位于箱体右侧。安装时不可位于箱体中心部分，以免内部结构影响颜色传感器系统工作。

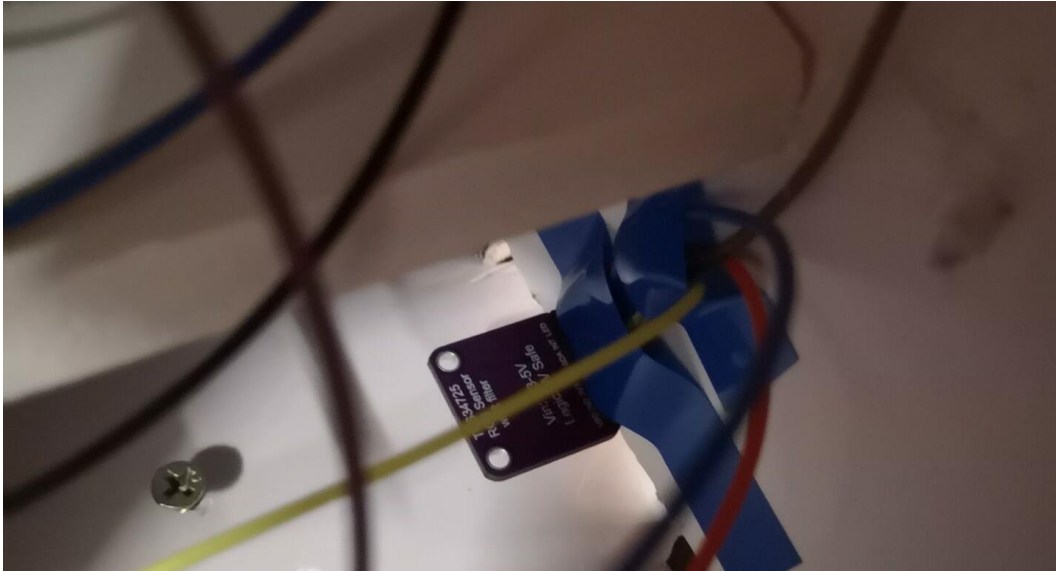


图 2.12 颜色传感器

2.2.3 距离传感器系统

为保证用户的使用体验与箱体安全，我们设置了距离传感器用以检测用户与箱体间的距离。

- 1) 距离传感器位于箱体正面，安卓开发板正上方，。
- 2) 8~30cm 为正常使用距离。
- 3) 用于距离箱体过远或过近都会由距离传感器感知并将采集数据传回主控制板处理，并且做出相应的响应。

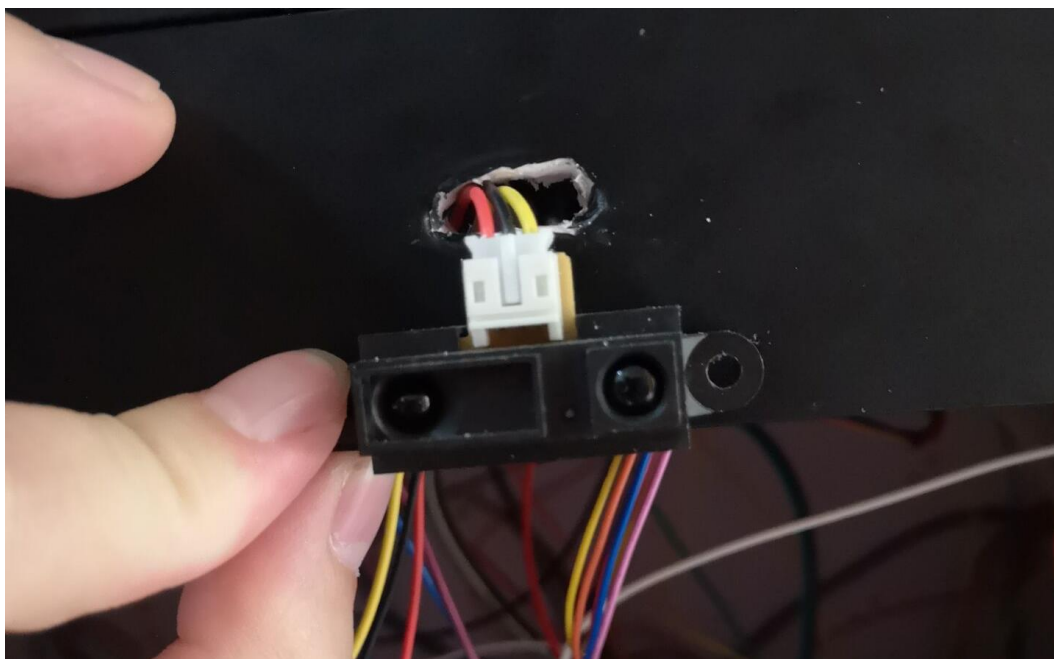


图 2.13 距离传感器

2.2.4 红外传感器系统

为检测棋子状态与棋局形式以便做出机械回应，我们设置了红外传感器系统。

- 1) 红外传感器位于落子单位结构中底座圆孔下，镶嵌于顶板中。
- 2) 当棋子落入圆孔凹槽中，红外传感器将落子信息传给主控制板进行数据处理。
- 3) 机器与用户的落子动作都会使红外传感器工作，以随时判断棋盘状态。

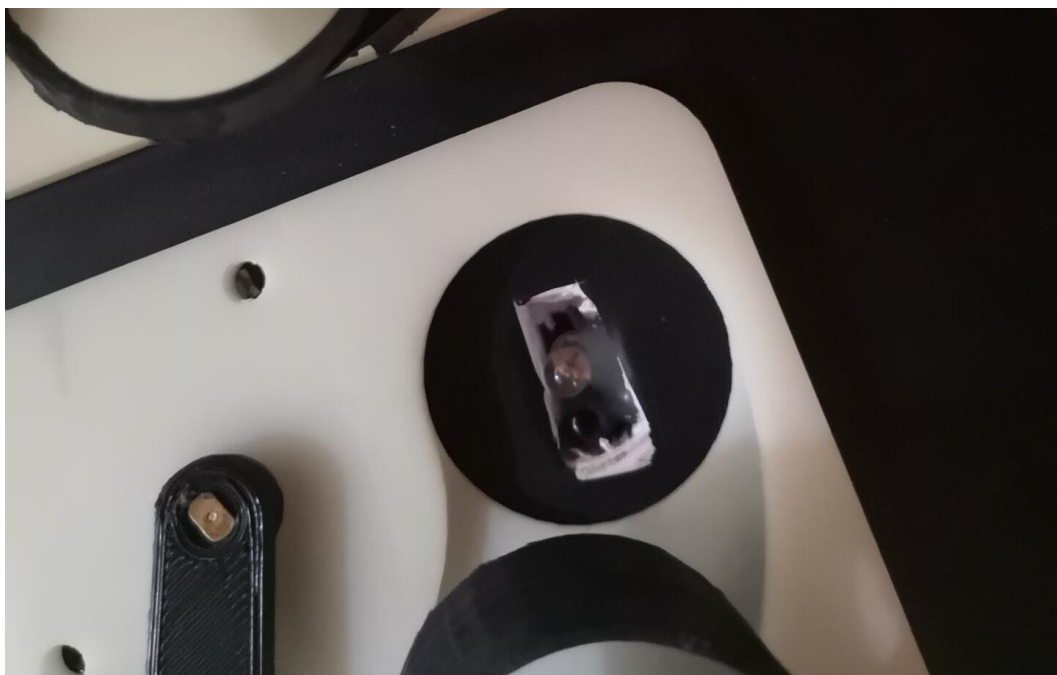


图 2.14 红外传感器

2.3 最终机械结构展示









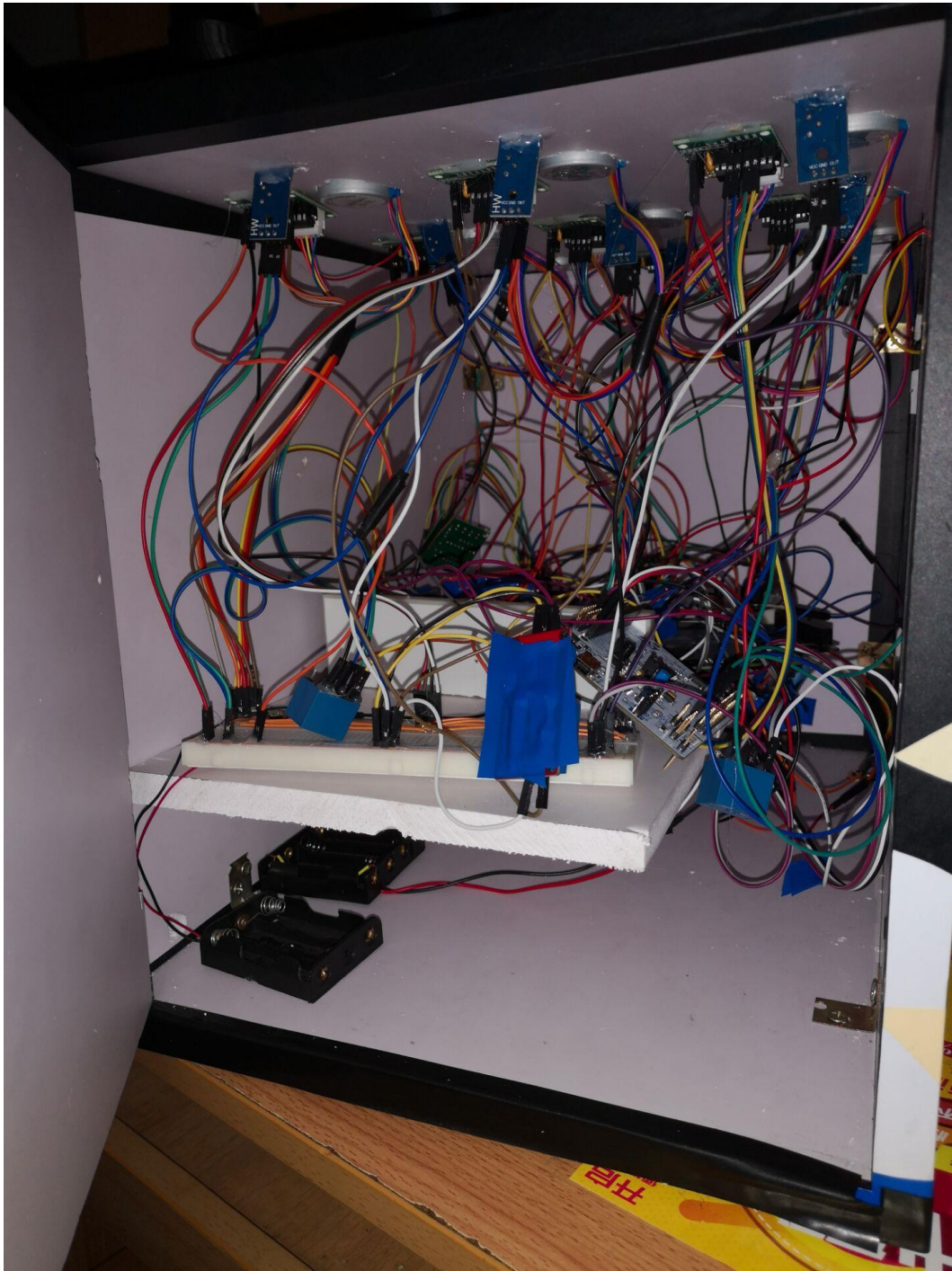


图 2.15~2.20 机械外观展示





第三章 数字系统设计

3.1 介绍

本项目所采用的 MCU 为 ST 公司生产的 STM32-L476 开发板与 ALTERA 公司的 FPGA(已经被 INTER 收购)。

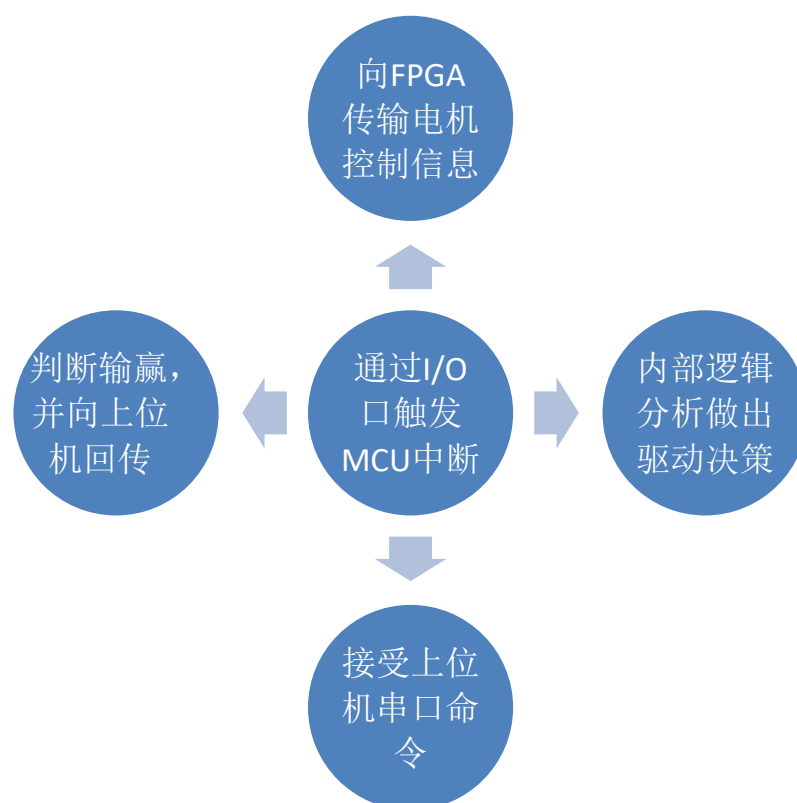
STM32 系列专为要求高性能、低成本、低功耗的嵌入式应用设计的 ARM Cortex®-M0, M0+, M3, M4 和 M7 内核。按内核架构分为不同产品,不同的封装保持引脚排列一致性,结合 STM32 平台的设计理念,开发人员通过选择产品可重新优化功能、存储器、性能和引脚数量,以最小的硬件变化来满足个性化的应用需求。本项目选用 STM32-L476。

FPGA 是现场可编程门阵列的缩写。它是一种半导体实现电路,其中大部分的电气功能都可以改变。它是非常灵活的功能可以改变的每一个电源设备。我们选择的 FPGA 包括芯片上的处理器、收发器 I/O 的 28Gbps(或更快)、RAM 块、DSP 引擎等等。ALTERA FPGA 支持长生命周期(15 年以上)。本项目选用旋风 II EP2C20F484C7N.1。

3.2 STM32-L476 开发

3.2.1 代码结构

其中数字系统整体设计为:



在主函数中，将颜色传感器、距离传感器的 I²C 口传输线初始化，并将串口波特率等设置初始化。

3.2.2 中断函数

在 MCU 的设计中，用中断来触发电机驱动和输赢判断函数。其中在用户落下棋子后红外传感器 OUT 由低电平跳变为高电平，从而设置高电平有效触发中断。



```
main.cpp X
1 #include "mbed.h"
2 #include <math.h>
3
4 InterruptIn mybutton1(PC_4); //中断定义
5 InterruptIn mybutton2(PB_13);
6 InterruptIn mybutton3(PB_14);
7 InterruptIn mybutton4(PB_15);
8 InterruptIn mybutton5(PB_1);
9 InterruptIn mybutton6(PB_2);
10 InterruptIn mybutton7(PC_8);
11 InterruptIn mybutton8(PB_12);
12 InterruptIn mybutton9(PA_11);
```

图 3.1 定义中断管脚

```
int main()
{

    mybutton1.fall(&play1);
    mybutton2.fall(&play2);
    mybutton3.fall(&play3); //定义上升沿触发中断
    mybutton4.fall(&play4);
    mybutton5.fall(&play5);
    mybutton6.fall(&play6);
    mybutton7.fall(&play7);
    mybutton8.rise(&play8);
    mybutton9.rise(&play9);
```

图 3.2 下降沿触发中断

将服务程序设置为中断模式有两大好处：一是可以随时接受上位机的串口信息，再一个是不必循环检查是否有红外传感器感受到信号变化。



```
void play1()
{
    color[0]=1;//中断程序
    //wait(1);
    if(interrupt_flag[0]==1)
    {pc.printf("input is 1\n");play();interrupt_flag[0]=0;}//进入电机驱动模块，并将中断使能关闭
    return;
}
```

图 3.3 中断服务函数

不同的红外传感器触动不同的中断，其中在每个中断服务函数的第一层，首先将对应的棋子标识为置“1”。然后检查中断使能是否存在，若可以中断使能，则进入“play”函数，驱动电机与向上位机反馈输赢信息。由于不管输或者赢均只有 8 中情况，所以对输赢判断使用了简单的 if 函数来判断。一旦判断胜利条件成立，则串口传输“W”，并立即退出中断服务程序。同理在输时串口传输“L”，平局是串口传输“E”。

```
if((color[0]==2&&color[1]==2&&color[2]==2)|| (color[3]==2&&color[4]==2&&color[5]==2))//判断输
{pc.printf("L");return;}
if((color[6]==2&&color[7]==2&&color[8]==2)|| (color[0]==2&&color[3]==2&&color[6]==2))
{pc.printf("L");return;}
if((color[1]==2&&color[4]==2&&color[7]==2)|| (color[2]==2&&color[5]==2&&color[8]==2))
{pc.printf("L");return;}
if((color[0]==2&&color[4]==2&&color[8]==2)|| (color[2]==2&&color[4]==2&&color[6]==2))
{pc.printf("L");return;}
if(color[0]!=0&&color[1]!=0&&color[2]!=0&&color[3]!=0&&color[4]!=0&&color[5]!=0&&color[6]!=0&&color[7]!=0&&color[8]!=0)
{pc.printf("E");return;}//判断平局
```

图 3.4 输赢判断函数

当进入 play 函数，且用户没有胜利时，则进入电机驱动模块。其中驱动线有 4 根，用二进制编码的方式向 FPGA 传输驱动对应电机的信息。游戏设有两个模式：简单模式与困难模式。在简单模式中机器只会顺序下棋，在困难模式中机器会做一定的预判，来围堵人下的棋，从而避免人下的棋连成一线。



```
if(user_number==0)//复杂模式
{
    if(color[4]==0)
    {
        pinout = 4; color[4]=2;
    }
    else if(color[0]==1&&color[1]==1&&color[2]==0)//极大极小值抑制判断下棋策略
    {
        pinout = 2; color[2]=2;
    }
    else if(color[1]==1&&color[2]==1&&color[0]==0)
    {
        pinout = 0; color[0]=2;
    }
    else if(color[0]==1&&color[3]==1&&color[6]==0)
    {
        pinout = 6; color[6]=2;
    }
}
```

图 3.5 困难模式下机器下棋算法

当 MCU 通过判断，决定机器应下再棋盘的几号位置后。对应的棋盘标记首先置 2，然后再将 pinout 置为对应的数值。然后将 pinout 二进制编码，向 FPGA 传输驱动信息。然后通过 wait 函数，控制电机的驱动时间。在一定时间后将电机的传输信息清零，电机停转。

```
switch(pinout)
{
    case 0:
        motivator_4=0;
        motivator_3=0; //二进制驱动电机
        motivator_2=0;
        motivator_1=1;
        break;
    case 1:
        motivator_4=0;
        motivator_3=0;
        motivator_2=1;
        motivator_1=0;
        break;
    case 2:
        motivator_4=0;
        motivator_3=0;
```



图 3.6 二进制编码驱动电机

```

interrupt_flag[pinout]=0;
wait(16.5);
motivator_4=0;
motivator_3=0; //电机停止
motivator_2=0;
motivator_1=0;

```

图 3.7 控制电机驱动时长

3.2.3 数据获取

由于颜色传感器和距离传感器，均使用 I²C 传输协议，故需提前设置 MCU 的 I²C 接口。并修改其 I²C 地址，读取地址中的数据。

```

char id_regval[1] = {146};
char data[1] = {0};
i2c.write(sensor_addr,id_regval,1, true);
i2c.read(sensor_addr,data,1,false);

// Initialize color sensor
green=0;
char timing_register[2] = {129,0};
i2c.write(sensor_addr,timing_register,2,false); // set the number

char control_register[2] = {143,0};
i2c.write(sensor_addr,control_register,2,false);

char enable_register[2] = {128,3};
i2c.write(sensor_addr,enable_register,2,false);

// Read data from color sensor (Clear/Red/Green/Blue)

```

图 3.8 I²C 地址等数据设置

在获取传感器的数据后，通过对 RGB 值的比较，来判断操作者的身份，其中蓝色的身份卡为维护工程师所用卡，可以控制上位机进入维护模式。其他颜色的卡为用户卡，可以进入游戏模式进行游戏。



```
while (true)
{
    ADCdata = -26.0*log(juli)-2.9;
    char clear_reg[1] = {148};
    char clear_data[2] = {0,0};
    i2c.write(sensor_addr,clear_reg,1, true);
    i2c.read(sensor_addr,clear_data,2, false);

    int clear_value = ((int)clear_data[1] << 8) | clear_data[0];

    char red_reg[1] = {150};
    char red_data[2] = {0,0}; //设置IIC传输方式
    i2c.write(sensor_addr,red_reg,1, true);
    i2c.read(sensor_addr,red_data,2, false);

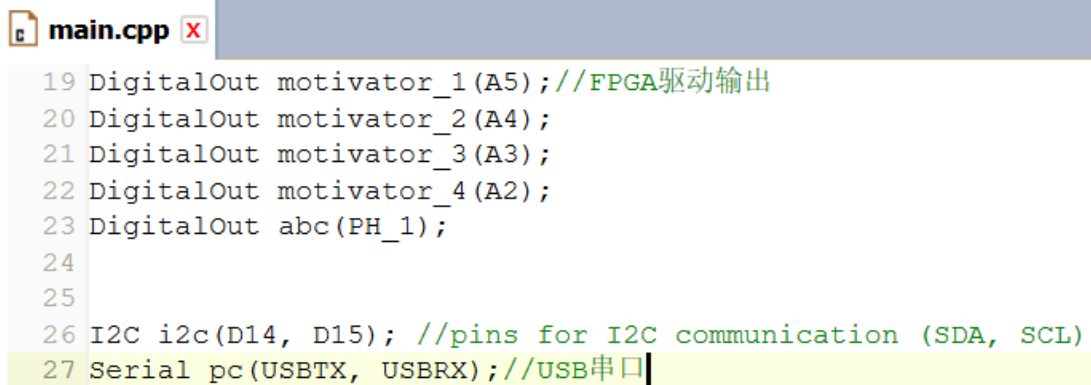
    int red_value = ((int)red_data[1] << 8) | red_data[0];

    char green_reg[1] = {152};
    char green_data[2] = {0,0};
}
```

图 3.9 获取颜色传感器中的数据

3.2.4 串口设置

串口用于上位机与MCU之间的通信,为了串口通信的顺畅与MCU的安全,我们使用了USB 串口,而非不可靠的TTL 串口。其中串口的使能设置如下:



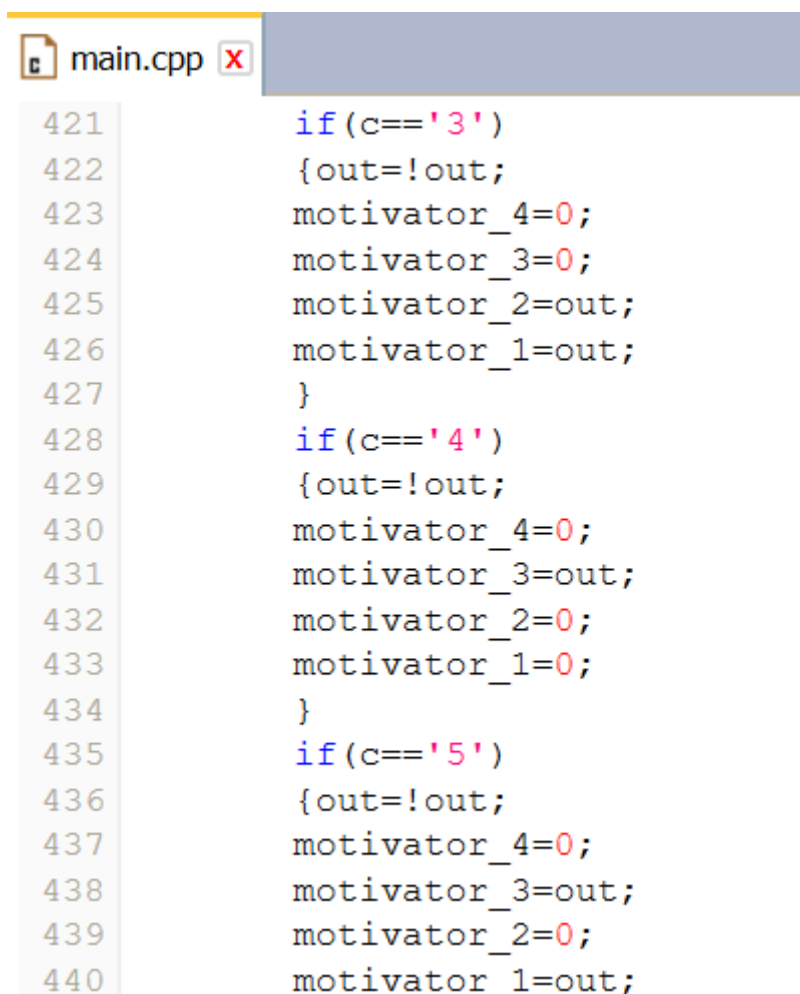
```
main.cpp
19 DigitalOut motivator_1(A5); //FPGA驱动输出
20 DigitalOut motivator_2(A4);
21 DigitalOut motivator_3(A3);
22 DigitalOut motivator_4(A2);
23 DigitalOut abc(PH_1);
24
25
26 I2C i2c(D14, D15); //pins for I2C communication (SDA, SCL)
27 Serial pc(USBTX, USBRX); //USB串口
```

图 3.10 串口等外设管脚设置

串口接受上位机的主要指令来源于上位机维护模式下,对颜色、距离的请求,以及对电机的强制驱动。其中当上位机发送 color 首字母“C”时,串口返回颜色信息。当上位机发送 distance 首字母“d”时,串口返回距离数值。当上位机发送 1-9 数字时,串口驱动对应电机工作,检查电机状态。



```
if(c=='C')
{pc.printf("Clear (%d), Red (%d), Green (%d), Blue (%d)\r\n", clear_value, red_value, green_value, blue_value);}
//获得颜色
if(c=='1')
{out=!out; //维护模式驱动电机
motivator_4=0;
motivator_3=0;
motivator_2=0;
motivator_1=out;
}
if(c=='2')
{out=!out;
motivator_4=0;
motivator_3=0;
motivator_2=out;
motivator_1=0;
}
```



```
421     if(c=='3')
422     {out=!out;
423     motivator_4=0;
424     motivator_3=0;
425     motivator_2=out;
426     motivator_1=out;
427     }
428     if(c=='4')
429     {out=!out;
430     motivator_4=0;
431     motivator_3=out;
432     motivator_2=0;
433     motivator_1=0;
434     }
435     if(c=='5')
436     {out=!out;
437     motivator_4=0;
438     motivator_3=out;
439     motivator_2=0;
440     motivator_1=out;
```

图 3.11~3.12 电机驱动程序和维护模式程序

3.3 FPGA 开发

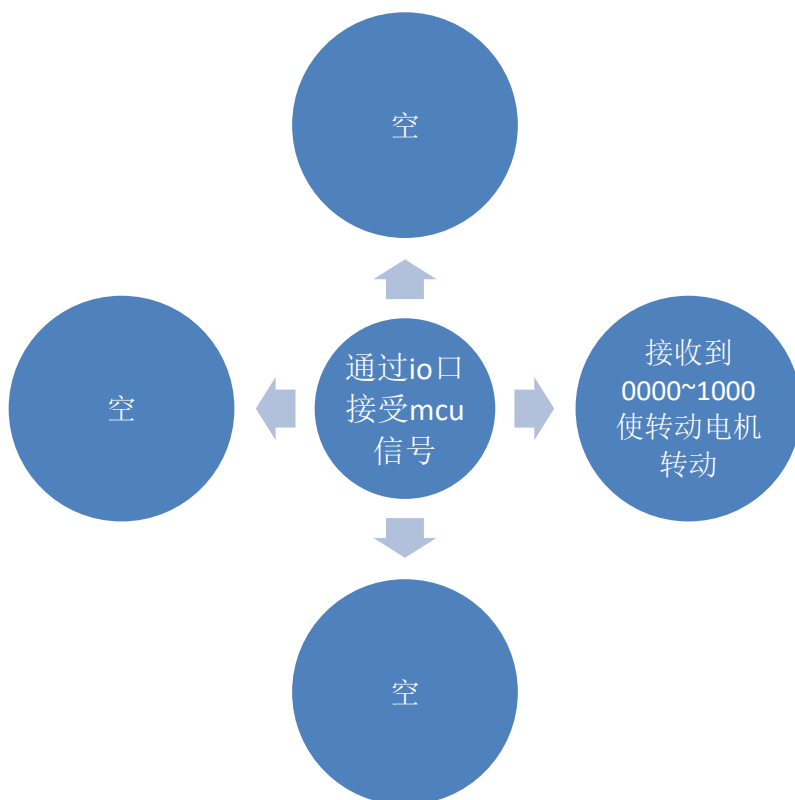
FPGA 控制本项目中的所有电机:九个直流电机。在 FPGA 的控制下, 直流电机应转动一圈。一旦收到来自 MBED 的信号, FPGA 单元将控制实际排序过



程的所有方面。我们会在数码管上显示从 MBED 接收到的信号在维护和运行模式下，FPGA 只需要根据 MBED 接收到的信号来控制直流电机旋转。

3.3.1 代码结构

一个总体模块加上一个分散模块分别驱动九个电机按照预定程序转动一圈



3.3.2 输入和输出声明以及子模块的实例化

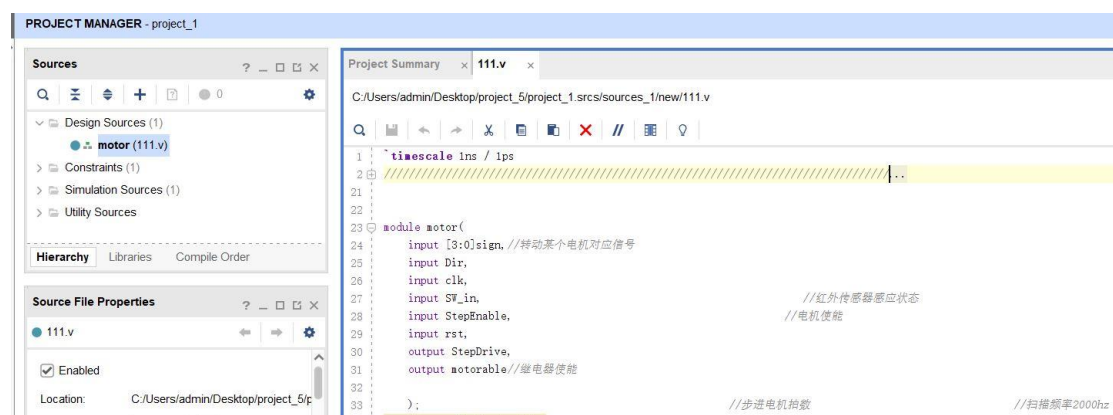


图 3.13 输入与输出



3.3.3 直流电机

在本项目中，我们只使用一种直流电机，所以在九个电机中应保证转动时间的一致性，我们通过测试和反复验证，发现直流电机转动二十秒恰好转满一圈，所以我们在代码中让继电器只为直流电机供电二十秒，电机通电即转，无电则停，功能完成度较高。

```

); //步进电机指数 //扫描频率2000hz
reg [8:0] motorable;
reg [8:0] StepDrive;
reg [2:0] state;
reg [31:0] StepCounter = 32'b0;
parameter [31:0] StepLockOut = 32'd400000; //步进电机频率250hz
reg InternalStepEnable;
always @(posedge clk or negedge rst)
begin
begin
if ( !rst )
begin
StepDrive <= 4'b0;
state <= 3'b0;
StepCounter <= 32'b0;
end
else
begin
if (StepEnable|~SW_in == 1'b1) InternalStepEnable <= 1'b1;
StepCounter <= StepCounter + 31'b1; //当电机使能或红外传感器检测到有人时，步进数加一
if (StepCounter >= StepLockOut)
begin
StepCounter <= 32'b0; //循环计数
end
if (InternalStepEnable == 1'b1)
begin
InternalStepEnable <= StepEnable|~SW_in;
if (Dir == 1'b1) state <= state + 3'b001; //正转
else if (Dir == 1'b0) state <= state - 3'b001; //反转
case (state)
3'b000 : StepDrive <= 4'b0001;
3'b001 : StepDrive <= 4'b0011;
3'b010 : StepDrive <= 4'b0010;
3'b011 : StepDrive <= 4'b0110;
3'b100 : StepDrive <= 4'b0100;
3'b101 : StepDrive <= 4'b1100;
3'b110 : StepDrive <= 4'b1000;
3'b111 : StepDrive <= 4'b1001; //四相八拍状态: A-AB-B-BC-C-CD-D-DA
end
end
end
end
end

```

图 3.14 电机转动的程序代码



```
57 if (InternalStepEnable == 1'b1)
58 begin
59     InternalStepEnable <= StepEnable|^SW_in ;
60     if (Dir == 1'b1)         state <= state + 3'b001 ; //正转
61     else if (Dir == 1'b0)    state <= state - 3'b001 ; //反转
62     case (state)
63         3'b000 : StepDrive <= 4'b0001 ;
64         3'b001 : StepDrive <= 4'b0011 ;
65         3'b010 : StepDrive <= 4'b0010 ;
66         3'b011 : StepDrive <= 4'b0110 ;
67         3'b100 : StepDrive <= 4'b0100 ;
68         3'b101 : StepDrive <= 4'b1100 ;
69         3'b110 : StepDrive <= 4'b1000 ;
70         3'b111 : StepDrive <= 4'b1001 ; //四相八拍状态: A-AB-B-BC-C-CD-D-DA
71     endcase
72 end
73 end
74 end
75 end
76
77 case(sign)
78     4'b0000 : motorable<= 9'b1 ;
79     4'b0001 : motorable<= 9'b1<<1;
80     4'b0010 : motorable<= 9'b1<<2 ;
81     4'b0011 : motorable<= 9'b1<<3 ;
82     4'b0100 : motorable<= 9'b1<<4 ;
83     4'b0101 : motorable<= 9'b1<<5 ;
84     4'b0110 : motorable<= 9'b1<<6;
85     4'b0111 : motorable<= 9'b1<<7 ;
86     4'b1000 : motorable<= 9'b1<<8 ;
87     default : motorable<= 0;
88 endcase
89 end
90
91
92
93 endmodule
94
```

图 3.15 电机工作逻辑关系上

3.3.4 UART

本项目中布线量较大，例如 MBED 和 FPGA 之间的电线，MBED 和传感器之间的电线，FPGA 和四个伺服电机之间的电线。虽然串行通信将会减少大量生产机器的成本，减少了电缆的使用。但是在设置引脚时模拟引脚充足，因此我们决定使用并行通信。

3.3.5 引脚规划结果



IO Ports												
Name	Neg Diff Pair	Direction	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination
motorable (9)		OUT		<input checked="" type="checkbox"/>	(Mu...	LVC MOS33*	3.300		12	SLOW	NONE	FP_VTT_50
motorabl...		OUT	A1	<input checked="" type="checkbox"/>	35	LVC MOS33*	3.300		12	SLOW	NONE	FP_VTT_50
motorabl...		OUT	A3	<input checked="" type="checkbox"/>	35	LVC MOS33*	3.300		12	SLOW	NONE	FP_VTT_50
motorabl...		OUT	A5	<input checked="" type="checkbox"/>	35	LVC MOS33*	3.300		12	SLOW	NONE	FP_VTT_50
motorabl...		OUT	A4	<input checked="" type="checkbox"/>	35	LVC MOS33*	3.300		12	SLOW	NONE	FP_VTT_50
motorabl...		OUT	A9	<input checked="" type="checkbox"/>	16	LVC MOS33*	3.300		12	SLOW	NONE	FP_VTT_50
motorabl...		OUT	A8	<input checked="" type="checkbox"/>	16	LVC MOS33*	3.300		12	SLOW	NONE	FP_VTT_50
motorabl...		OUT	A6	<input checked="" type="checkbox"/>	35	LVC MOS33*	3.300		12	SLOW	NONE	FP_VTT_50
motorabl...		OUT	A10	<input checked="" type="checkbox"/>	16	LVC MOS33*	3.300		12	SLOW	NONE	FP_VTT_50
motorabl...		OUT	A11	<input checked="" type="checkbox"/>	15	LVC MOS33*	3.300		12	SLOW	NONE	FP_VTT_50
sign (4)		IN		<input checked="" type="checkbox"/>	15	LVC MOS33*	3.300				NONE	NONE
sign[3]		IN	A13	<input checked="" type="checkbox"/>	15	LVC MOS33*	3.300				NONE	NONE
sign[2]		IN	A14	<input checked="" type="checkbox"/>	15	LVC MOS33*	3.300				NONE	NONE
sign[1]		IN	A15	<input checked="" type="checkbox"/>	15	LVC MOS33*	3.300				NONE	NONE
sign[0]		IN	A16	<input checked="" type="checkbox"/>	15	LVC MOS33*	3.300				NONE	NONE
StepDrive (4)		OUT		<input checked="" type="checkbox"/>	15	LVC MOS33*	3.300		12	SLOW	NONE	FP_VTT_50
StepDriv...		OUT	C12	<input checked="" type="checkbox"/>	15	LVC MOS33*	3.300		12	SLOW	NONE	FP_VTT_50
StepDriv...		OUT	B12	<input checked="" type="checkbox"/>	15	LVC MOS33*	3.300		12	SLOW	NONE	FP_VTT_50
StepDriv...		OUT	B13	<input checked="" type="checkbox"/>	15	LVC MOS33*	3.300		12	SLOW	NONE	FP_VTT_50
StepDriv...		OUT	B14	<input checked="" type="checkbox"/>	15	LVC MOS33*	3.300		12	SLOW	NONE	FP_VTT_50
Scalar ports (5)												
clk		IN	P17	<input checked="" type="checkbox"/>	14	LVC MOS33*	3.300				NONE	NONE
Dir		IN	A18	<input checked="" type="checkbox"/>	15	LVC MOS33*	3.300				NONE	NONE
rst		IN	B16	<input checked="" type="checkbox"/>	15	LVC MOS33*	3.300				NONE	NONE
StepEna...		IN	B18	<input checked="" type="checkbox"/>	15	LVC MOS33*	3.300				NONE	NONE
SW_in		IN	B17	<input checked="" type="checkbox"/>	15	LVC MOS33*	3.300				NONE	NONE

图 3.16 引脚规划结果



第四章 软件设计

4.1 设计目的

4.1.1 服务目的

- 1) 实现景区游戏机器人所需要的人机交互要求。
- 2) 通过软件的音乐播放功能、规则介绍等功能实现对用户的引导，提升游戏机器人的可玩性。

4.1.2 宣传目的

- 1) 宣传我们的机器人，为我们进一步打开市场
- 2) 设置广告栏，为甲方的宣传以及广告招商提供便利

4.1.3 维护目的

- 1) 通过维护模式检验机器人在实际运行当中是否存在 bug。
- 2) 通过软件的维护模式工程师可直接进行对设备的检查
- 3) 满足工程师随时检查随时处理的目的，提高维护效率。

4.2 技术信息

编程语言：java、xml 布局文件语言。

编程环境：Windows 10 操作系统

编程工具：Android studio

4.3 界面设计

4.3.1 主菜单界面

- 1) 界面效果



图 4.1 主菜单

2) 设计理念

- 浅蓝色的背景图片显得明朗静美。
- 在天空之上遨游的梦想之帆寓意用户可以在游戏世界里畅游。
- 用户区域分区明确，增强用户的易用性。
- 内容简介有效，确保用户的良好游戏体验。

3) 界面跳转

- 用户可以根据自己卡片的信息进行游戏难度选择。
- 点击开始游戏按钮，可进入游戏界面，此时要求必须在机器人面前。
- 插入工程师身份卡，点击维护模式按钮可进入维护模式界面。
- 左下角的语言选择按钮，可以选择用户界面的语言，支持中英文切换
- 用户可以通过右下角的音乐控制按钮控制规则介绍音乐的开始与停止。

4.3.2 规则介绍界面

1) 界面效果

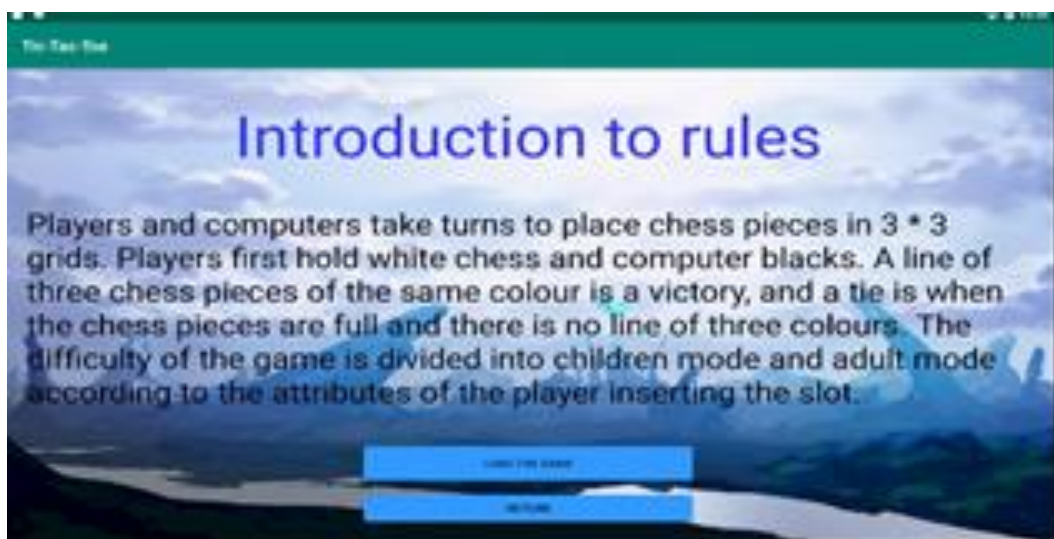


图 4.2 规则介绍界面

2) 设计理念

- 同样采用冷色调的设计
- 本界面详细介绍了井字棋游戏的游戏规则。
- 用户阅读结束后可选择进入游戏界面或者返回主菜单，更加人性化的设计，用户可据此选择继续游戏或者离开。

3) 界面跳转

- 点击加载游戏按钮可进入游戏界面。
- 点击退出按钮可返回主菜单。

4.3.3 游戏界面

1) 界面效果



图 4.3 游戏界面

2) 设计理念

- 醒目的游戏中提示可已提供给玩家正确的游戏信息。
- 玩家点机开始游戏按钮确认游戏开始。
- 二次元的画风旨在给用户提供良好的视觉享受。

3) 界面跳转

- 点击开始游戏按钮即可开始对局。
- 点击右下角的音乐按钮可选择关闭或打开游戏音乐，充分考量不同用户的需求。

4.3.4 维护模式界面

1) 界面效果



图 4.4 维护模式界面

2) 设计理念

- 背景图片是西电的标志性建筑观光塔，体现了我们的优秀的西电设计团队。
- 再加上二次元的表现力充分契合整个 app 的设计风格。
- 简洁的 Edit Text 可实现信息的精确发送。

4.3.5 游戏胜利界面

1) 界面效果





图 4.5 游戏胜利界面

2) 设计功能

- 游戏结束若玩家取得胜利 app 将会自动跳转至游戏胜利界面，及时反馈游戏结果，做到实时人机交互的功能。
- 在此界面玩家可以选择继续进行游戏或者返回主菜单。

3) 界面跳转

- 点击再来一次按钮可跳转至游戏界面，用户可再次进行游戏。
- 点机退出按钮可以返回主菜单，离开游戏。

4.3.6 游戏失败界面

1) 界面效果



图 4.6 游戏失败界面

2) 设计功能

- 游戏结束若玩家失败 app 将会自动跳转至游戏失败界面，及时反馈游戏结果，做到实时人机交互的功能。
- 在此界面玩家可以选择继续进行游戏或者返回主菜单。

3) 界面跳转



- 点击再来一次按钮可跳转至游戏界面，用户可再次进行游戏。
- 点机退出按钮可以返回主菜单，离开游戏。

4.3.7 平局界面

1) 界面效果



图 4.7 平局界面

2) 设计功能

- 游戏结束，若结果为平局 app 将会自动跳转至平局界面，及时反馈游戏结果，做到实时人机交互的功能。
- 在此界面玩家可以选择继续进行游戏或者返回主菜单。

3) 界面跳转

- 点击再来一次按钮可跳转至游戏界面，用户可再次进行游戏。
- 点机退出按钮可以返回主菜单，离开游戏。

4.4 部分后台逻辑设计

MainActivity

```
public class MainActivity extends AppCompatActivity {  
    static boolean isPlay=true;//设置音乐播放状态
```



```
MediaPlayer mediaPlayer; //定义音乐播放器对象
Button music_btn; //定义控制音乐播放按钮

private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        switch (intent.getAction()) {
            case UsbService.ACTION_USB_PERMISSION_GRANTED: //
                USB PERMISSION GRANTED
                Toast.makeText(context, "USB Ready",
                    Toast.LENGTH_SHORT).show();
                break;
            case
                UsbService.ACTION_USB_PERMISSION_NOT_GRANTED: //
                USB PERMISSION NOT GRANTED
                Toast.makeText(context, "USB Permission not granted",
                    Toast.LENGTH_SHORT).show();
                break;
            case UsbService.ACTION_NO_USB: // NO USB CONNECTED
                Toast.makeText(context, "No USB connected",
                    Toast.LENGTH_SHORT).show();
                break;
            case UsbService.ACTION_USB_DISCONNECTED: // USB DISCONNECTED
                Toast.makeText(context, "USB disconnected",
                    Toast.LENGTH_SHORT).show();
                break;
            case UsbService.ACTION_USB_NOT_SUPPORTED: // USB
```




第五章 传感器系统

5.1 TCS3472_I2C 色彩传感器

本系统在设计的过程中需要识别人员的状态，区别普通用户与维护人员，以便确定只有维护人员可以进入维护模式。所以在设计读卡时，采用了两种颜色的卡片。用户适用带有缺口的白色卡片用来识别用户信息，维护人员适用蓝色卡片以表明维护人员身份。所以需要对卡片颜色进行读取。

```
#include "mbed.h"

I2C i2c(D15, D14); //pins for I2C communication (SDA, SCL)
Serial pc(USBTX, USBRX);

int sensor_addr = 41 << 1;

DigitalOut green(LED1);
DigitalOut power(A0);

int main() {
    pc.baud(9600);
    green = 1; // off

    // Connect to the Color sensor and verify whether
    //we connected to the correct sensor.

    i2c.frequency(100000); //修改为100000，否则报错

    char id_regval[1] = {146};
```



```
char data[1] = {0};
i2c.write(sensor_addr,id_regval,1, true);
i2c.read(sensor_addr,data,1,false);

if (data[0]==68) {
    green = 0;
    wait (2);
    green = 1;
} else {
    green = 1;
}

// Initialize color sensor

char timing_register[2] = {129,0};
i2c.write(sensor_addr,timing_register,2,false);

char control_register[2] = {143,0};
i2c.write(sensor_addr,control_register,2,false);
char enable_register[2] = {128,3};
i2c.write(sensor_addr,enable_register,2,false);

// Read data from color sensor (Clear/Red/Green/Blue)

while (true) {
    char clear_reg[1] = {148};
    char clear_data[2] = {0,0};
    i2c.write(sensor_addr,clear_reg,1, true);
    i2c.read(sensor_addr,clear_data,2, false);

    int clear_value = ((int)clear_data[1] << 8) | clear_data[0];

    char red_reg[1] = {150}; //读取红色数值
    char red_data[2] = {0,0};
    i2c.write(sensor_addr,red_reg,1, true);
    i2c.read(sensor_addr,red_data,2, false);

    int red_value = ((int)red_data[1] << 8) | red_data[0];
```




```
char green_reg[1] = {152}; //读取绿色传感器数值
char green_data[2] = {0,0};
i2c.write(sensor_addr, green_reg, 1, true);
i2c.read(sensor_addr, green_data, 2, false);

int green_value = ((int)green_data[1] << 8) | green_data[0];

char blue_reg[1] = {154}; //读取蓝色数值
char blue_data[2] = {0,0};
i2c.write(sensor_addr, blue_reg, 1, true);
i2c.read(sensor_addr, blue_data, 2, false);

int blue_value = ((int)blue_data[1] << 8) | blue_data[0];
pc.printf("Clear (%d), Red (%d), Green (%d), Blue (%d)\r\n",
clear_value, red_value, green_value, blue_value);
//green!=green;
wait(0.5);
}
}
```

图 5.1~5.4 色彩传感器部分代码

5.2 VL6180 TOF 传感器

在本系统中需要保证用户与机器人距离合适，所以需要采用高精度传感器实现。

```
#define VL6180x_h

#define VL6180x_FAILURE_RESET -1

#define VL6180X_IDENTIFICATION_MODEL_ID 0x0000
#define VL6180X_IDENTIFICATION_MODEL_REV_MAJOR 0x0001
#define VL6180X_IDENTIFICATION_MODEL_REV_MINOR 0x0002
#define VL6180X_IDENTIFICATION_MODULE_REV_MAJOR 0x0003
#define VL6180X_IDENTIFICATION_MODULE_REV_MINOR 0x0004
#define VL6180X_IDENTIFICATION_DATE 0x0006 //16bit value
#define VL6180X_IDENTIFICATION_TIME 0x0008 //16bit value

#define VL6180X_SYSTEM_MODE_GPIO0 0x0010
#define VL6180X_SYSTEM_MODE_GPIO1 0x0011
#define VL6180X_SYSTEM_HISTORY_CTRL 0x0012
#define VL6180X_SYSTEM_INTERRUPT_CONFIG_GPIO 0x0014
#define VL6180X_SYSTEM_INTERRUPT_CLEAR 0x0015
#define VL6180X_SYSTEM_FRESH_OUT_OF_RESET 0x0016
#define VL6180X_SYSTEM_GROUPED_PARAMETER_HOLD 0x0017
```



```

#define VL6180X_SYSRANGE_START 0x0018
#define VL6180X_SYSRANGE_THRESH_HIGH 0x0019
#define VL6180X_SYSRANGE_THRESH_LOW 0x001A
#define VL6180X_SYSRANGE_INTERMEASUREMENT_PERIOD 0x001B
#define VL6180X_SYSRANGE_MAX_CONVERGENCE_TIME 0x001C
#define VL6180X_SYSRANGE_CROSSTALK_COMPENSATION_RATE 0x001E
#define VL6180X_SYSRANGE_CROSSTALK_VALID_HEIGHT 0x0021
#define VL6180X_SYSRANGE_EARLY_CONVERGENCE_ESTIMATE 0x0022
#define VL6180X_SYSRANGE_PART_TO_PART_RANGE_OFFSET 0x0024
#define VL6180X_SYSRANGE_RANGE_IGNORE_VALID_HEIGHT 0x0025
#define VL6180X_SYSRANGE_RANGE_IGNORE_THRESHOLD 0x0026
#define VL6180X_SYSRANGE_MAX_AMBIENT_LEVEL_MULT 0x002C
#define VL6180X_SYSRANGE_RANGE_CHECK_ENABLES 0x002D
#define VL6180X_SYSRANGE_VHV_RECALIBRATE 0x002E
#define VL6180X_SYSRANGE_VHV_REPEAT_RATE 0x0031

#define VL6180X_SYSALS_START 0x0038
#define VL6180X_SYSALS_THRESH_HIGH 0x003A
#define VL6180X_SYSALS_THRESH_LOW 0x003C
#define VL6180X_SYSALS_INTERMEASUREMENT_PERIOD 0x003E

#define VL6180X_SYSALS_ANALOGUE_GAIN 0x003F
#define VL6180X_SYSALS_INTEGRATION_PERIOD 0x0040

#define VL6180X_RESULT_RANGE_STATUS 0x004D
#define VL6180X_RESULT_ALS_STATUS 0x004E
#define VL6180X_RESULT_INTERRUPT_STATUS_GPIO 0x004F
#define VL6180X_RESULT_ALS_VAL 0x0050
#define VL6180X_RESULT_HISTORY_BUFFER 0x0052
#define VL6180X_RESULT_RANGE_VAL 0x0062
#define VL6180X_RESULT_RANGE_RAW 0x0064
#define VL6180X_RESULT_RANGE_RETURN_RATE 0x0066
#define VL6180X_RESULT_RANGE_REFERENCE_RATE 0x0068
#define VL6180X_RESULT_RANGE_RETURN_SIGNAL_COUNT 0x006C
#define VL6180X_RESULT_RANGE_REFERENCE_SIGNAL_COUNT 0x0070
#define VL6180X_RESULT_RANGE_RETURN_AMB_COUNT 0x0074
#define VL6180X_RESULT_RANGE_REFERENCE_AMB_COUNT 0x0078
#define VL6180X_RESULT_RANGE_RETURN_CONV_TIME 0x007C
#define VL6180X_RESULT_RANGE_REFERENCE_CONV_TIME 0x0080

#define VL6180X_READOUT_AVERAGING_SAMPLE_PERIOD 0x010A

```



```
#define VL6180X_FIRMWARE_BOOTUP 0x0119
#define VL6180X_FIRMWARE_RESULT_SCALER 0x0120
#define VL6180X_I2C_SLAVE_DEVICE_ADDRESS 0x0212
#define VL6180X_INTERLEAVED_MODE_ENABLE 0x02A3

/**
 * gain settings for ALS
 *Data sheet shows gain values as binary list
 */
enum vl6180x_als_gain {

GAIN_20 = 0, // Actual ALS Gain of 20
GAIN_10,     // Actual ALS Gain of 10.32
GAIN_5,      // Actual ALS Gain of 5.21
GAIN_2_5,    // Actual ALS Gain of 2.60
GAIN_1_67,   // Actual ALS Gain of 1.72
GAIN_1_25,   // Actual ALS Gain of 1.28
GAIN_1,      // Actual ALS Gain of 1.01
GAIN_40,     // Actual ALS Gain of 40

struct VL6180xIdentification {
/** Model Number */
uint8_t idModel;
/** Model Number */
uint8_t idModelRevMajor;
/** Model Number */
uint8_t idModelRevMinor;
/** Module major revision */
uint8_t idModuleRevMajor;
/** Module minor revision */
uint8_t idModuleRevMinor;
/** Manufacture date */
uint16_t idDate;
/** Manufacture time seconds after midnight */
uint16_t idTime;
};
/**
 * VL6180x tof/als sensor example
 *
 * @code
```



```

private:
    //Store address given when the class is initialized.
    //This value can be changed by the changeAddress() function
    I2C m_i2c;
    int m_addr;
/**
 * read 8 bit register
 * @param registerAddr address for register
 * @returns contents of register
 */
    uint8_t VL6180x_getRegister(uint16_t registerAddr);
/**
 * read 16 bit register
 * @param registerAddr address for register
 * @param returns contents of register
 */
    uint16_t VL6180x_getRegister16bit(uint16_t registerAddr);

uint16_t VL6180x_getRegister16bit(uint16_t registerAddr);
/**
 * write 8 bit register
 * @param registerAddr address for register
 * @returns ERROR
 */
    void VL6180x_setRegister(uint16_t registerAddr, uint8_t data);
/**
 * write 16 bit register
 * @param registerAddr address for register
 * @returns ERROR
 */
    void VL6180x_setRegister16bit(uint16_t registerAddr, uint16_t data);
};

#endif

#include "VL6180x.h"

VL6180x::VL6180x(PinName sda, PinName scl, uint8_t addr) : m_i2c(sda, scl), m_addr(addr) {}
int VL6180x::VL6180xInit(void){
    uint8_t data; //for temp data storage

    data = VL6180x_getRegister(VL6180X_SYSTEM_FRESH_OUT_OF_RESET);
    wait_ms(50);
    if(data != 1) return VL6180x_FAILURE_RESET;

    //Required by datasheet
    //http://www.st.com/st-web-ui/static/active/en/resource/technical/document/application_note.
    VL6180x_setRegister(0x0207, 0x01);
    VL6180x_setRegister(0x0208, 0x01);
    VL6180x_setRegister(0x0096, 0x00);
    VL6180x_setRegister(0x0097, 0xfd);
    VL6180x_setRegister(0x00e3, 0x00);
    VL6180x_setRegister(0x00e4, 0x04);
    VL6180x_setRegister(0x00e5, 0x02);
    VL6180x_setRegister(0x00e6, 0x01);

```



```
VL6180x::~VL6180x(void) {
};

void VL6180x::VL6180xDefaultSettings(void) {
    //Recommended settings from datasheet
    //http://www.st.com/st-web-ui/static/active/en/resource/technical/document/application_note/DM00122600.pdf

    //Enable Interrupts on Conversion Complete (any source)
    VL6180x_setRegister(VL6180X_SYSTEM_INTERRUPT_CONFIG_GPIO, (4 << 3) | (4)); // Set GPIO1 high when sample complete

    VL6180x_setRegister(VL6180X_SYSTEM_MODE_GPIO1, 0x10); // Set GPIO1 high when sample complete
    VL6180x_setRegister(VL6180X_READOUT_AVERAGING_SAMPLE_PERIOD, 0x30); //Set Avg sample period
    VL6180x_setRegister(VL6180X_SYSALS_ANALOGUE_GAIN, 0x46); // Set the ALS gain
    VL6180x_setRegister(VL6180X_SYSRANGE_VHV_REPEAT_RATE, 0xFF); // Set auto calibration period (Max = 255)/(OFF = 0)
    VL6180x_setRegister(VL6180X_SYSALS_INTEGRATION_PERIOD, 0x63); // Set ALS integration time to 100ms
    VL6180x_setRegister(VL6180X_SYSRANGE_VHV_RECALIBRATE, 0x01); // perform a single temperature calibration
    //Optional settings from datasheet
    //http://www.st.com/st-web-ui/static/active/en/resource/technical/document/application_note/DM00122600.pdf
    VL6180x_setRegister(VL6180X_SYSRANGE_INTERMEASUREMENT_PERIOD, 0x09); // Set default ranging inter-measurement period to 100ms

    VL6180x_setRegister(VL6180X_SYSRANGE_INTERMEASUREMENT_PERIOD, 0x09); // Set default ranging inter-measurement period to 100ms
    VL6180x_setRegister(VL6180X_SYSALS_INTERMEASUREMENT_PERIOD, 0x0A); // Set default ALS inter-measurement period to 100ms
    VL6180x_setRegister(VL6180X_SYSTEM_INTERRUPT_CONFIG_GPIO, 0x24); // Configures interrupt on 缺棉ew Sample Ready threshold event缺?
    //Additional settings defaults from community
    VL6180x_setRegister(VL6180X_SYSRANGE_MAX_CONVERGENCE_TIME, 0x32);
    VL6180x_setRegister(VL6180X_SYSRANGE_RANGE_CHECK_ENABLES, 0x10 | 0x01);
    VL6180x_setRegister16bit(VL6180X_SYSRANGE_EARLY_CONVERGENCE_ESTIMATE, 0x7B);
    VL6180x_setRegister16bit(VL6180X_SYSALS_INTEGRATION_PERIOD, 0x64);

    VL6180x_setRegister(VL6180X_READOUT_AVERAGING_SAMPLE_PERIOD, 0x30);
    VL6180x_setRegister(VL6180X_SYSALS_ANALOGUE_GAIN, 0x40);
    VL6180x_setRegister(VL6180X_FIRMWARE_RESULT_SCALER, 0x01);
}

void VL6180x::getIdentification(struct VL6180xIdentification *temp) {
    temp->idModel = VL6180x_getRegister(VL6180X_IDENTIFICATION_MODEL_ID);
    temp->idModelRevMajor = VL6180x_getRegister(VL6180X_IDENTIFICATION_MODEL_REV_MAJOR);
    temp->idModelRevMinor = VL6180x_getRegister(VL6180X_IDENTIFICATION_MODEL_REV_MINOR);
    temp->idModuleRevMajor = VL6180x_getRegister(VL6180X_IDENTIFICATION_MODULE_REV_MAJOR);
    temp->idModuleRevMinor = VL6180x_getRegister(VL6180X_IDENTIFICATION_MODULE_REV_MINOR);
}

uint8_t VL6180x::changeAddress(uint8_t old_address, uint8_t new_address) {

    //NOTICE: IT APPEARS THAT CHANGING THE ADDRESS IS NOT STORED IN NON-VOLATILE MEMORY
    // POWER CYCLING THE DEVICE REVERTS ADDRESS BACK TO 0X29

    if( old_address == new_address) return old_address;
    if( new_address > 127) return old_address;

    VL6180x_setRegister(VL6180X_I2C_SLAVE_DEVICE_ADDRESS, new_address);
    // mbed needs the new address
    m_addr=new_address<<1;
    return VL6180x_getRegister(VL6180X_I2C_SLAVE_DEVICE_ADDRESS);
}

uint8_t VL6180x::getDistance() {
    uint8_t distance;
    VL6180x_setRegister(VL6180X_SYSRANGE_START, 0x01); //Start Single shot mode
    wait_ms(10);
}

uint8_t VL6180x::getDistance() {
    uint8_t distance;
    VL6180x_setRegister(VL6180X_SYSRANGE_START, 0x01); //Start Single shot mode
    wait_ms(10);
    distance = VL6180x_getRegister(VL6180X_RESULT_RANGE_VAL);
    VL6180x_setRegister(VL6180X_SYSTEM_INTERRUPT_CLEAR, 0x07);
    return distance;
}

float VL6180x::getDistance_m() {
    float distance;
    VL6180x_setRegister(VL6180X_SYSRANGE_START, 0x01); //Start Single shot mode
    wait_ms(10);
    distance = 0.001*(float) VL6180x_getRegister(VL6180X_RESULT_RANGE_VAL);
    VL6180x_setRegister(VL6180X_SYSTEM_INTERRUPT_CLEAR, 0x07);
    return distance;
}
```



```

float VL6180x::getAmbientLight(vl6180x_als_gain VL6180X_ALS_GAIN)
{
    //First load in Gain we are using, do it everytime incase someone changes it on us.
    //Note: Upper nibble should be set to 0x4 i.e. for ALS gain of 1.0 write 0x46
    VL6180x_setRegister(VL6180X_SYSALS_ANALOGUE_GAIN, (0x40 | VL6180X_ALS_GAIN)); // Set the ALS gain

    //Start ALS Measurement
    VL6180x_setRegister(VL6180X_SYSALS_START, 0x01);

    wait_ms(100); //give it time...

    VL6180x_setRegister(VL6180X_SYSTEM_INTERRUPT_CLEAR, 0x07);

    //Retrieve the Raw ALS value from the sensor
    unsigned int alsRaw = VL6180x_getRegister16bit(VL6180X_RESULT_ALS_VAL);

    //Get Integration Period for calculation, we do this everytime incase someone changes it on us.
    unsigned int alsIntegrationPeriodRaw = VL6180x_getRegister16bit(VL6180X_SYSALS_INTEGRATION_PERIOD);

    float alsIntegrationPeriod = 100.0 / alsIntegrationPeriodRaw ;

    float alsGain = 0.0;

    switch (VL6180X_ALS_GAIN){
        case GAIN_20: alsGain = 20.0; break;
        case GAIN_10: alsGain = 10.32; break;
        case GAIN_5: alsGain = 5.21; break;
        case GAIN_2_5: alsGain = 2.60; break;
        case GAIN_1_67: alsGain = 1.72; break;
        case GAIN_1_25: alsGain = 1.28; break;
        case GAIN_1: alsGain = 1.01; break;
        case GAIN_40: alsGain = 40.0; break;
    }

    //Calculate LUX from formula in AppNotes

    float alsCalculated = (float)0.32 * ((float)alsRaw / alsGain) * alsIntegrationPeriod;

    return alsCalculated;
}

uint8_t VL6180x::VL6180x_getRegister(uint16_t registerAddr)
{
    uint8_t data;
    char data_write[2];
    char data_read[1];
    data_write[0] = (registerAddr >> 8) & 0xFF; //MSB of register address
    data_write[1] = registerAddr & 0xFF; //LSB of register address
    m_i2c.write(m_addr, data_write, 2, 0);
    m_i2c.read(m_addr, data_read, 1, 1);
    //Read Data from selected register
    data=data_read[0];
    return data;
}

uint16_t VL6180x::VL6180x_getRegister16bit(uint16_t registerAddr)
{
    uint8_t data_low;
    uint8_t data_high;
    uint16_t data;

```



```
void VL6180x::VL6180x_setRegister(uint16_t registerAddr, uint8_t data)
{
    char data_write[3];
    data_write[0] = (registerAddr >> 8) & 0xFF; //MSB of register address
    data_write[1] = registerAddr & 0xFF; //LSB of register address
    data_write[2] = data & 0xFF;
    m_i2c.write(m_addr, data_write, 3);
}

void VL6180x::VL6180x_setRegister16bit(uint16_t registerAddr, uint16_t data)
{
    char data_write[4];
    data_write[0] = (registerAddr >> 8) & 0xFF; //MSB of register address
    data_write[1] = registerAddr & 0xFF; //LSB of register address
    data_write[2] = (data >> 8) & 0xFF;
    data_write[3] = data & 0xFF;
    m_i2c.write(m_addr, data_write, 4);
}

#include "mbed.h"
#include <VL6180x.h>

/*const float GAIN_1      = 1.01; // Actual ALS Gain of 1.01
const float GAIN_1_25    = 1.28; // Actual ALS Gain of 1.28
const float GAIN_1_67    = 1.72; // Actual ALS Gain of 1.72
const float GAIN_2_5     = 2.6;  // Actual ALS Gain of 2.60
const float GAIN_5       = 5.21; // Actual ALS Gain of 5.21
const float GAIN_10      = 10.32; // Actual ALS Gain of 10.32
const float GAIN_20      = 20;    // Actual ALS Gain of 20
const float GAIN_40      = 40;    // Actual ALS Gain of 40
*/

#define VL6180X_ADDRESS 0x29
Serial pc(USBTX, USBRX);
VL6180xIdentification identification;
// mbed uses 8bit addresses shift address by 1 bit left
VL6180x sensor(D14, D15, VL6180X_ADDRESS<<1);

void printIdentification(struct VL6180xIdentification *temp){
```



```
void printIdentification(struct VL6180xIdentification *temp){
    printf("Model ID = ");
    printf("%d\n",temp->idModel);

    printf("Model Rev = ");
    printf("%d",temp->idModelRevMajor);
    printf(".");
    printf("%d\n",temp->idModelRevMinor);

    printf("Module Rev = ");
    printf("%d",temp->idModuleRevMajor);
    printf(".");
    printf("%d\n",temp->idModuleRevMinor);

    printf("Manufacture Date = ");
    printf("%d", ((temp->idDate >> 3) & 0x001F));
    printf("/");
    printf("%d", ((temp->idDate >> 8) & 0x000F));
    printf("/1");
}

int main() {

    wait_ms(100); // delay .1s

    sensor.getIdentification(&identification); // Retrieve manufacture info from device memory
    printIdentification(&identification); // Helper function to print all the Module information

    if(sensor.VL6180xInit() != 0){
        pc.printf("FAILED TO INITIALIZE\n"); //Initialize device and check for errors
    };

    sensor.VL6180xDefaultSettings(); //Load default settings to get started.

    wait_ms(1000); // delay 1s
```

图 5.5~5.23 距离传感器部分代码