

第九届

# 全国大学生集成电路创新创业大赛

报告类型： 技术报告

参赛杯赛： 海云捷讯杯

作品名称： 基于海云捷讯 FPGA 的机械臂精确控制系统设计

队伍编号： CICC0902660

团队名称： I'm milk dragon

# 基于海云捷讯 FPGA 的机械臂精确控制系统设计 —— —— 快速预览简介

## 一、核心系统架构

本作品以海云捷讯 AWC\_C4 DVK 开发板的 Cyclone IV 系列 FPGA (EP4CE6F17C8N) 为核心，构建了集图像采集、处理、机械臂控制于一体的精确控制系统。系统外接 OV5640 摄像头（720P 分辨率，RGB565 格式）实时采集图像，经 FPGA 处理后通过 VGA 接口显示；SDRAM 模块（HY57V2562GTR，256Mbit）用于图像缓存，采用乒乓操作实现跨时钟域（84MHz 摄像头输入 / 75MHz VGA 输出）数据同步；通过串口与 6 自由度飞特总线舵机通信，完成物块抓取与放置。

## 二、关键技术亮点

### 1. 视觉处理优化方案

**色块识别：**采用 RGB 转 YCbCr 颜色空间转换（公式：

$Y=0.299R+0.587G+0.114B$ ，Cb、Cr 同理），相比 HSV 减少 FPGA 逻辑资源占用，实现 95%-100% 的颜色识别正确率（红色、蓝色 100%，黄色 98%，黑色 95%）。

**边缘检测：**基于 Sobel 算子的 4 级流水处理，通过缓存 3 列像素计算水平 / 垂直梯度，快速提取物块边框，为形状识别奠定基础。

**角度计算：**结合颜色角点（Z 字型扫描逐次逼近获取）与边缘角点（边框范围内检测），利用 CORDIC 算法实现反正切计算，正方形、三角形角度误差  $\pm 2^\circ - 6^\circ$ ，圆形、六边形通过对称性修正后精度提升。

### 2. 机械臂控制高效实现

**逆运动学解算：**采用基于余弦定理的方案，通过 Matlab 预计算所有可能坐标对应的关节角度及 PWM 值，生成 ROM 表（如以 X、Y 为地址存储 PWM0 和 R 值），FPGA 查表调用，减少 90% 以上实时计算资源占用，角度误差  $\pm 0.1^\circ$ ，末端定位误差  $< 1\text{mm}$ 。

**PWM 控制：**12 位精度（4096 级），舵机角度误差  $\pm 1$  步进，配合状态机实现抓取 - 放置全流程自动化，单目标处理链路延迟  $< 20\text{ms}$ 。

## 三、核心性能指标

**识别精度：**颜色识别综合正确率 98%，形状识别中正方形 98%、三角形 93%、圆形 90%、六边形 89%；角度计算误差最大  $6^\circ$ （正方形  $135^\circ$  时）。

**控制精度：**机械臂抓取成功率 86%-96%（中心区域最高 96%），逆运动学解算末端误差  $< 1\text{mm}$ ，PWM 输出精度满足舵机  $0.09^\circ/\text{step}$  控制需求。

**实时性：**图像处理支持 30fps 视频流，单帧处理时间  $< 33\text{ms}$ ；控制闭环（图像采集到指令输出）延迟  $< 20\text{ms}$ 。

**资源占用：**采用 ROM 查表方案后，FPGA 逻辑资源占用 84%，记忆资源占用 75%。

系统已满足赛题难度 1、2 要求，当前物块放置精度 90%。

## 目录

1 系统架构分析.....	6
1.1 赛题介绍.....	6
1.2 系统总体设计.....	6
1.3 硬件设计.....	7
1.3.1 SDRAM 模块.....	8
1.3.2 摄像头模块.....	9
1.3.3 VGA 显示接口模块.....	11
1.3.4 USB Hub 电路模块.....	12
1.3.5 串口通信电路模块.....	12
1.3.6 USB Blaster II 电路模块.....	13
1.3.7 机械臂模块.....	14
2 关键技术分析.....	17
2.1 视觉处理部分.....	17
2.1.1 色块识别.....	17
2.1.2 Sobel 边缘检测.....	18
2.1.3 旋转角度识别.....	21
2.2 机械臂逆运动学计算部分.....	26
2.3 机械臂控制部分.....	28
2.4 动作组.....	30
2.4.1 抓取.....	30
2.4.2 仓库部分.....	31
3 性能分析.....	32
3.1 图像处理算法仿真验证.....	32
3.1.1 RGB 转 YCbCr 算法仿真验证.....	32
3.1.2 二值化算法仿真验证.....	32

3.1.3 腐蚀膨胀算法仿真验证.....	33
3.1.4 生长算法验证.....	34
3.1.5 CORDIC 算法.....	34
3.2 机械臂控制算法仿真验证.....	35
3.2.1 PWM 生成仿真验证.....	35
3.2.2 数字转字符串仿真验证.....	35
3.2.3 机械臂指令发送模块验证.....	35
3.2.4 上板测试.....	36
3.3 控制精度分析.....	36
3.3.1 逆运动学解算精度.....	36
3.3.2 PWM 精度与舵机控制.....	36
3.4 系统实时性分析.....	37
3.4.1 数据流与图像处理实时性.....	37
3.4.2 控制链路响应速度.....	37
3.5 FPGA 资源利用率分析.....	37
4 性能分析与测试.....	39
4.1 颜色识别准确度测试.....	39
4.2 图形形状识别.....	40
4.3 机械臂运动精度.....	41
4.4 角度计算精度.....	41
4.5 透明外框角度识别.....	42
5 结论.....	43

# 1 系统架构分析

## 1.1 赛题介绍

本赛题内容为控制多自由度机械臂完成将一定数量含有不同形状和颜色图案的正方形物块，按照指定规则进行仓库至目的地的移动、拼接和还原。

示意图如图 1 所示：

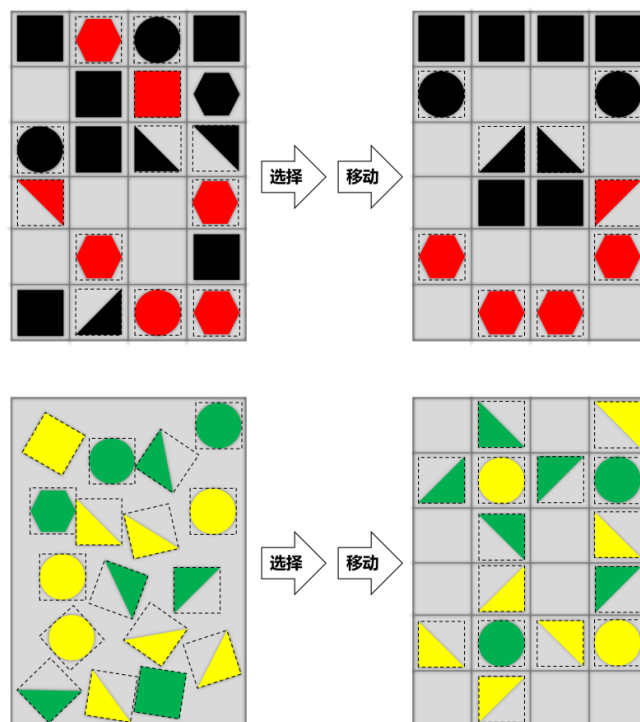


图 1.1 设计要求示意图

本赛题要实现的内容：

1、传感器数据处理：FPGA 可以实时处理来自 OV5640 图像传感器的数据，对图像数据实时进行 RGB 转 YCbCr、图像二值化、生长算法、目标检测、颜色识别、计算坐标等算法实现。

2、运动控制算法实现：通过 FPGA 实现机械臂的运动控制算法，在 Matlab 中完成逆运动学计算、ROM 设计、轨迹规划控制。将这些算法可以通过硬件描述语言实现，利用 FPGA 的并行计算能力对算法加速处理。

3、通信接口设计：本设计需要用到 IIC、UART、VGA 等接口，实现摄像头参数配置、机械臂串口通信、调试串口通信、图像显示等功能。各接口通过 FPGA 并行处理能力实现同步运行，确保摄像头配置、机械臂控制、调试监控及图像显示功能高效协同。

## 1.2 系统总体设计

本系统以 FPGA（EP4CE6F17C8）为核心控制器，构建起集图像处理、特征提取、机械臂控制于一体的高精度运动控制系统，实现对目标物体的精准抓取与

操作。系统外接 OV5640 摄像头实时采集场景图像，经 FPGA 内部图像处理模块进行 RGB565 转 YCbCr、灰度二值化、腐蚀膨胀算法、目标检测生长算法和颜色识别等处理，提升图像质量并提取目标特征、坐标等数据，处理后将图像数据存入 SDRAM 缓存，同时通过 VGA 接口输出至显示器以便实时监控。特征提取模块对处理后的图像进行分析，计算目标物体的坐标、形状、颜色等关键信息并传递给机械臂控制模块，该模块结合逆运动学计算、轨迹规划等算法生成机械臂各关节控制指令，通过机械臂控制串口发送至驱动模块，精准驱动机械臂执行抓取动作。目标图片 RAM 用于存储目标物体特征信息以便快速调用匹配，PC 通过调试串口与 FPGA 通信，可发送目标信息或接收系统状态数据进行调试监控。工作时，OV5640 先采集图像，经 FPGA 处理与特征提取识别目标，机械臂控制模块中有预先在 Matlab 中计算好的逆运动学 ROM，驱动机械臂完成抓取，同时 VGA 实时显示图像、PC 端辅助调试确保系统稳定。本组利用 FPGA 并行计算特性实现图像处理与控制算法高效运行，缩短响应时间，提升实时性，模块化设计增强可维护性与扩展性，各模块协同保障机械臂运动的高精度与稳定性，通过整合多模块功能实现图像感知与机械臂控制的深度融合，为机械臂精确控制提供高效可靠的解决方案。整个系统框图如下 图 1.2 所示：

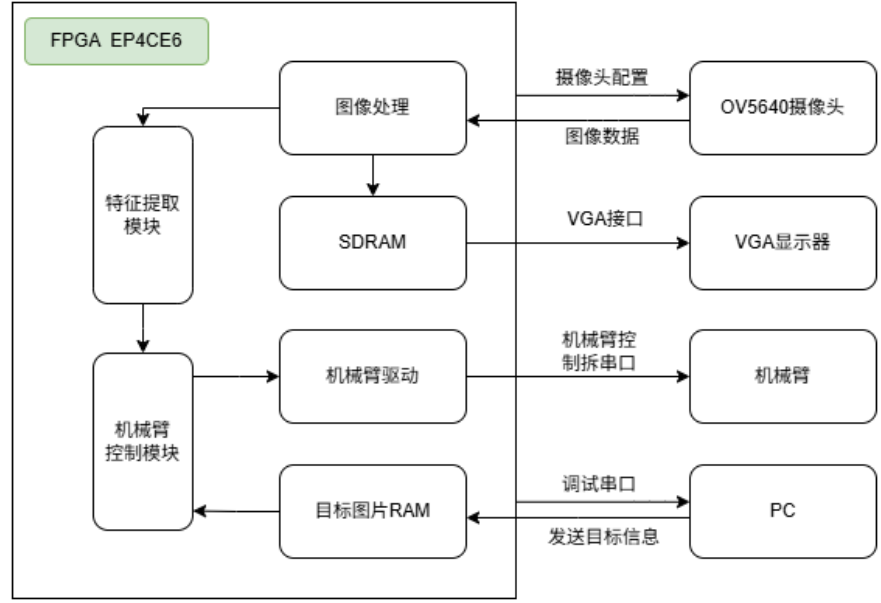


图 1.2 系统总体框图

### 1.3 硬件设计

AWC\_C4 DVK 是 Cyclone IV E 系列 EP4CE6F17C8N 的一款开发板，芯片采用 28 纳米工艺。它集成了 6272 个逻辑单元、179 个 I/O 接口，封装为 FBGA - 256 形式，工作电压范围 1.15V~1.25V，工作温度范围 0℃~+85℃，总 RAM 位数达 276480bit。该芯片具备高性能，依托先进工艺与充足逻辑单元，可实现高速运算与存储；支持用户根据需求编程，灵活实现不同场景下的功能；开发板板载 SDRAM 为 256Mb，用于缓存图像数据、中间计算结果等，

支持 FPGA 高速读写，保障系统数据处理流畅。VGA 接口 传输模拟信号，能连接显示器输出处理后的图像，便于调试与显示。摄像头模块接口可接入摄像头，为 FPGA 提供图像采集输入，适用于物体识别、图像处理等场景。按键为轻触式，用于触发功能或调整参数；拨码开关则通过滑动设置状态组合，直观输入配置信息。拓展口可外接传感器、控制板等设备，灵活扩展系统功能，增强开发板适用性。其他如电源、指示灯等部分为基础支持，确保各模块稳定运行。下图为开发板示意图如 1.3 所示：

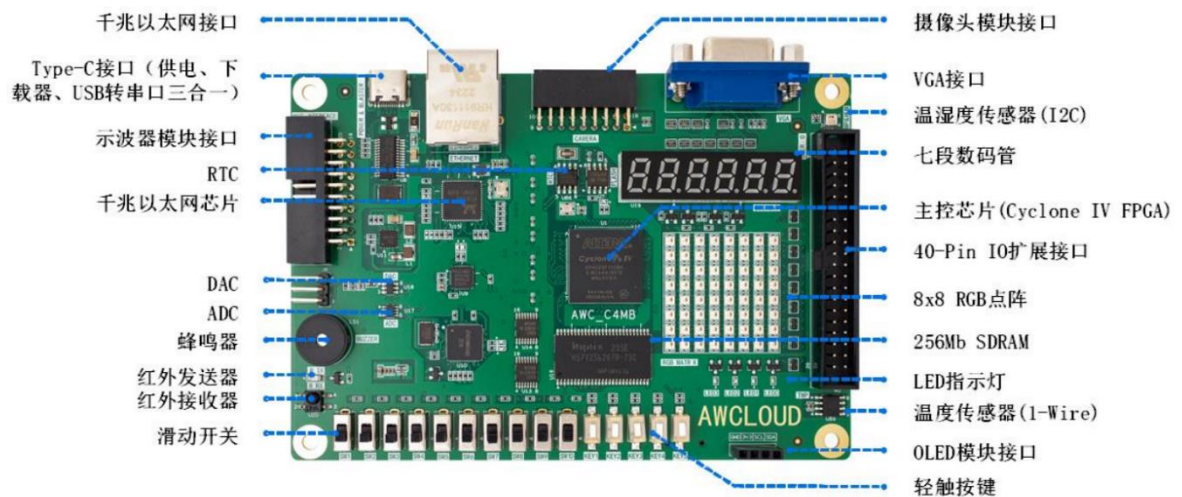


图 1.3 开发板示意图

### 1.3.1 SDRAM 模块

AWC\_C4 DVK 上设计有 SDRAM 电路，主要用于协助主控芯片内部的命令发送以及数据传输等。在图像处理系统中，需要对图像视频数据进行缓存，所以需要大容量存储器。SDRAM 有价格低廉、容量大等优点，常作为 FPGA 的外挂存储器。同步是指其工作需要同步时钟，命令的发送与数据的传输都以时钟为基准；动态是指存储阵列需要不断的刷新来保证电容存储体中的数据不丢失；随机是指数据可以自由指定地址进行数据读写。该开发板采用的是 SK Hynix 公司 SDRAM，芯片型号为：HY57V2562GTR，容量为 256Mbit，（4Mx4Bankx16IO）同步动态随机存储器。使用的数据位宽为 16bit，最大工作频率为 133MHz。

在项目中，OV5640 摄像头负责图像采集，利用 SDRAM 对处理后的图像数据进行的乒乓操作，和跨时钟域处理。乒乓操作是一种高效的数据缓冲与处理技术，利用存储模块交替进行数据写入和读出。在第一个缓冲周期，数据写入模块 1；第二个周期，数据写入模块 2，同时读出模块 1 的数据，循环往复。通过输入输出数据流选择单元的协同切换，使输入输出数据流连续不断，可实现数据的无缝缓冲与处理。而跨时钟域操作，因 OV5640 输入的图像时钟 PCLK 为 84Mhz，VGA 的输出时钟为 75Mhz，直接操作易引发数据错误，利用 SDRAM 进行跨时钟域同步，有效解决因时钟域不同导致的数据一致性问题。



SDRAM 电路图如下图 1.4 所示：

## SDRAM

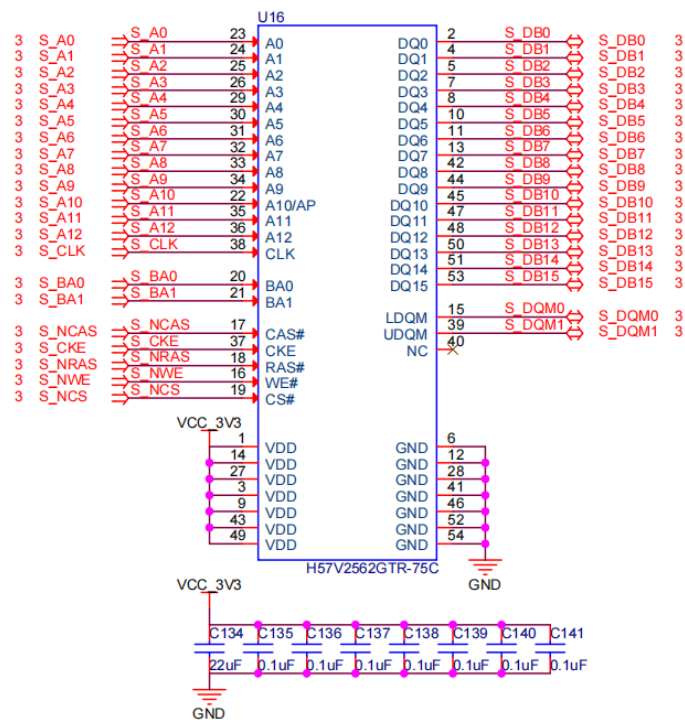


图 1.4 SDRAM 电路图

SDRAM 模块引脚定义如

表 1.1 所示：

表 1.1 SDRAM 模块引脚定义表

信号名称	端口说明
S_DB0 ~ S_DB15	SDRAM 数据线 0 ~ 15
S_DQM0 ~ S_DQM1	SDRAM 数据掩码 0 ~ 1
S_A0 ~ S_A12	SDRAM 地址线 0 ~ 12
S_BA0 ~ S_BA1	SDRAM Bank 地址线 0 ~ 1
S_CLK	SDRAM 时钟输入
S_NCAS	SDRAM 列地址选通
S_CKE	SDRAM 时钟使能
S_NRAS	SDRAM 行地址选通
S_NWE	SDRAM 写使能
S_NCS	SDRAM 片选

### 1.3.2 摄像头模块

该项目用到摄像头模块，型号为 OV5640，实物图如图 1.5 所示：

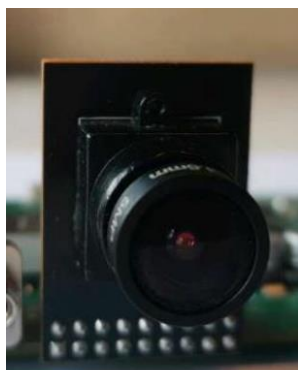


图 1.5 摄像头实物图

OV5640 是一款 CMOS 类型数字图像传感器，该传感器支持输出最大为 500 万像素的图像 (2592x1944 分辨率)，支持使用 VGA 时序输出图像数据，输出图像的数据格式支持 YUV(422/420)、YCbCr422、RGB565 以及 JPEG 格式。它还可以对采集得的图像进行补偿，支持伽玛曲线、白平衡、饱和度、色度等基础处理。根据不同的分辨率配置，传感器输出图像数据的帧率从 15-60 帧可调。其内部有许多寄存器，用来配置摄像头的工作方式、图像格式等等；在实际应用时需要先使用 SCCB 协议或者是 I2C 协议配置寄存器，使摄像头按照常见的 VGA 时序输出图像数据。在本次设计中，使用 I2C 协议配置摄像头，并将 OV5640 分辨率配置为 720P、数据格式为 RGB565 输出。摄像头接口原理图如下图 1.6 所示：

## Camera

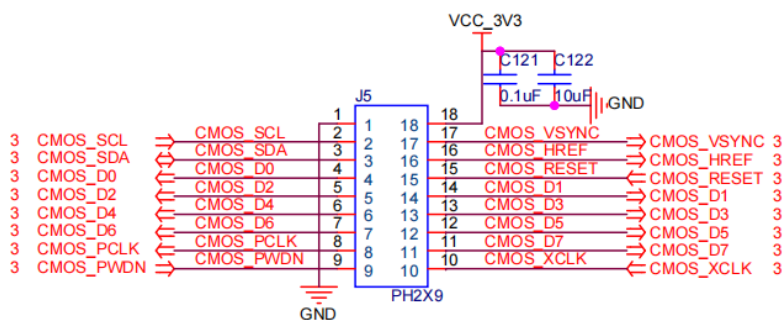


图 1.6 摄像头接口原理图

摄像头接口模块引脚定义如

表 1.2 所示：

表 1.2 摄像头接口引脚定义表

信号名称	端口说明
CMOS_D0 ~ CMOS_D7	摄像头数据线 D0 ~ D7
CMOS_SCL	摄像头 I2C 时钟线
CMOS_SDA	摄像头 I2C 数据线

CMOS_PWDN	摄像头掉电使能
CMOS_HREF	摄像头行同步信号
CMOS_VSYNC	摄像头帧同步信号
CMOS_RESET	摄像头复位
CMOS_XCLK	摄像头驱动时钟
CMOS_PCLK	摄像头像素时钟

### 1.3.3 VGA 显示接口模块

VGA 是视频图像阵列，通常指代 VGA 接口。它是 IBM 在 1987 年推出的一种视频传输标准，具有分辨率高、显示速率快、颜色丰富等优点，在彩色显示器领域得到了广泛的应用。它的显示原理是：CRT（阴极射线管）以“Z”型扫描的方式将阴极射线枪发出的电子束打在涂有荧光粉的荧光屏上，产生 RGB 三基色合成一个彩色像素，每扫完一行，电子枪都要回到屏幕的下一行左边的起始位置，并且在换行的过程中，电子枪需要消隐，避免破坏已经成像的像素。

AWC\_C4 DVK 上设计有 1 个 15 针的 D-SUB 接口，主要用于 VGA 输出。开发板设计采用的是 R-2R 电阻梯形网络模拟 DAC（数模转换器）的简易方案替换例如 ADV 系列的视频转换芯片，将数字信号转换为三种基色（红、绿、蓝）模拟信号，以此降低成本。输出图像目前能支持 1024\*768、1280\*1024 等分辨率。VGA 接口电路示意图如图 1.7 所示：

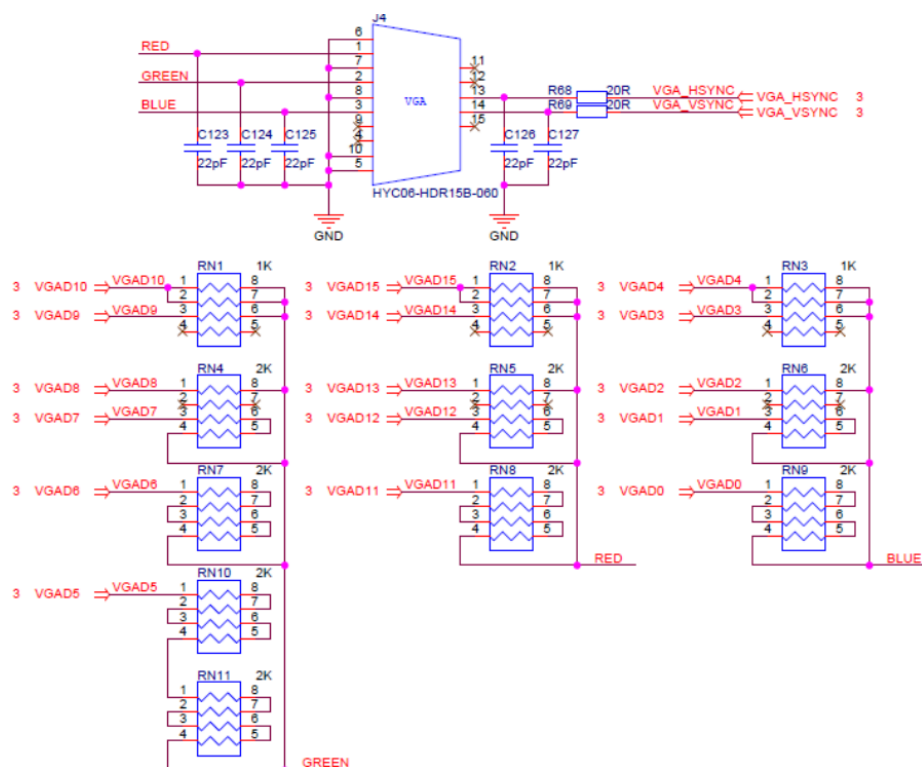


图 1.7 VGA 接口电路图

VGA 接口模块引脚定义如 表 1.3 所示：

表 1.3 VGA 接口模块引脚定义表

信号名称	端口说明
VGAD0 ~ VGAD4	BLUE 数据输出 0 ~ 4
VGAD5 ~ VGAD10	GREEN 数据输出 0 ~ 5
VGAD11 ~ VGAD15	RED 数据输出 0 ~ 4

1.3.4 USB Hub 电路模块

AWC\_C4 DVK 上设计有 1 个 USB Hub 电路，主要用于扩展 USB 接口，以便于减少板上 USB 连接器数量从而优化板卡结构及空间。该部分电路采用 TERMINUS 公司的 USB 芯片，型号为 FE1.1S，分出两路 USB 数据通道。其中一路用于串口通信，另一路则为 USB Blaster II 下载器使用。USB Hub 电路如图 1.8 所示：

USB Hub

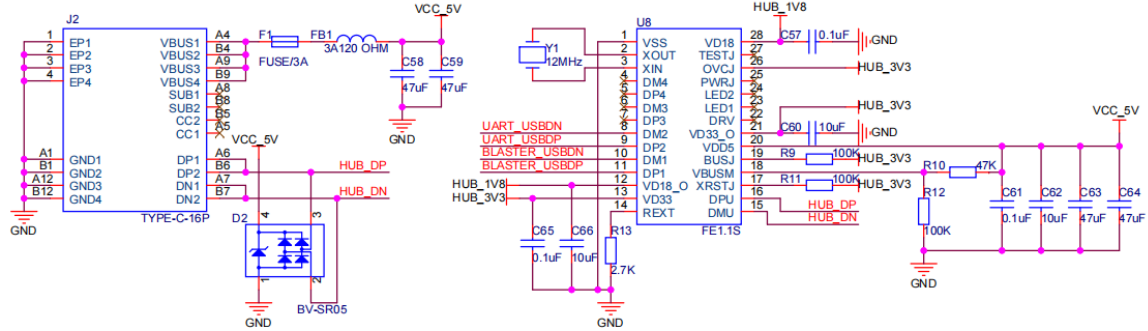


图 1.8 USB Hub 电路图

USB Hub 模块引脚定义如 表 1.4 所示：

表 1.4 USB Hub 模块引脚定义表

信号名称	端口说明
HUB_DP	USB 差分信号 P
HUB_DN	USB 差分信号 N

1.3.5 串口通信电路模块

AWC\_C4 DVK 上设计有 1 个 USB 转 UART 电路，主要用于串口通信。转换芯片采用的是 SILICON LABS 公司的 USB 芯片，型号为 CP2102-GMR，硬件电路如图 1.9 所示：

## UART-&gt;USB

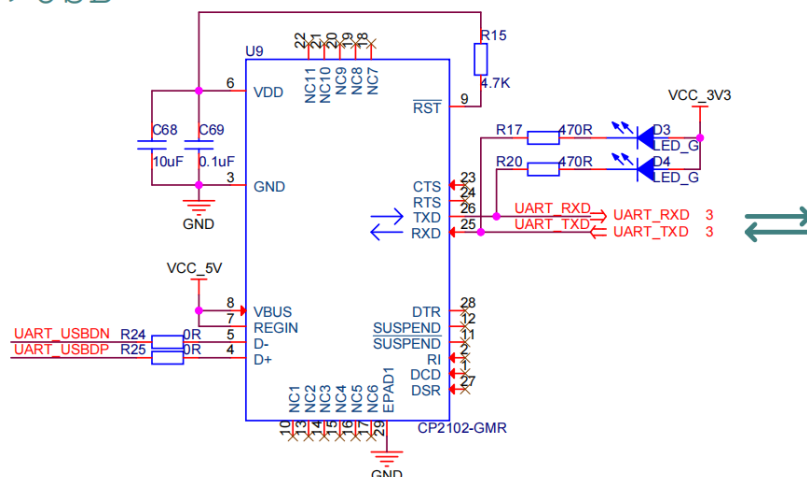


图 1.9 USB 硬件原理图

串口模块引脚定义如

表 1.5 所示:

表 1.5 串口模块引脚定义表

信号名称	端口说明
UART_TXD	UART 数据输出
UART_RXD	UART 数据输入

### 1.3.6 USB Blaster II 电路模块

AWC\_C4 DVK 上设计有 1 个 USB Blaster II 电路，主要用于主控芯片程序下载及调试。通常情况下，开发板上会设计有 1 个 JTAG 接口，可通过使用 USB Blaster 下载器连接电脑 PC 和该接口，即可下载程序到主控芯片或者固化程序到 Flash 后对系统进行调试与测试。但在本次设计中，不仅仅将 USB Blaster 下载器板载到开发板上节约成本，并且将其迭代至二代大大提升程序下载速率。USB Blaster 二代下载器芯片采用的是小梅哥的 JTAG 芯片，型号为 JTAG18M01HS2。此外，为匹配主控芯片上专用 JTAG 引脚电平要求（2.5V），采用 TI 公司的 4 位双电源总线收发器芯片进行配置电压转换，型号为 SN74AVC4T245PWR。USB Blaster II 电路如图 1.10 所示：

USB Blaster II

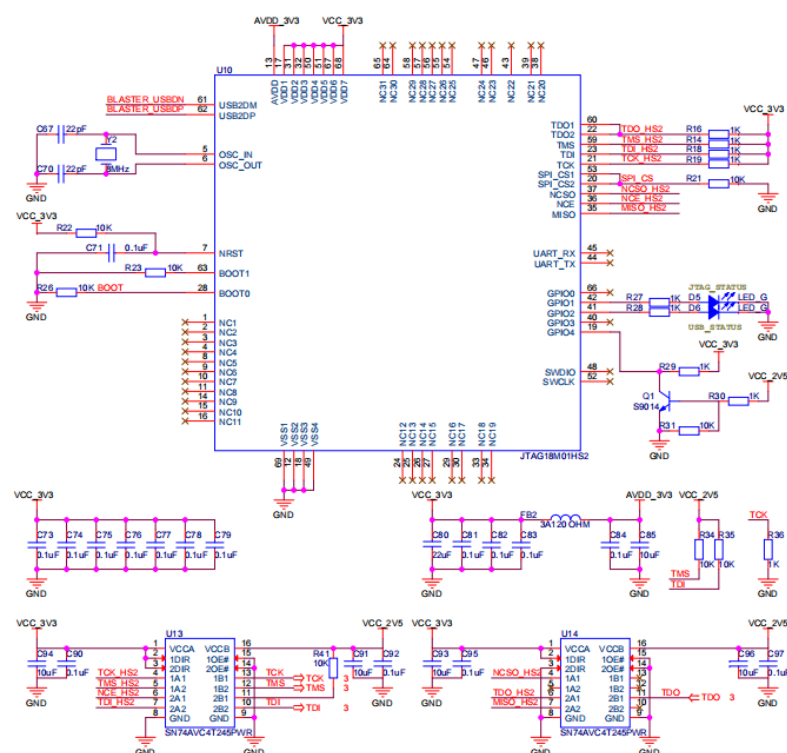


图 1.10 USB Blaster II 电路图

USB Blaster II 电路引脚定义如 表 1.6 所示：

表 1.6 USB Blaster II 电路引脚定义表

信号名称	端口说明
TCK	JTAG 时钟输入
TMS	JTAG 模式选择输入
TDI	JTAG 数据输入
TDO	JTAG 数据输出

1.3.7 机械臂模块

本项目使用到的机械臂为飞特的 6 自由度智能机械臂，实物图如图 1.11 所示：



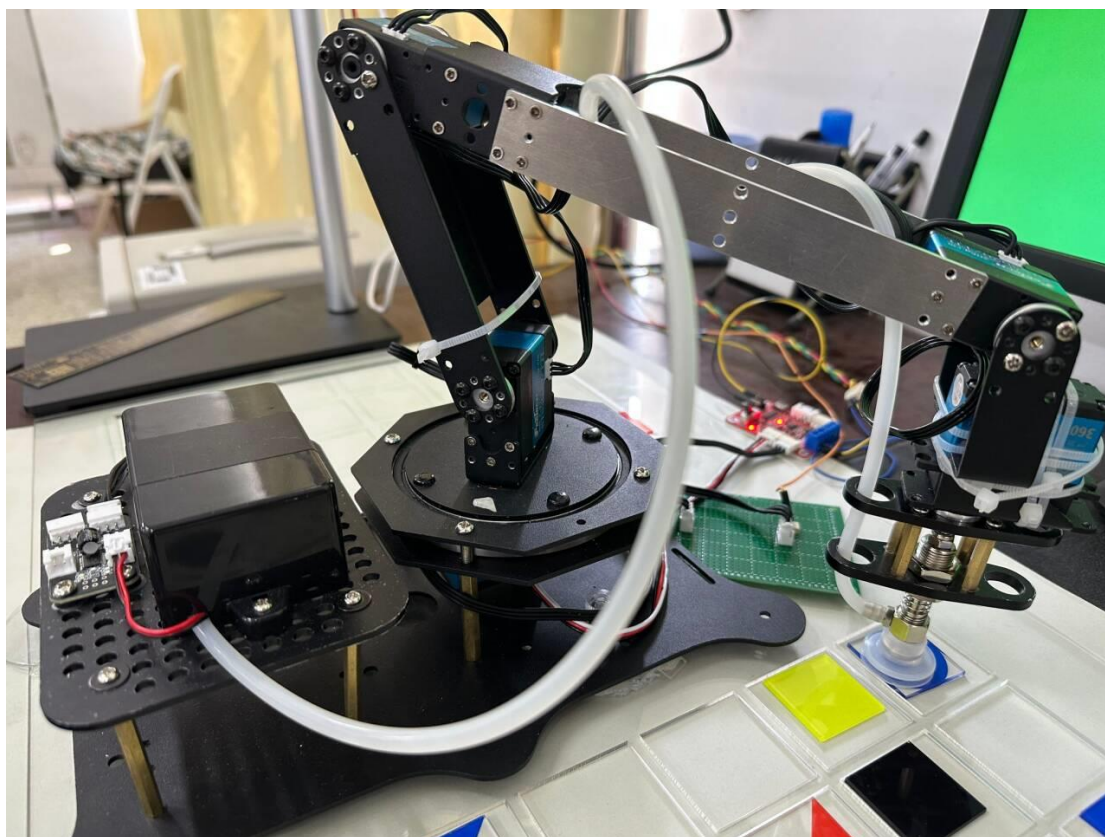


图 1.11 机械臂示意图

六自由度机械臂是一种具备六个独立运动自由度的机器人臂，通过六个关节如基座旋转、下臂摆动、上臂伸缩弯曲、肘部扭转、手腕俯仰、手腕偏航等，实现旋转或直线运动，使末端执行器能在三维空间中灵活调整位置与姿态。其关节由电机驱动系统控制，通过改变各关节位置和角度达成所需动作。该机械臂的舵机采用的是飞特的总线舵机。舵机如图 1.12 所示：

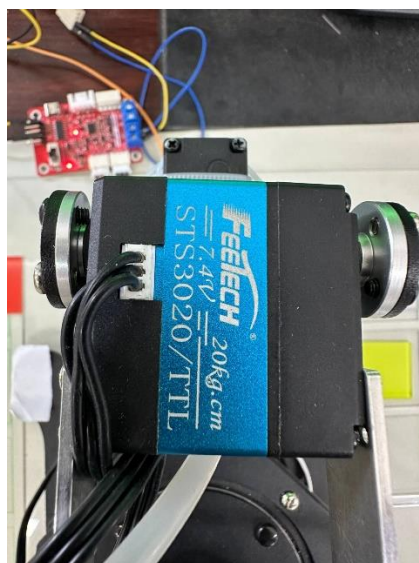


图 1.12 舵机实物图

下图是课题使用舵机驱动电路，该总线舵机可支持串口通信和 PWM 直驱。驱动电路如图 1.13 所示：

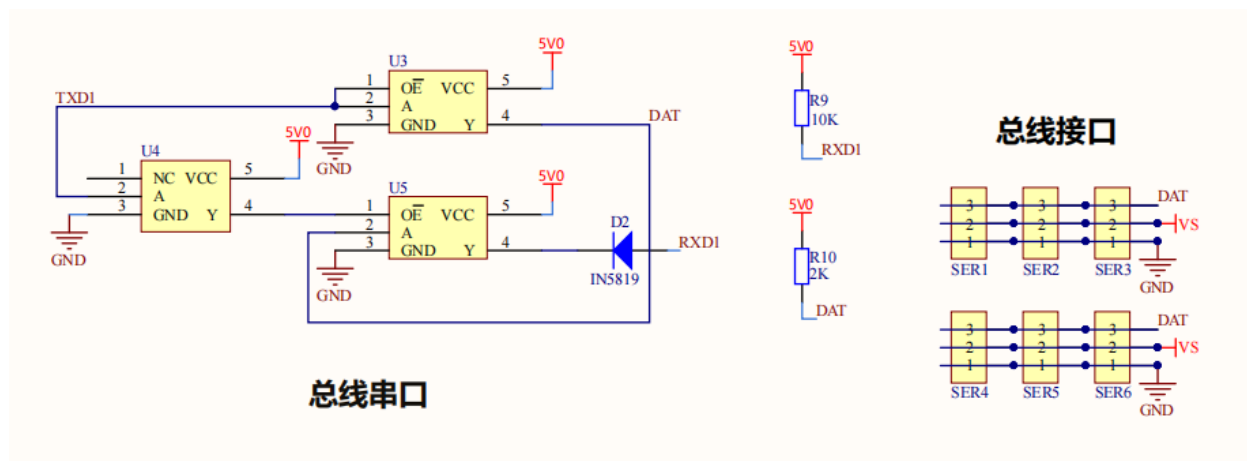


图 1.13 舵机驱动电路图

总线舵机是一种集成了电机、减速齿轮组、控制电路和通信接口的智能执行器。它的主要功能是接收来自控制器的指令，并通过精确控制电机的转动来实现特定角度的输出。

它突破了传统舵机的局限，具备诸多优势：布线极简，支持串联，通过少数几条线即可控制多个舵机，避免大量线缆缠绕；能实时反馈位置、温度、速度、电压、电流等数据，便于主控监控与调整，预防堵转等故障；采用数字传输信号，抗干扰强，通信灵活可靠且支持双向通信；每个舵机有唯一 ID，可通过指令精准控制单个或多个舵机，还能实现开机回中位等功能。其应用广泛，涵盖机器人、机械臂、智能车等领域，能简化系统设计，提升控制精度与灵活性，即使非专业人员也能快速上手，缩短开发周期，为相关项目开发（如仿人



机器人、机械臂）提供高效便捷的解决方案，减少主控板端口占用，方便调试与维护。

## 2 关键技术分析

### 2.1 视觉处理部分

#### 2.1.1 色块识别

本赛题主要识别以下目标色块，共有两种图像识别算法。目标色块如图 2.1 所示：

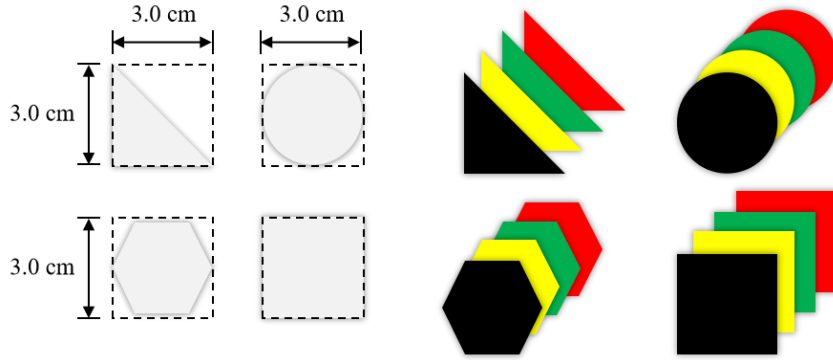


图 2.1 目标颜色色块图

(1) 图像处理算法总共有两种方案：

方案一：将 RGB 数据转化为 HSV 颜色空间，再分别对不同阈值进行二值化，得到不同颜色的二值图像。公式如下：

$$H = \begin{cases} 0^\circ, & \text{if max} = \text{min} \\ 60^\circ \times \frac{g-b}{\text{max}-\text{min}} + 0^\circ, & \text{if max} = \text{randg} \geq b \\ 60^\circ \times \frac{g-b}{\text{max}-\text{min}} + 360^\circ, & \text{if max} = \text{randg} < b \\ 60^\circ \times \frac{b-r}{\text{max}-\text{min}} + 120^\circ, & \text{if max} = g \\ 60^\circ \times \frac{r-g}{\text{max}-\text{min}} + 240^\circ, & \text{if max} = b \end{cases} \quad \text{式 (2-1)}$$

$$S = \begin{cases} 0, & \text{if max} = 0 \\ \frac{\text{max}-\text{min}}{\text{max}} = 1 - \frac{\text{min}}{\text{max}}, & \text{otherwise} \end{cases} \quad \text{式 (2-2)}$$

$$V = \text{max} \quad \text{式 (2-3)}$$

方案二：将 RGB 数据转化为 YCbCr 颜色空间，再分别对不同阈值进行二值化，得到不同颜色的二值图像。公式如下：

$$\begin{cases} Y = 0.299R + 0.587G + 0.114B \\ Cb = -0.1687R - 0.3313G + 0.5B + 128 \\ Cr = 0.5R - 0.4187G - 0.0813B + 128 \end{cases} \quad \text{式 (2-4)}$$

根据上述两个方案，YCrCb 转换逻辑简单，调试与验证容易，开发周期短。HSV 转换因复杂运算与条件判断，硬件描述语言实现时易出错，资源占用率大，增加开发成本与风险。在 FPGA 资源有限的情况下，YCrCb 更易实现稳

定、可靠的设计，确保系统整体性能。所以在本次赛题中 YCbCr 计算复杂度、实时性、兼容性及硬件实现难度上更适配，能高效满足机械臂精确控制系统对图像快速处理与实时传输的需求，是更优选择。

(2) 目标检测总共有两种方案：

方案一：多目标检测方法，将图像中的多个目标同时检测，一共需要检测 15 个目标，对 15 个目标的特征信息如颜色、形状、坐标等实时检测。示意图如下图 所示：

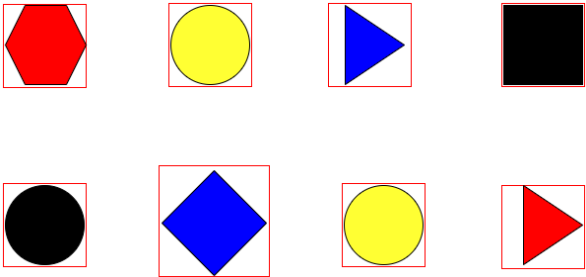


图 2.2 目标示意图

方案二：单目标检测方法，只检测单一目标，当机械臂将该目标搬走后，再识别下一目标，得出相关特征信息。示意图如下图 2. 所示：

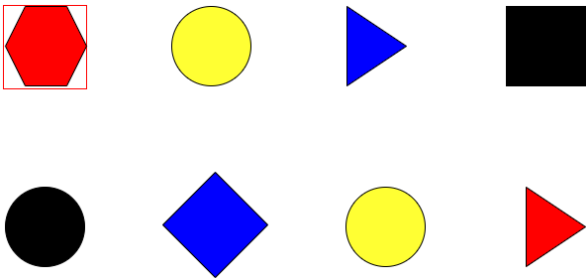


图 2.3 相关特征示意图

根据上述两种方案对比仿真得出，在 FPGA 资源有限，方案二单目标检测仅需一套检测电路，大幅减少逻辑资源占用，存储只需缓存当前目标数据，需求低，且处理任务简单使功耗降低，代码量少、结构清晰，开发调试更便捷，资源利用高效，更适合该题目。

### 2.1.2 Sobel 边缘检测

边缘检测模块基于 Verilog 实现，用于对转换来的 Y（亮度）分量进行边缘检测。通过 Sobel 算子计算图像像素梯度，识别出图像中的边缘信息。得到物块的边框

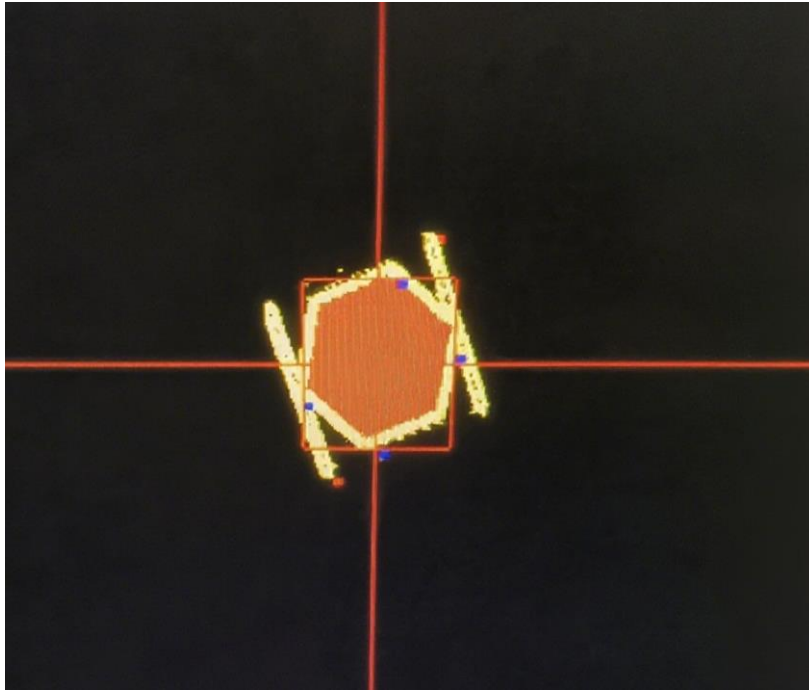


图 2.4 边缘检测边框

算法实现采用经典 Sobel 算子模板，分别定义 x（水平）、y（垂直）方向模板系数：

```

/***** 注释 *****/

Sobel算子模板系数：

y          x
-1  0  1    1  2  1
-2  0  2    0  0  0
-1  0  1   -1 -2 -1

g = |x_g| + |y_g|

*****/

```

图 2.5 模板系数

使用 quartusIP 核例化内存为 8 位 RAM 模块做窗口像素核 4 级流水计算 Y 分量梯度，最后通过比较梯度大小与设定的阈值进行比较，获得边缘信息。

```

// 缓存3列 - 第一级流水
always @(posedge clk or negedge rst_n) begin
    if(~rst_n) begin
        line0_0 <= {DATA_WIDTH{1'b0}};
        line0_1 <= {DATA_WIDTH{1'b0}};
        line0_2 <= {DATA_WIDTH{1'b0}};
        line1_0 <= {DATA_WIDTH{1'b0}};
        line1_1 <= {DATA_WIDTH{1'b0}};
        line1_2 <= {DATA_WIDTH{1'b0}};
        line2_0 <= {DATA_WIDTH{1'b0}};
        line2_1 <= {DATA_WIDTH{1'b0}};
        line2_2 <= {DATA_WIDTH{1'b0}};
    end
    else if(vld[0]) begin
        line0_0 <= taps0; line0_1 <= line0_0; line0_2 <= line0_1;
        line1_0 <= taps1; line1_1 <= line1_0; line1_2 <= line1_1;
        line2_0 <= taps2; line2_1 <= line2_0; line2_2 <= line2_1;
    end
end

// 第二级流水 - 计算完整卷积和
always @(posedge clk or negedge rst_n) begin
    if(~rst_n) begin
        x0_sum <= {SUM_WIDTH{1'b0}};
        x2_sum <= {SUM_WIDTH{1'b0}};
        y0_sum <= {SUM_WIDTH{1'b0}};
        y2_sum <= {SUM_WIDTH{1'b0}};
        x_conv <= {SUM_WIDTH{1'b0}};
        y_conv <= {SUM_WIDTH{1'b0}};
    end
    else if(vld[1]) begin
        // x方向卷积计算: [1 2 1]
        x0_sum <= line0_0 + (line0_1 << 1) + line0_2;
        x2_sum <= line2_0 + (line2_1 << 1) + line2_2;
        // y方向卷积计算: [1 2 1]^T
        y0_sum <= line0_0 + (line1_0 << 1) + line2_0;
        y2_sum <= line0_2 + (line1_2 << 1) + line2_2;

        // 完整Sobel卷积计算
        x_conv <= x0_sum - x2_sum;
        y_conv <= y0_sum - y2_sum;
    end
end

// 第3级流水 - 计算x、y方向梯度绝对值
always @(posedge clk or negedge rst_n) begin
    if(~rst_n) begin
        x_abs <= {ABS_WIDTH{1'b0}};
        y_abs <= {ABS_WIDTH{1'b0}};
    end
    else if(vld[2]) begin
        x_abs <= (x_conv[SUM_WIDTH-1] == 1'b1) ? (~x_conv + 1'b1) : x_conv;
        y_abs <= (y_conv[SUM_WIDTH-1] == 1'b1) ? (~y_conv + 1'b1) : y_conv;
    end
end

// 第4级流水 - 计算梯度
always @(posedge clk or negedge rst_n) begin
    if(~rst_n) begin
        g <= {GRAD_WIDTH{1'b0}};
    end
    else if(vld[3]) begin
        g <= x_abs + y_abs; // 绝对值之和近似平方和开根号
    end
end

// 打拍控制信号
always @(posedge clk or negedge rst_n) begin
    if(~rst_n) begin
        sop <= 4'd0;
        eop <= 4'd0;
        vld <= 4'd0;
    end
    else begin
        sop <= {sop[2:0], din_sop};
        eop <= {eop[2:0], din_eop};
        vld <= {vld[2:0], din_vld};
    end
end

// 阈值比较 - 使用优化后的阈值
assign dout = g >= THRESHOLD;
assign dout_sop = sop[3];
assign dout_eop = eop[3];
assign dout_vld = vld[3];
endmodule

```

图 2.6 边缘信息获取代码

## 2.1.3 旋转角度识别

### 2.1.3.1 颜色角点

使用摄像头从左到右边，从上到下的 Z 字型扫描逻辑，当二值化获得物块的颜色信息有效时，记录当前的 xy 坐标分量，逐步逼近的思想获得角点信息。

这里用获取上角点的坐标举例，当二值化信息有效，即当前点的颜色信息为目标颜色时，第一次与设定的初值 VGA 屏幕上最下面的点做对比，若大于这个值，则用 target\_pos[21:11]s 寄存器去寄存，当第二次扫描到的点二值化信息有效时，若此时 y 坐标小于 target\_pos[21:11]的值，即更新

target\_pos[21:11]，这样逐次逼近得目标物块的上角点。以下为具体代码：

```
//----- 目标检测核心逻辑 -----
reg [44:0] target_pos ; // {valid, ymax, xmax, ymin, xmin}
reg [44:0] target_pos_1;

wire [10:0] target_bottom = (target_pos[43:33] < end_y - MIN_DIST) ? (target_pos[43:33] + MIN_DIST) : end_y ; //下边界的像素坐标
wire [10:0] target_right = (target_pos[32:22] < end_x - MIN_DIST) ? (target_pos[32:22] + MIN_DIST) : end_x ; //右边界的像素坐标
wire [10:0] target_top = (target_pos[21:11] > start_y + MIN_DIST) ? (target_pos[21:11] - MIN_DIST) : start_y ; //上边界的像素坐标
wire [10:0] target_left = (target_pos[10:0] > start_x + MIN_DIST) ? (target_pos[10:0] - MIN_DIST) : start_x ; //左边界的像素坐标

always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        target_pos <= 45'd0;
    end
    else if(sop[0]) begin
        target_pos <= 45'd0;
    end
    else if(vld[0] && din && !black_en) begin //在区域范围内以及非已检测区域
        if(!target_pos[44]) begin // 新目标检测
            target_pos <= {1'b1, y_cnt, x_cnt, y_cnt, x_cnt};
            target_pos_1 <= {1'b1, y_cnt, x_cnt, y_cnt, x_cnt};
        end
        else begin // 扩展已有目标
            if((x_cnt >= target_left) && (x_cnt <= target_right) && (y_cnt >= target_top) && (y_cnt <= target_bottom)) begin
                target_pos[44] <= 1'b1;
                if(x_cnt <= target_pos[10:0] && (target_pos[32:22] - target_pos[10:0] <= MAX_BOX)) begin //若X坐标小于左边界，则将其X坐标扩展为左边界
                    target_pos[10:0] <= x_cnt ; //防止粘连 左角点(x,y){target_pos[10:0],target_pos_1[10:0]}
                    target_pos_1[10:0] <= y_cnt;
                end
                if(x_cnt >= target_pos[32:22] && (target_pos[32:22] - target_pos[10:0] <= MAX_BOX)) begin //若X坐标大于右边界，则将其X坐标扩展为右边界
                    target_pos[32:22] <= x_cnt ; //右角点
                    target_pos_1[32:22] <= y_cnt;
                end
                if(y_cnt <= target_pos[21:11] && (target_pos[43:33] - target_pos[21:11] <= MAX_BOX)) begin //若Y坐标小于上边界，则将其Y坐标扩展为上边界
                    target_pos[21:11] <= y_cnt ; //上角点
                    target_pos_1[21:11] <= x_cnt;
                end
                if(y_cnt >= target_pos[43:33] && (target_pos[43:33] - target_pos[21:11] <= MAX_BOX)) begin //若Y坐标大于下边界，则将其Y坐标扩展为下边界
                    target_pos[43:33] <= y_cnt ; //下角点
                    target_pos_1[43:33] <= x_cnt ;
                end
            end
        end
    end
end
```

图 2.7 角点信息获取代码

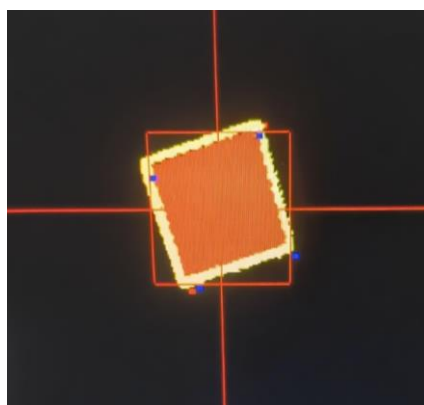


图 2.8 角点示意图

### 2.1.3.2 边缘角点

当检测到六边形和圆形时，由于两种图形具有对称性，因此无法仅从颜色角点获得正确旋转角度。此时需要通过边缘识别获得边框的角点。

通过获取颜色角点的逻辑，在获得 4 个颜色角点坐标的基础上，可以获得物块的中心坐标，该中心坐标同样也为正方形方片的中心坐标，可以中心坐标为圆心设置半径画圆去框定边缘检测的范围，在这个范围内去做边缘检测角点判断，角点判定逻辑使用识别颜色角点逐次逼近的逻辑，只需要把有效点换成边缘检测的有效点就可以了，代码如下：

```
wire sobel_dot = (((x_cnt_r > X_center[10:0]-11'd55 && x_cnt_r < X_center[10:0]+11'd55) && (y_cnt_r > Y_center[10:0]-11'd55 && y_cnt_r < Y_center[10:0]+11'd55)) && sobel);

reg [21:0] sobel_top_spot;
reg [21:0] sobel_bottom_spot;

always@(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        sobel_top_spot <= {11'd0, 11'd0};
        sobel_bottom_spot <= {11'd0, 11'd1200};
    end

    else if(eop[1]) begin
        // 帧结束时重置边缘点，准备下一帧检测
        sobel_top_spot <= {11'd0, 11'd0};
        sobel_bottom_spot <= {11'd0, 11'd1200};
    end

    else if(sobel_dot) begin
        sobel_top_spot <= (y_cnt_r >= sobel_top_spot[10:0]) ? {x_cnt_r, y_cnt_r} : sobel_top_spot;
        sobel_bottom_spot <= (y_cnt_r <= sobel_bottom_spot[10:0]) ? {x_cnt_r, y_cnt_r} : sobel_bottom_spot;
    end
end
```

图 2.9 边缘检测代码

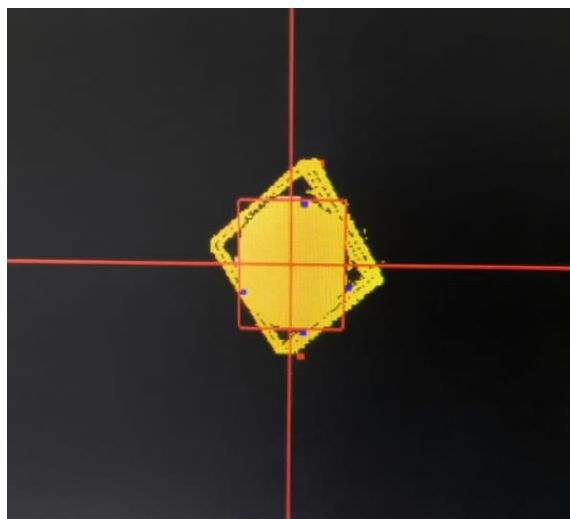


图 2.10 边缘检测的角点

### 2.1.3.3 正方形

提取目标物的 Top\_JiaoDian（上角点）和 bottom\_JiaoDian（下角点）坐标，通过这两个角点确定水平与垂直方向的长度差。

将角点的水平长度差  $H\_Length$  ( $Top\_JiaoDian[21:11] - bottom\_JiaoDian[21:11]$ ) 和垂直长度差  $V\_Length$  ( $Top\_JiaoDian[10:0] - bottom\_JiaoDian[10:0]$ )，右移 1 位后作为 CORDIC 模块的输入  $x$ 、 $y$ ，计算原始角度。

```

else if (shape_out==1)begin //正方形
    /*
    正方形逻辑为找到色域的上下角点，求得夹角，最大为135°
    实际角度为求得角度-45
    */
    H_Length<=Top_JiaoDian[21:11] - bottom_JiaoDian[21:11];
    V_Length<=Top_JiaoDian[10:0] - bottom_JiaoDian[10:0];
    x<=H_Length[10:1];
    y<=V_Length[10:1];
end

// 实例化CORDIC反正切模块
atan u_atan(
    .sys_clk      (clk      ),
    .sys_rst_n    (rst_n    ),
    .en           (eop      ),
    .X            (x        ),
    .Y            (y        ),
    .valid        (theta_valid),
    .deg          (theta    )
);

reg    age_vilid;
always@(posedge clk or negedge rst_n)begin
    if(!rst_n)begin
        age<=0;
        age_vilid<=0;

    end
    else if(shape_out==1&&bottom_JiaoDian[10:0]<=left_JiaoDian[10:0]+'d10)begin
        age <=0;
        Box_theta_out<=0;
    end
    else if(shape_out==1&&theta_valid&&bottom_JiaoDian[10:0]>=left_JiaoDian[10:0]+'d10)begin
        age <=theta[8]?8'd255 - theta[7:0] + 1-'d45:theta[7:0]-'d45;
        Box_theta_out<=theta[8]?8'd255 - theta[7:0] + 1-'d45:theta[7:0]-'d45;
        age_vilid<=1;
    end
end

```

图 2.11 边缘检测代码

#### 2.1.3.4 三角形

先通过目标物的水平长度  $x\_length$  和垂直长度  $y\_length$ ，结合中心坐标  $X\_center$ 、 $Y\_center$  与角点 ( $Top\_JiaoDian$ 、 $bottom\_JiaoDian$ 、 $left\_JiaoDian$ 、 $Right\_JiaoDian$ ) 的位置关系，判定 直角点位置 ( $zj\_dot\_pos$ ，分上、下、左、右 4 种情况)。

1. 上直角点 ( $zj\_dot\_pos==0$ )：角度加  $90^\circ$

```

else if(shape_out==8&&theta_valid&&zj_dot_pos==0)begin//上角点
Box_theta_out<=theta[8]?8'd255 - theta[7:0]+'d90 + 1:theta[7:0]+'d90;
age_vilid<=1;

end

```

2. 下直角点 (zj\_dot\_pos==1)：直接使用 CORDIC 原始输出

```

else if(shape_out==8&&theta_valid&&zj_dot_pos==1)begin//下角点
Box_theta_out<=theta[8]?8'd255 - theta[7:0] + 1:theta[7:0] ;
age_vilid<=1;

end

```

3. 左直角点 (zj\_dot\_pos==2)：角度减 45° 后加 270°

```

else if(shape_out==8&&theta_valid&&zj_dot_pos==2)begin
Box_theta_out<=theta[8]?8'd255 - theta[7:0] - 'd45+'d270 + 1:theta[7:0] - 'd45++ 'd270 ;
age_vilid<=1;

end

```

4. 右直角点 (zj\_dot\_pos==3)：角度加 90°

```

else if(shape_out==8&&theta_valid&&zj_dot_pos==3)begin
Box_theta_out<=theta[8]?8'd255 - theta[7:0] + 1+'d90:theta[7:0]+'d90 ;
age_vilid<=1;

end

```

### 2.1.3.5 六边形

提取边缘特征点 sobel\_top\_spot (上边缘点) 和 sobel\_bottom\_spot (下边缘点)，通过这两个点的坐标差确定水平与垂直长度。

将水平长度差 H\_Length (sobel\_top\_spot[21:11] - sobel\_bottom\_spot[21:11]) 和垂直长度差 V\_Length (sobel\_top\_spot[10:0] - sobel\_bottom\_spot[10:0])，右移 1 位后作为 CORDIC 模块输入。



```

else if(shape_out==2)begin//六边形
/*
六边形逻辑为找到边缘域的上下角点，求得夹角，最大为135°
实际角度为求得角度-45
*/
H_Length<=sobel_top_spot[21:11] - sobel_bottom_spot[21:11];
V_Length<=sobel_top_spot[10:0] - sobel_bottom_spot[10:0];
x<=H_Length[10:1];
y<=V_Length[10:1];

```

利用六边形几何对称性，最终角度需在 CORDIC 原始输出基础上 减去 45°，并将结果与 90° 做修正（Box\_theta\_out<='d90-age'）

```

else if(shape_out==2&&theta_valid)begin
age <=theta[8]?8'd255 - theta[7:0] + 1-'d45:theta[7:0]-'d45;
Box_theta_out<='d90-age;
age_vilid<=1;

```

#### 2.1.3.6 圆形

提取边缘特征点 sobel\_top\_spot（上边缘点）和 sobel\_bottom\_spot（下边缘点），通过这两个点的坐标差确定水平与垂直长度

将水平长度差 H\_Length（sobel\_top\_spot[21:11] - sobel\_bottom\_spot[21:11]）和垂直长度差 V\_Length（sobel\_top\_spot[10:0] - sobel\_bottom\_spot[10:0]），右移 1 位后作为 CORDIC 模块输入。

```

else if(shape_out==4)begin//圆形
/*
圆形逻辑为找到边缘域的上下角点，求得夹角，最大为135°
实际角度为求得角度-45
*/
H_Length<=sobel_top_spot[21:11] - sobel_bottom_spot[21:11];
V_Length<=sobel_top_spot[10:0] - sobel_bottom_spot[10:0];
x<=H_Length[10:1];
y<=V_Length[10:1];
end

```

最终角度需在 CORDIC 原始输出基础上 减去 45°，并将结果与 90° 做修正

```

else if(shape_out==4&&theta_valid)begin
age <=theta[8]?8'd255 - theta[7:0] + 1-'d45:theta[7:0]-'d45;
Box_theta_out<='d90-age;
age_vilid<=1;

end

```

## 2.2 机械臂逆运动学计算部分

在本次项目中，由于 L3 是永远垂直水平面的，所以解算时可以相对取巧计算，在逆运动学求解算法上有两种方案求取：

方案一如图 2.11 所示：

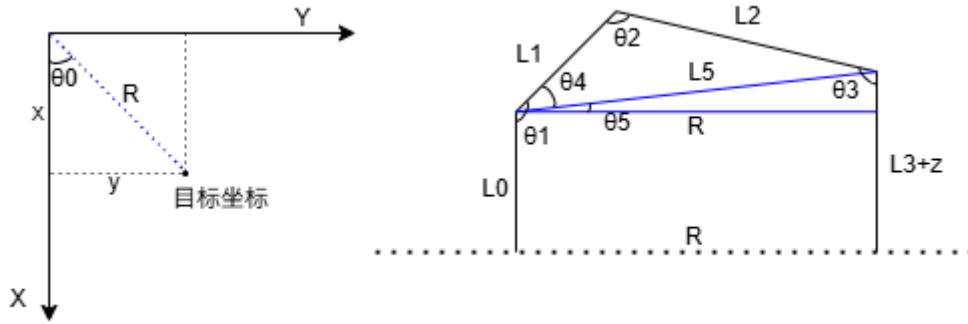


图 2.11 方案一示意图

该方案解算图如图所示，这种解算方法利用余弦定理，可以将各机械臂之间的夹角计算出来，具体解算方法如下：

在得到目标坐标  $(x, y)$  后，通过式 2-5、式 2-6 可以得到

$$R = \sqrt{x^2 + y^2} \quad \text{式 (2-6)}$$

$$L5 = \sqrt{(L3 + Z - L0)^2 + R^2} \quad \text{式 (2-7)}$$

根据余弦定理公式 2-7 可得出  $\theta_2$ 。

$$\cos \theta_2 = \frac{L_1^2 + L_2^2 - L_5^2}{2 \cdot L_1 \cdot L_2} \quad \text{式 (2-8)}$$

由公式 2-8 可得出  $\theta_4$ 。

$$\cos \theta_4 = \frac{L_1^2 + L_5^2 - L_2^2}{2 \cdot L_1 \cdot L_5} \quad \text{式 (2-9)}$$

由公式 2-9 可得出  $\theta_5$ 。

$$\cos \theta_5 = \frac{R}{L_5} \quad \text{式 (2-10)}$$

综式 2-10、2-11 就可以计算出  $\theta_1$ 、 $\theta_2$ 、 $\theta_3$  的值，最后换算为各机械臂舵机 PWM 值。

$$\theta_1 = \theta_4 + \theta_5 + 90^\circ \quad \text{式 (2-11)}$$

$$\theta_3 = 360^\circ - \theta_1 - \theta_2 \quad \text{式 (2-12)}$$

方案二如图 2.12 所示：

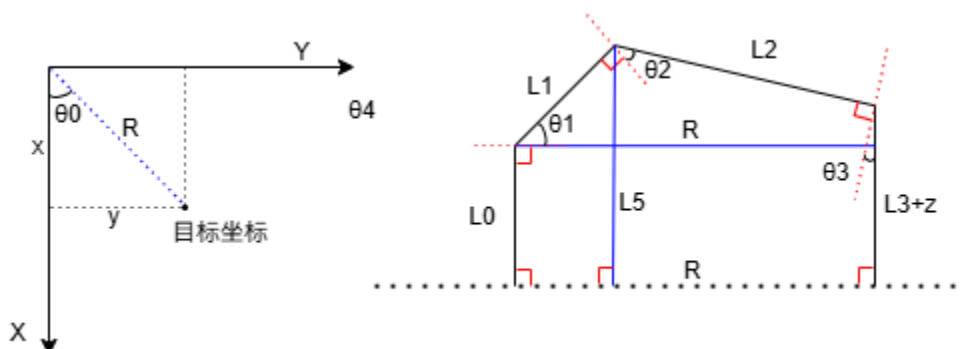


图 2.12 方案二示意图

该方案解算图如图所示，这种解算方法直接转化为简单数学表示，可以将各机械臂之间的夹角计算出来，具体解算方法如下：

由于五边形内角和  $540^\circ$ ，所以可得式 2-12。

$$\theta_1 + \theta_2 + \theta_3 = 90^\circ \quad \text{式 (2-13)}$$

由简单数学变换得出式 2-13、2-14。

$$L_1 \cos \theta_1 + L_2 \cos \theta_3 = R \quad \text{式 (2-14)}$$

$$L_0 + \sin \theta_1 L_1 = L_3 + z + \sin \theta_3 L_2 \quad \text{式 (2-15)}$$

首先，将第三个方程改写为式 2-15。

$$L_1 \sin \theta_1 - L_2 \sin \theta_3 = L_3 + z - L_0 \quad \text{式 (2-16)}$$

令

$$C = L_3 + z - L_0 \quad \text{式 (2-17)}$$

则方程 2 和 3 可表示为式 2-17、2-19。

$$L_1 \cos \theta_1 + L_2 \cos \theta_3 = R \quad \text{式 (2-18)}$$

$$L_1 \sin \theta_1 - L_2 \sin \theta_3 = C \quad \text{式 (2-19)}$$

接下来，将这两个方程平方后相加，利用三角恒等式得到式 2-19。

$$(L_1 \cos \theta_1 + L_2 \cos \theta_3)^2 + (L_1 \sin \theta_1 - L_2 \sin \theta_3)^2 = R^2 + C^2 \quad \text{式 (2-20)}$$

展开并简化得到式 2-20。

$$L_1^2 + L_2^2 + 2L_1L_2 \cos(\theta_1 + \theta_3) = R^2 + C^2 \quad \text{式 (2-21)}$$

从而得到式 2-21。

$$\cos(\theta_1 + \theta_3) = \frac{R^2 + C^2 - L_1^2 - L_2^2}{2L_1L_2} \quad \text{式 (2-22)}$$

令

$$\phi = \theta_1 + \theta_3 \quad \text{式 (2-23)}$$

则

$$\phi = \arccos\left(\frac{R^2 + C^2 - L_1^2 - L_2^2}{2L_1L_2}\right) \quad \text{式 (2-24)}$$

接下来解方程组：

$$\begin{cases} L_1 \cos \theta_1 + L_2 \cos \theta_3 = R \\ L_1 \sin \theta_1 - L_2 \sin \theta_3 = C \end{cases} \quad \text{式 (2-25)}$$

将 $\theta_3$ 表示为 $\phi - \theta_1$ ，代入方程式 2-24 并利用三角恒等式展开，得到关于 $\theta_1$ 的线性方程组式 2-25 和式 2-26。通过克莱姆法则求解，得到：

$$\cos \theta_1 = \frac{(L_1 + L_2 \cos \phi)R - L_2 \sin \phi C}{(L_1 + L_2 \cos \phi)^2 + (L_2 \sin \phi)^2} \quad \text{式 (2-26)}$$

$$\sin \theta_1 = \frac{(L_1 + L_2 \cos \phi)C + L_2 \sin \phi R}{(L_1 + L_2 \cos \phi)^2 + (L_2 \sin \phi)^2} \quad \text{式 (2-27)}$$

求出 $\theta_1$ ，然后根据式 2-27 和 2-28

$$\theta_3 = \phi - \theta_1 \quad \text{式 (2-28)}$$

$$\theta_2 = 90^\circ - \theta_1 - \theta_3 \quad \text{式 (2-29)}$$

解出 $\theta_1$ 、 $\theta_2$ 、 $\theta_3$ ，最后换算为各机械臂舵机 PWM 值。

但由于方案二计算难度较大，不易计算。且计算精度与方案一所差无几，所以本项目选择方案一作为本次解算方案。

## 2.3 机械臂控制部分

选择完逆运动学解算方案后，我们需要考虑如何实现这一方案，由于考虑到该计算存在大量浮点数、反三角函数、乘除法计算。

方案一：直接在 FPGA 中实时计算，会使用到 DIV、Mult、cordic 等大量 IP 核。

方案二：将这一部分逆运动学求解算法在 Matlab 中实现，根据已知 (x, y)，可以得出 R、 $\theta_0$ ，并转化为 PWM，拼接后可以以 X、Y 为地址生成 ROM1（图 2.13）。当 R 唯一确定时，可以得出 $\theta_1$ 、 $\theta_2$ 、 $\theta_3$ ，从而可以得出 PWM1、PWM2、PWM3。以 R 为地址，PWM1、PWM2、PWM3 拼接在一起为数据，得到 ROM2（图 2.14）。

```
% 定义 MIF 文件的参数
width = 21; % 数据宽度 (20 位无符号数)
depth = 4096; % 深度
address_radix = 'UNS'; % 地址基数
data_radix = 'UNS'; % 数据基数
% 地址 12 位 前 6 位 x 后 6 位 y 返回值 21 位 前 9 位投影长 1 后 12 位位置值
% 生成 MIF 文件内容
mif_content = sprintf('WIDTH=%d;\nDEPTH=%d;\nADDRESS_RADIX=%s;\nDATA_RADIX=%s;\nCONTENT BEGIN\n', width, depth, address_radix, data_radix);
for i = 0 : 53
    x = i; % 0 - 53 6 位
    realx = x * 5 - 132.5 - 3; % 中间为初始值和偏差修正
    for j = 0 : 35
        y = j; % 0 - 35 6 位
        realy = y * 5 + 85;
        %-----pwm 获取与 1 计算
        range0 = rad2deg(atan(realx/realy));
        l = round(sqrt(realx^2 + realy^2)-3); % 0-300 9 位
        pwm_out0 = round((range0*4096/360)+1024); % 0-4096 12 位

        store_value = bitshift(uint32(l), 12);
        store_value = bitor(store_value, uint32(pwm_out0));

        address = bitshift(uint32(x), 6);
        address = bitor(address, uint32(y));
        mif_content = [mif_content, sprintf(' %d : %d;\n', address, store_value)];
    end
end
% 结束 MIF 文件内容
mif_content = [mif_content, 'END;'];

% 写入 MIF 文件
fid = fopen('bottom_rom.mif', 'w');
fwrite(fid, mif_content, 'char');
fclose(fid);

disp('MIF 文件生成成功!');
```

图 2.13 matlab 生成底轴 rom 表代码图

```
% 定义 MIF 文件的参数
width = 36; % 数据宽度 (20 位无符号数)
depth = 1024; % 深度
address_radix = 'UNS'; % 地址基数
data_radix = 'UNS'; % 数据基数
%地址 17位 前9位x 后8位y 返回值20位 前8位投影长1 后12位pwm值
% 生成 MIF 文件内容
mif_content = sprintf('WIDTH=%d;\nDEPTH=%d;\nADDRESS_RADIX=%s;\nDATA_RADIX=%s;\nCONTENT BEGIN\n', width, depth, address_radix, data_radix);

% 机械臂参数 (mm)
L0 = 101;
L1 = 105;
L2 = 200;
L3 = 134;

for i = 0:1
    flag = i;%0-低 1-高 1位
    if(flag==0)
        Z = 1 * 10;%0-128 7位 2-低 5-高
    else
        Z = 6 * 10;
    end
    for j = 85 : 288
        L = j;% 85-280 9位
        % 计算中间变量
        L5 = sqrt(L^2 + (L3 + Z - L0)^2);
        % 计算关节角度
        cos_theta3 = (L1^2 + L2^2 - L5^2) / (2*L1*L2);
        theta3 = rad2deg(acos(cos_theta3));
        cos_theta1 = (L1^2 + L5^2 - L2^2) / (2*L1*L5);
        theta1 = rad2deg(acos(cos_theta1));
        theta2 = rad2deg(acos(L / L5));

        %计算角度
        J1 = 90-theta1-theta2;
        J2 = 180-theta3;
        J3 = theta1+theta2+theta3-90;

        % 计算对应pwm值-----
        pwm_out1 = round((J1/360)*4096 + 2048);%500 - 2500 12位
        pwm_out2 = round((J2/360)*4096 + 2048);
        pwm_out3 = round((J3/360)*4096 + 2048) - 95;

        % 拼接打包
        store_value = bitshift(uint64(pwm_out1), 24);
        store_value = bitor(store_value, bitshift(uint64(pwm_out2), 12));
        store_value = bitor(store_value, uint64(pwm_out3));
        store_value_bin = dec2bin(store_value, 36);

        address = bitshift(uint32(flag), 9);
        address = bitor(address, uint32(L));
        address_bin = dec2bin(address, 10);
        mif_content = [mif_content, sprintf(' %d : %d;\n', address, store_value)];
    end
end
% 结束 MIF 文件内容
mif_content = [mif_content, 'END;'];

% 写入 MIF 文件
fid = fopen('arm_rom.mif', 'w');
fwrite(fid, mif_content, 'char');
fclose(fid);

disp('MIF 文件生成成功!');
```

图 2.14 matlab 生成机械臂 rom 表代码图

机械臂需要运动面积如图 2.15 所示:

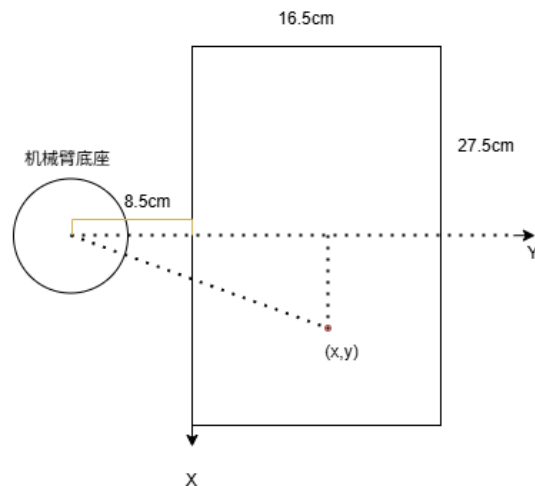


图 2.15 机械臂运动面积示意图

当目标坐标  $(x, y)$  唯一确定时, 机械臂底座夹角  $\theta_0$  就唯一确定; 目标坐标到机械臂底座距离  $R$  由下式 4-8 可得:

$$R = \sqrt{x^2 + y^2} \quad \text{式 (2-30)}$$

所以可以用 Matlab 根据 x、y，计算出所有角度、PWM0、和 R 的值，以 {X, Y} 为地址，PWM0 和 R 为数据生成 .mif 文件

综上两种方案，方案一资源占用过高，这些 IP 核尤其是乘法器 (Mult) 和除法器 (DIV) 会消耗大量 FPGA 的 DSP 单元、逻辑单元 (LE) 等资源，可能导致芯片资源紧张，限制其他功能模块的实现；其次，时序设计困难，复杂 IP 核如 CORDIC 运算延迟较大，多级 IP 核级联易引发时序违规，难以满足高速时钟要求，影响系统实时性；而方案二在 Matlab 中实现算法，生成 ROM 供 FPGA 查表，减少 FPGA 资源占用，时序设计简单，在 Matlab 中开发调试高效，算法调整时仅需重新生成 ROM，维护成本低，且功耗更低，数据精度与可靠性更高。所以在本项目中选择方案二作为机械臂控制方案。

## 2.4 动作组

整个动作组在收到视觉处理模块传回的坐标，角度，颜色，形状和使能信号后开始执行，具体动作如下。

### 2.4.1 抓取

机械臂的控制为输入坐标 (x, y) 抓取和放置标志位以及使能信号，末端即可到达坐标点。具体实现通过状态机依次进行位拼接读取 rom 中对应地址的数据，以此返回机械臂各舵机的偏转角度，如图 2.16 所示。

```
reg [11:0] address_atan = 12'd0; // 12位 6位x 6位y
wire [20:0] atan_q; // 21位 9位 12位pwm0
reg [4:0] first_step = 4'd0;
reg signed [31:0] temp2 = 32'd0;
reg signed [31:0] temp3 = 32'd0;
reg [5:0] x_data_in, y_data_in;
always @(posedge sys_clk or negedge sys_rst_n) begin
    if (!sys_rst_n) begin
        first_step <= 4'd0;
        ready_flag <= 1'b0;
    end else begin
        case (first_step)
            5'd0:begin
                ready_flag <= 1'b0;
                if (start_flag) begin // address_atan <= {x/5, y/5};
                    denom_in <= 'd5;
                    numer_in <= x_data;
                    first_step <= first_step + 1;
                end else begin
                    address_atan <= 12'd0;
                    first_step <= 5'd0;
                end
            end
            5'd1:begin
                x_data_in <= quotient_out;
                first_step <= first_step + 1;
            end
            5'd2:begin
                denom_in <= 'd5;
                numer_in <= y_data;
                first_step <= first_step + 1;
            end
            5'd3:begin
                y_data_in <= quotient_out;
                first_step <= first_step + 1;
            end
            5'd4:begin
                address_atan <= {x_data_in, y_data_in};
                first_step <= first_step + 1;
            end
            5'd5:begin
                first_step <= first_step + 1;
            end
            5'd6:begin
                first_step <= first_step + 1;
            end
        endcase
    end
end
```

图 2.16 rom 表读取代码图

### 2.4.2 仓库部分

在抓取后，状态机运行至仓库读取位置，根据对应的物块的形状和颜色，将物块放入对应的仓库中。仓库中的数据通过串口屏进行输入，保证放置与赛题要求一致。

```
reg [11:0] ware_house1 [0:6]; // 0-angle0 1-angle1 2-angle2 3-angle3 4-color 5-shape 6- 1为已有物品 0为空闲
initial begin // 红 三角 一号
    ware_house1[0] = 12'd3385;
    ware_house1[1] = 12'd2460;
    ware_house1[2] = 12'd2700;
    ware_house1[3] = 12'd2905;
    ware_house1[4] = 12'd1;
    ware_house1[5] = 12'd1;
    ware_house1[6] = 12'd0;
end
```

图 2.17 部分仓库数据一览表

在完成所有动作后，机械臂回到初始位置，等待下一个使能信号。

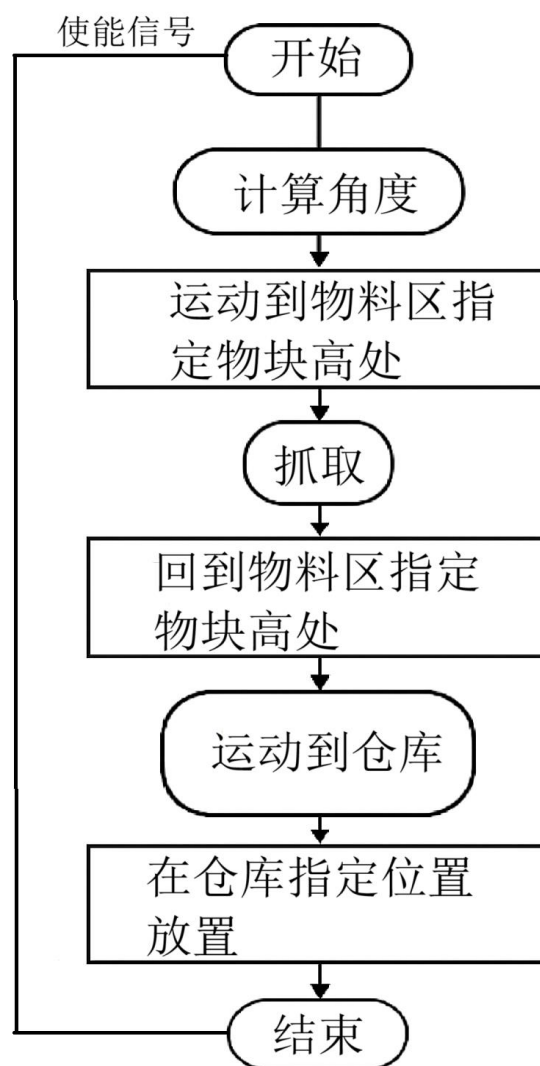


图 2.18 动作组一览表

## 3 性能分析

### 3.1 图像处理算法仿真验证

#### 3.1.1 RGB 转 YCbCr 算法仿真验证

使用仿真工具来进行测试，并观察仿真输出。确保仿真过程中没有出现错误，并检查 YCbCr 分量的输出是否符合预期。经过 RGB 转 YcbCr 仿真结果如图 3.3 所示：

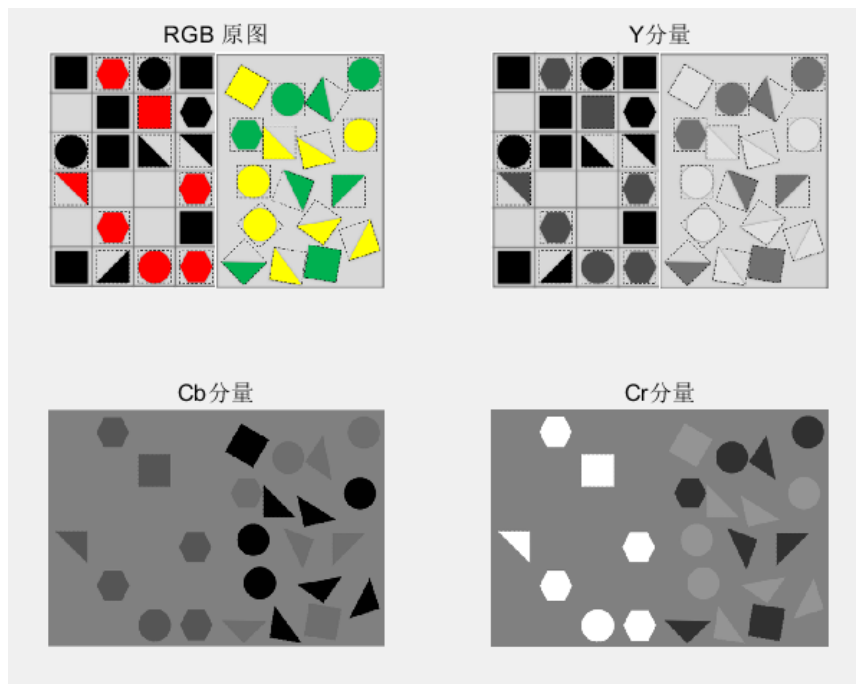


图 3.3 仿真结果图

#### 3.1.2 二值化算法仿真验证

YCBCR 颜色空间中，Y 代表亮度分量，Cb 和 Cr 代表色度分量。将图像从 RGB 转换到 YCBCR 后，亮度和色度分离。在二值化时，对 Cb、Cr 色度分量设置阈值进行操作，可减少光照变化对颜色判断的影响。因为光照变化主要影响亮度（Y 分量），而对 Cb、Cr 影响小。也避免不同颜色之间的粘连问题，便于后续处理，减少目标和复杂度如目标识别、形状分析、特征提取等处理会更简便高效。本项目是根据不同颜色的阈值得出不同颜色的二值化图像再处理的。二值化仿真图如下图 3.4 所示：



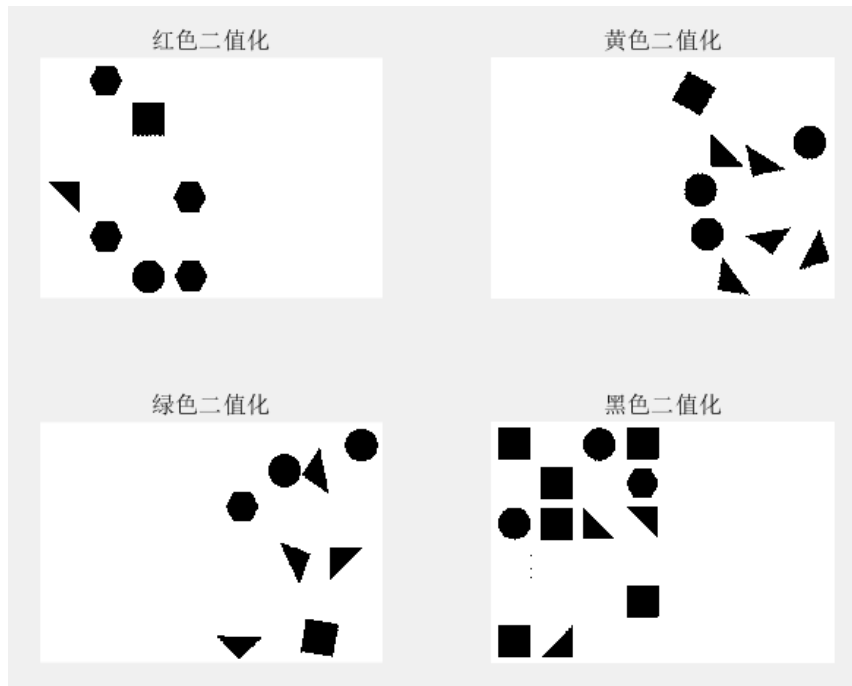


图 3.4 二值化仿真图

### 3.1.3 腐蚀膨胀算法仿真验证

腐蚀和膨胀算法在 FPGA 中的实现原理相差无异，均是采用移位寄存器生成  $3 \times 3$  的模板，最后实现相与或者相或实现腐蚀膨胀算法。本项目采用先腐蚀后膨胀的开操作图像处理方式，可以有效避除噪点且不失原有效图像数据。效果如图 3.5 所示：

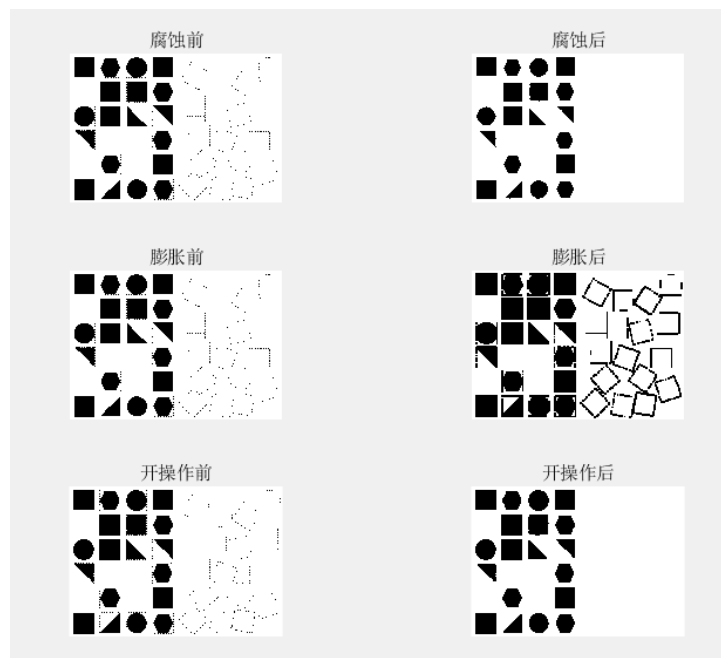


图 3.5 腐蚀膨胀仿真效果图

### 3.1.4 生长算法验证

区域生长算法是一种基于像素相似性的图像分割方法，其核心思想是从一个或多个种子点出发，将与种子点具有相似属性的相邻像素逐步合并到同一区域中，直至满足特定的停止条件。假设黄色三角形为种子点，算法会检测其邻域像素，若邻域像素与种子点的属性差异在设定阈值内，便将该像素纳入生长区域。如此循环，不断扩张区域。效果图如下图 3.6 所示：

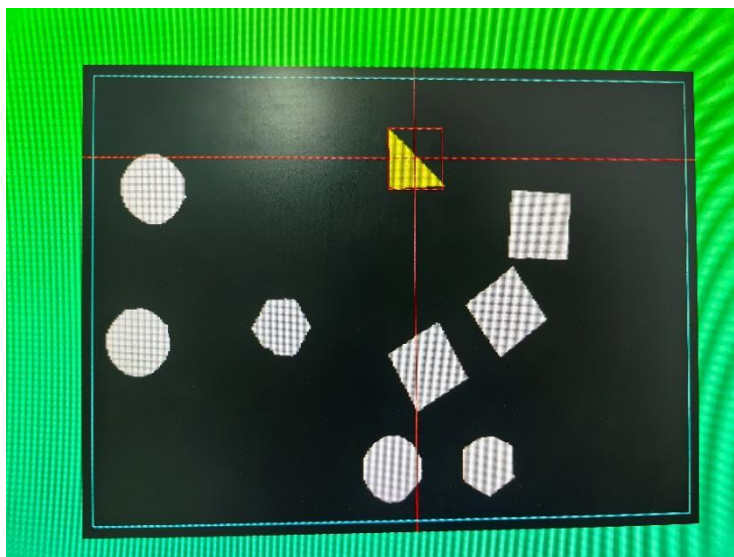


图 3.6 生长算法效果图

### 3.1.5 CORDIC 算法

在 FPGA 中，传统的浮点三角函数计算复杂度较高，不利于硬件高效实现。因此，本系统采用 CORDIC (COordinate Rotation DIgital Computer) 算法，在仅使用加减、移位与查表操作的基础上，实现反正切函数计算。CORDIC 算法基于坐标旋转思想，通过一系列小角度迭代旋转逼近目标角度。对于初始点 $(x_1, y_1)$ ，将其围绕原点旋转角度 $\theta$ 后得到新坐标 $(x_2, y_2)$ ：

```
// 关闭CORDIC防止初值块
atan u_atan(
    .sys_clk      (clk      ),
    .sys_rst_n    (rst_n    ),
    .en           (eop      ),
    .X            (x        ),
    .Y            (y        ),
    .valid        (theta_valid),
    .deg          (theta    )
);
```

## 3.2 机械臂控制算法仿真验证

### 3.2.1 PWM 生成仿真验证

该模块是由特征提取模块得到的  $x$ 、 $y$  值，再根据 Matlab 设计好的 ROM，提取出各机械臂对应的 PWM 值，仿真图形如下图 3.7 所示：

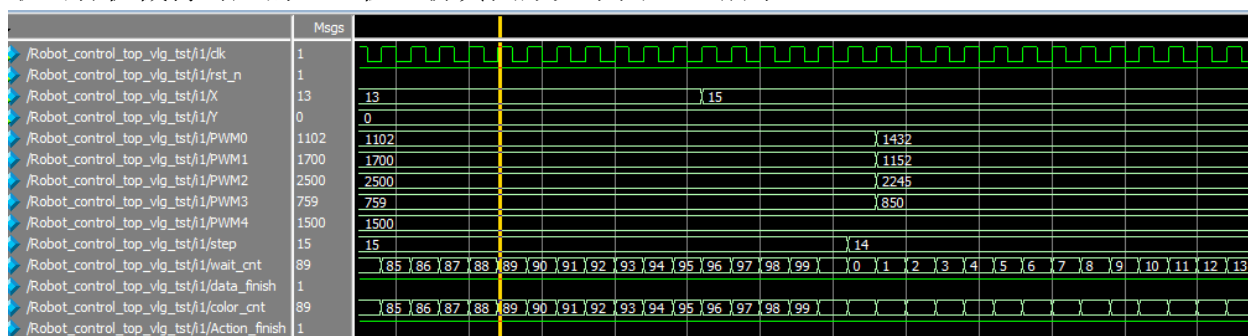


图 3.7 PWM 仿真图

### 3.2.2 数字转字符串仿真验证

数字转 ASCII 模块，是将对应的数字或特定字符转化为 ASCII 形式，在后续合成为控制指令发送给机械臂动作，仿真图如下图 3.8 所示：

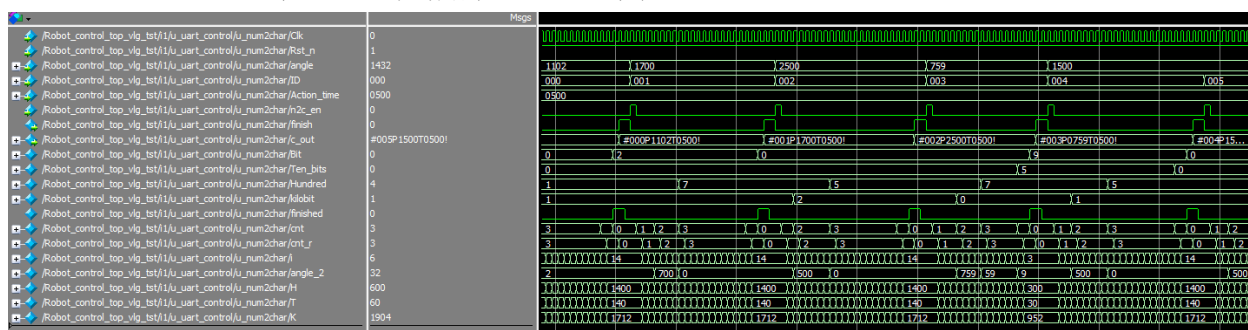


图 3.8 数字转字符串仿真图

### 3.2.3 机械臂指令发送模块验证

在经过数字转 ASCII 后，需要合成为一串特定指令，并通过串口发送到机

机械臂做对应动作，该模块仿真图如下图 3.9 所示：

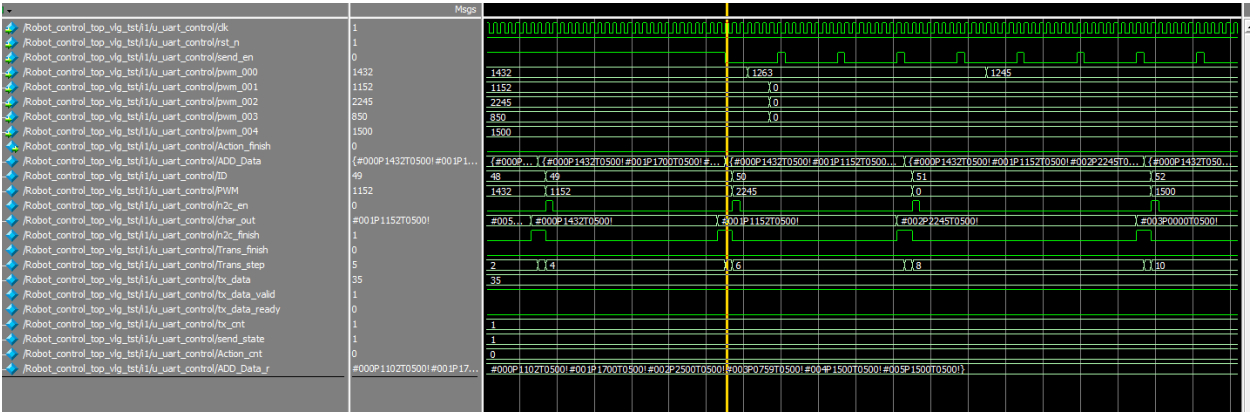


图 3.9 机械臂指令发送模块仿真图

### 3.2.4 上板测试

经上板进行实物测试，系统功能正常，图像识别精度高，机械臂分拣迅速准确，验证了该系统的可行性。硬件搭建如图 3.10 所示：

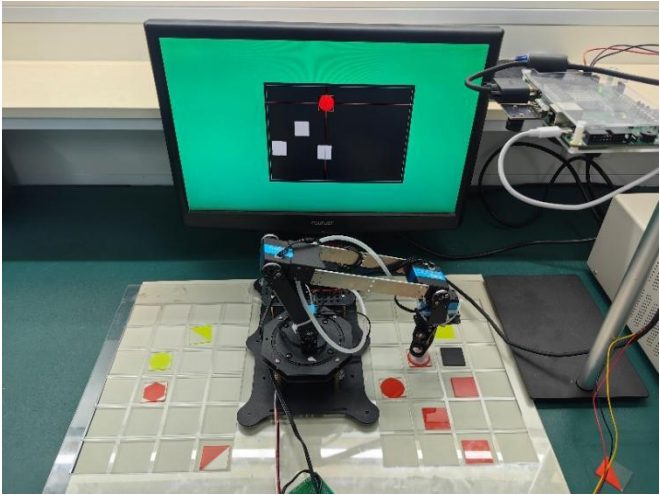


图 3.10 实物演示图

## 3.3 控制精度分析

### 3.3.1 逆运动学解算精度

项目采用两种逆运动学求解方案，分别为：

方案一：基于余弦定理，通过公式逐步计算关节角度。

方案二：通过数学模型与三角恒等式推导角度，利用 Matlab 实现并查表获取。最终选择方案二，核心优势在于数值稳定性高、查表方式减少了浮点误差的累计。由于查表法可预生成高精度 ROM 表项，在 12 位舵机 PWM 精度下，角度误差在  $\pm 0.1^\circ$  以内，转化到末端执行器的定位误差小于 1mm，远优于传统嵌入式计算方式。

### 3.3.2 PWM 精度与舵机控制

舵机 PWM 由 FPGA 内部模块直接驱动，通过定时器与占空比调节实现脉宽输出。PWM 分辨率为 12 位，即 4096 级控制，配合机械臂舵机本身的角度分辨率

（一般为  $0.09^\circ / \text{step}$ ），最终实现了较高的动作精度。

实际测试表明，舵机角度输出误差控制在  $\pm 1$  步进以内，满足高精度搬运、定位任务需求。

### 3.4 系统实时性分析

#### 3.4.1 数据流与图像处理实时性

系统图像采集频率为 84MHz，VGA 输出频率为 75MHz，中间引入乒乓缓存机制，确保图像读写操作并发进行，避免因带宽竞争引起的数据丢失或延迟。

由于图像处理任务如颜色识别、腐蚀膨胀等均在 FPGA 中并行实现，系统可在 1 帧图像周期内完成处理任务，确保在 30fps 视频流中做到每帧均参与计算，满足大多数动态目标识别与控制需求。

#### 3.4.2 控制链路响应速度

从图像采集到控制信号输出，整个控制闭环延迟在 20ms 以内。即：

- 图像采集处理：< 10ms
- 逆运动学解算：< 5ms（查表方式）
- PWM 指令生成与下发：< 5ms

此响应速度足以满足赛题要求的控制任务。

### 3.5 FPGA 资源利用率分析

方案一直接在 FPGA 中进行反三角、乘除法等复杂运算，会调用大量如 CORDIC、Mult、DIV 等 IP 核。这些核单元对 FPGA 资源消耗极大，尤其是 DSP 块与逻辑单元（LEs）：

CORDIC 模块：需多级流水线与 ROM 支持，占用大量寄存器与 LE；

DIV 除法器：需要大量时钟周期和控制逻辑。

而方案二则完全绕开了这些高消耗计算资源，采用 Matlab 离线计算 + ROM 查表方式，实际仅使用少量 BRAM 资源与查找逻辑单元。由查表 ROM 控制机械臂的方法占 FPGA 资源的 16%左右，为视觉部分留足空间。如图 3.11 所示，最后整体逻辑资源占用在 84%左右，记忆资源占用在 75%左右。


Flow Summary	
 <<Filter>>	
Flow Status	Successful - Mon May 19 17:21:43 2025
Quartus Prime Version	23.1std.0 Build 991 11/28/2023 SC Lite Edition
Revision Name	C4MB_test
Top-level Entity Name	top
Family	Cyclone IV E
Device	EP4CE6F17C8
Timing Models	Final
Total logic elements	5,245 / 6,272 ( 84 % )
Total registers	2428
Total pins	87 / 180 ( 48 % )
Total virtual pins	0
Total memory bits	208,628 / 276,480 ( 75 % )
Embedded Multiplier 9-bit elements	7 / 30 ( 23 % )
Total PLLs	2 / 2 ( 100 % )

图 3.11 编译占用一览图

## 4 性能分析与测试

### 4.1 颜色识别准确度测试

本项目围绕基于 YCbCr 色彩空间的颜色识别技术展开，其算法流程为：摄像头采集 RGB 图像信息，FPGA 使用流水线处理将 RGB 色域转化 YCbCr 色域，再通过设定目标阈值实现对目标物体的识别。为验证该算法在实际使用环境下的稳定性，本组在实际使用场景下开展了物体颜色识别，并统计得出不颜色识别的准确率。

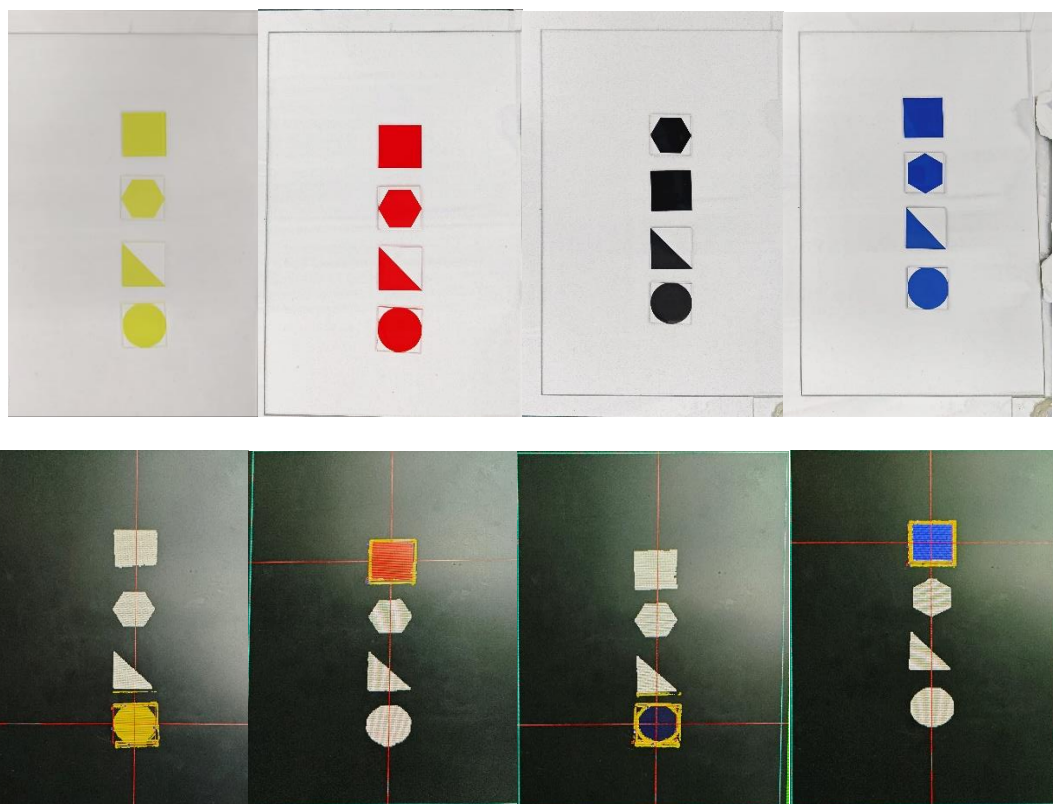


图 4.1 实际环境下的颜色识别实验

表 4.1 实际场景下颜色识别实验表

颜色	识别次数	颜色正确次数	正确率
黄色	100	98	98%
红色	100	100	100%
黑色	100	95	95%
蓝色	100	100	100%



通过上述实验数据可知，在实际场景下，物块颜色识别的综合正确率可达98%，可以较好得达到系统的要求。然而，本实验只是对一般的场景进行的模拟未对强光、弱光等非常规环境进行考虑，针对不同环境应根据实际的光照条件确定颜色阈值以达到一个较好的识别效果。

### 4.2 图形形状识别

物块经过颜色识别之后，经过腐蚀膨胀算法去噪后，经过二值化处理后筛选出需要识别的目标区域，系统再对筛选区域进行面积阈值的判断以区分不同形状的物块。

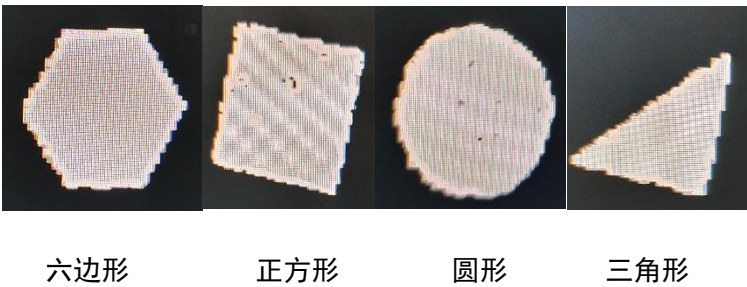


图 4.2 形状检测实验结果

从实验的现象可以看出，经过对形状识别阈值阈值的初步优化对不同形状的识别有了初步的效果，为了进一步测试系统的稳定性和优化形状识别阈值对不同形状进行多次识别进行统计评估。

表 4.2 形状识别实验表

形状	识别次数	形状正确次数	正确率
六边形	100	89	89
正方形	100	98	98%
圆形	100	90	90
三角形	100	93	93

从实验的统计数据可以看出，整体的形状准确度在一个较好的水平，基本可以达到系统的要求，但是对于圆形和六边形这类面积区分较小的图形识别准确度会下降明显，后续考虑引入角点个数分析和细化面积阈值以进一步提高形



状识别的准确度。

4.3 机械臂运动精度

结合实际的应用场景和机械臂机械精度的极限，综合考虑实现难度和效果，我们将地图分割成  $33 \times 55$  的等分，利用 MATLAB 软件计算出每个分割点对应的机械臂姿态角度，存入 FPGA 的 ROM 存储器中，从而实现机械臂对地图物块的抓取。

在实验中，首先通过视觉识别系统确定目标物体的位置坐标，并将这些坐标传输给 FPGA 进行处理。FPGA 查找内部 ROM 读取对应坐标下的机械臂控制信息从而控制机械臂实现抓取。

位置坐标	吸取次数	吸取成功次数	吸取成功率
(0,0)	50	43	86%
(0,55)	50	44	88%
(33,0)	50	45	90%
(33,55)	50	45	90%
(16,0)	50	46	92%
(16,55)	50	44	88%
(0,27)	50	43	86%
(33,27)	50	45	90%
(16,27)	50	48	96%

实现取地图边缘点，中间点等特征点以估计机械臂的抓取准确度，根据表中数据，机械臂抓取整体准确度在 87% 以上，基本满足系统要求，在地图边缘点的抓取上误差较大，中间点的抓取上误差较小，考虑对姿态计算公式做出改进或引入视觉修正以求达到更加准确的抓取效果。

4.4 角度计算精度

物块的角度识别

FPGA 通过颜色角点计算物块与水平方向夹角，获取物块摆放的角度信息。圆形和六边形对称性强，换为边框角度识别。

表 4.3 物块图案角度识别表（单位/°）

正方形			三角形		
实际角度	识别角度	误差	实际角度	识别角度	误差

0	5	5	0	0	0
45	42	3	45	43	2
90	88	2	90	89	1
135	129	6	135	128	7

通过统计的信息可以看出正方与三角形的角度识别相对准确，在系统接收的误差范围以内，也可以看出在角度偏移较大或较小时角度计算误差变大，考虑优化角点识别算法，和角度计算算法以达到更高的角度计算精度。

## 4.5 透明外框角度识别

对圆形和六边形的处理利用边框的角点算出物块与水平方向的夹角，与色块角度测量配合计算出物块的旋转角度。

表 4.4 物块图案角度识别表（单位/°）

正方形		
实际角度	识别角度	误差
0	7	7
45	38	7
90	85	5
135	122	13

通过实验数据看出，边框角度的计算误差比色块的计算偏大，这与边沿检测算法的敏感性和透明边框特征难以提取有关。考虑优化边缘检测算子以提高边框角度识别精度。

## 5 结论

在本项目实施过程中，已成功达成赛题所设定的难度 1 与难度 2 要求。现阶段，物块抓取精准性表现良好，但受限于机械误差因素，在将物块放置于仓库时，其精准度尚存在一定提升空间，经统计，当前物块放置精准度约为 90%。在未来项目的持续开发与优化阶段，研究团队将着重针对机械误差展开精细校准工作，并对现有算法进行深度优化，致力于实现物块放置精准度的进一步提升，以期达到更为卓越的作业精准水平。