综合课程设计-信息组

设计报告

基于 PYNQ 的手势识别

04017419 高佳峻

04017445 刘昊东

04017429 寇周斌

2020.4 - 2020.6

1. 实验目的

1）学习手势识别的基本原理与核心方法

2）掌握手势识别在 vscode 软件平台上的 C/C++语言实现

3）掌握使用 HLS 工具对 C/C++语言算法硬件化的方法

4）掌握使用 Block designer 完成 ARM 核，外设协议以及 HLS 生成的 IP 的互联互通

2. 实验设备

A. PYNQ-Z2 实验平台

PYNQ 全称为 Python Productivity for Zynq，即在原有 Zynq 架构的基础上，添加了对 python 的支持。 Zynq 是赛灵思公司推出的行业第一个可扩展处理平台系列，在芯片中集成了 ARM 处理器和 FPGA 可编程逻辑器件，旨在为视频监视、汽车驾驶员辅助以及工厂自动化等高端嵌入式应用提供所需的处理与计算性能水平。PYNQ 希望能够借助 python 语言本身易用易学、扩展库多而全、社区活跃贡献度高等特性，有效降低 Zynq 嵌入式系统的开发门槛。 PYNQ 将 ARM 处理器与 FPGA 器件的底层交互逻辑完全封装起来，顶层封装使用 python，只需要 import 对应的模块名称即可导入对应的硬件模块即可进行底层到上层数据的交互或者为系统提供硬件加速。

简单来说，它直接对硬件底层进行的封装，用户借助封装库文件

可以直接使用 python 语言操作硬件 I/O 管脚等功能。对于软件工程师来说他们不需要再使用复杂繁琐的开发工具，使用基于浏览器的 Jupyter Notebook 工具就可以直接编辑工程代码，系统架构师借助 PYNQ 可以设计更清晰的软件接口和系统架构，对于硬件工程师而言他们设计的硬件平台能够让更多不同开发背景的人使用。

B. Vivado HLx2017.4 开发软件（包含 Vivado 和 HLS 工具）

Vivado HLS 能提高系统设计的抽象层次，为设计人员带来切实的帮助。Vivado HLS 通过下面两种方法提高抽象层次：

1）使用 C/C++作为编程语言，充分利用该语言中提供的高级结构；

2）提供更多数据原语，便于设计人员使用基础硬件构建块（位向量、队列等）。

与使用 RTL 相比，这两大特性有助于设计人员使用 Vivado HLS 更轻松地解决常见的协议系统设计难题。最终简化系统汇编，简化 FIFO 和存储器访问，实现控制流程的抽象。HLS 的另一大优势是便于架构研究和仿真。Vivado HLS 把 C++函数视为模块，函数定义等效于模块的 RTL 描述，函数调用等效于模块实例化。这种方法能减少需要用户编写的代码量，进而显著简化用于系统描述的结构代码，最终加速系统汇编进程。在 Vivado HLS 中，存储器或 FIFO 可通过两种方法访问。一种是通过合适的对象（比如对流对象的读写）。另一种是直接访问综合工具随后将实现为 Block RAM 或分布式 RAM 的标准 C 阵列。综合工具会根据需要处理额外的信令、同步或寻址问题。从控制流的角度，Vivado HLS 从简单的 FIFO 接口到完整的 AXI4-

Stream 均可提供整套流控制感知接口。使用这些接口，设计人员可直接访问数据，无需检查背压或数据可用性。Vivado HLS 会适当地调度执行，应对一切紧急情况，同时确保正确完成执行。Vivado HLS 具有简便的架构研究功能，用户只需在代码中插入程序指令（如使用 GUI 或批处理模式时的 Tcl 命令），就可以把设计所需特性传递给综合工具。这样用户可以在不修改设计代码本身的情况下研究大量备选架构方案。研究的范围可以是模块流水线化等根本性问题，也可以是 FIFO 队列深度等较常见的问题。最后，C 和 RTL 仿真是 Vivado HLS 另一个大放异彩的地方。设计一般采用两步流程验证：第一步是 C 语言仿真。这个步骤中 C/C++的编译和执行与常见的 C/C++程序相同；第二步是 C/RTL 协仿真。在这步骤中，Vivado HLS 会根据 C/C++测试平台自动生成 RTL 测试平台，然后设置并执行 RTL 仿真，检查实现方案吧的正确性。如能充分发挥这些优势，这将对于用户的系统设计大有裨益。这不仅体现在开发时间和生产力上，还由于 Vivado HLS 代码更加紧凑的特点，体现在代码可维护性和可读性上。此外通过高层次综合，用户仍能有效控制架构及其特性。正确理解和使用 Vivado HLS 程序对实现这一控制起着根本作用。

HLS 视频函数库提供了 OpenCV 接口函数，完成标准 OpenCV 数据类型与 AXI4 流协议之间的数据转换；提供了 AXI4 -流 I/O 函数，允许将 AXI4 流协议转换为所使用的 hls::Mat 数据类型；具有视频处理功能，兼容操作和处理视频的标准 OpenCV 函数。XAPP1167 与 UG902 操作手册都为我们的设计提供了指导与帮助。

C. VSCODE

VSCODE 支持大量 opencv 库与插件的调用。文件目录管理很强大可以，自定义配置，主题，自动保存，可以设置延迟毫秒后保存，也可以设置文件失去焦点时自动保存；集成 Git，快速 diff,而且修改文件后会左边会显示指示器，比如删除会显示红色，增加显示绿色，修改则显示蓝色;智能提示很强大,作为一款编辑器优于 sublime 和 atom；Go to Definition 很方便,按 F12 自动跳转到方法定义处，如果不想跳转，可以直接 shift+F12 实现 Peek 功能，这些都为前期设计方案的 C/C++语言的实现与验证提供了便利。

3. 实验步骤

1）学习整理并确定手势识别的各类算法，在 VSCODE 软件框架进行验证分析

2）通过 HLS 工具将 C++语言的手势识别算法，进行 C 仿真验证分析，并封装为硬件 IP

3）在 Vivado 2017.4 工具中编写 SCCB 协议以完成数据读写与控制，并进行行为仿真

4）在 Vivado 2017.4 的 block designer 工具中将 HLS IP、ZYNQ ARM 以及 SCCB 数据传输 IP 进行级联，生成 bit 流和 tcl 文件（由于疫情原因无法上板验证）

4. 实验分工

04017419 高佳峻：在本实验中完成 HLS 数据类型与 directive 接口框架；HLS 肤色检测，边缘检测，提取 FASTX 角点蒙版，打印手掌重心与分类结果；Hls-ip 相关 block designer 搭建（with arm/without arm）与 SDK 程序编写；Jupyter Notebook 上的 python 代码实现；课上进度汇报与问题汇总；部分实验报告撰写。

04017445 刘昊东：在本实验中完成 HLS 中手势指尖识别部分、连通域检测与提取部分，霍夫直线检测部分，提取 FASTX 角点检测，边缘检测；vscode 中基于 C++的 opencv 代码仿真验证算法性能效果，部分实验报告撰写汇总。

04017429 寇周斌：在本实验中完成硬件部分 OV5640 驱动时序的编写与验证（仿真）；基于 OV5640 外设的验证功能的 block design 搭建与自定义 IP 的编写；整体 block design 的数据转换连接，关于 axi4 协议的调用；部分实验报告撰写。

5. 实验算法原理

A. 肤色检测

1）基于 RGB 颜色空间的简单阈值肤色识别

理论简易判别算式：

$$R > 95 \;\&\&\; G > 40 \;\&\&\; B > 20 \;\&\&\; R > G \;\&\&\; R > B$$

$$\&\& \; \mathrm{Max}(R, G, B) - \mathrm{Min}(R, G, B) > 15 \;\&\&\; \mathrm{Abs}(R - G) > 15$$

均匀光照下：

$$R > 95 \&\& G > 40 \&\& B > 20 \&\& R > G \&\& R > B$$

$$\&\& \mathrm{Max}(R, G, B) - \mathrm{Min}(R, G, B) > 15 \&\& \mathrm{Abs}(R - G) > 15$$

侧光拍摄环境下：

$$R > 220 \&\& G > 210 \&\& B > 170 \&\& R > B \&\& G > B \&\& \mathrm{Abs}(R - G) \leq 15$$

这种皮肤检测的完整性并不高，一些稍微光线不好的区域也没法检测出皮肤来。另外，这种基于 RBG 范围来判定皮肤的算法太受光线的影响了，鲁棒性确实不好。

2) **基于椭圆皮肤模型的皮肤检测**

经过大量的皮肤统计信息可得，如果将皮肤信息映射到 YCrCb 空间，则在 CrCb 二维空间中这些皮肤像素点近似成一个椭圆分布。因此如果我们得到了一个 CrCb 的椭圆，对于一个坐标(Cr, Cb)，我们只需判断它是否在椭圆内（包括边界），如果是，则可以判断其为皮肤，否则就是非皮肤像素点。

3）**YCrCb 颜色空间 Cr 分量+Otsu 法阈值分割**

先简单介绍 YCrCb 颜色空间：YCrCb 即 YUV，其中“Y”表示明亮度（Luminance 或 Luma），也就是灰阶值；而“U”和“V” 表示的则是色度（Chrominance 或 Chroma），作用是描述影像色彩及饱和度，用于指定像素的颜色。“亮度”是透过 RGB 输入信号来建立的，方法是将 RGB 信号的特定部分叠加到一起。“色度”则定义了颜色的两个方面—色调与饱和度，分别用 Cr 和 Cb 来表示。其中，Cr 反映

了 RGB 输入信号红色部分与 RGB 信号亮度值之间的差异。而 Cb 反映的是 RGB 输入信号蓝色部分与 RGB 信号亮度值之间的差异。

该方法的步骤如下：

a.将 RGB 图像转换到 YCrCb 颜色空间，提取 Cr 分量图像

b.对 Cr 做自二值化阈值分割处理（Otsu 法）

4）基于 YCrCb 颜色空间 Cr,Cb 范围筛选法

资料显示，正常黄种人的 Cr 分量大约在 133 至 173 之间，Cb 分量大约在 77 至 127 之间。可以根据自己项目需求放大或缩小这两个分量的范围，产生不同的效果。在本实验中，基于 Hls 流处理像素点的特性以及实现复杂度的要求，我们最终选取了本方案

B. 边缘检测

1）索贝尔算子法

在技术上，它是一离散性差分算子，用来运算图像亮度函数的灰度之近似值。在图像的任何一点使用此算子，将会产生对应的灰度矢量或是其法矢量，Sobel 卷积因子为：

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

图 1：Sobel 卷积横向及纵向因子

该算子包含两组 3x3 的矩阵，分别为横向及纵向，将之与图像作平面

卷积，即可分别得出横向及纵向的亮度差分近似值。如果以 A 代表原始图像，P_X 及 P_Y 分别代表经横向及纵向边缘检测的图像灰度值，其公式如下：

$$P_X = \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix} A, \quad P_Y = \begin{pmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} A$$

图像的每一个像素的横向及纵向灰度值通过以下公式结合，来计算该点灰度的大小：

$$\mathrm{G} = \sqrt{G_X^2 + G_Y^2}$$

通常，为了提高效率 使用不开平方的近似值，如果梯度 G 大于某一阀值 则认为该点(x,y)为边缘点：

$$|G| = |G_X| + |G_Y|$$

然后可用以下公式计算梯度方向：

$$\Theta = \arctan\left(\frac{G_Y}{G_X}\right)$$

Sobel 算子根据像素点上下、左右邻点灰度加权差，在边缘处达到极值这一现象检测边缘。对噪声具有平滑作用，提供较为精确的边缘方向信息，边缘定位精度不够高。当对精度要求不是很高时，是一种较为常用的边缘检测方法，但是在 Hls 中 hls::Sobel 可提供的调节参数比较有限，实现也与 VSCODE 中的 Sobel 有较大不同，检测到的边缘很难为之后的角点指尖检测提供便利。

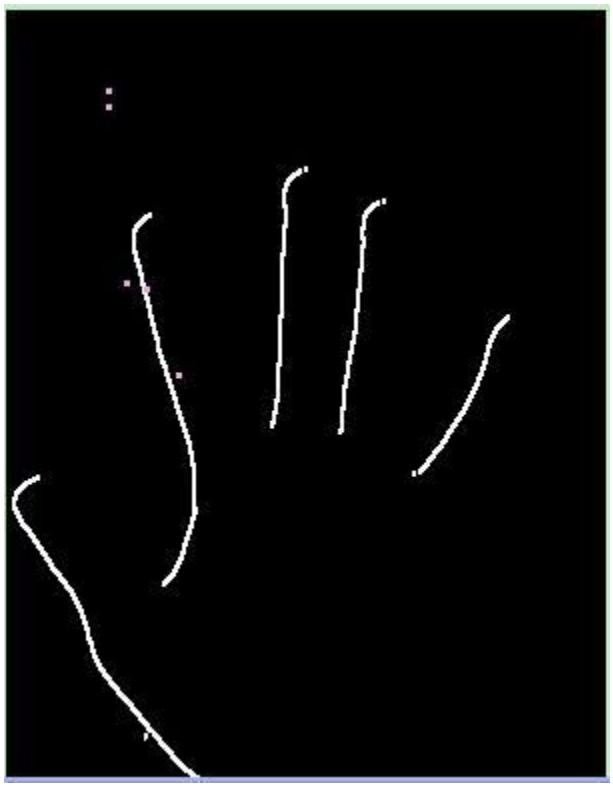


图 2：Hls 中的 Sobel 卷积结果

2）**其他卷积算子边缘检测法**

a. 罗伯茨交叉边缘检测（Roberts Cross operator）

+1	0
0	-1

Gx

0	+1
-1	0

Gy

图 3：罗伯茨交叉边缘检测横向及纵向因子

Roberts 算子采用对角线方向相邻两像素之差近似梯度幅值检测边缘。检测水平和垂直边缘的效果好于斜向边缘，定位精度高，对噪声敏感

b. 普利维特算子(Prewitt operate)

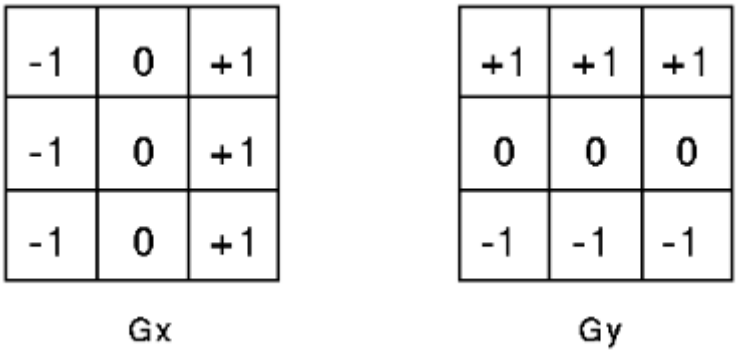


图 4：普利维特算子横向及纵向因子

Prewitt 算子利用像素点上下、左右邻点灰度差，在边缘处达到极值检测边缘。对噪声具有平滑作用，定位精度不够高。

以上两种算子法与 Sobel 的计算方法基本相同，但是在 hls 中均无对应实现库，在 VSCODE 中的表现也并不好，因此不予采用。

3）**一阶梯度边缘检测法**

在实际情况中理想的灰度阶跃及其线条边缘图像是很少见到的，同时大多数的传感器件具有低频滤波特性，这样会使得阶跃边缘变为斜坡性边缘，看起来其中的强度变化不是瞬间的，而是跨越了一定的距离。而在我们的实验实现中，由于已经做出了肤色检测提取，图像的灰度阶跃比较明显，又为了避免核卷积矩阵操作对于手掌（尤其是指尖部分）边缘信息的丢失以及整个程序的流处理操作思想，可以直接判断灰度值的跳变完成对边缘的检测，也为了指尖检测提供了尽可能丰富的信息。

C. 霍夫曼变换直线检测

霍夫变换（Hough Transform）于 1962 年由 Paul Hough 首次提出，后于 1972 年由 Richard Duda 和 Peter Hart 推广使用，是图像处

理领域内从图像中检测几何形状的基本方法之一。经典霍夫变换用来检测图像中的直线，后来霍夫变换经过扩展可以进行任意形状物体的识别，例如圆和椭圆。霍夫变换运用两个坐标空间之间的变换，将在一个空间中具有相同形状的曲线或直线映射到另一个坐标空间的一个点上形成峰值，从而把检测任意形状的问题转化为统计峰值问题。Hough 直线检测的基本原理在于利用点与线的对偶性，在我们的直线检测任务中，即图像空间中的直线与参数空间中的点是一一对应的，参数空间中的直线与图像空间中的点也是一一对应的。这意味着我们可以得出两个非常有用的结论：

1）图像空间中的每条直线在参数空间中都对应着单独一个点来表示；

2）图像空间中的直线上任何一部分线段在参数空间对应的是同一个点。

因此 Hough 直线检测算法就是把在图像空间中的直线检测问题转换到参数空间中对点的检测问题，通过在参数空间里寻找峰值来完成直线检测任务。

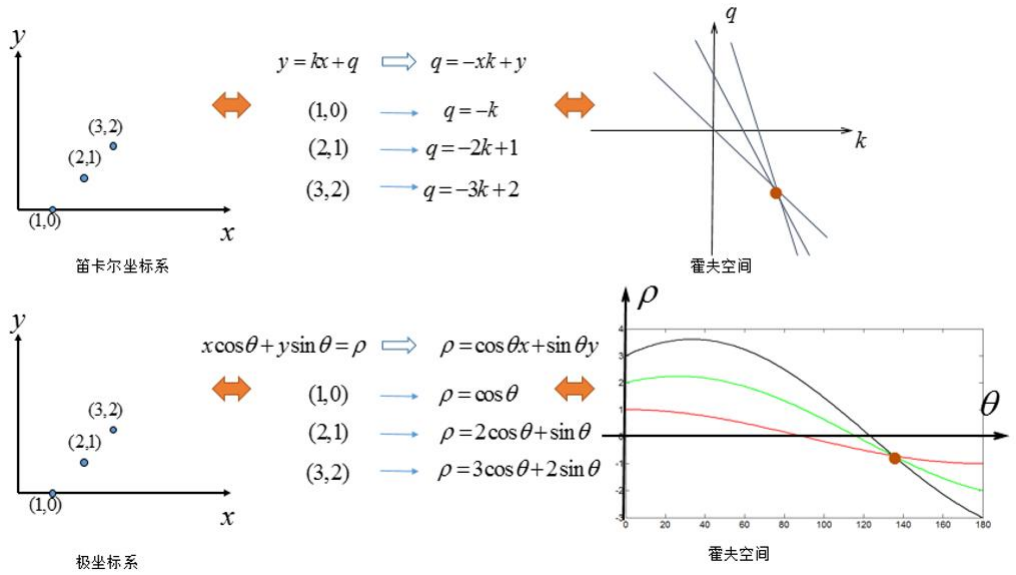


图 5：霍夫变换直线检测原理

使用霍夫变换检测直线具体步骤如下：

1.彩色图像->灰度图

2.去噪（高斯核）

3.边缘提取

4.二值化（判断此处是否为边缘点，就看灰度值==255）

5.映射到霍夫空间（准备两个容器，一个用来展示 hough-space 概况，一个数组 hough-space 用来储存 voting 的值，因为投票过程往往有某个极大值超过阈值，多达几千，不能直接用灰度图来记录投票信息）

6.取局部极大值，设定阈值，过滤干扰直线

7.绘制直线、标定角点

霍夫直线检测的优点在于 Hough 直线检测的优点是抗干扰能力强，对图像中直线的残缺部分、噪声以及其它共存的非直线结构不敏感，能容忍特征边界描述中的间隙，并且相对不受图像噪声的影响。

其缺点是 Hough 变换算法的特点导致其时间复杂度和空间复杂度都很高，并且在检测过程中只能确定直线方向，丢失了线段的长度信息。由于霍夫检测过程中进行了离散化，因此检测精度受参数离散间隔制约。

在我们一开始的尝试中使用了霍夫直线检测，正常人手的几何特性导致其很难分辨出目标手指的直线（例如图片中张开五指应该可以分辨出 10 个目标直线），使其无法收集比较长度信息的缺点暴露无疑，最终在 Hls 中我们放弃了使用霍夫直线检测。

D. FAST 角点检测

FAST 的全称为 Features From Accelerated Segment Test。Rosten 等人将 FAST 角点定义为：若某像素点与其周围领域内足够多的像素点处于不同的区域，则该像素点可能为角点。也就是某些属性与众不同，考虑灰度图像，即若该点的灰度值比其周围领域内足够多的像素点的灰度值大或者小，则该点可能为角点。FAST 算子如其名，计算速度快，可以应用与实时场景中。在 FAST 特征提出之后，实时计算机视觉应用中特征提取性能才有显著改善。目前以其高计算效率(computational performance)、高可重复性(high repeatability)成为计算机视觉领域最流行的角点检测方法。

算法步骤如下：

1）对固定半径圆上的像素进行分割测试，通过逻辑测试可以去处大量的非特征候选点

2）基于分类的角点特征检测，利用 ID3 分类器根据 16 个特征判决候选点是否为角点特征，每个特征的状态为-1，0，1

3）利用非极大值抑制进行角点特征的验证

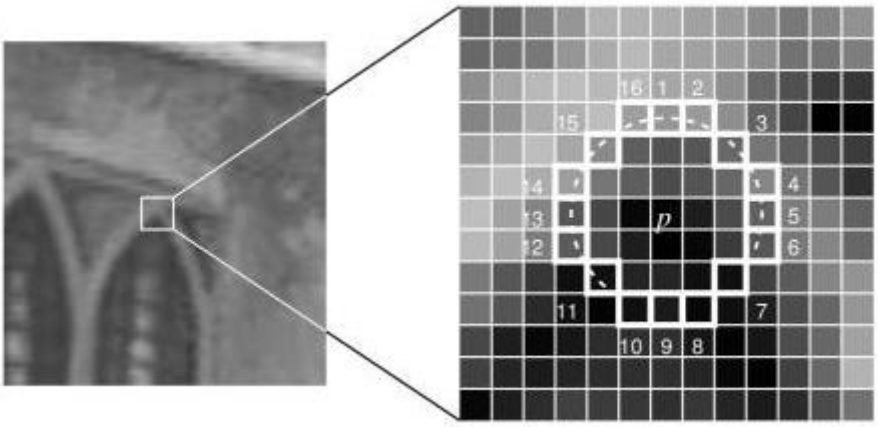


图 6：FAST 角点检测原理

E. 基于指尖与手掌重心距离的手势分类

1）针对肤色检测后的结果，对于整个检测出的手掌部分可以先求取其重心(X_C, Y_C)，利用如下公式计算：

$$X_C = \frac{\sum P_i x_i}{\sum P_i}, \qquad Y_C = \frac{\sum P_i y_i}{\sum P_i}$$

其中(x_i, y_i)是像素点的坐标，p_i是该点的像素值

2）针对边缘检测后的结果，利用 FAST 算法提取出角点

3）遍历角点，逐个计算角点与 1）中得到重心的距离并比较，计算最大值保持的点数，如果大于，那么就暂且认为这个是指尖

4）剔除低于手心（重心）的点和与判别指尖距离过近的重复点

5）在图像上打印指尖点，手掌重心点，手掌边缘以及在右上角标出手势分类后的结果

6. VSCODE 实现

我们首先使用基于 c++的 opencv 对算法效果进行了验证：

- 霍夫直线检测

在 opencv 中对手势进行直线检测，我们希望可以通过手指直线的数目来判断手势数字，效果如下：

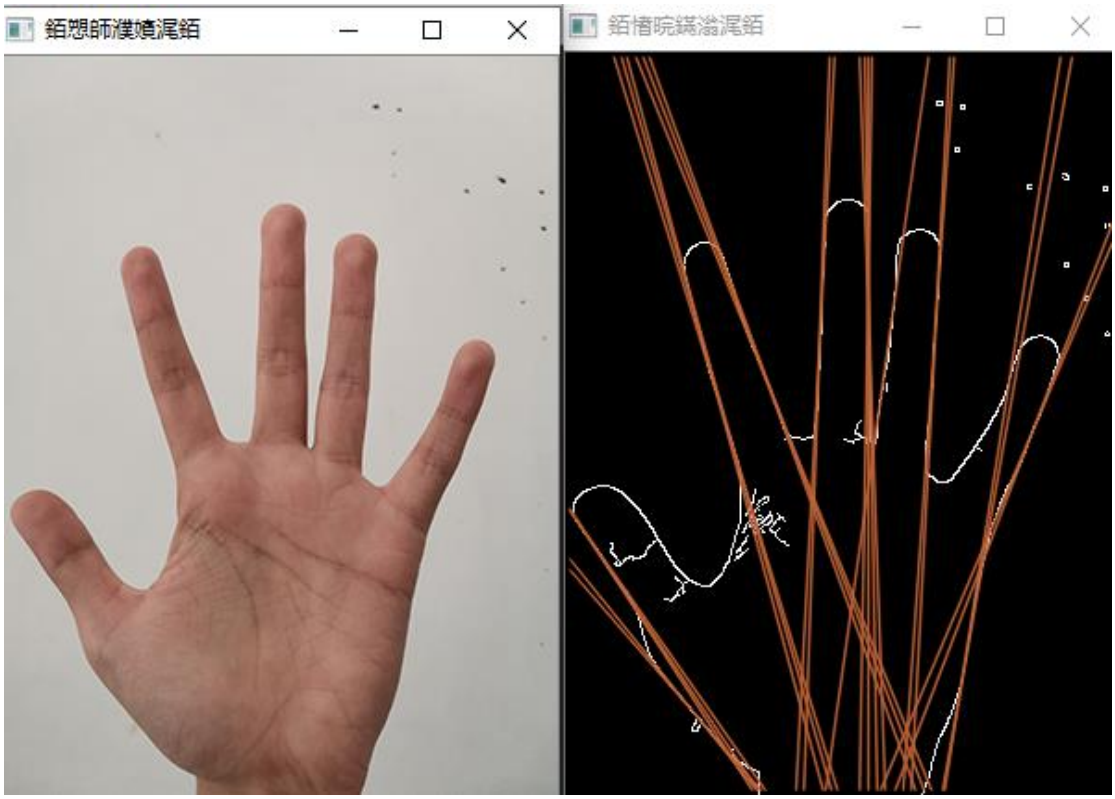


图 7：霍夫直线检测结果

可以看到在 opencv 中对手势检测效果一般，误差大概在 3 条左右，但是在 HLS 中我们实现的效果更差，很难检测出直线个数，所以最后我们放弃了这个方案。

- 指尖检测：

最后我们采用了指尖检测的方案，利用统计分析技术——一种通过统

计样本特征向量来确定分类器的基于概率统计理论的分类方法。可以采用此模型提取手势的指尖的数量和位置，将指尖和掌心连线，采用距离公式计算各指尖到掌心的距离，再采用反余弦公式计算各指尖与掌心连线间的夹角，将距离和夹角作为选择的特征。

具体方案为：手势图像经过二值化处理后，提取手势图像的几何矩特征，取出几何矩特征七个特征分量中的四个分量，形成手势的几何矩特征向量。在灰度图基础上直接检测图像的边缘，利用直方图表示图像的边界方向特征。再通过设定两个特征的权重来计算图像间的距离，再对指尖进行识别：

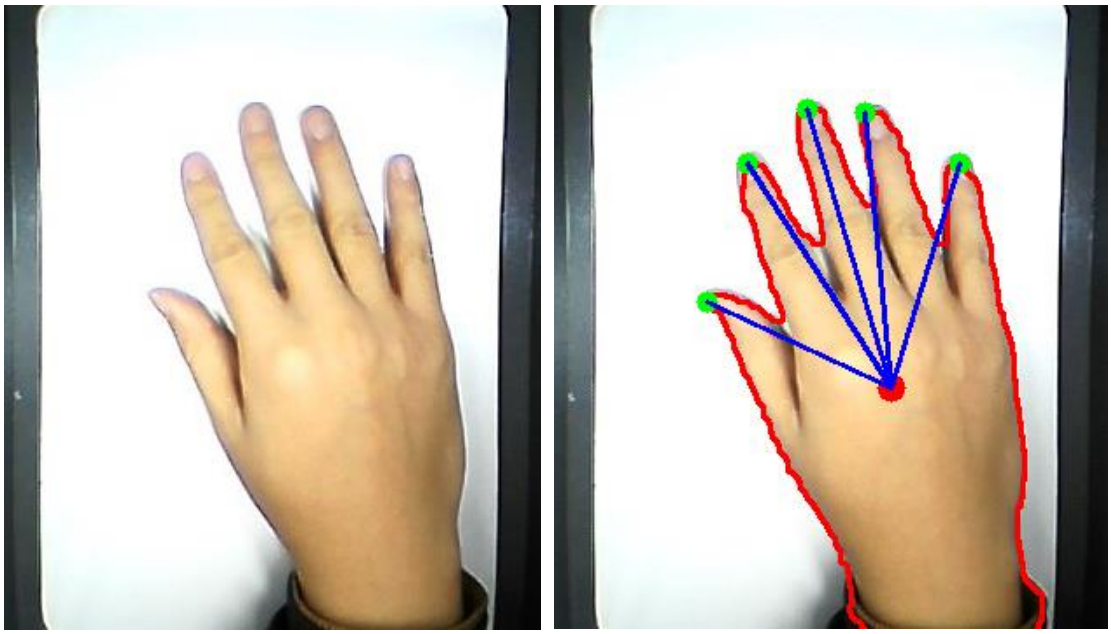


图 8：指尖检测结果

可以看到，基于指尖检测的方案效果较好，可以准确的识别手势数字，我们的程序最后采用基于指尖检测来编程。

- 连通域检测

连通域检测是为了对背景复杂的情况进行程序的完善，基于颜色识别

后，从复杂的噪声背景中提取出手的部分，不进行连通域检测的效果较差，利用种子填充法（在后文介绍），我们在 opencv 中进行验证如下：

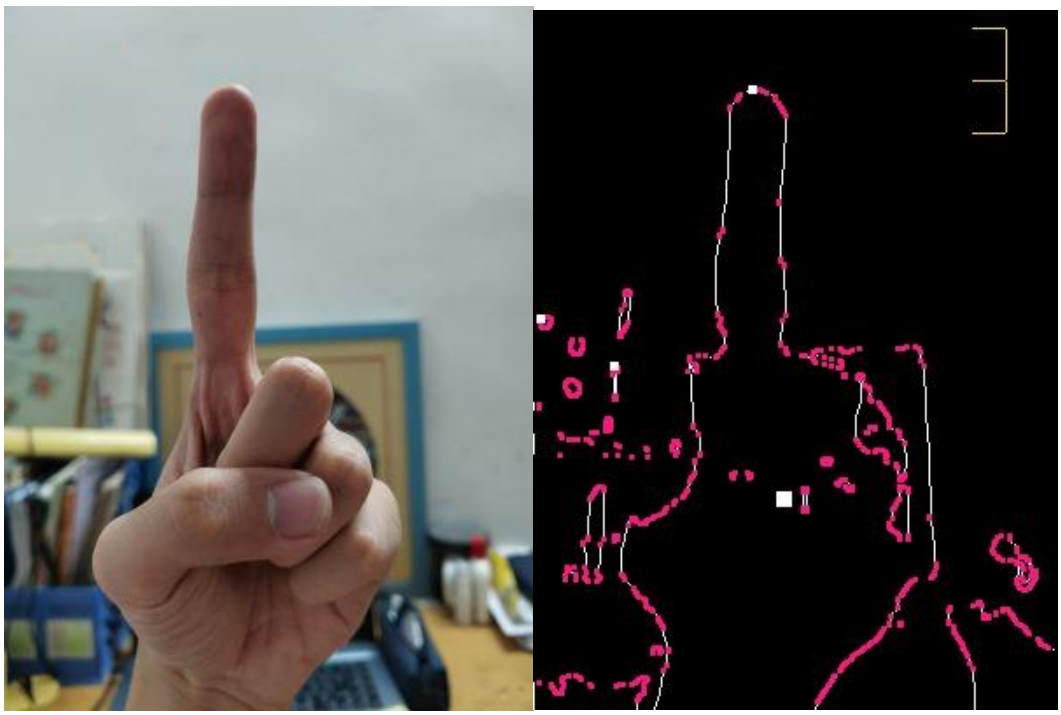


图 9：原图与 HLS 检测结果

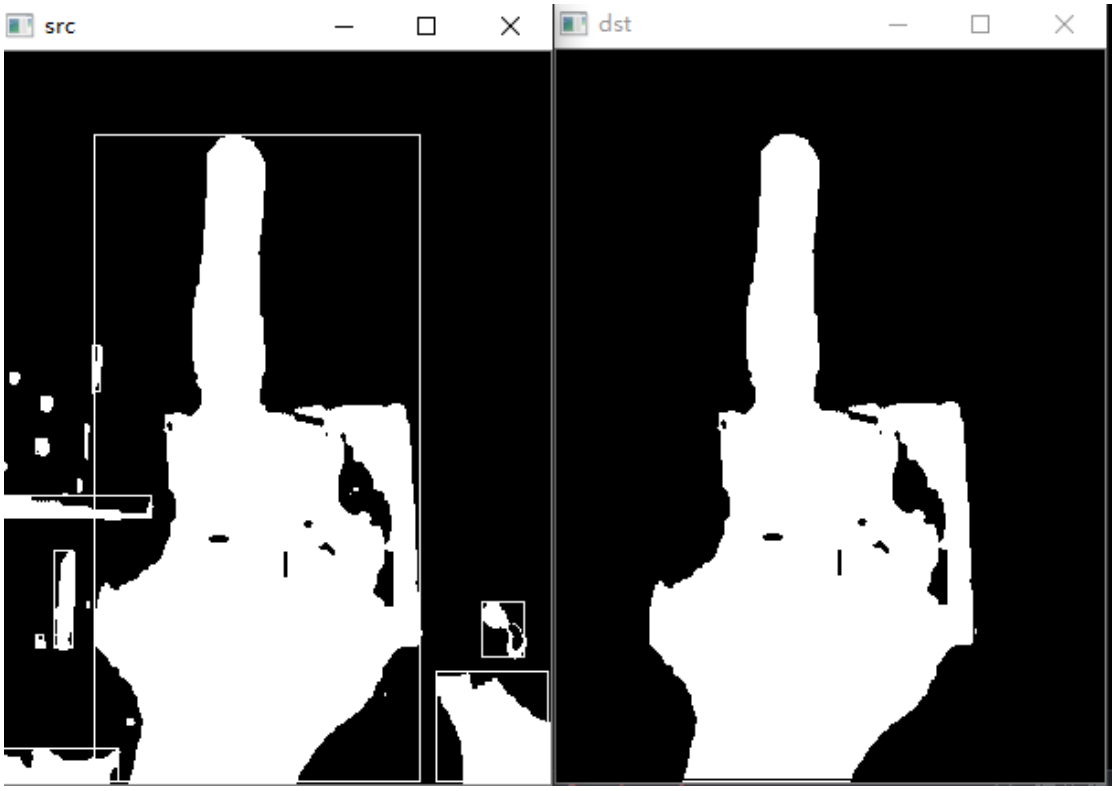


图 10：Opencv 连通域检测并提取结果

7. Vivado Hls 实现

1） 数据类型定义

原始类型	自定义类型	描述
unsigned char	uchar	像素值采用无符号 8 位宽数据表示
hls::stream<ap_axiu<24,1,1,1> >	AXI_STREAM	AXI 视频流处理接口
hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC3>	RGB_IMAGE	RGB3 通道矩阵
hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC1>	GRAY_IMAGE	灰度矩阵
hls::Scalar<3, unsigned char>	RGB_PIXEL	像素点 8 位无符号表示

2） 函数定义

void **hls_skin_dection**(RGB_IMAGE& src, RGB_IMAGE& dst,int rows, int cols, int y_lower,int y_upper,int cb_lower,int cb_upper,int cr_lower,int cr_upper)

提供肤色检测的接口，输入输出均为 RGB 图像，并定义了行数列数以及颜色空间检测的阈值。

void **hls_trouble**(RGB_IMAGE& input, RGB_IMAGE& output, int rows, int cols)

提供边缘检测以及重心提取的接口，将提取到的重心打印到命令行，输入输出均为 RGB 图像，并定义了行数列数

int **bwLabel**(RGB_IMAGE &src, Feather featherList[])

用于连通域检测的函数，最后得到连通域特征的清单（标号），也返回连通域的个数。

void **changecolor**(Feather featherList[], RGB_IMAGE &src, RGB_IMAGE &dst)

用于根据连通域的标签提取最大的连通域图像，保存到 dst 中。

void **ImgProcess_Top**(AXI_STREAM& input, AXI_STREAM& output,int rows, int cols, int y_lower,int y_upper,int cb_lower,int cb_upper,int cr_lower,int cr_upper)

为整个 hls 工程的 top 函数，输入输出为 AXI_STREAM 类型，便于与 block designer 和 test 文件中的接口进行交互，同时保留了行列数和阈值等关键信息便于调参。

图 11：Vivado Hls 代码流程框架

在 testbench 的整个流程中，首先将图片导入为 opencv 库中的 IplImage 类型，然后将其转换为 AXI_STREAM 类型（视频流），接下来转换为 hls::Mat 类型利用我们自己编写的模块以及 hls 图像处理的库函数进行 pipeline 操作，值得注意的是，这种处理方式要求一个图像在被处理之后不能再次被使用，因此要适时使用 hls::duplicate 函数进行一分二的操作，以利用之前图像的信息,输出过程与输入过程相反，详情请见图 7 中的代码流程框架。

3） Directive 选择

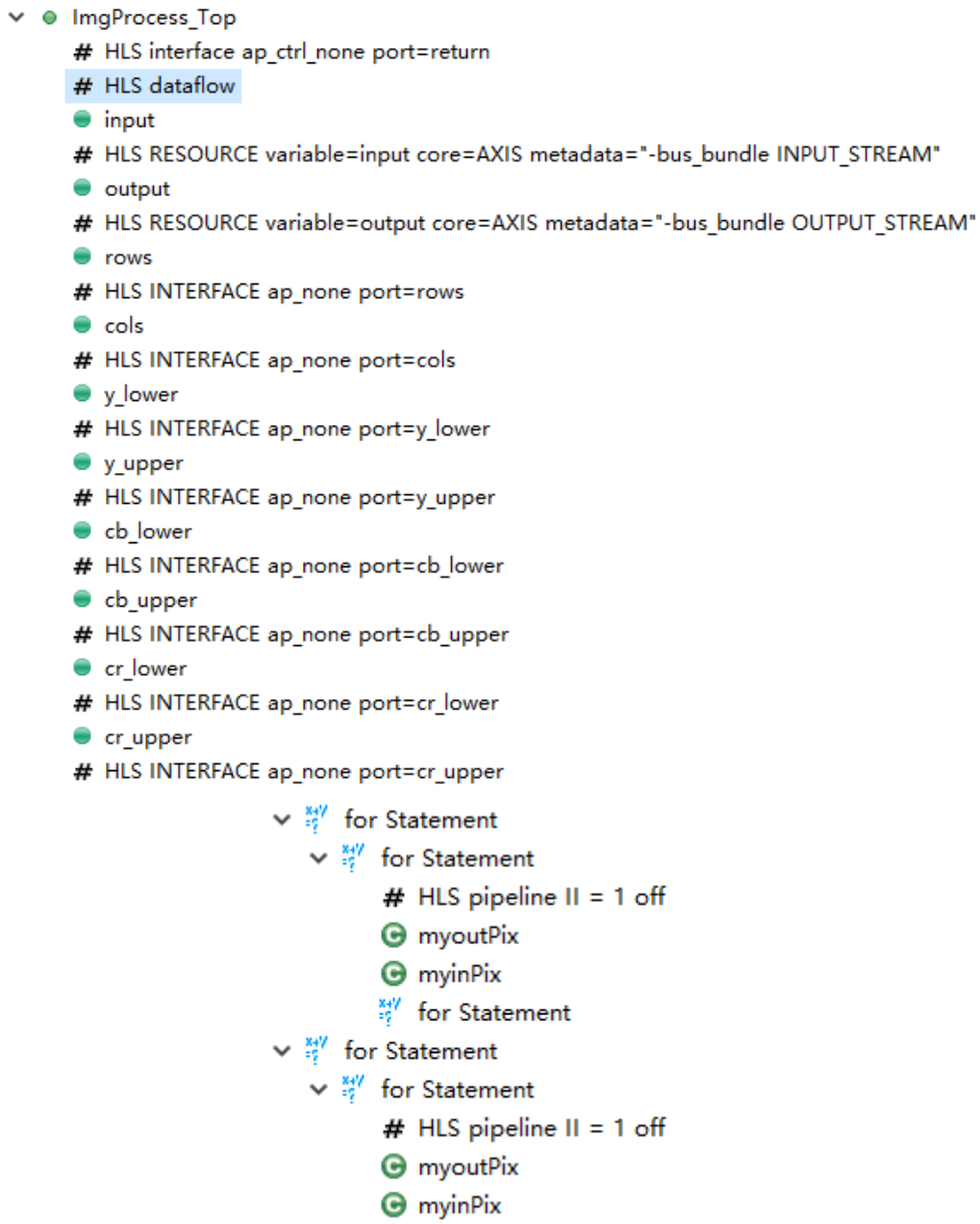


图 12：Vivado Hls directive

在 directive 的选择中，我们将 input 和 output 定义为视频流接口，形式如**#pragma** HLS RESOURCE variable=XX core=AXIS metadata="-bus_bundle INPUT_STREAM"；将行列参数以及颜色空间阈值定义为 HLS INTERFACE ap_none port；为了 pipeline 操作，使用数据流处理

方式，添加 HLS dataflow，完成二维到一维的数据转换，加快处理速度，同时对于 for 循环操作定义 HLS pipeline II =1 off，减小冗余。

4）程序运行结果（以手势 3、5 为例）

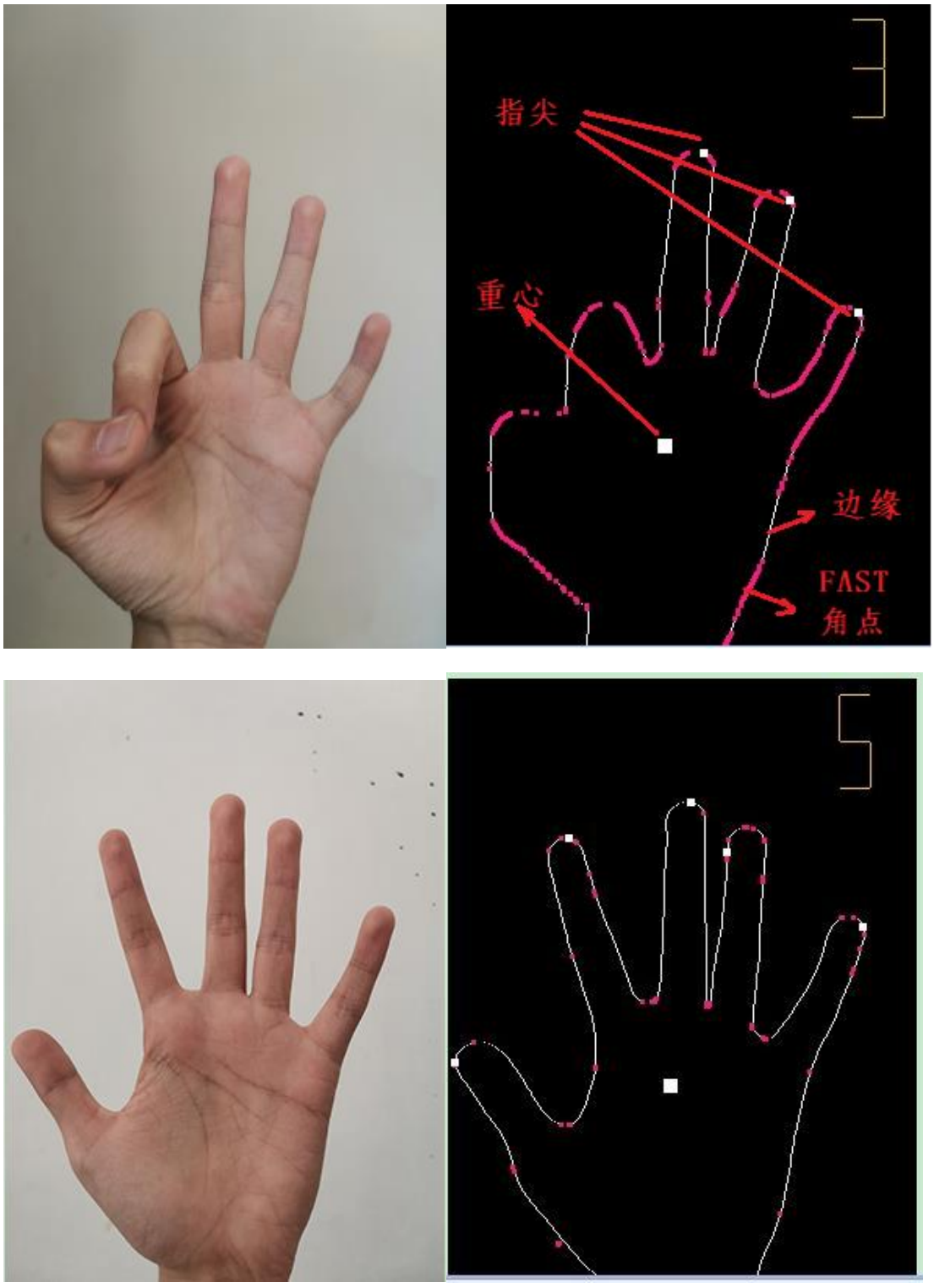


图 13：Vivado Hls testbench 仿真结果

5）程序鲁棒性提升

以上为我们在白色纯色背景下检测的手势识别，实现了基本功能。为了解决我们工作初始复杂背景无法较好识别的问题，提升我们程序的鲁棒性，针对一些背景复杂情况进行修改：

我们使用连通域检测的方法进行编程，识别出连通域最大的部分，即手部分，滤除其他多余的环境干扰。由于 HLS 中 opencv 的库不全，无法使用原来的库函数直接进行操作，所以采用原始的算法进行连通域检测和提取。

我们采用种子填充算法（Seed Filling）：

种子填充方法来源于计算机图形学，常用于对某个图形进行填充。思路：选取一个前景像素点作为种子，然后根据连通区域的两个基本条件（像素值相同、位置相邻）将与种子相邻的前景像素合并到同一个像素集合中，最后得到的该像素集合则为一个连通区域。

下面给出基于种子填充法的连通区域分析方法：

（1）扫描图像，直到当前像素点 B(x,y) == 1：

a、将 B(x,y)作为种子（像素位置），并赋予其一个 label，然后将该种子相邻的所有前景像素都压入栈中；

b、弹出栈顶像素，赋予其相同的 label，然后再将与该栈顶像素相邻的所有前景像素都压入栈中；

c、重复 b 步骤，直到栈为空；

此时，便找到了图像 B 中的一个连通区域，该区域内的像素值被标记为 label；

（2）重复第（1）步，直到扫描结束；

扫描结束后，就可以得到图像 B 中所有的连通区域；Seed-Filling 算法示意图如下：

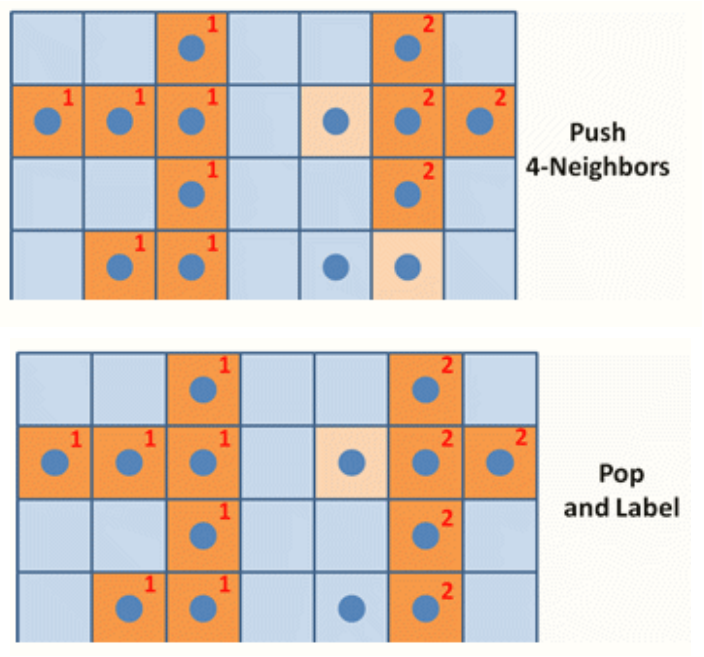
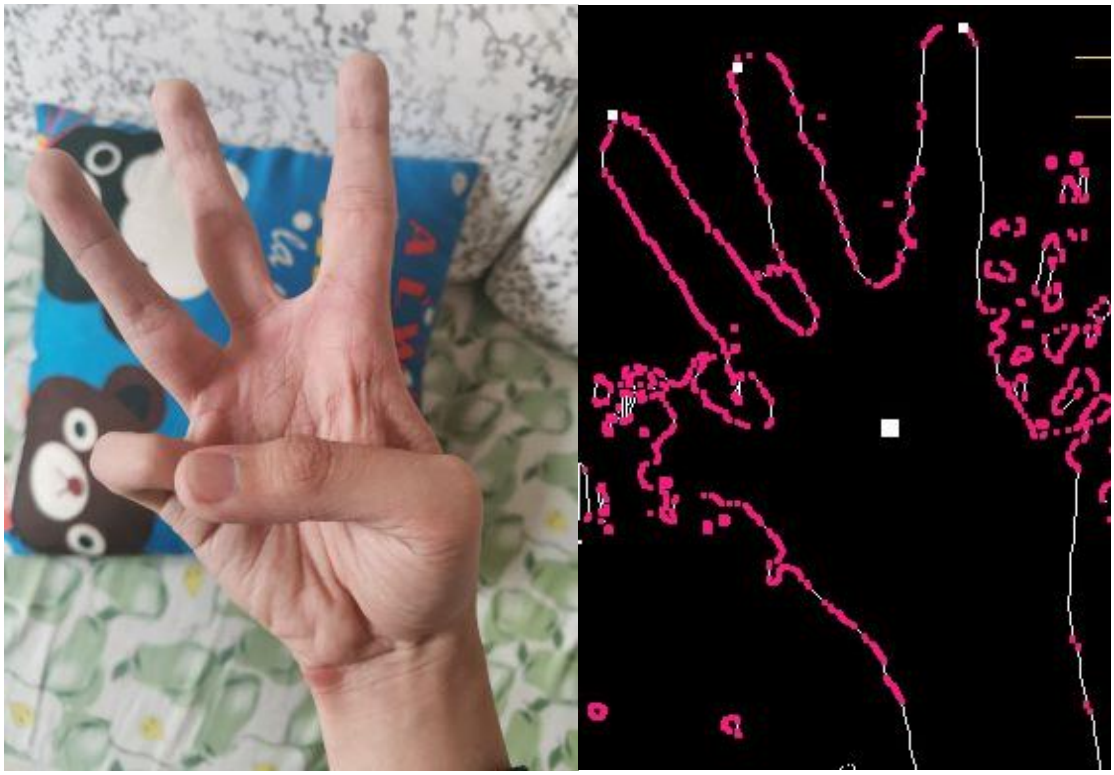


图 14：种子填充算法（Seed Filling）示意图

采用该算法对程序进行完善，得到结果效果较为理想：



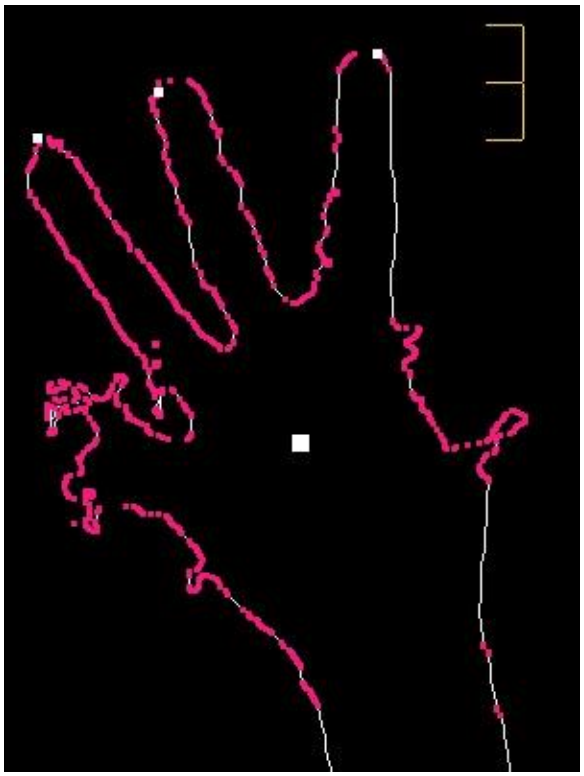


图 15：Vivado Hls 结合连通域检测的仿真结果

可以看到，在背景复杂的情况下，不进行连通域检测会识别错误，补充该算法后，很好的识别出了手势 3，程序鲁棒性有了提升。

6）程序综合结果

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	9.63	1.25

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	216
FIFO	0	-	411	2602
Instance	531	22	19092	30336
Memory	260	-	128	6
Multiplexer	-	-	-	81
Register	-	-	9	-
Total	791	22	19640	33241
Available	1000	900	343800	171900
Utilization (%)	79	2	5	19

图 16：Vivado Hls 综合结果

可以发现，时钟时序满足要求，系统的资源也没有耗尽，满足设计要求。我们随后将其生成 IP catalog，方便 Vivado block design 的调用。

8. 外设 OV5640 实现

1）硬件实现基本目标

在课程设计准备之初，我们计划采用摄像头 OV5640 作为外设从而实现图像的实时采集，在这个过程中，利用 Vivado VHDL 实现对摄像头的驱动，并导出 IP 实现 BlockDesign 间接，与图像处理模块实现连接。

2）基于 sccb 协议的驱动时序编写

通过查阅资料了解到 sccb 协议是与 IIC 协议十分类似的通信协议，且相比于 IIC 协议，sccb 主要的区别是每读完一个字节主机不需要像 IIC 一样主机发送一个 NA 信号。

a.工作模式：sccb 有两种工作模式，分别为一主机多从机模式和一主一从模式，主要的区别在于对 SCCB_E 信号的处理上，在此处我们使用一主一从模式即可，对 SCCB_E 信号默认赋值为低。

其中 SIO_C 相当于 IIC 中的 SCL 控制信号，总线空闲时主机驱动此引脚为 1；当驱动 SIO_E 为 0 时，主机驱动此引脚为 0 或 1；当挂起时主机驱动 SIO_C 为 0；SIO_D 只能在 SIO_C 为 0 时发生变化；

SIO_D 相当于 IIC 中的 SDC 信号，当总线空闲时保持浮动，状态不固定（0、1 或高阻态），相当于数据位。

Figure 1-1 SCCB Functional Block Diagram

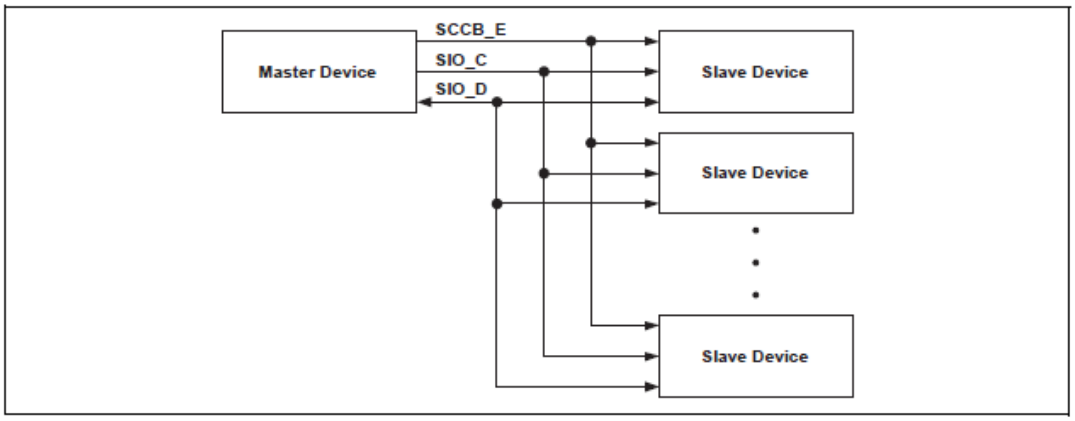
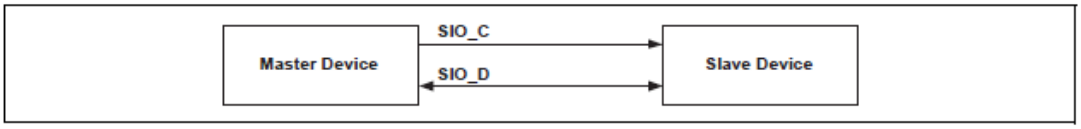
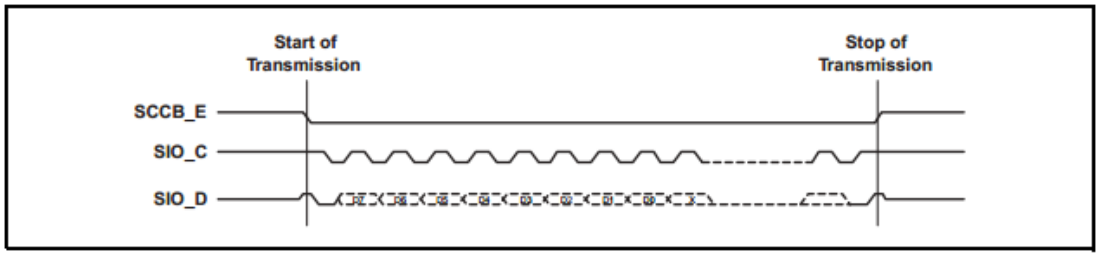


Figure 1-2 2-Wire SCCB Functional Block Diagram

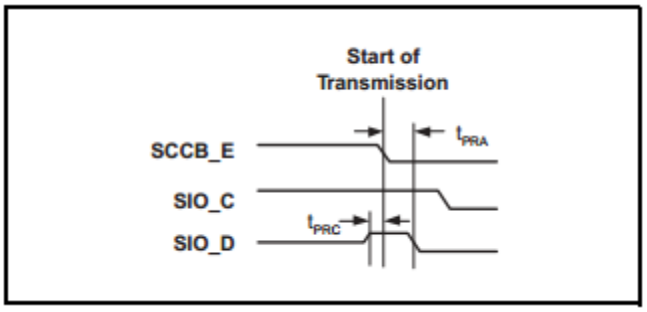


b.SCCB 时序分析



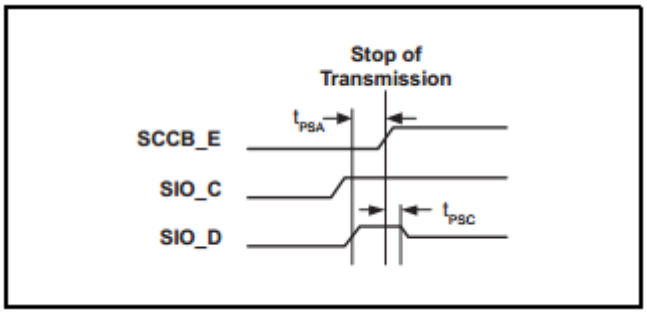
根据上图可知，SCCB_E 为低电平时传输有效，SIO_C 为高电平时 SIO_D 读取数据（SIO_C 为低电平时 SIO_D 改变）。

在开始传输时，主要关注 SIO_C 与 SIO_D 之间的时序变化：



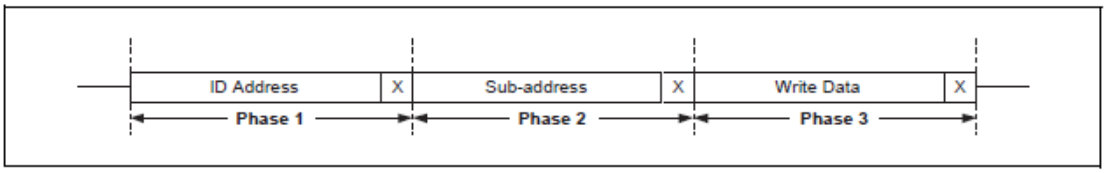
在输出模式下，SIO_D 先置高，当经过 t_{PRC} 和 t_{PRA} 后，若 SIO_C 为高电平则 SIO_D 恢复低电平。

传输结束时，在 SIO_C 为高电平时 SIO_D 从低置为高，并将 SCCB_E 置为 1，表示传输结束。

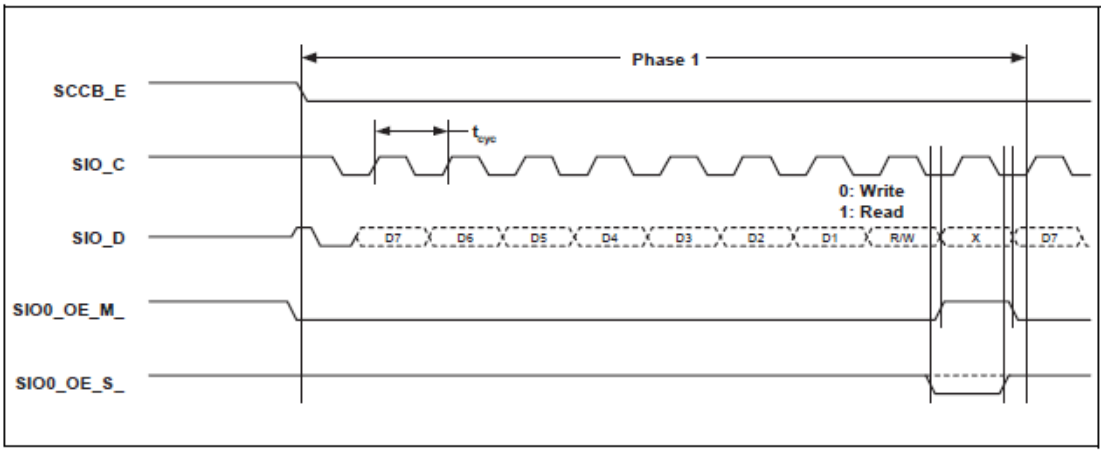


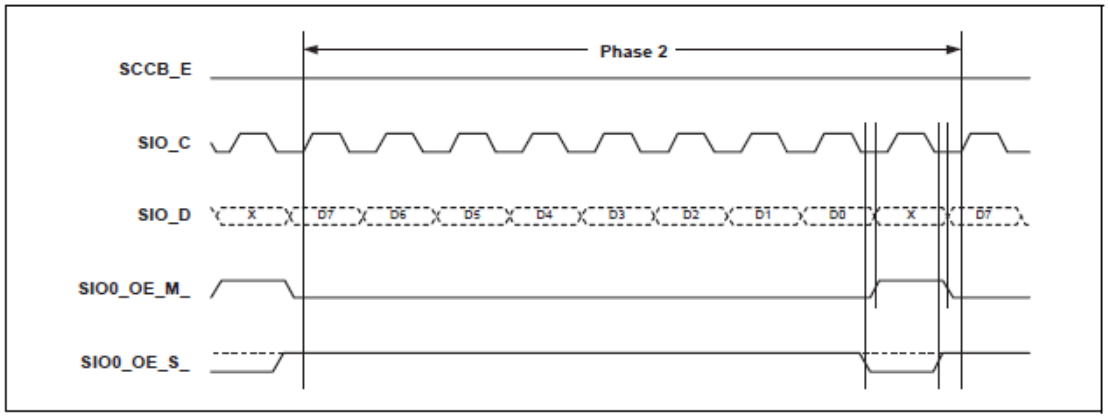
读写规则：

数据传输时为 3 相循环传输，



三个相位的具体读写时序规则如下所示





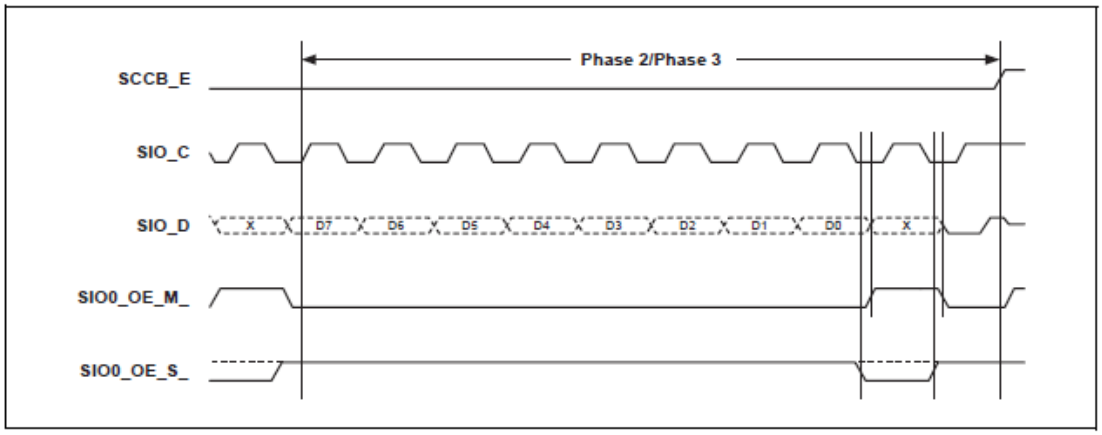


图 17：相位读写时序

3）sccb 时序控制仿真验证

根据以上时序规则编写出 sccb 协议的基本驱动方式，并通过行为及仿真的方式验证编写的正确性：

由于只能进行行为及仿真，所以在时序文件中设置写标志与结束标志自动翻转，在传输的寄存器中存入值 654321，可以看到在仿真图中能够实现三相传输，由此可以证明 sccb 时序驱动的正确性。

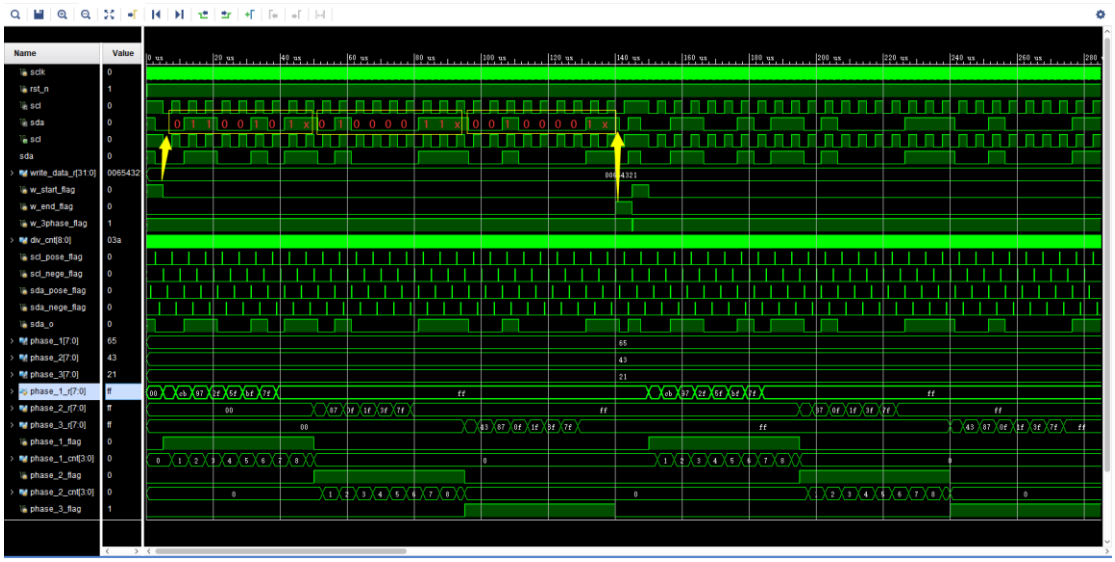


图 18：OV5640 驱动时序验证

4）输出 IP 实现与 blockdesign 设计

通过将编写好的 sccb 协议驱动导出为自定义 IP，并将其与现有的

process system 的 IP 进行自定义管脚和连接，可以实现摄像头控制模块的 Blockdesign 的初步工程，

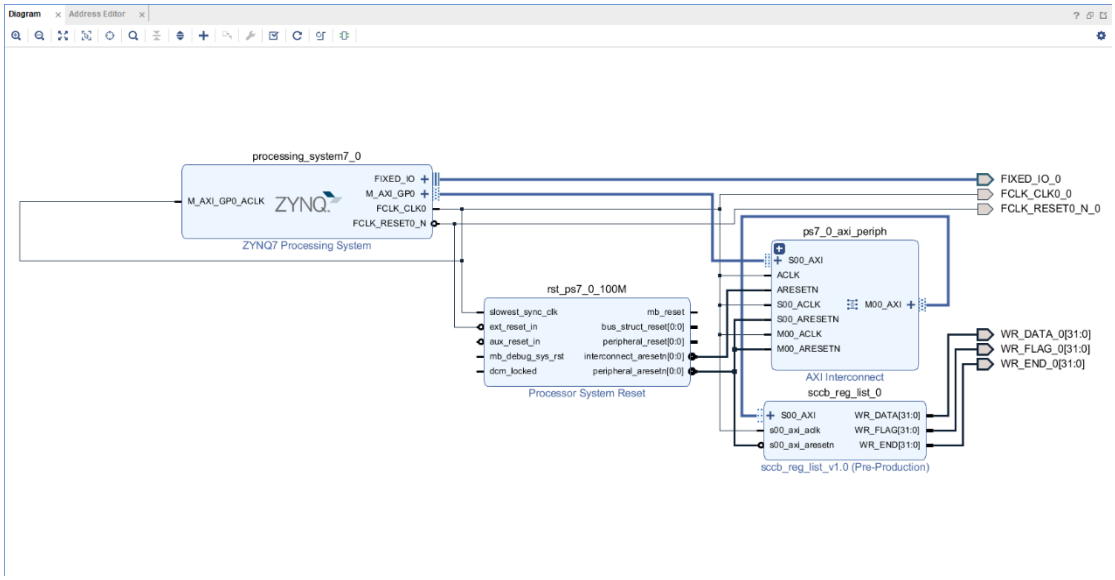


图 19：关于 OV5640 的 blosk design 设计

这里利用 process system 提供一个系统的 clk 时钟，使用 axi4 的协议进行连接，其中对于 sccbIp 的输出，我们设置了一个寄存器作为最终的输出端口，主要的输出为两个写的标志位和数据传输位，这一部分能够为硬件实物验证提供条件，通过将输出的比特流文件导入 ZYNQ 的板子上，可通过示波器抓取时序的波形，进行验证。

9. Block design 实现

1） 不使用 ZYNQ 的 ARM 核以及 DMA

Xilinx vivado 下通常的视频流设计，都采用 Vid In to axi4 stream --> VDMA write --> MM --> VDMA read --> axi4 stream to video out 这样的路径。官方参考有 xapp521。但这套方案明显的问题是，缓存图像带来帧延迟，对于一些延迟要求高的任务，反倒不希望有 VDMA 的

参与。此外，对于一些简单的应用，不希望有 Zynq PS 核的参与配置，简化系统复杂度。因此就有了 "Video In to AXI4-stream" to "AXI4-stream to Video Out" directly with VTC without VDMA 的需求。在本课程设计中对于此进行了尝试。

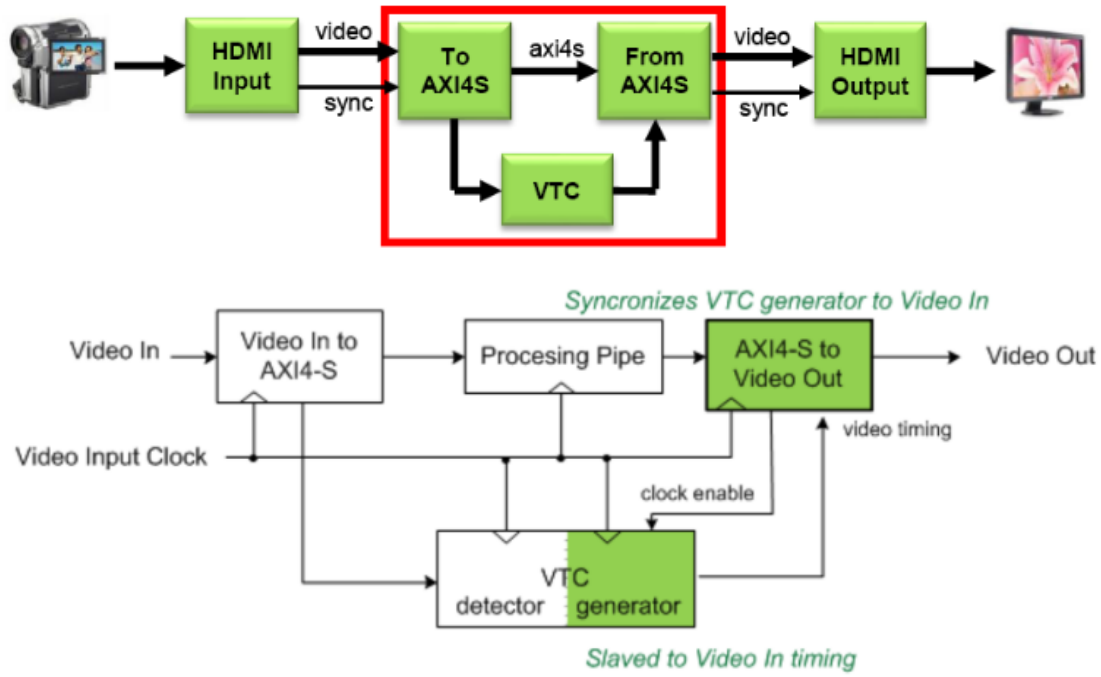


图 20：不使用 ZYNQ 的 ARM 核以及 DMA 的 BD 流程图

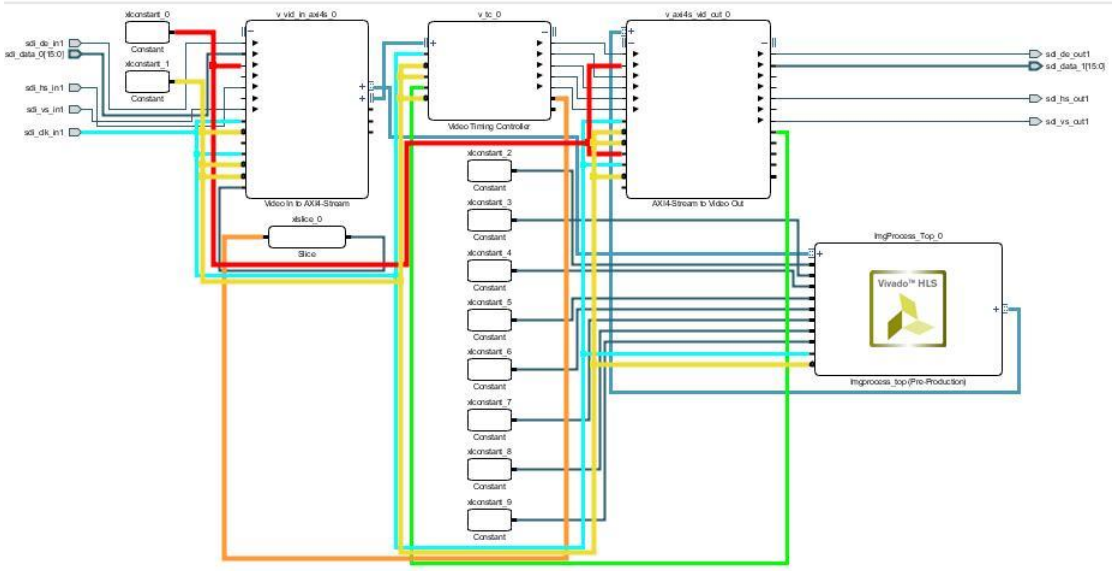


图 21： 本方案 Block Design 流程图

注意根据参考文档上的描述，要把 axis_enable 接 vtc 的 INTC 中的第

8 位（Detect Locked 信号），去 INTC 中第 8 位出来的 IP 使用 slice 来截取；这种没有 VDMA 的框架下 timing mode 要选 slave 模式。

2） 使用 ZYNQ 的 ARM 核以及 DMA

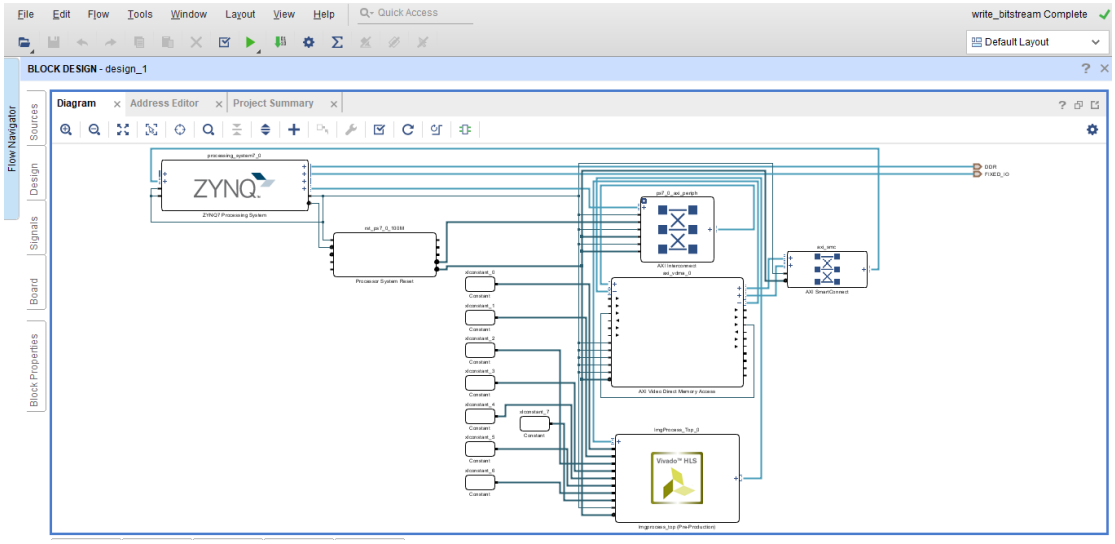


图 22： 本方案 Block Design 流程图

根据实验指导书中的连接方式，进行 HLS ip 通过 DMA 与 Zynq 的 ARM 核的交互，在 ZYNQ 中预先放好一组图片解析后的数据，完成整个系统的互联互通。在图中也可以观察到 **bit 文件已经生成成功**。之后我们 Export hardware 并且 launch SDK，对于图像的输入要求，我们利用 Matlab 工具将一副图像分割成为 B,G,R 三个通道并且作为数组写入 SDK 的 helloworld.c 中（图像数据波形产出见 generate_waveform 函数），定义了 DMA 方式的启动、读写、判别错误的规范（见 init_dma 和 main 函数），由于疫情影响手中也无可用开发板，故无法上板验证。

图 23： Matlab 图像数据处理缩略图

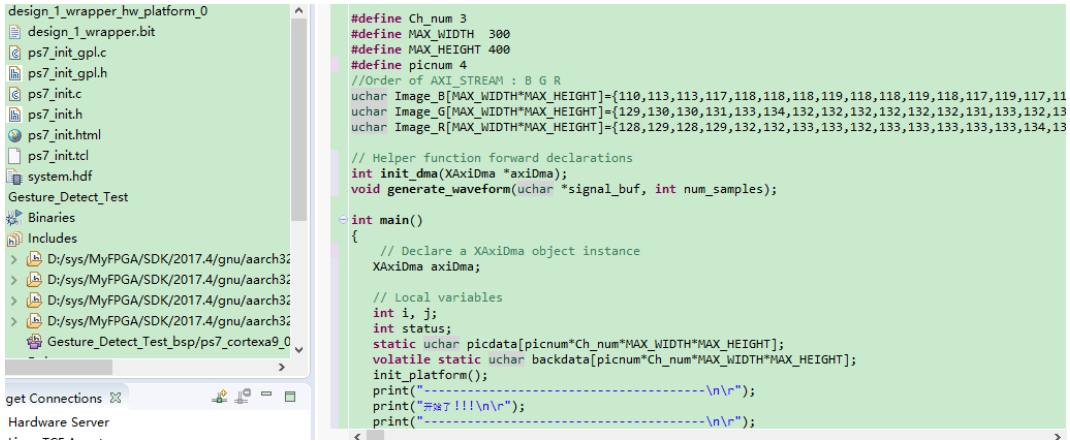


图 24： SDK 缩略图

10. Python 实现

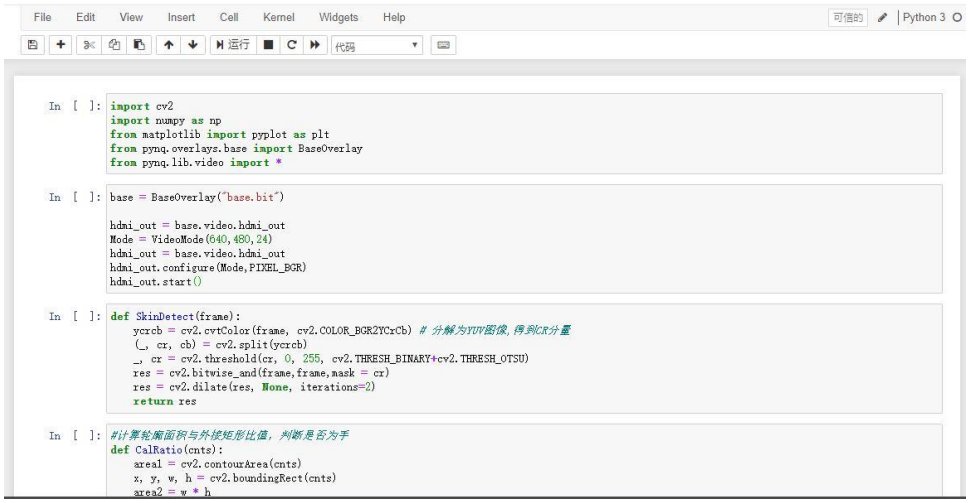


图 25： Jupyter Notebook 缩略图

在本实验中我们还尝试了编写了与 pynq 交互的 jupyter notebook 文件，采用 hdmi_in 和 out 的方式，采用 opencv 库先对图片进行简要的处理，然后完成手势识别。由于疫情影响，同样无法上板验证。

11. 实验总结

在本次课程设计中我们通过一个平台 PYNQ-Z2（含一个芯片 XC7Z035）、一套设计工具 Vivado−HDL/SysGen/HLS/BD-SDK）达到了如下目的：

i. 熟悉指标-方案-设计-仿真-调试-集成的系统设计流程

ii. 掌握软、硬件设计方法

iii. 熟练使用设计工具，将算法转化成板级数字系统

iv. 提高文档编写能力

v. 培养团队协作精神

本课程设计的进行过程中，基本完成了预期的目标，当然也存在着一些需要改进的地方，例如识别样本的多样化和复杂化等等。另外，配置 Block Designer 的 VDMA_IP 时，当视频流宽度不是 2 的幂次时，会造成系统资源一定程度的增加，而我们的 RGB 则要求数据位宽为 24 位，这是一个一直存疑未能解决的问题。

12. 致谢

在本实验的进行过程中，黄鹤老师为实验提供了许多指导，对数据类型以及 Block Designer 实现提出了很多指导性的建议，我们对 HLS-C 的区别与传统 C++的用法更加的熟悉，使得我们更加深入地理解了 Vivado 各个流程之间的关联操作，对于我们完成课程设计有很大的帮助。