上海交通大学

课程大作业报告

课程名称: 微纳电子系统综合设计-I

学　　期: 2019-2020 学年第 1 学期

学院(系): 电子信息与电气工程学院 微纳电子学系

专　　业: 微电子科学与工程

学生姓名: 杨文曦　学号: 517030910221

2020 年 1 月 10 日

目录

图像压缩算法

小组成员：杨文曦，罗恬，刘健伟

1 选题过程

1.1 选题依据

本组在开题答辩时选择的项目是 Google 的 Guetzli 图像压缩开源项目[1]。这个选题是老师推荐的项目之一，我们认为这个项目具有一定的实用价值和一定的挑战性，所以决定选择本题目。

在正式开始大作业前，我们尝试了复现 Guetzli 图像压缩算法，并在开题答辩前在 Visual Studio 2017 中运行了本项目，然后大家了解了算法思路和逻辑后开始分工阅读代码和相关参考资料。此时我们觉得虽然工程量较大，但应该最终可以实现目标功能。

但是我们组的前期调研和论证并不充分。在开题答辩后大家开始工作时，发现 Guetzli 算法的代码遵循 C++11 标准，而 Vivado HLS 2016.4 并不支持较高版本的 C++标准，因为需要修改的错误过多，我们在一段时间的尝试后决定放弃 Guetzli 算法，将目光转向了 Butteraugli 算法。但最终我们发现 Butteraugli 算法虽然相比 Guetzli 算法工作量较少，但仍然不太可能在截止期前完成。大家讨论后决定按照 Guetzli 算法的部分主要逻辑重写代码并进行实现[2]。在查阅相关资料和用 Python 中的 OpenCV 库简单尝试后，我们确定了选题。

所以，我们最终实现的选题是基于 Guetzli 算法思路的一个较简单的图像压缩算法，其大致是从 Guetzli 算法的多条压缩路径中分离出的算法之一，而且整个文件的结构组织较为清晰，论证后大家一致认为可以实现，于是选定了此题目。最终的实现结果证明我们重写的算法可以有效地对图像进行压缩。

1.2 课题背景

图像压缩是指在满足一定图像质量的前提下，用尽可能少的数据量来表示图像的方法，分为无损压缩和有损压缩。由于本次项目研究的是 JPEG 图像压缩，所以此处主要对有损压缩的基本研究背景进行综述。

图像之所以可以被压缩，主要是因为图像中存在大量冗余，例如相邻像素间的相关性引起的空间冗余，不同彩色平面或频谱带的相关性引起的频谱冗余等，而本次实验中优化的冗余主要是视觉冗余，是因为人眼对亮度变化相对敏感，而

对色度变化相对不敏感；在高亮度区对物体边缘相对敏感，对内部区域相对不敏感，因此高频部分一些细微的颜色差别并不容易被肉眼感知，这是我们进行图像压缩的主要依据。

图片压缩在现代生活中很重要。现如今互联网上绝大部分图片都使用了JPEG 压缩技术，也就是大家使用的 jpg 文件。

从 2004 年到 2008 年，出现了关于在不修改表示图像的情况下进一步压缩 JPEG 图像中包含的数据的方法的新研究。这适用于原始图像仅以 JPEG 格式可用，并且需要减小其大小以进行存档或传输的情况。标准的通用压缩工具无法显着压缩 JPEG 文件[3]。通常，此类方案利用了对朴素方案的改进来对 DCT 系数进行编码，该方案未考虑以下因素：

1）同一块中相邻系数的大小之间的相关性；

2）相邻块中相同系数的大小之间的相关性；

3）不同通道中相同系数/块的大小之间的相关性；

采用 DCT 压缩算法实现过程的主要原理：

1）将原始图像分为 8×8 的小块；

2）将图像中每个 8×8 的 block 进行 DCT 变换；

3）量化。

此原理即本次项目中采用的算法原理。其逻辑图如下：

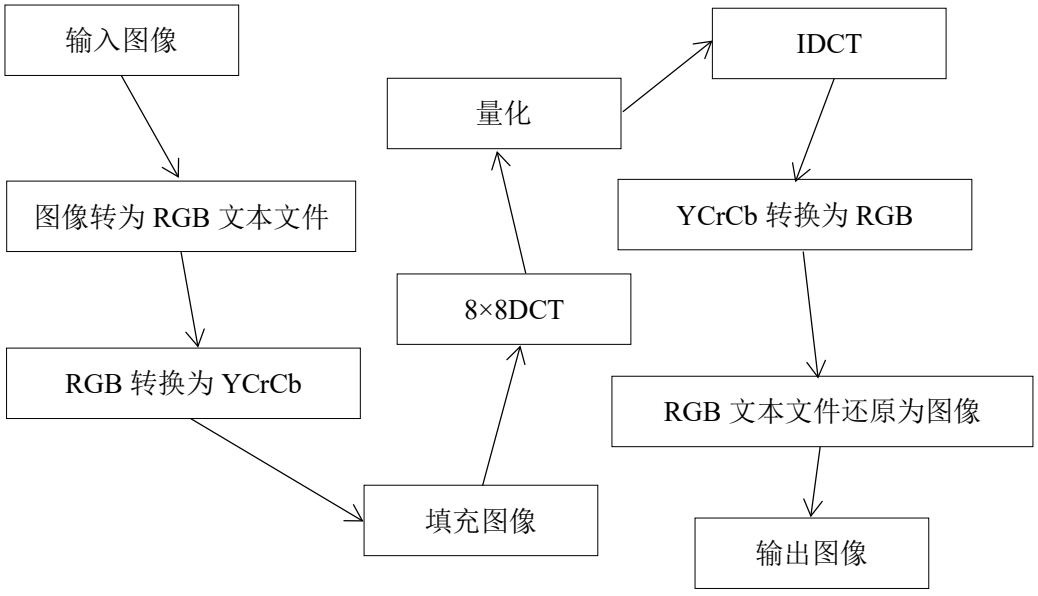

Figure 1 本次项目算法思路[1]

DCT 变换将低频部分集中在每个 8×8 块的左上角，高频部分在右下角，所谓 8×8 块的左上角，高频部分在右下角，所谓 JPEG 的有损压缩，损的是量化过程中的高频部分。因为有这样一个前提：低频部分比高频部分要重要得多，去除 50%的高频信息可能对于编码信息只损失了 5%。所谓量化就是用像素值除以量化表对应值所得的结果。

由于量化表左上角的值较小，右上角的值较大，这样就起到了保持低频分量，抑制高频分量的目的[3]。

而我们此前尝试的 Guetzli 算法是图像压缩算法的一个新的研究方向，其寻求在原有基础上继续压缩图像，根据 Google 的 Butteraugli 项目得到的模型除去图片中肉眼不敏感的部分，对实际图像进行了进一步的有损压缩，但肉眼难以察觉。其算法思路如 Fig.1 所示。但这一项目由于模型并不成熟，算法为了追求运行效果而极大地增加了算法复杂度，目前还处在研究阶段。

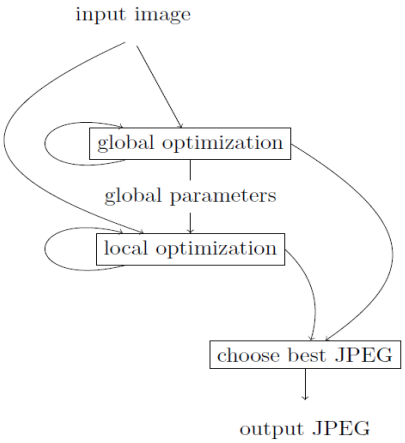

Figure 2 Guetzli **算法思路**[1]

1.3 项目目标与任务分工

本次项目的目标如下：

HLS 实现

- 总体方案描述：编写可以运行和综合的图像压缩算法，并进行优化。
- 工作量预估：理论上可以两个人合作完成，实际 debug 需要共同完成；
- 子任务划分：
  1）总体规划（头文件）；
  2）文件预处理，输入输出；
  3）色域转换；
  4）DCT 和 IDCT，以及相关辅助函数；
  5）testbench；
  6）debug；
  7）HLS Directive 优化。

嵌入式实现

- 总体方案描述：将算法加载到 Zynq 板上并实现提速。
- 工作量预估：理论上一个人可以完成，实际 debug 需要两个人共同完成；
- 子任务划分：
  1）代码修改；
  2）debug。

而本次项目的小组分工如下：

Table 1 项目分工

组员	HLS 任务	嵌入式任务
杨文曦	头文件，文件预处理，色域转换，debug	Debug
罗恬	DCT/IDCT 和辅助函数	代码修改，debug
刘健伟	Testbench，Directive 优化	—

本次项目的架构图如下：

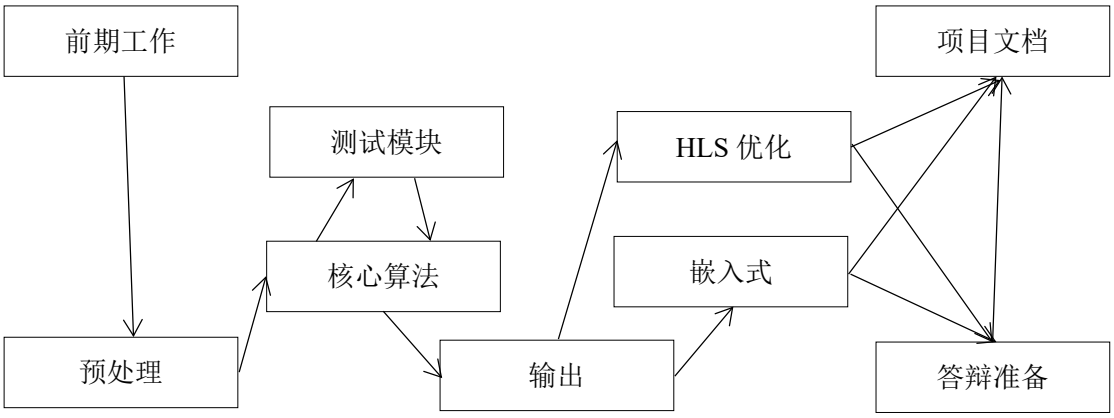

Figure 3 项目架构图

2 实施过程

2.1 个人任务

在本次项目中，我负责的任务主要在算法部分，即编写可以用于 HLS 优化和嵌入式实现的图像压缩算法。具体地，我主要负责理解算法原理并编写相应的可综合的代码，准备答辩材料，并在整个项目进行的过程中协调和组织每个人的工作，具体包括：

1）理解图像压缩算法并将其分割，安排各个组员分工。此部分工作的目标是理解整个算法的原理和代码编写逻辑，并从中拆分出可独立完成的任务分配给各个组员。其目标是能够得到可以综合的图像压缩算法的各个模块，属于任务开始阶段的基础工作，是整个项目的前提；

2）图像的编码解码。这一部分主要体现为一组 python 文件，可以将待处理的图片转化为包含图像每个像素点 RGB 亮度的文本文件，方便读入项目中进行处理。这一部分使用 opencv 进行实现，优势是避免了在项目中使用 jpeg 或 libpng 库，操作更加简单。属于算法的预处理和结果处理工作，得到可以展示的算法实现效果；

3）图像的色彩空间转换。这一部分主要体现为核心算法中的一个 cpp 文件，主要负责输入输出文件在 RGB 色彩空间和 YCbCr 色彩空间之间的转换。转换空间是为了方便进行更大的压缩，因为肉眼对色度的高频变化不敏感，因此对色度分量取平均值可以对图像进行压缩。这一部分对图像进行了一定程度的压缩，更主要的是为后续离散余弦变换(DCT)操作进一步衰减图像高频分量进行准备。

4）将大家的工作综合在一起，确保功能正确。这一部分是整个项目的综合工作之一，确保算法可以正常运行且可以在 Vivado HLS 的环境里进行模拟和综合，

完成后交由组员进行 HLS 优化。

5）准备答辩材料。这一部分主要是准备答辩材料(如总文档，展示材料等)和进行讲解，需要对整个项目有足够的理解。

总的来说，我在本次项目中负责组织项目进行和一些基础但必不可少的算法上的任务。

除此之外，前期对 Guetzli 算法和 Butteraugli 算法的理解和修改也花费了大量时间，虽然没能在最终项目中体现出来，但还是从中学到了很多 Vivado HLS 相关的知识，尤其是各类错误的错误原因和解决方法。

2.2 难点分析

就我个人的完成过程和最终结果来看，我认为个人任务的难点有以下几点：

难点 1：算法的理解

由于在此之前很少接触图像处理的相关算法，所以在本次项目的前期准备时花费了很多时间了解图像格式、色彩空间和图像压缩算法，尤其是在前期尝试 Guetzli 算法和 Butteraugli 算法时。其中较容易理解的是编码解码部分 Huffman 编码的使用，以及零替换 DCT 系数的方法，但是由于其调用了 jpeg 和 libpng 库，而对库中函数的使用并不熟悉，因此我花费了一定时间查阅资料和阅读库代码，最终成功找到了将图像文件用 opencv 转换为文本文件的方式，将核心算法处理的信息转变为了多维数组，避开了核心算法中库的使用。

难点 2：寻求报错的解决办法

这一部分或许是最难的部分，报错主要是在外部环境(例如 Visual Studio 2017，CLion 等)中可以运行的代码在 Vivado HLS 中模拟和综合时遇到的。这一部分花了我和罗恬大概两个通宵……这里列出一些比较有代表性的报错和最终的解决办法；

1）ERROR: [SYNCHK 200-71] function '' has no function body；

解决方案：这个错误是由于调用 C/C++一些库中函数导致的。这些库函数在外部可以正常使用，在 Vivado HLS 中则会出现报错。网络上他人提供的解决办法是添加 extern C 并将命名空间改为 HLS。但是这一解决方案似乎并不适用我的情况，最终我的解决办法是查找该函数在库中定义，在代码中重写了该库函数。对于其他一些函数，只要不包括在综合顶层文件中便不会出现此问题。此类库函数有 allocator ()，readdir ()，csqrt()等；

2）ERROR: [SYNCHK 200-72] unsupported c/c++ library function '';

解决方案：这个错误是由于将一些 C/C++库函数包括在了综合的顶层文件里导致的，将这些函数分离出来即可避免这类错误。此类函数有 fopen()，fwrite()，strcmp()等；

3）ERROR: [SYNCHK 200-74] Recursive functions are not supported；

解决方案：Vivado HLS 不支持递归，因为 FPGA 板上没有支持递归的硬件。改写的办法是用 while 循环和栈操作手动递归。

4）ERROR: [SYNCHK 211-100] Simulation failed: SIGSEGV.

解决方案：之前希望处理的文件太大，导致了栈溢出。多次尝试了图片大小后，我们将图片的尺寸限制在了 400×400 像素以下。

这类报错信息我们遇到了很多。总结起来，主要由以下几种因素导致：

1）Vivado HLS 不支持高版本 C++标准(如 C++11 及以上)；

2）FPGA 板上缺少相关算法的执行硬件(如递归调用)；

3）一些库文件中的函数无法作为顶层函数在 Vivado HLS 中综合(如 strcmp)。

难点 3：对色彩空间的理解和运用

使用 opencv 可以将图片导出为较易编辑的文本文件，其内容为图像中每个像素点的 RGB 亮度信息。但由于人眼容易感受到 RGB 通道，而算法需要对高频的色度进行编辑，所以将色彩空间转换为 YCrCb 更易于后续的压缩[4][5]，可以对色度分量进行平均取值，从而减少其所需的色彩范围，达到压缩目的。转换方式是固定的，这在后续个人实现步骤处会有更为详细的说明。

2.3 设计与实现步骤

2.3.1 任务 1：预处理

预处理部分的流程图如下所示：

这一部分主要由两份 Python 代码实现，需要调用 opencv 库。设计的 RGB 文本文件格式为：

第一行：输入图像的宽与高像素信息；

第 u×v 行：图像中坐标为(u, v)的像素点的 RGB 信息，每一行为 3 个 0-255

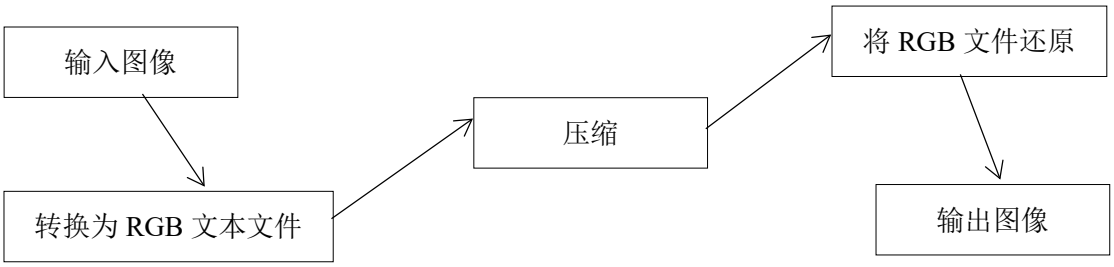

Figure 4 预处理流程图

的数值，分别代表红色，绿色，蓝色亮度，数值间以空格分隔。

而将图像转换为 RGB 文本文件的算法伪代码如下：

Table 2 jpg to dat 伪代码

```
jpg_to_dat
输入：jpg 文件
输出：dat 文件
read jpg file
get u,v,rgb by opencv  // 使用 opencv 库读取 jpg 文件
create dat file
write u, v at line 1 of dat file
for i←1 to u do  // 遍历宽
  for j←1 to v do  // 遍历高
    write rgb[i][j][1,2,3] to dat file  // 写入 RGB 亮度
    write \n
```

而将 RGB 文本文件还原为图像的算法伪代码如下：

Table 3 dat to jpg 伪代码

```
dat_to_jpg
输入：dat 文件
输出：jpg 文件
read png file
read u, v at line 1 of dat file  // 读取图片宽和高
create jpg file
for i←1 to u do  // 遍历宽
  for j←1 to v do  // 遍历高
    read r,g,b value  // 读取 RGB 亮度
    put pixel(u,v) by rgb values  // 画出此像素
```

最终实现效果参见 3.1 节。

2.3.2 任务 2：色域转换

色域转换部分的流程图如下所示：

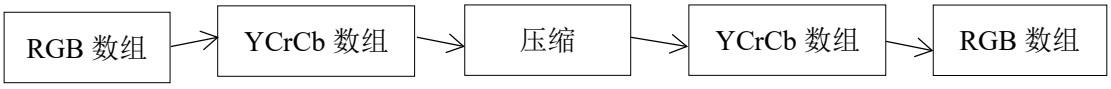

Figure 5 色域转换流程图

这一部分主要由核心算法中的一个函数实现。RGB 色彩空间到 YCbCr 色彩空间的转换公式如下[5]：

$$\begin{cases} Y = 16 + \dfrac{65.738R}{256} + \dfrac{129.057G}{256} + \dfrac{25.064B}{256} \\ C_B = 128 - \dfrac{37.945R}{256} - \dfrac{74.494G}{256} + \dfrac{112.439B}{256} \\ C_R = 128 + \dfrac{112.439R}{256} - \dfrac{94.154G}{256} - \dfrac{18.285B}{256} \end{cases}$$

而算法的伪代码如下：

Table 4 color convert 伪代码

```
color_convert()
输入：输入图像信息数组 image[u][v][3],标识符 flag
输出：输出图像信息数组 image[u][v][3]
for i←1 to u do  // 遍历宽
  for j←1 to v do  // 遍历高
    if flag is 1 then  // 从 RGB 转 YCrCb
      calculate Y, Cr, Cb  // 用转换公式计算
      for val in Y, Cr, Cb do
        if val < 0 then  // 除去高频信息
          val = 0
        else if val > 0 then
          val = 255
        store val in image[][][val]
    else do    // 从 YCrCb 转 RGB
      calculate R, G, B  // 用转换公式计算
      for val in R, G, B do
        if val < 0 then  // 除去高频信息
          val = 0
        else if val > 0 then
          val = 255
        store val in image[][][val]
```

最终实现效果参见 3.1 节。

3 结果分析

3.1 测试结果

任务一预处理测试结果如下：

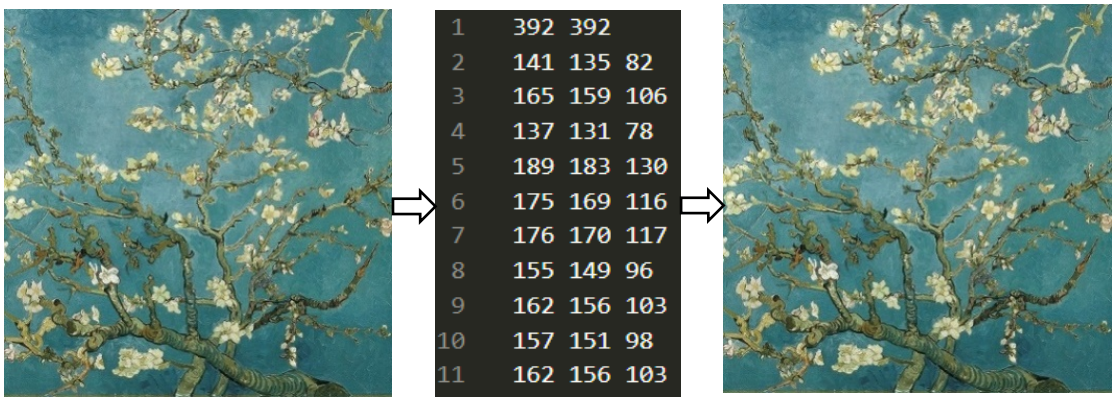

Figure 6 预处理测试结果

其中，左侧为输入文件；中间为用 jpg_to_dat 算法转换得到的 dat 文件，第一行为像素参数，其余行为各个像素点 RGB 的亮度；右侧为用 dat_to_jpg 算法还原得到的输出文件。可以看出，图像可以完全还原，没有任何差别，说明此项功能正确。

任务二色域转换由于只是核心算法中的一部分，单独测试无法看到效果，故此处给出最终的算法测试效果，如下所示：

Figure 7 集成测试结果

其中，左侧为输入文件，中间为质量系数设为 80 时的输出图像，右侧为质量系数设为 60 时的压缩图像。文件大小如下表所示：

Table 5 集成测试结果

输入图像	Q=80	Q=60
87.9KB	27.0KB	12.1KB

可以看出在文件大小有明显的压缩效果，压缩比可以达到 7：1。而图像质量虽然不能量化，但放大后可以看出压缩后的图像质量有明显下降，振铃效应比较

明显，尤其是质量系数较低时的压缩图像肉眼可见有明显分块，说明在图像质量上压缩效果并不理想。具体原因会在 3.2 节讨论。

但是由测试结果可以看出，压缩算法可以正常运行，因此色域转换模块可以正常运行。

3.2 不足与展望

首先要讨论的是个人任务的不足之处。个人任务运用了外部的 opencv 库在 Python 中实现图像预处理，因此每次运行图像压缩算法需要启动两个 Python 文件和一个 testbench.cpp 文件。更理想的解决方案是使用 jpeg 库直接用 C++实现图像读取和输出，但这一部分由于能力限制未能实现。一种可替代的解决方案是编写一个更顶层的 Python 文件，并通过此文件去依次运行压缩算法的三个文件。

而更主要的是整个项目中的不足之处。这一部分较多，以下分点列写：

1）压缩效果不理想。从上面的集成测试结果可以看出，虽然图像可以被压缩到非常小(就集成测试结果来看，压缩比为 7:1，而在质量参数更低的情况下可以达到 15:1)，但压缩质量较差，不能与流行的图像压缩算法相比较，振铃效应比较明显。究其原因，我们认为有两个方面：一是我们的图像分块做得不够小，导致最后的压缩图像分块肉眼可见；二是我们处理的文件本身像素就不够高，压缩后效果只会更差；

2）输入图像大小有限制。这一点是受限于上述 2.2 节难点 2 的第 4 条，图像过大的时候，我们用于存储图像信息的数组 image[][][3]就会占用较大的内存空间，最终导致栈溢出。多次测试后我们将图片大小上限限制在了 400×400 像素。这一点算法上的可行解决办法是将其放到堆空间而不是栈空间上进行处理，必要时申请虚拟内存；

3）没能将浮点数改为定点数。这一部分本来是分配给其他组员的任务，但组员因为其他部分任务开发进度问题，最终没有如期完成浮点数改定点数。如果此项任务实现，HLS 综合可以实现更大程度的优化。

4）没能完成嵌入式实现。这一部分虽然是分配给其他组员的任务，但组员因为前段时间患传染性疾病被学校要求隔离，在小组成员的沟通和 FPGA 硬件开发板的使用上出现了障碍，因此组内交换了一些任务，将需要开发板的嵌入式实现交给了另一个组员，但该组员的嵌入式开发遇到了一些问题。在迟迟没有进展的情况下，大家都一定程度上参与了嵌入式开发。但是由于 IP 核的一些问题，以及修改的代码在监视循环和监测时间的函数调用上存在一些无法解决的问题，在截至日期前没能完成实现。

而展望的话，首先是希望能够完成嵌入式开发，证明优化后在 FPGA 上运行可以达到更好的效果；其次是希望优化算法，比如解决第 2 点中的输入图像大小限制，解决第 1 点中的图像压缩质量问题，然后将操作简单化。但总的说来，这次项目还是加深了我们小组的成员对图像存储原理，图像压缩算法和 Vivado HLS 优化，嵌入式开发等方面的知识的理解。

参考文献

[1]. Alakuijala, Jyrki, et al. "Guetzli: Perceptually guided jpeg encoder." arXiv preprint arXiv: 1703.04421 (2017).

[2]. Fu, Sizhe, Ping Shi, and Da Pan. "A modified algorithm of Guetzli encoder." 2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC). IEEE, 2018.

[3]. Hudson, Graham, et al. "JPEG-1 standard 25 years: past, present, and future reasons for a success." Journal of Electronic Imaging 27.4 (2018): 040901.

[4]. RGB, https://en.wikipedia.org/wiki/RGB

[5]. YCbCr, https://en.wikipedia.org/wiki/YCbCr