****

*硬件设计课程设计实验报告*

**基于FPGA的图像边缘检测及与CPU的对比**

姓 名：肖子原

学 号：1120151861

组 员：朱威浦，沈瀚文，肖子原，廖汉龙

学 院：计算机学院

邮 箱：[liamliaohl@gmail.com](mailto:liamliaohl@gmail.com)

2018年5月14日 星期一

目录

[一、实验名称 3](#_Toc516394976)

[二、实验目的 3](#_Toc516394977)

[三、实验原理 3](#_Toc516394978)

[3.1 边缘检测 3](#_Toc516394979)

[3.2 Laplace算子 3](#_Toc516394980)

[3.3 拉普拉斯滤波原理 4](#_Toc516394981)

[四、实验环境与配置 4](#_Toc516394982)

[五、实验内容 5](#_Toc516394983)

[5.1 设计框架 5](#_Toc516394984)

[5.2 实验步骤与过程 8](#_Toc516394985)

[六、实验结果与对比 12](#_Toc516394986)

[6.1 实验结果 12](#_Toc516394987)

[6.2 实验对比 12](#_Toc516394988)

[6.3 处理速度对比 13](#_Toc516394989)

[七、遇到的问题及解决 14](#_Toc516394990)

[八、分工与学习过程 14](#_Toc516394991)

[九、实验心得感悟 14](#_Toc516394992)

## 一、实验名称

基于FPGA的图像边缘检测及与CPU运行效果检测

## 二、实验目的

通过使用硬件描述语言Verilog实现对图片的边缘检测，掌握FPGA的在视频图像处理中的作用并了解FPGA的并行计算特点。通过与直接用CPU的进行边缘检测的处理效果进行比对，了解FPGA在并行计算与串行计算的优缺点。

## 三、实验原理

### 3.1 边缘检测

边缘是图像的最基本特征,是图像局部亮度变化最显著的部分，是物体的轮廓或物体不同表面之间的交界在图像中的反映，也是图像分割、纹理特征提取和形状提取等图像分析的重要基础.。边缘检测是利用物体和背景在某种图像特性上的差异来实现，这些差异包括灰度、颜色或者纹理特征。在实时图像处理中，由于实现边缘检测算法涉及的运算数据量很大，一般运算器的顺序执行或用传统的软件方法都很难达到实时性要求.，而采用FPGA的并行结构和流水线技术，可提高图像边缘检测的处理速度,满足系统实时性要求。

### 3.2 Laplace算子

Laplace算子作为一种优秀的边缘检测算子，在边缘检测中得到了广泛的应用。该方法通过对图像求图像的二阶倒数的零交叉点来实现边缘的检测，公式表示如下：



由于Laplace算子是通过对图像进行微分操作实现边缘检测的，所以对离散点和噪声比较敏感。于是，首先对图像进行高斯卷积滤波进行降噪处理，再采用Laplace算子进行边缘检测，就可以提高算子对噪声和离散点的鲁棒性，如此，拉普拉斯高斯算子Log（Laplace of Gaussian）就诞生了。

高斯卷积（Gaussian convolution ），高斯函数的表达式如下：



在实际运用中，常常使用离散的逼近方式来进行卷积操作。2D高斯拉普拉斯算子可以通过任何一个方形核进行逼近，只要保证该核的所有元素的和或均值为0，如下一个3×3的核进行逼近：



矩阵的卷积操作在图像处理中实际上分两步进行，首先对图像灰度矩阵该采用预定义的3×3窗口做矩阵乘法，完成一次矩阵乘法后滑动窗口再进行乘法，迭代直至整个图片处理完成。卷积运算的核心操作如下：



基于矩阵乘法运算的实质，以上运算可以整理为：



### 3.3 拉普拉斯滤波原理

中值滤波原理在实时图像采集中，不可避免的会引入噪声，尤其是干扰脉冲和椒盐类噪声。噪声的存在严重影响边缘检测的效果。中值滤波是一种基于排序统计理论的非线性平滑技术，能有效平滑噪声，且能有效保护图像的边缘信息，所以被广泛用于数字图像的边缘提取，其基本原理是把数字图像或数字序列中一点的值用该点一个邻域内各点的中值来代替。对二维中值滤波，其公式表示如下 ：

其中分别为原始图像和输出图像，W为二维窗口，大小为，(n一般为奇数)，可以是３×３或５×５等，常用的形状有方形、圆形和十字形等。

## 四、实验环境与配置

|  |  |
| --- | --- |
| 实验硬件开发板 | Intel Altera Dei2-150 |
| 操作系统 | yocto |
| 处理器 | Atom |
| RAM | 8G |
| 开发平台 | Quartus II 13.0 (64-bit) Web Edition |

表1

|  |  |
| --- | --- |
| 实验主机 | 惠普暗影精灵2pro |
| 操作系统 | Win10 家庭中文版 |
| 处理器 | Intel(R) Core(TM) i5-7300HQ |
| RAM | 8G |
| 开发平台 | Visual Studio 2017 Community |

表2

## 五、实验内容

本实验通过使用DE2i-150实验板板载FPGA定制内核和Atom通用CPU执行简单的边缘检测算法。使用高斯拉普拉斯算子（LoG）（本实验的主要图像处理IP，本质上是一种简单的矩阵卷积）计算原始图像像素的单个边缘强度值，然后成为新的边缘检测图像中的像素。将灰度.bmp文件用作演示的输入，输出对应的仅具有边缘.bmp图像文件。同时通过自配笔记本CPU用Canny算子对同一图片进行边缘检测，对比效果图。

### 5.1 设计框架

框架主要包含硬件和软件两大模块，这里的硬件和软件不是传统意义上的硬件和软件。硬件模块主要包含用于下板FPGA的verilog文件，软件模块则是在Atom系统上跑的C程序源文件等。其中软件模块处于外层，硬件模块处于内层。它们和用户的关系如下图所示。用户通过软件模块输入待处理的图片文件名，然后硬件模块进行相应的图像处理，软件模块再读取处理结果返还给用户。软件模块和硬件模块之间的交互通过内存来完成，软件模块和硬件模块都会对内存进行操作。

用户

软件模块

硬件模块

图1

整个框架的流程图如下，其中详细拆分了软件模块里的几个函数，硬件模块则没有细分出来。其具体内容见后面的硬件模块代码说明。

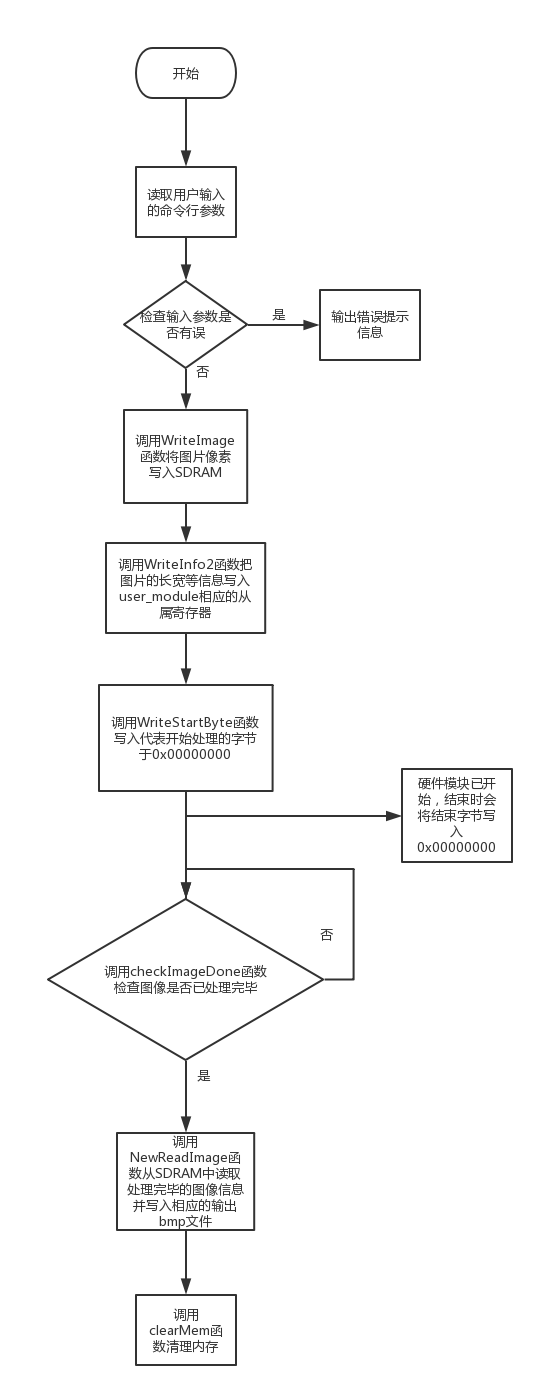


图2

此边缘检测应用程序中有一个主要项目特定块：用户模块。所有算术计算均在该块中编程。用户模块是设计的“大脑”，控制顶层设计来执行某些操作和计算。它具有作为开始/停止字节和图像信息寄存器的从属读/写块。用户模块也负责启动SDRAM的读/写操作。对于在Atom上运行的软件，处理器通过PCIe总线进行内存读取和写入。PCIe和Avalon与提供的桥接模块进行通信。

### 5.2 实验步骤与过程

#### 5.2.1 设备启动

检查板上的所有连接，确保将以太网，鼠标，键盘，VGA监视器和USB Blaster连接到PC，打开DE2i150电路板，等待yoctor操作系统启动。

#### 5.2.2 工程创建

启动quartus，新建名为edgedet的工程项目，按照预期功能编写项目，搭建SOC。此边缘检测应用程序中有一个主要项目特定块：用户模块。所有算术计算均在该块中编程，控制顶层设计来执行某些操作和计算。它具有作为开始/停止字节和图像信息寄存器的从属读/写块。用户模块也负责启动SDRAM的读/写操作。对于在Atom上运行的软件，处理器通过PCIe总线进行内存读取和写入。PCIe和Avalon与提供的桥接模块进行通信，如图3所示：

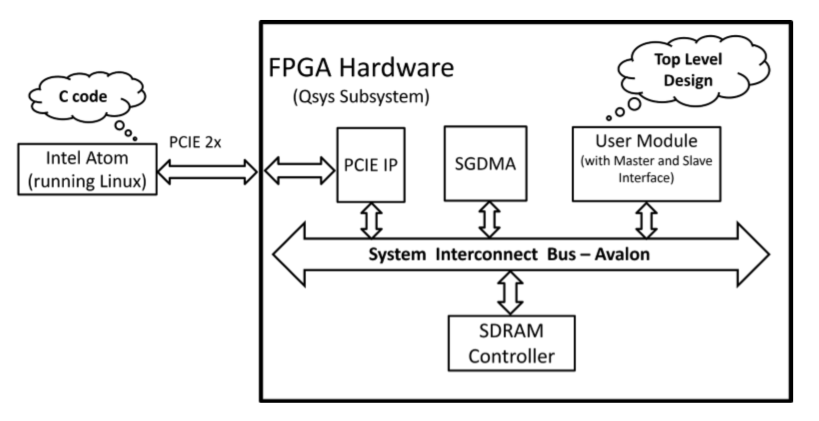


图3

#### 5.2.3 硬件模块代码说明

工程中各文件的实现功能如下（代码见附件）：

|  |  |
| --- | --- |
| 文件名称 | 文件功能 |
| arit.v | 输入为9个像素点的值，对其卷积运算，输出结果。 |
| ImageSpecRegs.v | 存储图像的维数，并为矩阵的中行和底行提供基地址 |
| readCounter.v | 对当前处理的像素的列和行计数 |
| shift\_register.v | 通过移位方式将9个像素点的值分别存放在寄存器中 |
| outputlogic.v | 输出卷积运算结果 |
| top\_levelu.v | 例化arit、imagespecregs、readcounter、shift\_register、outputlogic模块 |
| user\_module.v | 从上位机接收来自PCIE IP的输入命令，还包括整个边缘检测过程的主要状态机。 |

表3

对arit.v文件做重点说明，该文件以72位的图片信息作为输入，该输入代表了9个8位的像素点，完成计算后将一个8位的结果输出。为了防止溢出，我们计算过程中的所有中间数据都以10位存放，我们以输入数据的第32位到第39位作为pixel4,将它左移两位（相当于乘4）后作为pixel4\*4，将第8位到第15位作为pixel1，并取反，同理可以计算出pixel3，pixel5，pixel7，注意最后还需要在前面补全符号位凑足10位。

得到了以上几部分之后累加即可得到一次矩阵乘法运算的结果，完成这些操作的verilog代码如下所示：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | assign ext\_reg\_4 = {1'b0,in\_pixels[39:32],2'b00};  // pixel4 \* 4  assign ext\_reg\_1 = {3'b111, (~in\_pixels[15:8])};  assign ext\_reg\_3 = {3'b111,(~in\_pixels[31:24])};  assign ext\_reg\_5 = {3'b111,(~in\_pixels[47:40])};  assign ext\_reg\_7 = {3'b111,(~in\_pixels[63:56])};  // pixel1 \* -1    //Matrix Convolution  always @(neg\_reg\_1,neg\_reg\_3,ext\_reg\_4,neg\_reg\_5,neg\_reg\_7)  begin  w\_out\_p1 = neg\_reg\_1 + neg\_reg\_3;  w\_out\_p2 = neg\_reg\_5 + neg\_reg\_7;  w\_out = w\_out\_p1 + w\_out\_p2 + ext\_reg\_4;  // pixel1\*4-(pixel1+pixle3……)  end |

#### 5.2.4 工程综合下板

综合分析项目并生成sof文件，通过Hardware Manager将此文件烧录至FPGA中，如图4所示：

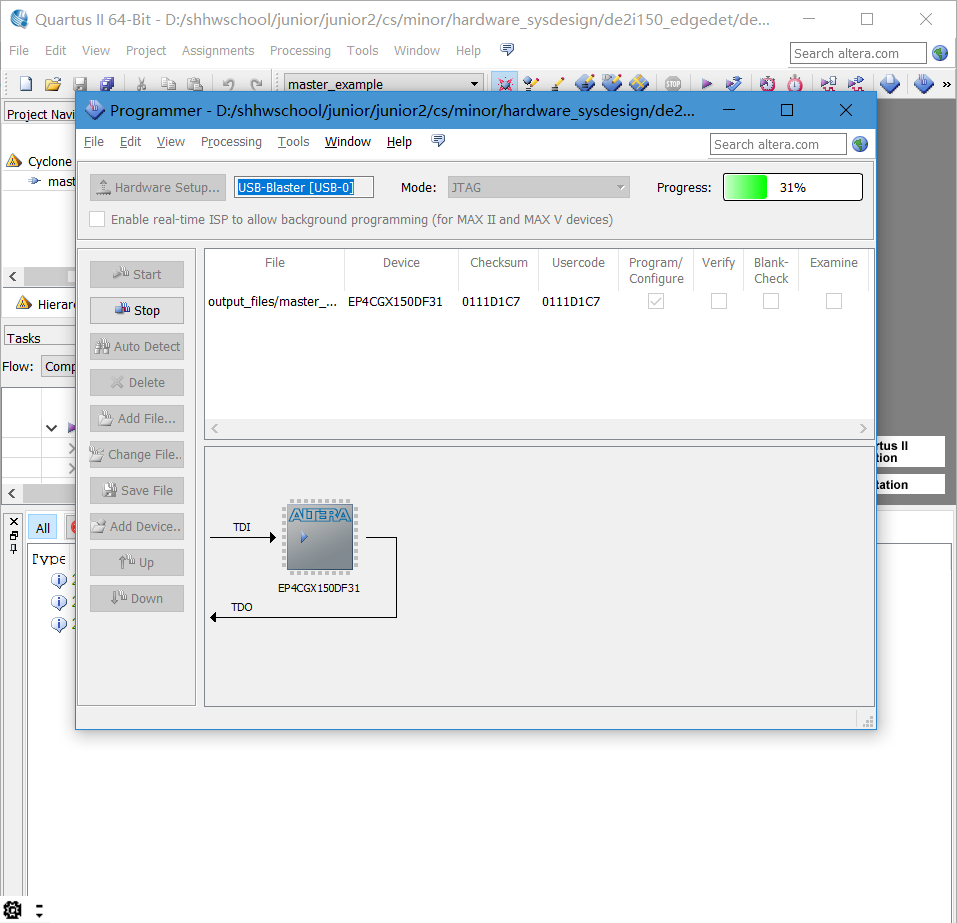


图4

#### 5.2.5 加载设计

启动yoctor操作系统后，进入驱动程序所在目录，输入指令

sh ./ load\_terasic\_qsys\_pcie\_driver.sh

安装PCIe驱动程序，如图5所示：

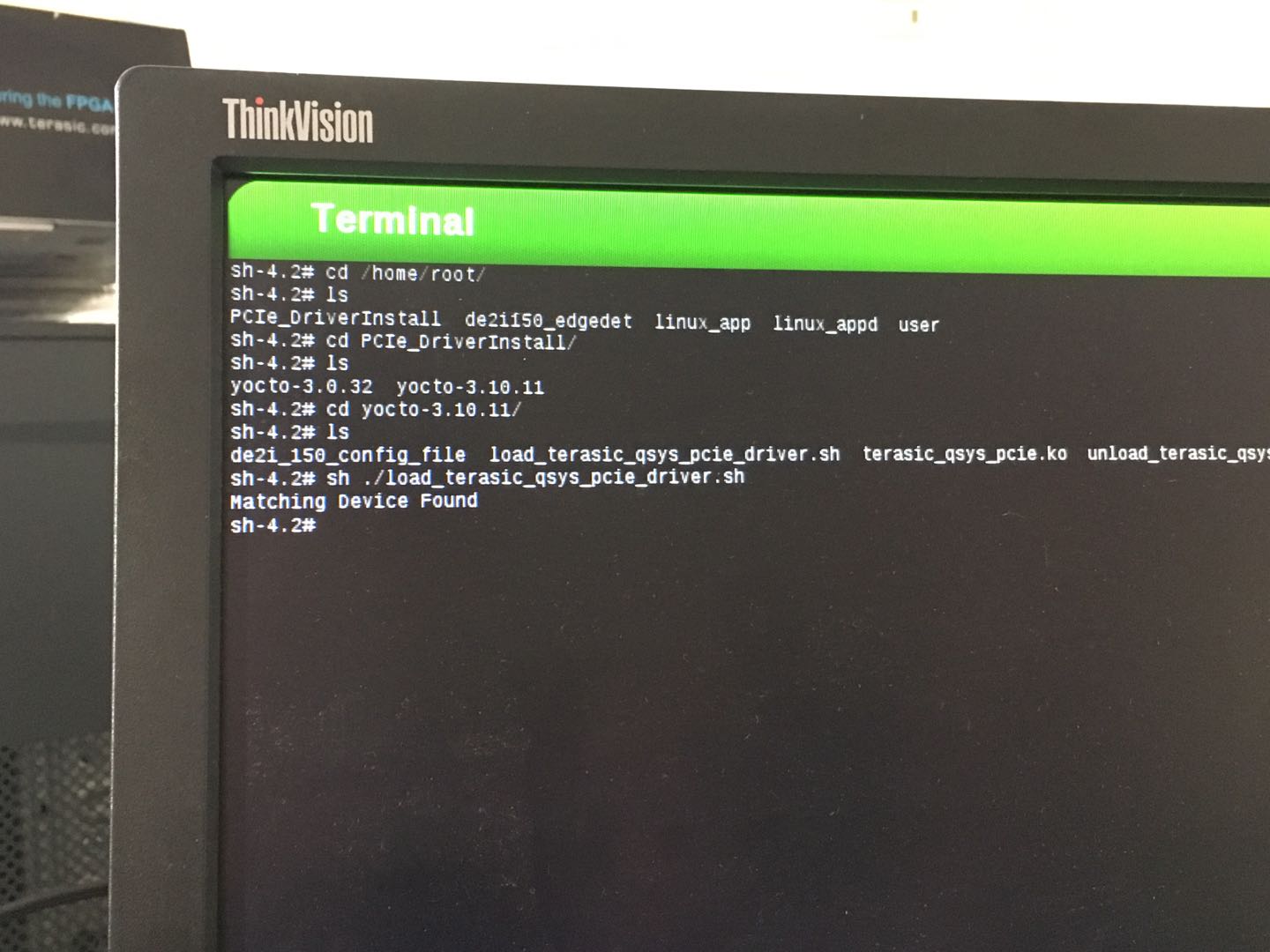


图5

安装完成后，重启操作系统。

系统启动后，使用从开源社区的app进行PCIec传输。进入app所在文件夹，通过以下命令编译：

make clean

make

对应生成app即为传输用的可执行文件，如图6所示：

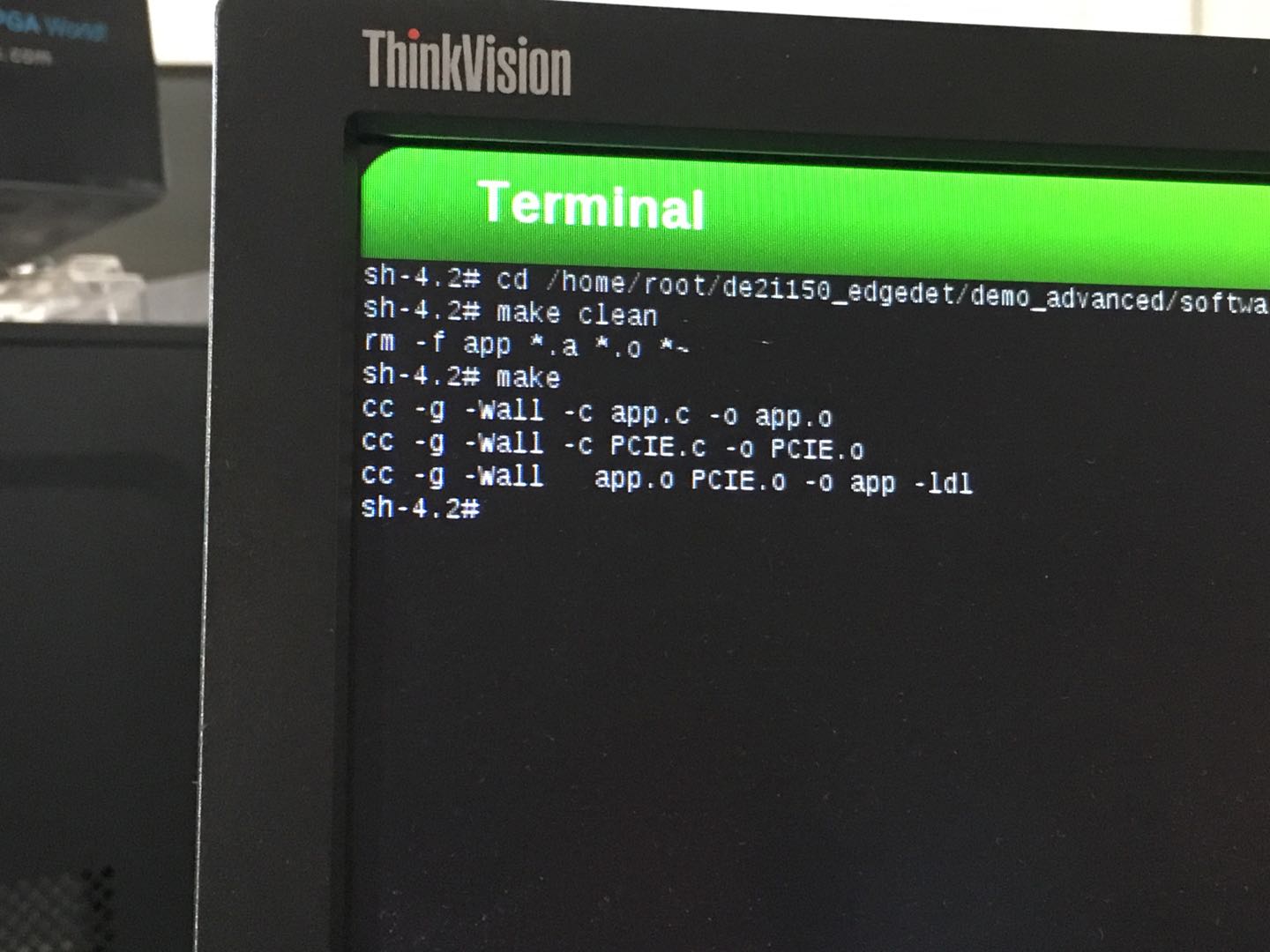


图6

将需要进行边缘检测的图片放置app根目录中，以dancing.bmp图片为例，输入如下指令

./app -d dancing.bmp

如图7所示：

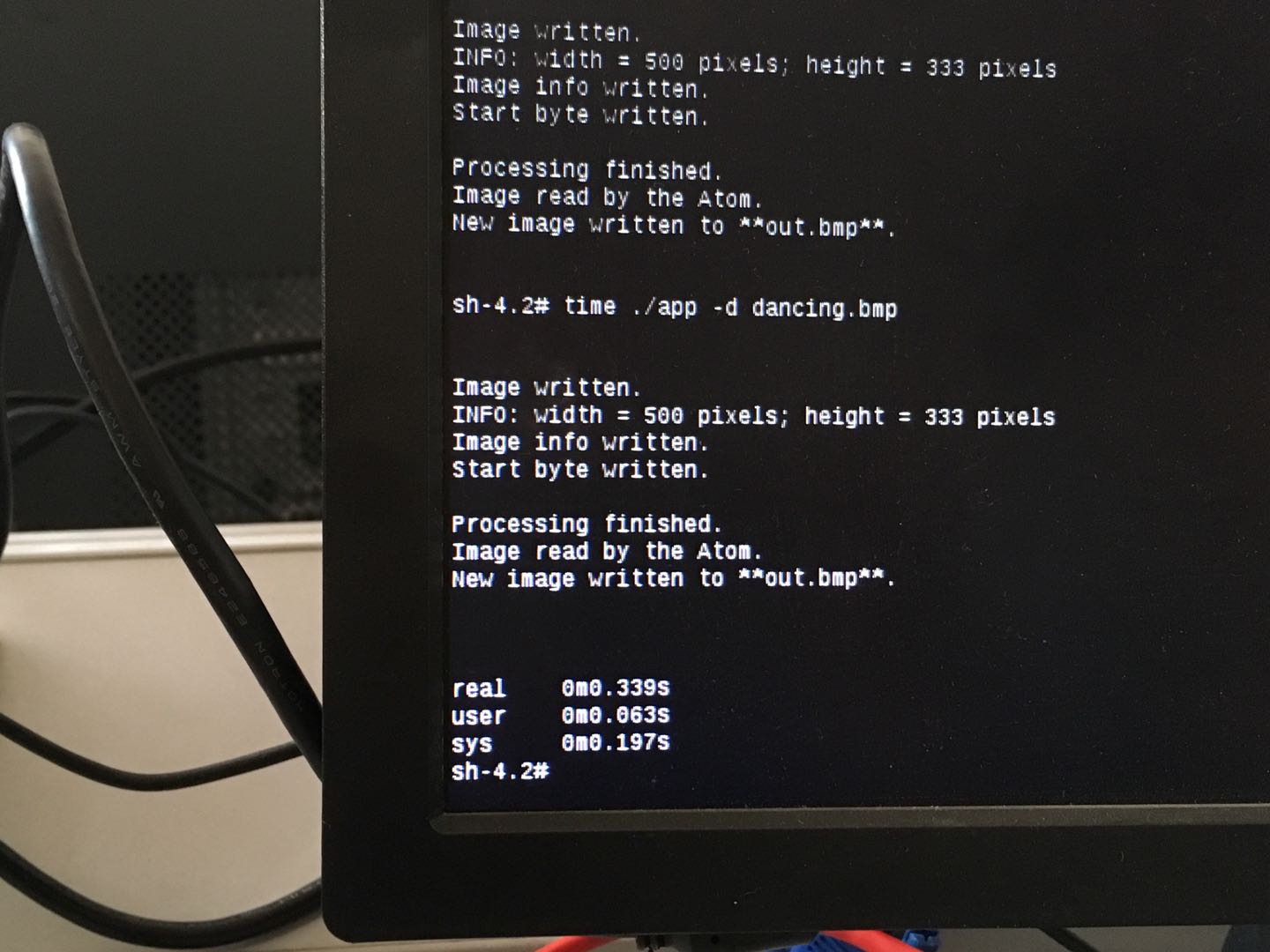


图7

## 六、实验结果与对比

### 6.1 实验结果

原始图片和处理后图片的对比如图8所示：

图8

### 6.2 实验对比

为了更加直观的了解FPGA对于图像处理的优势，我们采用自配置笔记本CPU通过Canny算法对同一张图片进行处理，FPGA处理与CPU处理图片的对比如图9所示：

 FPGA处理效果图 笔记本CPU用laplacian算子处理效果图

图9

从两个图的对比可以看出，尽管OpenCV自带的Laplacian算子的3\*3卷积核与FPGA采取的卷积核是一样的，处理出的结果却有明显不同。CPU中的结果保留了过多的细节。这里面的原因有很多，比如OpenCV自带的laplacian算子在深度为1时的具体卷积操作是怎样的，我们无从得知。同时，CPU上在不同类型的图片类型转换的时候，可能像素值发生了一些变化。所以不能说FPGA的效果就更好，只能说FPGA完全可以实现比较理想的边缘检测效果。另外值得说明的是，我们的程序只接受灰度图。

### 6.3 处理速度对比

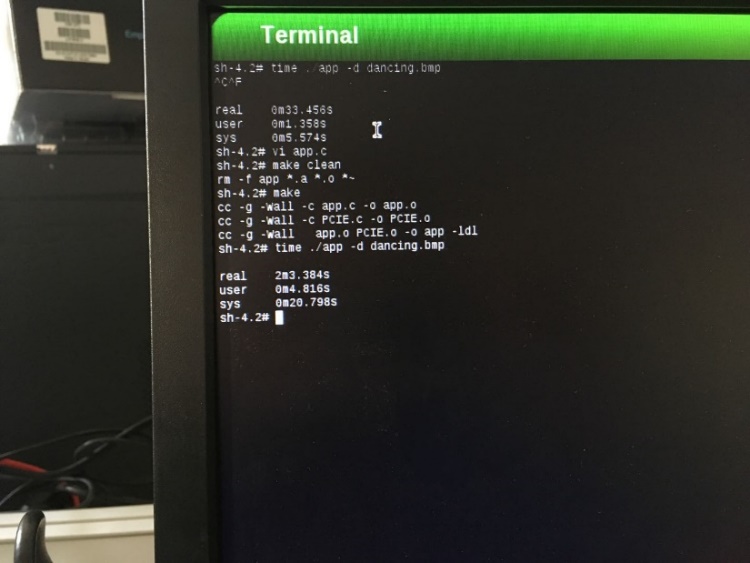
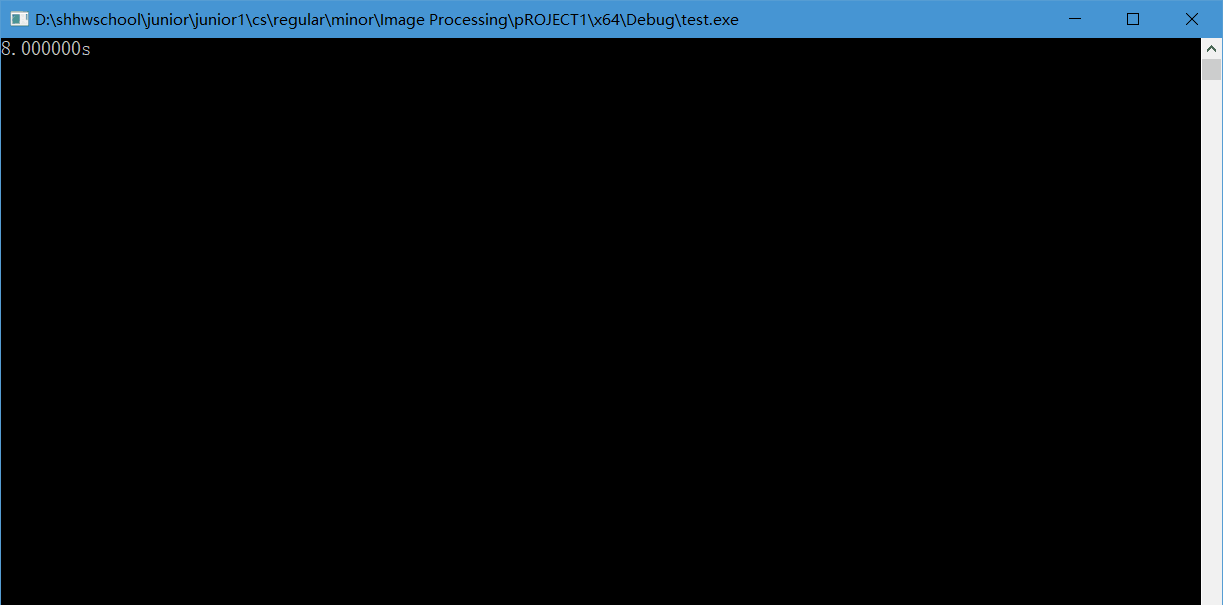
 

图 10

这是处理1000次的时间对比图。可以看到，FPGA用时大约是4秒，而CPU的用时是8秒，而且这还是在CPU利用了多核优化循环的情况下得出的结果。当然这个时间里面有诸多不确定的因素，比如DE2i-150板子的访存时间必然与普通笔记本电脑访存时间有所区别；同时DE2i-150的软件模块和硬件模块都得访存，而笔记本电脑只需要CPU访存；另外两者用的算法步骤不一。FPGA的用时大概是普通笔记本CPU的一半，这能充分显示出FPGA并行运算的强大之处。我们小组之后要研究和解决的问题就是如何进一步控制变量，并对一些变量进行进一步的量化分析等。

## 七、遇到的问题及解决

我们遇到的一个比较重大的问题就是程序跑不起来，其原因有：项目sof和PCIE驱动sof的下板顺序颠倒；下板后没有及时reboot。最终我们得到的跑程序的步骤是：先下整个项目的sof，再下PCIE驱动的sof，当然两次下板后都得及时重启。最后在yocto的终端上运行指定的命令。

## 八、分工与学习过程

本次实验，我主要是对边缘检测算法的部分做了理解，对相应的代码作出了修改与理解，并且完成了相应部分的实验报告。

## 九、实验心得感悟

1)从OPENCL的尝试中得到的经验

在实验三进行选题时我们组对OPENCL在FPGA上的应用产生了极大的兴趣。OPENCL把以往用在CPU上的并行编程方法移植到了FPGA上，使得不熟悉的FPGA的传统程序员也能在FPGA上编写出高效的并行程序。因为之前在学习最优化方法课程的时候，编写了一个遗传算法的程序，遗传算法当中存在大量的并行操作，当时就想用并行来做优化。所以这次在选题时我们就想做一个OPENCL下FPGA的并行遗传算法和传统遗传算法的对比。

为此我们学习了OPENCL的基本知识，下载了各种各样的工具和开发环境。但正如老师所说的我们面临的最大问题可能不是编程的问题而是搭建开发环境的问题，确实虽然我们进行了大量的尝试，但最后发现我们安装的是破解版的quartus软件，一个license始终获取，因此不得不放弃OPENCL的尝试。

(2)算法硬件实现和软件实现的不同

在OPENCL的尝试受到挫折之后，我们并没有放弃做FPGA并行计算和传统CPU对比的想法，我们组的一个同学之前修过图像处理，他提出了可以尝试对比CPU下的图像边沿检测和FPGA下并行的边沿检测。最终通过对比，我们明显地感受到了FPGA在执行时间上的优异性，可见FPGA对于并行计算有着较大的优势。

之前在一篇博客中曾经看到，一个计算器件越是通用，执行某种特定计算的能力就越弱，这一点在我们这学期的实验得到了很好的验证。CPU可以进行各种各样的计算，功能好比瑞士军刀，但是对于某种特定的计算而言，其他的功能是用不到的，而FPGA的设计理念则是允许开发者专注于某一种特点的计算，这样在节约大量硬件的同时，同时也降低了成本和能耗。因此，我觉得FPGA有着广泛的运用前景。

(3)我对于FPGA未来学习的计划

一个学期的硬件课程结束了，但是我还有一些意犹未尽，一方面我们对于Verilog语言的掌握还不够熟练，一方面这次由于条件限制没能完成OPENCL的实验多少有些遗憾。在实验的过程中，我也充分地感受到了计算机硬件的有趣，我本人在毕业后可能会到中科院计算所的硬件方向继续深造，因此在未来我对于FPGA的学习之路可能才刚刚开始。希望利用假期的机会巩固一下Verilog语言和学习一些更加深入的FPGA知识。