Simple ROM 实验报告

刘群*

地球系统科学研究中心

2015 年 5 月 15 日

FPGA (Field Programmable Gate Arrays) 是一种以数据流为编程方式的可编程芯片，在现在的 Mexeler 的平台上主要要编写 3 个函数，包括 CPU 的程序，Kernel 的程序和 Manager 的程序。

本次作业是要实现 FPGA 中的输入输出的功能，主要是 ROM (Read-Only Memory) 的编写。ROM 适用于存储小规模的数组，在 FPGA 中是通过 BRAM 实现的。在 ROM 的实现中，一个是要把数据先写入到 ROM 中，然后是通过地址对 ROM 进行访问，读取里面的内容。本次具体任务是完成对一幅大小为 256×256 的图像的处理将图像处理, 我们要将用到的的系数存到 ROM 中，然后实现对 ROM 的访问，从中读取数据，完成对图像的处理。具体的系数表达式如下：

$$c_i = \begin{cases} 1.0 - \dfrac{1.0}{X/2} \times (\dfrac{X}{2} - i) & \text{if } i < \dfrac{X}{2} \\ 1.0 - \dfrac{1.0}{X/2} \times (i - \dfrac{X}{2}) & \text{if } i \geq \dfrac{X}{2} \end{cases}$$

其中，c_i 是每行图像中第 i 个像素点的系数，X 为图像的宽度, 即图像的列数。

1 Kernel 程序实现 1

(这个 Kernel 程序在服务器上的路径为：/home/hpc03/liuqun_workspace/tutorial-chap12-exercise1-roms/EngineCode/src/romexercise/RomExerciseKernel.maxj)

对于这个问题，我的第一个想法就是要将所有的系数全部存下来，这时候就要特别注意如何对每个像素点的三个 RGB 的值乘以相同的系数, 同时，还需注意的是，实际上，每行对应位置上所乘的系数也是相同的，这样都存下来实际上造成了很大的浪费 (后面将对这个进行改进)。下面介绍一下具体的 Kernel 程序实现。

由于实际的图像在内存中是按照每行的像素来存储的一维数组，每个像素点由三个 RGB 的值构成，因此在处理的时候，我采用对索引标号对 3 取整，对图像宽度 256 取余的方法来得到每个像素点在每行中是属于哪一列。

```
 // 图像宽度
int X = 256;
// 定义内存大小
final int dataSize = X*X*3;

// Define and calculate the coefficients table
double contents[] = new double[dataSize];
//对索引标号对3取整，对图像宽度256取余的方法来得到每个像素点在每行中的列数。
for (int i = 0; i < dataSize; i ++)
{
    if (i/3%X < X/2)
```

*电子邮件: liu-q14@mails.tsinghua.edu.cn，学号: 2014211591

```
        contents[i]=1.0-1.0/(X/2)*(X/2-i/3%X);
    else
        contents[i]=1.0-1.0/(X/2)*(i/3%X-X/2);
}
```

接下来，就是将所定义的数组写入到 ROM 中，

```
// Define ROM and write the coefficients table to the ROM
Memory<DFEVar> table = mem.alloc(dfeFloat(8,24), dataSize);
table.setContents(contents);
```

然后定义一个一维的计数器获得 ROM 的地址，从中读取相应的系数值，与输入的图像像素点相乘，得到新的像素，在这里特别要注意的是数据转化的问题，由于 ROM 中存放的数据是 float 格式，图像像素点是无符号整型数据，因此先将图像数据转换成 float，与系数相乘得到新的结果后再转换成无符号整型数据。

```
CounterChain chain = control.count.makeCounterChain();
DFEVar n = chain.addCounter(X*X*3, 1);

// Input
DFEVar inImage = io.input("imageInput", dfeUInt(32));
DFEVar coef = table.read(n);
inImage = inImage.cast(dfeFloat(8,24)) * coef;
inImage = inImage.cast(dfeUInt(32));
```

最后就是将计算结果输出。

```
// Output
io.output("imageOutput", inImage, dfeUInt(32));
```

2 Kernel 程序实现 2

(这个 Kernel 程序在服务器上的路径为：/home/hpc03/liuqun_workspace/tutorial-chap12-exercise1-roms_loop/EngineCode/src/romexercise/RomExerciseKernel.maxj)

这里的改进主要是针对前一个程序的 ROM 占用的空间太大，而且里面的系数有太多相同的值，完全没有必要全部存下来，因此做如下改进。

首先就是对要存放的数组的改进，只需存放大小为图像宽度的数据即可.

```
int X = 256;
// Define and calculate the coefficients table
double contents[] = new double[X];
for (int i = 0; i < X; i ++)
{
    if (i < X/2)
        contents[i]=1.0-1.0/(X/2)*(X/2-i);
    else
        contents[i]=1.0-1.0/(X/2)*(i-X/2);
}
// Define ROM and write the coefficients table to the ROM
Memory<DFEVar> table = mem.alloc(dfeFloat(8,24), X);
table.setContents(contents);
```

然后我通过引入三重循环计数器，针对读入的图像数据，读取相应的系数相乘即可，注意由于系数只与该像素点在一行中位置有关，因此系数是由第二维循环变量 j 决定的. 其它部分与实现 1 完全相同。

```
CounterChain chain = control.count.makeCounterChain();
DFEVar i = chain.addCounter(X, 1);
DFEVar j = chain.addCounter(X, 1);
```

```
DFEVar k = chain.addCounter(3, 1);

// Input
DFEVar inImage = io.input("imageInput", dfeUInt(32));
DFEVar coef = table.read(j);
inImage = inImage.cast(dfeFloat(8,24)) * coef;
inImage = inImage.cast(dfeUInt(32));

// Output
io.output("imageOutput", inImage, dfeUInt(32));
```

3 总结

通过这次作业，我感受主要有一下几点 (主要针对 Kernel 的编写):

1. FPGA 以数据流编程的方式，这与普通的编程是不同的，比如这里的图像数据，就是每个像素点按照时钟周期依次读入的，这里的时间先后顺序就代表了空间中的位置。

2. FPGA 中的数据格式与以前的编程中用到的有些不同, 可以任意指定浮点数的位数。

3. 我遇到的主要问题就是对于数据的转换出了问题，由于 ROM 中的数据是 dfeFloat(8,24) 格式的，而图像的像素数据是 dfeUInt(32) 格式的，因此需要这两者进行统一之后进行计算。我刚开始由于没有进行数据转换就直接相乘，因此老是出错。但是当我意识到这个问题之后进行转换时还是遇到了一个问题，就是我第一次是将 ROM 中读取的数据转换乘了 dfeUInt(32) 格式的与图像数据相乘，但是这样测试的结果是不对的。后来我就先将图像的数据转换成 dfeFloat(8,24) 的，再与 ROM 的数据相乘，之后再转乘 dfeUInt(32) 的就可以测试通过, 对于其中的原因我还是不了解。