中山大学本科生实验报告

（2017 学年春季学期）

课程名称：嵌入式系统案例分析与设计　　　　任课教师：王军　　　　助教：杨涵烁

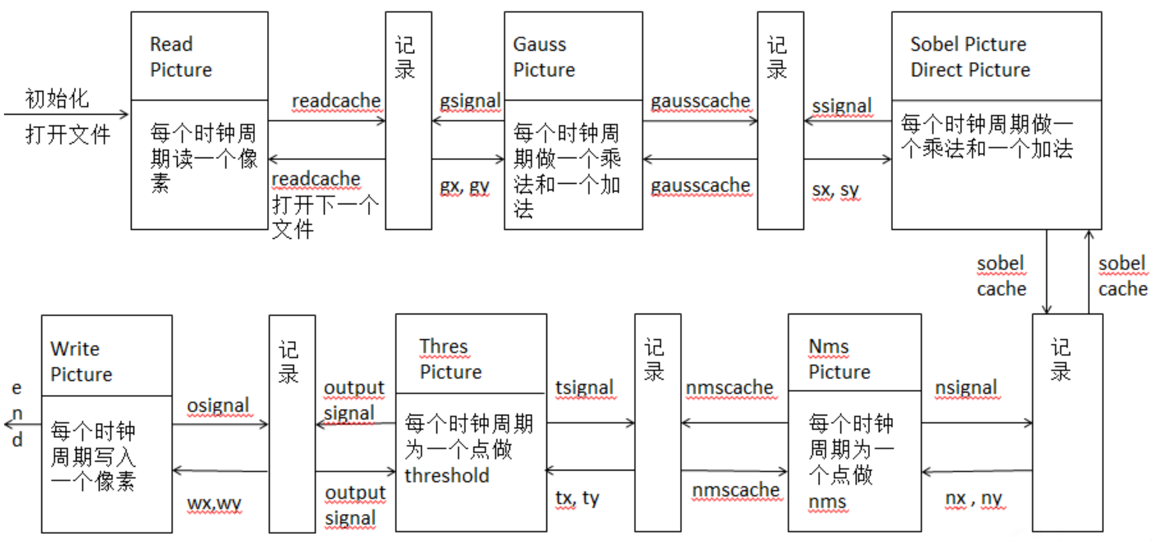
年级&班级	14 级教务二班	专业（方向）	软件工程（嵌入式方向）
学号	14331153	姓名	梁子仲
电话	18819253726	Email	473230218@qq.com
开始日期	2017/6/7	完成日期	2017/6/14

实验题目：实现 1280*720 大图的边缘检测（canny algorithm advanced）

一、实验要求：达到 15fps，实现流水线工作

二、实验内容：

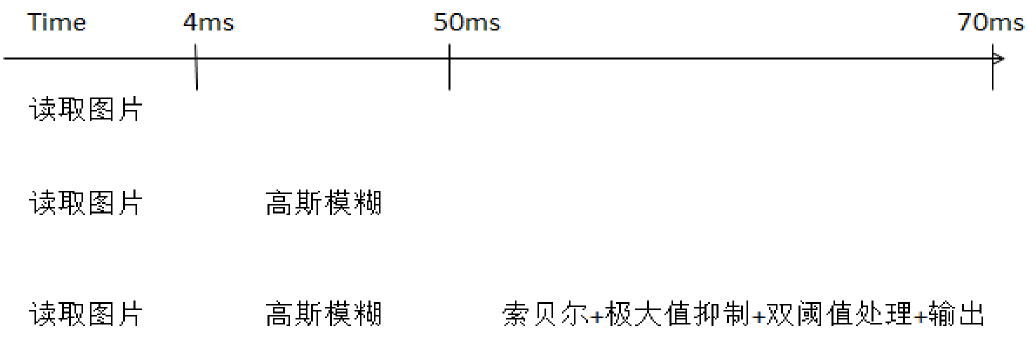
①建立体系结构图：



②分析

完成一张图的边缘检测需要以上6个步骤：读取图片，进行高斯模糊，索贝尔算法，极大值抑制，双阈值以及生成图片。

可以看到每一个步骤都依赖于上一个步骤，只有接收到上一步骤完成并发出的 cache 信号并且本步骤上一个图已经完成并发出 signal 信号，才会开始进行该步骤。在实际的工作中，发现读图花费时间较短，完成高斯需要大概50ms，完成索贝尔大概需要10ms，完成余下的所有步骤大概需要共10ms，因此第一张图生成需要共70ms，由于在本图在做高斯模糊的时候，同时开始的上一个图可以完成剩余的所有步骤，因此本质上这是一个三级流水线工序，如下图：



可以看到在这个三级流水线的工序当中，高斯模糊所用的时间最长，因此这个流水线的周期大概是46ms，即每46ms能够出一张图片，换句话说第一张图需要70ms生成，后面的每一张图需要46ms，假如需要做15张图片，那么计算时间为70+46*14 = 714ms，每个值存在误差，时间大概在705ms~725ms能够完成15张图片的边缘检测，这是>15fps的，符合要求。

其实可以将索贝尔，极大值抑制，以及双阈值，输出分开作为流水线的一个级，即变为一个6级流水线，只需要加一些条件阻塞他们之间的进行，但是这么做只会徒增时间消耗（第一张图片输出时间将多出46*3ms，后面图片不受影响）。将他们合为一个级显然时间消耗会少很多，感觉更加合理。

③代码实现

代码编写中，每一个步骤都由一个 always 控制，包括记录步骤。因此总共是11个 always：

设置时钟周期为2ns：

```
initial begin
   // Initialize Inputs
   clk = 0;
   reset = 1;
   #10;
   reset = 0;

   // Wait 100 ns for global reset to finish
   #10;
   reset = 1;
   forever
      begin
         #1;
         clk = ~clk;
      end
   // Add stimulus here

end
```

Reset 为初始化过程，这时除了初始值的赋值外，还要将第一张图片打开：

```
`define DATA_WIDTH 8
`define PICTURE_LENGTH 1280
`define PICTURE_WIDTH 720
`define PICTURE_MIX  1280*720

fileI = $fopen("./output/10.bmp","rb");
//fileI = $fopen("kodim22_200.bmp", "rb");
if (fileI == `NULL) $display("> FAIL: The file is not exist !!!\n");
else                $display("> SUCCESS : The file was read successfully.\n");

r = $fread(FILE_HEADER, fileI, 0, `LEN_HEADER);
```

读取头之后，进入 readPicture：

```
//read a picture 1280*720
always@(posedge clk)
   begin
      if(reset && rx < `PICTURE_LENGTH+1 && ry < `PICTURE_WIDTH+1)
         begin
            c = $fgetc(fileI);
            c = $fgetc(fileI);
            readsPicture[(`PICTURE_LENGTH-rx-1)*`PICTURE_WIDTH+ry] = $fgetc(fileI);
            rx = rx + 1;
            if(rx == `PICTURE_LENGTH)
               begin
                  rx = 0;
                  ry = ry + 1;
               end
            if(ry == `PICTURE_WIDTH)
               begin
                  rx = `PICTURE_WIDTH+1;
                  ry = `PICTURE_LENGTH+1;
                  readcache = 1;
                  $fclose(fileI);
               end
         end
   end
```

每个周期读取一个像素，将这个像素的8位存放在 readsPicture 中，在整个图片完成读取后，readcache 置为1,，意思是本步骤完成，并关闭文件。

记录步骤，作为一个过渡，使读取图片和高斯模糊相互不影响：

```
always@(posedge clk)
   begin
      if(reset && readcache && gsignal)
         begin
            for (i = 0; i < `PICTURE_MIX; i=i+1)
               begin
                  readsPictureCache[i] = readsPicture[i];
                  gaussPicture[i] = readsPicture[i];
               end
            readcache = 0;
            gsignal = 0;
            gx = 2;
            gy = 2;

            if(number < 25)
               begin
                  rx = 0;
                  ry = 0;
                  temp = number%10;
                  temps = number/10;
                  fr = {"./output/", "0"+temps, "0"+temp, ".bmp"};
                  fileI = $fopen(fr, "rb");
                  number = number + 1;

                  if (fileI == `NULL) $display("> FAIL: The file %s is not exist !!!\n", fr);
                  else                $display("> SUCCESS : The file %s was read successfully.\n", fr);

                  r = $fread(FILE_HEADER, fileI, 0, `LEN_HEADER);
               end
```

在第一张图开始前所有 signal 均置1，复制两个数组为高斯作准备，然后为高斯需要用到的 gx,gy 赋值，readcache 和 gsignal 置为0，由于上一个级是读图，因此打开下一张图片。使用字符串拼接的方式进行读取图片。

每一个记录步骤都只占一个时钟周期即2ns，在接收到上下两个步骤返回的完成信号后执行，然后再次启动上下两个步骤。由于每个记录步骤的作用类似，下面的记录步骤不做分析。

高斯模糊：

```
//gauss
always@(posedge clk)
   begin
      if(reset && gy < `PICTURE_LENGTH+1 && gx < `PICTURE_WIDTH+1)
         begin
            if(gx > 1 && gy < `PICTURE_LENGTH-2 && gy > 1 && gx < `PICTURE_WIDTH-2)
               begin
                  Sum = readsPictureCache[(gy+gi)*`PICTURE_WIDTH+gx+gj]*gf[(2+gi)*5+(2+gj)];
                  tpSum = tpSum + Sum;
                  gi = gi+1;
                  if(gi == 3)
                     begin
                        gi = -2;
                        gj = gj + 1;
                     end
                  if(gj == 3)
                     begin
                        gj = -2;
                        gaussPicture[gy*`PICTURE_WIDTH+gx] = (tpSum>>7);
                        tpSum = 0;
                        gx = gx + 1;
                     end
                  if(gx == `PICTURE_WIDTH-2)
                     begin
                        gx = 2;
                        gy = gy + 1;
                     end

         if(gy == `PICTURE_LENGTH-2)
            begin
               gx = `PICTURE_WIDTH+1;
               gy = `PICTURE_LENGTH+1;
               gausscache = 1;
               gsignal = 1;
            end
      end
   end
end
```

每一个周期做一个乘法，一个加法，25个周期做好一个点，并赋值到 gaussPicture 中，整个图的 gauss 做完后，gausscache 信号（反馈给前面的记录步骤）置1，gsignal 信号（给后面一个记录步骤）置1。

索贝尔：

索贝尔的做法和高斯模糊一致，也是每个时钟周期做一次乘法和一次加法，9个周期做好一个点，并赋值到相应的数组中，在做好一个点之后，需要生成方向，根据 fGx，fGy 得到方向值并赋值到相应数组。整个图的索贝尔和方向做好后，同样给上下的记录步骤发送完成信号。

Nms：

Nms 做法中，每个时钟周期做一个点的极大值抑制，比较这个点和在它方向上的左右两个点，并进行赋值。整个图的 nms 做好后，给上下的记录步骤发送完成信号。

Threshold：

双阈值处理做法中，每个时钟周期做一个点的双阈值，将这个点的值和大小阈值进行比较，并进行赋值。整个图的双阈值处理做好后，给上下的记录步骤发送完成信号。

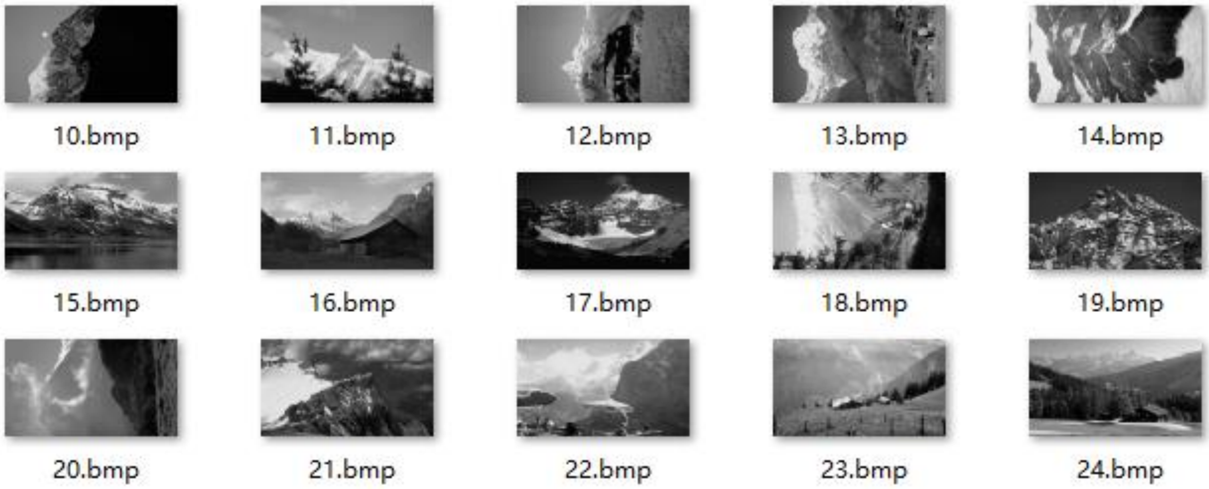
输出图片：

```
//write and output
always@(posedge clk)
   begin
      if(reset && wx < `PICTURE_LENGTH+1 && wy < `PICTURE_WIDTH+1)
         begin
            if (outputPicture[(`PICTURE_LENGTH-wx-1)*`PICTURE_WIDTH+wy] != 0)
               th = 8'hff;
            else th = 8'h00;
            //if(thresDirect[wx*`PICTURE_WIDTH+wy] == 0)
            //$fwrite(file0, "%c%c%c", sobelCache[(`PICTURE_LENGTH-wx-1)*`PICT
            $fwrite(file0, "%c%c%c", th,th,th);
            //$fwrite(file0, "%c%c%c", sobelPictureCache[(`PICTURE_LENGTH-wx-1
            //$fwrite(file0, "%c%c%c", sobelPicture[(`PICTURE_LENGTH-wx-1)*`PI
            //$fwrite(file0, "%c%c%c", gaussPicture[(`PICTURE_LENGTH-wx-1)*`PI
            //$fwrite(file0, "%c%c%c", readsPictureCache[(`PICTURE_LENGTH-wx-1
            wx = wx + 1;
            if(wx == `PICTURE_LENGTH)
               begin
                  wx = 0;
                  wy = wy + 1;
               end

            end
            if(wy == `PICTURE_WIDTH)
               begin
                  wx = `PICTURE_WIDTH+1;
                  wy = `PICTURE_LENGTH+1;
                  $display("> %s is created.\n", fw);
                  $fclose(file0);
                  osignal = 1;

                  temp = num%10;
                  temps = num/10;
                  temps = temps%10;
                  fw = {"./output/", "1", "0"+temps, "0"+temp, ".bmp"};
                  num = num + 1;
                  if(num == 116)
                     begin
                        $stop;
                     end
                  file0 = $fopen(fw, "wb");

                  for(p=1; p<55; p=p+1)
                     begin
                        $fwrite(file0, "%c", FILE_HEADER[p]);
                     end
               end
         end
   end
```

每一个周期输出一个像素，整个图完成输出后，关闭文件，并打开下一个需要写入的文件，同样使用字符串拼接的方式打开图片，在15张图片全部完成后，执行$stop 终止进程。

三、实验结果

准备15张1280*720的图片：



进行仿真：

```
ISim P.20131013 (signature 0x7708f090)
This is a Full version of ISim.
# run 1000 ns
Simulator is doing circuit initialization process.
Finished circuit initialization process.
> SUCCESS : The file   ./output/10.bmp was read successfully.

# run 1s
> SUCCESS : The file   ./output/11.bmp was read successfully.

> SUCCESS : The file   ./output/12.bmp was read successfully.

>  ./output/100.bmp is created.

> SUCCESS : The file   ./output/13.bmp was read successfully.

>  ./output/101.bmp is created.

> SUCCESS : The file   ./output/14.bmp was read successfully.

>  ./output/102.bmp is created.

> SUCCESS : The file   ./output/15.bmp was read successfully.

>  ./output/103.bmp is created.

> SUCCESS : The file   ./output/16.bmp was read successfully.

>  ./output/104.bmp is created.
```

```
> SUCCESS : The file   ./output/17.bmp was read successfully.

>  ./output/105.bmp is created.

> SUCCESS : The file   ./output/18.bmp was read successfully.

>  ./output/106.bmp is created.
```

```
> SUCCESS : The file   ./output/19.bmp was read successfully.
>  ./output/107.bmp is created.
> SUCCESS : The file   ./output/20.bmp was read successfully.
>  ./output/108.bmp is created.
> SUCCESS : The file   ./output/21.bmp was read successfully.
>  ./output/109.bmp is created.
> SUCCESS : The file   ./output/22.bmp was read successfully.
>  ./output/110.bmp is created.
> SUCCESS : The file   ./output/23.bmp was read successfully.
>  ./output/111.bmp is created.
> SUCCESS : The file   ./output/24.bmp was read successfully.
>  ./output/112.bmp is created.
>  ./output/113.bmp is created.
>  ./output/114.bmp is created.
Stopped at time : 709173609 ns :  in File "F:/case-analyse/canny_advanced/canny_advanced.v" Line 618
```

可以看到，由于是三级流水线，在第一次生成图片前，读取三次图片，之后每一个生成之前都有且只有一个读取图片，最后由于15张图片已经读取完成，因此只生成最后三张图片。

最后一行 stop 可以看到共用时709173609ns 即约709ms，>15fps，效率达标。

生成的图片：与原图进行比对


10.bmp

11.bmp

12.bmp

13.bmp

14.bmp

15.bmp

16.bmp

17.bmp

18.bmp

19.bmp

20.bmp

21.bmp

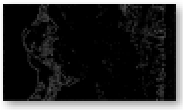
22.bmp

23.bmp

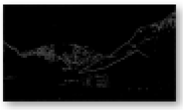
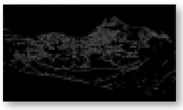
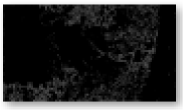
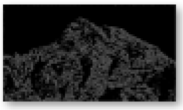
24.bmp

100.bmp

101.bmp

102.bmp

103.bmp

104.bmp

105.bmp

106.bmp

107.bmp

108.bmp

109.bmp

110.bmp

111.bmp

112.bmp

113.bmp

114.bmp

单独生成一张做比较：

效果还是不错的= 。=

至此，本次实验全部完成。

四、实验感想：

本次实验从一开始的只是填写算法，到后来将两个文件（tb,canny）整合，并加入 always 实现流水线。对于流水线的理解更加透彻，在整合 canny 算法过程中也遇到了很多不同的问题，对算法的理解也有了一定的提高。可以说这个实验提升了对于 verilog 语言，流水线的实现，边缘检测的算法，仿真软件的使用，获益良多。