به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

شبیه سازی شی گرای سیستم های الکترونیکی

CA4

علیرضا جابری راد

810196438

بهار 1401

توضیحات روند حل مسئله

بدین منظور از چنل های ساده ی آموزش داده شده در ابتدای بحث چنل ها بهره بردم که تصویر آن را در زیر مشاهده میفرمایید:

```
#include <systemc.h>

class put_if : virtual public sc_interface{
    public:
        virtual void put(sc_lv<8>) = 0;
};

class get_if : virtual public sc_interface{
    public:
        virtual void get(sc_lv<8> &) = 0;
};

class channel : public put_if, public get_if{
    bool full;
    sc_lv<8> data;
    sc_event put_event, get_event;

public:
    channel():full(false) {};
    ~channel(){};

    void put(sc_lv<8> in_data){
        if(full == true) wait(get_event);
        data = in_data;
        full = true;
        put_event.notify();
    }

    void get(sc_lv<8> &out_data){
        if(full == false) wait(put_event);
        out_data = data;
        full = false;
        get_event.notify();
    }

};
```

برای تک تک ماژول های دیگر، فقط یک ترد در نظر گرفته شده که این ترد، به گونه ای کد نویسی شده است که ماژول ما در هر لحظه، فقط مشغول به انجام یک عملیات باشد.

نکته ی مهمی که گفتن آن حائز اهمیت است، اینست که خروجی Edge Detector یک تصویر 510 در 510 خواهد بود، علت آن اینست که در لبه های تصویر، پیکسل های صفر در محاسبات لحاظ نکردم و خود تصویر خروجی Gray Scaler را برای پردازش در نظر میگیرم. در اولین سگمنتی که دریافت میشود، با انجام کانولوشن 6 سطر خروجی تولید میشود، در گام های بعد، دو سطر آخر سگمنت قبلی را در ابتدای آرایه ای که سگمنت بعدی را میخواهیم در آن ذخیره کنیم، قرار میدهیم و پردازش را انجام میدهیم. یعنی در سگمنت های دوم تا آخر، 8 سطر خروجی تولید میشود. کد مربوط به دریافت، محاسبات و ارسال سگمنت اول را در تصویر زیر مشاهده میفرمایید:

```
int Gx, Gy;
//getting the first segment
for(int i=2; i<10; i++){
    for(int j=0; j<512; j++){
        wait(clk->posedge_event());
        in->get(segment[i][j]);
    }
}

//processing the first segment
for(int i=1; i<7; i++){
    int k=i+2;
    for(int j=1; j<511; j++){
        Gx=-segment[k-1][j-1].to_uint()-2*segment[k-1][j].to_uint()-segment[k-1][j+1].to_uint()+segment[k+1][j-1].to_uint()+2*segment[k+1][j].to_uint()+segment[k+1][j+1].to_uint();
        gx[i-1][j-1]=abs(Gx);
        Gy=-segment[k-1][j-1].to_uint()-2*segment[k][j-1].to_uint()-segment[k+1][j-1].to_uint()+segment[k-1][j+1].to_uint()+2*segment[k][j+1].to_uint()+segment[k+1][j+1].to_uint();
        gy[i-1][j-1]=abs(Gy);
        //gx[i-1][j-1]=abs(gx[i-1][j-1])
        G[i-1][j-1] = gx[i-1][j-1].to_uint() + gy[i-1][j-1].to_uint();
    }
}

//sending the first segment to file writer
for(int i=0; i<6; i++){
    for(int j=0; j<510; j++){
        wait(clk->posedge_event());
        out->put(G[i][j]);
    }
}
```

برای سگمنت های دوم به بعد دستورات زیر کدنویسی شده است:

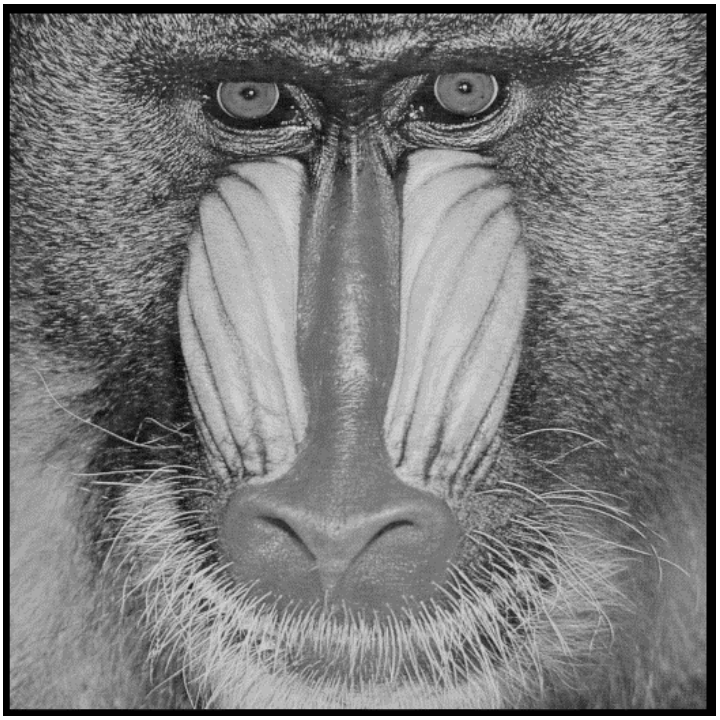
```
//getting the rest and processing the received data and transfering the result to file writer
 for(int it=0; it<63; it++){

    //transfering 2 last rows to 2 first rows of the new segment
     for(int i=0; i<2; i++){
        for(int j=0; j<512; j++)
            segment[i][j] = segment[i+8][j];
     }

    //getting 8 rows
     for(int i=2; i<10; i++){
        for(int j=0; j<512; j++){
            wait(clk->posedge_event());
            in->get(segment[i][j]);
        }
    }

    //processing the segment
    for(int i=1; i<9; i++){
        for(int j=1; j<511; j++){
            Gx=-segment[i-1][j-1].to_uint()-2*segment[i-1][j].to_uint()-segment[i-1][j+1].to_uint()+segment[i+1][j-1].to_uint()+2*segment[i+1][j].to_uint()+segment[i+1][j+1].to_uin
            gx[i-1][j-1]=abs(Gx);
            Gy=-segment[i-1][j-1].to_uint()-2*segment[i][j-1].to_uint()-segment[i+1][j-1].to_uint()+segment[i-1][j+1].to_uint()+2*segment[i][j+1].to_uint()+segment[i+1][j+1].to_uin
            gy[i-1][j-1]=abs(Gy);
            G[i-1][j-1] = gx[i-1][j-1].to_uint() + gy[i-1][j-1].to_uint();
        }
    }

    //sending the segment to file writer
    for(int i=0; i<8; i++){
        for(int j=0; j<510; j++){
            wait(clk->posedge_event());
            out->put(G[i][j]);
        }
    }
```

نکته ی دیگری که در انجام محاسبات در Gray Scaler و Edge Detector رعایت شده، اینست که داده های 8 بیتی ابتدا به عدد صحیح بدون علامت کست میشوند و پس از انجام محاسبات، در یک متغیر با تایپ صحیح ذخیره میشوند و در نهایت، عملیات تقسیم یا قدرمطلق گرفتن روی متغیر صحیح مذکور انجام میشود تا دقت محاسبات در بالاترین میزان ممکن حفظ شود. به طور مثال، در تصویر زیر متغیر temp همان متغیر صحیح میانی جهت انجام محاسبات در Gray Scaler است:

```
sc_lv<8> r,g,b;
int temp;

for(int i=0; i<512*512; i++){
    temp=0;

    wait(clk->posedge_event());
    in->get(r);
    temp+=r.to_uint();

    wait(clk->posedge_event());
    in->get(g);
    temp+=g.to_uint();

    wait(clk->posedge_event());
    in->get(b);
    temp+=b.to_uint();

    avg[i] = temp/3;
    wait(clk->posedge_event());
    out->put(avg[i]);

}
```

تصاویر خروجی

تصویر خروجی حاصل از پردازش Gray Scaler: (با اجرای فایل read_gray_file_show_img.py)

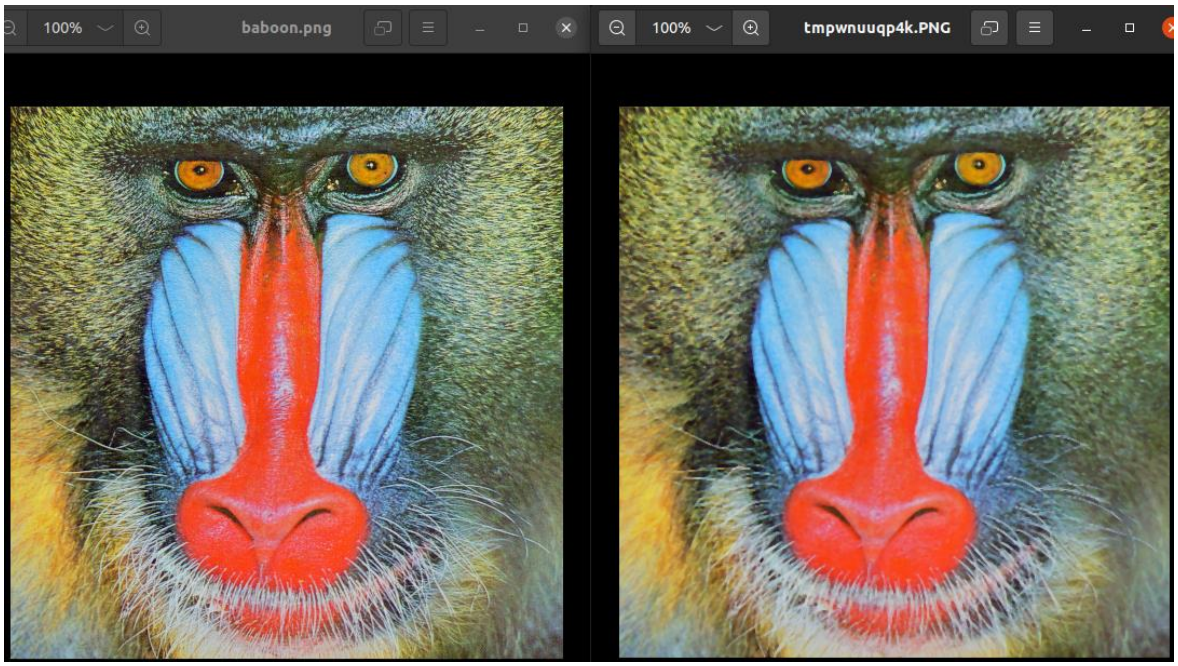


تصویر خروجی حاصل از پردازش Edge Detector:

(با اجرای فایل read_EdgeDetect_file_show_img.py)

بخش امتیازی

به منظور اضافه کردن ماژول محاسبه کننده میانه به مدار، یک سیگنال به file writer اضافه کردم تا پایان عملیات نوشتن فایل حاصل از محاسبات edge detector به ماژول های file reader (برای ارسال مجدد داده ها) و Median (برای دریافت داده ها و انجام محاسبات و ارسال مقادیر محاسبه شده به file writer) اطلاع داده شود. خروجی تصویر این بخش هم مانند edge detector ابعاد 510*510 دارد. محاسبه median هر پنجره از تصویر، با انتقال مقادیر آن پنجره به یک آرایه 9 خانه ای و مرتب کردن آن و سپس ذخیره کردن مقدار موجود در خانه پنجم آرایه(که میانه آن است) در سگمنت مقصد، انجام شده است. محاسبات این ماژول سگمنت به سگمنت انجام شده و مانند ماژول های قبل، یک ترد برای اجرا دارد و در هر لحظه فقط یک کار انجام میدهد. تصویر خروجی این بخش را که با استفاده از فایل پایتون تغییر یافته read_median_img_file.py نمایش داده شده است را در کنار تصویر اولیه مشاهده میکنید:



همانطور که انتظار داشتیم، خروجی حاصل از فیلتر Median تصویر خروجی را نسبت به حالت اصلی آن کمی تار کرده است.