**基于FPGA的摄像头数据采集与处理**

姓 名：崔柏乐

班 级： 2016级信息工程

学 号： 1628405038

联系方式 ： 18896809030

实验日期 ： 2018.12.20

**一、实验意义**

在信息化不断普及的情况下，我们一直在探索如何能够更好的模拟人类感知周围环境，因此摄像头的使用不断深入生活当中，摄像头又称为电子眼，电脑相机，可用于智能视频监控，在银行、交通、医疗、安保等领域均具有不可磨灭的作用，再比如智能人机交互，在计算机模拟人的视觉感知，对摄像头的数据采集与显示就变得极为重要。综上所述，处理电路需要对采集过来的图片或者视频来说，都需要良好的摄像头数据采集。

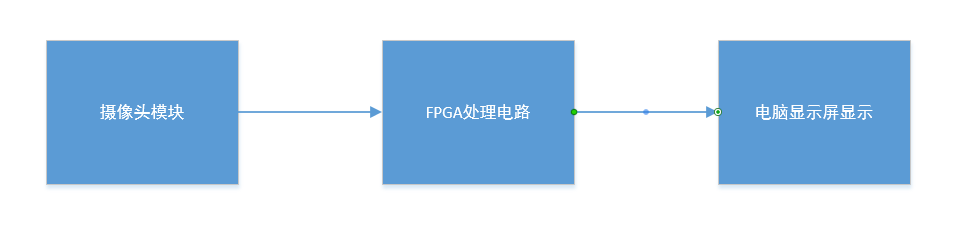
FPGA是现场可编程逻辑门阵列，是一个含有可编辑元件的半导体设备。如今单片机的使用非常常见，包含运算器、控制器、存储器等器件的单片机，可以实现一些基本的控制，虽然ARM芯片的速率够快，但是依旧存在速率收到限制的问题，因此使用FPGA便具有一定的优势，在数据处理上能够更胜一筹。

**二、工作指标**

本设计是使用FPGA对摄像头采集过来的数据进行显示，主要指标为当前图像能够实时显示，视频不出现间断现象、分辨率能达到640\*480、能够实现拍照功能，可以对图像进行简单的处理。

**三、设计方案**

（一）系统总体框图



（二）设计思路

1、摄像头模块

（1）OV7670摄像头简介

OV7670，图像传感器，体积小，工作电压低，提供单片VGA摄像头和影像处理器的所有功能。通过SCCB总线控制，可以输入整帧、子采样、取窗口等方式的各种分辨率8位影像数据。该产品VGA图像最高达到30帧/秒。用户可以完全控制图像质量、数据格式和传输方式。所有图像处理功能过程包括伽玛曲线、白平衡、饱和度、色度等都可以通过SCCB接口编程。OmmiVision图像传感器应用独有的传感器技术，通过减少或消除光学或电子缺陷如固定图案噪声、托尾、浮散等，提高图像质量，得到清晰的稳定的彩色图像。实物图如下所示。

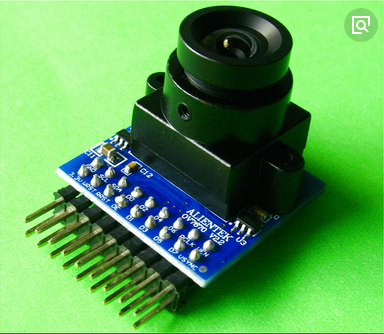


图1 OV7670摄像头模块

（2）摄像头传感器简单原理

景物通过镜头(LENS)生成的光学图像投射到图像传感器表面上，然后转为电信号，经过A/D (模数转换)转换后变为数字图像信号，再送到数字信号处理芯片(DSP)中加工处理，再通过USB接口传输到电脑中处理，通过显示器就可以看到图像了

（3）0V7670数据传输

本模块自带FIFO，作为数据缓冲，因此数据采集便会简单很多，只需要进行正确的读取，便可实现摄像头数据的采集。

本设计中使用专用的OV7670与VGA转接板与FPGA进行通信，通过自制的PCB板进行信号的有效连接。OV7670与VGA转接板原理图如下所示。

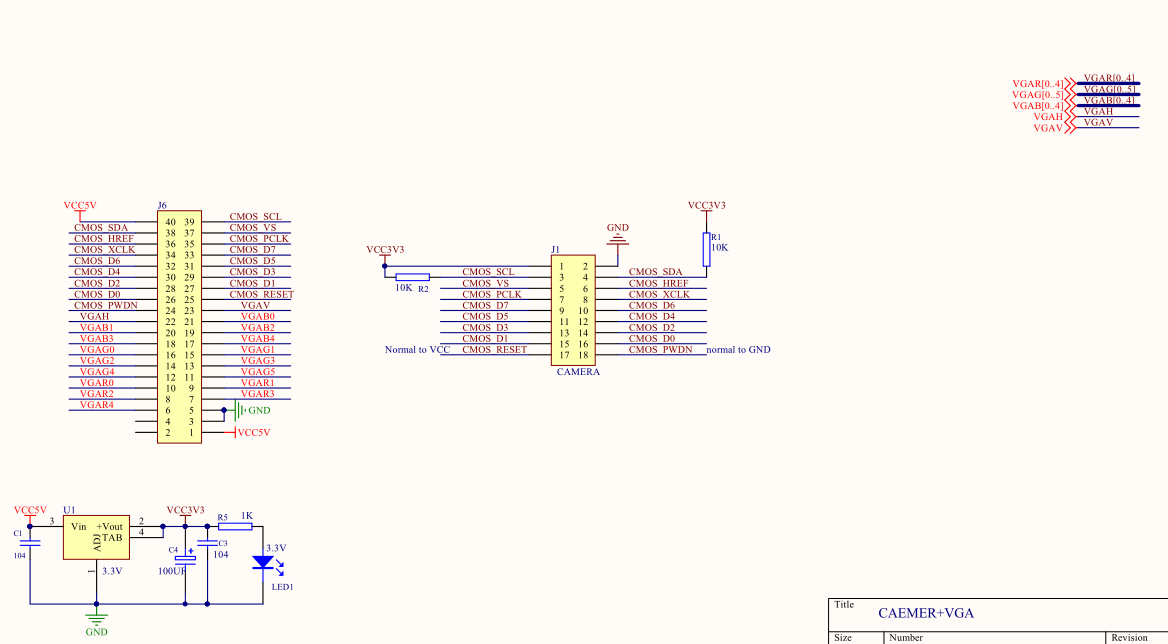


图2 OV7670转VGA模块原理图

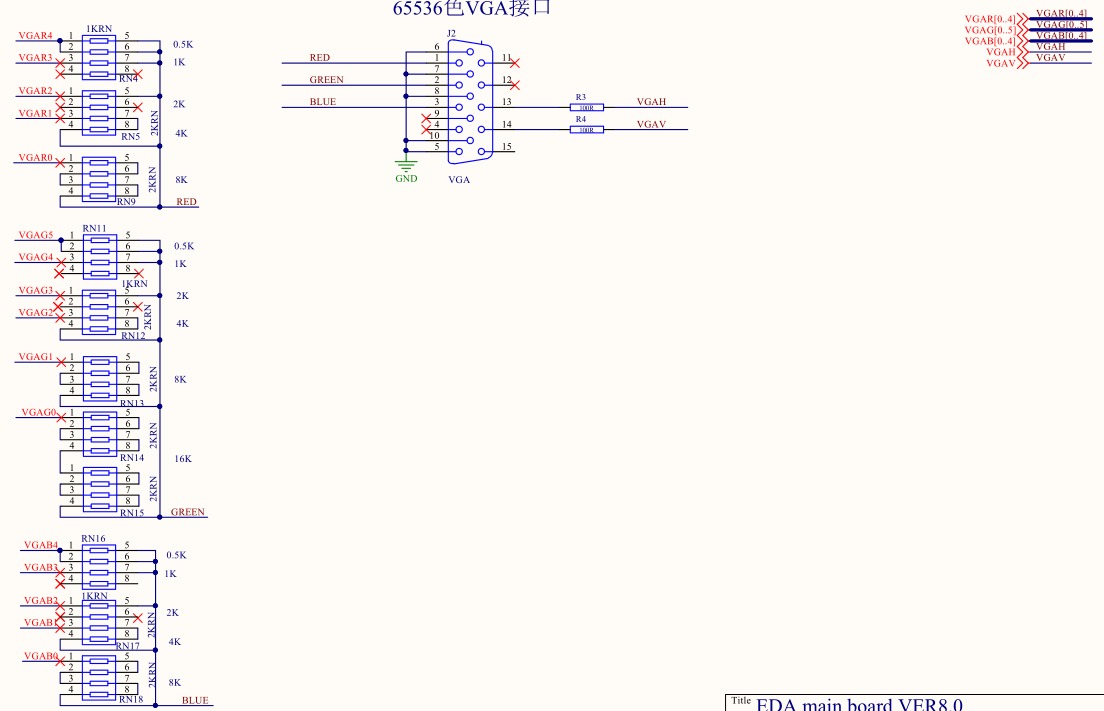


图3 OV7670转VGA模块原理图

为了方便转接板与FPGA相互连接，通过以下简单的PCB进行信号的连接，避免使用多根杜邦线，防止出现信号的丢失。

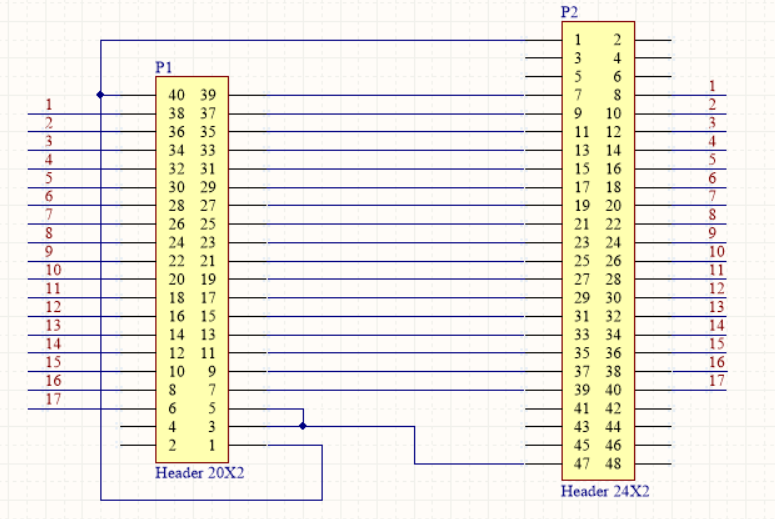


图4 OV7670转VGA模块与FPGA连接原理图

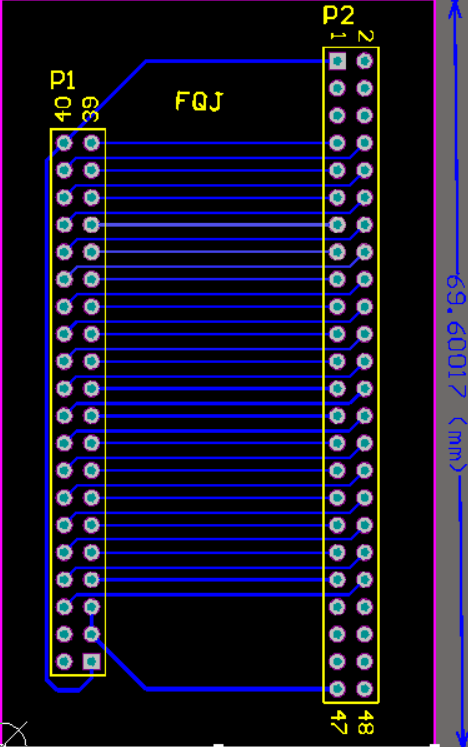


图5 OV7670转VGA模块与FPGA连接PCB图

2、显示模块

通过VGA接口，将FPGA处理好的数据输出来，在电脑屏幕上显示当前摄像头的显示以及实现拍照功能显示。实物图如下所示。



图6 电脑显示屏

3、处理器模块

（1）处理器简单介绍

处理器主要采用artix-7核心板，其采用Xilinx XC7A35TCSG324-1芯片，含有2600 个逻辑 Slices，每个包含4个6输入LUT和 8个触发器，内部时钟频率达450MHz，片上自带模数转换器，含900Kbits的Block RAM和45 个DSP48E单元，有5个时钟管理单元

（2）开发平台

本设计中使用vivado进行硬件编程。Vivado 设计分为 Project Mode 和 Non-project Mode 两种模式，一般简单设计中，我们常用的是 Project Mode。

该开发平台可支持的硬件编程语言有VHDL，Verilog等常见的硬件编程语言，本设计中使用Verilog进行数据的采集与处理。

（3）FPGA实物图



图7 FPGA实物图

4、数据处理

将从摄像头中读取的数据放到RAM中进行数据的后续处理，将摄像头采集过来的三色数据相加后与阈值进行比较，大于阈值则输出黑色或者白色，反之取相反的颜色。即对数据进行简单的二值化。

将采集到的图像进行预处理，送入卷积层进行卷积。本设计中使用CNN卷积对图片进行训练与学习，采用如下结构：

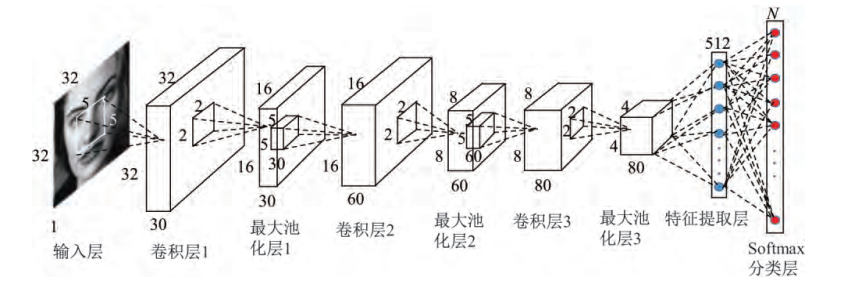


图8 CNN结构体

摄像头输入是RGB三通道640\*480像素，输入层是经过压缩和二值化（经过TensorFlow仿真得到阈值为1.38时候的输出图像，反映的特征较全）预处理后输出为32\*32像素的图片，经过30\*5\*5的卷积核卷积后输出为30\*32\*32（padding=2，完全卷积）的特征图，经过30\*2\*2的最大池化输出为30\*16\*16的特征图，之后再经过60\*30\*5\*5的卷积后输出为60\*16\*16的卷积层2，,再经过池化得到60\*8\*8的池化层2，再经过80\*60\*5\*5的卷积核得到80\*8\*8的卷积层3，再经过80\*2\*2得到80\*4\*4的池化层3。然后通过512个神经元组成的全连接层，最后softmax分类输出（N=10）。

为了防止过拟合，在前两层卷积层之后加入了LRN，对卷积输出的特征图进行聚合统计，激活函数采用RELU激活。

因为受到板卡存储资源的局限，即使不对代码进行并行优化，综合出的电路也超过了芯片的资源，无法实现，所以，根据各层占用资源的情况，去掉了LRN，全连接层有512捡到了128.

**四、实验结果**

（一）FPGA编程仿真原理图与源码

（1）软件设计流程图

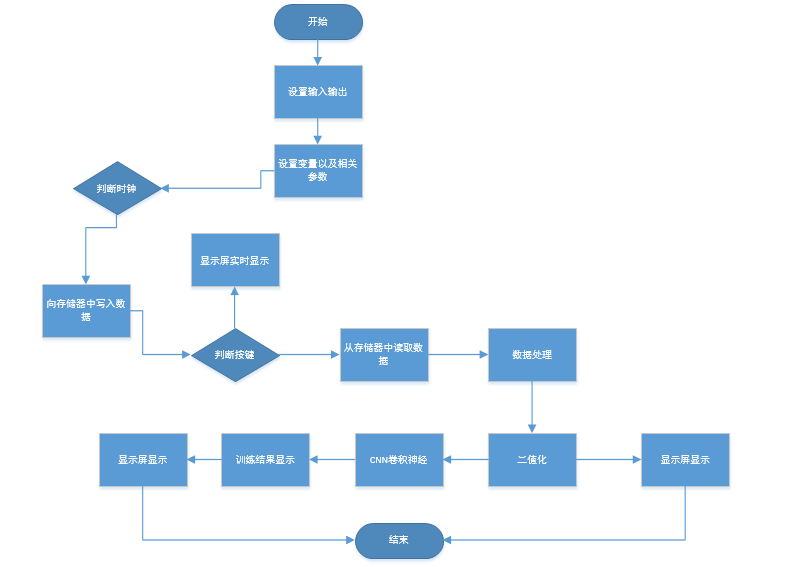


图9 摄像头数据采集与处理流程图

（2）对Verilog代码综合后得到如下原理图

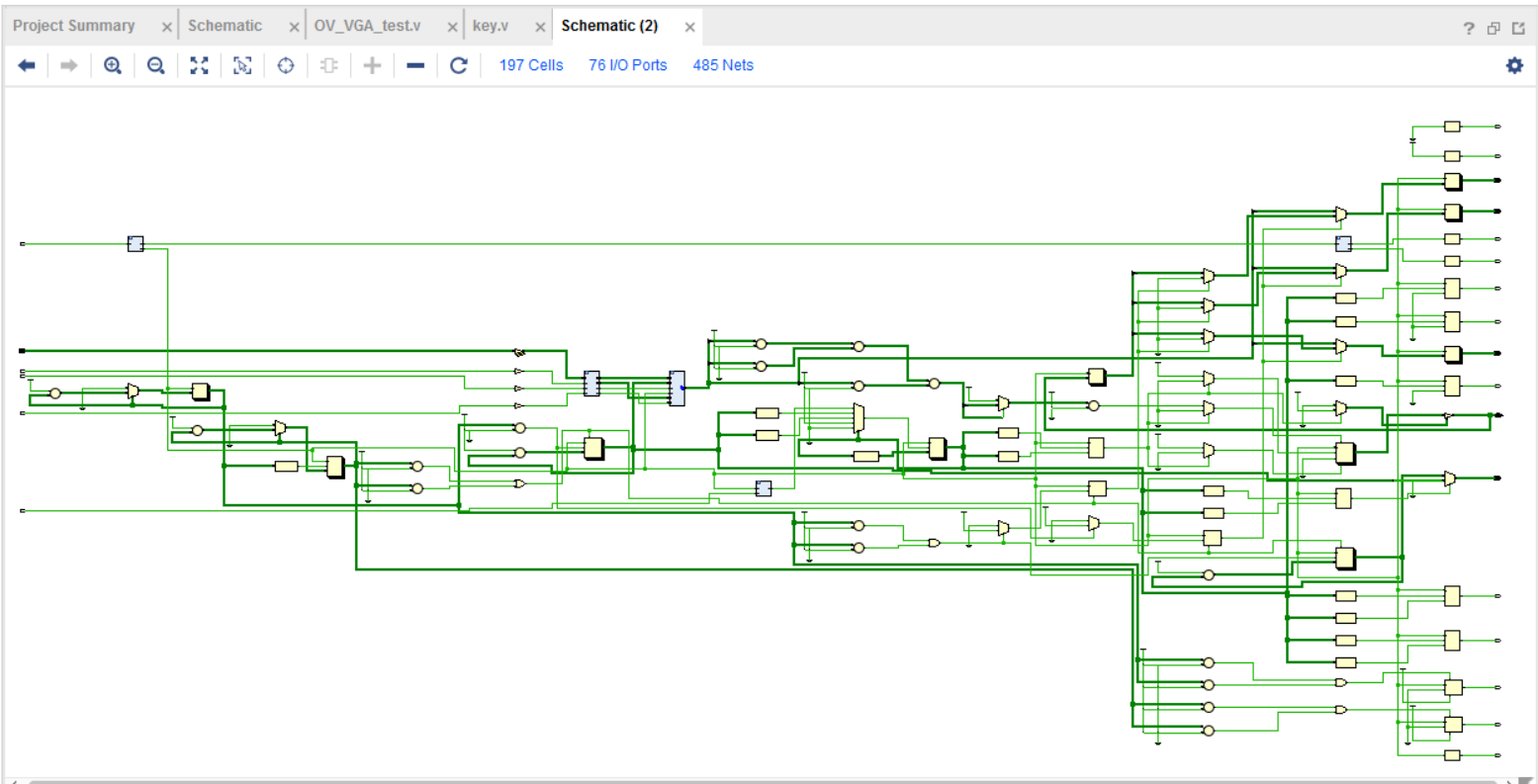


图10 综合后的原理图

（3）源码

见附件

（二）实现摄像头图像显示

随着摄像头的移动，能够实时捕捉当下的图像，并且没有延迟。

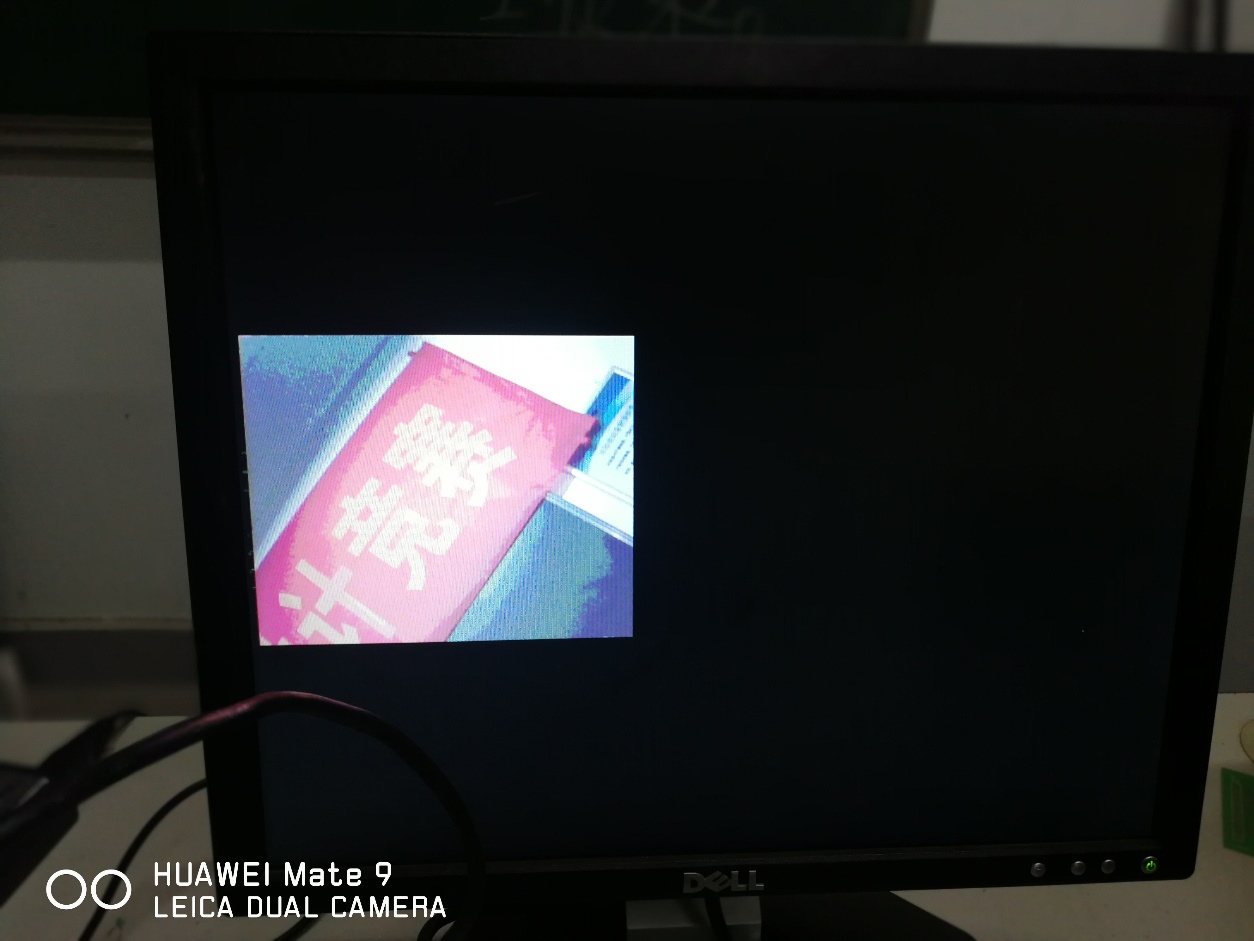


图11 实时显示图像

（三）实现拍照功能

当按键后，拍下当时的图形，并进行二值化处理后显示在液晶屏上。



图12 拍照功能（预处理后）

（四）作品展示

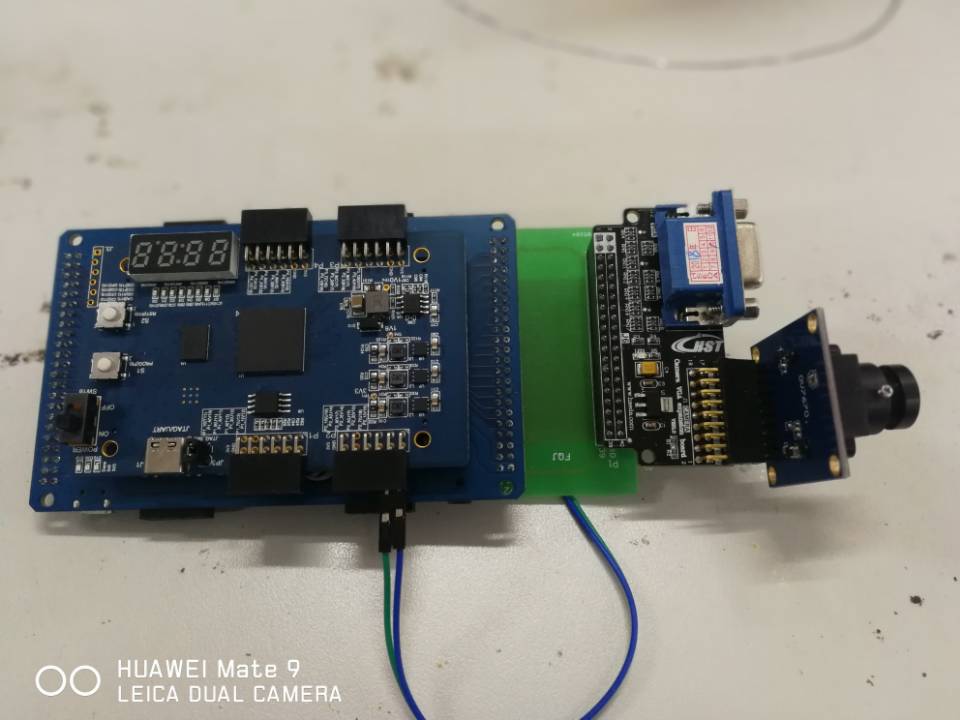


图13 作品展示

**五、参考文献**

[1]改进Softmax分类器的深度卷积神经网络及其在人脸识别中的应用\_冉鹏

[2]基于FPGA的卷积神经网络卷积层并行加速结构设计\_陈煌

[3]camer介绍

[4]基于MATLAB的人脸识别系统

[5]OV7670 datasheet

[6]Verilog HDL 入门教程

[9]Vivado 开发平台教程

**附件 程序源码**

**摄像头模块代码**

module OV\_VGA\_TEST(

input clk100,

input wire pclk,

input wire vsync,

input wire href,

input wire [7:0] d,

input wire key,

//--------------------------SRAM

inout wire [15:0] sram\_data,

output reg sram\_oe\_r,

output reg sram\_ce\_r,

output reg sram\_we\_r,

output reg sram\_ub\_r,

output reg sram\_lb\_r,

output wire [18:0] sram\_addr,

//------------------------------

output reg[4:0] vga\_red,

output reg[5:0] vga\_green,

output reg[4:0] vga\_blue,

output reg vga\_hsync,

output reg vga\_vsync,

output wire xclk,

output wire I2C\_SCLK,

output wire I2C\_SDAT,

output wire pwdm,

output wire reset\_cam

);

parameter hRez = 800;

parameter hStartSync = 840;

parameter hEndSync = 800+40+128;

parameter hMaxCount = 1056;

parameter vRez = 600;

parameter vStartSync = 601;

parameter vEndSync = 600+1+4;

parameter vMaxCount = 628;

parameter hsync\_active =0;

parameter vsync\_active = 0;

reg[10:0] hCounter;

reg[10:0] vCounter;

reg[16:0] address;

reg blank;

wire [15:0] frame\_pixel;

wire [16:0] frame\_addr;

wire [10:0] HCnt;

wire [10:0] VCnt;

wire clk25;

wire clk40;

//----------------------------cap

reg [15:0] cap\_pixel;

reg blank\_cap = 1;

reg [18:0] cap\_addr;

//------------------------------

initial hCounter = 10'b0;

initial vCounter = 10'b0;

initial address = 17'b0;

initial blank = 1'b1;

assign pwdm = 1'b0;

assign reset\_cam = 1'b1;

assign xclk = clk40; //摄像头时钟

assign frame\_addr = address;

assign HCnt = hCounter;

assign VCnt = vCounter;

always @(posedge clk40)begin

if( hCounter == hMaxCount-1 )

begin

hCounter <= 10'b0;

if (vCounter == vMaxCount-1 )

vCounter <= 10'b0;

else

vCounter <= vCounter+1;

end

else

hCounter <= hCounter+1;

if (blank ==0)

begin

vga\_red <= frame\_pixel[15:11] ;

vga\_green <= frame\_pixel[10:5];

vga\_blue <= frame\_pixel[4:0];

end

//-------------------------------------

else if(blank\_cap == 0) begin

vga\_red <= cap\_pixel[15:11] ;

vga\_green <= cap\_pixel[10:5];

vga\_blue <= cap\_pixel[4:0];

end

//-------------------------------------

else begin

vga\_red <= 5'b0;

vga\_green <= 6'b0;

vga\_blue <= 5'b0;

end;

if( vCounter >= 360+50 || vCounter < 120+50) begin

address <= 17'b0;

blank <= 1;

//----------

cap\_addr <= 0;

blank\_cap <= 1;

//----------

end

else begin

if ( hCounter < 480-160 && hCounter >= 160-160) begin

blank <= 0;

address <= address+1;

end

else

blank <= 1;

//------------------------------------------------------- if ( hCounter < 480+250 && hCounter >= 160+250) begin

blank\_cap <= 0;

cap\_addr <= cap\_addr+1;

end

else

blank\_cap <= 1;

//----------------------------------------------------- end;

if( hCounter > hStartSync && hCounter <= hEndSync)

vga\_hsync <= hsync\_active;

else

vga\_hsync <= ~ hsync\_active;

if( vCounter >= vStartSync && vCounter < vEndSync )

vga\_vsync <= vsync\_active;

else

vga\_vsync <= ~ vsync\_active;

end

//-------------------------------

wire wclk;

wire we;

wire [15:0] dout;

wire [16:0] addr;

//--------------------------------

clk\_wiz\_0 clk\_wiz\_0

(

// Clock out ports

.clk\_out1(clk25), // output clk\_out1

.clk\_out2(clk40),

// Clock in ports

.clk\_in1(clk100)); // input clk\_in1

//----------------------

blk\_mem\_gen\_0 blk\_mem\_gen\_0 (

.clka(wclk), // input wire clka

.wea(we), // input wire [0 : 0] wea

.addra(addr), // input wire [16 : 0] addra

.dina(dout), // input wire [15 : 0] dina

.clkb(clk40), // input wire clkb

.addrb(frame\_addr), // input wire [16 : 0] addrb

.doutb(frame\_pixel) // output wire [15 : 0] doutb

);

//---------------------------

camCap\_0 camCap\_0 (

.pclk(pclk), // input wire pclk

.vsync(vsync), // input wire vsync

.href(href), // input wire href

.d(d), // input wire [7 : 0] d

.addr(addr), // output wire [16 : 0] addr

.dout(dout), // output wire [15 : 0] dout

.we(we), // output wire we

.wclk(wclk) // output wire wclk

);

//------------------------

IICctrl\_0 IICctrl\_0 (

.iCLK(clk25), // input wire iCLK

.I2C\_SCLK(I2C\_SCLK), // output wire I2C\_SCLK

.I2C\_SDAT(I2C\_SDAT) // inout wire I2C\_SDAT

);

//----------------------key

wire key1\_flag;

key2 key2(

.clk (clk40),

.key1 (key),

.key1\_flag (key1\_flag)

);

//-----------------SRAM\_TIMING

parameter SRAM\_ADDR\_END = 76800;

reg [2:0] state = 0;

reg en = 0;

reg [15:0] sram\_din;

reg read = 0;

assign sram\_data = (en)? 16'bz:sram\_din;

assign sram\_addr = (read)? cap\_addr:{2'b00,frame\_addr}; //与BRAM的地址一致，保存BRAM的图像数据

wire [15:0]R\_temp,G\_temp,B\_temp;

wire [7:0]EX\_R={frame\_pixel[15:11],frame\_pixel[13:11]};

wire [7:0]EX\_G={frame\_pixel[10:5],frame\_pixel[6:5]};

wire [7:0]EX\_B={frame\_pixel[4:0],frame\_pixel[2:0]};

wire [16:0]Gray\_temp;

wire [7:0]Gray;

assign R\_temp=EX\_R\*8'd76;

assign G\_temp=EX\_G\*8'd150;

assign B\_temp=EX\_B\*8'd30;

assign Gray\_temp=R\_temp+G\_temp+B\_temp;

assign Gray=(Gray\_temp[16])?8'hff:Gray\_temp[15:8];

//对图像进行二值化

parameter TH =80;

always @(negedge clk40)

if(en == 0)

begin

if(Gray<=TH)

sram\_din <= 16'h0000; //黑色

else

sram\_din <= 16'hffff; //白色

end

//---

always @(posedge clk40)

case(state)

0: if(key1\_flag == 1)

state <= 1;

1: if(frame\_addr == SRAM\_ADDR\_END-1)

state <= 2;

2: begin //开始写数据

sram\_oe\_r <= 1'b1;

sram\_we\_r <= 1'b0;

sram\_ce\_r <= 1'b0;

sram\_ub\_r <= 1'b0;

sram\_lb\_r <= 1'b0;

en <=0;

read <=0;

if(frame\_addr == SRAM\_ADDR\_END-1)

state <= 3;

end

3: begin //数据写完，开始读数据

sram\_oe\_r <= 1'b0;

sram\_we\_r <= 1'b1;

sram\_ce\_r <= 1'b0;

sram\_ub\_r <= 1'b0;

sram\_lb\_r <= 1'b0;

en <= 1;

read <= 1;

state <= 0;

end

default: ;

endcase

//---------------

always @(negedge clk40)

// if(en == 1)

cap\_pixel <= sram\_data;

//cap\_pixel <= frame\_pixel;

// else

// cap\_pixel <= 16'b00011\_000011\_11111;

Endmodule

**//按键**

module key2(

input wire clk,

input wire key1,

output reg key1\_flag

);

reg [24:0] cnt;

parameter CNT\_END = 33000000; //递增频率

parameter TIME = 99; //消抖时间

always @(posedge clk)

if(key1) //按键弹起

cnt<=0;

else if(cnt==CNT\_END)

cnt<=0;

else if(!key1) //按键按下

cnt<=cnt+1;

//---

always @(posedge clk)

if(cnt==TIME)

key1\_flag<=1;

else

key1\_flag<=0;

//---

Endmodule

**CNN卷积核**

#include "layer.h"

#include <math.h>

#include <ap\_fixed.h>

#include <ap\_int.h>

/\*

typedef ap\_int<16> d\_int;

typedef ap\_fixed<16,4> fixed\_wb;

typedef ap\_fixed<16,9> fixed\_a;

\*/

void conv\_layer(fixed\_a \*input, fixed\_wb \*weights, fixed\_wb \*bias,

fixed\_a \*output, d\_int height, d\_int width, d\_int input\_depth,

d\_int output\_depth,d\_int kernel\_size, d\_int pad, d\_int stride) {

d\_int output\_height = (height - kernel\_size + 2 \* pad) / stride + 1;

d\_int output\_width = (width - kernel\_size + 2 \* pad) / stride + 1;

conv\_layer\_label1:for (d\_int f = 0; f < output\_depth; f++) {

conv\_layer\_label6:for (d\_int out\_h = 0; out\_h < output\_height; out\_h++) {

conv\_layer\_label5:for (d\_int out\_w = 0; out\_w < output\_width; out\_w++) {

fixed\_a sum = bias[f];

conv\_layer\_label4:for (d\_int c = 0; c < input\_depth; c++) {

conv\_layer\_label3:conv\_layer\_label0:for (d\_int k\_h = 0; k\_h < kernel\_size; k\_h++) {

conv\_layer\_label2:for (d\_int k\_w = 0; k\_w < kernel\_size; k\_w++) {

d\_int in\_h = out\_h \* stride + k\_h - pad;

d\_int in\_w = out\_w \* stride + k\_w - pad;

if (in\_h < 0 || in\_w < 0 || in\_h >= height || in\_w >= width)

continue;

d\_int wgt\_idx = f \* input\_depth \* kernel\_size \* kernel\_size +

c \* kernel\_size \* kernel\_size + k\_h \* kernel\_size +

k\_w;

d\_int in\_idx = c \* height \* width + in\_h \* width + in\_w;

sum += weights[wgt\_idx] \* input[in\_idx];

}

}

}

d\_int out\_idx =

f \* output\_height \* output\_width + out\_h \* output\_width + out\_w;

output[out\_idx] = sum;

}

}

}

}

void fc\_layer(fixed\_a \*input, fixed\_wb \*weights, fixed\_wb \*bias,

fixed\_a \*output, d\_int nrows, d\_int ncols) {

fc\_layer\_label8:for (d\_int r = 0; r < nrows; r++) {

fixed\_a sum = bias[r];

fc\_layer\_label7:for (d\_int c = 0; c < ncols; c++)

sum += input[c] \* weights[r \* ncols + c];

output[r] = sum;

}

}

void maxpool\_layer(fixed\_a \*input, fixed\_a \*output, d\_int height, d\_int width,

d\_int depth, d\_int kernel\_size, d\_int stride, d\_int pad) {

d\_int output\_height = (height - kernel\_size + 2 \* pad) / stride + 1;

d\_int output\_width = (width - kernel\_size + 2 \* pad) / stride + 1;

maxpool\_layer\_label13:for (d\_int c = 0; c < depth; c++) {

maxpool\_layer\_label12:for (d\_int out\_h = 0; out\_h < output\_height; out\_h++) {

maxpool\_layer\_label11:for (d\_int out\_w = 0; out\_w < output\_width; out\_w++) {

d\_int out\_idx =

c \* output\_height \* output\_width + out\_h \* output\_width + out\_w;

maxpool\_layer\_label10:for (d\_int k\_h = 0; k\_h < kernel\_size; k\_h++) {

maxpool\_layer\_label9:for (d\_int k\_w = 0; k\_w < kernel\_size; k\_w++) {

d\_int in\_h = out\_h \* stride + k\_h - pad;

d\_int in\_w = out\_w \* stride + k\_w - pad;

if (in\_h < 0 || in\_w < 0 || in\_h >= height || in\_w >= width)

continue;

d\_int in\_idx = c \* height \* width + in\_h \* width + in\_w;

if (k\_h == 0 && k\_w == 0)

output[out\_idx] = input[in\_idx];

else

output[out\_idx] = (output[out\_idx] > input[in\_idx])

? output[out\_idx]

: input[in\_idx];

}

}

}

}

}

}

void relu\_layer(fixed\_a \*input, fixed\_a \*output, d\_int height, d\_int width,

d\_int depth) {

relu\_layer\_label16:for (d\_int c = 0; c < depth; c++) {

relu\_layer\_label15:for (d\_int h = 0; h < height; h++) {

relu\_layer\_label14:for (d\_int w = 0; w < width; w++) {

d\_int idx = c \* height \* width + h \* width + w;

output[idx] = (input[idx] < (fixed\_a)0) ? (fixed\_a)0 : input[idx];

}

}

}

}

void softmax\_layer(fixed\_a \*input, fixed\_a \*output, d\_int len) {

fixed\_a sum = 0;

softmax\_layer\_label18:for (d\_int i = 0; i < len; i++) {

output[i] = exp((float)input[i]);

sum += output[i];

}

softmax\_layer\_label17:for (d\_int i = 0; i < len; i++)

output[i] /= sum;

}

//

/\*

void lrn(float \*pool,float \*lrn, int localSize, float alpha, float beta, int k, int W, int H, int M )

{

//int W=27, H=27, M=96;

// set lrn to zero

lrn\_label21:for(int w=0; w<W; w++)

{

lrn\_label20:for(int h=0; h<H; h++)

{

lrn\_label19:for(int m=0; m<M; m++)

{

// lrn[w][h][m]= 0;

\*(lrn + m\*(W\*H) + w\*H + h)=0;

//pool[w][h][m]=0;

// \*(pool + m\*(27\*27) + w\*27 + h)=0;

}

}

}

// compute lrn

int lsby2=localSize/2; // local size by 2

// std::cout << " lsby2 = " << lsby2 << "\n" ;

lrn\_label25:for(int w=0; w<W; w++)

{

lrn\_label24:for(int h=0; h<H; h++)

{

lrn\_label23:for(int m=0; m<M; m++)

{

// formulae

// lrn[w][h][m] = pool[w][h][m] / ((k+alpha/localSize\*sum(pool(w,h,mStart:mEnd).^2))^beta)

float mStart, mEnd, mMinuslsby2, mPluslsby2, sum2=0, temp,expTempBeta;

mMinuslsby2=m-lsby2;

mPluslsby2=m+lsby2;

mStart=(mMinuslsby2 > 0)?(mMinuslsby2):0;

mEnd=(mPluslsby2 < (M-1))? mPluslsby2 : (M-1);

//sum2\_lrn\_kernel(sum2, pool, w, h, mStart, mEnd);

float temp\_sum2=0;

lrn\_label22:for(int k1=mStart; k1<=mEnd; k1++)

{

//temp\_sum2 += pool[w][h][k1]\*pool[w][h][k1];

temp\_sum2 += (\*(pool + k1\*(W\*H) + w\*H + h))\*(\*(pool + k1\*(W\*H) + w\*H + h)) ;

}

sum2=temp\_sum2;

// std::cout << " sum2 = " << sum2 << "\n" ;

temp = ( k + alpha/localSize\*sum2 ) ;

// std::cout << " temp = " << temp << "\n" ;

expTempBeta=pow(temp,beta);

// std::cout << " expTempBeta = " << expTempBeta << "\n" ;

// exponent(expTempBeta, temp , beta);

//lrn[w][h][m] = pool[w][h][m] / ( exponent(( k + alpha/localSize\*sum2 ) , beta));//(((k + (alpha/localSize\*sum2))^beta)

// std::cout << " temp = pool[w][h][m] / (expTempBeta) " << pool[w][h][m] / (expTempBeta)<< "\n" ;

// lrn[w][h][m] = pool[w][h][m] / (expTempBeta) ;//(((k + (alpha/localSize\*sum2))^beta)

\*(lrn + m\*(W\*H) + w\*H + h)= (\*(pool + m\*(W\*H) + w\*H + h)) / expTempBeta;

}

}

}

}

\*/

//

//#include <string>

//#include<iostream>

//#include<fstream>

//#include<typeinfo>

//using namespace std;

void cnn\_net(

fixed\_a input[32][32],

fixed\_a prob[10]

/\*fixed\_a p1,

fixed\_a p2,

fixed\_a p3,

fixed\_a p4,

fixed\_a p5,

fixed\_a p6,

fixed\_a p7,

fixed\_a p8,

fixed\_a p9,

fixed\_a p10\*/

)

{

fixed\_wb conv1\_weights[30][5][5];//={

// #include "c1\_w.h"

//};

fixed\_wb conv1\_bias[30];//={

// #include "c1\_b.h"

// };

fixed\_wb conv2\_weights[60][30][5][5];//={

//#include "c2\_w.h"

//};

fixed\_wb conv2\_bias[60];//={

// #include "c2\_b.h"

// };

fixed\_wb conv3\_weights[80][60][5][5];//={

//#include "c3\_w.h"

// };

fixed\_wb conv3\_bias[80];//={

// #include "c3\_b.h"

// };

fixed\_wb ip\_weights[1280][128];//={

// #include "ip\_w.h"

//};

fixed\_wb ip\_bias[128];//={

// #include "ip\_b.h"

// };

//

fixed\_a conv1[30][32][32];

fixed\_a relu1[30][32][32];

// float lrn1 [30][32][32];

fixed\_a pool1[30][16][16];

fixed\_a conv2[60][16][16];

fixed\_a relu2[60][16][16];

// float lrn2 [60][16][16];

fixed\_a pool2[60][8][8];

fixed\_a conv3[80][8][8];

fixed\_a relu3[80][8][8];

fixed\_a pool3[80][4][4];

fixed\_a ip[128];

fixed\_a relu4[128];

/////////////////////////////////////////////////////////////////

conv\_layer((fixed\_a \*)input, (fixed\_wb \*)conv1\_weights, (fixed\_wb \*)conv1\_bias, (fixed\_a \*)conv1, 32, 32, 1, 30, 5, 2, 1);

relu\_layer((fixed\_a \*)conv1, (fixed\_a \*)relu1, 32, 32, 30);

//lrn((float \*)relu1,(float \*)lrn1,5,0.0005,0.75,1,32,32,30);

maxpool\_layer((fixed\_a \*)relu1, (fixed\_a \*)pool1, 32, 32, 30, 2, 2, 0);

conv\_layer((fixed\_a \*)pool1, (fixed\_wb \*)conv2\_weights, (fixed\_wb \*)conv2\_bias, (fixed\_a \*)conv2, 16, 16, 30, 60, 5, 2, 1);

relu\_layer((fixed\_a \*)conv2, (fixed\_a \*)relu2, 16, 16, 60);

// lrn((float \*)relu2,(float \*)lrn2,5,0.0005,0.75,1,16,16,60);

maxpool\_layer((fixed\_a \*)relu2, (fixed\_a \*)pool2, 16, 16, 60, 2, 2, 0);

conv\_layer((fixed\_a \*)pool2, (fixed\_wb \*)conv3\_weights, (fixed\_wb \*)conv3\_bias, (fixed\_a \*)conv3, 8, 8, 60, 80, 5, 2, 1);

relu\_layer((fixed\_a \*)conv3,(fixed\_a \*) relu3, 8, 8, 80);

maxpool\_layer((fixed\_a \*)relu3,(fixed\_a\*) pool3, 8, 8, 80, 2, 2, 0);

fc\_layer((fixed\_a \*)pool3, (fixed\_wb \*)ip\_weights,(fixed\_wb \*) ip\_bias, (fixed\_a \*)ip, 128, 1280);

relu\_layer((fixed\_a \*)ip,(fixed\_a \*) relu4, 1, 128, 1);

softmax\_layer((fixed\_a \*)relu4, (fixed\_a\*)prob, 10);

/\*

p1=prob[0];

p2=prob[1];

p3=prob[2];

p4=prob[3];

p5=prob[4];

p6=prob[5];

p7=prob[6];

p8=prob[7];

p9=prob[8];

p10=prob[9];