Basic Image processing

G.Gowtham, Priyankar Tejwan

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

FPGA LAB – EE5811

Indian Institute of Technology Hyderabad

April 26, 2019

Contents

Contents

Introduction

- FPGA's are ideal for control applications because they can run in extremely fast, highly deterministic loop rates.
- Basic image processing functions like filtering, edge detection etc.are very much used for other purposes like object detection etc. in real life.
- Through this project we try to make module to perform some basic image processing using icoboard and raspberryPi.

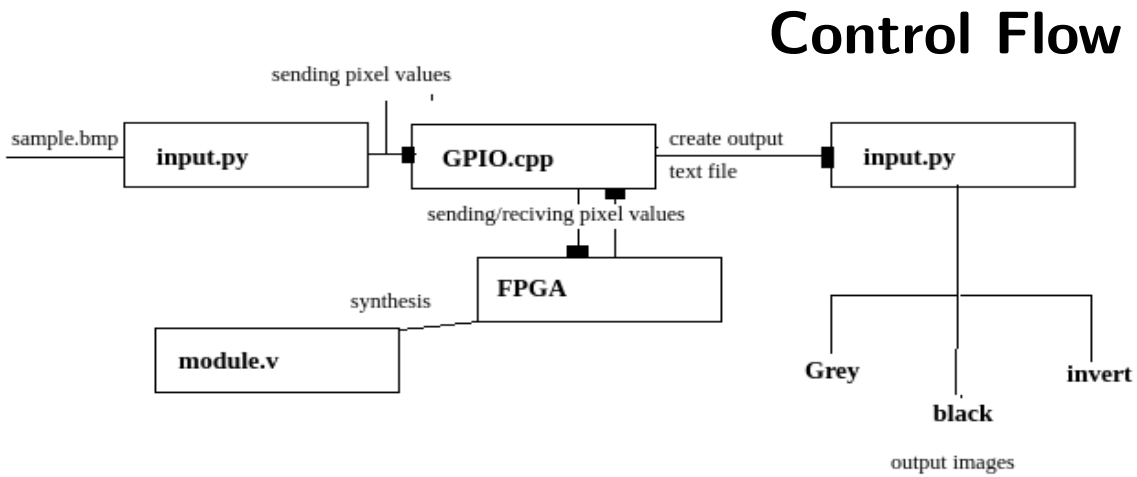
Contents

Setup

- Download and flash the Raspbian image and install ico-tools[1][2].
- Make sure you have all tools required for icoboard namely wiringPi,Icoprog, Icotools, Arachne-pnr, Yosys.
- You can also install tools on existing OS.[3]
- Setup your Icoboard on Raspberrypi.
- Try to synthesis an example module on FPGA using Makefile.[1]

Contents

Project



Files

- a Sample_image
- Python scripts for image reading and writing iamge file.
- C++ file for communication between FPGA and Raspberrypi
- Verilog Module
- Pin Configuration File

Python scripts

- A python file is used to read the sample image and extract the pixel values.
- The output of this file is pipelined as input to the gpio.cpp executable file.
- -1 is sent to indicate end of the input.
- Another python script is used to read the processed output txt file and output the greyscale, black-white, inverted image of the original sample file.

Module

- The module.v is synthesized on FPGA
- A clock signal is sent from Raspberry pi to sync the input and output.
- All signals through RP_IO pins in FPGA.
- All the pixels are received bit by bit in binary form.
- Processing is done on every eighth clock cycles.

module.v

```
module image(
input wire r_to_v_clk,
input wire pred,pblu,pgreen,
output reg grey,black
);
integer counter;
reg [7:0]red;
reg [7:0] green;
reg [7:0] blue;
reg [9:0] grey2;
reg [9:0] grey3;
reg [7:0] grey1;
reg [1:0] det_edge;
initial begin
    counter = 0;
    grey1 =8'b0;
    grey2=8'b0;
    red=8'b0;
    green=8'b0;
    blue=8'b0;
    grey=0;
end

always@(negedge r_to_v_clk)
begin
    red[counter] = pred;
    green[counter] = pgreen;
    blue[counter] = pblu;

    grey=grey1[7-counter];
        counter = counter + 1;
   if(counter == 8) begin
    grey2=(red+green+blue);
    grey1=8'd0;

    grey3=((grey2>>4)+(grey2>>2)+(grey2>>6)+(grey2>>8));
    grey1=grey3;
    grey2=10'd0;
    if(grey3>100) begin
        black=1;
    end
    else begin
         black=0;
    end

   counter = 0;
   end
end
endmodule
```

GPIO.cpp code

- Clock signal is generated and sent via this file to FPGA.
- It reads pixel values via the previous python code and converts them to binary.
- Each bit for each colour is sent using gpio pins on negative value of clock.
- Then output from FPGA is read, and then clock is again set to high.
- The input(binary) is processed and stored in a text file to be read by python scripts.

gpio.cpp

```cpp
#include <stdio.h>
#include <wiringPi.h>
#include <signal.h>
#include <iostream>
#include <atomic>
using namespace std;

int main(void){
  if (wiringPiSetup () == -1)
    return 1 ;
  std::atomic<bool> clock (false);
  pinMode (1, OUTPUT);
  pinMode (4,OUTPUT);
  pinMode (27,OUTPUT);
  pinMode(28,OUTPUT);
  pinMode (29,INPUT);
  pinMode(24,INPUT);
  FILE *ptr=fopen("black.txt","wb");

int value = 0,grey=0,value1=0;
int j=0;
long int c=3;
int cp=0,a=2;
int r=0,g,b;
    while (r!=-1)
    {
        j++;
        std::cin>>r;
        std::cin>>g;
        std::cin>>b;
         if(r!=-1)
         {
            for(int i=0;i<8;i++)
            {
                digitalWrite(27,r%2);r=r>>1;
                digitalWrite(4,b%2);b=b>>1;
                digitalWrite(28,g%2);g=g>>1;
              digitalWrite(1,clock);
              clock=!clock;
              digitalWrite(1,clock);
              clock=!clock;
              value = digitalRead(29);
              grey=(grey<<1)+value;
            }
             value1 = digitalRead(24);
             fprintf(ptr,"%d\n",value1);
             printf("%d\n",grey);
             grey=0;
          }

```

Communication

- To get the GPIO pins and corresponding RP_io pins, use wiringPi pin numbers to find the gpio pin number, run **"gpio readall"** in your terminal, check for corresponding RP_io pin number using schematic table for icoboard[4].
- Using GPIO pins directly for communication increased the data communication rate effectively.
- Use the RP_io pin name in pcf file accordingly.

Contents

How to run

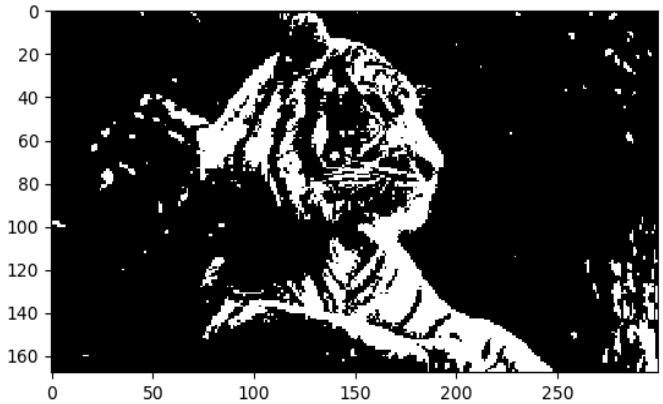
Commands

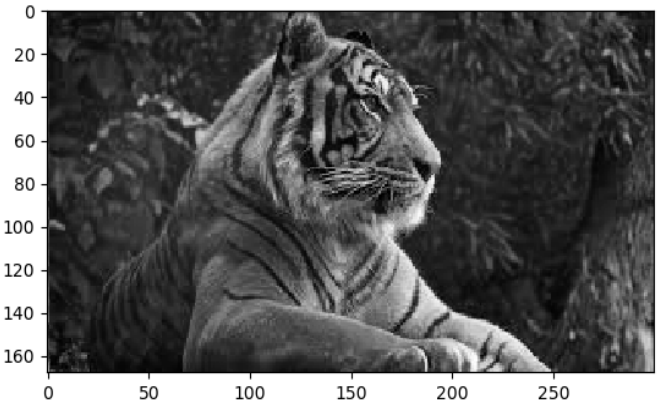
- make v_fname=image
- g++ GPIO.cpp -lwiringPi -std=c++11

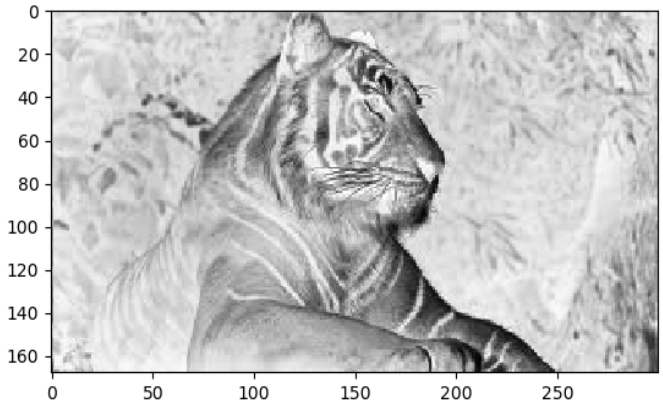
- python input.py ||./*a.out grey.txtpythonimage_cons.py*

Contents

Result


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

Contents

Further projects

- Our projects deals with pixel operation, focusing communication using raspberry pi only, which can parallelized using all the input pins to input many pixels at a time.
- This make it enable to run group-pixel operation like image detection using sobel algorithm etc also.
- Creation of sram need some more research but can be helpful for things operation like image compression.

Biblography I


Get started with icoboard guide.
http://icoboard.org/get-started-with-your-icoboard-and-a-raspi.html.


wiringPi GPIO interface library for RaspberryPI.
http://wiringpi.com/download-and-install/.


Project IceStorm.
http://www.clifford.at/icestorm/.


Trenz-electronic.
Schematics of icoboard.
http://www.trenz-electronic.de/fileadmin/docs/Trenz_Electronic/Modules_and_Module_Carriers/special/TE0887/REV03/Documents/SCH-TE0887-03.PDF.