CardCounter

Final Project Report

Team Members:
Christos Savvopoulos
Hugh Gordon
Nathan Rogan

Table of Contents

1. Introduction

We have written a playing card recognition system using the CCD camera provided with the DE2 board. Using camera input and specialized histogramming techniques, our project detects the value and suite of a given playing card. Given 10 minutes of training, our system will work with a large variety of different playing cards.

The image data comes from the CCD camera into the DE2 board as a stream of pixels. As each pixel is received, its color is determined using a set of hard coded thresholds. Each time a pixel of a given color arrives, a register corresponding to that color is incremented. Once the entire picture is received, the software can use the collected data to determine how many pixels of each color are in the picture.

Each playing card has a unique number of colored pixels on it. The software program that we wrote on top of the NIOS processor uses this data to determine which card has been placed. While this method is not 100% effective, mainly due to noisy data from the CCD camera, using range and averaging techniques has allowed us to achieve very high rates of accuracy in card recognition.

It is worth noting that throughout the development process we implemented unsuccessfully various solutions to solve different issues that had arisen. Some of these failed solutions and the reasoning behind there implementation as well as their failure are included in Appendix A.

2. Project Design & System Architecture

i. Design

When we set off to design this system we had two goals in mind: recognizing cards by counting pixels and separating the tasks better suited to software or hardware. Some parts of the design are inherently hardware, such as the CCD Capture logic and the VGA output. Other tasks, such as an algorithm that recognizes cards are clearly better done in software. To count the number of pixels of a certain color, there are multiple approaches. In the end, we decided to implement the system such that as the CCD_Capture unit reads the data, they are counted by multiple histogram components. When a new frame arrives, the counter is stored in a register.

On the software side of things, we want to know the number of pixels for each color. The communication between the two worlds happens through the Avalon bus. In addition to that, to make the system more flexible we decided to give the program the power to define the color of each counter on the fly. This is done specifying a minimum and maximum red, green, and blue component.

Not only did this save a huge amount of compiling time (since software compiles a lot faster than hardware) but it also made the system a lot more expandable, since different algorithms could try to change the thresholds on the fly.

ii. Components

CCD_Capture:

Reads from the CCD Camera, and outputs the data component of the currently read pixel.
Clock speed: 25 MHZ
Input: Clock, GPIO_1
Output: data (10 bits), new frame flag, new pixel flag

RAW2RGB:

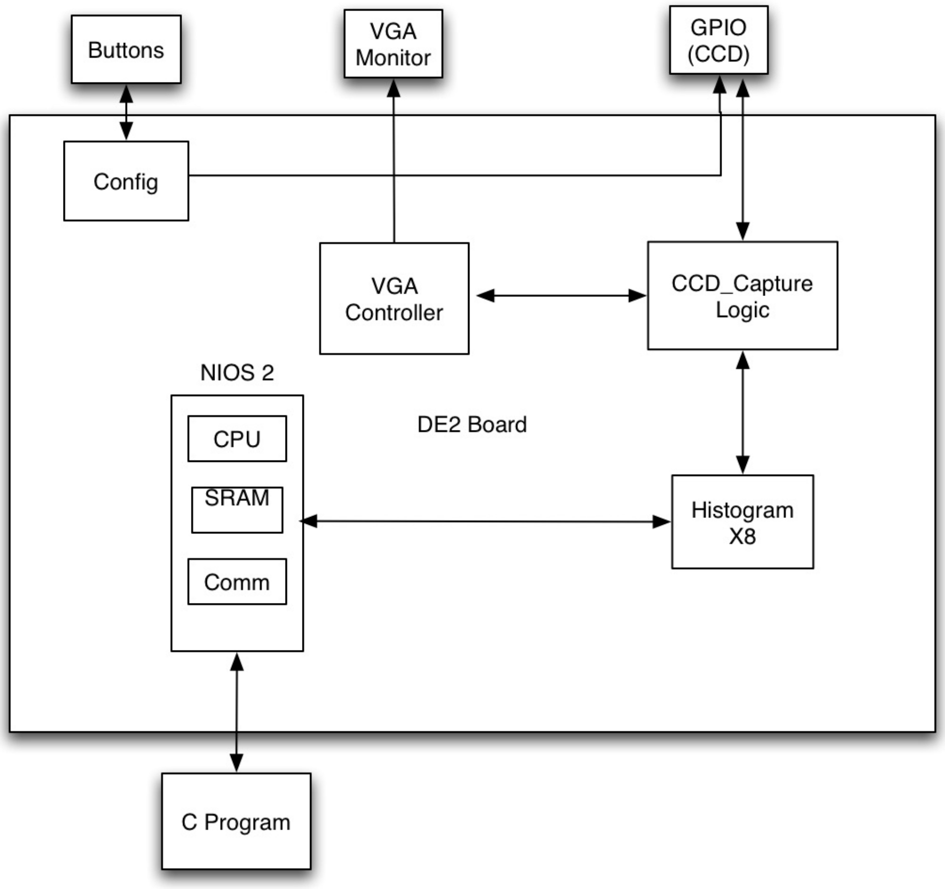
The CCD is a 2D Array of *either* red, green, or blue sensors. This component does some basic image processing (averaging pixels) and gives out the red, green and blue components for each pixel.
Input: pixel clock, data (10 bits), X & Y position
Output: red, green, blue (10 bits each)

Histogram:

This histogram logic takes in the stream of RGB data from the camera logic. Each module (there are 8) is configurable from the software to count up pixels that match a given set of tolerances placed on the red, green and blue values. For instance, to match red, one could set the R tolerance to be between 700 and 1023 and the G and B tolerances to be from 0 to 300. The suitability of these values is highly dependent on the environment.

Input: min and max threshold (32 bits each)

Output: Count (32 bit wide) of how many pixels fit the RGB constraints



VGA Controller:

The VGA controller takes the data from the camera logic and turns it into a VGA signal so that it can be displayed on a normal computer monitor.

Input: r, g, b: 10bits each

Output: VGA signals

Config and Buttons:

The buttons were configured as input to control the settings on the camera. We needed a way to adjust the exposure and turn the camera on and off. The buttons on the DE2 board provided a simple solution.

NIOS 2e:

The standard processor used with the DE2 board. The edition we used had a 50MHZ clock, 524K of onboard SRAM and used the JTAG UART system to communicate with the terminal.

Communication:

This component, as the name suggests, was responsible for communicating data between the hardware part and the software part of our solution. It uses the Avalon bus to perform 32 bit reads (accum) and writes (min, max) from/to the hardware.
Input: 8 x accum (32 bits each)
Output: 8 x min, max (32 bits each)

iii. Software

Aside from counting the pixels of various colors, this is where all the image processing got accomplished. The software program has two modes of operation. The first mode is the learning mode. In this mode, the user is asked to place cards down and type in their values (i.e. king of spades or 2 of clubs). The software then records range of values that are read for number of red and black pixels over a large number of frames. These two ranges (one for red and one for black) are unique for each card. Once the ranges are paired with a given card, that card can be recognized. The second mode of operation is the reorganization mode. In this mode, the software accesses its previously learned data and attempts to match a given card to it. In this case, instead of taking the range of values over a large number of frames, it takes the average of these values and looks to see weather or not this average fits inside each given remembered range. If it does, the corresponding card value is printed out, otherwise, it tries again. Pseudo code is provided below.

pseudocode

```
setThresholds();

//Learning phase
while(1) {
      read card number and suit from user
      if user enters 0 as number, proceed to next phase
      take 2000 readings from camera
      store the minimum and maximum results for each color, under the
card index
}

//Recognizing phase
```

```
while(1) {
      take 1000 readings and average the results for each color
      for each card X that we learned in the previous phase
            if the averages are within the ranges of X
                  keep the card that is the best fit*
      print that card
}
```

*falls most closely in the centre of the range

3. Implementation Issues

i. Hardware & Interfacing

During the proposal period, we decided to break the project into three separate pieces: interfacing with camera, hardware processing and software. We assumed that interfacing with the software would be simple, as we had already done that previously.

This approach worked, until we tried to combine software and hardware. With the timing differences between the camera logic and the processor, the software would not upload on the DE2 board. The camera module runs at 25MHz. In addition to that, an inverted version of that clock is also used that can optionally have an offset. The processor, as well as the rest of the system, runs at 50MHz.

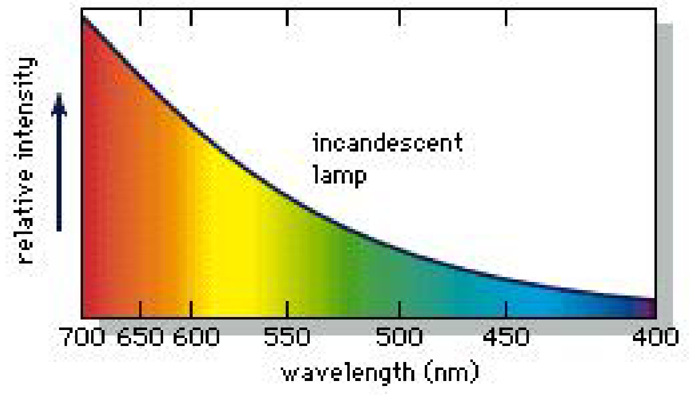
The problem being that we could not upload the software, we tried many different approaches. First, we tried to follow the lab3 tutorial over and over again, making sure we added every NIOS_System component correctly. The next approach was to strip down the system, component by component so as to find the cause of the trouble. Though, we were able to confine the problem within ~30 lines of code, we were not able to resolve the problem. Having tracked down the problem, we experimented with re-writing the obtained code to VHDL, and thalso with generating the NIOS System in Verilog.

Our next approach was to research how the NIOS system works and try to track down online discussions from people who had the same issues. The consensus seemed to be the reason we mention above, namely the timing differences. A careful review of the info/warnings produced by Quartus confirmed this.

The multiple clocks in a single module solution could have been solved in two ways. Add a method to align clock skew consistently over the module or to remove the component from the NIOS system and connect the two in the top level component. We chose the latter.

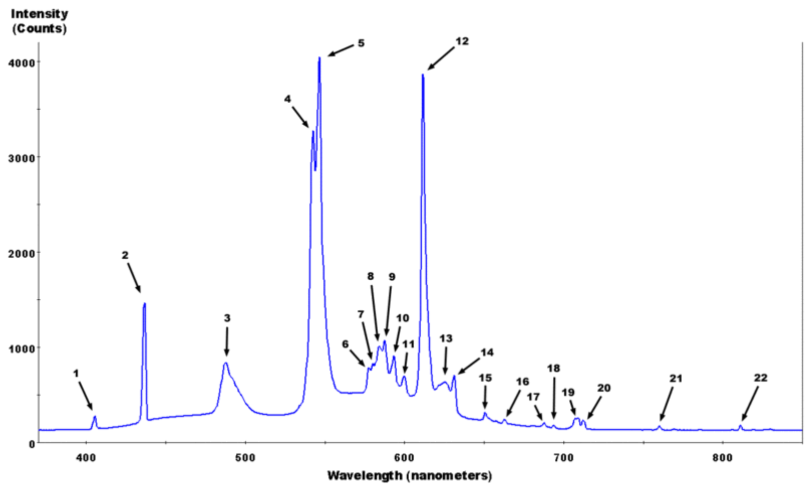
ii. Lighting

With any form of computer vision, obtaining good lighting conditions is paramount. In this project, it was especially important that we had reproducible and optimal lighting conditions. To this end, we researched and tested numerous different lighting choices to find one that would work optimally for us. A central concern was whether to use incandescent light or fluorescent light. Through our research, we learned that the issue at hand was obtaining a proper white balance for our CCD camera. We learned that incandescent light has a waveform that would offset our image in a negative way:



From http://users.mis.net/~pthrush/lighting/glow.html

As one can see from the above diagram, incandescent light has a lot of red in it. This red interfered significantly with our ability to detect red in cards. Fluorescent lamps however proved to be much more evenly spread:



From
http://en.wikipedia.org/wiki/Image:Fluorescent_lighting_spectrum_peaks_labelled.gif

The above spectra is from a typical fluorescent lamp, as can be seen, the peaks are much more in the green and blue regions, which could easily ignore. We also realized that multiple light sources were significantly better than one as many light sources delivered much more uniform light to the face of the card. Having uniform light hitting the card is very important. It means that moving the card does not affect the intensity perceived by the camera.

iii. Physical

Similar to lighting, we had a lot of physical issues with this project. It turned out that the placement of the camera, and the isolation of the camera-card system was highly important. Again, we did significant testing in this area, trying everything from a wood box with a green felt bottom to a cardboard box lit with Christmas lights to an opaque cake cover with a hole in the top for the camera.

4. Lessons Learned

i. Chris

Contribution:

- Designed the system and the specifics of the hardware architecture.
- Developed camera and communication logic that ultimately allowed for successful use of the Avalon bus.
- Developed C interface.

Lesson: This project taught me the importance of being patient with other team members. There were some times when I thought that the voiced ideas were wrong, but upon further consideration proved insightful. Also, when comparing this experience with the PLT project, I can understand how important it is to appoint a team leader. Finally, I wish we had researched the workings of NIOS more systematically. We could have avoided a lot of time-consuming experimentation if we had done so.

ii. Hugh

Contribution:

- Helped with VHDL.
- Built a stabilized box which allowed for optimal data acquisition.

Lesson: Doing something this low level in system design was a real eye opener. Besides sharpening my VHDL skills and broadening my understanding of digital systems, I was shown the importance of good communication and planning amongst team members. We could have planned and shared out the project much more effectively. We were sometimes working on the same thing. Good planning and communication could have made us much more effective as a working unit. Finally, I learned that the lower level you are, the more time you need to leave for debugging. Debugging this took orders of magnitude more time than my most complex operating systems endeavors.

iii. Nate

Contribution:

- Tested various algorithms for suitability.
- Extensively researched cause of failure with the Avalon and tried different hardware configurations.
- Developed C program.

Lesson: The main thing I learned in this project was that a clear review of a system on paper is worth days in development time. The ability to understand the tools design implementation from the top level allows for a cleaner solution and a less buggy implementation. While I did take a systematic debugging approach when our system was show to have a critical flaw I did not have enough sense to stop and reanalyze the system. Working with team members gave a fresh review of the system and eventually led to the solution from Chris. There were some fun times when things went well. There were other times when the tension was so thick it was palpable. A calm demeanor and patience as hard as it was to obtain at those times led to a better working environment.

5. Code

The code for the following components follows in the order shown below. They were printed to .pdf from the editors of these files, and then appended to the final pdf (for highlighting).

i. C code
ii. lab5 (top-level entity)
iii. histo (histogram counter)
iv. communication
v. CCD_Capture (edited)

Edited: new frame flag, used by the accumulators.

```c
/*
 * "Hello World" example.
 *
 * This example prints 'Hello from Nios II' to the STDOUT stream. It runs on
 * the Nios II 'standard', 'full_featured', 'fast', and 'low_cost' example
 * designs. It runs with or without the MicroC/OS-II RTOS and requires a STDOUT
 * device in your system's hardware.
 * The memory footprint of this hosted application is ~69 kbytes by default
 * using the standard reference design.
 *
 * For a reduced footprint version of this template, and an explanation of how
 * to reduce the memory footprint for a given application, see the
 * "small_hello_world" template.
 *
 */
#include <io.h>
#include <system.h>
#include <stdio.h>

typedef unsigned int uint;
#define N 2

uint pack(uint red, uint green, uint blue) {
    return ((red&0x3FF)<<20)
        | ((green&0x3FF)<<10)
        | ((blue&0x3FF));
}

void put(int n, int minR, int minG, int minB, int maxR, int maxG, int maxB) {
    IOWR_32DIRECT(COMMUNICATION_0_BASE, 4*n*2,     pack(minR, minG, minB));
    IOWR_32DIRECT(COMMUNICATION_0_BASE, 4*(n*2+1), pack(maxR, maxG, maxB));
}

uint get(int n) {
    return IORD_32DIRECT(COMMUNICATION_0_BASE, 4*n);
}

struct range {
    uint min,max;
};

struct range db[15][4][2]; // { spades, clubs, hearts, diamonds } * 14 cards * {redRange,
blackRange}
int vv[15][4];

int cardIndex(char c) {
    switch(c) {
        case 's': case 'S': return 0;
        case 'c': case 'C': return 1;
        case 'h': case 'H': return 2;
        case 'd': case 'D': return 3;
        default: return -1;
    }
}
const char cardLookup[4] = { 'S', 'C', 'H', 'D' };

uint min(uint a, uint b) {
    if(a<b)
        return a;
    else
        return b;
}
```

```
uint max(uint a, uint b) {
    if(a>b)
        return a;
    else
        return b;
}

uint sqClamp(uint a) {
    if(a>=1000)
        return 1000*1000;
    else
        return a*a;
}

void multiGet(struct range col[8], int n) {
    int val,i,j;

    for(i=0;i<8;i++) {
        col[i].min=0xffffffff;
        col[i].max=0;
    }

    //printf("Reading");
    for(i=0; i<1000;i++) {
        usleep(5000);

        for(j=0;j<8;j++) {
            val=get(j);
            col[j].min=min(val,col[j].min);
            col[j].max=max(val,col[j].max);
        }
    }
    //printf(" Done!\n");
}

void waitStable() {
    struct range col[8];
    int i;
    while(1) {
        multiGet(col,300);
        for(i=0;i<8;i++)
            if(col[i].max-col[i].min > 1000)
                break;
        return;
    }
}

int main()
{
    int n,s,i,j;
    uint red,black;
    char c;
    struct range col[8];

    memset(vv,0,sizeof(int)*15*4);

    put(0, 700,0,0, 1023,300,300);
    put(1, 0,0,0,  300,300,300);


    while(1) {
        printf("Please enter card number (1-14) followed by [s]pade, [c]lub, [h]eart,
[d]iamond, and press enter to 'teach' that card: ");
        scanf("%i%c",&n,&c);
        s=cardIndex(c);

        if(n==0)
            break;

        printf("Reading... ");
        multiGet(col,1000);
        printf("Done!\n");
        db[n][s][0]=col[0];
        db[n][s][1]=col[1];
        vv[n][s]=1;

        db[n][s][0].min*=0.96; db[n][s][0].max*=1.04;
        db[n][s][1].min*=0.96; db[n][s][1].max*=1.04;

        printf("red=(%u,%u) black=(%u,%u)\n", db[n][s][0].min,db[n][s][0].max,db[n][s]
[1].min,db[n][s][1].max);
    }


    while(1) {
        waitStable();
        printf("Recognizing... ");

        const int samples = 1000;
        uint i,red=0,black=0;
        for(i=0; i<samples; i++) {
            usleep(5000);
            red+=get(0);
            black+=get(1);
        }
        red/=samples;
        black/=samples;
        //printf("Red=%u Black=%u\n",red,black);

        uint closest=0xffffffff;
        uint closeN=0, closeS=0;
        for(i=1;i<=14;i++) {
            for(j=0;j<4;j++) {
                if(!vv[i][j])
                    continue;
                if(db[i][j][0].min <= red && red <= db[i][j][0].max &&
                      db[i][j][1].min <= black && black <= db[i][j][1].max) {
                    uint redAvg = (db[i][j][0].min+db[i][j][0].max)/2;
                    uint blackAvg=(db[i][j][1].min+db[i][j][1].max)/2;

                    if( sqClamp(red-redAvg)+sqClamp(black-blackAvg) < closest) {
                        closest=sqClamp(red-redAvg)+sqClamp(black-blackAvg);
                        closeN=i; closeS=j;
                    }
                }
            }
        }
        if(closeN)
                printf("%u %c\n", closeN, cardLookup[closeS]);
    }

    return 0;
}
```

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity lab5 is

  port (
    signal CLOCK_50 : in std_logic;                -- 50 MHz clock

    SRAM_DQ : inout std_logic_vector(15 downto 0); -- Data bus 16 Bits
    SRAM_ADDR : out std_logic_vector(17 downto 0); -- Address bus 18 Bits
    SRAM_UB_N,                                     -- High-byte Data Mask
    SRAM_LB_N,                                     -- Low-byte Data Mask
    SRAM_WE_N,                                     -- Write Enable
    SRAM_CE_N,                                     -- Chip Enable
    SRAM_OE_N : out std_logic;                      -- Output Enable

    KEY : in std_logic_vector(3 downto 0);         -- Push buttons
    SW : in std_logic_vector(17 downto 0);         -- DPDT switches

    HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7 -- 7-segment displays
       : out std_logic_vector(6 downto 0);
    LEDG : out std_logic_vector(8 downto 0);       -- Green LEDs
    LEDR : out std_logic_vector(17 downto 0);      -- Red LEDs
    GPIO_1 : inout std_logic_vector(35 downto 0);

    -- SDRAM
    DRAM_DQ : inout std_logic_vector(15 downto 0); -- Data Bus
    DRAM_ADDR : out std_logic_vector(11 downto 0); -- Address Bus
    DRAM_LDQM,                                     -- Low-byte Data Mask
    DRAM_UDQM,                                     -- High-byte Data Mask
    DRAM_WE_N,                                     -- Write Enable
    DRAM_CAS_N,                                    -- Column Address Strobe
    DRAM_RAS_N,                                    -- Row Address Strobe
    DRAM_CS_N,                                     -- Chip Select
    DRAM_BA_0,                                     -- Bank Address 0
    DRAM_BA_1,                                     -- Bank Address 0
    DRAM_CLK,                                      -- Clock
    DRAM_CKE : out std_logic;                      -- Clock Enable

    -- VGA output
    VGA_CLK,                                           -- Clock
    VGA_HS,                                            -- H_SYNC
    VGA_VS,                                            -- V_SYNC
    VGA_BLANK,                                         -- BLANK
    VGA_SYNC : out std_logic;                          -- SYNC
    VGA_R,                                             -- Red[9:0]
    VGA_G,                                             -- Green[9:0]
    VGA_B : out std_logic_vector(9 downto 0)            -- Blue[9:0]

  );

end lab5;

architecture rtl of lab5 is

    component nios_system is    port (
              -- 1) global signals:
                 signal clk : IN STD_LOGIC;
                 signal reset_n : IN STD_LOGIC;

              -- the_communication_0
                 signal Count0_to_the_communication_0 : IN STD_LOGIC_VE
CTOR (31 DOWNTO 0);
                 signal Count1_to_the_communication_0 : IN STD_LOGIC_VE
CTOR (31 DOWNTO 0);
                 signal Count2_to_the_communication_0 : IN STD_LOGIC_VE
CTOR (31 DOWNTO 0);
                 signal Count3_to_the_communication_0 : IN STD_LOGIC_VE
CTOR (31 DOWNTO 0);
                 signal Count4_to_the_communication_0 : IN STD_LOGIC_VE
CTOR (31 DOWNTO 0);
                 signal Count5_to_the_communication_0 : IN STD_LOGIC_VE
CTOR (31 DOWNTO 0);
                 signal Count6_to_the_communication_0 : IN STD_LOGIC_VE
CTOR (31 DOWNTO 0);
                 signal Count7_to_the_communication_0 : IN STD_LOGIC_VE
CTOR (31 DOWNTO 0);
                 signal max0_from_the_communication_0 : OUT STD_LOGIC_V
ECTOR (31 DOWNTO 0);
                 signal max1_from_the_communication_0 : OUT STD_LOGIC_V
ECTOR (31 DOWNTO 0);
                 signal max2_from_the_communication_0 : OUT STD_LOGIC_V
ECTOR (31 DOWNTO 0);
                 signal max3_from_the_communication_0 : OUT STD_LOGIC_V
ECTOR (31 DOWNTO 0);
                 signal max4_from_the_communication_0 : OUT STD_LOGIC_V
ECTOR (31 DOWNTO 0);
                 signal max5_from_the_communication_0 : OUT STD_LOGIC_V
ECTOR (31 DOWNTO 0);
                 signal max6_from_the_communication_0 : OUT STD_LOGIC_V
ECTOR (31 DOWNTO 0);
                 signal max7_from_the_communication_0 : OUT STD_LOGIC_V
ECTOR (31 DOWNTO 0);
                 signal min0_from_the_communication_0 : OUT STD_LOGIC_V
ECTOR (31 DOWNTO 0);
                 signal min1_from_the_communication_0 : OUT STD_LOGIC_V
ECTOR (31 DOWNTO 0);
                 signal min2_from_the_communication_0 : OUT STD_LOGIC_V
ECTOR (31 DOWNTO 0);
                 signal min3_from_the_communication_0 : OUT STD_LOGIC_V
ECTOR (31 DOWNTO 0);
                 signal min4_from_the_communication_0 : OUT STD_LOGIC_V
ECTOR (31 DOWNTO 0);
                 signal min5_from_the_communication_0 : OUT STD_LOGIC_V
```

```
ECTOR (31 DOWNTO 0);
                  signal min6_from_the_communication_0 : OUT STD_LOGIC_V
ECTOR (31 DOWNTO 0);
                  signal min7_from_the_communication_0 : OUT STD_LOGIC_V
ECTOR (31 DOWNTO 0);

               -- the_sram
                  signal SRAM_ADDR_from_the_sram : OUT STD_LOGIC_VECTOR
(17 DOWNTO 0);
                  signal SRAM_CE_N_from_the_sram : OUT STD_LOGIC;
                  signal SRAM_DQ_to_and_from_the_sram : INOUT STD_LOGIC_
VECTOR (15 DOWNTO 0);
                  signal SRAM_LB_N_from_the_sram : OUT STD_LOGIC;
                  signal SRAM_OE_N_from_the_sram : OUT STD_LOGIC;
                  signal SRAM_UB_N_from_the_sram : OUT STD_LOGIC;
                  signal SRAM_WE_N_from_the_sram : OUT STD_LOGIC
               );
    end component;

component histo is port (
    clk          : in  std_logic; --50MHz
    R            : in  std_logic_vector(9 downto 0);
    G            : in  std_logic_vector(9 downto 0);
    B            : in  std_logic_vector(9 downto 0);
    new_pixel    : in  std_logic;
    new_frame    : in  std_logic;
    Min          : in  std_logic_vector(31 downto 0);
    Max          : in  std_logic_vector(31 downto 0);
    count        : out std_logic_vector(31 downto 0) );
end component;

component VGA_Controller is port (
    signal iRed : in std_logic_vector(9 downto 0);
    signal iGreen : in std_logic_vector(9 downto 0);
    signal iBlue : in std_logic_vector(9 downto 0);
    signal oRequest : out std_logic;
    --  signal VGA Side
    signal oVGA_R : out std_logic_vector(9 downto 0);
    signal oVGA_G : out std_logic_vector(9 downto 0);
    signal oVGA_B : out std_logic_vector(9 downto 0);
    signal oVGA_H_SYNC : out std_logic;
    signal oVGA_V_SYNC : out std_logic;
    signal oVGA_SYNC : out std_logic;
    signal oVGA_BLANK : out std_logic;
    signal oVGA_CLOCK : out std_logic;
    --  Control Signal
    signal iCLK : in std_logic;
    signal iRST_N : in std_logic );
end component;

component Reset_Delay is port (
    signal iCLK  : in  std_logic;
    signal iRST  : in  std_logic;
    signal oRST_0: out std_logic;
    signal oRST_1: out std_logic;
    signal oRST_2: out std_logic );
end component;
```

```

component CCD_Capture is port (
    signal oDATA        : out std_logic_vector( 9 downto 0);
    signal oDVAL        : out std_logic;
    signal oX_Cont      : out std_logic_vector(10 downto 0);
    signal oY_Cont      : out std_logic_vector(10 downto 0);
    signal oFrame_Cont  : out std_logic_vector(31 downto 0);
    signal oNewFrame    : out std_logic;
    signal iDATA        : in  std_logic_vector( 9 downto 0);
    signal iFVAL        : in  std_logic;
    signal iLVAL        : in  std_logic;
    signal iSTART       : in  std_logic;
    signal iEND         : in  std_logic;
    signal iCLK         : in  std_logic;
    signal iRST         : in  std_logic  );
end component;

component RAW2RGB is port (
    signal oRed     : out std_logic_vector( 9 downto 0);
    signal oGreen   : out std_logic_vector( 9 downto 0);
    signal oBlue    : out std_logic_vector( 9 downto 0);
    signal oDVAL    : out std_logic;
    signal iX_Cont  : in  std_logic_vector(10 downto 0);
    signal iY_Cont  : in  std_logic_vector(10 downto 0);
    signal iDATA    : in  std_logic_vector( 9 downto 0);
    signal iDVAL    : in  std_logic;
    signal iCLK     : in  std_logic;
    signal iRST     : in  std_logic     );
end component;

component Sdram_Control_4Port is port (
    --  HOST Side
    signal REF_CLK          : in  std_logic;
    signal RESET_N          : in  std_logic;
    --  FIFO Write Side 1
    signal WR1_DATA         : in  std_logic_vector(15 downto 0);
    signal WR1              : in  std_logic;
    signal WR1_ADDR         : in  std_logic_vector(23 downto 0);
    signal WR1_MAX_ADDR     : in  std_logic_vector(23 downto 0);
    signal WR1_LENGTH       : in  std_logic_vector( 8 downto 0);
    signal WR1_LOAD         : in  std_logic;
    signal WR1_CLK          : in  std_logic;
    signal WR1_FULL         : out std_logic;
    signal WR1_USE          : out std_logic_vector( 8 downto 0);
    --  FIFO Write Side 2
    signal WR2_DATA         : in  std_logic_vector(15 downto 0);
    signal WR2              : in  std_logic;
    signal WR2_ADDR         : in  std_logic_vector(23 downto 0);
    signal WR2_MAX_ADDR     : in  std_logic_vector(23 downto 0);
    signal WR2_LENGTH       : in  std_logic_vector( 8 downto 0);
    signal WR2_LOAD         : in  std_logic;
    signal WR2_CLK          : in  std_logic;
    signal WR2_FULL         : out std_logic;
    signal WR2_USE          : out std_logic_vector( 8 downto 0);
    --  FIFO Read Side 1
    signal RD1_DATA         : out std_logic_vector(15 downto 0);
    signal RD1              : in  std_logic;
    signal RD1_ADDR          : in  std_logic_vector(23 downto 0);
    signal RD1_MAX_ADDR      : in  std_logic_vector(23 downto 0);
    signal RD1_LENGTH        : in  std_logic_vector( 8 downto 0);
    signal RD1_LOAD          : in  std_logic;
    signal RD1_CLK           : in  std_logic;
    signal RD1_EMPTY         : out std_logic;
    signal RD1_USE           : out std_logic_vector( 8 downto 0);
    --  FIFO Read Side 2
    signal RD2_DATA          : out std_logic_vector(15 downto 0);
    signal RD2               : in  std_logic;
    signal RD2_ADDR          : in  std_logic_vector(23 downto 0);
    signal RD2_MAX_ADDR      : in  std_logic_vector(23 downto 0);
    signal RD2_LENGTH        : in  std_logic_vector( 8 downto 0);
    signal RD2_LOAD          : in  std_logic;
    signal RD2_CLK           : in  std_logic;
    signal RD2_EMPTY         : out std_logic;
    signal RD2_USE           : out std_logic_vector( 8 downto 0);
    --  SDRAM Side
    signal SA                : out std_logic_vector(11 downto 0);
    signal BA                : out std_logic_vector( 1 downto 0);
    signal CS_N              : out std_logic;
    signal CKE               : out std_logic;
    signal RAS_N             : out std_logic;
    signal CAS_N             : out std_logic;
    signal WE_N              : out std_logic;
    signal DQ                : inout std_logic_vector(15 downto 0);
    signal DQM               : out std_logic_vector( 1 downto 0);
    signal SDR_CLK           : out std_logic     );
end component;


component SEG7_LUT_8 is port (
    signal oSEG0,oSEG1,oSEG2,oSEG3,oSEG4,oSEG5,oSEG6,oSEG7
                : out std_logic_vector( 6 downto 0);
    signal iDIG : in  std_logic_vector(31 downto 0)     );
end component;

component I2C_CCD_Config is port (
        signal iCLK : IN std_logic;
        signal iRST_N : IN std_logic;
        signal iExposure : IN std_logic_vector(15 downto 0);
        signal I2C_SCLK : OUT std_logic;
        signal I2C_SDAT : INOUT std_logic );
end component;

component Mirror_Col is port (  --  Input Side
                    iCCD_R      : in  std_logic_vector(9 downto 0);
                    iCCD_G      : in  std_logic_vector(9 downto 0);
                    iCCD_B      : in  std_logic_vector(9 downto 0);
                    iCCD_DVAL   : in  std_logic;
                    iCCD_PIXCLK : in  std_logic;
                    iRST_N      : in  std_logic;
                    --  Output Side
                    oCCD_R      : out std_logic_vector(9 downto 0);
                    oCCD_G      : out std_logic_vector(9 downto 0);
                    oCCD_B      : out std_logic_vector(9 downto 0);
                    oCCD_DVAL   : out std_logic     );
```

```vhdl
end component;


--  CCD
signal  CCD_DATA : std_logic_vector( 9 downto 0);
signal  CCD_SDAT : std_logic;
signal  CCD_SCLK : std_logic;
signal  CCD_FLASH : std_logic;
signal  CCD_FVAL : std_logic;
signal  CCD_LVAL : std_logic;
signal  CCD_PIXCLK : std_logic;
signal  CCD_MCLK : std_logic;   --  CCD Master Clock
signal  Read_DATA1 : std_logic_vector(15 downto 0);
signal  Read_DATA2 : std_logic_vector(15 downto 0);
signal  VGA_CTRL_CLK : std_logic;
signal  AUD_CTRL_CLK : std_logic;
signal  mCCD_DATA : std_logic_vector( 9 downto 0);
signal  mCCD_DVAL : std_logic;
signal  mCCD_DVAL_d : std_logic;
signal  X_Cont : std_logic_vector(10 downto 0);
signal  Y_Cont : std_logic_vector(10 downto 0);
signal  X_ADDR : std_logic_vector(9 downto 0);
signal  Frame_Cont : std_logic_vector(31 downto 0);
signal  mCCD_R : std_logic_vector(9 downto 0);
signal  mCCD_G : std_logic_vector(9 downto 0);
signal  mCCD_B : std_logic_vector(9 downto 0);
signal  DLY_RST_0 : std_logic;
signal  DLY_RST_1 : std_logic;
signal  DLY_RST_2 : std_logic;
signal  Read : std_logic;
signal  rCCD_DATA : std_logic_vector(9 downto 0);
signal  rCCD_LVAL : std_logic;
signal  rCCD_FVAL : std_logic;
signal  sCCD_R : std_logic_vector(9 downto 0);
signal  sCCD_G : std_logic_vector(9 downto 0);
signal  sCCD_B : std_logic_vector(9 downto 0);
signal  sCCD_DVAL : std_logic;
signal  mNewFrame : std_logic;
signal clk : std_logic;

    signal counter : std_logic_vector(15 downto 0);
    signal reset_n : std_logic;
    signal Count0,Count1,Count2,Count3,Count4,Count5,Count6,Count7 : st
d_logic_vector(31 downto 0);
    signal min0,min1,min2,min3,min4,min5,min6,min7 : std_logic_vector(3
1 downto 0);
    signal max0,max1,max2,max3,max4,max5,max6,max7 : std_logic_vector(3
1 downto 0);

begin

clk <= CLOCK_50;
--  LEDR(17) <= '1';
--  LEDR(16) <= '1';

  process (CLOCK_50)
  begin
```

```
    if CLOCK_50'event and CLOCK_50 = '1' then
      if counter = x"ffff" then
        reset_n <= '1';
      else
        reset_n <= '0';
        counter <= counter + 1;
      end if;
    end if;
  end process;

  nios : nios_system port map (
    clk                          => CLOCK_50,
    reset_n                      => reset_n,
    Count0_to_the_communication_0 => Count0,
    Count1_to_the_communication_0 => Count1,
    Count2_to_the_communication_0 => Count2,
    Count3_to_the_communication_0 => Count3,
    Count4_to_the_communication_0 => Count4,
    Count5_to_the_communication_0 => Count5,
    Count6_to_the_communication_0 => Count6,
    Count7_to_the_communication_0 => Count7,
    min0_from_the_communication_0 => min0,
    max0_from_the_communication_0 => max0,
    min1_from_the_communication_0 => min1,
    max1_from_the_communication_0 => max1,
    min2_from_the_communication_0 => min2,
    max2_from_the_communication_0 => max2,
    min3_from_the_communication_0 => min3,
    max3_from_the_communication_0 => max3,
    min4_from_the_communication_0 => min4,
    max4_from_the_communication_0 => max4,
    min5_from_the_communication_0 => min5,
    max5_from_the_communication_0 => max5,
    min6_from_the_communication_0 => min6,
    max6_from_the_communication_0 => max6,
    min7_from_the_communication_0 => min7,
    max7_from_the_communication_0 => max7,

    --     leds_from_the_leds           => LEDR(15 downto 0),
    SRAM_ADDR_from_the_sram      => SRAM_ADDR,
    SRAM_CE_N_from_the_sram      => SRAM_CE_N,
    SRAM_DQ_to_and_from_the_sram => SRAM_DQ,
    SRAM_LB_N_from_the_sram      => SRAM_LB_N,
    SRAM_OE_N_from_the_sram      => SRAM_OE_N,
    SRAM_UB_N_from_the_sram      => SRAM_UB_N,
    SRAM_WE_N_from_the_sram      => SRAM_WE_N
  );


    --histo
    h0: histo port map(clk,mCCD_R,mCCD_G,mCCD_B,mCCD_DVAL_d,mNewFrame,m
in0,max0,Count0);
    h1: histo port map(clk,mCCD_R,mCCD_G,mCCD_B,mCCD_DVAL_d,mNewFrame,m
in1,max1,Count1);
    h2: histo port map(clk,mCCD_R,mCCD_G,mCCD_B,mCCD_DVAL_d,mNewFrame,m
in2,max2,Count2);
    h3: histo port map(clk,mCCD_R,mCCD_G,mCCD_B,mCCD_DVAL_d,mNewFrame,m
in3,max3,Count3);
    h4: histo port map(clk,mCCD_R,mCCD_G,mCCD_B,mCCD_DVAL_d,mNewFrame,min4,max4,Count4);
    h5: histo port map(clk,mCCD_R,mCCD_G,mCCD_B,mCCD_DVAL_d,mNewFrame,min5,max5,Count5);
    h6: histo port map(clk,mCCD_R,mCCD_G,mCCD_B,mCCD_DVAL_d,mNewFrame,min6,max6,Count6);
    h7: histo port map(clk,mCCD_R,mCCD_G,mCCD_B,mCCD_DVAL_d,mNewFrame,min7,max7,Count7);

    CCD_DATA(0) <=  GPIO_1(0);
    CCD_DATA(1) <=  GPIO_1(1);
    CCD_DATA(2) <=  GPIO_1(5);
    CCD_DATA(3) <=  GPIO_1(3);
    CCD_DATA(4) <=  GPIO_1(2);
    CCD_DATA(5) <=  GPIO_1(4);
    CCD_DATA(6) <=  GPIO_1(6);
    CCD_DATA(7) <=  GPIO_1(7);
    CCD_DATA(8) <=  GPIO_1(8);
    CCD_DATA(9) <=  GPIO_1(9);
    GPIO_1(11)  <=  CCD_MCLK;
--  GPIO_1(15)  <=  CCD_SDAT;
--  GPIO_1(14)  <=  CCD_SCLK;
    CCD_FVAL    <=  GPIO_1(13);
    CCD_LVAL    <=  GPIO_1(12);
    CCD_PIXCLK  <=  GPIO_1(10);
    LEDR        <=  SW;
    LEDG        <=  Y_Cont(8 downto 0);
    VGA_CTRL_CLK<=  CCD_MCLK;


    --halve module
    process (CLOCK_50)
    begin
        if CLOCK_50'event and CLOCK_50 = '1' then
            CCD_MCLK <= not CCD_MCLK;
        end if;
    end process;

    process (CCD_PIXCLK)
    begin
        if CCD_PIXCLK'event and CCD_PIXCLK='1' then
            rCCD_DATA   <=  CCD_DATA;
            rCCD_LVAL   <=  CCD_LVAL;
            rCCD_FVAL   <=  CCD_FVAL;
        end if;
    end process;

    u1 : VGA_Controller port map(
                            iRed => Read_DATA2(9 downto 0),
                            iGreen => Read_DATA1(14 downto 10)&Read_DATA2(14 downto 10),
                            iBlue => Read_DATA1(9 downto 0),
                            oRequest => Read,
                            --  VGA Side
                            oVGA_R => VGA_R,
                            oVGA_G => VGA_G,
                            oVGA_B => VGA_B,
                            oVGA_H_SYNC => VGA_HS,
                            oVGA_V_SYNC => VGA_VS,
                            oVGA_SYNC => VGA_SYNC,
                            oVGA_BLANK => VGA_BLANK,
                            oVGA_CLOCK => VGA_CLK,
                            --  Control Signal
                            iCLK => VGA_CTRL_CLK,
                            iRST_N => DLY_RST_2 );

    u2: Reset_Delay     port map(
                            iCLK => CLOCK_50,
                            iRST => KEY(0),
                            oRST_0 => DLY_RST_0,
                            oRST_1 => DLY_RST_1,
                            oRST_2 => DLY_RST_2 );

    u3: CCD_Capture     port map(
                            oDATA => mCCD_DATA,
                            oDVAL => mCCD_DVAL,
                            oX_Cont => X_Cont,
                            oY_Cont => Y_Cont,
                            oFrame_Cont => Frame_Cont,
                            oNewFrame   => mNewFrame,
                            iDATA => rCCD_DATA,
                            iFVAL => rCCD_FVAL,
                            iLVAL => rCCD_LVAL,
                            iSTART => '1',
                            iEND => not(KEY(2)),
                            iCLK => CCD_PIXCLK,
                            iRST => DLY_RST_1   );

    u4: RAW2RGB         port map(
                            oRed => mCCD_R,
                            oGreen => mCCD_G,
                            oBlue => mCCD_B,
                            oDVAL => mCCD_DVAL_d,
                            iX_Cont => X_Cont,
                            iY_Cont => Y_Cont,
                            iDATA => mCCD_DATA,
                            iDVAL => mCCD_DVAL,
                            iCLK => CCD_PIXCLK,
                            iRST => DLY_RST_1   );

    u5: SEG7_LUT_8      port map(
                            oSEG0 => HEX0,oSEG1 => HEX1,
                            oSEG2 => HEX2,oSEG3 => HEX3,
                            oSEG4 => HEX4,oSEG5 => HEX5,
                            oSEG6 => HEX6,oSEG7 => HEX7,
                            iDIG => Frame_Cont );

    u6: Sdram_Control_4Port port map(
                            --  HOST Side
                            REF_CLK => CLOCK_50,
                            RESET_N => '1',
                            --  FIFO Write Side 1
                            WR1_DATA =>     '0'&sCCD_G(9 downto 5)&sCCD
    _B(9 downto 0),
                            WR1 => sCCD_DVAL,
                            WR1_ADDR => "0000000000000000000000",
                            WR1_MAX_ADDR => "0000101000000000000000000",
                            WR1_LENGTH => "100000000",
                            WR1_LOAD => not(DLY_RST_0),
                            WR1_CLK => CCD_PIXCLK,
                            --  FIFO Write Side 2
                            WR2_DATA => '0'&sCCD_G(4 downto 0)&sCCD_R(9
     downto 0),
                            WR2 => sCCD_DVAL,
                            WR2_ADDR => "0001000000000000000000",
                            WR2_MAX_ADDR => "0001010100000000000000000",
                            WR2_LENGTH => "100000000",
                            WR2_LOAD => not(DLY_RST_0),
                            WR2_CLK => CCD_PIXCLK,
                            --  FIFO Read Side 1
                            RD1_DATA => Read_DATA1,
                            RD1 => Read,
                            RD1_ADDR => "0000000000101000000000000",
                            RD1_MAX_ADDR => "00000000000000001111100000",
                            RD1_LENGTH => "100000000",
                            RD1_LOAD => not(DLY_RST_0),
                            RD1_CLK => VGA_CTRL_CLK,
                            --  FIFO Read Side 2
                            RD2_DATA => Read_DATA2,
                            RD2 => Read,
                            RD2_ADDR => "0001000000101000000000000",
                            RD2_MAX_ADDR => "0001010011011000000000000",
                            RD2_LENGTH => "100000000",
                            RD2_LOAD => not(DLY_RST_0),
                            RD2_CLK => VGA_CTRL_CLK,
                            --  SDRAM Side
                            SA => DRAM_ADDR,
                            BA(1)=>DRAM_BA_1,
                            BA(0)=>DRAM_BA_0,
                            CS_N => DRAM_CS_N,
                            CKE => DRAM_CKE,
                            RAS_N => DRAM_RAS_N,
                            CAS_N => DRAM_CAS_N,
                            WE_N => DRAM_WE_N,
                            DQ => DRAM_DQ,
                            DQM(1) => DRAM_UDQM,
                            DQM(0) => DRAM_LDQM,
                            SDR_CLK => DRAM_CLK );

        u7: I2C_CCD_Config      port map (
                            iCLK => CLOCK_50,
                            iRST_N => KEY(1),
                            iExposure => SW(15 downto 0),
                            I2C_SCLK => GPIO_1(14),
                            I2C_SDAT => GPIO_1(15) );

        u8: Mirror_Col          port map(   --  Input Side
                            iCCD_R => mCCD_R,
                            iCCD_G => mCCD_G,
                            iCCD_B => mCCD_B,
                            iCCD_DVAL => mCCD_DVAL_d,
                            iCCD_PIXCLK => CCD_PIXCLK,
                            iRST_N => DLY_RST_1,
                            --  Output Side
                            oCCD_R => sCCD_R,
                            oCCD_G => sCCD_G,
                            oCCD_B => sCCD_B,
                            oCCD_DVAL => sCCD_DVAL);

end rtl;

```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;



entity histo is port (
    clk          : in  std_logic; --50MHz
    R            : in  std_logic_vector(9 downto 0);
    G            : in  std_logic_vector(9 downto 0);
    B            : in  std_logic_vector(9 downto 0);
    new_pixel    : in  std_logic;
    new_frame    : in  std_logic;
    Min          : in  std_logic_vector(31 downto 0);
    Max          : in  std_logic_vector(31 downto 0);
    count        : out std_logic_vector(31 downto 0) );
end histo;

architecture rtl of histo is

signal zero : std_logic;
signal counter,counter2 : std_logic_vector(31 downto 0);
signal RedMin,RedMax,GreenMin,GreenMax,BlueMin,BlueMax : std_logic_vect
or(9 downto 0);
begin

    count <= counter2;
    RedMin  <= Min(29 downto 20);
    GreenMin<= Min(19 downto 10);
    BlueMin <= Min( 9 downto  0);
    RedMax  <= Max(29 downto 20);
    GreenMax<= Max(19 downto 10);
    BlueMax <= Max( 9 downto  0);


    process (clk)
    begin
        if clk'event and clk='1' then
            if new_frame='0' and zero='1' then
                zero    <='0';
                counter <= "00000000000000000000000000000000";
            end if;

            if new_frame='1' then
                counter2 <= counter;
                zero    <='1';
            end if;

            if new_pixel='1' then
                if  R >= RedMin and R<=RedMax and
                    G >= GreenMin and G<=GreenMax and
                    B >= BlueMin and B<=BlueMax then
                    counter <= counter+1;
                end if;
            end if;
        end if;
    end process;

end rtl;
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity communication is

  port (
    avs_s1_clk        : in  std_logic;
    avs_s1_reset_n    : in  std_logic;
    avs_s1_read       : in  std_logic;
    avs_s1_write      : in  std_logic;
    avs_s1_chipselect : in  std_logic;
    avs_s1_address    : in  std_logic_vector(7 downto 0);
    avs_s1_readdata   : out std_logic_vector(31 downto 0);
    avs_s1_writedata  : in  std_logic_vector(31 downto 0);

    Count0,Count1,Count2,Count3,Count4,Count5,Count6,Count7
        : in  std_logic_vector(31 downto 0);
    min0,max0,min1,max1,min2,max2,min3,max3,min4,max4,min5,max5,min6,max6,min7,max7
        : out std_logic_vector(31 downto 0)
  );

end communication;

architecture rtl of communication is

signal reset : std_logic;
signal  rmin0,rmax0,rmin1,rmax1,rmin2,rmax2,rmin3,rmax3,rmin4,rmax4,rmin5,rmax5,rmin6,rmax6,rmin7,rmax7
        : std_logic_vector(31 downto 0);
signal  rCount0,rCount1,rCount2,rCount3,rCount4,rCount5,rCount6,rCount7
        : std_logic_vector(31 downto 0);

begin
reset<='0';
    min0 <= rmin0; min1 <= rmin1; min2 <= rmin2; min3 <= rmin3; min4 <= rmin4; min5 <= rmin5; min6 <= rmin6; min7 <= rmin7;
    max0 <= rmax0; max1 <= rmax1; max2 <= rmax2; max3 <= rmax3; max4 <= rmax4; max5 <= rmax5; max6 <= rmax6; max7 <= rmax7;
    rCount0 <= Count0; rCount1 <= Count1; rCount2 <= Count2; rCount3 <= Count3; rCount4 <= Count4; rCount5 <= Count5; rCount6 <= Count6; rCount7 <= Count7;

  process (avs_s1_clk)
  begin
    if avs_s1_clk'event and avs_s1_clk = '1' then
      if reset = '1' then
        avs_s1_readdata <= (others => '0');
      else
        if avs_s1_chipselect = '1' then
            if avs_s1_read = '1' then
                if      avs_s1_address="00000000" then avs_s1_readdata <=rCount0;
                elsif   avs_s1_address="00000001" then avs_s1_readdata <=rCount1;
                elsif   avs_s1_address="00000010" then avs_s1_readdata <=rCount2;
                elsif   avs_s1_address="00000011" then avs_s1_readdata <=rCount3;
                elsif   avs_s1_address="00000100" then avs_s1_readdata <=rCount4;
                elsif   avs_s1_address="00000101" then avs_s1_readdata <=rCount5;
                elsif   avs_s1_address="00000110" then avs_s1_readdata <=rCount6;
                elsif   avs_s1_address="00000111" then avs_s1_readdata <=rCount7;
                else                                avs_s1_readdata <="10101010101010101010101010101010";
                end if;
            elsif avs_s1_write = '1' then
                if      avs_s1_address="00000000" then rmin0  <=avs_s1_writedata;
                elsif   avs_s1_address="00000001" then rmax0  <=avs_s1_writedata;
                elsif   avs_s1_address="00000010" then rmin1  <=avs_s1_writedata;
                elsif   avs_s1_address="00000011" then rmax1  <=avs_s1_writedata;
                elsif   avs_s1_address="00000100" then rmin2  <=avs_s1_writedata;
                elsif   avs_s1_address="00000101" then rmax2  <=avs_s1_writedata;
                elsif   avs_s1_address="00000110" then rmin3  <=avs_s1_writedata;
                elsif   avs_s1_address="00000111" then rmax3  <=avs_s1_writedata;
                elsif   avs_s1_address="00001000" then rmin4  <=avs_s1_writedata;
                elsif   avs_s1_address="00001001" then rmax4  <=avs_s1_writedata;
                elsif   avs_s1_address="00001010" then rmin5  <=avs_s1_writedata;
                elsif   avs_s1_address="00001011" then rmax5  <=avs_s1_writedata;
                elsif   avs_s1_address="00001100" then rmin6  <=avs_s1_writedata;
                elsif   avs_s1_address="00001101" then rmax6  <=avs_s1_writedata;
                elsif   avs_s1_address="00001110" then rmin7  <=avs_s1_writedata;
                elsif   avs_s1_address="00001111" then rmax7  <=avs_s1_writedata;
                end if;
            end if;
        end if;
      end if;
    end if;
  end process;

end rtl;

```

```verilog
module CCD_Capture(  oDATA,
                     oDVAL,
                     oX_Cont,
                     oY_Cont,
                     oFrame_Cont,
                     oNewFrame,
                     iDATA,
                     iFVAL,
                     iLVAL,
                     iSTART,
                     iEND,
                     iCLK,
                     iRST    );

input    [9:0]   iDATA;
input            iFVAL;
input            iLVAL;
input            iSTART;
input            iEND;
input            iCLK;
input            iRST;
output   [9:0]   oDATA;
output   [10:0]  oX_Cont;
output   [10:0]  oY_Cont;
output   [31:0]  oFrame_Cont;
output           oNewFrame;
output           oDVAL;
reg              Pre_FVAL;
reg              mCCD_FVAL;
reg              mCCD_LVAL;
reg      [9:0]   mCCD_DATA;
reg      [10:0]  X_Cont;
reg      [10:0]  Y_Cont;
reg      [31:0]  Frame_Cont;
reg              mSTART;
reg              rNewFrame;

assign  oX_Cont     =   X_Cont;
assign  oY_Cont     =   Y_Cont;
assign  oFrame_Cont =   Frame_Cont;
assign  oDATA       =   mCCD_DATA;
assign  oDVAL       =   mCCD_FVAL&mCCD_LVAL;
assign  oNewFrame   =   rNewFrame;

always@(posedge iCLK or negedge iRST)
begin
    if(!iRST)
    mSTART  <=  0;
    else
    begin
        if(iSTART)
        mSTART  <=  1;
        if(iEND)
        mSTART  <=  0;
    end
end
```

```
always@(posedge iCLK or negedge iRST)
begin
    if(!iRST)
    begin
        Pre_FVAL    <=  0;
        mCCD_FVAL   <=  0;
        mCCD_LVAL   <=  0;
        mCCD_DATA   <=  0;
        X_Cont      <=  0;
        Y_Cont      <=  0;
    end
    else
    begin
        Pre_FVAL    <=  iFVAL;
        if( ({Pre_FVAL,iFVAL}==2'b01) && mSTART )
        mCCD_FVAL   <=  1;
        else if({Pre_FVAL,iFVAL}==2'b10)
        mCCD_FVAL   <=  0;
        mCCD_LVAL   <=  iLVAL;
        mCCD_DATA   <=  iDATA;
        if(mCCD_FVAL)
        begin
            if(mCCD_LVAL)
            begin
                if(X_Cont<1279)
                X_Cont  <=  X_Cont+1;
                else
                begin
                    X_Cont  <=  0;
                    Y_Cont  <=  Y_Cont+1;
                end
            end
        end
        else
        begin
            X_Cont  <=  0;
            Y_Cont  <=  0;
        end
    end
end

always@(posedge iCLK or negedge iRST)
begin
    if(!iRST)
    Frame_Cont  <=  0;
    else
    begin
        if( ({Pre_FVAL,iFVAL}==2'b01) && mSTART )
        begin
            Frame_Cont  <=  Frame_Cont+1;
            rNewFrame <= 1'b1;
        end
        else
        begin
            rNewFrame <= 0;
        end
    end
```

```
end
endmodule

```