Infrared Thermography

Final Project Report

December 5, 2017

E155

Lupe Carlos and David Kwan

Abstract:

In this project, the team set out to gather data from an Adafruit AMG8833 8x8 thermal camera sensor to the Raspberry Pi 3 Model B board through I2C communication protocols, expand and blur that data to a 32x32 image, transfer that data from the Raspberry Pi to the FPGA on the μMudd Board IV through SPI communication protocols, and then display the expanded image on a CRT monitor through a VGA connection. In the end, the team was able to properly setup the I2C communication protocols, generate a 32x32 enlarged image from the 8x8 IR sensor data through bilinear interpolation, transfer the data through SPI communication protocols to the FPGA, and then generate the proper signals needed by the CRT monitor and properly connect it through VGA. In doing so, the CRT monitor displayed a 32x32 heat map generated by the 8x8 IR sensor that refreshed at 3.4Hz and could gather the shapes of objects as intricate as fingers.

Introduction:

Thermal imaging cameras have many applications in the real world because they can give multiple temperature readings at once and provide an intuitive platform for those readings. These devices can be applied in fields ranging from firefighters seeing through smoke to noticing air leaks in homes for maximizing heating efficiency.

This project is partitioned into four main stages: sensor data collection, data processing in the Raspberry Pi, communication between Pi and FPGA, and the video driver. Figure 1 depicts the four main hardware modules in participating in the project.

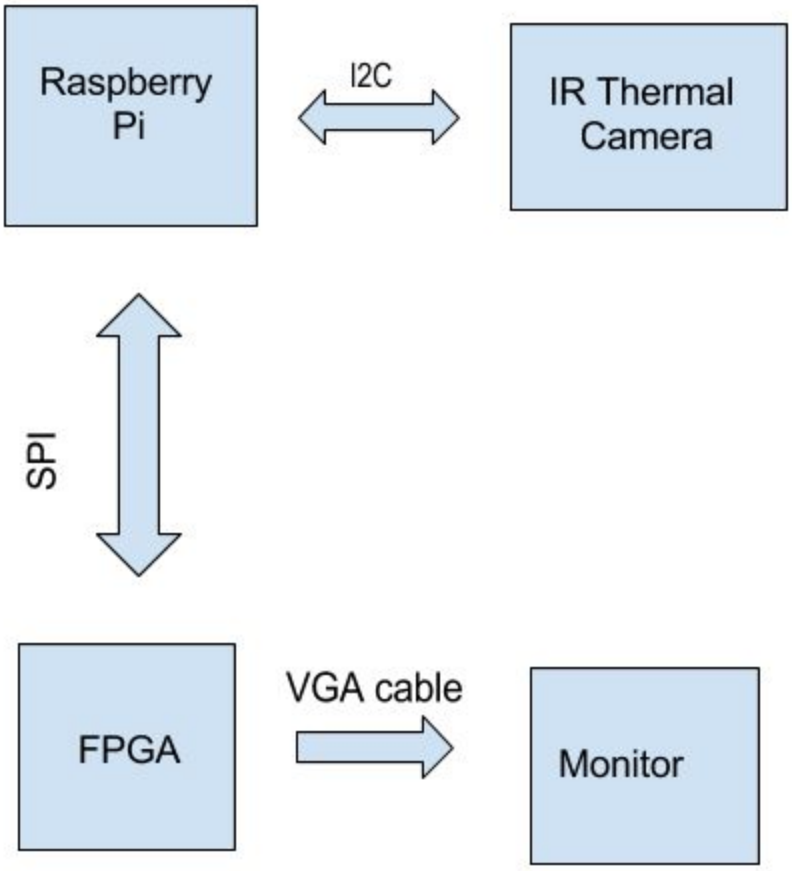


Figure 1. Block diagram of Infrared Thermography system

First, the Raspberry Pi communicates with the thermal camera sensor through I2C to collect temperature readings. Once the Pi has the 8x8 temperature readings, the array is linearly interpolated twice to obtain a 32x32 array of temperature readings. In this system, a single pixel can be described using 4 bits, thus 2 temperature values can be described using a single byte. So, the 32x32 array is then converted into a 16x32 array where each byte describes 2 temperature readings. These values are then sent to the FPGA over SPI. The FPGA simply takes in data from the Pi on the positive edge of the SPI clock that only runs when data is being sent. Then, the FPGA shifts in the necessary data from the Raspberry Pi and uses a series of counters and comparators to properly assign signals that run through all of the pixels on the CRT monitor as well as to gather the correct parts of the intake data to display based

on the which pixel is being colored at the correct time and to display the full set of pixels at 60 frames per second to avoid a choppy-looking image.

New Hardware:

The team has implemented two pieces of new hardware, an 8x8 IR sensor and a CRT monitor.

8x8 Infrared Temperature Sensor

Infrared temperature sensors are able to sense electromagnetic waves in the 700nm to the 14,000nm range. These sensors contain photodetectors that are able to convert infrared energy emitted by heat sources to a proportional electrical signal.

The AMG8833 IR thermal camera works similarly and is compact enough to fit on a breadboard. The sensor has a 60° field of view in the horizontal and vertical and is rated to detect temperatures of 0℃ to 80℃ at distances up to 5m. The sensor will return an 8x8 array of individual infrared temperature readings over I2C. The 0.25℃ resolution and 10 Hz frame rate of the sensor limits the final resolution and framerate of the team's system.

CRT Monitor

A cathode ray tube monitor is a type of computer monitor used popularly in the 1980's and only outsold by LCD screens in 2003. The CRT monitor utilizes a travelling electron beam that strikes a phosphorescent surface. The electron beam travels horizontally across the phosphor-coated screen one spot at a time (pixel) until it reaches the end and then travels down a row of spots and then travels across the screen again. The signals that tell the electron beam where to be at a particular time are called Hsync and Vsync. Because of the nature of the CRT monitor, these signals must be very particular. The Hsync must be off for 640 + 16 "front porch" + 48 "back porch" clks, then on for 96 clks, for a total of 800 clks for a total cycle called a scan line. Then, the Vsync signal must be off for 480 + 11 "front porch" + 32 "back porch" scan lines, then on for 2 scan lines for a total of 525 scan lines. The other three signals that

the CRT receives are the Red, Green, and Blue intensity signals. The color signals can only be displayed during the 640 clks and 480 scan lines. These signals receive between 0 and 0.7V, the greater voltage corresponding to a greater intensity on the screen. Timed with the Hsync and Vsync signals, the CRT monitor can receive the right color at the right time to display a desired image. The CRT also receives grounds for each of the color signals, a ground for the sync signals, and an overall ground. All of these signals are gathered through VGA connection.

Raspberry Pi Design:

In this project, the Raspberry Pi drives the data collection and processing of the temperature readings, then sends the processed data to the FPGA.

Software

The AMG8833 is able to connect to the Raspberry Pi through I2C communication protocol. The I2C peripheral is also made available through the pioInit() memory mapping. I2C register mapping on the Pi is done in the EasyPIO.h header file. I2C is initialized with the i2cInit function which sets the SDA and SCL pins, sets the SCLK to 100 kHz, and sets the AMG8833 address register as 0x69. The camera is set to capture data at 10 frames/second through writing 0x00 to the 0x02 register. The i2cWriteToReg function clears the READ bit indicating a write, writes the data to the FIFO, then writes the register address to the FIFO. Camera data is obtained through the i2cRead function which only takes the register address as the input. The function writes the address to the FIFO, then reads the slaves response as the slave outputs the data from the specified address.

The data processing portion reads all 64 temperature values from the camera, interpolates that data into a 32x32 array, and sends that to the FPGA video driver. The readPixels function reads all of the temperature registers and outputs an 8x8 array of temperature values. The output of the camera is multiplied by the temperature resolution of the camera (0.25℃) to get the temperature reading. This data

is then sent through linearInterp16 and linearInterp32 where it goes through a bilinear interpolation twice to end up with the 32x32 array of temperature readings. New interpolated values are averages of its neighboring horizontal (rows) or vertical (columns) elements. The final row is an exact copy of the penultimate row since there are an odd amount of spaces in the vertical averaging.

The 32x32 array is then converted into a 16x32 array. Every 0.4℃ in the range of 26℃ to 32℃ will be represented by an integer 0-15. Since 0-15 can be represented using 4 bits, two temperature values can be described in one byte. So, the number of elements per row in the 32x32 array is halved such that each byte represents two temperature readings. The result is a 16x32 array of integers in the range of 0-15. This conversion is done in pixelConvert which encodes the temperature values and bytePixelConvert which combines row elements such that each byte represents two temperature readings.

The final array is sent to the FPGA over SPI using the spiSendReceive function. Additionally, while sending, the Pi will drive the controlPin high to signal to the FPGA that a transfer is in place. When the transfer is done, the Pi will drive the controlPin low.

Hardware

The AMG8833 camera is connected to the Pi through two wires connecting SCL1 (I2C) and SDA1 (I2C). The camera is powered through 3.3V from the Pi and the grounds of the camera, Pi, and FPGA are all tied together.

The Pi connects to the FPGA through three wires: SCLK (SPI), MOSI (SPI), and controlPin (Pin 21).

FPGA Design:

The FPGA on the μMudd Board IV takes in the 16x32 byte array, 4096-bit data in from the Raspberry Pi and displays it on the CRT monitor through VGA.

Software

The signals needed out of the FPGA and into the VGA are Hsync, Vsync, and the three color signals, red, green, and blue. The clk must oscillate at 25MHz if the whole screen is to be displayed at near 60Hz, so the team used Quartus's clk wizard to multiply the 40MHz clk on the FPGA by 5 and then to divide by 8 to get 25MHz. The slower clock is used to count through counterH and counterV. CounterH is reset when it reaches 799, and then whenever it reaches 799, counterV is counted. CounterV is reset when it reaches 524. Hsync and Vsync are created based off of these counters. An ON signal is created to be on in the areas where color was allowed based off of Hsync and Vsync.

Based off of a value between 0 and 3 for brightness for five colors (red, green, blue, and then teal and yellow based on combinations of those base colors), there is a range of 0-15 that can be utilized for a range of temperatures since all of the 0 brightness values for all of the colors is the same, black. The team created a decoder to get from a value between 0 and 15 and display a corresponding color and brightness. The decoder goes from black to blue to teal to green to yellow to red in terms of temperature; the darker the version of each color, the lower the temperature.

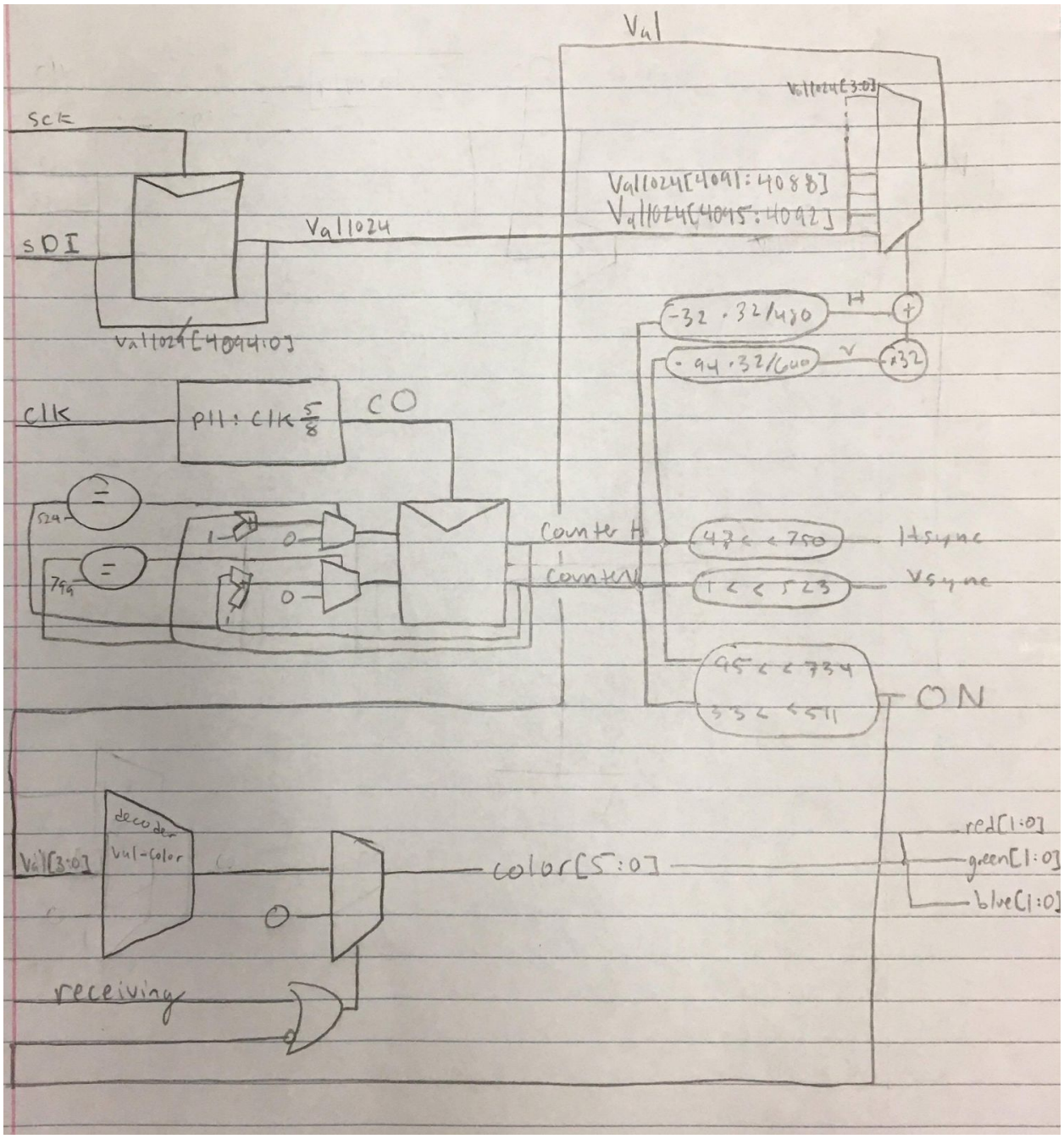
The input value to the FPGA will be a long value containing all of the information for every pixel. The team will have the least significant four bits correspond to the upper left pixel, and then the second least significant four bits correspond to one to the right, and so on, wrapping around so that the 33rd least significant bits correspond to the pixel sitting right below the first pixel. The team offset the counterV and counterH signals so that they started at 0 and counted to 480 and 640 respectively. Then, the team multiplied these by 32 and divided by 480 and 640 respectively to get values that ranged from 0 to 31 for both the vertical and horizontal counters. With this, the team could gather the correct pixel data from the data carrying all 1024 pixels worth of data corresponding to the correct range of pixels on the monitor by indexing the 1024 pixel array by the two 0-31 counters.

Hardware

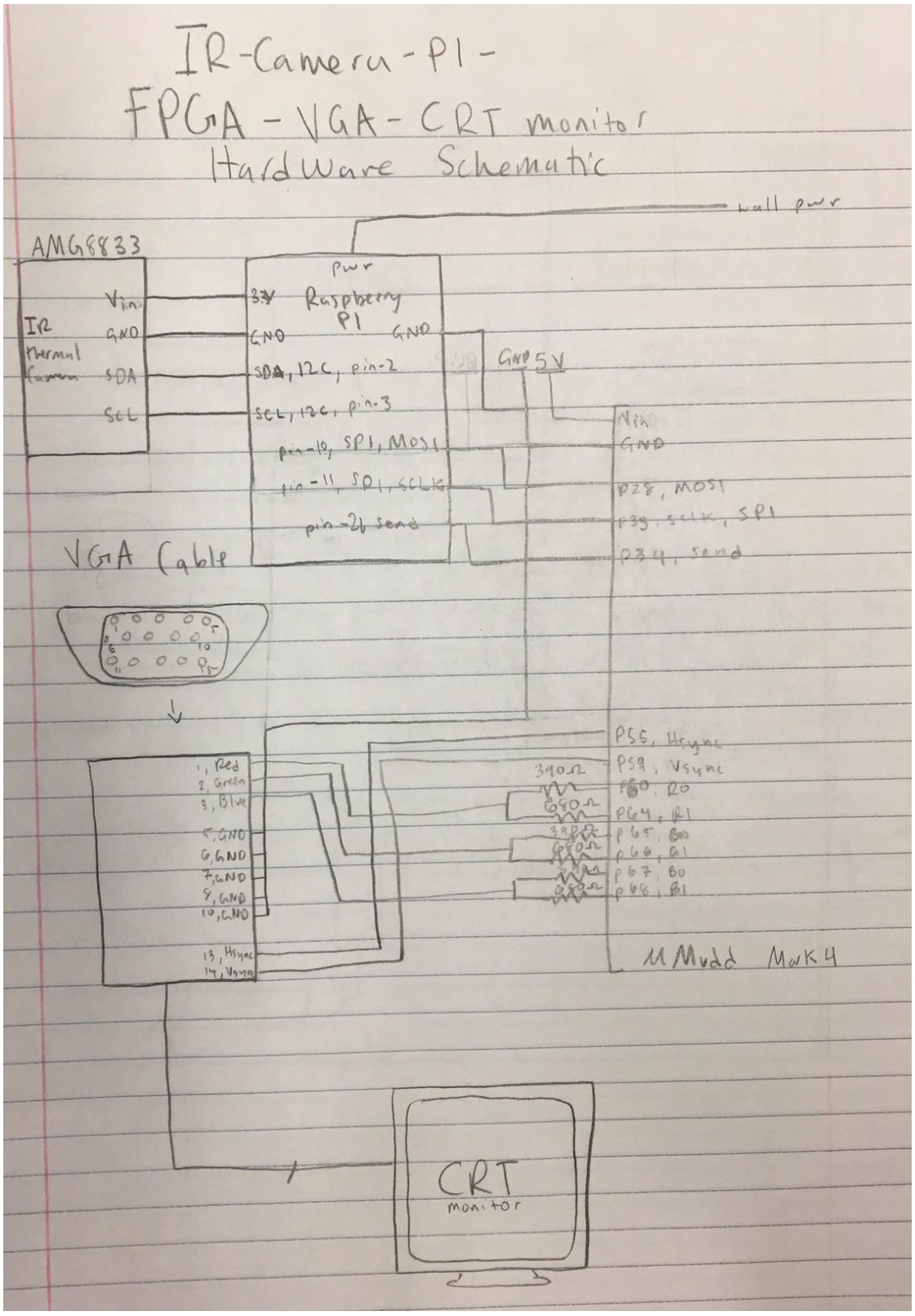
The FPGA has to be wired to the CRT monitor through a VGA connection. The VGA connection takes in red, green, and blue color inputs as well as Hsync and Vsync signals. Additionally, it has five ground pin connections. The Hsync and Vsync inputs can take in any logical square wave, so they could be connected directly to the VGA cord without worry of overpowering or not providing enough power to the circuit. However, the red, green, and blue color signals receive a range of 0-0.7V, corresponding to brightness. In addition, the team desires two bits of information to be delivered to the CRT monitor for each color, also corresponding to brightness. This results in four different levels of brightness for each color. To accomplish this, the team designed a voltage divider superposition circuit. By using two pins, each outputting 3.3V, the team could achieve the four different levels of brightness between 0 and 0.7V by having one either output 0 or 0.46V through a voltage divider, and then another providing 0 or 0.24V through the same circuit. In this way, if both are on, the VGA pin will receive the full 0.7V, and the amount of voltage can be adjusted by altering which pins are on. The team did the following math to find the necessary resistors: $V_{out} = V_1 * \frac{R_3//R_1}{R_3//R_2+R_1} + V_2 * \frac{R_3//R_2}{R_3//R_2+R_1}$ where the function R1//R2 means in parallel with, so, $V_{out} = V_1 * \frac{R_3R_1}{R_3R_1+R_2(R_3+R_1)} + V_2 * \frac{R_3R_2}{R_3R_2+R_1(R_3+R_2)}$, now, we want Vout to be 0.7 when V1 and V2 are 3.3V, and we want V1 to provide 2 times the voltage as V2, so we can say $\frac{2}{3}\frac{0.7V}{3.3V} = \frac{R_3R_1}{R_3R_1+R_2(R_3+R_1)}$, $\frac{1}{3}\frac{0.7V}{3.3V} = \frac{R_3R_2}{R_3R_2+R_1(R_3+R_2)}$ now we have two equations and three unknowns, but it is known that the typical impedance from a VGA cable is R3=70 Ω. So now we have two equations and two unknowns, and we can solve for R1 and R2. They turn out to be R1 = 390 Ω and R2=780 Ω. The closest values in the lab are 390 Ω and 680 Ω

Schematics:

FPGA HDL Block Diagram



Overall Hardware



Results:

The project properly displays the thermal camera data onto a CRT monitor. When a warm and cold cup of water are held in front of the camera, a user is able to distinguish between the two just through looking at the monitor. The resolution of the final image was high enough that one was able to distinguish fingers on a hand held in front of the camera.

The resolution and framerate of the final product are not as high quality as the team would have liked. Currently, the project is limited with the thermal camera's 8x8 resolution. Further interpolation of the data would enlarge and blur the final image. Additionally, the final image is displayed at a 3.4Hz refresh rate. This is noticeable to the human eye which results in a choppy display. The camera is able to output data at a rate of 10Hz. With more time, the team would have addressed this issue next. Another bug that the team would address in the future is the slight flicker on the screen. When the Pi is loading new data into the FPGA, horizontal black bars appear on the screen. This could possibly be addressed if the framerate of the displayed image was fast enough that one wouldn't notice the black bars on the screen.

The team ultimately met all of the proposed goals of the project.

References:

https://forum.allaboutcircuits.com/gallery/photos/vga-pinout.1812/

http://whatis.techtarget.com/definition/cathode-ray-tube-CRT

https://cdn-learn.adafruit.com/assets/assets/000/043/261/original/Grid-EYE_SPECIFICATIONS%28Reference%29.pdf?1498680225

http://www.surecontrols.com/infrared-temperature-sensors

Parts List:

- µMudd Board IV
- Raspberry Pi 3 Model B
- Sony CPD-200ES - New - 17" CRT Monitor
- Adafruit AMG8833 8x8 Thermal Camera Sensor
- 3x 390Ω resistors
- 3x 680Ω resistors
- 10x male to female jumper cables
- 5x male to male jumper cables

Appendices

Verilog HDL Code

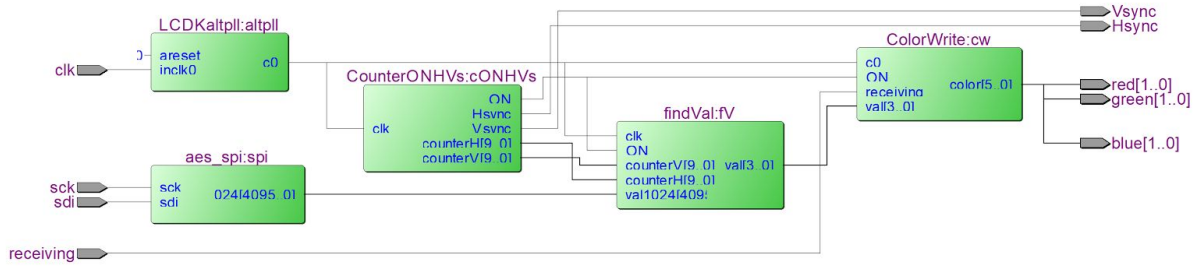
```
////////////////////////////////////////////////
// LCDK_FP
//   Top level module with SPI interface and SPI core
////////////////////////////////////////////////
module LCDK_FP( input logic clk,
            //SPI
            input logic sck, sdi, receiving,
            //CRT thru VGA
            output logic [1:0] red, green, blue,
            output logic Hsync, Vsync);

  //the input to this function will be clk, sck, and sdi
  //the output will be the red, green, and blue two bit values
  //and the hsync and vsync values, all for the CRT monitor
  //the function will display a 32x32 pixel image on a CRT monitor
  //each pixel will have a range of 0-15 for it's temperature
  //this means four bits for each pixel for a total incoming
  //transmission of 4096 pixels.
  //first let's get the 4096 bit array through SPI
  logic [4095:0] val1024;
  aes_spi spi(sck, sdi, val1024);

  //then let's generate a 25MHz clk from the 40MHz FPGA clk.
  logic c0;
  LCDKaltpll altpll(0, clk, c0);
  //then let's create the counterV, counterH, ON, Hsync, and Vsync signals
  //the Hsync and Vsync are direct outputs, and the counterH and counterV
  //will be helpful later on in the code for timing issues. ON will
  //dictate whenever color should be displayed.
  logic ON;
  logic [9:0] counterH, counterV;
  CounterONHVs cONHVs(c0, counterH, counterV, ON, Hsync, Vsync);

  //now let's search for the 4-bit value we want from the 4096 bit val1024
  //based on the counters counterV/H that are helpful for timing!
  logic[3:0] val;
  findVal fV(c0, counterV, counterH, val1024, ON, val);

  //now that we have the value, let's decode it to find the color wanted!
  logic [5:0] color;
  ColorWrite cw(c0, ON, val, receiving, color);

  //now that we have the color string, let's split it into it's RGB components!
  assign red = color[5:4];
  assign green = color[3:2];
  assign blue = color[1:0];
  //and we're done!

endmodule

////////////////////////////////////////////////
// aes_spi
```

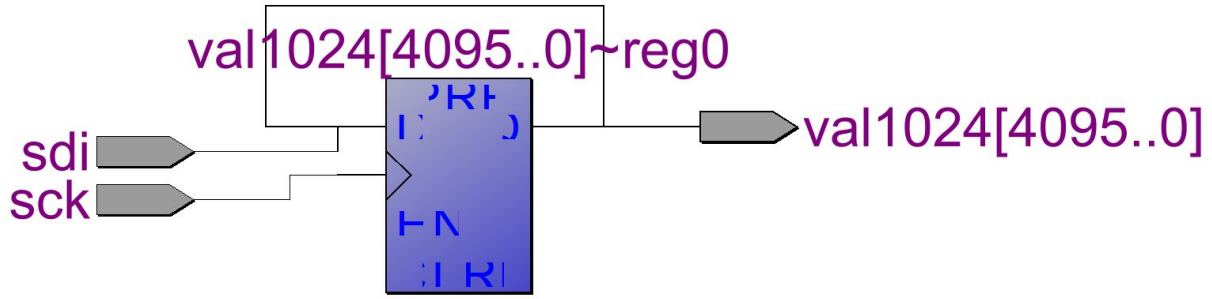
```
//   SPI interface.  Shifts in val1024
////////////////////////////////////////////
module aes_spi(input  logic sck,
               input  logic sdi,
               output logic [4095:0] val1024);


               //let's shift in the data!
               always_ff @(posedge sck)
  val1024 <= {val1024[4094:0], sdi};

endmodule
module findVal(input logic clk,
               input logic [9:0] counterV, counterH,
               input logic [4095:0] val1024,
               input logic ON,
               output logic [3:0] val);


  //create some values to normalize by
  logic [9:0] Hb, Vb, H, V;
  logic [1023:0] index;

  //get the start of Hb to be 0
  assign Hb = counterH - 94;
  //now normalize H to between 0 and 640
  assign H = Hb*32/640;

  //get the start of Vb to be 0
  assign Vb = counterV - 32;
  //now normalize V to between 0 and 480
  assign V = Vb*32/480;

  //then let's make an index value that gets at the base
  //value of whichever 4 val bits I want
  assign index = 32*V + H;
  //now for some fancy verilog syntax to access steadily
  //increasing parts of a long string. I want to grab 4
  //bits and then increase the base bit by 4:
  assign val = val1024[4*index +: 4];

endmodule
module ColorWrite(input logic c0, ON,
               input logic [3:0] val,
  input logic receiving,
               output logic [5:0] color);
 //val | color
 //0 | black
 //1 | dark blue
 //2 | blue
 //3 | bright blue
 //4 | dark teal
 //5 | teal
 //6 | bright teal
 //7 | dark green
 //8 | green
 //9 | bright green
```

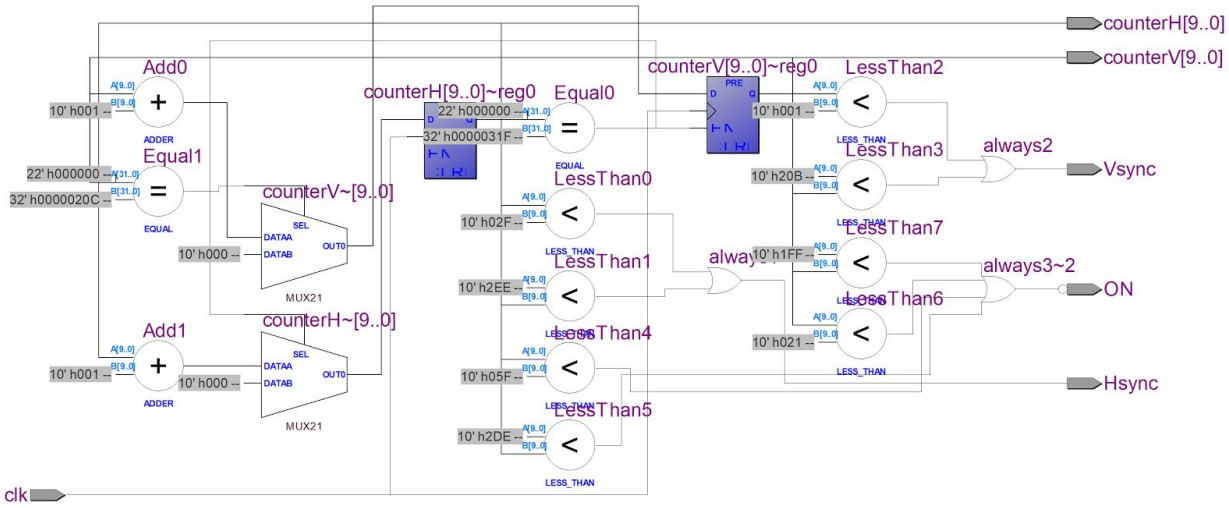
```
 //A | dark yellow
 //B | yellow
 //C | bright yellow
 //D | dark red
 //E | red
 //F | bright red

  always_comb
  if (receiving) color = 6'b000000;
  else if(~ON) color = 6'b000000;
  else
  case(val)
  //black
   4'b0000: color = 6'b000000;
  //blue
   4'b0001: color = 6'b000001;
   4'b0010: color = 6'b000010;
   4'b0011: color = 6'b000011;
  //teal
   4'b0100: color = 6'b000101;
   4'b0101: color = 6'b001010;
   4'b0110: color = 6'b001111;
  //green
   4'b0111: color = 6'b000100;
   4'b1000: color = 6'b001000;
   4'b1001: color = 6'b001100;
  //yellow
   4'b1010: color = 6'b010100;
   4'b1011: color = 6'b101000;
   4'b1100: color = 6'b111100;
  //red
   4'b1101: color = 6'b010000;
   4'b1110: color = 6'b100000;
   4'b1111: color = 6'b110000;
   default: color = 6'b000000;
  endcase

endmodule
module CounterONHVs(input logic clk,
            output logic [9:0] counterH, counterV,
            output logic ON, Hsync, Vsync);

  //counting to 799 and 524 to start the Hsync and Vsync signals
  //and get the timing on them
  always_ff@(posedge clk)
   if (counterH == 799)
            begin
            counterH <= 0;
            if (counterV == 524) counterV <= 0;
            else counterV <= counterV + 1;
            end
   else counterH <= counterH + 1;

  //next, for the actual Hsync and Vsync signals
  //following the Hsync and Vsync rules
  always_comb
  if (counterH < 47 | counterH > 750) Hsync = 1;
```

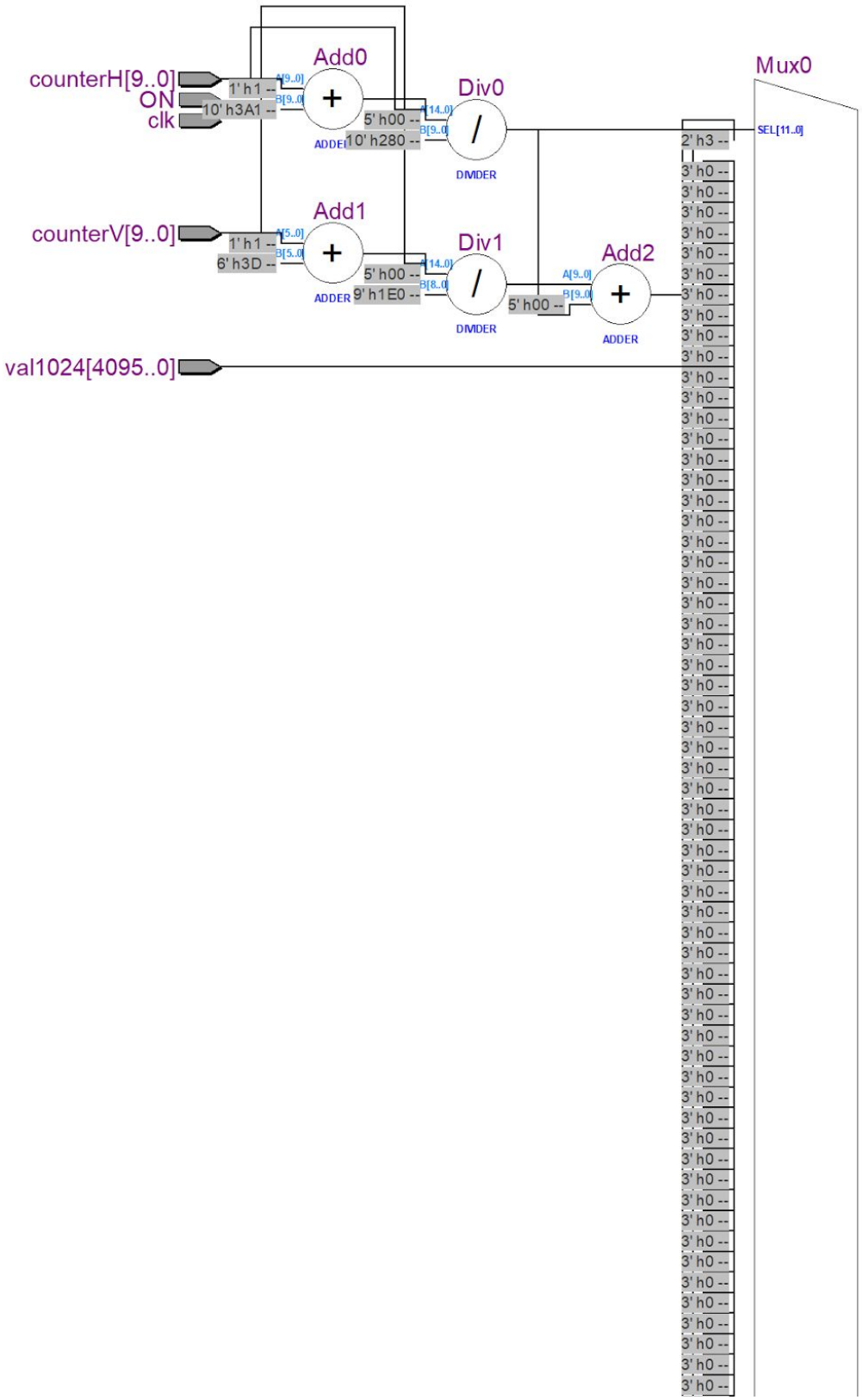
```
  else Hsync = 0;

  always_comb
  if (counterV < 1 | counterV > 523) Vsync = 1;
  else Vsync = 0;
  //now for the ON signal, when colors should be displayed
  //according to the VGA rules
  always_comb
  if ((counterH < 95) | (counterH > 734) | (counterV < 33) | (counterV > 511)) ON = 1'b0;
  else ON = 1'b1;
Endmodule
```

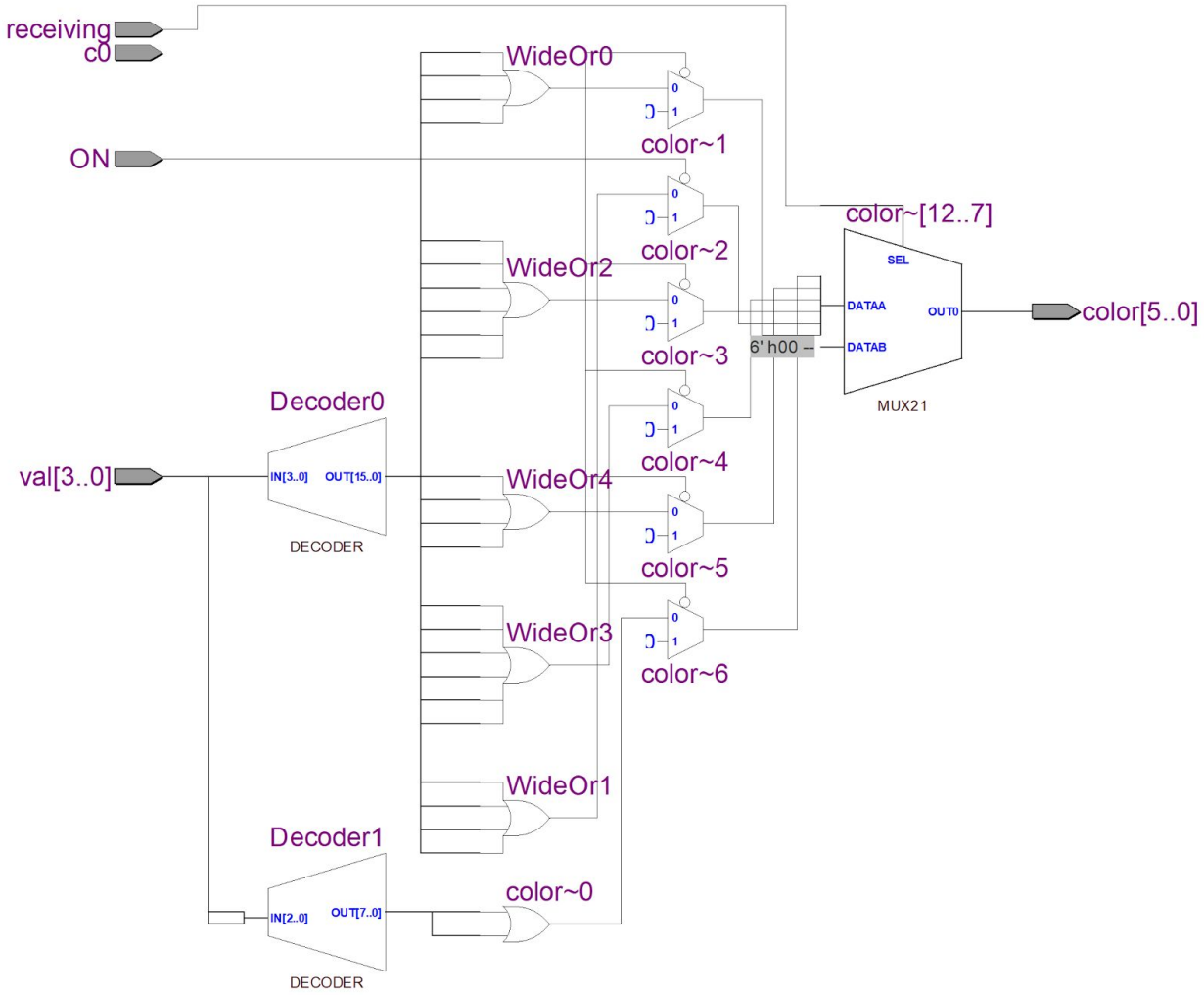
RTL Viewer




val1024[4095..0]~reg0
sdi
sck
val1024[4095..0]


counterH[9..0]
counterV[9..0]
Add0
Add1
Equal0
Equal1
counterH[9..0]~reg0
counterV[9..0]~reg0
counterV~[9..0]
counterH~[9..0]
LessThan0
LessThan1
LessThan2
LessThan3
LessThan4
LessThan5
LessThan6
LessThan7
always2
always3~2
Vsync
ON
Hsync
clk
MUX21
ADDER
EQUAL
LESS_THAN


counterH[9..0]
ON
clk
counterV[9..0]
val1024[4095..0]
Add0
Add1
Div0
Div1
Add2
Mux0
1' h1 --
10' h3A1 --
5' h00 --
10' h280 --
6' h3D --
9' h1E0 --
2' h3 --
3' h0 --
SEL[11..0]
ADDER
DIVIDER


receiving
c0
ON
val[3..0]
WideOr0
WideOr2
WideOr4
WideOr3
WideOr1
color~0
color~1
color~2
color~3
color~4
color~5
color~6
Decoder0
IN[3..0]
OUT[15..0]
DECODER
Decoder1
IN[2..0]
OUT[7..0]
DECODER
color~[12..7]
6' h00
SEL
DATAA
DATAB
OUT0
MUX21
color[5..0]

C Code

```
/*      ThermalCam.c
 *              Created:                8 December 2017
 *                                              David Kwan (dkwan@hmc.edu)
 *
 *      This is the main file for the Infrared Thermography project.
 *
*/
#include "EasyPIO.h"
#define controlPin 21
#define INPUT 0
#define OUTPUT 1
int main(void){

        ////////////////////////////////////////////////
        //Variables
        ////////////////////////////////////////////////
        int i;
        float *p,*n;
        char *f,*b;

        ////////////////////////////////////////////////
        //Init functions
        ////////////////////////////////////////////////
        pioInit();
        spiInit(3300000,0);
        i2cInit(100000,0x69);
        pinMode(controlPin, OUTPUT); //set pin 21 as output

        ////////////////////////////////////////////////
        //I2C Read/Write
        ////////////////////////////////////////////////
        i2cWriteToReg(0x02,0x00); //set to 10 frames/second


        ////////////////////////////////////////////////
        //Interpolation
        ////////////////////////////////////////////////
        //n = linearInterp16(p);
        //n = linearInterp32(n);

        ////////////////////////////////////////////////
        //Categorize
        ////////////////////////////////////////////////
        //f = pixelConvert(n);

        ////////////////////////////////////////////////
        //Byte Format
        ////////////////////////////////////////////////
        //b = bytePixelConvert(f);

        ////////////////////////////////////////////////
```

```
	//Read, Interpolate, Categorize, Format, Send
	////////////////////////////////////////////////
	while(1){
		p = readPixels();
		n = linearInterp16(p);
		n = linearInterp32(n);
		f = pixelConvert(n);
		b = bytePixelConvert(f);
		for(i=0; i<512; i++){
			spiSendReceive(*(b+i));
			digitalWrite(controlPin,1);
		}
		digitalWrite(controlPin,0);
	}

	return 0;
}
```

```
/*	EasyPIO.h
 *		Created:		8 December 2017
 *							David Kwan (dkwan@hmc.edu)
 *
 *	This is the header file for the Infrared Thermography project.
 *	This is a modified EasyPIO.h to include I2C and AMG8833 data processing functions
*/

#ifndef EASY_PIO_H
#define EASY_PIO_H

// Include statements
#include <sys/mman.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>


//////////////////////////////////////////////////////////////////
// Constants
//////////////////////////////////////////////////////////////////

// GPIO FSEL Types
#define INPUT  0
#define OUTPUT 1
#define ALT0   4
#define ALT1   5
#define ALT2   6
#define ALT3   7
#define ALT4   3
#define ALT5   2
```

```
//I2C
#define SDA_PIN 2
#define SCLK_PIN 3
#define HIGH 1
#define LOW 0


//////////////////////////////////////////////////////////////////
// Memory Map
//////////////////////////////////////////////////////////////////

// These #define values are specific to the BCM2835, taken from "BCM2835 ARM Peripherals"
//#define BCM2835_PERI_BASE       0x20000000
// Updated to BCM2836 for Raspberry Pi 2.0 Fall 2015 dmh
#define BCM2835_PERI_BASE       0x3F000000

#define GPIO_BASE               (BCM2835_PERI_BASE + 0x200000)
#define SPI0_BASE                                   (BCM2835_PERI_BASE + 0x204000)
#define I2C_BASE                        (BCM2835_PERI_BASE + 0x804000)

#define BLOCK_SIZE (4*1024)

////////////////////////////
//AMG8833
////////////////////////////
#define AMG8833_I2CADDR                 0x69; //address for AMG8833
#define AMG8833_PCTL                                    0x00;
#define AMG8833_RST                     0x01;
#define AMG88xx_FPSC                    0x02
#define AMG88xx_INTC                    0x03
#define AMG88xx_STAT                                    0x04
#define AMG88xx_SCLR                                    0x05
//0x06 reserved
#define AMG88xx_AVE                     0x07
#define AMG88xx_INTHL                                   0x08
#define AMG88xx_INTHH                                   0x09
#define AMG88xx_INTLL                                   0x0A
#define AMG88xx_INTLH                                   0x0B
#define AMG88xx_IHYSL                                   0x0C
#define AMG88xx_IHYSH                                   0x0D
#define AMG88xx_TTHL                                    0x0E
#define AMG88xx_TTHH                                    0x0F
#define AMG88xx_INT_OFFSET              0x010
#define AMG88xx_PIXEL_OFFSET                            0x80

//AMG8833 Operating Modes
#define AMG88xx_NORMAL_MODE                             0x00
#define AMG88xx_SLEEP_MODE                              0x01
#define AMG88xx_STAND_BY_60                             0x20
#define AMG88xx_STAND_BY_10                             0x21

//sw resets
#define AMG88xx_FLAG_RESET                              0x30
#define AMG88xx_INITIAL_RESET                           0x3F
```

```
//frame rates
#define AMG88xx_FPS_10                  0x00
#define AMG88xx_FPS_1                   0x01

//int enables
#define AMG88xx_INT_DISABLED            0x00
#define AMG88xx_INT_ENABLED             0x01

//int modes
#define AMG88xx_DIFFERENCE          0x00
#define AMG88xx_ABSOLUTE_VALUE          0x01

#define AMG88xx_PIXEL_ARRAY_SIZE  64
#define AMG88xx_PIXEL_TEMP_CONVERSION  .25 //floats???
#define AMG88xx_THERMISTOR_CONVERSION  .0625


// Pointers that will be memory mapped when pioInit() is called
volatile unsigned int *gpio; //pointer to base of gpio
volatile unsigned int *spi;  //pointer to base of spi registers
volatile unsigned int *i2c; //pointer to base of i2c registers

//////////////////////////////////////////////////////////////////////////
// GPIO Registers
//////////////////////////////////////////////////////////////////////////

// Function Select
#define GPFSEL    ((volatile unsigned int *) (gpio + 0))
typedef struct
{
   unsigned FSEL0      : 3;
   unsigned FSEL1      : 3;
   unsigned FSEL2      : 3;
   unsigned FSEL3      : 3;
   unsigned FSEL4      : 3;
   unsigned FSEL5      : 3;
   unsigned FSEL6      : 3;
   unsigned FSEL7      : 3;
   unsigned FSEL8      : 3;
   unsigned FSEL9      : 3;
   unsigned            : 2;
}gpfsel0bits;
#define GPFSEL0bits (*(volatile gpfsel0bits*) (gpio + 0))
#define GPFSEL0 (*(volatile unsigned int*) (gpio + 0))

typedef struct
{
   unsigned FSEL10      : 3;
   unsigned FSEL11      : 3;
   unsigned FSEL12      : 3;
   unsigned FSEL13      : 3;
   unsigned FSEL14      : 3;
   unsigned FSEL15      : 3;
```

```
   unsigned FSEL16      : 3;
   unsigned FSEL17      : 3;
   unsigned FSEL18      : 3;
   unsigned FSEL19      : 3;
   unsigned             : 2;
}gpfsel1bits;
#define GPFSEL1bits (*(volatile gpfsel1bits*) (gpio + 1))
#define GPFSEL1 (*(volatile unsigned int*) (gpio + 1))

typedef struct
{
   unsigned FSEL20      : 3;
   unsigned FSEL21      : 3;
   unsigned FSEL22      : 3;
   unsigned FSEL23      : 3;
   unsigned FSEL24      : 3;
   unsigned FSEL25      : 3;
   unsigned FSEL26      : 3;
   unsigned FSEL27      : 3;
   unsigned FSEL28      : 3;
   unsigned FSEL29      : 3;
   unsigned             : 2;
}gpfsel2bits;
#define GPFSEL2bits (* (volatile gpfsel2bits*) (gpio + 2))
#define GPFSEL2 (* (volatile unsigned int *) (gpio + 2))

typedef struct
{
   unsigned FSEL30      : 3;
   unsigned FSEL31      : 3;
   unsigned FSEL32      : 3;
   unsigned FSEL33      : 3;
   unsigned FSEL34      : 3;
   unsigned FSEL35      : 3;
   unsigned FSEL36      : 3;
   unsigned FSEL37      : 3;
   unsigned FSEL38      : 3;
   unsigned FSEL39      : 3;
   unsigned             : 2;
}gpfsel3bits;
#define GPFSEL3bits (* (volatile gpfsel3bits*) (gpio + 3))
#define GPFSEL3 (* (volatile unsigned int *) (gpio + 3))


typedef struct
{
   unsigned FSEL40      : 3;
   unsigned FSEL41      : 3;
   unsigned FSEL42      : 3;
   unsigned FSEL43      : 3;
   unsigned FSEL44      : 3;
   unsigned FSEL45      : 3;
   unsigned FSEL46      : 3;
   unsigned FSEL47      : 3;
```

```
   unsigned FSEL48      : 3;
   unsigned FSEL49      : 3;
   unsigned             : 2;
}gpfsel4bits;
#define GPFSEL4bits (* (volatile gpfsel4bits*) (gpio + 4))
#define GPFSEL4 (* (volatile unsigned int *) (gpio + 4))

typedef struct
{
   unsigned FSEL50      : 3;
   unsigned FSEL51      : 3;
   unsigned FSEL52      : 3;
   unsigned FSEL53      : 3;
   unsigned             : 20;
}gpfsel5bits;
#define GPFSEL5bits (* (volatile gpfsel5bits*) (gpio + 5))
#define GPFSEL5 (* (volatile unsigned int *) (gpio + 5))

// Pin Output Select
#define GPSET    ((volatile unsigned int *) (gpio + 7))
typedef struct
{
   unsigned SET0       : 1;
   unsigned SET1       : 1;
   unsigned SET2       : 1;
   unsigned SET3       : 1;
   unsigned SET4       : 1;
   unsigned SET5       : 1;
   unsigned SET6       : 1;
   unsigned SET7       : 1;
   unsigned SET8       : 1;
   unsigned SET9       : 1;
   unsigned SET10      : 1;
   unsigned SET11      : 1;
   unsigned SET12      : 1;
   unsigned SET13      : 1;
   unsigned SET14      : 1;
   unsigned SET15      : 1;
   unsigned SET16      : 1;
   unsigned SET17      : 1;
   unsigned SET18      : 1;
   unsigned SET19      : 1;
   unsigned SET20      : 1;
   unsigned SET21      : 1;
   unsigned SET22      : 1;
   unsigned SET23      : 1;
   unsigned SET24      : 1;
   unsigned SET25      : 1;
   unsigned SET26      : 1;
   unsigned SET27      : 1;
   unsigned SET28      : 1;
   unsigned SET29      : 1;
   unsigned SET30      : 1;
   unsigned SET31      : 1;
```

```
}gpset0bits;
#define GPSET0bits (* (volatile gpset0bits*) (gpio + 7))
#define GPSET0 (* (volatile unsigned int *) (gpio + 7))

typedef struct
{
   unsigned SET32       : 1;
   unsigned SET33       : 1;
   unsigned SET34       : 1;
   unsigned SET35       : 1;
   unsigned SET36       : 1;
   unsigned SET37       : 1;
   unsigned SET38       : 1;
   unsigned SET39       : 1;
   unsigned SET40       : 1;
   unsigned SET41       : 1;
   unsigned SET42       : 1;
   unsigned SET43       : 1;
   unsigned SET44       : 1;
   unsigned SET45       : 1;
   unsigned SET46       : 1;
   unsigned SET47       : 1;
   unsigned SET48       : 1;
   unsigned SET49       : 1;
   unsigned SET50       : 1;
   unsigned SET51       : 1;
   unsigned SET52       : 1;
   unsigned SET53       : 1;
   unsigned             : 10;
}gpset1bits;
#define GPSET1bits (* (volatile gpset1bits*) (gpio + 8))
#define GPSET1 (* (volatile unsigned int *) (gpio + 8))

// Pin Output Clear
#define GPCLR    ((volatile unsigned int *) (gpio + 10))
typedef struct
{
   unsigned CLR0       : 1;
   unsigned CLR1       : 1;
   unsigned CLR2       : 1;
   unsigned CLR3       : 1;
   unsigned CLR4       : 1;
   unsigned CLR5       : 1;
   unsigned CLR6       : 1;
   unsigned CLR7       : 1;
   unsigned CLR8       : 1;
   unsigned CLR9       : 1;
   unsigned CLR10      : 1;
   unsigned CLR11      : 1;
   unsigned CLR12      : 1;
   unsigned CLR13      : 1;
   unsigned CLR14      : 1;
   unsigned CLR15      : 1;
   unsigned CLR16      : 1;
```

```
   unsigned CLR17      : 1;
   unsigned CLR18      : 1;
   unsigned CLR19      : 1;
   unsigned CLR20      : 1;
   unsigned CLR21      : 1;
   unsigned CLR22      : 1;
   unsigned CLR23      : 1;
   unsigned CLR24      : 1;
   unsigned CLR25      : 1;
   unsigned CLR26      : 1;
   unsigned CLR27      : 1;
   unsigned CLR28      : 1;
   unsigned CLR29      : 1;
   unsigned CLR30      : 1;
   unsigned CLR31      : 1;
}gpclr0bits;
#define GPCLR0bits (* (volatile gpclr0bits*) (gpio + 10))
#define GPCLR0 (* (volatile unsigned int *) (gpio + 10))

typedef struct
{
   unsigned CLR32      : 1;
   unsigned CLR33      : 1;
   unsigned CLR34      : 1;
   unsigned CLR35      : 1;
   unsigned CLR36      : 1;
   unsigned CLR37      : 1;
   unsigned CLR38      : 1;
   unsigned CLR39      : 1;
   unsigned CLR40      : 1;
   unsigned CLR41      : 1;
   unsigned CLR42      : 1;
   unsigned CLR43      : 1;
   unsigned CLR44      : 1;
   unsigned CLR45      : 1;
   unsigned CLR46      : 1;
   unsigned CLR47      : 1;
   unsigned CLR48      : 1;
   unsigned CLR49      : 1;
   unsigned CLR50      : 1;
   unsigned CLR51      : 1;
   unsigned CLR52      : 1;
   unsigned CLR53      : 1;
   unsigned            : 10;
}gpclr1bits;
#define GPCLR1bits (* (volatile gpclr1bits*) (gpio + 11))
#define GPCLR1 (* (volatile unsigned int *) (gpio + 11))

// Pin Level
#define GPLEV    ((volatile unsigned int *) (gpio + 13))
typedef struct
{
   unsigned LEV0       : 1;
   unsigned LEV1       : 1;
```

```
   unsigned LEV2      : 1;
   unsigned LEV3      : 1;
   unsigned LEV4      : 1;
   unsigned LEV5      : 1;
   unsigned LEV6      : 1;
   unsigned LEV7      : 1;
   unsigned LEV8      : 1;
   unsigned LEV9      : 1;
   unsigned LEV10     : 1;
   unsigned LEV11     : 1;
   unsigned LEV12     : 1;
   unsigned LEV13     : 1;
   unsigned LEV14     : 1;
   unsigned LEV15     : 1;
   unsigned LEV16     : 1;
   unsigned LEV17     : 1;
   unsigned LEV18     : 1;
   unsigned LEV19     : 1;
   unsigned LEV20     : 1;
   unsigned LEV21     : 1;
   unsigned LEV22     : 1;
   unsigned LEV23     : 1;
   unsigned LEV24     : 1;
   unsigned LEV25     : 1;
   unsigned LEV26     : 1;
   unsigned LEV27     : 1;
   unsigned LEV28     : 1;
   unsigned LEV29     : 1;
   unsigned LEV30     : 1;
   unsigned LEV31     : 1;
}gplev0bits;
#define GPLEV0bits (* (volatile gplev0bits*) (gpio + 13))
#define GPLEV0 (* (volatile unsigned int *) (gpio + 13))


typedef struct
{
   unsigned LEV32      : 1;
   unsigned LEV33      : 1;
   unsigned LEV34      : 1;
   unsigned LEV35      : 1;
   unsigned LEV36      : 1;
   unsigned LEV37      : 1;
   unsigned LEV38      : 1;
   unsigned LEV39      : 1;
   unsigned LEV40      : 1;
   unsigned LEV41      : 1;
   unsigned LEV42      : 1;
   unsigned LEV43      : 1;
   unsigned LEV44      : 1;
   unsigned LEV45      : 1;
   unsigned LEV46      : 1;
   unsigned LEV47      : 1;
   unsigned LEV48      : 1;
```

```
   unsigned LEV49      : 1;
   unsigned LEV50      : 1;
   unsigned LEV51      : 1;
   unsigned LEV52      : 1;
   unsigned LEV53      : 1;
   unsigned            : 10;
}gplev1bits;
#define GPLEV1bits (* (volatile gplev1bits*) (gpio + 14))
#define GPLEV1 (* (volatile unsigned int *) (gpio + 14))

//////////////////////////////////////////////////////////////////////////
// SPI Registers
//////////////////////////////////////////////////////////////////////////

typedef struct
{
        unsigned CS             :2;
        unsigned CPHA           :1;
        unsigned CPOL           :1;
        unsigned CLEAR                  :2;
        unsigned CSPOL          :1;
        unsigned TA             :1;
        unsigned DMAEN                  :1;
        unsigned INTD           :1;
        unsigned INTR           :1;
        unsigned ADCS           :1;
        unsigned REN            :1;
        unsigned LEN            :1;
        unsigned LMONO                  :1;
        unsigned TE_EN          :1;
        unsigned DONE           :1;
        unsigned RXD            :1;
        unsigned TXD            :1;
        unsigned RXR            :1;
        unsigned RXF            :1;
        unsigned CSPOL0         :1;
        unsigned CSPOL1         :1;
        unsigned CSPOL2         :1;
        unsigned DMA_LEN        :1;
        unsigned LEN_LONG       :1;
        unsigned                        :6;
}spi0csbits;
#define SPI0CSbits (* (volatile spi0csbits*) (spi + 0))
#define SPI0CS (* (volatile unsigned int *) (spi + 0))

#define SPI0FIFO (* (volatile unsigned int *) (spi + 1))
#define SPI0CLK (* (volatile unsigned int *) (spi + 2))
#define SPI0DLEN (* (volatile unsigned int *) (spi + 3))


//////////////////////////////////////////////////////////////////////////
// I2C Registers
//////////////////////////////////////////////////////////////////////////
typedef struct
```

```
{
        unsigned READ           :1;
        unsigned                :3;
        unsigned CLEAR          :2;
        unsigned                :1;
        unsigned ST             :1;
        unsigned INTD           :1;
        unsigned INTT           :1;
        unsigned INTR           :1;
        unsigned                :4;
        unsigned I2CEN          :1;
        unsigned                :16;
}i2ccbits;
#define I2CCBITS (* (volatile i2ccbits*) (i2c + 0))

typedef struct
{
        unsigned TA             :1;
        unsigned DONE           :1;
        unsigned TXW            :1;
        unsigned RXR            :1;
        unsigned TXD            :1;
        unsigned RXD            :1;
        unsigned TXE            :1;
        unsigned RXF            :1;
        unsigned ERR            :1;
        unsigned CLKT           :1;
        unsigned                :22;
}i2csbits;
#define I2CSBITS (* (volatile i2csbits*) (i2c + 1))

typedef struct
{
        unsigned DLEN           :16;
        unsigned                :32;
}i2cdlenbits;
#define I2CDLENBITS (* (volatile i2cdlenbits*) (i2c + 2))

typedef struct
{
        unsigned ADDR           :7;
        unsigned                :25;
}i2cabits;
#define I2CABITS (* (volatile i2cabits*) (i2c + 3))

typedef struct
{
        unsigned DATA           :8;
        unsigned                :24;
}i2cfifobits;
#define I2CFIFOBITS (* (voltaile i2cfifobits*) (i2c +4))

typedef struct
{
```

```
        unsigned CDIV           :16;
        unsigned                :32;
}i2cdivbits;
#define I2CDIVBITS (* (volatile i2cdivbits*) (i2c + 5))

typedef struct
{
        unsigned REDL           :16;
        unsigned FEDL           :16;
}i2cdelbits;
#define I2CDELBITS (* (volatile i2cdelbits*) (i2c + 6))

typedef struct
{
        unsigned TOUT           :16;
        unsigned                :32;
}i2cclktbits;
#define I2CCLKTBITS (* (volatile i2cclktbits*) (i2c + 7))

#define I2CC (* (volatile unsigned int *) (i2c + 0))
#define I2CS (* (volatile unsigned int *) (i2c + 1))
#define I2CDLEN (* (volatile unsigned int *) (i2c + 2))
#define I2CA (* (volatile unsigned int *) (i2c + 3))
#define I2CFIFO (* (volatile unsigned int *) (i2c + 4))
#define I2CDIV (* (volatile unsigned int *) (i2c + 5))
#define I2CDEL (* (volatile unsigned int *) (i2c + 6))
#define I2CCLKT (* (volatile unsigned int *) (i2c + 7))


//////////////////////////////////////////////////////////////////////////
// General Functions
//////////////////////////////////////////////////////////////////////////
void pioInit() {
        int  mem_fd;
        void *reg_map;

        // /dev/mem is a psuedo-driver for accessing memory in the Linux filesystem
        if ((mem_fd = open("/dev/mem", O_RDWR|O_SYNC) ) < 0) {
            printf("can't open /dev/mem \n");
            exit(-1);
        }

        reg_map = mmap(
          NULL,             //Address at which to start local mapping (null means don't-care)
    BLOCK_SIZE,       //Size of mapped memory block
    PROT_READ|PROT_WRITE,// Enable both reading and writing to the mapped memory
    MAP_SHARED,       // This program does not have exclusive access to this memory
    mem_fd,           // Map to /dev/mem
    GPIO_BASE);       // Offset to GPIO peripheral

        if (reg_map == MAP_FAILED) {
    printf("gpio mmap error %d\n", (int)reg_map);
    close(mem_fd);
    exit(-1);
```

```
    }

        gpio = (volatile unsigned *)reg_map;


        reg_map = mmap(
          NULL,             //Address at which to start local mapping (null means don't-care)
    BLOCK_SIZE,       //Size of mapped memory block
    PROT_READ|PROT_WRITE,// Enable both reading and writing to the mapped memory
    MAP_SHARED,       // This program does not have exclusive access to this memory
    mem_fd,           // Map to /dev/mem
    I2C_BASE);       // Offset to GPIO peripheral

        if (reg_map == MAP_FAILED) {
    printf("i2c mmap error %d\n", (int)reg_map);
    close(mem_fd);
    exit(-1);
    }

        i2c = (volatile unsigned *)reg_map;

    reg_map = mmap(
          NULL,             //Address at which to start local mapping (null means don't-care)
    BLOCK_SIZE,       //Size of mapped memory block
    PROT_READ|PROT_WRITE,// Enable both reading and writing to the mapped memory
    MAP_SHARED,       // This program does not have exclusive access to this memory
    mem_fd,           // Map to /dev/mem
    SPI0_BASE);       // Offset to SPI peripheral

    if (reg_map == MAP_FAILED) {
    printf("spi mmap error %d\n", (int)reg_map);
    close(mem_fd);
    exit(-1);
    }

    spi = (volatile unsigned *)reg_map;


}

//////////////////////////////////////////////////////////////////////////
// GPIO Functions
//////////////////////////////////////////////////////////////////////////

void pinMode(int pin, int function) {
    int reg      =  pin/10;
    int offset   = (pin%10)*3;
    GPFSEL[reg] &= ~((0b111 & ~function) << offset);
    GPFSEL[reg] |=  ((0b111 &  function) << offset);
}

void digitalWrite(int pin, int val) {
    int reg = pin / 32;
    int offset = pin % 32;
```

```
    if (val) GPSET[reg] = 1 << offset;
    else     GPCLR[reg] = 1 << offset;
}

int digitalRead(int pin) {
    int reg = pin / 32;
    int offset = pin % 32;

    return (GPLEV[reg] >> offset) & 0x00000001;
}

void pinsMode(int pins[], int numPins, int fxn) {
    int i;
    for(i=0; i<numPins; ++i) {
        pinMode(pins[i], fxn);
    }
}

void digitalWrites(int pins[], int numPins, int val) {
    int i;
    for(i=0; i<numPins; i++) {
        digitalWrite(pins[i], (val & 0x00000001));
        val = val >> 1;
    }
}

int digitalReads(int pins[], int numPins) {
    int i, val = digitalRead(pins[0]);

    for(i=1; i<numPins; i++) {
        val |= (digitalRead(pins[i]) << i);
    }
    return val;
}


//////////////////////////////////////////////////////////////////////
// SPI Functions
//////////////////////////////////////////////////////////////////////

void spiInit(int freq, int settings) {
    //set GPIO 8 (CE), 9 (MISO), 10 (MOSI), 11 (SCLK) alt fxn 0 (SPI0)
    pinMode(8, ALT0);
    pinMode(9, ALT0);
    pinMode(10, ALT0);
    pinMode(11, ALT0);

    //Note: clock divisor will be rounded to the nearest power of 2
    SPI0CLK = 250000000/freq;   // set SPI clock to 250MHz / freq
    SPI0CS = settings;
    SPI0CSbits.TA = 1;          // turn SPI on with the "transfer active" bit
}
```

```
char spiSendReceive(char send){
   SPI0FIFO = send;           // send data to slave
   while(!SPI0CSbits.DONE);    // wait until SPI transmission complete
   return SPI0FIFO;           // return received data
}

short spiSendReceive16(short send) {
   short rec;
   SPI0CSbits.TA = 1;         // turn SPI on with the "transfer active" bit
   rec = spiSendReceive((send & 0xFF00) >> 8); // send data MSB first
   rec = (rec << 8) | spiSendReceive(send & 0xFF);
   SPI0CSbits.TA = 0;         // turn off SPI
   return rec;
}



////////////////////////////////////////////////////////////////////
// I2C Functions
////////////////////////////////////////////////////////////////////

//Initialize i2c clk, pins, and slave address
void i2cInit(int freq, int adr) {
  //set GPIO 2 (SDA1), 3 (SCL1)
        pinMode(2, ALT0);
        pinMode(3, ALT0);

        I2CDIV = 250000000/freq; //core clock/CDIV typical is 100kHz
        I2CA = adr; //chip address
}

//Clear status flags
void i2c_ClearStatus(void){
        I2CSBITS.CLKT = 1;
        I2CSBITS.ERR = 1;
        I2CSBITS.DONE = 1;
}

//Wait until transfer is done
void i2c_waitDone(void)
{
        int timeout = 50;
        while((!(I2CSBITS.DONE)) && --timeout){
                usleep(1000);
        }
        if (timeout == 0)
                printf("Error: Timeout. \n");
}

//Write to FIFO
void i2cWrite(char data) {
        I2CCBITS.READ = 0; //Write packet transfer
        I2CDLEN = 1; //1 byte to be written
        //data = data & 0xFF;
```

```
	I2CFIFO = data;
	i2c_ClearStatus();
	I2CCBITS.I2CEN = 1; //Turn on BSC controller
	I2CCBITS.ST = 1; //start transfer
	i2c_waitDone();

}

//Write to a register
void i2cWriteToReg(char data, char reg) {
	I2CCBITS.READ = 0; //Write packet transfer
	I2CDLEN = 2; //2 byte to be written
	//data = data & 0xFF;
	I2CFIFO = data;
	i2c_ClearStatus();
	I2CCBITS.I2CEN = 1; //Turn on BSC controller
	I2CCBITS.ST = 1; //start transfer
	//i2c_waitDone();
	I2CFIFO = reg;
	//i2c_ClearStatus();
	//I2CCBITS.I2CEN = 1;
	//I2CCBITS.ST = 1;
	i2c_waitDone();

}

//Read from a register
char i2cRead(char reg) {
	i2cWrite(reg);
	I2CDLEN = 1;
	i2c_ClearStatus();
	I2CCBITS.READ = 1; //Read packet transfer
	I2CCBITS.CLEAR = 1; //Clear FIFO
	I2CCBITS.I2CEN = 1; //Turn on BSC controller
	I2CCBITS.ST = 1; //start new transfer
	i2c_waitDone();
	return I2CFIFO;
}



////////////////////////////////////////////////////////////////////
// AMG8833
// readPixels
//  Gets 8x8 array of temperature readings from sensor
////////////////////////////////////////////////////////////////////

float* readPixels(void){

	static float pixelArray[64];
	int i,temp;
	float tempFloat;
```

```
	for(i=0;i<128;i++){
		if(i%2 == 0){ //if even, lower byte
			temp = (float)i2cRead(0x80 + i);
		}
		else{ //if odd, upper byte, concatenate with lower byte
			tempFloat = (float) (temp | (i2cRead(0x80 + 1) << 8));
			pixelArray[i/2] = tempFloat / 4; //temperature conversion
		}
	}


	//print array
	printf("--------------------------------\r\n");
	printf("          readPixels          \r\n");
	printf("--------------------------------\r\n");
	for(int j=0; j < 64; j++) {
		if(j%8 == 0){
		printf("\r\n");
		}
		printf("%f, ",pixelArray[j]);
	}
	printf("\r\n");

	return pixelArray;

}

////////////////////////////////////////////////////////////////////////
//Interpolation
////////////////////////////////////////////////////////////////////////

//Linear 8x8 -> 16x16
float* linearInterp16(float array[64]){
	static float newArray[256];
	int i;
	int j=0;

	//place original in correct places, everything else is 0
	for(i=0; i<256; i++){
		if( (i%2 == 0) && ((i%32) < 16 ) ){
			newArray[i] = array[j];
			j++;
		}
		else{
			newArray[i] = 0;
		}
	}


	//vertical averaging, except for first and last row
	for(i=16; i<240; i++){
		if( (i%2 == 0) && ((i%32) > 15 ) ){
			newArray[i] = (newArray[i-16] + newArray[i+16])/2;
		}
```

```
	}


	//copy last given row into 16th row
	for(i=240; i<256; i++){
		newArray[i] = newArray[i-16];
	}

	//Row linear interpolation, average
	for(i=0; i<256; i++){
		if( i%2 == 1 ) {
			newArray[i] = (newArray[i-1] + newArray[i+1])/2;
		}
		if( i==255 ){
			newArray[i] = newArray[i-16];
		}
	}

	//print array
	printf("--------------------------------\r\n");
	printf("	linearInterp16		\r\n");
	printf("--------------------------------\r\n");
	for(int j=0; j < 256; j++) {
		if(j%16 == 0){
		printf("\r\n");
		}
		printf("%.2f, ",newArray[j]);
	}
	printf("\r\n");
	printf("\r\n");

	return newArray;

}

//Linear interpolation 16x16 -> 32x32
float* linearInterp32(float array[256]){
	int i;
	int j=0;
	static float newArray[1024];

	//place original in correct places, everything else is 0
	for(i=0; i<1024; i++){
		if( (i%2 == 0) && ((i%64) < 32 ) ){
			newArray[i] = array[j];
			j++;
		}
		else{
			newArray[i] = 0;
		}
	}


	//vertical averaging, except for first and last row
```

```
	for(i=16; i<992; i++){
		if( (i%2 == 0) && ((i%64) > 31 ) ){
			newArray[i] = (newArray[i-32] + newArray[i+32])/2;
		}
	}


	//copy last given row into 16th row
	for(i=992; i<1024; i++){
		newArray[i] = newArray[i-32];
	}

	//Row linear interpolation, middle value so average
	for(i=0; i<1024; i++){
		if( i%2 == 1 ) {
			newArray[i] = (newArray[i-1] + newArray[i+1])/2;
		}
		if(i==1023){
			newArray[i] = newArray[i-32];
		}
	}

	//print array
	printf("--------------------------------\r\n");
	printf("       linearInterp32         \r\n");
	printf("--------------------------------\r\n");
	for(int j=0; j < 1024; j++) {
		if(j%32 == 0){
		printf("\r\n");
		}
		printf("%.2f, ",newArray[j]);
	}
	printf("\r\n");
	printf("\r\n");

	return newArray;
}



////////////////////////////////////////////////////////////////
//Categorize
//Range: 26-32 C Change color every 0.4 C
////////////////////////////////////////////////////////////////

char* pixelConvert(float array[1024]){

	static char pixelArray[1024];
	int i,j;
	//Reversed order
	for(i=0;i<1024; i++){
		if(array[i] > 32)
			pixelArray[i] = 15;
		else if (array[i] > 26 && array[i] <= 26.4)
```

```
                pixelArray[i] = 1;
        else if (array[i] > 26.4 && array[i] <= 26.8)
                pixelArray[i] = 2;
        else if (array[i] > 26.8 && array[i] <= 27.2)
                pixelArray[i] = 3;
        else if (array[i] > 27.2 && array[i] <= 27.6)
                pixelArray[i] = 4;
        else if (array[i] > 27.6 && array[i] <= 28)
                pixelArray[i] = 5;
        else if (array[i] > 28 && array[i] <= 28.4)
                pixelArray[i] = 6;
        else if (array[i] > 28.4 && array[i] <= 28.8)
                pixelArray[i] = 7;
        else if (array[i] > 28.8 && array[i] <= 29.2)
                pixelArray[i] = 8;
        else if (array[i] > 29.2 && array[i] <= 29.6)
                pixelArray[i] = 9;
        else if (array[i] > 29.6 && array[i] <= 30)
                pixelArray[i] = 10;
        else if (array[i] > 30 && array[i] <= 30.4)
                pixelArray[i] = 11;
        else if (array[i] > 30.4 && array[i] <= 30.8)
                pixelArray[i] = 12;
        else if (array[i] > 30.8 && array[i] <= 31.2)
                pixelArray[i] = 13;
        else if (array[i] > 31.2 && array[i] <= 31.6)
                pixelArray[i] = 14;
        else if (array[i] > 31.6 && array[i] <= 32)
                pixelArray[i] = 15;
        else //less than 26C
                pixelArray[i] = 0;
}


//Reverse the columns to fix image reversal
for(int row=0; row<32; row++){
        for(int col=0; col<16; col++){
                char temp;
                temp = pixelArray[(row*32)+col];
                pixelArray[(row*32)+col] = pixelArray[(row*32)+(31-col)];
                pixelArray[(row*32)+(31-col)] = temp;
        }
}




//print array
printf("--------------------------------\r\n");
printf("       pixelConvert          \r\n");
printf("--------------------------------\r\n");
for(int j=0; j < 1024; j++) {
        if(j%32 == 0){
        printf("\r\n");
```

```
                }
                printf("%d, ",pixelArray[j]);
        }
        printf("\r\n");
        printf("\r\n");

        return pixelArray;

}

////////////////////////////////////////////////////////////
//Format into 16x32.
//Combine row values into a single byte
//Two temperature readings per byte
////////////////////////////////////////////////////////////
//16x16 now. turn into 8x16
char* bytePixelConvert(char array[1024]){
        static char bytePixelArray[512];
        int i,j;
        for(i=0;i<1024;i++){
                if(i%2!=0){ //if odd, append to last
                        bytePixelArray[i/2] = array[i] | (array[i-1] << 4);
                }
        }

        //print array
        printf("--------------------------------\r\n");
        printf("       bytePixelConvert      \r\n");
        printf("--------------------------------\r\n");
        for(int j=0; j < 512; j++) {
                if(j%16 == 0){
                printf("\r\n");
                }
                printf("%d, ",bytePixelArray[j]);
        }
        printf("\r\n");
        printf("\r\n");

        return bytePixelArray;
}
#endif
```