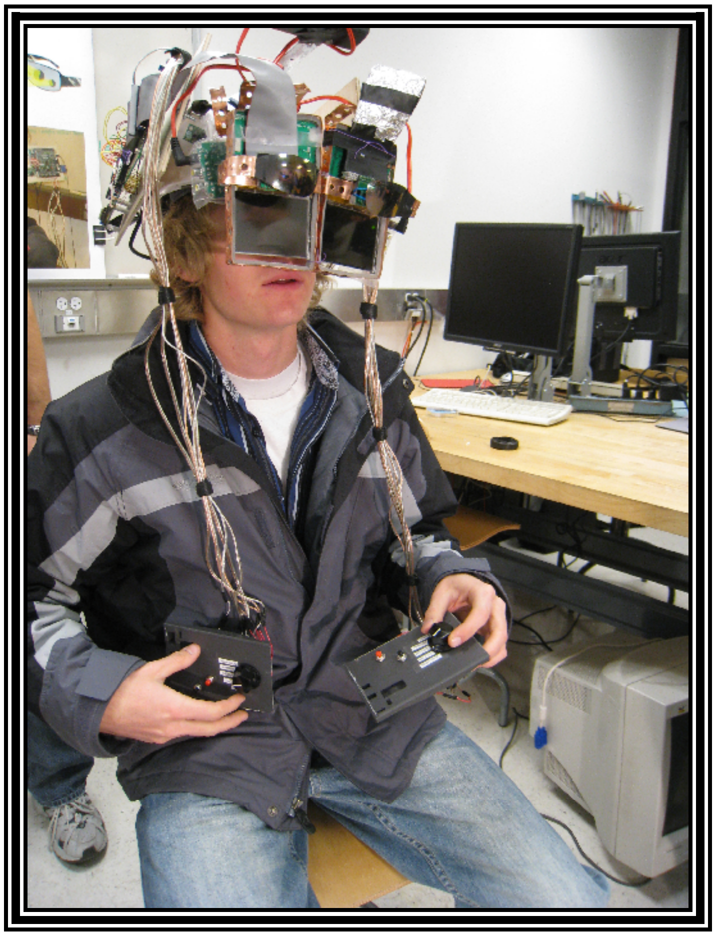
Dynamic Source Filtering Sunglasses

University of Utah

Computer Engineering Senior Project December 2009



Dan Parker
Torrey Atcitty
Dana Todd
Jason Hansen

TABLE OF CONTENTS

Introduction

The result of this project was the development of intelligent sunglasses. The goal of the sunglasses was to dim bright lights while leaving a person's remaining field of view undisturbed. These sunglasses are far superior to standard sunglasses in situations where the distribution of light is uneven. For example light is unevenly distributed when either driving into the sun or driving into oncoming headlights. The presence of bright lights causes the eye to adjust which makes the surrounding area darker by comparison. By dimming the bright spots, the eyes can remain dilated and continue to detect details from the surrounding area with much greater accuracy. The glasses consist of four major components: clear Liquid Crystal Display (LCD) screens, digital cameras, microcontrollers/processors, and chassis. The design consists of two separate systems that are functionally identical, where each system is dedicated to one eye.

Project Overview

The basic procedure the glasses go through starts with the glasses using a camera to take a picture of what the viewer sees. The camera streams that data to the microcontroller that processes each pixel and determines whether dimming is necessary or not. Inside the microcontroller is a dedicated pixel pipeline that has the 2D transformations from camera coordinates to screen coordinates in hardware. In parallel to the screen transforms is the brightness comparison logic. If dimming is needed, the pipeline has already calculated the position on the LCD screen that corresponds to the position of the bright spot. It writes data out to the dual ported frame buffer where the LCD controller will read it upon refreshing the screen. Between the LCD controller and the Frame buffer is an Overscan unit that can calculate if a bright spot exists near (or on) the current pixel. If so, the pixel is dimmed to block the light appropriately. This happens repeatedly and rapidly so that new lights are dimmed and light sources that have disappeared are no longer dimmed. The project is split into three design categories: physical placement and design, hardware component design, and software design.

The physical placement required all components to be mounted on a person's head so that they can look around. The main chassis is mounted on a construction helmet with all microcontrollers, LCDs, and cameras securely mounted to the helmet either by screws or copper wiring. The LCD screens are held in front of the viewers eyes so that a large portion of their field of view is through the screens. The cameras are mounted pointing in the same direction as the viewer is facing and are as close to the LCD screens as possible. The microcontroller and other components are mounted on the sides of the helmet.

The hardware component design is based mainly on the minimum function requirements of the system. The LCD can not inhibit light transmission to the point of being impossible to see through and the system must have a quick response time to any change in lighting. Also, all parts must be able to communicate with one another efficiently. Special camera and LCD controllers were designed to ensure both components can be used to their full potential in terms of response time. In order to guarantee that the microcontroller could satisfy the full output of the camera, a special pixel pipeline was designed. It has five stages of adds, multiplies, and compares. The values used to add, multiply, and compare are stored in registers that are accessible by the microcontroller through memory-mapped I/O. This way the microcontroller has the throughput required, and is also dynamically adjustable. Another specialized component was the Overscan unit and frame buffer. In order to check nearby pixels for brightness, the frame buffer needed to be split into 16 block RAMs. Each block RAM corresponds to lines on the screen that alternate through all 16. This way, any 16 adjacent lines can be read simultaneously by the Overscan unit. In order to check if pixels to the left and right are bright, the Overscan unit merely reads the leading edge of the area prior to arriving at a pixel and then shifts that information out as it passes. The actual design has 9 rows of shift registers to store the current window of data. It then does a logarithmic compare to find the brightest value and dims the current pixel accordingly. That information is then passed to the LCD controller.

The software design required code to be programmed for an embedded microcontroller. The microcontroller is a modified CR-16 instruction set and all of the code was programmed in assembly language. The only function of the microcontroller is to change the values used in the pixel pipeline based on user input. It has a loop that consists of checking the inputs, calculating the new values, and storing them (repeat). A few complications arose due to the ability to zoom in and out at non-integer levels. In order to allow those operations, we needed a fractional multiply. And in order to bound the screen, we needed to write a fractional divide routine in software.

Software and Hardware Design Overview

A critical aspect of the glasses' operation relied on the relative positions between the camera and the screen. The fundamental problem is that what the camera can see is slightly different from what the person wearing the glasses can see. This phenomenon is referred to as parallax and can be used to measure distances, but in the sunglasses it is perceived as a variable error that is dependent on distance. The error is more apparent in objects close to the viewer and less apparent in objects at a large distance. The ideal location for the camera was exactly in the same place as the eye; this is obviously unfeasible as the eye occupies that space. The next most ideal location was in the center of the screen, or at least along the center line of the person's vision. This made the center of the camera the same as the center of the screen; unfortunately, this was also unfeasible as it would obscure the person's field of view. There was no perfect location for the camera to reside, but there were a few optimal locations given the previous physical constraints: above or below the screen in the center and to the left or right in the center. Each of these positions shares a center line with the human's eye and therefore minimizes either the side to side error or the up and down error (parallax). The camera position that was the most effective for this project was above the screens in the center as that was the most convenient for construction and the least likely spot to be accidentally damaged.

The cameras were placed as close to the LCDs as possible, but it was hard to know the exact measurements before they were physically set in place. Because the error was so heavily dependent on the exact location of the cameras, a simulation was required so that the camera's location could be changed and the respective error observed. In the simulation the viewer's eye was represented three inches behind the screen and the camera was represented an inch above and an inch forward from the screen. In the simulation there was one light source that could be moved around. Figure 1 shows this simulation. To represent what the viewer could see, the light source was projected onto the LCD screen. The light source was also projected onto a ¼ inch screen at the camera location and superimposed on top of the image on the LCD screen. This is shown in Figure 2. From there, the change in error could be observed depending on where the light source was located. Beyond one mile, the error is nonexistent; the camera sees exactly what the viewer sees. The sun would fall into the category of objects that will have no error between what the camera sees and what the viewer sees. On objects 7 feet away, the error becomes so significant that what the viewer sees is in a completely different location than what the camera sees. To account for this the sunglasses overscan what the camera sees to cover more area. The overscan could not be too great because it would obscure vision. A width of 2 millimeters, or 10 pixels, darkened in a border around the bright object would not be obtrusive to vision and would allow the LCD glasses to block light sources as close as 12 feet accurately. If the cameras were to be placed closer, that distance could be further decreased, but 12 feet was the worst case scenario.

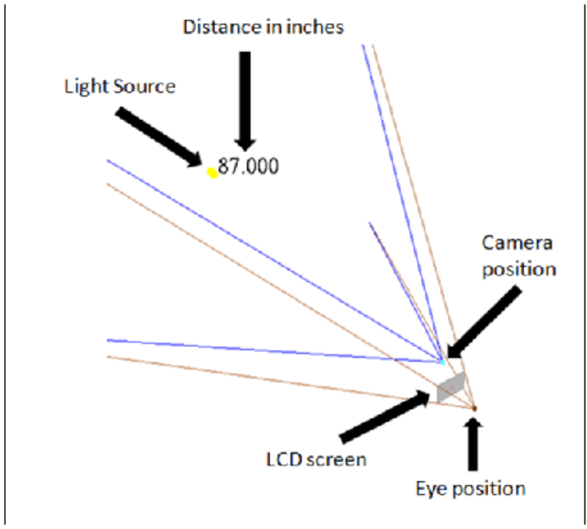


Figure 1: Glasses Operation Simulation External Perspective

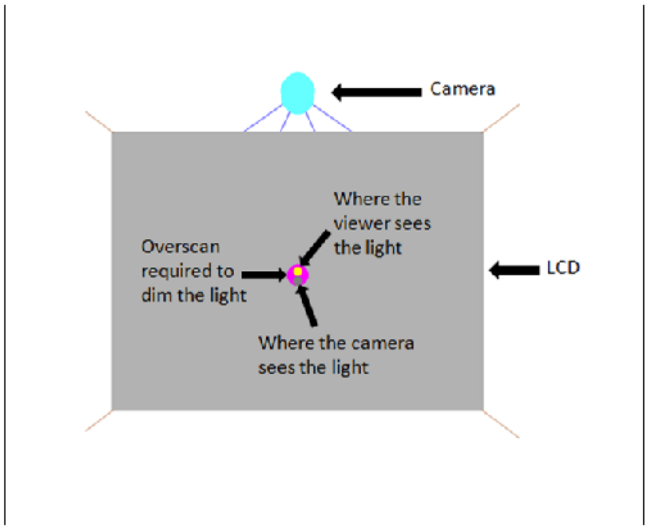

Camera
Where the viewer sees the light
Overscan required to dim the light
LCD
Where the camera sees the light

Figure 2: Glasses Operation Simulation Viewer's Perspective

Because the cameras send data to the microcontroller at a high frequency, we initially thought the processing time could be an issue if it was longer than the time it takes for the pupil to react. It turned out not to be a problem with the implementation of the pixel pipeline.

Pupil Response Time

A research study for the IEEE Engineering Medical Biology Society provided information on papillary response time. The group tested the papillary response time of subjects exposed to a single flash of light. The pupil response time was recorded for three seconds. The results are shown in Figure 3.

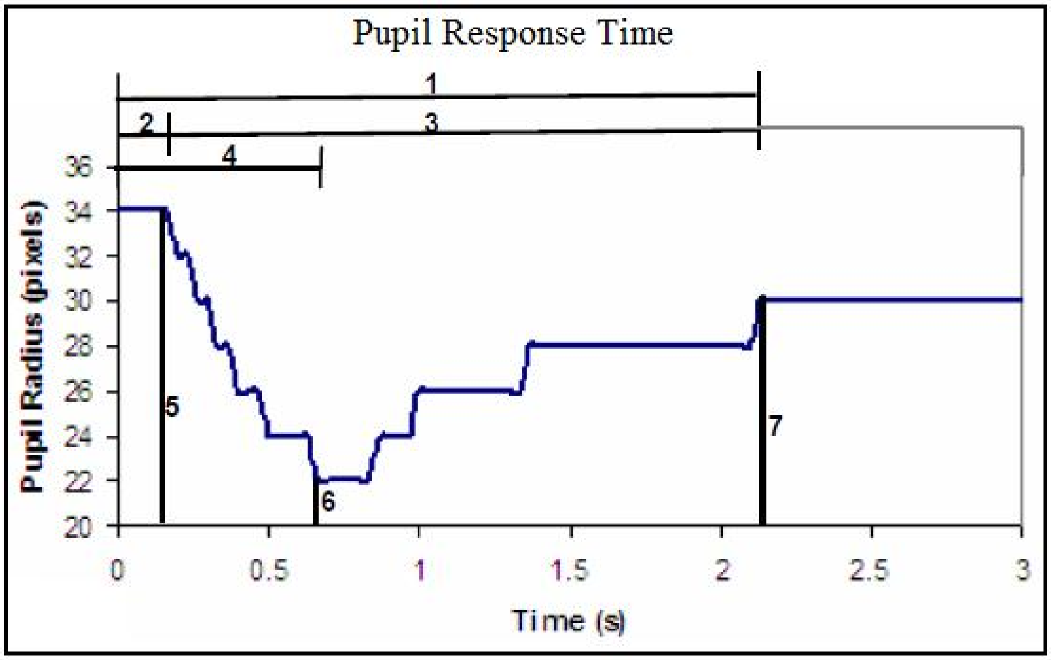


Fig. 3 Pupil response and parameters measure (1-Latency time to reach the plateau, 2-Latency time to the beginning of constriction, 3-Duration of constriction, 4-Latency time to maximum constriction, 5-Radius of pupil before flash, 6-Radius of pupil at maximum constriction, 7-Radius of pupil at teaching plateau).

The research group found that the latency time from the single flash exposure to the start of pupil constriction was (240±36ms). They also found that the time to total pupil constriction was between 0.6 and 0.7 seconds.

This data means that the code will have to be optimized so that the system processing time will fall below 240ms. The worst case scenario for the system's processing time will have to be shorter than the time it takes the eye to completely constrict which is between 0.6 and 0.7 seconds.

Bill of Materials

The sunglasses were comprised of a few parts from different manufacturers. They required LCDs, microcontroller boards, cameras, physical hardware for mounting, and various electrical components for interfacing. Table 1 shows the components and project costs.

Table 1: Projected Project Costs

Product	Quantity	Manufacturer	Price
LCD Screen LCD35VGAN	4	AIE Components	$977.00

LCD Screen PT0353224-A102	2	Palmtech	$50.00
Digital Camera C3188A-6018	3	OmniVision	$57.00
Spartan-3 FPGA Board	2	Digilent Inc.	$119.00
Physical Head Mounting Hardware	1	Industrial Supply Salt Lake City	$40.00
Assorted Electrical Components	1	-	$40.00
Total			$1283.00

The total project cost was 1283 dollars. We initially went with the PT0353224-A102 LCDs from Palmtech and acquired them early on in the project. As time went on however we were not able to interface with them. The Palmtech LCD pins were 0.5mm spaced and we were not equipped to handle such small spacing. We attempted but were not able to etch a fanned, pin out for them. We then went with the LCD35VGAN LCDs instead. Due to time constraints, we required an LCD module that contained a much easier interface but was also transmissive so that we could look through it. The LCD35VGAN LCDs worked perfectly as they were fully contained modules that were capable of VGA output. Shortly before we began mounting the LCDs onto the helmet we noticed that they were not responsive 100 percent of the time. We figured that due to abusive handling, the ribbon cables connecting the LCDs to their components began showing signs of metal fatigue (e.g. the wires inside cracked and would no longer make a reliable connection). We ordered two more LCDs and carefully mounted them to the helmet. Once in place the ribbon cables were secure and we encountered no further problems with them.

Microcontroller

The microcontroller was implemented using a field programmable gate array (FPGA). The FPGA was purchased from Digilent Inc. in the Spartan-3 Board package. The FPGA design was essential for the sunglasses due to their ability to sustain multiple parallel processing pipelines in the same package. The standard microcontroller, such as the HCS12 from Freescale, did not have the data throughput required for the sunglasses. The FPGA contained a 50 MHz crystal oscillator with 20 Block Rams containing a total of 360kbits of Block Ram. This allowed for high processor speed as well as memory for the camera and LCD data buffers.

Digital Cameras

The digital cameras that we purchased were model C3188A-6018 manufactured by Omnivision. The cameras were capable of operating at a resolution of 320x240 pixels at 60 frames per second, and a resolution of 640x480 pixels at 30 frames per second. High frame rates were ideal for fast response time (preventing blindness), and high resolutions were desirable for wide angle viewing and increased accuracy. The cameras were capable of outputting data in the YCrCb format, a format which allowed information regarding the brightness value of a pixel to be separated easily. The sunglasses only relied on the brightness (Y) values from the color vectors for processing. We also used some custom lens to get a wider field of view from each camera. A dark lens was also placed in front of each camera to simplify the calculation of the threshold for dimming pixels. This was mainly done to inhibit the cameras from gamma correcting.

Interfacing the Components

A few standard wires and adapters were required to interface each component together. The LCDs, Cameras, and FPGAs all communicated using a 3.3v TTL standard. There were also a few knobs and buttons required for individual adjustments. This was necessary since eye distance changes from person to person. The knobs controlled five variables: zoom in/out, up/down, left/right, threshold, and contrast. A photo of our adjustment control is shown in Figure 4. The total cost of miscellaneous electrical components was 40 dollars.

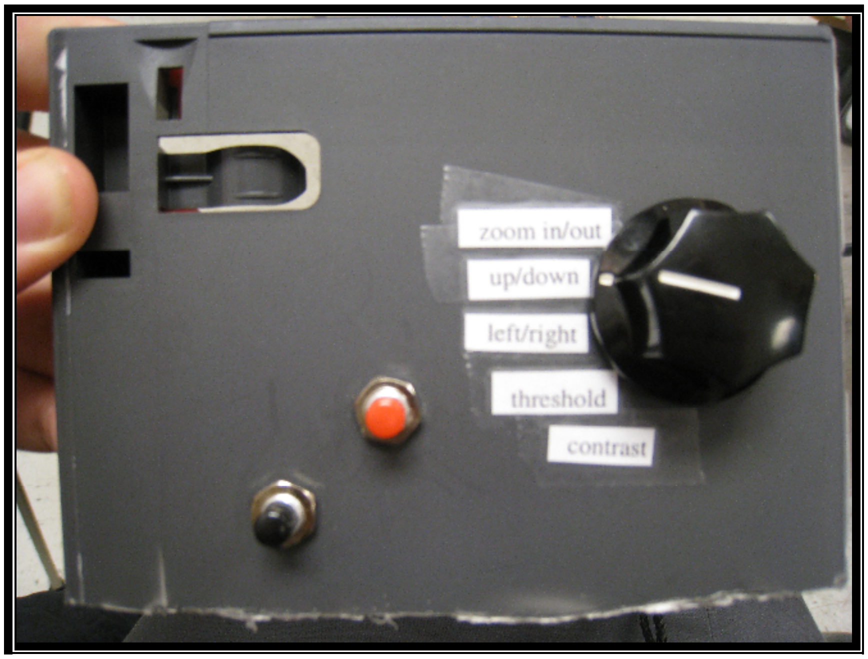


Fig. 4 Personal control unit

The sunglasses needed physical mounting gear in the form of a helmet with brackets to hold the LCDs and cameras in a rigid position. We chose a construction helmet since it was wearable by most people for demonstration purposes. The construction helmet was also durable enough for us to drill screws into it. The screws held two platforms on each side of the head to hold the microcontroller components. We used copper piping ties to mount the LCDs and cameras to the front of the helmet. Since we needed a gentle connection we used weather stripping to form a connection between the components and the copper. Once we were satisfied with the camera positions above the LCDs we soldered them into place. We also soldered the darkening glasses over the cameras to account for the gamma correct from the cameras. A photo of our components interfaced together is shown in Figure 5. Most parts involved in the physical construction were purchased at home depot and the total cost was 40 dollars.

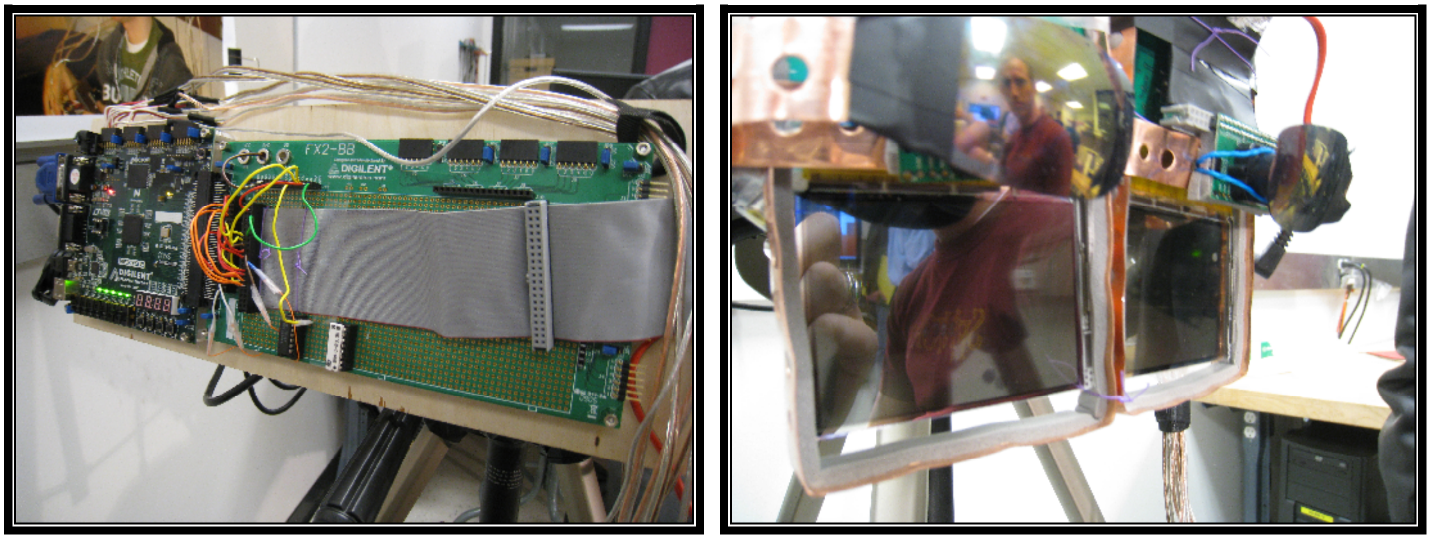


Fig. 5 Interfaced Components. Left: FPGA control Right: LCDs and Cameras with Gamma correction lenses.

Interface Issues

Several interface issues were discovered while performing initial project scoping. One such issue definitely affected our ability to interface with our LCD screens. Our initial LCD solution interfaces through 54 pin ribbon cable with a 0.5mm spacing. We quickly found that we were not able to interface with the 0.5mm spacing and chose to go with the LCD35VGAN LCD modules instead. They allowed us to use a standard VGA interface to communicate to them.

When connecting one of the cameras to the FPGA board, we noticed that there was substantially more noise on its screen than its duplicate sister-system had. To fix this we wrapped the cable in aluminum foil (used Chipotle wrappers) and grounded the foil. This completely removed the noise.

Another issue presented itself in the form of interfacing with the camera implementation. While attempting to make use of the I2C interface within the camera, we noticed that we were not able to get the camera to accept any data that we were sending. The use of the I2C interface was meant to control all aspects of the camera itself. Such settings as

Hue, Gamma Correction, Contrast, Saturation, and more are all determined by hardware registers within the camera. Originally we had designed an I2C controller on our FPGA in order to disable gamma correction within the camera but were unable to figure out why the camera was not accepting commands. To counter this, we attached dark sunglass lenses to the front of the cameras to disable gamma correction externally for our purposes.

Testing

The error on the sunglasses was small enough so that is didn't obscure the vision of the wearer and didn't miss light that required coverage. The error was measured using one main light sources: A 500 watt work lamp. The work lamp was used in lab testing as a variable distance light source to test the error on objects ranging from five to 30 feet away.

Testing was also performed on the physical head gear that was used to mount the LCDs, camera's, etc. This testing on the helmet or glasses apparatus was on all 4 members of the group as well as several students chosen at random from either the Warnock or Merrill Engineering Buildings to ensure a good fit across multiple head sizes.

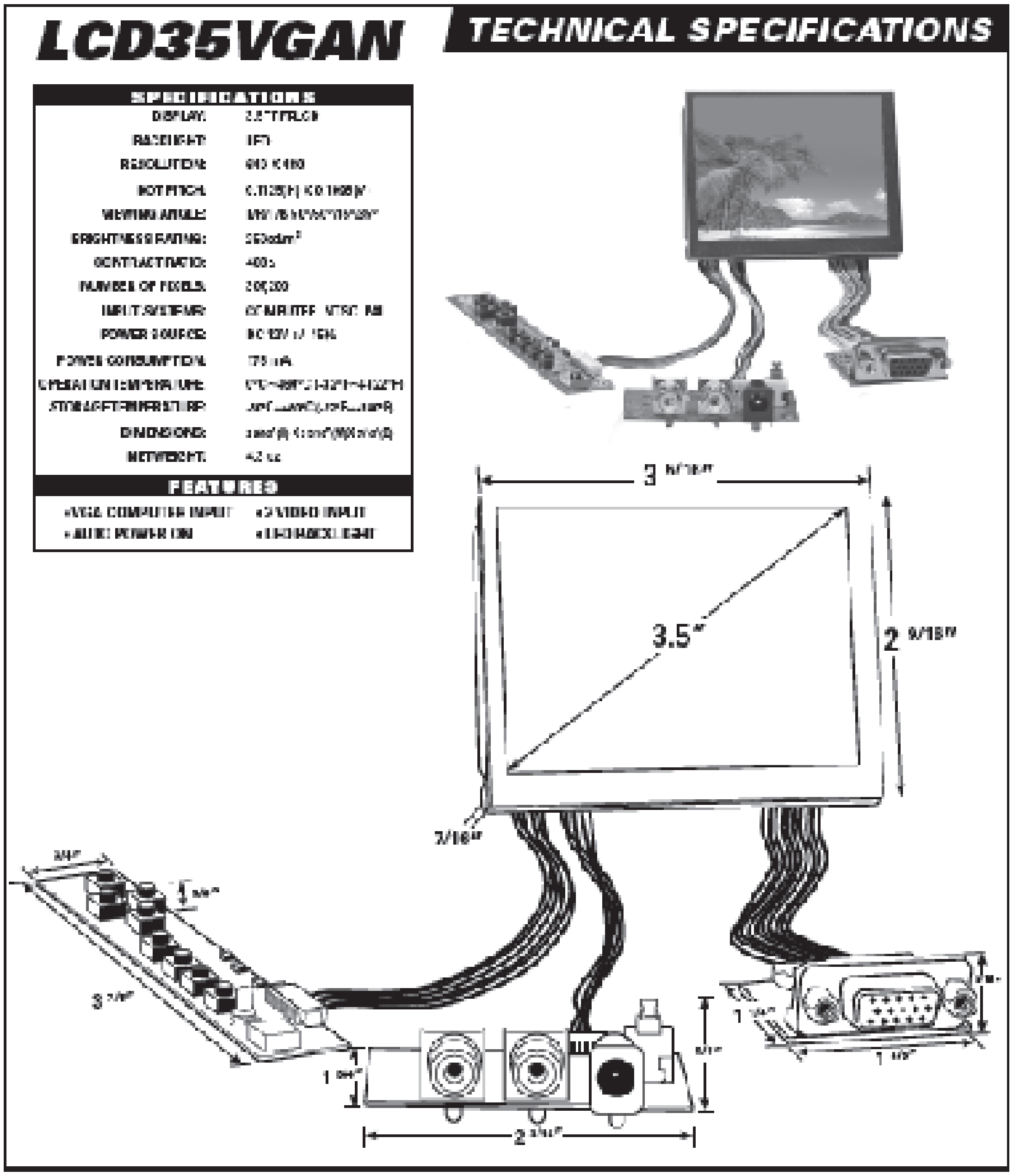
Because the LCDs we eventually used had a VGA interface, testing them consisted of loading a program that was known to have a working VGA output and observing that on the screens. After we confirmed that we could communicate, we loaded a program that had a black spot in an otherwise white background. We looked through that scene at lights to confirm that the black was dark enough, and the white was clear. At that point we knew the LCDs worked in the way we needed, and we knew could interface to them with ease.

Testing on the camera modules initially consisted of observing waveforms on an oscilloscope to ensure that the camera was indeed set to a default setting that we were familiar with, YUV. After the data was found to be correct with our own intuition, we resorted to manual observation of the data the camera was outputting. In order to do this, we designed a camera controller within our FPGA and attached the camera to the board. Once we verified that we could capture data by lighting up debug LEDs on the project board, we integrated the controller to our project so that we could make use of the frame buffer and VGA controller to actually observe, in real-time, the images the camera was seeing on an external CRT monitor in the hardware lab. After we were able to observe actual images being displayed on an external monitor, all subsequent testing of the camera simply involved observing the actual frames that the camera sent to the FPGA.

Appendix A - References

Ferrari GL, Marques JL, Gandhi RA, Emery CJ, Tesfaye S, Heller SR, Schneider FK, Gamba HR., **An approach to the assessment of diabetic neuropathy based on dynamic pupillometry.,** Conf Proc IEEE Eng Med Biol Soc. 2007;2007:557-60.

APPENDIX B - LCD Specs



LCD35VGAN **TECHNICAL SPECIFICATIONS**

SPECIFICATIONS	
DISPLAY:	3.5" TFT LCD
BACKLIGHT:	LED
RESOLUTION:	640 X 480
DOT PITCH:	0.1125(H) X 0.1065(V)
VIEWING ANGLE:	[illegible]
BRIGHTNESS RATING:	250cd/m²
CONTRAST RATIO:	400:1
NUMBER OF PIXELS:	230,400
INPUT SYSTEMS:	COMPUTER, NTSC, PAL
POWER SOURCE:	DC 12V +/- 10%
POWER CONSUMPTION:	175 mA
OPERATING TEMPERATURE:	[illegible]
STORAGE TEMPERATURE:	[illegible]
DIMENSIONS:	[illegible]
NET WEIGHT:	4.2 oz

FEATURES

- VGA COMPUTER INPUT
- 2 VIDEO INPUTS
- AUTO POWER ON
- LED BACKLIGHT

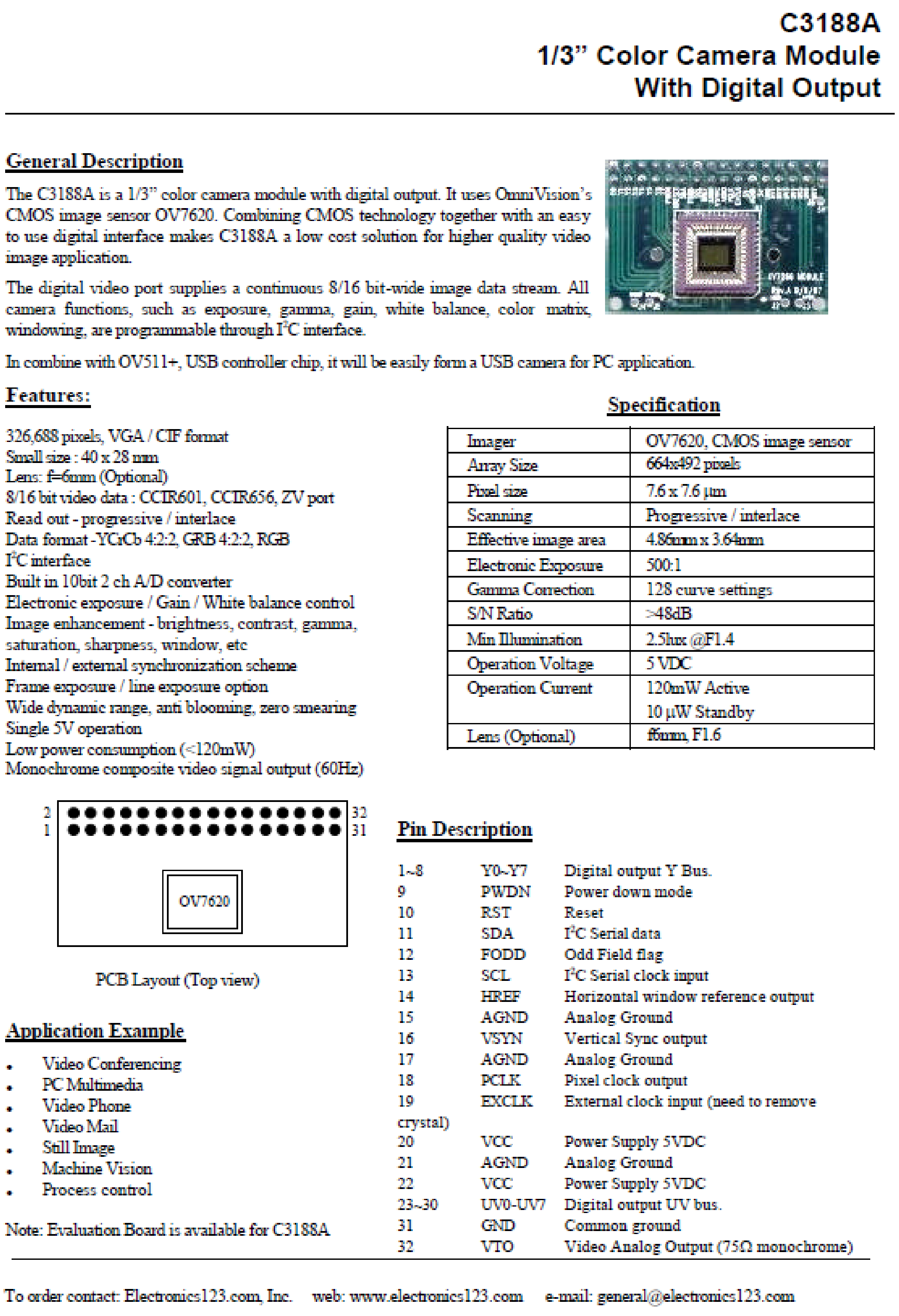
AEI Components
A Division of [illegible]

www.aeicorp.com

CONTACT:___________________________

(562) 809-8262 • FAX: (562) 856-5640

APPENDIX C - Camera Specs

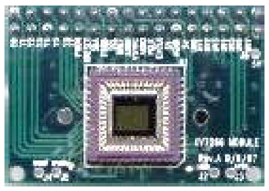


C3188A
1/3" Color Camera Module
With Digital Output

General Description

The C3188A is a 1/3" color camera module with digital output. It uses OmniVision's CMOS image sensor OV7620. Combining CMOS technology together with an easy to use digital interface makes C3188A a low cost solution for higher quality video image application.

The digital video port supplies a continuous 8/16 bit-wide image data stream. All camera functions, such as exposure, gamma, gain, white balance, color matrix, windowing, are programmable through I^{2}C interface.



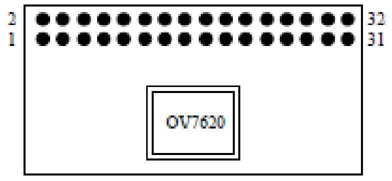
In combine with OV511+, USB controller chip, it will be easily form a USB camera for PC application.

Features:

326,688 pixels, VGA / CIF format
Small size : 40 x 28 mm
Lens: f=6mm (Optional)
8/16 bit video data : CCIR601, CCIR656, ZV port
Read out - progressive / interlace
Data format -YCrCb 4:2:2, GRB 4:2:2, RGB
I^{2}C interface
Built in 10bit 2 ch A/D converter
Electronic exposure / Gain / White balance control
Image enhancement - brightness, contrast, gamma, saturation, sharpness, window, etc
Internal / external synchronization scheme
Frame exposure / line exposure option
Wide dynamic range, anti blooming, zero smearing
Single 5V operation
Low power consumption (<120mW)
Monochrome composite video signal output (60Hz)

Specification

Imager	OV7620, CMOS image sensor
Array Size	664x492 pixels
Pixel size	7.6 x 7.6 µm
Scanning	Progressive / interlace
Effective image area	4.86mm x 3.64mm
Electronic Exposure	500:1
Gamma Correction	128 curve settings
S/N Ratio	>48dB
Min Illumination	2.5lux @F1.4
Operation Voltage	5 VDC
Operation Current	120mW Active 10 µW Standby
Lens (Optional)	f6mm, F1.6



PCB Layout (Top view)

Pin Description

1~8	Y0~Y7	Digital output Y Bus.
9	PWDN	Power down mode
10	RST	Reset
11	SDA	I^{2}C Serial data
12	FODD	Odd Field flag
13	SCL	I^{2}C Serial clock input
14	HREF	Horizontal window reference output
15	AGND	Analog Ground
16	VSYN	Vertical Sync output
17	AGND	Analog Ground
18	PCLK	Pixel clock output
19	EXCLK	External clock input (need to remove crystal)
20	VCC	Power Supply 5VDC
21	AGND	Analog Ground
22	VCC	Power Supply 5VDC
23~30	UV0-UV7	Digital output UV bus.
31	GND	Common ground
32	VTO	Video Analog Output (75Ω monochrome)

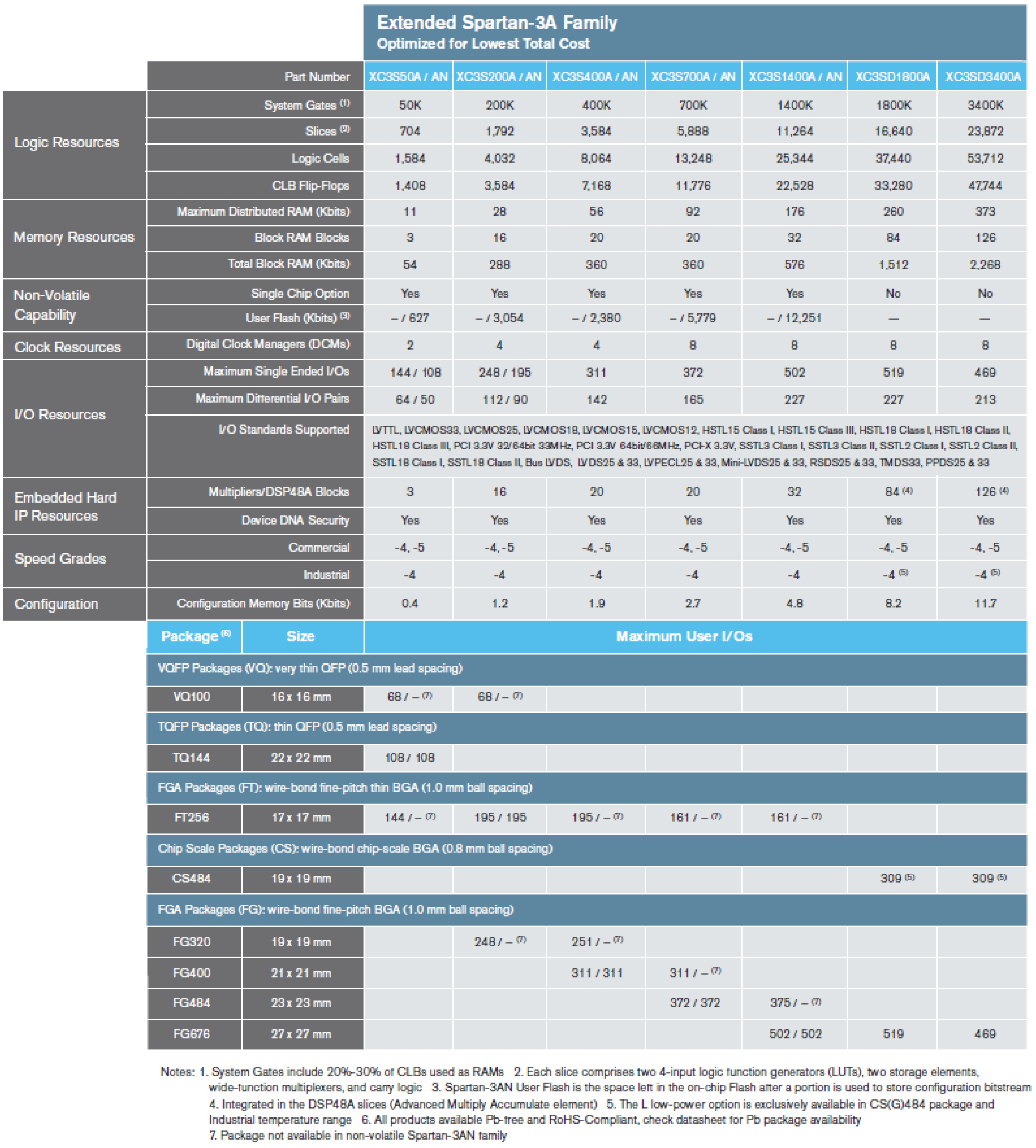
Application Example

- Video Conferencing
- PC Multimedia
- Video Phone
- Video Mail
- Still Image
- Machine Vision
- Process control

Note: Evaluation Board is available for C3188A

To order contact: Electronics123.com, Inc. web: www.electronics123.com e-mail: general@electronics123.com

APPENDIX D – Microcontroller Specs

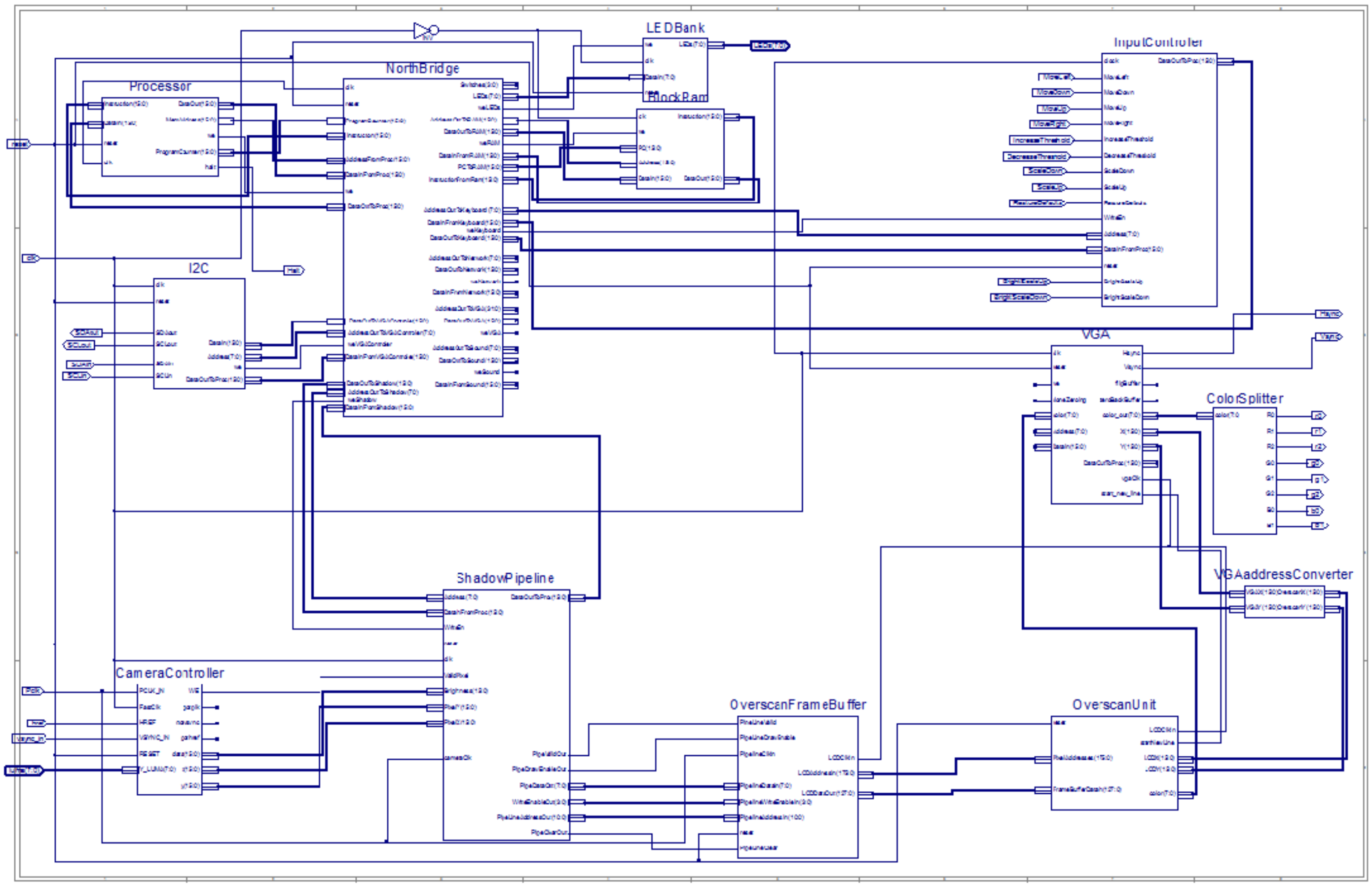


Extended Spartan-3A Family
Optimized for Lowest Total Cost

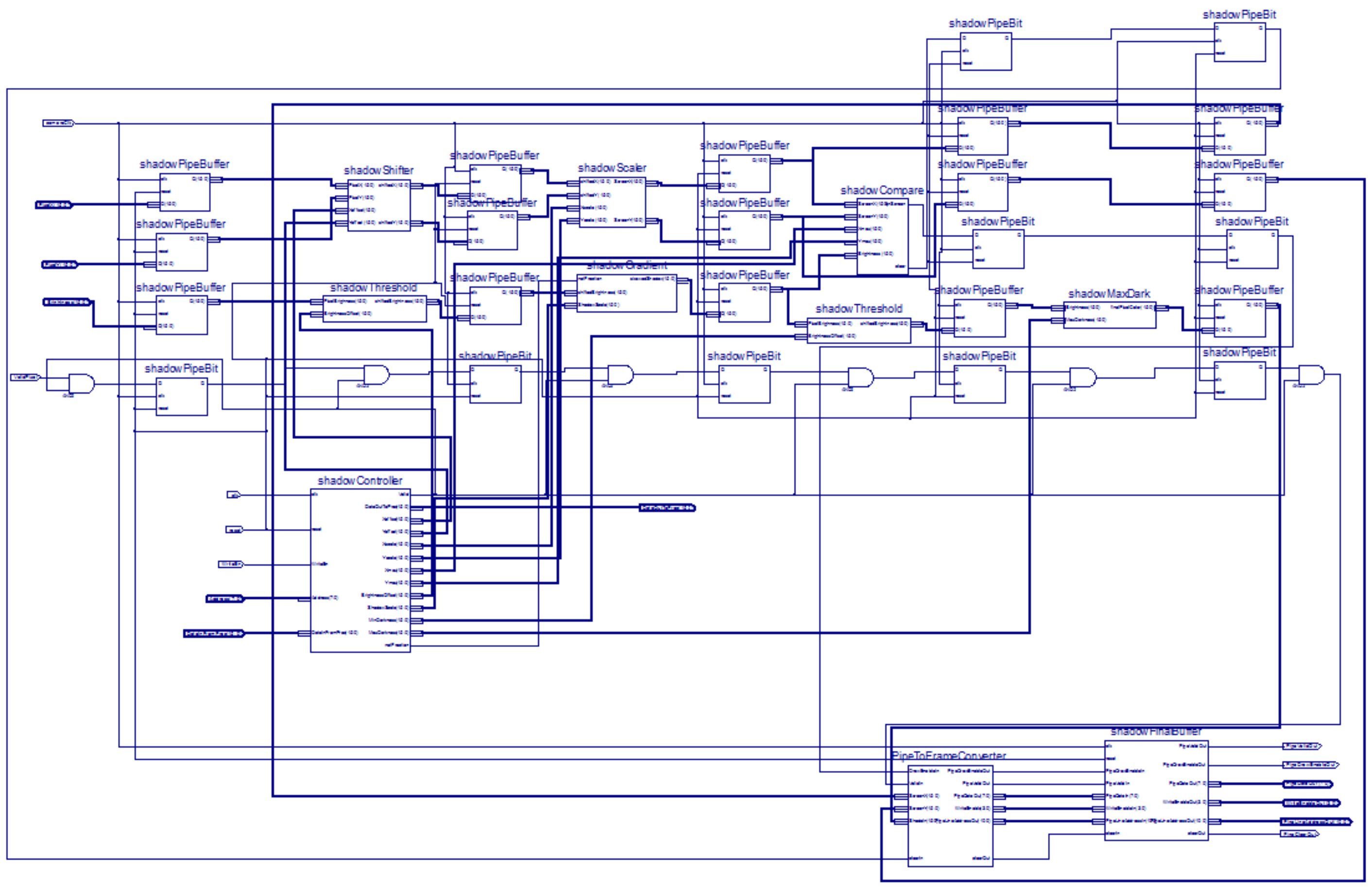
	Part Number	XC3S50A / AN	XC3S200A / AN	XC3S400A / AN	XC3S700A / AN	XC3S1400A / AN	XC3SD1800A	XC3SD3400A
Logic Resources	System Gates [1]	50K	200K	400K	700K	1400K	1800K	3400K
	Slices [2]	704	1,792	3,584	5,888	11,264	16,640	23,872
	Logic Cells	1,584	4,032	8,064	13,248	25,344	37,440	53,712
	CLB Flip-Flops	1,408	3,584	7,168	11,776	22,528	33,280	47,744
Memory Resources	Maximum Distributed RAM (Kbits)	11	28	56	92	176	260	373
	Block RAM Blocks	3	16	20	20	32	84	126
	Total Block RAM (Kbits)	54	288	360	360	576	1,512	2,268
Non-Volatile Capability	Single Chip Option	Yes	Yes	Yes	Yes	Yes	No	No
	User Flash (Kbits) [3]	– / 627	– / 3,054	– / 2,380	– / 5,779	– / 12,251	—	—
Clock Resources	Digital Clock Managers (DCMs)	2	4	4	8	8	8	8
I/O Resources	Maximum Single Ended I/Os	144 / 108	248 / 195	311	372	502	519	469
	Maximum Differential I/O Pairs	64 / 50	112 / 90	142	165	227	227	213
	I/O Standards Supported	LVTTL, LVCMOS33, LVCMOS25, LVCMOS18, LVCMOS15, LVCMOS12, HSTL15 Class I, HSTL15 Class III, HSTL18 Class I, HSTL18 Class II, HSTL18 Class III, PCI 3.3V 32/64bit 33MHz, PCI 3.3V 64bit/66MHz, PCI-X 3.3V, SSTL3 Class I, SSTL3 Class II, SSTL2 Class I, SSTL2 Class II, SSTL18 Class I, SSTL18 Class II, Bus LVDS, LVDS25 & 33, LVPECL25 & 33, Mini-LVDS25 & 33, RSDS25 & 33, TMDS33, PPDS25 & 33						
Embedded Hard IP Resources	Multipliers/DSP48A Blocks	3	16	20	20	32	84 [4]	126 [4]
	Device DNA Security	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Speed Grades	Commercial	-4, -5	-4, -5	-4, -5	-4, -5	-4, -5	-4, -5	-4, -5
	Industrial	-4	-4	-4	-4	-4	-4 [5]	-4 [5]
Configuration	Configuration Memory Bits (Kbits)	0.4	1.2	1.9	2.7	4.8	8.2	11.7
Package [6]	**Size**	**Maximum User I/Os**						
VQFP Packages (VQ): very thin QFP (0.5 mm lead spacing)								
VQ100	16 x 16 mm	68 / – [7]	68 / – [7]					
TQFP Packages (TQ): thin QFP (0.5 mm lead spacing)								
TQ144	22 x 22 mm	108 / 108						
FGA Packages (FT): wire-bond fine-pitch thin BGA (1.0 mm ball spacing)								
FT256	17 x 17 mm	144 / – [7]	195 / 195	195 / – [7]	161 / – [7]	161 / – [7]		
Chip Scale Packages (CS): wire-bond chip-scale BGA (0.8 mm ball spacing)								
CS484	19 x 19 mm						309 [5]	309 [5]
FGA Packages (FG): wire-bond fine-pitch BGA (1.0 mm ball spacing)								
FG320	19 x 19 mm		248 / – [7]	251 / – [7]				
FG400	21 x 21 mm			311 / 311	311 / – [7]			
FG484	23 x 23 mm				372 / 372	375 / – [7]		
FG676	27 x 27 mm					502 / 502	519	469

Notes: 1. System Gates include 20%-30% of CLBs used as RAMs 2. Each slice comprises two 4-input logic function generators (LUTs), two storage elements, wide-function multiplexers, and carry logic 3. Spartan-3AN User Flash is the space left in the on-chip Flash after a portion is used to store configuration bitstream 4. Integrated in the DSP48A slices (Advanced Multiply Accumulate element) 5. The L low-power option is exclusively available in CS(G)484 package and Industrial temperature range 6. All products available Pb-free and RoHS-Compliant, check datasheet for Pb package availability 7. Package not available in non-volatile Spartan-3AN family

APPENDIX E – the Complete System on the FPGA



APPENDIX F – The Shadow Pipeline



APPENDIX G – The Camera Controller

```
module CameraController(
       input PCLK_IN,
       input FastClk,
       input [7:0] Y_LUMA,
       input HREF,
       input VSYNC_IN,
       input RESET,
       output reg WE,
       output reg [15:0] data,
       output reg [15:0] x ,
       output reg [15:0] y,
       output reg gotplk = 0,
       output reg gotvsync = 0,
       output reg gothref = 0

    );
       reg [15:0] HsyncCount = 0;
       reg [15:0] VsyncCount = 0;
       reg [25:0] PclkCount = 0;

       always @(posedge FastClk)
       if (PCLK_IN)
         gotplk <= 1;

       always @(posedge FastClk)
       if (VSYNC_IN)
         gotvsync <= 1;

       always @(posedge FastClk)
       if (HREF)
         gothref <= 1;
       //Resolution = 640x480
        //10bits for 640 dec representation
       //9bits for 480 dec representation
       //Y is within range 16 < Y < 235

       reg prevH;
       always@(posedge PCLK_IN)
       begin
              if(RESET == 1)
              begin
                     prevH<=0;
                     x<=0;
                     y<=0;

              end
              else
              begin
                     prevH<=HREF;
                     if(HREF == 1)
                     begin
                            x <= x + 1;
                     end
                     else if(VSYNC_IN)
                     begin
```

```
                              x <= 0;
                              y <= 0;
                      end
                      else if(HREF == 0 && prevH== 1)
                      begin
                         y <= y + 1;
                         x <= 0;
                      end
              end
       end
       always @ (posedge FastClk)
       begin
              WE<=HREF;
              data <= {8'b00000000,Y_LUMA};
       end

endmodule
```

APPENDIX H – The Frame Buffer

```
endmodule module OverscanFrameBuffer(
    input [7:0] PipelineDataIn,
    input [3:0] PipelineWriteEnableIn,
        input PipeLineValid,
        input PipeLineDrawEnable,
        input PipeLineClear,
    input PipelineClkIn,
    input [10:0] PipelineAddressIn,
    output [127:0] LCDDataOut,
    input [175:0] LCDAddressIn,
    input LCDClkIn,
        input reset
    );

wire we;
assign we=PipeLineValid & PipeLineDrawEnable;
wire [7:0] colorValue;

assign colorValue=PipelineDataIn & {8{PipeLineClear}};

reg [7:0] Line0 [1999:0];
reg [7:0] Line1 [1999:0];
reg [7:0] Line2 [1999:0];
reg [7:0] Line3 [1999:0];
reg [7:0] Line4 [1999:0];
reg [7:0] Line5 [1999:0];
reg [7:0] Line6 [1999:0];
reg [7:0] Line7 [1999:0];
reg [7:0] Line8 [1999:0];
reg [7:0] Line9 [1999:0];
reg [7:0] Line10 [1999:0];
reg [7:0] Line11 [1999:0];
reg [7:0] Line12 [1999:0];
reg [7:0] Line13 [1999:0];
reg [7:0] Line14 [1999:0];
reg [7:0] Line15 [1999:0];


initial
begin
$readmemh("blankFrameBuf.dat", Line0);
$readmemh("blankFrameBuf.dat", Line1);
$readmemh("blankFrameBuf.dat", Line2);
$readmemh("blankFrameBuf.dat", Line3);
$readmemh("blankFrameBuf.dat", Line4);
$readmemh("blankFrameBuf.dat", Line5);
$readmemh("blankFrameBuf.dat", Line6);
$readmemh("blankFrameBuf.dat", Line7);
$readmemh("blankFrameBuf.dat", Line8);
$readmemh("blankFrameBuf.dat", Line9);
$readmemh("blankFrameBuf.dat", Line10);
$readmemh("blankFrameBuf.dat", Line11);
$readmemh("blankFrameBuf.dat", Line12);
$readmemh("blankFrameBuf.dat", Line13);
$readmemh("blankFrameBuf.dat", Line14);
```

```
$readmemh("blankFrameBuf.dat", Line15);
end

reg [7:0] Line0out;
reg [7:0] Line1out;
reg [7:0] Line2out;
reg [7:0] Line3out;
reg [7:0] Line4out;
reg [7:0] Line5out;
reg [7:0] Line6out;
reg [7:0] Line7out;
reg [7:0] Line8out;
reg [7:0] Line9out;
reg [7:0] Line10out;
reg [7:0] Line11out;
reg [7:0] Line12out;
reg [7:0] Line13out;
reg [7:0] Line14out;
reg [7:0] Line15out;

assign LCDDataOut={Line15out,Line14out,Line13out,Line12out,Line11out,Line10out,
                   Line9out,Line8out,Line7out,Line6out,Line5out,Line4out,
                                              Line3out,Line2out,Line1out,Line0out};

reg [15:0] individualWE;
always @(*)
begin
case({we,PipelineWriteEnableIn})
      5'b10000: individualWE <= 16'b0000000000000001;
      5'b10001: individualWE <= 16'b0000000000000010;
      5'b10010: individualWE <= 16'b0000000000000100;
      5'b10011: individualWE <= 16'b0000000000001000;
      5'b10100: individualWE <= 16'b0000000000010000;
      5'b10101: individualWE <= 16'b0000000000100000;
      5'b10110: individualWE <= 16'b0000000001000000;
      5'b10111: individualWE <= 16'b0000000010000000;
      5'b11000: individualWE <= 16'b0000000100000000;
      5'b11001: individualWE <= 16'b0000001000000000;
               5'b11010: individualWE <= 16'b0000010000000000;
      5'b11011: individualWE <= 16'b0000100000000000;
      5'b11100: individualWE <= 16'b0001000000000000;
      5'b11101: individualWE <= 16'b0010000000000000;
      5'b11110: individualWE <= 16'b0100000000000000;
      5'b11111: individualWE <= 16'b1000000000000000;
default:  individualWE <= 16'b0000000000000000;
endcase
end


always@(posedge PipelineClkIn)
begin
  if (individualWE[0])
      Line0[PipelineAddressIn] <= colorValue;
  if (individualWE[1])
      Line1[PipelineAddressIn] <= colorValue;
  if (individualWE[2])
      Line2[PipelineAddressIn] <= colorValue;
  if (individualWE[3])
      Line3[PipelineAddressIn] <= colorValue;
  if (individualWE[4])
      Line4[PipelineAddressIn] <= colorValue;
  if (individualWE[5])
      Line5[PipelineAddressIn] <= colorValue;
  if (individualWE[6])
      Line6[PipelineAddressIn] <= colorValue;
  if (individualWE[7])
      Line7[PipelineAddressIn] <= colorValue;
  if (individualWE[8])
      Line8[PipelineAddressIn] <= colorValue;
  if (individualWE[9])
      Line9[PipelineAddressIn] <= colorValue;
  if (individualWE[10])
      Line10[PipelineAddressIn] <= colorValue;
  if (individualWE[11])
      Line11[PipelineAddressIn] <= colorValue;
  if (individualWE[12])
      Line12[PipelineAddressIn] <= colorValue;
  if (individualWE[13])
      Line13[PipelineAddressIn] <= colorValue;
  if (individualWE[14])
      Line14[PipelineAddressIn] <= colorValue;
  if (individualWE[15])
      Line15[PipelineAddressIn] <= colorValue;
end

always@(posedge LCDClkIn)
begin
       Line0out <= Line0[&LCDAddressIn[10:0] ? 1999 : LCDAddressIn[10:0]];
       Line1out <= Line1[&LCDAddressIn[21:11] ? 1999 : LCDAddressIn[21:11]];
       Line2out <= Line2[&LCDAddressIn[32:22] ? 1999 : LCDAddressIn[32:22]];
       Line3out <= Line3[&LCDAddressIn[43:33] ? 1999 : LCDAddressIn[43:33]];
       Line4out <= Line4[&LCDAddressIn[54:44] ? 1999 : LCDAddressIn[54:44]];
       Line5out <= Line5[&LCDAddressIn[65:55] ? 1999 : LCDAddressIn[65:55]];
       Line6out <= Line6[&LCDAddressIn[76:66] ? 1999 : LCDAddressIn[76:66]];
       Line7out <= Line7[&LCDAddressIn[87:77] ? 1999 : LCDAddressIn[87:77]];
       Line8out <= Line8[&LCDAddressIn[98:88] ? 1999 : LCDAddressIn[98:88]];
       Line9out <= Line9[&LCDAddressIn[109:99] ? 1999 : LCDAddressIn[109:99]];
       Line10out <= Line10[&LCDAddressIn[120:110] ? 1999 : LCDAddressIn[120:110]];
       Line11out <= Line11[&LCDAddressIn[131:121] ? 1999 : LCDAddressIn[131:121]];
       Line12out <= Line12[&LCDAddressIn[142:132] ? 1999 : LCDAddressIn[142:132]];
       Line13out <= Line13[&LCDAddressIn[153:143] ? 1999 : LCDAddressIn[153:143]];
       Line14out <= Line14[&LCDAddressIn[164:154] ? 1999 : LCDAddressIn[164:154]];
       Line15out <= Line15[&LCDAddressIn[175:165] ? 1999 : LCDAddressIn[175:165]];
end
endmodule
```

APPENDIX I – The Overscan Unit

```
module OverscanUnit(
    input [127:0] FrameBufferDatain,
    output reg [175:0] PixelAddresses,
    input LCDClkIn,
         input reset,
         input startNewLine,
         output[7:0] color,
         input [15:0] LCDX,
         input [15:0] LCDY
    );


reg [15:0] Xline0;
reg [15:0] Xline1;
reg [15:0] Xline2;
reg [15:0] Xline3;
reg [15:0] Xline4;
reg [15:0] Xline5;
reg [15:0] Xline6;
reg [15:0] Xline7;
reg [15:0] Xline8;
reg [15:0] Xline9;
reg [15:0] Xline10;
reg [15:0] Xline11;
reg [15:0] Xline12;
reg [15:0] Xline13;
reg [15:0] Xline14;
reg [15:0] Xline15;

always @ ( * )
begin
Xline0 <= LCDX;
Xline1 <= LCDX;
Xline2 <= LCDX;
Xline3 <= LCDX;
Xline4 <= LCDX;
Xline5 <= LCDX;
Xline6 <= LCDX;
Xline7 <= LCDX;
Xline8 <= LCDX;
Xline9 <= LCDX;
Xline10 <= LCDX;
Xline11 <= LCDX;
Xline12 <= LCDX;
Xline13 <= LCDX;
Xline14 <= LCDX;
Xline15 <= LCDX;
case (LCDY[3:0])
   0:begin
           Xline12 <= LCDX+1;
           Xline13 <= LCDX+3;
           Xline14 <= LCDX+3;
           Xline15 <= LCDX+4;
           Xline0 <= LCDX+4;
           Xline1 <= LCDX+4;
```

```
        Xline2 <= LCDX+3;
        Xline3 <= LCDX+3;
        Xline4 <= LCDX+1;
      end
1:begin
        Xline13 <= LCDX+1;
        Xline14 <= LCDX+3;
        Xline15 <= LCDX+3;
        Xline0 <= LCDX+4;
        Xline1 <= LCDX+4;
        Xline2 <= LCDX+4;
        Xline3 <= LCDX+3;
        Xline4 <= LCDX+3;
        Xline5 <= LCDX+1;
      end
2:begin
        Xline14 <= LCDX+1;
        Xline15 <= LCDX+3;
        Xline0 <= LCDX+3;
        Xline1 <= LCDX+4;
        Xline2 <= LCDX+4;
        Xline3 <= LCDX+4;
        Xline4 <= LCDX+3;
        Xline5 <= LCDX+3;
        Xline6 <= LCDX+1;
      end
3:begin
        Xline15 <= LCDX+1;
        Xline0 <= LCDX+3;
        Xline1 <= LCDX+3;
        Xline2 <= LCDX+4;
        Xline3 <= LCDX+4;
        Xline4 <= LCDX+4;
        Xline5 <= LCDX+3;
        Xline6 <= LCDX+3;
        Xline7 <= LCDX+1;
      end
4:begin
        Xline0 <= LCDX+1;
        Xline1 <= LCDX+3;
        Xline2 <= LCDX+3;
        Xline3 <= LCDX+4;
        Xline4 <= LCDX+4;
        Xline5 <= LCDX+4;
        Xline6 <= LCDX+3;
        Xline7 <= LCDX+3;
        Xline8 <= LCDX+1;
      end
5:begin
        Xline1 <= LCDX+1;
        Xline2 <= LCDX+3;
        Xline3 <= LCDX+3;
        Xline4 <= LCDX+4;
        Xline5 <= LCDX+4;
        Xline6 <= LCDX+4;
        Xline7 <= LCDX+3;
        Xline8 <= LCDX+3;
        Xline9 <= LCDX+1;
      end
6:begin
        Xline2 <= LCDX+1;
        Xline3 <= LCDX+3;
        Xline4 <= LCDX+3;
        Xline5 <= LCDX+4;
        Xline6 <= LCDX+4;
        Xline7 <= LCDX+4;
        Xline8 <= LCDX+3;
        Xline9 <= LCDX+3;
        Xline10 <= LCDX+1;
      end
7:begin
```

```
        Xline3 <= LCDX+1;
        Xline4 <= LCDX+3;
        Xline5 <= LCDX+3;
        Xline6 <= LCDX+4;
        Xline7 <= LCDX+4;
        Xline8 <= LCDX+4;
        Xline9 <= LCDX+3;
        Xline10 <= LCDX+3;
        Xline11 <= LCDX+1;
      end
8:begin
        Xline4 <= LCDX+1;
        Xline5 <= LCDX+3;
        Xline6 <= LCDX+3;
        Xline7 <= LCDX+4;
        Xline8 <= LCDX+4;
        Xline9 <= LCDX+4;
        Xline10 <= LCDX+3;
        Xline11 <= LCDX+3;
        Xline12 <= LCDX+1;
      end
9:begin
        Xline5 <= LCDX+1;
        Xline6 <= LCDX+3;
        Xline7 <= LCDX+3;
        Xline8 <= LCDX+4;
        Xline9 <= LCDX+4;
        Xline10 <= LCDX+4;
        Xline11 <= LCDX+3;
        Xline12 <= LCDX+3;
        Xline13 <= LCDX+1;
      end
10:begin
        Xline6 <= LCDX+1;
        Xline7 <= LCDX+3;
        Xline8 <= LCDX+3;
        Xline9 <= LCDX+4;
        Xline10 <= LCDX+4;
        Xline11 <= LCDX+4;
        Xline12 <= LCDX+3;
        Xline13 <= LCDX+3;
        Xline14 <= LCDX+1;
      end
11:begin
        Xline7 <= LCDX+1;
        Xline8 <= LCDX+3;
        Xline9 <= LCDX+3;
        Xline10 <= LCDX+4;
        Xline11 <= LCDX+4;
        Xline12 <= LCDX+4;
        Xline13 <= LCDX+3;
        Xline14 <= LCDX+3;
        Xline15 <= LCDX+1;
      end
12:begin
        Xline8 <= LCDX+1;
        Xline9 <= LCDX+3;
        Xline10 <= LCDX+3;
        Xline11 <= LCDX+4;
        Xline12 <= LCDX+4;
        Xline13 <= LCDX+4;
        Xline14 <= LCDX+3;
        Xline15 <= LCDX+3;
        Xline0 <= LCDX+1;
      end
13:begin
        Xline9 <= LCDX+1;
        Xline10 <= LCDX+3;
        Xline11 <= LCDX+3;
        Xline12 <= LCDX+4;
        Xline13 <= LCDX+4;
```

```
            Xline14 <= LCDX+4;
            Xline15 <= LCDX+3;
            Xline0 <= LCDX+3;
            Xline1 <= LCDX+1;
          end
   14:begin
            Xline10 <= LCDX+1;
            Xline11 <= LCDX+3;
            Xline12 <= LCDX+3;
            Xline13 <= LCDX+4;
            Xline14 <= LCDX+4;
            Xline15 <= LCDX+4;
            Xline0 <= LCDX+3;
            Xline1 <= LCDX+3;
            Xline2 <= LCDX+1;
          end
   15:begin
            Xline11 <= LCDX+1;
            Xline12 <= LCDX+3;
            Xline13 <= LCDX+3;
            Xline14 <= LCDX+4;
            Xline15 <= LCDX+4;
            Xline0 <= LCDX+4;
            Xline1 <= LCDX+3;
            Xline2 <= LCDX+3;
            Xline3 <= LCDX+1;
          end
   default:
          begin
            Xline11 <= LCDX+1;
            Xline12 <= LCDX+3;
            Xline13 <= LCDX+3;
            Xline14 <= LCDX+4;
            Xline15 <= LCDX+4;
            Xline0 <= LCDX+4;
            Xline1 <= LCDX+3;
            Xline2 <= LCDX+3;
            Xline3 <= LCDX+1;
          end
endcase
end

wire [15:0] Frame = LCDY[15:4] * 160;
wire [15:0] backwardFrame = Frame - (LCDY[3] ? 0 : 160);
wire [15:0] forwardFrame  = Frame + (LCDY[3] ? 160 : 0);


always@ ( * )
begin
   //By default  the adress is all ones, which will return 'clear' from mem
       PixelAddresses[10:0]    <= 11'b11111111111;
       PixelAddresses[21:11]   <= 11'b11111111111;
       PixelAddresses[32:22]   <= 11'b11111111111;
       PixelAddresses[43:33]   <= 11'b11111111111;
       PixelAddresses[54:44]   <= 11'b11111111111;
       PixelAddresses[65:55]   <= 11'b11111111111;
       PixelAddresses[76:66]   <= 11'b11111111111;
       PixelAddresses[87:77]   <= 11'b11111111111;
       PixelAddresses[98:88]   <= 11'b11111111111;
       PixelAddresses[109:99]  <= 11'b11111111111;
       PixelAddresses[120:110] <= 11'b11111111111;
       PixelAddresses[131:121] <= 11'b11111111111;
       PixelAddresses[142:132] <= 11'b11111111111;
       PixelAddresses[153:143] <= 11'b11111111111;
       PixelAddresses[164:154] <= 11'b11111111111;
       PixelAddresses[175:165] <= 11'b11111111111;
   if (forwardFrame < 120*160 && forwardFrame >= 0) //if the the next frame is not off the end
       begin
       if (Xline0 >= 0 && Xline0 < 160 && ~startNewLine)
              PixelAddresses[10:0]    <=forwardFrame + Xline0; //if line15 is the current
index, line0 must advance to the next frame
```

```
                if (Xline1 >= 0 && Xline1 < 160 && ~startNewLine)
                        PixelAddresses[21:11]   <= forwardFrame + Xline1;
                if (Xline2 >= 0 && Xline2 < 160 && ~startNewLine)
                        PixelAddresses[32:22]   <=forwardFrame + Xline2;
                if (Xline3 >= 0 && Xline3 < 160 && ~startNewLine)
                        PixelAddresses[43:33]   <=forwardFrame + Xline3;
        end
        if (Xline4 >= 0 && Xline4 < 160 && ~startNewLine)
                PixelAddresses[54:44]   <= Frame + Xline4;
        if (Xline5 >= 0 && Xline5 < 160 && ~startNewLine)
                PixelAddresses[65:55]   <= Frame + Xline5;
        if (Xline6 >= 0 && Xline6 < 160 && ~startNewLine)
                PixelAddresses[76:66]   <= Frame + Xline6;
        if (Xline7 >= 0 && Xline7 < 160 && ~startNewLine)
                PixelAddresses[87:77]   <= Frame + Xline7;
        if (Xline8 >= 0 && Xline8 < 160 && ~startNewLine)
                PixelAddresses[98:88]   <= Frame + Xline8;
        if (Xline9 >= 0 && Xline9 < 160 && ~startNewLine)
                PixelAddresses[109:99]  <= Frame + Xline9;
        if (Xline10 >= 0 && Xline10 < 160)
                PixelAddresses[120:110] <= Frame + Xline10;
        if (Xline11 >= 0 && Xline11 < 160 && ~startNewLine)
                PixelAddresses[131:121] <= Frame + Xline11;
        if (backwardFrame >= 0 && backwardFrame < 120*160) //if the previous frame is not 
negative
        begin
                if (Xline12 >= 0 && Xline12 < 160 && ~startNewLine)
                        PixelAddresses[142:132] <=backwardFrame + Xline12; //if line 0 is the 
current line, line 12 must be one frame behind it
                if (Xline13 >= 0 && Xline13 < 160 && ~startNewLine)
                        PixelAddresses[153:143] <=backwardFrame + Xline13;
                if (Xline14 >= 0 && Xline14 < 160 && ~startNewLine)
                        PixelAddresses[164:154] <=backwardFrame + Xline14;
                if (Xline15 >= 0 && Xline15 < 160 && ~startNewLine)
                        PixelAddresses[175:165] <=backwardFrame + Xline15;
        end
end


wire [7:0] array [15:0];
assign array[0]=FrameBufferDatain[7:0];
assign array[1]=FrameBufferDatain[15:8];
assign array[2]=FrameBufferDatain[23:16];
assign array[3]=FrameBufferDatain[31:24];
assign array[4]=FrameBufferDatain[39:32];
assign array[5]=FrameBufferDatain[47:40];
assign array[6]=FrameBufferDatain[55:48];
assign array[7]=FrameBufferDatain[63:56];
assign array[8]=FrameBufferDatain[71:64];
assign array[9]=FrameBufferDatain[79:72];
assign array[10]=FrameBufferDatain[87:80];
assign array[11]=FrameBufferDatain[95:88];
assign array[12]=FrameBufferDatain[103:96];
assign array[13]=FrameBufferDatain[111:104];
assign array[14]=FrameBufferDatain[119:112];
assign array[15]=FrameBufferDatain[127:120];

wire [3:0] lineCurrentIndex = LCDY[3:0];

wire [3:0] lineUp4_index=lineCurrentIndex - 4;
wire [3:0] lineUp3_index=lineCurrentIndex - 3;
wire [3:0] lineUp2_index=lineCurrentIndex - 2;
wire [3:0] lineUp1_index=lineCurrentIndex - 1;
wire [3:0] lineCurrent_index=lineCurrentIndex;
wire [3:0] lineDown1_index=lineCurrentIndex + 1;
wire [3:0] lineDown2_index=lineCurrentIndex + 2;
wire [3:0] lineDown3_index=lineCurrentIndex + 3;
wire [3:0] lineDown4_index=lineCurrentIndex + 4;

wire [7:0] lineUp4In = array[lineUp4_index];
wire [7:0] lineUp3In = array[lineUp3_index];
wire [7:0] lineUp2In = array[lineUp2_index];
wire [7:0] lineUp1In = array[lineUp1_index];
wire [7:0] lineCurrentIn = array[lineCurrent_index];
wire [7:0] lineDown1In = array[lineDown1_index];
wire [7:0] lineDown2In = array[lineDown2_index];
wire [7:0] lineDown3In = array[lineDown3_index];
wire [7:0] lineDown4In = array[lineDown4_index];

reg [7:0] lineUp4 [2:0];
reg [7:0] lineUp3 [6:0];
reg [7:0] lineUp2 [6:0];
reg [7:0] lineUp1 [8:0];
reg [7:0] lineCurrent [8:0];
reg [7:0] lineDown1 [8:0];
reg [7:0] lineDown2 [6:0];
reg [7:0] lineDown3 [6:0];
reg [7:0] lineDown4 [2:0];

integer i;

always @(posedge LCDClkIn)
begin
	if (reset | startNewLine)
		begin
		for (i=0;i<3;i=i+1)
			lineUp4[i]<=0;
		for (i=0;i<7;i=i+1)
			lineUp3[i]<=0;
		for (i=0;i<7;i=i+1)
			lineUp2[i]<=0;
		for (i=0;i<9;i=i+1)
			lineUp1[i]<=0;
		for (i=0;i<9;i=i+1)
			lineCurrent[i]<=0;
		for (i=0;i<9;i=i+1)
			lineDown1[i]<=0;
		for (i=0;i<7;i=i+1)
			lineDown2[i]<=0;
		for (i=0;i<7;i=i+1)
			lineDown3[i]<=0;
		for (i=0;i<3;i=i+1)
			lineDown4[i]<=0;
		end
	else
		begin
		for (i=1;i<3;i=i+1)
			lineUp4[i]<=lineUp4[i-1];
		for (i=1;i<7;i=i+1)
			lineUp3[i]<=lineUp3[i-1];
		for (i=1;i<7;i=i+1)
			lineUp2[i]<=lineUp2[i-1];
		for (i=1;i<9;i=i+1)
			lineUp1[i]<=lineUp1[i-1];
		for (i=1;i<9;i=i+1)
			lineCurrent[i]<=lineCurrent[i-1];
		for (i=1;i<9;i=i+1)
			lineDown1[i]<=lineDown1[i-1];
		for (i=1;i<7;i=i+1)
			lineDown2[i]<=lineDown2[i-1];
		for (i=1;i<7;i=i+1)
			lineDown3[i]<=lineDown3[i-1];
		for (i=1;i<3;i=i+1)
			lineDown4[i]<=lineDown4[i-1];

		lineUp4[0] <= lineUp4In;
		lineUp3[0] <= lineUp3In;
		lineUp2[0] <= lineUp2In;
		lineUp1[0] <= lineUp1In;
		lineCurrent[0] <= lineCurrentIn;
		lineDown1[0] <= lineDown1In;
		lineDown2[0] <= lineDown2In;
```

```
               lineDown3[0] <= lineDown3In;
               lineDown4[0] <= lineDown4In;
               end
end

reg [7:0] up4Color;
reg [7:0] up3Color;
reg [7:0] up2Color;
reg [7:0] up1Color;
reg [7:0] currentColor;
reg [7:0] down1Color;
reg [7:0] down2Color;
reg [7:0] down3Color;
reg [7:0] down4Color;

wire [487:0] colors;
genvar k;
generate
begin
       for (k=0;k<3;k=k+1) begin: part1
               assign colors[k*8 + 7: k*8] = lineUp4[k];
               end
       for (k=0;k<7;k=k+1) begin: part2
               assign colors[(k + 3)*8 + 7: (k + 3)*8] = lineUp3[k];
               end
       for (k=0;k<7;k=k+1) begin: part3
               assign colors[(k + 10)*8 + 7: (k + 10)*8] = lineUp2[k];
               end
       for (k=0;k<9;k=k+1) begin: part4
               assign colors[(k + 17)*8 + 7: (k + 17)*8] = lineUp1[k];
               end
       for (k=0;k<9;k=k+1) begin: part5
               assign colors[(k + 26)*8 + 7: (k + 26)*8] = lineCurrent[k];
               end
       for (k=0;k<9;k=k+1) begin: part6
               assign colors[(k + 35)*8 + 7: (k + 35)*8] = lineDown1[k];
               end
       for (k=0;k<7;k=k+1) begin: part7
               assign colors[(k + 44)*8 + 7: (k + 44)*8] = lineDown2[k];
               end
       for (k=0;k<7;k=k+1) begin: part8
               assign colors[(k + 51)*8 + 7: (k + 51)*8] = lineDown3[k];
               end
       for (k=0;k<3;k=k+1) begin: part9
               assign colors[(k + 58)*8 + 7: (k + 58)*8] = lineDown4[k];
               end
end
endgenerate

wire [7:0] tempColor;
FullColorChooser colorChoice (.colors(colors),.colorOut(tempColor));

reg [7:0] outColor;
always@ ( posedge LCDClkIn )
begin
if (reset)
       outColor <=0;
else
       outColor <=tempColor;
end
assign color = ~outColor;

endmodule
```