A Fpga-based Architecture For Led Backlight Driving

2010

Zhaoshi Zheng
University of Central Florida

A FPGA-BASED ARCHITECTURE FOR LED BACKLIGHT DRIVING

by

ZHAOSHI ZHENG

B.E. Dalian University of Technology, 2008

A thesis submitted in partial fulfillment of the requirements
for the degree of Master in Science of Computer Science
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at University of Central Florida
Orlando, Florida

Summer Term
2010

ABSTRACT

In recent years, Light-emitting Diodes (LEDs) have become a promising candidate for backlighting Liquid Crystal Displays [1] (LCDs). Compared with traditional Cold Cathode Fluorescent Lamps (CCFLs) technology, LEDs offer not only better visual quality, but also improved power efficiency. However, to fully utilized LEDs' capability requires dynamic independent control of individual LEDs, which remains as a challenging topic.

A FPGA-based hardware system for LED backlight control is proposed in this work. We successfully achieve dynamic adjustment of any individual LED's intensity in each of the three color channels (Red, Green and Blue), in response to a real time incoming video stream. In computing LED intensity, four video content processing algorithms have been implemented and tested, including averaging, histogram equalization, LED zone pattern change detection and non-linear mapping. We also construct two versions of the system. The first employs an embedded processor which performs the above-mentioned algorithms on pre-processed video data; the second embodies the same functionality as the first on fixed hardware logic for better performance and power efficiency. The system servers as the backbone of a consolidated display, which yields better visual quality than common commercial displays, we build in collaboration with a group of researchers from CREOL at UCF.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS/ABBREVIATIONS

CCFL	Cold Cathode Fluorescent Lamp
CDF	Cumulative Distribution Function
CREOL	The Center for Research and Education in Optics and Lasers
DCA	Delta Color Adjustment
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
HSMC	High Speed Mezzanine Card
HSYNC	Horizontal Synchronization
IDE	Integrated Development Environment
IP	Intellectual Property
KB	Kilobyte
LC	Liquid Crystal
LCD	Liquid Crystal Display
LED	Light Emitting Diode
IMF	The Inverse of a Mapping Function
NIOSII	Altera's Embedded Processor Soft Core

PDF	Probability Density Function
PLL	Phase Locked Loop
RTL	Register Transfer Level
SOPC	System On a Programmable Chip
SPI	Serial Peripheral Interface Bus
RAM	Random Access Memory
RGB	Red, Green, and Blue Color Channels
UCF	University of Central Florida
VLSI	Very Large Scale Integration
VSYNC	Vertical Synchronization
XGA	Extended Graphics Array

CHAPTER ONE: INTRODUCTION

In recent years, Light-emitting Diodes (LEDs) have become a promising candidate for backlighting Liquid Crystal Displays [1] (LCDs). Compared with traditional Cold Cathode Fluorescent Lamps (CCFLs) technology, LEDs offer not only better visual quality, but also improved power efficiency. However, to fully utilized LEDs' capability requires dynamic independent control of individual LEDs, which remains as a challenging topic.

Field Programmable Gate Arrays (FPGAs) have enjoyed developers' favor for a long time because of their flexibility. With advancement of VLSI technology in the past decade, FPGAs have become more powerful in both sense of performance and capacity. Such advantages make FPGAs ideal platforms for hardware prototyping and even small scale production. They also extend product life cycle by allowing upgrade be as simple and cost effective as re-programming a FPGA rather than replacing the entire circuit board.

We have collaborated with a group of researchers from CREOL at UCF in building a novel display that maximizes the optical benefits brought by using LEDs as backlight. Our contribution to the project includes design and implementation of a video content processing

system on FPGA platform which dynamically computes the brightness of each color channel of any single backlight LED.

Our display is consisted of three major components: a front LC panel, a backlight LED array panel with LED driving circuitry, and the video content processing system. The LC panel used is in compliance with the XGA standard [2][3], supporting 1024 by 768 resolution at 60Hz frame rate. Our backlight panel has 12 by 9 LEDs geometrically arranged in square grids. This paper focuses on the FPGA-based video content processing system we designed for computing LED brightness from an input video stream.

1.1 Development Platform

Our design employs Altera's FPGA solution. StratixIII [4] is the FPGA product of choice along with design software suite provided by Altera. Currently we are using a StratixIII FPGA development board for design space exploration and prototype verification; the design software version is 9.1.

QuartusII [5] is the center piece of all software involved in FPGA development of this project. It performs design source compilation, FPGA placement and routing, timing analysis, and FPGA programming file generation. Other software introduced here play supporting roles to QuartusII. SOPC (System On a Programmable Chip) Builder provides a user-friendly GUI to

rapidly prototype a system with both Altera's IP cores and custom logic. Video processing algorithms are developed in NIOSII (Altera's embedded processor) IDE [6][7]. SignalTapII is a software logic analyzer released by Altera. It injects hardware logic additional to the design for recording and extracting run time signals in the FPGA chip. Signals retrieved by the analyzer can be transmitted to a host PC connected to the development board via a FPGA download cable. The design also employs several Altera IP cores, among which are: embedded microprocessor NIOSII, on-chip memory module, SPI communication, count-down timer working as watchdog, and PLL generating SOPC system clock.

The rest of this paper is organized as follow: chapter 2 specifies input, output, and performance requirement of our video content processing system. The next chapter describes implemented brightness computation algorithms with their benefits. Chapter 4 and 5 present detailed design and test of our implementation on FPGA. Finally, chapter 6 summarizes our contribution and future research direction.

CHAPTER TWO: SYSTEM REQUIREMENT ANALYSIS

In this chapter we specify the video input and brightness output requirements of our video content processing system. We also analyze the performance demand of our system under the specified input and output conditions. Moreover, two techniques are introduced for better video processing performance: pixel sub-sampling and dual-ported buffer.

2.1 Video Input Specification

Our video content processing system assumes a digitized video stream at the resolution of 1024 by 768 is coming into the FPGA at 60 frames per second. Pixel data are in the format of parallel RGB, with 10 bits per color channel at most. Also included in the video input are separated vertical and horizontal synchronization (VSYNC and HSYNC) signals, indicating new frame and new row respectively. Finally, all above signals are synchronous with a pixel clock at the frequency of 65MHz, as specified by the XGA standard [2][3].

In the system, one frame is divided into zones according to the backlight LEDs. To void ambiguity, we uses following terms to describe subsets of the 1024 by 768 pixels in one frame. One sub-frame (or one strip) is several consecutive rows of pixels within a frame. The number of pixel rows is determined by the number of rows of backlight LEDs. For example, if there are eight rows of LEDs, then a sub-frame is consisted of 96 rows, as 768 divided by 8. One region

(or one LED zone) is several consecutive columns of a sub-frame. The number of columns of a region is determined by the number of individual LEDs in one row of the backlight LED array. For example, if there are 16 LEDs per row, the number of columns of a region is 64, as 1024 divided by 16.

Taking the two examples together, one region has 64 by 96 pixels; and all these pixels contribute to the brightness information of one LED zone. Figure1 explains the terms described above.

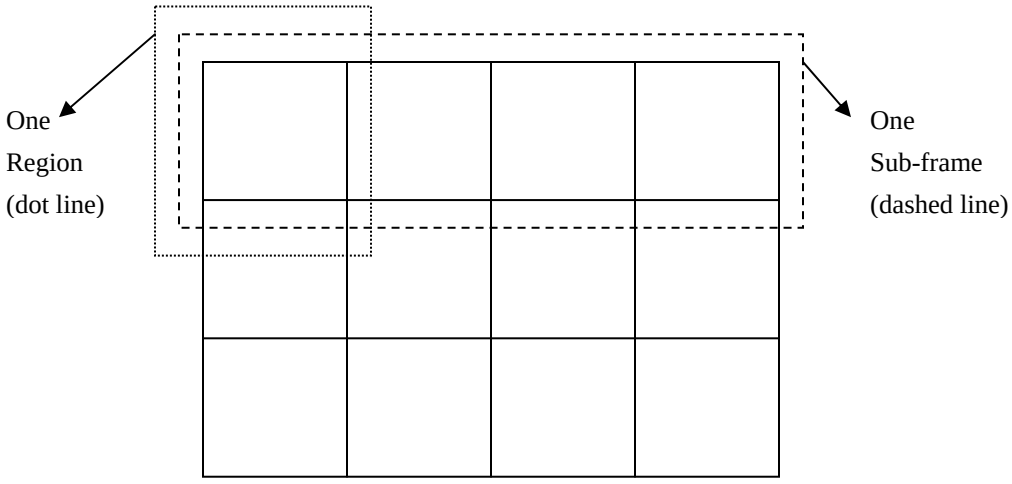


Figure 1: Terms Used in Description of One Frame's Subsets.

According to the XGA standard, the number of pixels transmitted for a frame is actually greater than that determined by the screen resolution. For example, if the display resolution is 1024 by 768, the actual number of transmitted pixels is 1344 by 806 per frame, resulting in a 65MHz

pixel clock at the frame rate of 60Hz. These extra pixels are all blank and discarded by the display's control logic; thus they do not affect the displayed image of one frame. The 1024 by 768 displayed pixels form an effective window with those blank pixels surrounding them. Figure2 illustrates the concept of effective window.

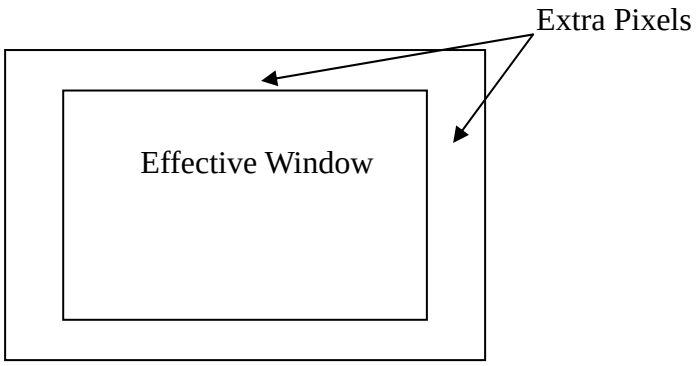


Figure 2: Effective Window and Extra Pixels.

To better understand the effective window of XGA [2][3] in detail, take the pixels in one scan line for example. The extra pixels of a scan line form three portions of the gap between the last effective pixel of the previous row and the first effective pixel of the next row: front porch, sync pulse and back porch. As visualized in Figure3, front porch follows immediately after the effective window; then sync pulse holds for a while; and finally back porch precedes the effective window of the next row. Similarly, front porch, sync pulse and back porch also exist on the vertical direction of a frame, despite that they are formed by scan lines rather than pixels. Table1 details the exact values for the timing portions just explained. All the extra pixels have

to be filtered to ensure that only effective pixels are passed through to the video content processing logic. Implementation of such XGA filter is explained in chapter 4.

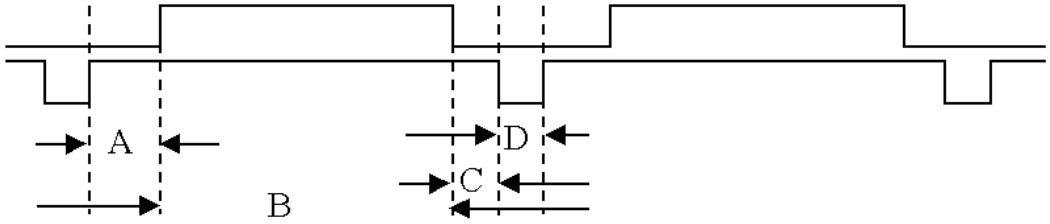


Figure 3: XGA Timing Diagram. (A) Back Porch; (B) Effective Window; (C) Front Porch; (D) Sync Pulse.

Table 1: XGA Timing Chart.

Video Format	Pixel Clock (MHz)	Horizontal (in pixels)				Vertical (in lines)			
		Effective Window	Front Porch	Sync Pulse	Back Porch	Effective Window	Front Porch	Sync Pulse	Back Porch
1024x768, 60Hz	65.000	1024	24	136	160	768	3	6	29

2.2 Performance Analysis

In the NIOSII-based system, input pixels are first stored in internal buffers. The processor reads from these buffers upon the completion of each sub-frame and calculates LED brightness accordingly. That is, the output values for one row of LEDs are computed when all their corresponding regions are buffered in their entirety. Therefore, the system must finish processing one sub-frame and outputting LED brightness within the time period it takes to input the next sub-frame. The actual allowed processing time can be calculated by the following equation:

$$T_s = n_c^t \times \left\lfloor n_r^e \div l_r \right\rfloor \times \frac{1}{f_p} \quad (1)$$

Here T_s represents the time allowed to process one strip. n_c^t and n_r^e denote total number of pixel columns and effective number of pixel rows respectively. l_r indicates the row number of backlight LEDs. Moreover, f_p is the pixel clock of input video stream. Since we are using the XGA video source for a 12 by 9 LED backlight array, T_s is:

$$1344 \times \lfloor 768 \div 9 \rfloor \times \frac{1}{65 \times 10^6} \approx 1.7ms$$

To have a rough estimation, assume the NIOSII needs 10 processor clock cycles on average, including all overhead, to process one pixel. Hence, for each sub-frame, the total processing time for NIOSII is:

$$1024 \times \left\lceil \frac{768}{9} \right\rceil \times 10 = 880640$$

cycles; and the required processor clock frequency is:

$$\frac{880640}{1.7 \times 10^{-3}} \approx 520MHz$$

However, clock speed test of NIOSII in our system results in a maximum frequency slightly over 200MHz. To avoid any delay on outputting LED brightness, the total number of pixels to be processed within each frame must be reduced to allow the system responding timely. Such reduction is achieved by two dimensional subsampling. Horizontally, the system “merges” four adjacent pixels by calculating their average. Vertically, only the middle row of each three

consecutive rows is passed through, from the second row, the fifth, and the eighth… to the 767th row of a frame. Therefore, the achieved pixel reduction factor is 12 (4 horizontally multiplied by 3 vertically). Detailed implementation of subsampling is described in chapter 4. Although such reduction factor appears to be aggressive, software simulation and actual hardware test both verify that our subsampling scheme does not introduce noticeable artifact in the final LED brightness output.

In the fixed logic implementation of our video content processing system, we employ a "come-and-serve" strategy: each pixel is processed at the time it arrives. As a result, brightness information of a region is available very shortly after its last pixel has been received. Any additional computation required by the algorithm in use is performed at the regional brightness level, which does not affect operations on input pixel stream, since these two stages of computation is separated by an internal dual-port buffer. We also retain our pixel reduction stage in this implementation for better power efficiency and heat dissipation.

2.3 Brightness Output Requirements

We have implemented two schemes for sending the final brightness values to LED driving circuitry. The first utilizes SPI [8] interface at a clock rate of 1MHz as required by the receiving end. In this method, LED brightness value and sync flags are all sent in the unit of byte. The NIOSII controls a SPI peripheral functioning as the transmitting master attached to its address

bus. All communication is single-directed: from the master to the slave. Also there is no buffer present at the receiving end, requiring an intermittent manner of communication. We managed to insert an adjustable interval between sending two bytes by software running on NIOSII. After a tedious process of trial and error, we have finalized the protocol in sending brightness values. The first byte is fixed at 0x55, signaling the start of transmission. The next byte indicates the row address of the LEDs being updated. Subsequently we send 12 brightness bytes of Red channel corresponding to the 12 LEDs in the row. After Red channel, 0x54 denotes the start of Blue channel, which is followed 0x53 as the head of Green channel. We prohibit values used for sync flags and row address (0x01 ~ 0x09) being sent as brightness value; 0x00 and 0xff are also disabled to avoid false activation of the receiving end. Figure4 specifies the format of our SPI packet.

0	1	2 13	14	15 26	27	28 39
0x55	Row Address	12 Bytes of Red	0x54	12 Bytes of Blue	0x53	12 Bytes of Green

Figure 4: Layout of the SPI Packet in LED Brightness Communication. The first row is byte addresses and the second row represents corresponding fields in the packet.

However, SPI communication is vulnerable to external noises since the signals are transmitted over three wires at a relatively high clock speed. It also consumes an expensive portion of NIOSII's CPU time, excluding complex but promising video processing algorithms. To avoid the inherited drawbacks of the SPI scheme, our second method employs a dual-port buffer with dual clock support. The video processing system stores LED brightness value into this buffer

while the LED driving circuitry reads from it. This buffer is configured to always supply the most up-to-date content on its read port. Therefore, the NIOSII system is free from synchronization with LED driving circuitry and can focus its full effort on video content processing. This dual-port buffer scheme is also used in the fixed logic implementation.

CHAPTER THREE: VIDEO PROCESSING ALGORITHMS

This chapter elaborates the video content processing algorithms used in computation of final LED brightness output. All method described here have been implemented and tested in software of NIOSII. We will explore more algorithms as well as their hardware implementation in the future.

The most straightforward and fundamental algorithm is to averaging all pixel values within a LED zone. For each region, we obtain this brightness information on each color channel by processing RGB separately. Despite of its simplicity, the averaging stage takes significant amount of time in the NIOSII-based system. This is due to that the 10-bit color data of each channel are packed into a 32-bit word. After reading this word from buffer, the processor has to execute a sequence of logic AND, shift, and addition instructions for each color. Although these serialized operations can fit into the processing time determined by equation (1) in section 2.2, post computation as needed by other algorithms is penalized by the inadequate time left before the processor must move to next sub-frame. We extended the NOIS2 with a vector addition instruction to boost performance of the averaging computation. With our custom instruction enabled, processor time consumed by averaging computation is reduced by a factor over two,

allowing incorporation of more complex and visually appealing algorithms. Implementation details of the custom instruction module can be found in section 4.2.

Averaging also serves as the first stage of other video processing algorithms. Averaging results of each sub-frame are stored in a two-dimensional array, forming a LED image. Further operations (if needed) like computing the histogram of a frame are performed at the LED image level. Such operations are applied to the LED image every time regional averages of one row are updated. As a result, for any row's LED brightness output being generated, regional averages of rows above and including the row of interest are calculated from the current frame while that of rows below it are obtained from the previous frame. We adopt this "partially-current-frame" strategy for the concern of delay time between refreshing pixel values on the liquid crystal front panel and adjusting backlight LEDs brightness accordingly. If the system waits until regional averages of the entire frame are computed, not only the delay time of different LED rows is not uniform, also a disparity between LC front panel and backlight LEDs is created: while pixels of the first sub-frame is being updated on the front panel, brightness value driving corresponding backlight LEDs are derive from the last frame. Our strategy successfully avoids this problem by always using the latest partial LED image.

3.1 Histogram Equalization

Histogram Equalization [9][10] is a commonly used technique for contrast ratio enhancement in digital image processing. This method effectively "stretches" histogram of the original image to the full grayscale range of target display system. Thus pixels in a dark region are adjusted to lowers intensity values while those in a bright region are assigned higher intensity values. In NIOSII software implementation, upon updating each row's regional brightness, we first collect the probability density function (PDF) of the LED image:

$$p(i) = \frac{n_i}{n} \tag{2}$$

where *p(i)* represents the probability density of the i^{th} grayscale level, n_i is the number of LEDs at this level and n is the total number of backlight LEDs. We then convert the derived PDF into cumulative distribution function (CDF) by applying the following summation at each grayscale level:

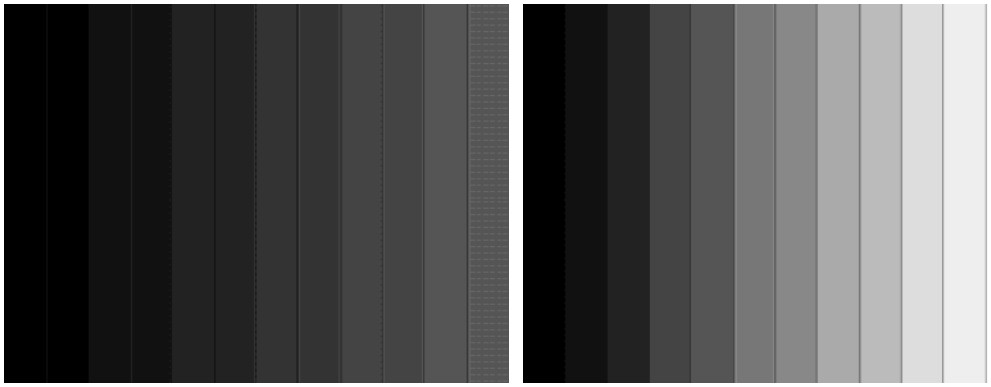
$$cdf(i) = \sum_{j=0}^{i} p(j) \tag{3}$$

Finally, brightness values are computed by normalizing CDF to the full grayscale range used in LED driving, in our system, [0, 255]:

$$B(i) = \left\lfloor \frac{cdf(i) - cdf_{\min}}{n - cdf_{\min}} \times 255 \right\rfloor \tag{4}$$

where cdf_{min} is the minimum non-zero value of the CDF. Note that this step only applies to those grayscale levels specified by regional averages of the row of interest, since the LED image will be modified when computing brightness output for other rows.

To observe the effect of histogram equalization, take the test image in Figure5 (a) as an example: this image exhibits a gradient pattern at low brightness range as shown by its probability density function in Figure6. After histogram equalization, pixels of the test image are spread over the full grayscale range in Figure5 (b), compared to previously concentrated at the lower portion. Figure7 highlights the difference between this method and pixel averaging by plotting their mapping functions. Clearly, histogram equalization utilizes the full scope of allowed brightness where averaging restricts itself at the scope defined by the input image.



(a) (b)

Figure 5: Test Image (a) and Result Image (b) of Histogram Equalization.

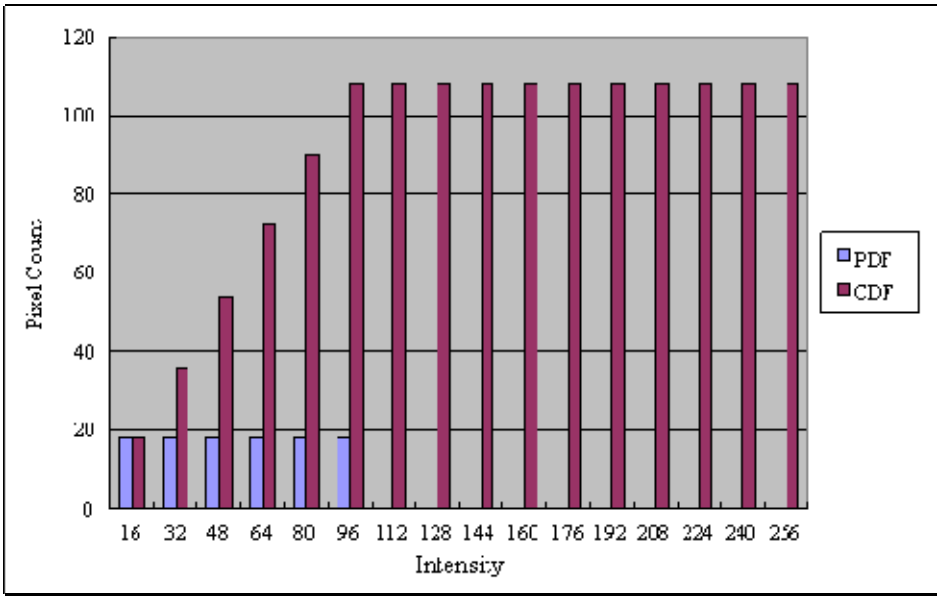


Figure 6: Probability Density Function and Cumulative Distribution Function of the Test Image in Figure5 (a).

In case it needs to shrink program size and CPU processing time, we employ a reduced brightness scale in software implementation. In this scheme, only the most significant bits of regional brightness are used for indexing the histogram and CDF. Experiment results show that using 64 grayscale levels produces the best trade-off between software complexity and visual quality.

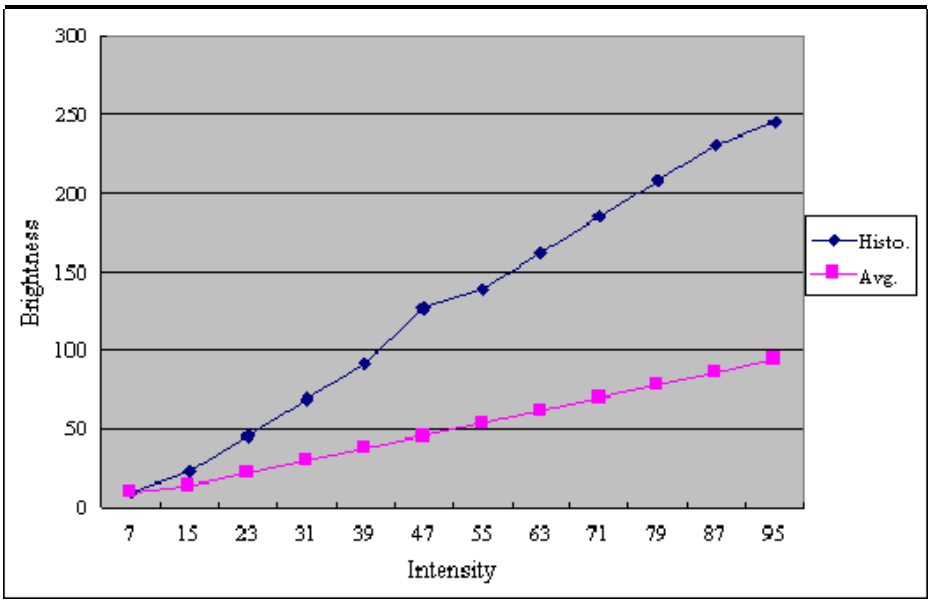


Figure 7: Mapping Functions of Histogram Equalization and Pixel Averaging Method.

3.2 Non-Linear Mapping

We noticed that for some display applications, like thermal or medical imagery, the object of interest is usually rendered by particularly darker or brighter intensity against its background. In such cases, uniformly enhancing each brightness level may not yield the best visual quality. Rather, suppose the display is rendering a dark object, if the bright background can be enhanced while the object remains at its low intensity range, it will appear to be "darker" against the "brighter" background, making it easier for human eyes to pick up.

Based on above observation, we propose the Non-linear Mapping method for video content processing. By this algorithm, regional averages are mapped to LED brightness output

according to one of several (nine in our implementation) predefined mapping curves. Choice of mapping function is made by global intensity of the image. Ideally, mapping curves should exhibit a "∫" shape and shift along the axis of regional averages. We approximate this idealism by using the following mapping function developed from experiment:

$$B(i) = f^{-1}(i) \tag{5}$$

$$f(i) = w\left(\frac{1+\sin[(-\frac{1}{2}+(\frac{i}{256})^{-4})\pi]}{2}\times 255\right) + (1-w)\left(\frac{1+\sin[(-\frac{1}{2}+(\frac{i}{256})^{4})\pi]}{2}\times 255\right) \tag{6}$$

where i represents the grayscale level of regional averages and w is the weighting parameter, ranging from 0.1 to 0.9 by 0.1 at each step. However, it is mathematically impossible to derive an analytical form of $f^{-1}(i)$. This problem is addressed by first computing values of $f(i)$, then numerically inversing the result and storing $f^{-1}(i)$ in look-up tables. Figure8 illustrates the nine mapping functions used in our software implementation.

Each time the LED image is updated, we first calculate the global average of regional brightness and then divide this number by 26. The resulted quotient is used to index which mapping function will be applied. For example, if the quotient is zero, LED brightness outputs are generated by the mapping curve with w = 0.9, keeping the bright object while dimming the dark background.

From Figure8 we can observe another benefit brought by the non-linear mapping method. It offers relatively high contrast ratio within the brightness range of featured objects. As the mapping curves shift from left to right, high CR range also moves from low brightness levels to high brightness levels, maintaining details of highlighted objects.

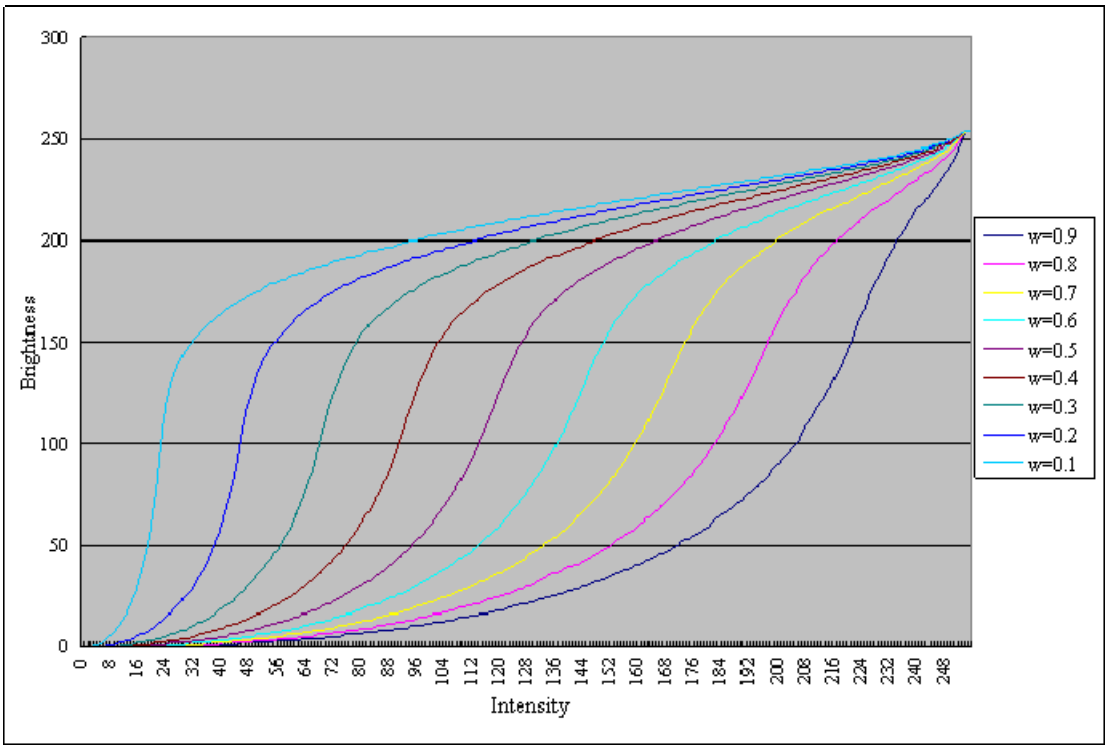


Figure 8: Mapping Functions in Non-linear Mapping Algorithm.

3.3 Regional Pattern Change Detection

The above three algorithms perform fairly well in detecting significant changes of a LED zone between two frames. Nonetheless, if the image content within one region altered but average of pixels remains the same, brightness output will not be able to reflect such pattern change, which

is very common for industrial scenarios. To better our system's capability, we introduce the fourth video content processing algorithm: Regional Pattern Change Detection.

The basic idea of this method is to derive a characteristic value for each LED zone, which mirrors both intensity and spatial distribution of all pixels within the region. Thus, even if the average brightness stays the same, difference of the characteristic values between two frames indicates image content of the region has been changed. In our implementation, if the absolute value of such difference is greater than a predefined threshold, LED brightness output is automatically increased by 32.

To approach this goal, we use the Mass Center of a region as its characteristic value. The mass center is defined as a *(x, y)* pair where x is the weighted average of pixels' local address within on the horizontal direction and y is that on the vertical dimension. The weight value associated with each pixel is defined as the maximum of its RGB channels. Finding the weight can also be done by the custom instruction of NIOSII; such compare-and-select logic is incorporated into our vector addition unit. For each pixel, its local row address and column addresses are first multiplied by the weight value and the products are then separately accumulated. After processing of the last pixel, accumulated results are divided by the summation of all pixels' weight value, forming the *(x, y)* pair. The algorithm then compares *(x, y)* against the region's

mass center in the previous frame to determine if the difference in total exceeds the predefined threshold.

CHAPTER FOUR: DESIGN AND IMPLEMENTATION

This chapter presents the detailed design and implementation of our video content processing system on FPGA.

Our system is divided into two major stages: video preprocessing and the brightness generation. The preprocessing unit first implements the XGA filter ensuring that only valid pixels are streamed through, then it performs pixel number reduction as described in Chapter2; pixels' row and column addresses are also generated in this stage along with control signals including Data Enable, sub-frame Sync, and memory Chip Selection. The two versions of our system diverge at the brightness generation stage. In the NIOSII-based implementation, we build a SOPC system using Altera's IPs along with our custom video controller and vector addition unit. The NIOSII processor fetches pixel data from on-chip memory buffers, computes regional brightness on each of the three color channels, and sends the results via SPI protocol or stores them in a result buffer, depending on which communication method is used. In the fixed logic implementation, we construct a pipelined structure consisting of a pixel dispatcher and 12 region accumulators. Brightness output in this implementation is stored in a result buffer for LED driving circuitry to read. Figure9 is a brief block diagram of the hardware system. Section

4.1 describes details of the video preprocessing stage. The next section elaborates the NIOSII-based system. Finally, the fixed logic implementation is explained in Section 4.3.

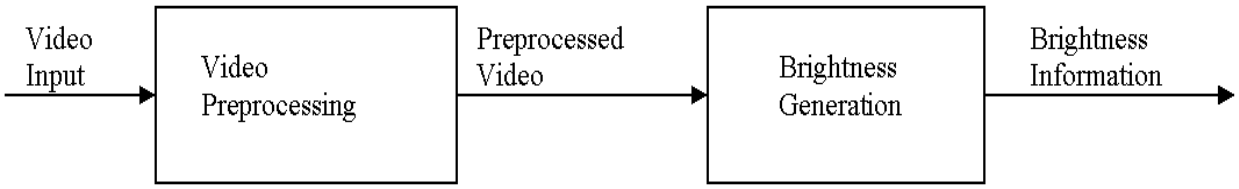


Figure 9: An Overview of the Video Content Processing System

4.1 Video Preprocessing Stage

The video preprocessing stage is consisted of three components: a XGA filter specifically designed for the XGA standard, a pixel merger and an address generation unit; the last two together implements the pixel reduction functionality.

4.1.1 XGA Filter

The primary design objective of our XGA filter is to allow only effective pixels being sent to the reduction logic. Pixel filtering within one row is achieved through triggering an incremental counter by the pixel clock; a pixel valid signal is asserted only if the value of this counter is within the horizontal effective range. Further, horizontal sync pulse resets this counter at the start of each scan line. The upper and lower bound of effective range is parameterized in case these numbers change for future adoption of other video standard. Line filtering within a frame is implemented in a similar manner, but using horizontal sync as triggering signal and vertical

sync as reset signal of the vertical counter. The pixel valid signal and line valid signal are ANDed together, producing the *vga_data_valid* output signal.

Another responsibility of the XGA filter is re-formatting HSYNC and VSYNC signals so that they are delayed to the first effective pixel of one row or one frame respectively and hold for only one clock cycle. This is done by simply compare values in the two counters against back porch numbers listed in Table1. For HSYNC, we also check if the vertical counter is in the effective range of scan lines.

4.1.2 Pixel Merger

Our Pixel Merger performs pixel number reduction on the horizontal dimension. It calculates the average of four adjacent pixels on each of the three color channels through a four-state machine. Gray Encoding [11] is employed in state transition to avoid erroneous intermediate state. Also in this module, pixel clock is divided by a factor of 4, producing the "reduced" clock. Moreover, there is a 2-bit saturating reset counter which increments at vertical synchronization. Only when the reset counter reaches a predefined value, the “reduced” clock output is enabled; otherwise, it remains in logical low and thus can not drive any down stream logic in the system. This waste-of-frame (2 to 3) feature allows a time period long enough for SOPC system initialization.

4.1.3 Address Generation Unit

The Address Generation Unit is designed to achieve two functionalities: generating both row and column addresses of a pixel along with control signals, and pixel number reduction on the vertical dimension. The control signals include row Data Enable (DE), sub-frame Sync (SYNC), and memory Chip Selection (CS) for the NIOSII-based system. DE is only asserted for the rows selected by vertical pixel reduction. Upon completing the last row of each sub-frame, SYNC is asserted to signal such event. CS is used to select one of the two video data buffers in the NIOSII-based system, implementing a double-buffering mechanism.

This module maintains two 10-bit counters for address signals and three single bit registers for control signals. The column address counter is incremented at positive edge of pixel clock and is reset at horizontal synchronization. The row address counter is incremented at effective horizontal sync and is reset at vertical sync. DE is asserted when the remainder of dividing row address by 3 is 1. Thus only one row in three consecutive rows is sent to brightness generation logic. CS is initialized to zero and flips at the end of each sub-frame. Meanwhile SYNC is hold for several hundred cycles during the last row (which is not chosen by vertical reduction) of each sub-frame. Finally, we have configured this module to output either original address or reduced address, depending on whichever is needed by its down stream logic. Figure10 illustrates the interconnection among the three modules of video preprocessing stage.

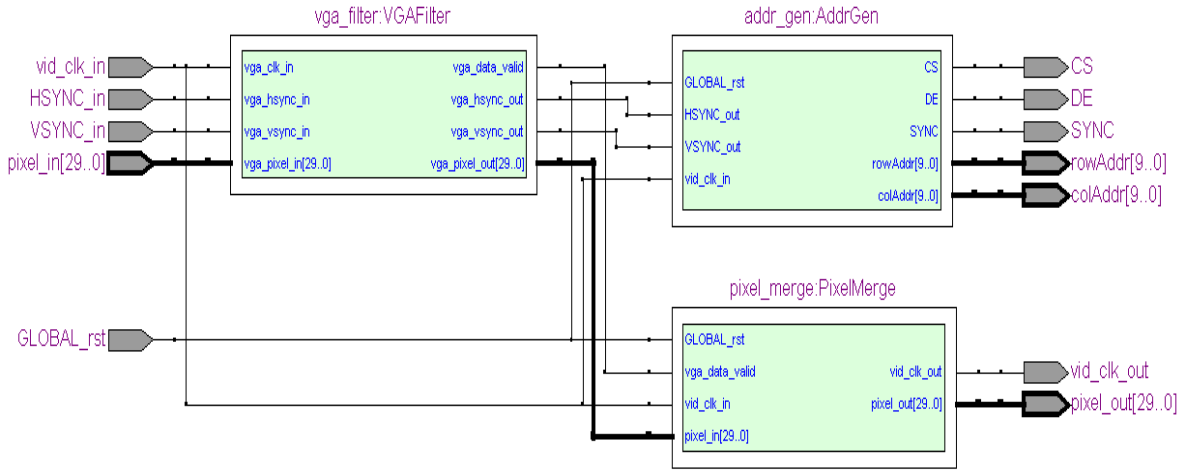


Figure 10: RTL View of Video Preprocessing Stage in QuartusII

With our pixel reduction scheme, the image resolution presented to brightness generation logic is 256 by 256. However, our backlight unit has a 12 by 9 LED array. Thus, pixel rows and columns can not be evenly distributed among all LEDs. To be specific,

$$256 \div 12 = 21 \cdots 4 \text{, and}$$

$$256 \div 9 = 28 \cdots 4 \text{,}$$

That is, four rows and four columns are to some extent "redundant".

In order to resolve this issue, overlapping LED zones is appealing on the sense of visual quality but it imposes significant complexity in video processing algorithm implementation. Dropping the last 4 rows and columns is simple to implement but results in misalignment between front LC panel and backlight LED panel. To address this problem, we propose a "Center Jump" technique which jumps over 4 rows and columns that are scattered across the frame as evenly as

possible. Specifically, rows and columns with reduced addresses 31, 95, 159, and 223 are victimized in brightness calculation of any LED. As a result, image processed by brightness generation stage is resized to 252 by 252.

We have modified the address generation unit to implement the Center Jump technique. First, DE is deasserted at the victim addresses. Moreover, we added horizontal and vertical jump counters which increment at the victim addresses and reset at the end of each row or frame. Finally, output row and column addresses are original reduced addresses subtracted by two jump counters' value respectively.

4.2 The SOPC System

Our first implementation of the brightness generation stage is a SOPC system built by using several Altera's IPs and two custom logic modules: the video controller and the vector addition unit. These IPs include a NIOSII embedded processor; four on-chip RAM modules; a PLL; a count-down timer; a SPI transmitter; and a system ID register. The NIOSII processor is extended with our vector addiction unit through its custom instruction capability [12]. In this section, we first cover parameter settings of adopted Altera IPs. Sub-section 4.2.1 and 4.2.2 present details of our custom logic.

The NIOSII processor is configured to the “fast” version with hardware multiplication and division options enabled to accelerate certain software operations. It is also equipped with separated instruction cache (4KB) and data cache (2KB). The JTAG debug module associated with the CPU is level 3 to support SignalTapII logic analyzer.

The first three on-chip memory modules are all of 32 KB. Two of them (*data0* and *data1*) implements double buffering of incoming pixel data. Suppose the first sub-frame is being written into *data0*. When *data0* is filled, the CPU reads pixel data out from it and process the first sub-frame; meanwhile the second sub-frame is being written into *data1*. When *data1* is filled, the CPU is done with the first sub-frame and switches to *data1*; *data0* is ready for being written again. In the current design, one sub-frame written into on-chip memory contains 7056 (one ninth of 252 squared) pixels and each pixels occupies 4 bytes of memory. If there are more rows of LEDs, the requirement of on-chip memory size will be reduced from 32KB. The address ranges of the two buffers are 0x00000 to 0x07fff and 0x20000 to 0x27fff respectively; 18-bit address width is chosen to allow expanding these buffers in case input video resolution increases. The FPGA chip currently in use can support 64KB per memory module. Another piece of on-chip memory (*code*) is used to hold executable of NIOSII's software. Program size varies from roughly 3KB to 22KB depending on which algorithms is used. The last on-chip memory is configured in dual-port mode and servers as the result buffer, holding computed

brightness output for LED driving circuitry to read. Currently we allocate 1KB to this buffer, more than enough to hold brightness output of the 9 LED rows (recall that the size of our SPI packet is 40 bytes). Also, the result buffer is not needed if SPI communication is used.

The FPGA development board currently in use has a 50MHz oscillator. The PLL in the SOPC system pulls this clock up to the system clock frequency, 200MHz. We use simplest configuration of this IP so that it does not require any run time control.

The count-down timer in the system works as a watch dog: when the timer reaches zero, a reset signal is sent to the whole SOPC system. By writing the timer’s control register, CPU can set it to a predefined value. The expiration period is set to 2ms, slightly longer than the time it takes to receive two sub-frames by the system.

If in use, the SPI core is configured to master mode with 1MHz transmitting clock. It is also configured to control only one SPI slave using eight bits per datum. Clock polarity and clock phase bits of SPI are all set to zero, in which combination data are effective at the clock's rising edge and changed at the falling edge.

The system ID register has no affect on functionality of the system. It is included according to Altera's recommendation. The value of this register is a random number defined at SOPC system generation time and can not be modified at run time. The NIOSII IDE includes this value into compiled program. When software binary code is downloaded to or compiled together with the hardware system, this value is compared against the system ID register to ensure that the software is compiled for the current version of hardware.

4.2.1 Vector Addition Unit

We add a vector addition instruction to the NIOSII processor for accelerating software video content processing. At the moment, all implemented algorithms require to calculate the sum of a number of pixels and then their average. Since the three color channels are packed in a 32-bit word, to processing one pixel the processor needs to first separate color data and then perform addition on each color. This results in a sequence of more than ten instructions and they are serially executed, taking a large bulk of processor time. Our vector adder, on the other hand, performs color separation and addition in parallel by hardware. It only needs the pixel word and function code from the NIOSII, reducing the processing of a pixel to one single instruction. Figure11 illustrates the design of our vector addition unit.

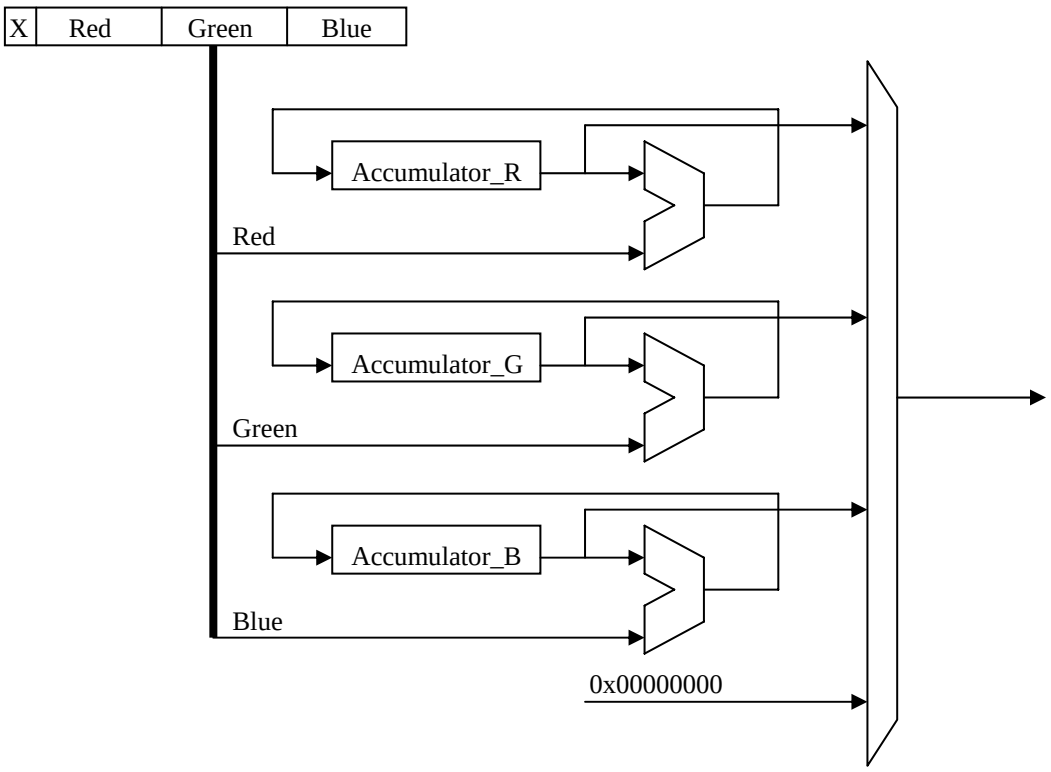


Figure 11: The Vector Addition Unit.

At each clock cycle, operation performed by the vector addition unit is controlled by function code issued from NIOSII along with pixel data. Currently, we only need 5 different operations hence the width of function code is 3. For each pixel, the unit accumulates RGB channel separately and stores the results to its internal registers. After the last pixel of a region is sent, NIOSII reads out summation values of the three channels sequentially using different function codes. If regional pattern change detection algorithm is used, our vector additions unit outputs the maximum of RGB channels during accumulation instructions. Division involved in computing average and multiplication of pattern change detection is performed by NIOSII

rather than the vector addition unit. Given the functionality requirement of our vector addition unit, it is implemented in compliance with fixed multi-cycle and extended types specified by [12]. Table2 summarizes function code values and their corresponding outputs.

Table 2: Operations of the Vector Addition Unit

Function Code	Operation	Output
000	Clear Internal Registers	0x00000000
001	Accumulation	0x00000000 or Max of RGB
010	Red Output	Sum of Red Channel
011	Green Output	Sum of Green Channel
100	Blue Output	Sum of Blue Channel
101	Unused	0x00000000
110	Unused	0x00000000
111	Unused	0x00000000

4.2.2 Video Controller

Our custom video controller serves as the interface between the video preprocessing stage and the SOPC system. Pixel data are written into on-chip data buffers (*data0* and *data1*) in a double-buffering fashion by the video controller. It also maintains a *status* register for the software to determine which on-chip data buffer is ready to be read and which sub-frame does the ready buffer hold. Moreover, the video controller sends an interrupt signal to the CPU upon the end of one sub-frame, requesting brightness computation for the latest strip.

Upon receiving each effective pixel, the video controller converts its row and column address to on-chip memory address in NIOSII's linear address space. At the same time, it asserts chip

selection and write enable signals of *data0* and *data1*. Choice of which buffer being written is made by using the *CS* input as the most significant bit of writing address. However, the input pixel clock (tens of MHz) is much slower than the 200MHz SOPC system clock. Also since they are generated from two different sources, alignment of their clock phase may vary from time to time. To avoid timing issue in SOPC system interconnection fabrics, outputs of the video controller must be registered by the system clock, despite that memory write operation is triggered by the input video clock. Therefore, proper functioning of the video controller requires crossing the two clock domains. This is achieved by detecting the rising edges of the pixel clock by two serially connected 1-bit registers (*cap0* and *cap1*) driven by the SOPC clock. If cap0 holds one while cap1 is zero, a *vid_posedge* signal is asserted for only one SOPC cycle to signify a rising edge of the pixel clock. The *vid_posedge* is further ANDed with input DE. A new pixel in a row of interest is received by the video controller from the video preprocessing unit when the AND gate outputs logical true.

The video controller maintains a 15-bit counter which increments by 1 upon receiving of an effective pixel. This counter is reset when its value equals the number of pixels in a subsampled sub-frame. The value of this counter is left shifted by 2 and concatenated after the chip selection signal, forming an 18-bit byte address for on-chip buffers. The write enable signal is also asserted at logical true of the above-mentioned AND gate. There is also a 2-bit register

flipping both bits upon a switch of CS. This register is initialized to 01 and indicates which on-chip buffer the NIOSII should read during operation. For example, 01 means *data0* is ready. Moreover, input row address is divided by the number of rows within a sub-frame, forming a 4-bit row address of the backlight LEDs. This 4-bit LED row address is concatenated before the 2-bit memory indicator register, producing a 6-bit *status* output for NIOSII's inquiry. Finally, an interrupt signal is asserted at the end of each sub-frame and de-asserted on either write to the video controller's control register by the CPU or 32 pixels of the new sub-frame are received. The self-de-asserting capability is adopted to ensure that the CPU can only respond to an interrupt request timely enough to allow sufficient time for video processing and possible sending the results via SPI. Our video controller is implemented in compliance with Altera's Avalon Memory-Mapped Interface [13]. Figure12 is a detailed block diagram of our SOPC system using SPI communication.

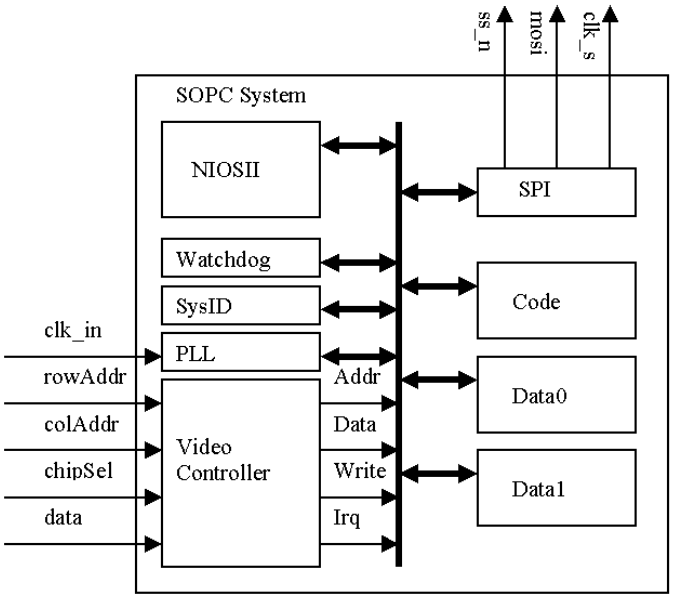


Figure 12: Detailed Block diagram of The SOPC System Using SPI Communication

4.2.3 External Read Bridge

As mentioned in section 2.3, to avoid potential issues associated with the SPI method, we also implemented a result buffer scheme for LED brightness communication. Nevertheless, memory modules of the SOPC system are not exposed to external circuitry for read operations. We implement a bridge serving as the agent of external circuitry in the SOPC system. This External Read Bridge is controlled by a 3-state machine. At "ready" state, it registers read command and address posted from the outside and transits to "read" state; it then issues read request to the result buffer and remains at "read" state until requested data is returned. During the "output" state, data are posted to external circuitry with a data valid signal asserted. Upon receiving its

data, external circuitry should return an acknowledge signal, setting the bridge back to "ready" state. Figure13 illustrates our SOPC system using the result buffer method.

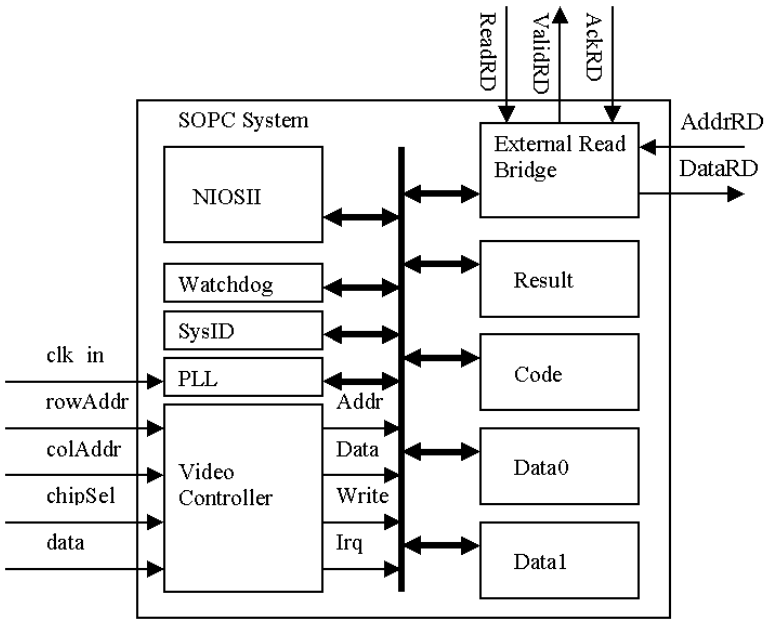


Figure 13: Detailed Block diagram of The SOPC System Using Result Buffer

4.2.4 Program Structure

The program running on NIOSII follows a typical work-on-interrupt manner. At initialization, the main program first registers the interrupt handler and initializes the watchdog. It then enters a dead *while* loop, which does nothing but feeding the watchdog. During this loop, interrupt requests happen and the program in-execution is shifted to the interrupt handler.

The interrupt handler first disables the same type of request, preventing embedded interrupt of “buffer ready”, which can easily cause synchronization issues. It then clears the interrupt bit in the control register of the video controller, reads its *status* output so the program can determine which on-chip buffer is ready and which sub-frame is held by that buffer. After that, the program reads pixels data from the ready buffer and calculates brightness for each of RGB color channels of every region within that sub-frame. The result is then filled into a SPI packet or stored into the result buffer. Finally, the interrupt handler feeds the watchdog, enables the same type of interrupt, and returns to the dead loop in the main program. Figure14 is the flow chart of our video content processing program.

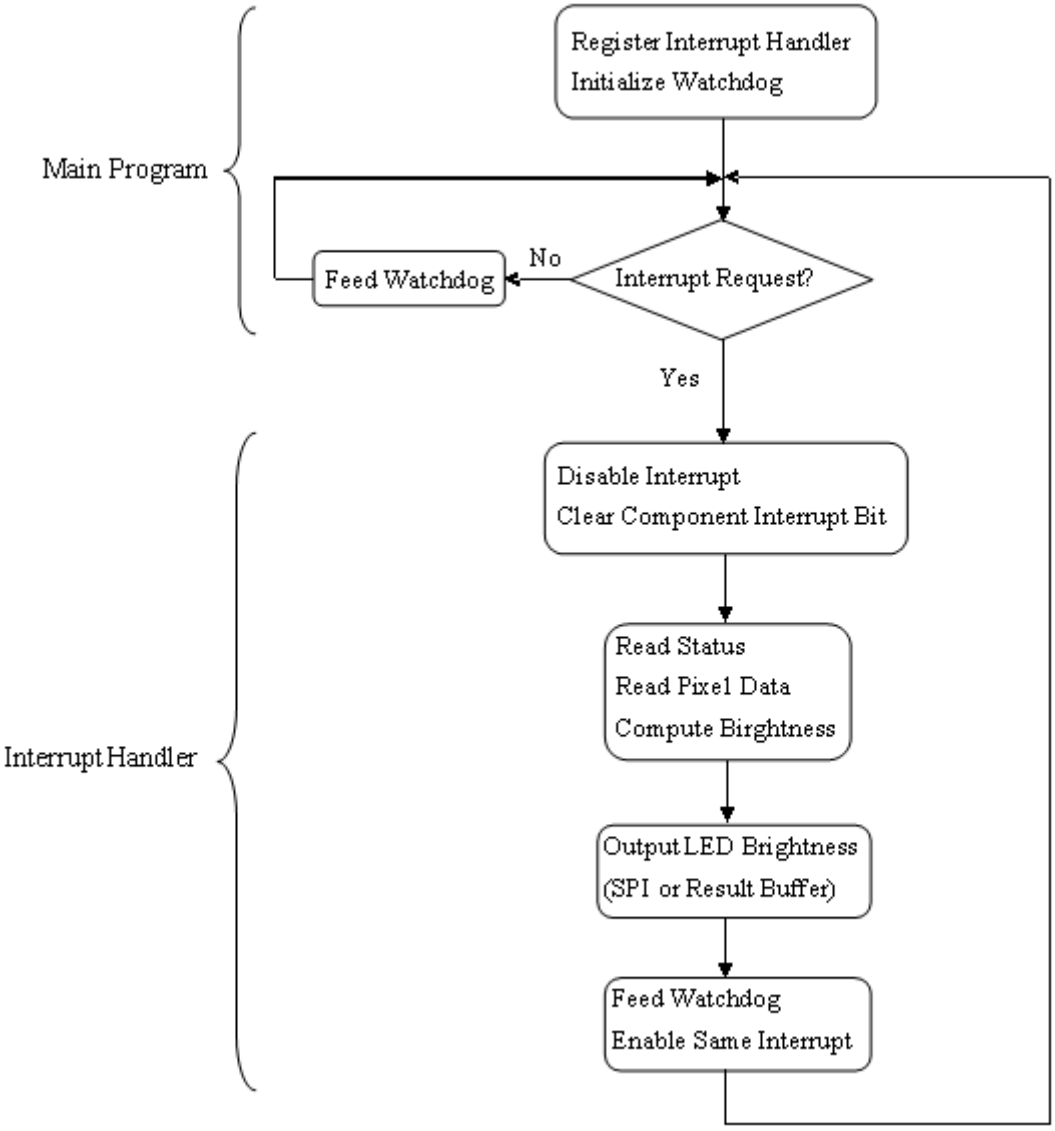


Figure 14: Flow Chart of Video Content Processing Program on the NIOSII.

4.3 Fixed Logic Implementation

In this section we describe the fixed logic implementation of LED brightness generation stage. The design is consisted of a Pixel Dispatcher and 12 Region Accumulators. Generally, the pixel

dispatcher determines which region a pixel belongs to and sends the pixel data along with its regional local addresses to the corresponding region accumulator, which performs summation and averaging of all pixels in its region. Finally, LED brightness outputs are stored in a dual-port result buffer for LED driving circuitry to read. Currently, we have implemented averaging and pattern change detection algorithms on this hardware platform for their simplicity. Other video processing algorithms can be implemented by adding a post process stage that reads regional averages from the result buffer, performs computation on the LED image level, and stores the final brightness output in another buffer. Figure15 illustrates the pipeline of our fixed logic brightness generation stage.

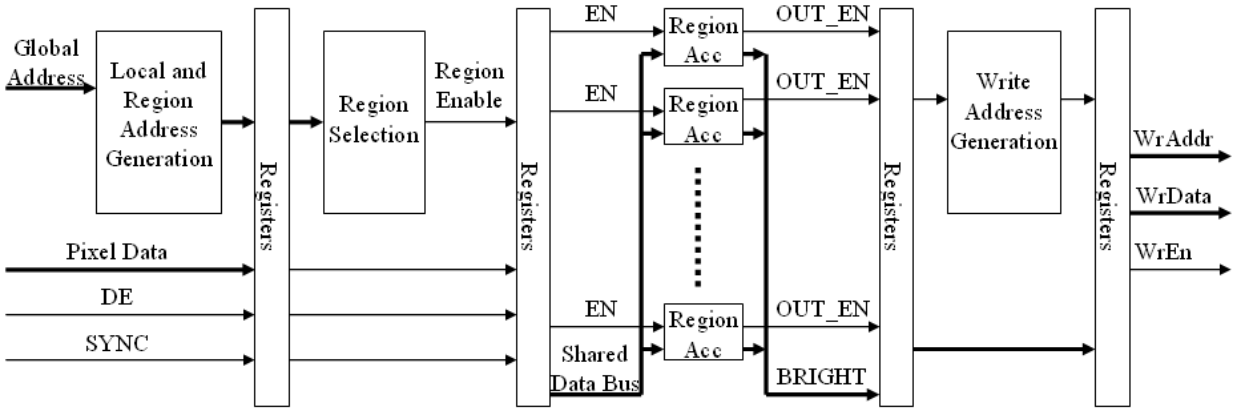


Figure 15: Block Diagram of Pipelined Brightness Generation Stage in the Fixed Logic Implementation.

4.3.1 Pixel Dispatcher

Our pixel dispatcher utilizes a pipeline structure in sending pixel data to region accumulators and writing regional averages to the result buffer. The first stage computes row and column

addresses of the current region as well as local regional addresses of the current pixel. It then selects the region accumulator to which pixel data and its regional addresses should be sent over the bus shared among all accumulators. Each accumulator also has an enable (EN) input individually controlled by the pixel dispatcher. During the third pipeline stage, pixel data and local address are posted on the shared bus while only the EN of selected accumulator is asserted, ensuring current pixel is dispatched to the only region it belongs to. Similarly, output enable (OUT_EN) of each accumulator is connected directly to the last pipeline stage. Only data and address (CH, 2 bits, indicating which color channel) output from the region accumulator whose OUT_EN is asserted are registered by the pixel dispatcher for writing the result buffer. The write address is formed by concatenated current region's row address, column address and the registered CH. Finally performing an OR operation of all OUT_ENs yields the write enable signal.

4.3.2 Region Accumulator

The design of region accumulator is fairly simple and straightforward. It maintains a counter which increments by 1 every time input EN is asserted. This counter resets when its value reaches the total number of pixels within a region. Every received pixel is splitted into RGB values and added to corresponding internal registers. Upon processing the last pixel, the 4-state output control machine is activated, streaming out regional averages of the three channels, one color per state. The 2-bit state value is also submitted to pixel dispatcher as CH.

CHAPTER FIVE: TEST AND VERIFICATION

In this chapter we first describe test methodology of our video content processing system. We then present test results with analysis for both individual components and the entire system.

5.1 Test Methodology

The video input in our experiments is provided by a DVI receiver daughter card manufactured by Terasic [14]. This card is plugged onto our StratixIII development board through the HSMC [15] port B. Pin assignments and I/O standard setting are done in QuartusII's pin planner. The daughter card's 8 bits per channel color data output are wired to the lower 8 bits of pixel data input our system, which is designed for 10 bits per channel at maximum. Moreover, CLK, VSYNC, HSYNC output are feed into our XGA filter.

We use the SignalTapII software logic analyzer in verifying our results. By using SignalTapII, FPAG internal signals can be sampled and transmitted back to a host computer for analysis. Depending on the specific design unit under test, original pixel clock, reduced pixel clock, the SOPC system clock or a test clock generated by on-chip PLL is used as the timing unit in data sampling.

Our experiments are conducted on both versions of the video content processing system. For each version, we first examine waveforms retrieved at component level for our custom modules, including: the video preprocessing stage, the video controller, the vector addition unit for the SOPC-based system and the pixel dispatcher including region accumulators in the fixed logic implementation. After component test, we verify the system functionality by inspecting brightness output. For the result buffer communication mechanism, we place a test pattern generator issuing memory read request periodically aside of our system in the FPGA, since LED driving circuitry is yet prepared for this scheme.

Finally, we use static image in most of the tests for clarity and consistency of signals returned to the host computer.

5.2 Component Test of the NIOSII-Based Implementation

5.2.1 Video Preprocessing Unit

We begin our component level verification by analyzing the XGA filter. Sampled signals include the data valid output, the two sync outputs and values of its vertical and horizontal counters. The input pixel clock is used as the sampling clock. Figure16 captures the vertical counter behavior at transitions of input VSYNC signal. Clearly, if VSYNC is asserted, the vertical counter is reset at the next rising edge of input HSYNC, as in Figure16 (a). On the other hand, it begins incremental counting by 1 after VSYNC is deasserted, as shown by Figure16 (b).

Also we can observe that the vertical counter runs from 1 to 800, the exact number of total scan lines occupied by effective window, front porch and back porch on the vertical dimension of one frame, as listed by Table1 in section 2.1.

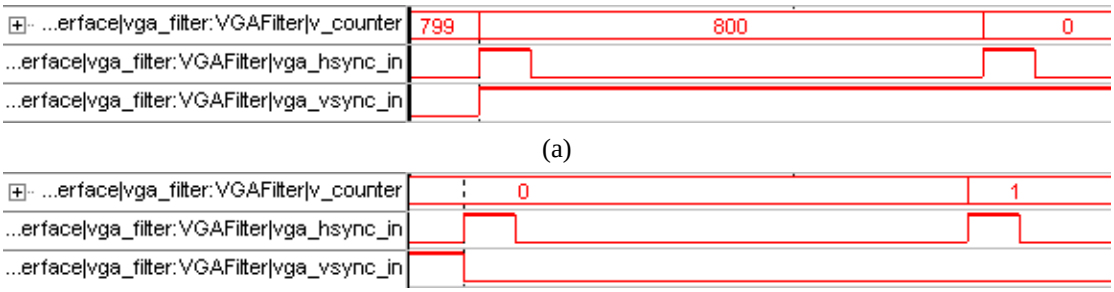


Figure 16: Vertical Counter Behavior in the XGA Filter at Rising (a) and Falling (b) Edge of VSYNC

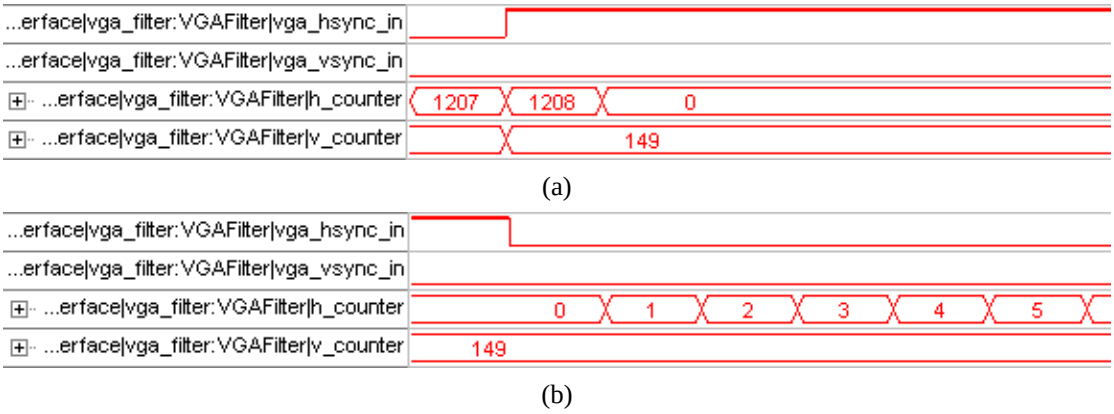


Figure 17: Counters' Behavior in the XGA Filter at Rising (a) and Falling (b) Edge of HSYNC

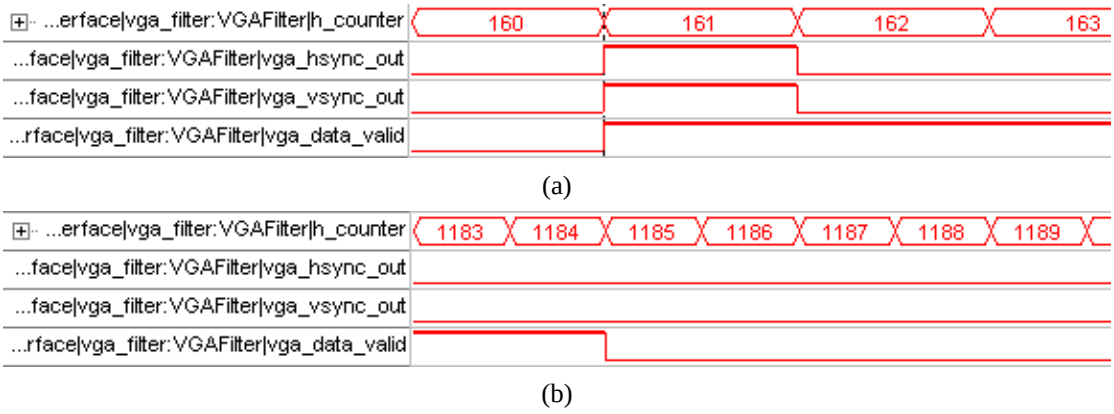


Figure 18: Assertion (a) and Deasseration (b) of Data Valid with HSYNC and VSYNC Transition in the XGA Filter.

As illustrated by Figure17, the horizontal counter in our XGA filter behaves in the same fashion with the vertical counter. Rising edge of input HSYNC resets it to zero while the falling edge starts counting. Moreover, its maximum value, 1208, equals total number of pixels of effective window, front porch and back porch in a row. Finally, Figure18 reflects transition of data valid and sync outputs of the XGA filter. At the first effective pixel of one effective row, both horizontal and vertical sync outputs rise and hold for one pixel clock cycle. The data valid signal also becomes effective at this cycle. Specifically, these events happen when the horizontal counter equals 161, right after the 160 pixels in the back porch of a scan line. From Figure18(b), we can safely conclude that the data valid signal returns to logical low after 1024 cycles, confirming that our XGA filter performs as it is designed.

5.2.2 Pixel Merger and Address Generation Unit

Pixel merger and address generation unit are connected in parallel next to the XGA filter, forming the next stage on the system's data path. We retain the input pixel clock as the sampling clock in testing of these to components. Figure19 mirrors internal operations of the pixel merger upon the start of a valid row. The 4-state machine in pixel merger is activated at the rising edge of *vga_data_valid* from the XGA filter. Four input pixels are then accumulated before their average is posted on the output bus. The reduced pixel clock also transits to logical high in sync with the machine return to the initial state, completing a "merge" cycle.

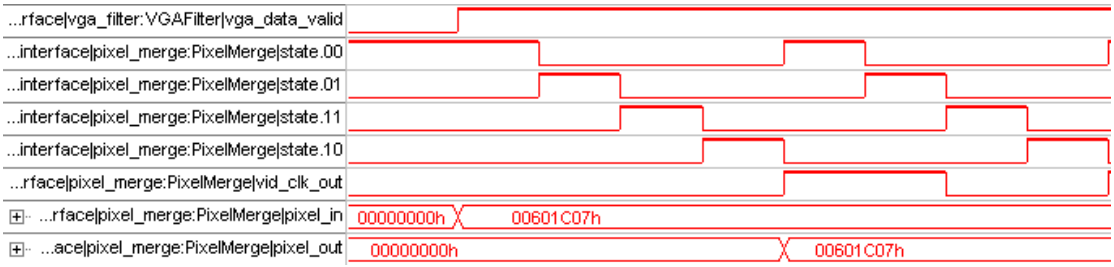
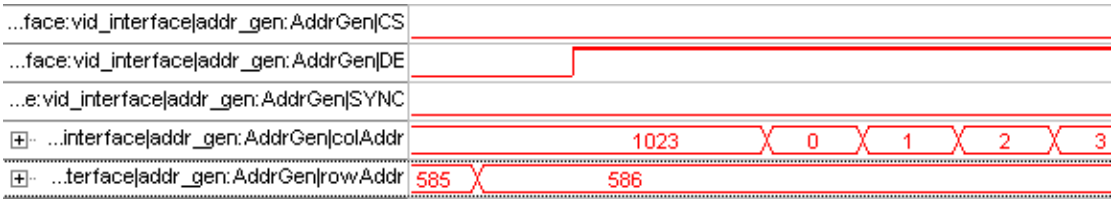


Figure 19: State Transition, Pixel and Clock Output of the Pixel Merger.

Figure20 exemplifies column and row address registers' behavior during an effective scan line. Firstly, row address is incremented by 1 and an internal column address register is set to 0 at the rising edge of the XGA filter's HSYNC output. However, since the pixel merger and address generation unit work in parallel, column address should be delayed so that its arrival at the video controller is synchronized with that of merged pixel data, which is delayed for 4 cycles by the output state machine of pixel merger. Moreover, left shifting the column address by 2 automatically produces the reduced column address. Therefore, we decide to delay the column address output by three cycles to ensure that the reduced address is stable at the rising edges of reduced pixel clock. As shown in Figure20 (a), the column address output becomes 0 three cycles later than the transition of row address. Also in this figure, the DE output has one cycle delay due to its dependence to row address. Nonetheless, the first rising edge of reduced pixel clock happens after the fourth cycle from row address change, excluding timing issues at the video controller end. Figure20 (b) shows the column address change toward the end of a scan line. Upon reaching the value of 1023 it freezes until the next rising edge of HSYNC input.

Also shown in this figure are small notches on the waveform of DE output which signifies "jumped" columns.



(a)

(b)

Figure 20: Assertion (a) and Transition (b) of DE within a Valid Row in the Address Generation Unit.

Finally, CS and SYNC outputs are verified in Figure21. SYNC is asserted at the beginning of the scan line following the last effective row in the sub-frame. Recall that we vertically sub-sampled a frame by three so there are two non-effective rows before the next effective one. SYNC holds for one entire row to make sure down stream logic receive it correctly. Further more, we delay the flip of CS for 256 cycles to avoid pipelining effect: the new state of CS is mistakenly used due to delay in the system.

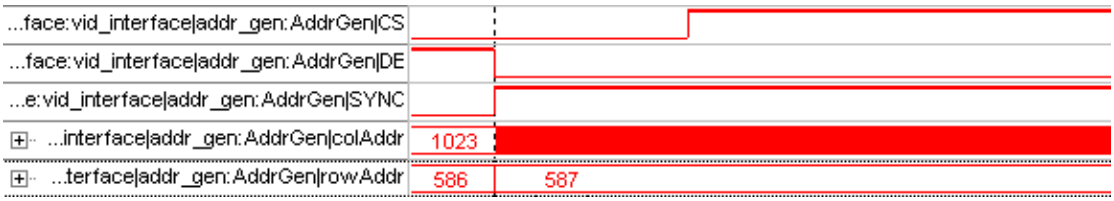


Figure 21: CS, DE and SYNC Behavior in the Address Generation Unit at the End of One Sub-frame.

5.2.3 Video controller

In this section we examine the behavior of our video controller which bridges the video preprocessing stage and the SOPC system. Results presented here are sampled by the SOPC system clock. Figure22 illustrates transitions of its internal pixel counter, which is part of the write address, and write enable output to on-chip buffers. After input DE becomes effective (a), the counter increments by 1 every time a pixel is received. Simultaneously, write enable is asserted at the first cycle of the new counter value. During a small period when DE is not asserted (b), the counting stops and no memory write request is issued, corresponding to a "jumped" pixel. At the end of a row (c), the counter freezes and write enable stays at logical low since the input pixel clock is disabled by the XGA filter through the pixel merger.

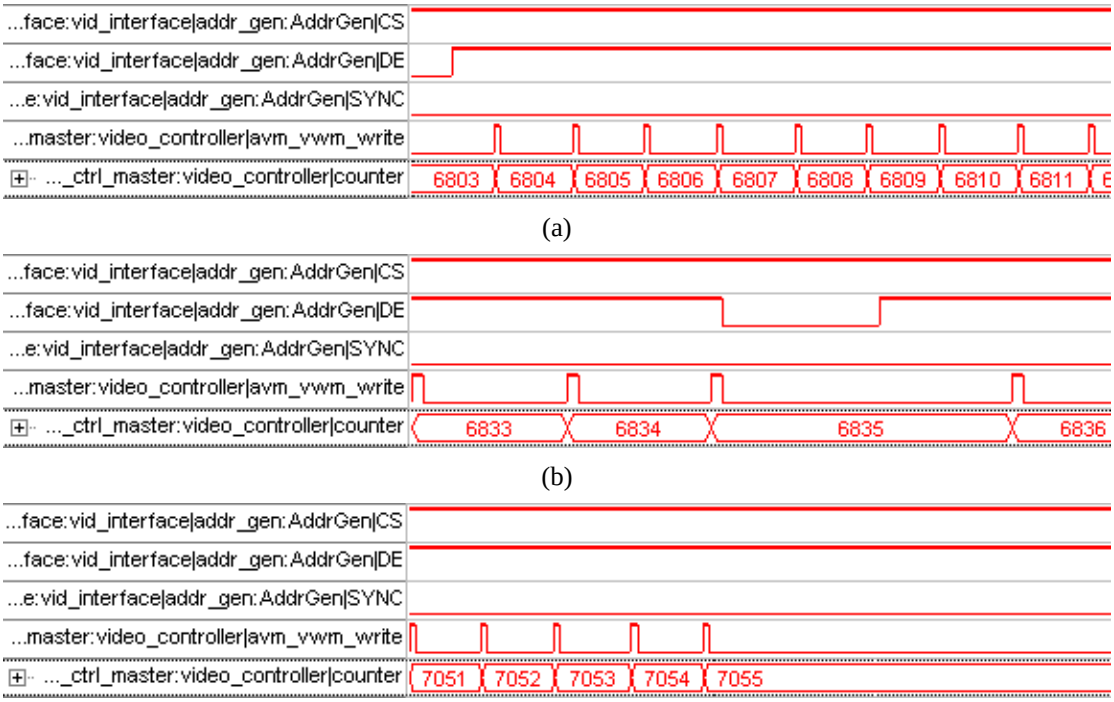


Figure 22: Video Controller Behavior at the Beginning (a), Center Jump (b), and End (c) of a Valid Row.

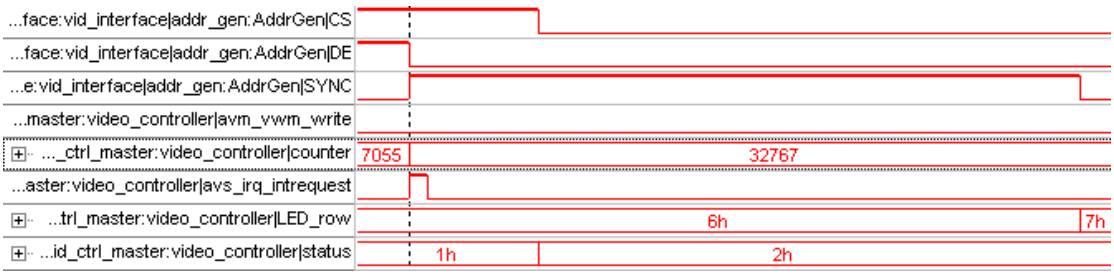
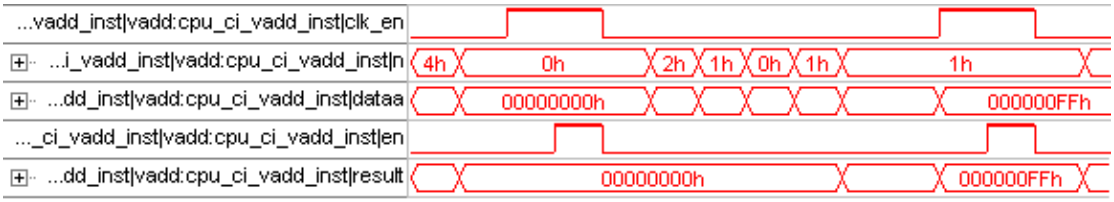


Figure 23: Video Controller Behavior at the End of a Sub-frame.

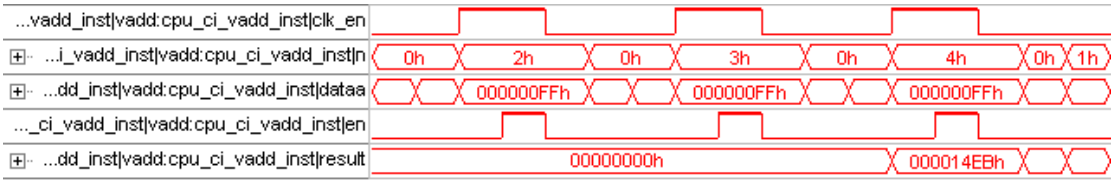
As shown in Figure23, after receiving the last pixel of one sub-frame, the video controller sends an interrupt request to NIOSII. At the same time, SYNC input resets the internal counter to 32767; this value is chosen so that the first pixel of next sub-frame is associated with counter value 0, routing it to the first 32-bit word in a memory buffer. Importantly, update of *LED_row* and *status* is delayed for allowing NIOSII to read old yet accurate values in determining which sub-frame is waiting for processing in which buffer.

5.2.4 Vector Addition Unit

The vector addition unit should receive special attention in the SOPC system since it implements a vital stage in all algorithms specified in chapter 3. For clarity, we use a full blue image as the test pattern for this unit. Test results presented in this section are sampled by the SOPC system clock.



(a)



(b)

Figure 24: Initialization and Reading Result of the Vector Addition Unit.

Figure24 highlights internal state and output transition of the vector addition unit in following scenarios. Firstly, the NIOSII issues function code 0x0 that clears internal data registers of the custom instruction module. Next, for each pixel the NIOSII posts function code is 0x1 and pixel data in sync with clock enable. If regional pattern change detection is enable, the module selects the maximum of RGB channels and routes this value to its result output, as shown in Figure24 (a). Finally, NIOSII collects accumulated RGB result by issuing function code 0x2, 0x3, and 0x4 correspondingly, after 21 pixels, one row of a LED zone, are sent to the vector addition unit. Given the test pattern used, red and green channels should be zero while blue channel is expected to be 5355 (255 multiplied by 21), which number is 0x14eb in hexadecimal format. Figure24 (b) conforms that arithmetic operation of the vector addition unit is correct.

5.2.5 Performance Concern of the SPI Communication Method

Before rushing into brightness result verification, it is instructive to measure our video content processing program's performance in comparison with allowed processing time for each sub-frame. We use a 4MHz test clock in sampling since it is fast enough to capture SPI activity at 1MHz and also slow enough to record events like NIOSII interrupt over multiple sub-frames. Also, we use the simple averaging algorithm in this test.

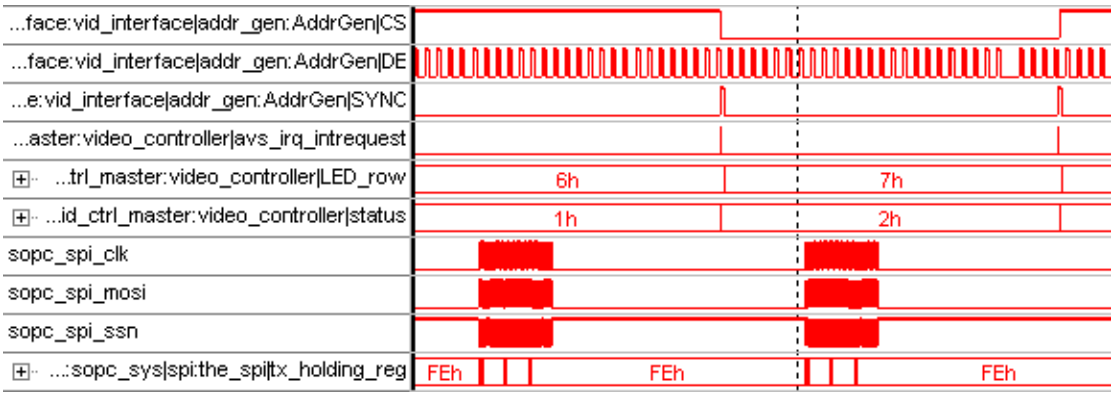


Figure 25: Time Consumption of SPI Communication Method in the Time Period of One Sub-frame.

Figure25 reflects time consumption of the SPI output during the processing time of one sub-frame. Clearly, outputting brightness result takes nearly one quarter of the time between two interrupt requests. Since the SPI peripheral does not have internal buffer, the output packet has to be streamed to SPI's transmitting register byte by byte in software. Also, the program function controlling SPI only returns after one byte is sent out over the serial wire. Therefore, NIOSII can not overlap SPI communication with other task in order to improve utilization rate, wasting precious time could be used for video content processing.

5.3 Brightness Result Verification of the NIOSII-based Implementation

Through section 5.2, we have shown that our custom logic modules in the video content processing system function as they are designed. In this section we examine the final brightness output using the four test images shown in Figure26. Test patterns include gray scale gradient (a), pure black back ground with white "jumped" columns (b), alternating white and black (c) and RGB strips (d). Since all the four images do not variant vertically, we only record SPI output of one LED row. Lastly, we use the same test clock in section 5.2.5 for sampling and choose histogram equalization as the video processing algorithm.

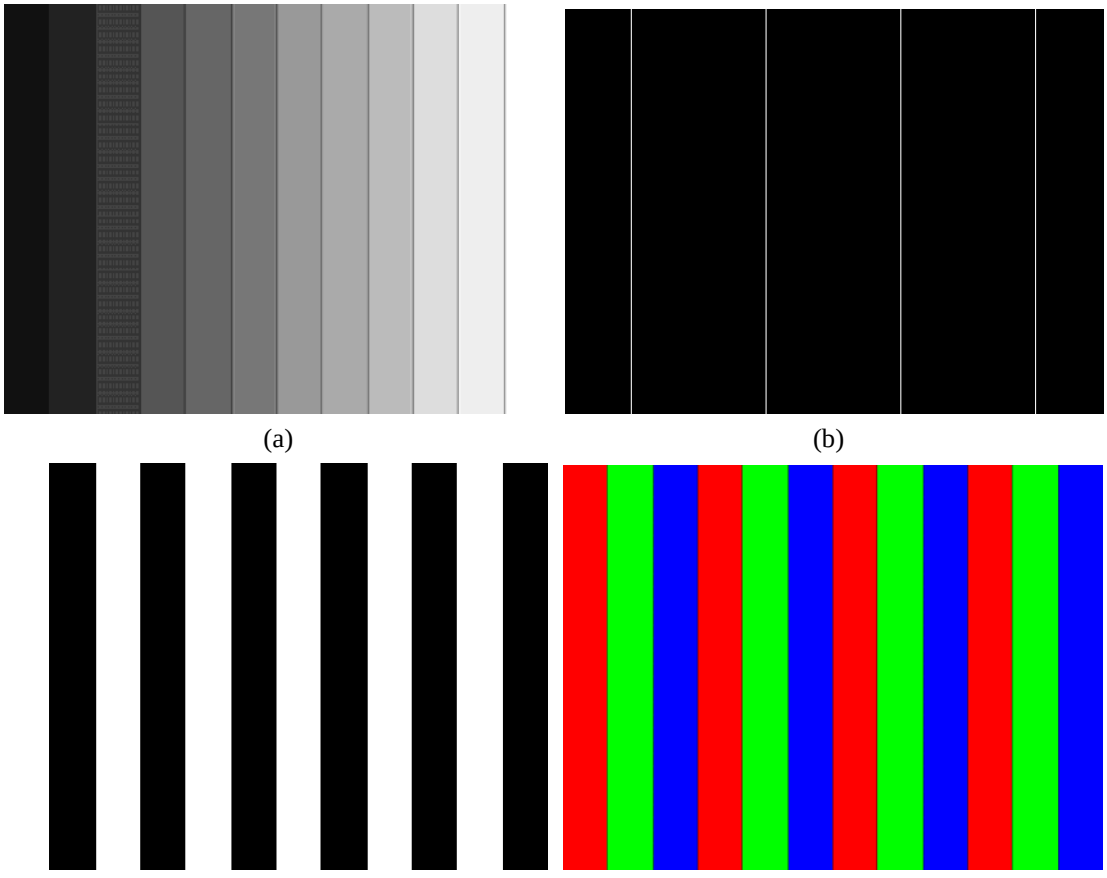


Figure 26: Test Images Used for Verifying SPI Brightness Output

The observed SPI output results are tabulated in Table3. For image (a), LED brightness demonstrates linear increase across the row in accordance with intensity change of the tested pattern. In the next group of results, all LEDs are set to the minimum value of our SPI communication protocol, indicating the white columns in image (b) are "jumped" in video processing. Moreover, results group (c) and (d) reproduce the alternating pattern of test image (c) and (d) respectively. Evidently, our video content processing system adjusts LED brightness on individual RGB channels in response to the input image.

Table 3: SPI Brightness Output of the Four Test Images in Figure26.

Image/LED		0	1	2	3	4	5	6	7	8	9	10	11
(a)	R	10	23	46	69	92	115	139	162	185	208	231	254
	G	10	23	46	69	92	115	139	162	185	208	231	254
	B	10	23	46	69	92	115	139	162	185	208	231	254
(b)	R	10	10	10	10	10	10	10	10	10	10	10	10
	G	10	10	10	10	10	10	10	10	10	10	10	10
	B	10	10	10	10	10	10	10	10	10	10	10	10
(c)	R	254	10	254	10	254	10	254	10	254	10	254	10
	G	254	10	254	10	254	10	254	10	254	10	254	10
	B	254	10	254	10	254	10	254	10	254	10	254	10
(d)	R	254	10	10	254	10	10	254	10	10	254	10	10
	G	10	254	10	10	254	10	10	254	10	10	254	10
	B	10	10	254	10	10	254	10	10	254	10	10	254

5.3.1 Verification of the Result Buffer Communication Method

We have also tested the alternative Result Buffer method for brightness communication. We modified the NIOSII program to store SPI packet of each LED row into the result buffer in a circular manner: after the last row, NIOSII overwrites the first SPI packet with updated

information. Also, Regional Pattern Change Detection is enabled in addition to histogram equalization for increasing the processor's workload. We retained the SPI module for outputting one byte (the LED row address) near the end of the interrupt handler to signify completion of video processing for one sub-frame. To mimic memory read requests to the result buffer from external circuitry, we implement a test pattern generator operating at the 4MHz test clock. For conciseness, a full blue image is used as test pattern.

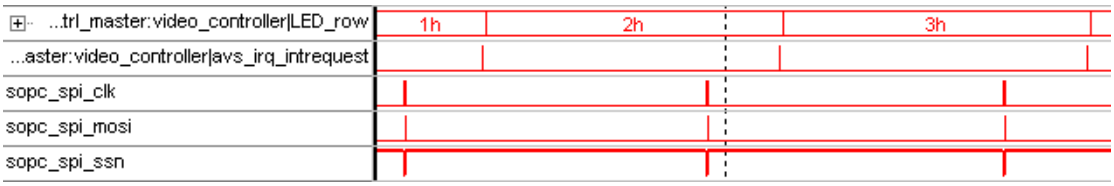


Figure 27: Time Consumption of Result Buffer Communication Method in the Time Period of One Sub-frame.

As shown by Figure27, processing time of one sub-frame is better utilized since the bulky SPI communication is removed from the program. The Pattern Change Detection algorithm increases computation amount for each pixel by a factor of nearly 3 but can still fit into the allowed time period with room left for further computation. If SPI communication is used, this algorithm is clearly excluded from the list due to inadequate time.

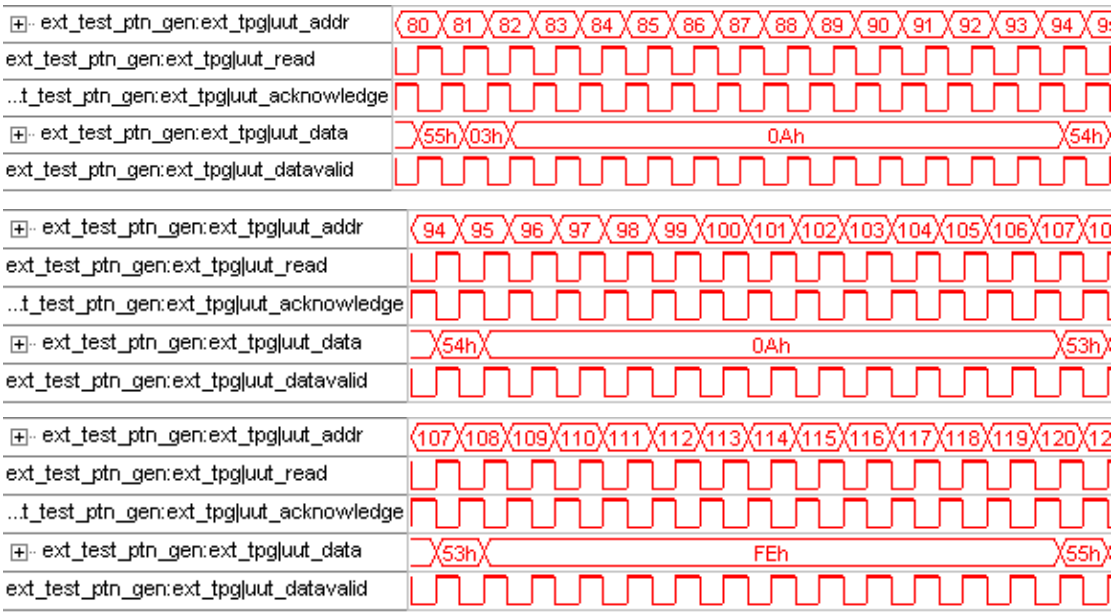


Figure 28: Content of the Result Buffer from Address 80 to 120.

Figure28 pulls up the third LED row's brightness packet from our result buffer. 80 is the start address since packets of the first and the second row each occupy 40 bytes. Obviously, data stored in the result buffer from address 80 through 120 match the packet format exactly, validating that our result buffer method is ready for test with external circuitry.

5.4 Test of the Fixed Logic Implementation

In this section we analyze test results of our fixed logic implementation of the brightness generation stage. Waveforms presented here are sampled by the reduced pixel clock. The test image used is composed of full (0xff) red, zero (0x00) green and half (0x0f) blue. In addition, we use averaging as the video processing algorithm.

5.4.1 Pixel Dispatching

Figure29 provides an overview of the pixel dispatcher's operation. Firstly, pixel's local region column address counts from 0 to 20 but has a pausation if DE is deasserted for "jumped" columns. Secondly, we maintain two region column addresses differentiated by 1: the first selects which region accumulator should be enabled for the current pixel; the delayed one is use as part of the write address of result buffer. Finally, enable signal of one accumulator is asserted one cycle later from pixel's local address becoming 0, as region selection is done in the second stage of pixel dispatcher's pipeline.

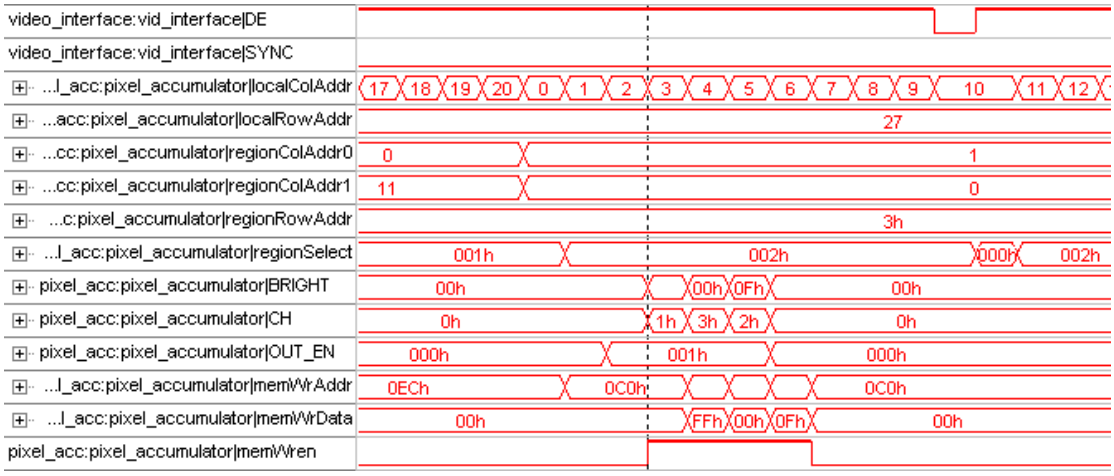


Figure 29: Transitions of Signals in the Pixel Dispatcher

Figure30 exemplifies overall (a) and detailed (b) view of pixel counter behavior inside of regional accumulators. Clearly, a counter is only activated by the enable input of the accumulator it belongs to, as reflected by Figure30 (a). The counter remains unchanged as long

as it is not selected by the pixel dispatcher, including the cycles associated with a "jumped" column.

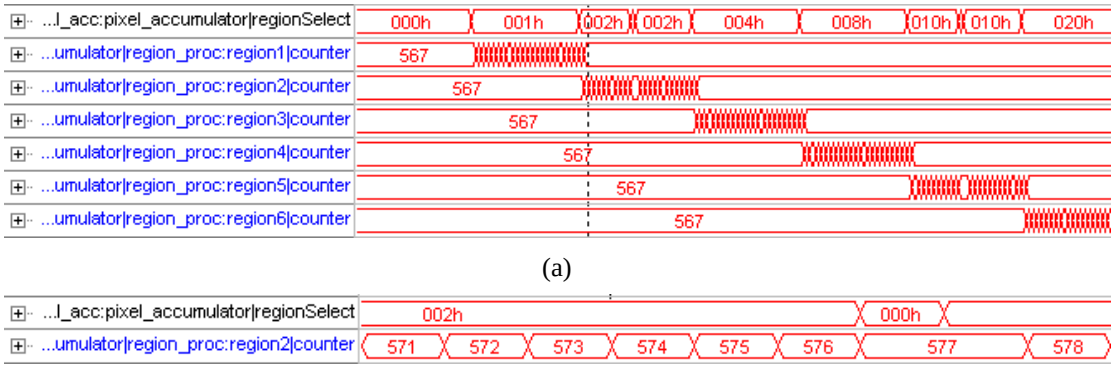


Figure 30: Overview (a) and Detail (b) of Pixel Counter Behavior in Region Accumulators.

5.4.2 Result Collection

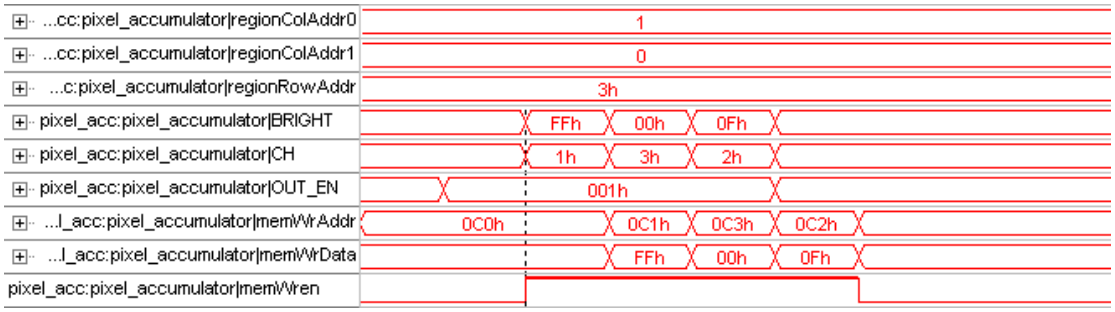


Figure 31: Writing Regional Averages to Result Buffer.

Figure31 is a zoomed-in view of part of Figure28 with special interest in collecting regional averages and writing them into result buffer. The 10-bit writing address is produced by concatenating 4-bit region row address, 4-bit region column address and the 2-bit state output of a region accumulator. For example, 0x3, 0x0 and 01 together produce 0x0c1 as the target address in result buffer. Finally, since memory interfacing signals are registered in pixel

dispatcher, their transitions appear on the output ports one cycle later than that of corresponding signals from a region accumulator, as shown in the figure.

5.4.3 Brightness Result Verification

Our last test is aimed to verify that the result buffer mechanism functions properly in the fixed logic implementation of brightness generation stage. Again we use an internal test pattern generator for memory read request. This time we issue the request in sync with the reduced pixel clock.

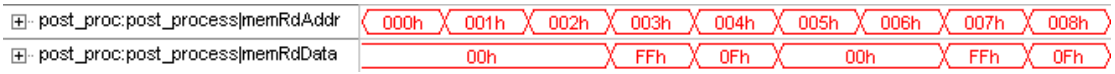


Figure 32: Reading Regional Averages from Result Buffer.

In Figure32, it is clear that brightness information of individual LED color channels is accurately retrieved from the result buffer. Brightness data of blue channel is arranged before that of the green channel in memory layout due to using of Gray Encoding in region accumulators' output state machine. It seems to be a problem that output data are delayed 2 cycles from its corresponding address is sent to the result buffer. Nevertheless this phenomenon is expected since we configure the RAM module of result buffer to have registered I/O ports.

CHAPTER SIX: CONCLUSION AND FUTURE WORKS

In this paper, we present our design of a FPGA-based video content processing system for LED backlight driving used in LCDs. Our system takes a 1024 by 768 resolution and 60Hz frame rate video input, computes brightness for a 12 by 9 LED backlight array, and outputs brightness information to LED driving circuitry through either SPI or Result Buffer mechanism. Our contribution includes: Firstly, we managed to filter out blank pixels in transmitted video stream. After the filtering, we successfully reduce the frame resolution to 252 by 252 for timely processing, retaining alignment between front LC panel and backlight LED array by utilizing the "Center Jump" technique. Moreover, we build two implementations of the brightness generation stage, one is based on Altera's IP cores (NIOSII, etc.) while the other is constructed from our custom hardware logic. We have validated the NIOSII-based system with SPI communication by integrating our FPGA system with the LED driving circuitry into a consolidated display. In addition, by using internal test pattern generators, we can conclude that both implementations of brightness generation function properly with the result buffer scheme.

On video content processing, four algorithms have been implemented and tested on the NIOSII-based system, including averaging, histogram equalization, LED zone pattern change

detection and non-linear mapping. At the time this paper is written, averaging and pattern change detection have been ported to the fixed logic implementation.

Our future research effort is two-fold: first, we are interested in evaluating more video content processing algorithms proposed by the display community, like IMF [15] and DCA [16]. Further more, we would like to improve FPGA resource utilization and power efficiency of our fixed logic system. The current design is not efficient on that a region accumulator is only activated for one twelfth of the time. A rewarding direction is to maintain a register set for each LED zone and consolidate the accumulators to one regional processing unit switching among the register sets.

LIST OF REFERENCES

[1] H. Seetzen, et al “High Dynamic Range Display Systems,” *In Proceedings. of ACM SIGGRAPH 2004*, Aug. 2004

[2] VGA Video, http://web.mit.edu/6.111/www/s2004/NEWKIT/vga.shtml, retrieved at May 2010

[3] XGA Signal 1024 x 768 @ 60 Hz Timing, http://tinyvga.com/vga-timing/1024x768@60Hz, retrieved at May 2010

[4] Altera "StratixIII FPGA Device Handbook," *Version 10.0*, July 2010

[5] Altera "QuartusII Handbook," *Version 9.1*, November 2009

[6] Altera "Embedded Design Handbook," *Version 2.7*, March 2010

[7] Altera "NIOSII Software Developer's Handbook," *Version 9.1*, November 2009

[8] Wikipedia http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus, retrieved at June 2010

[9] R. C. Gonzalez and R. E. Woods "Digital Image Processing", *2nd Edition, Prentice-Hall Inc., New Jersey*, pp. 91-94, 2002

[10] Wikipedia http://en.wikipedia.org/wiki/Histogram_equalization, retrieved at June 2010

[11] Wikipedia http://en.wikipedia.org/wiki/Gray_Encoding, retrieved at May 2010

[12] Altera "NIOSII Custom Instruction User Guide," *Version 1.5*, May 2008

[13] Altera "Avalon Interface Specifications," *Version 1.2*, April 2009

[14] Terasic "DVI-HSMC Daughter Card Specification", *http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=68&No=359&PartNo=2,* retrieved at July 2010

[15] Altera "High Speed Mezzanine Card (HSMC) Specification", *Version 1.7*, June 2009

[16] F. C. Lin, et al "Dynamic Backlight Gamma on High Dynamic Range LCD TVs," *Journal of Display Technology*, Vol. 4, Issue 2, pp. 139-146, 2008

[17] G. Z. Wang, et al "Delta Color Adjustment (DCA) for Spatial Modulated Color Backlight Algorithm on High Dynamic Range LCD TVs", *Journal of Display Technology*, Vol. 6, Issue 6, pp. 215-220, 2010