

**ECE 532 design project 2009**

## **Group Report**

# **Real Time Color Replacement**



Chang Liu (995416250)  
Andrei Bajenov (995478763)  
Jeffery Sham (995384310)

April 9, 2009

## Table of Contents

---

|  | PAGE |
|--|------|
| 1. Overview .....                                    | 3    |
| 1.1 Goals .....                                      | 3    |
| 1.2 Project Background .....                         | 3    |
| 1.3 Invisibility Effect .....                        | 3    |
| 1.4 System Block Diagram .....                       | 5    |
| 1.5 Brief Description of IP .....                    | 5    |
| 1.6 Clock Domain .....                               | 7    |
| 2. Project Outcome .....                             | 7    |
| 2.1 Review of Original Plan .....                    | 7    |
| 2.2 Our Final Product .....                          | 8    |
| 2.3 Suggestions For the Future .....                 | 8    |
| 3. Description of Blocks .....                       | 9    |
| 3.1 MicroBlaze .....                                 | 9    |
| 3.2 PLB Bus .....                                    | 10   |
| 3.3 IIC Interface .....                              | 11   |
| 3.4 TFT Controller .....                             | 11   |
| 3.5 MPMC Controller .....                            | 12   |
| 3.6 Decoder Logic .....                              | 12   |
| 3.7.1 IP_to_Mem Custom Logic .....                   | 12   |
| 3.7.2 Color Detection Algorithm .....                | 15   |
| 3.8 Mem_to_IP Custom Logic .....                     | 16   |
| 3.9 Connection between IP_to_Mem and Mem_to_IP ..... | 18   |
| 4. Description of Our Design Tree .....              | 19   |
| 5. Reference .....                                   | 20   |
| 6. Appendix .....                                    | 21   |
| 6.1 PLB Burst Write Timing Diagram .....             | 21   |
| 6.2 PLB Burst Read Timing Diagram .....              | 22   |
| 6.3 PLB Single Beat Read Timing Diagram .....        | 23   |

# OVERVIEW

## *1.1 Goals*

---

The goal of our project was to develop a system which performs real-time color replacement of a video stream. In particular, our system will detect a certain colored pixel in the video frames and alter it with a pixel from a different image in the video output. Using this concept, we are not only able to implement a simple commercial Chroma Key using only hardware, but to also develop a device that can make you completely invisible on camera!

## *1.2 Project Background*

---

Chroma keying, more commonly known as “green screening”, is a technique used in movie production to film scenes that are too difficult to achieve with props, set pieces, and other special effects. In Chroma keying, the actor performs a scene in front of a coloured screen and later, the specified colour is filtered out and replaced with the desired background. There are a number of subtleties involved in perfecting Chroma keying that makes it a tedious process. This includes correctly defining the colour to key out in order to avoid “colour bleeding” and applying even lighting on the backdrop and subject to achieve the best contrast. However, for the purposes of our application, we are more interested in demonstrating that the process can be done in real-time, and will accept the slightly lower quality result given the limitations of our lighting and backdrop equipment. Thus, our goal is to be able to take in video frames from the camera, detect the pixels with a backdrop colour, and replace them with an image that is saved in memory. In order to satisfy the real-time criteria, the processing should be fast enough that when the foreground (subject) changes, there is no detectable glitch in the video output.

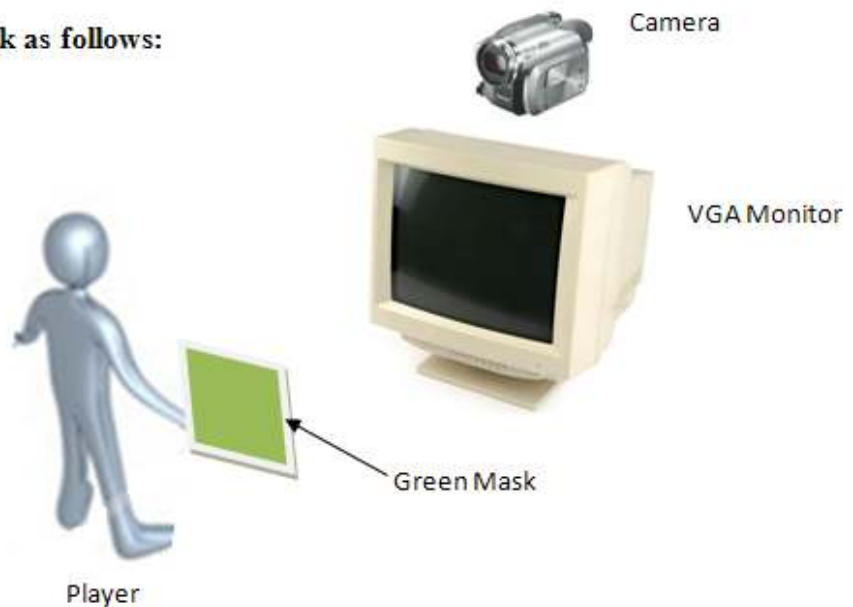
## *1.3 Invisibility Effect*

---

The trick behind the invisibility effect is taking a still shot of the background before the subject enters the set and replace the desired colour with the saved image to create the illusion that the subject has disappeared. This requires that the camera does not move and that the background is not changed. Failure to do so will cause discrepancies between the saved background and actual background, thus ruining the effect. In terms of the technical aspect, we first need to be able to save the background as a static frame somewhere in our memory. This will allow to know exactly what the background is at all times during the live stream even when something or someone is blocking it. There are key similarities in the implementation of Chroma key and the

invisibility effect that allowed us to achieve both with very little addition. In the original Chroma keying concept, the static image comes from non-volatile memory such as compact flash. In the invisibility effect, the static frame comes from the initial frame input from the camera. In both cases, we would need to use memory for the frame and we must create our hardware so that it will read in lines of data for the pixel replacement. The slight extension in implementing the invisibility effect is the extra logic required in our hardware to save the initial frame into DDR memory. However, it should be noted that if our goal was only to replace one colour with another colour, we can simply modify the video stream as it streams to VGA and bypass one of our bigger challenges of utilizing memory in our design. The diagram below shows a high level layout of our system.

**The equipment setup will look as follows:**



[1] Camera image : [http://www.mobilewhack.com/images/panasonic\\_vdr\\_d400\\_dvd\\_video\\_camera.jpg](http://www.mobilewhack.com/images/panasonic_vdr_d400_dvd_video_camera.jpg)

[2] CRT image: <http://www.computermonitors.us/crt-424.jpg>



|                                      |   |  |
|--------------------------------------|---|--|
|                                      |   | VideoCapture                                       |
| IP_To_Mem                            | Detect the target color from the incoming frame, replace it with pixels from the Mem_to_IP module, and then stores altered frame into memory.<br>Bus Master – uses burst write  | Custom Logic                                       |
| Mem_To_IP                            | Read a frame from the memory and sends the data to the Ip_to_mem module<br>Bus Master – uses burst read   | Custom Logic                                       |
| MicroBlaze_0 + dlmb + ilmb           | 1. send commands and configs to the IIC.<br>2. used by XMD to write configuration data to the memory  | Xilinx IP  |
| MPMC_controller                      | It allows us to access the same piece of memory from multiple ports   | Xilinx IP  |
| PLB_bus                              | Four PLB bus were used in this project:<br>1. Used for the microblaze to talk to the DDR and iic<br>2. Used between IP_To_Mem and memory<br>3. Used between Mem_To_IP and memory<br>4. Used by the TFT to get data from the DDR | Xilinx IP  |
| TFT_controller                       | Reads a video frame from the memory and sends it to the VGA monitor to be displayed   | Xilinx IP  |
| Software block inside the MicroBlaze |   |  |
| ddr_test.c                           | This configures the iic hardware module   | Modified from video_capture_0 from digilentinc.com |
| Devices outside the FPGA             |   |  |
| DDR memory                           | The DDR stores the static image of the background, the incoming frame of the live video, and the 32-bit information that controls the color detection algorithm configuration.  | Given  |
| VGA monitor                          | Display the output  | Given  |
| Video Camera                         | Capture the video and streams it to vdec in composite format  | Self supplied                                      |
| Video Decode Chip (vdec)             | Reads the video from the camera through composite and sends the data to the decoder_logic   | Given  |

## 1.6 Clock Domain

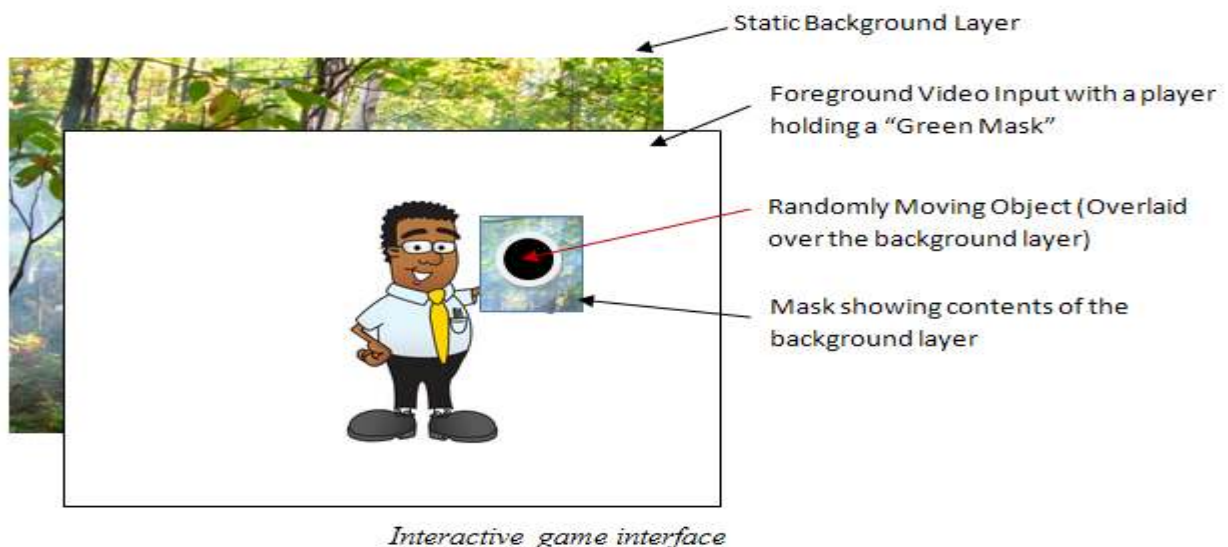
| Component  | Clock Speed |
|--|-------------|
| Video In   | 27MHz       |
| Custom Hardware (IP_to_mem, mem_to_IP),<br>other system components | 100MHz      |
| DDR Memory   | 133MHz      |
| VGA  | 27 MHz      |

The video in is in a clock domain of 27MHz which includes the Vdec, and decoder logic. The custom hardware runs at 100 MHz and this domain crossing is handled by the asynchronous double line buffer. The DDR memory runs at 133 MHz, therefore the MPMC handles all clock domain crossing issues between system components (PLB Bus, custom hardware) which all run at 100 MHz. Finally, the VGA output is running at 27 MHz and the crossing from 100 MHz to 27 MHz is handled by the TFT.

## PROJECT OUTCOME

### 2.1 Review of Original Plan

Chroma Key has been the key application of our project since the beginning, but the invisibly effect was not realized until a little pass halfway point of this course. Initially in our proposal (in addition to the Chroma Key system), we decided to apply the video color detection concept to create a single player interactive game. See figure below for the game play setup.



The player interacts with the game system by motioning a small green mask (square) in front of the camera. When the player moves the mask, a square on the monitor will move correspondingly. The player can see himself/herself in the video output but the green mask is replaced by part of a background image, like a cut-through to another layer. The game begins when the player finds a small circle placed randomly on the screen in the background layer. Once detected, the circle will begin to move in random directions and the game's objective is to keep the circle from leaving the scope of the green square mask on the monitor. Like all video games, this game system trains hand eye coordination, reflex, and the inverted aspect of the camera adds additional difficulty.

## *2.2 Our Final Product*

---

As mentioned earlier, we decided to replace the interactive video game with an invisibility effect system. We decided to change system due to a number of reasons:

- We also believe the invisibility effect is more innovative than a video game system. The interactive game idea has been implemented by groups in the past and is where we got the inspiration. In contrast, the invisibility effect has never been done before, which offered us more of a challenge.
- Since this course is mainly based on hardware we did not want to commit too much time on developing a game, which would include a large portion of software.

In the end, our system turned out to be successful. It performed fairly well under a reasonable range of lighting conditions and produced video output of acceptable quality. In addition to the features specified in the proposal, we were also able to build our system so that it was easily configurable to use different replacement techniques and to detect different colours. This was partially driven by a need to calibrate and debug quickly, but it turned out to be an excellent addition feature to the system.

## *2.3 Suggestion for the future.*

---

In terms of hardware modification, it would be interesting to implement some of the more sophisticated colour replacement algorithms we came upon in our research. These algorithms involved using surrounding pixels in the replacement decision and we believe that may outperform the current algorithms employed especially in handling the edges of a replaced object. In our current strategy, only the current pixel being processed is taken into consideration in replacement, which sometimes fails at the edges since the way light reflects off these points are often different from the rest of the object. The added difficult of implementing these algorithms



arise from the fact that more lines must be burst read and made available to the replacement logic per pixel.

Another interesting concept worth pursuing is for the custom hardware to keep track of the percentage of pixels on frame that are replaced and use that to determine whether to recapture the background image. For example, if less than 5% of the pixels are replaced, it would conclude that it is currently viewing the background and save it as a static frame. Later when the replacement target comes on screen, it would not try to save background. This notion of auto capture would allow for the background to change over time and even slow movement of the camera provided that the replacement target leaves the frame intermittently. This would compensate for one of the shortcomings of the current system, which only saves a background on initialization. A further improvement could be to periodically recapture the regions of pixels that are not being replaced, which would allow the background to update even when the replacement object does not leave the frame.

We can also improve the user friendliness of the system by building a menu and allow users to configure the device entirely through UART. This is much easier than the current method which requires the user to refer to a table and input the correct configuration word into XMD and write to memory. Using UART, the user can simply select the colour replacement options using keystrokes which is much more intuitive for a non-technical user.

## Description of the Blocks

### *3.1 Microblaze\_0 (MicroBlaze)*

---

The MicroBlaze is a built-in soft processor that was created for use in the FPGA. The processor is automatically created for us when we started the project wizard in EDK. Even though we did not make software applications in our system, (ie game code), we still used the Microblaze for two purposes:

1. The Microblaze contains the configuration instructions that are used by the IIC interface.
2. We used the XMD terminal to write the color-control word to a fixed memory location, and the XMD requires the existence of the Microblaze.

### 3.2 PLB Bus (plb\_v64)

PLB buses are obtained from the Xilinx IP catalog under the name of plb\_v64. The PLB bus provides a connection between an optional numbers of PLB masters and slaves. It became the key transportation of data from one module to another. To eliminate bottleneck to the Memory, we used for PLB buses in our system,

1. Used for the microblaze to communicate with the DDR and IIC
2. Used between IP\_To\_Mem and memory
3. Used between Mem\_To\_IP and memory
4. Used by the TFT to get data from the DDR

| bandwidth calculation                      |                                    |
|--|------------------------------------|
| Video speed calculation                    |                                    |
| Video Input Frame Size                     | 858x525 pixels                     |
| Pixel size                                 | 4 bytes / pixel                    |
| Clock Speed                                | 27 MHz                             |
| Time to Process One Line                   | 858 pixels / 27 MHz = 0.000032 s   |
| Bus speed calculation                      |                                    |
| Our bus clock speed is                     | 100 MHz                            |
| Reading and writing burst speed            | 4 bytes / clock cyc                |
| Time to Burst one Line                     | 640 pixels / 100 MHz = 0.0000064 s |
| number of transactions                     |                                    |
| - read the static frame                    |                                    |
| - write the new modified frame into memory |                                    |
| total number of transactions               | 2                                  |
|  |                                    |
| Burst time:                                | 0.000013 s                         |
| Allowed time:                              | 0.000032 s                         |

Although it seems that the bus has enough bandwidth to handle these two transactions, in practice it was not so. A word cannot be written on every clock cycle of the bus, because the buffers are not always ready. Moreover it takes several cycles to setup a burst write / read.. As a solution, we created two separate blocks for reading (Mem\_t\_IP) and processing writing (IP\_to\_mem). And each block has its own PLB bus connecting to the memory. Not only did this give us more bandwidth, it also made our project easier to understand and debug. See section 3.7 and 3.8 for more detailed description of the two blocks.

### 3.3 IIC\_interface and its configuration setting (ddr\_test.c)

---

The IIC module (called xps\_iic\_0 in the edk IP catalog) is a PLB IIC Bus interface that can control many applications depending on its configuration settings. Our system used the IIC module to configure the Digilent VDEC1 Video Decoder board. Because the IIC IP core is a slave, it requires configuration instructions sent from the MicroBlaze in order to operate the VDEC1. That's why we have added a ddr\_test.c file to our system that does the configuration.

We learned how to use this module by looking at the 'video\_capture\_rev\_1\_1.zip' example that was posted on the Digilent website (<http://www.digilentinc.com/Products/Detail.cfm?NavTop=2&NavSub=453&Prod=XUPV2P>). We cannot simply copy all the connections and the configuration code from this example project due to a number of reasons. See table below.

| Video_capture_1 example                       | What we need in our system  |
|---|---|
| • It uses a PPC processor                     | • We want to use the MicroBlaze processor   |
| • It uses OPB bus                             | • PLB bus instead of OPB  |
| • It uses the I2C module (outdated for v10.1) | • Because we are using the edk v10.1, it has the IIC interface instead of the I2C |

These modifications can be fixed by replacing the PPC, OPB and I2C in the video\_capture to MicroBlaze, PLB and IIC, and also a few modifications in the ddr\_test.c to configure the new IIC interface. (see ddr\_test.c for more detail)

Also, the example from 'video\_capture\_rev\_1\_1.zip' sends the video stream straight from the IIC interface to the VGA display, where in our system, we need to intercept that video stream from the IIC and be able to write each frame to memory. The interception of the video is done within our IP\_to\_mem custom module, so see section 3.6 and 3.7.1 for more detail.

### 3.4 TFT controller (xps\_tft\_0)

---

The XPS TFT controller is also an IP core from the EDK library. It connects to the PLB V4.6 bus and act as a PLB master. Its job is to read the video pixel data from the PLB attached video memory, and in our case, it starts from memory address 0x90000000. It handles a clock domain crossing and feeds the RGB data to a VGA monitor running at 27MHz.

### 3.5 MPMC & DDR Memory

---

MPMC v4.03 (multi-port memory controller) is new to the Xilinx IP library starting in version 10.1. It allows multiple buses to be connected to the same piece of memory through different ports, and the ports are able to access the memory in parallel of each other. From the block diagram and section 3.2, you can see that we opened up 4 out the 8 ports supported by MPMC. This feature was crucial to the success of our design since we found bottlenecks in transferring frames to and from memory which was solved by opening more ports and using additional buses.

### 3.6 Decoder Logic

---

The logic used for the decoder was mostly carried over from the videoCapture module and integrated into the IP\_to\_mem module. The components that were included for decoding were If\_decode.v, line\_buffer.v, neg\_edge\_detect.v, pipe\_line\_delay.v, special\_svga\_timing\_generation.v, svga\_defines.v, vp422\_444\_dup.v, ycrb2rgb.v.

These modules performed the task of converting the data from composite format to RGB and also kept track of the VSync and HSync to determine new frame and new line.

#### 3.7.1 IP\_to\_Mem Custom Logic

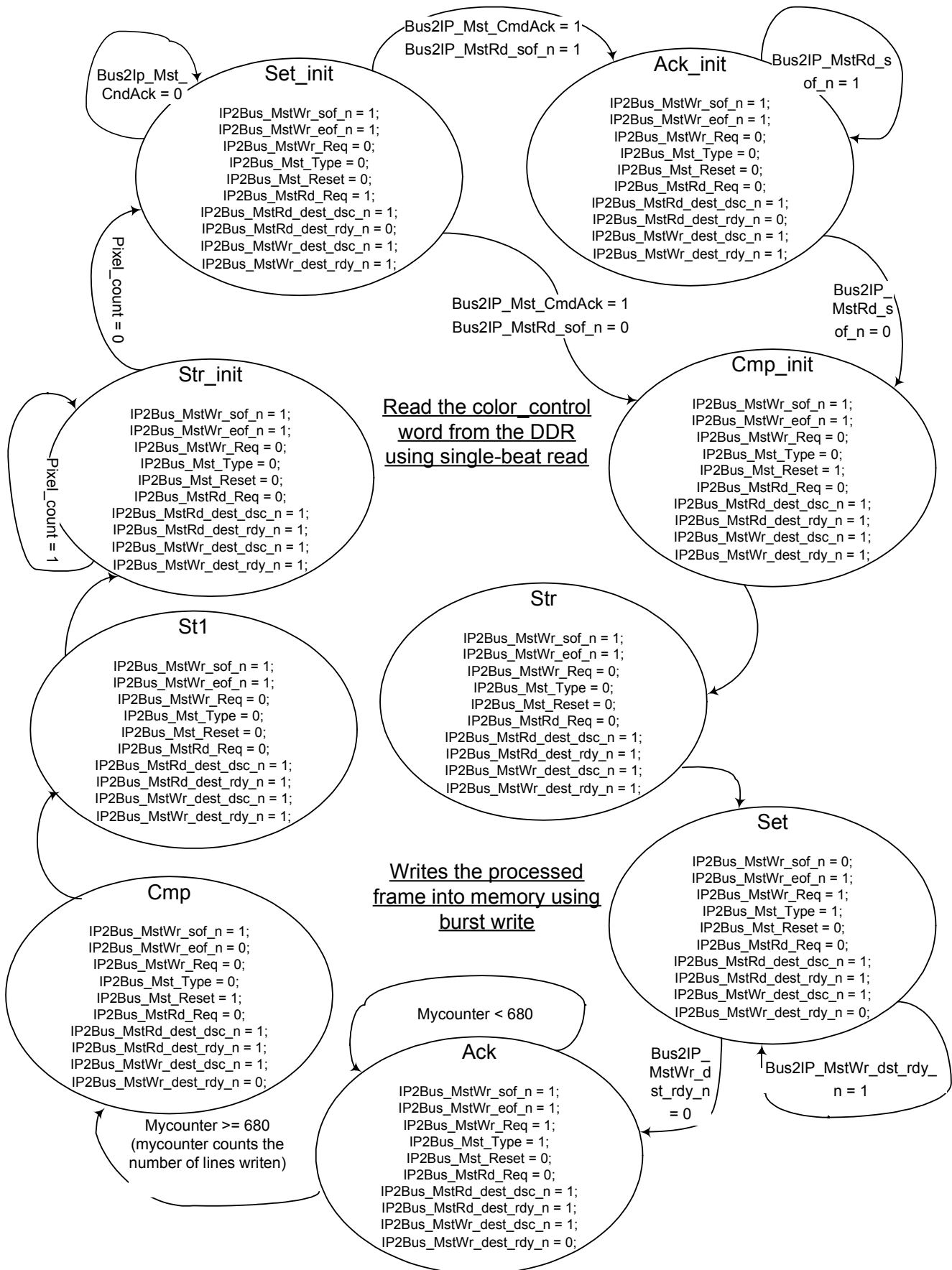
---

IP\_to\_mem operates as a finite state machine that saves a static frame in one memory region and then proceeds to save the live stream frames in another memory region. Please refer to the state diagram below for visualization of the process. The timing diagrams for the read and write operations are attached in the appendix for reference.

Note: most signals were active low with the exception of the requests which were active high.

| States  | Description  |
|---|--|
| <b>Str_init, Set_init, Ack_init, Cmp_init</b> | These states are responsible for reading the configuration word.   |
| Str_init                                      | This is the first state upon reset. It waits until pixel_count=0 to synchronize the finite state machine with the input for the write operation that takes place in later states. Bus address is set to the memory location of the configuration word. |
| Set_init                                      | Prepares for single beat read:<br>1.set master type to single beat (IP2Bus_Mst_Type = 0)<br>2. Raise master read request   |

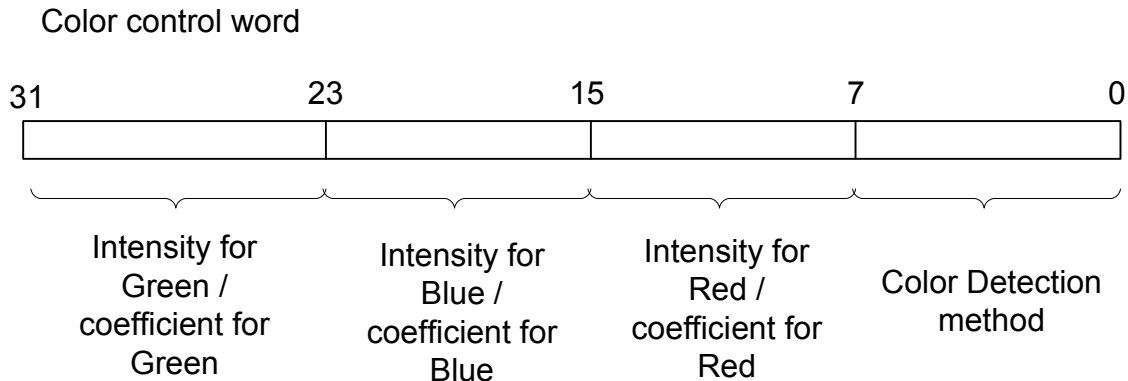
|                            |  |
|----------------------------|--|
|                            | 3.Indicate that the master is ready to read (IP2Bus_MstRd_dst_rdy_n= 0)  |
| Ack_init                   | Our FSM then waits of a Bus2IP_Mst_rd_sof_n signal which signifies that the data is ready to be read.  |
| Cmp_init                   | Signifies that the read has been completed. Transitions to the write operation (str state)   |
| <b>Str,set,ack,cmp,st1</b> | These states are responsible for the burst write.<br>Note: the switching from static frame memory region to live frame memory region is handled by a separate piece of logic but the write operation is identical.                               |
| Str                        | Intermediate state before the burst write operation.<br>Bus address switches to that of the live stream memory region.   |
| Set                        | Prepares for burst write:<br>1.set write type to burst (IP2Bus_Mst_Type = 1)<br>2. Raise master write request<br>3. indicate that data is ready (IP2Bus_MstWr_src_rdy_n = 0)<br>4.Indicate beginning of burst write (REG_IP2Bus_MstWr_sof_n = 0) |
| Ack                        | Transition to ack occurs when the bus responds with an acknowledge and destination ready.<br>The state returns to ack for 680 writes (MyCounter counts the number of writes)   |
| Cmp                        | The burst write has completed.<br>1. Indicate end of burst write (REG_IP2Bus_MstWr_eof_n = 0)  |
| St1                        | Intermediate state before returning to configuration word read operation.  |



### 3.7.2 Color Detection Algorithm

---

From the block diagram you will notice that the IP\_to\_Mem module also read a color control word from the memory. This 32 bit word contains the color detection parameters and method used in our detection algorithm.



Our system is capable of detect 5 major types of colors: red, green, blue, white and black. We implanted two different methods for detecting our target color.

#### Method 1 - Specifying Threshold:

- This method is more effective when you want to detect a very dark (black) or a very bright (white) color.
- To detect a dark color you have to compare the pixel data from the line buffer with intensity threshold value stored in the color control word. If all three of the color signal (R, G , B) are greater than the required threshold values, then we say the condition is satisfied and that pixel will be replaced.
- To detect a light color, you do the same thing, expect you want the RGB signal to be less than the required threshold value.
- You can simply adjust how bright or how dark of a color that you want to replace by adjusting the intensity value in the color control word.

$G > G_{\text{threshold intensity}}, B > B_{\text{threshold intensity}}, R > R_{\text{threshold intensity}}$  ← to detect a dark color

$G < G_{\text{threshold intensity}}, B < B_{\text{threshold intensity}}, R < R_{\text{threshold intensity}}$  ← to detect a bright color

#### Method 2 - Specifying coefficients:

- This method is more effective when you want to detect Green, Red or Blue.
- In this method, you compare the intensity of that color that you want to detect with the sum of the other two colors.

- For example, if you want to detect green, all you have to do is make sure that the Green signal is more intensified than the sum of the Red and Blue signals.
- We also multiplied the signal to an adjustable coefficient value for better result.
- The coefficient value can also be easily adjusted from the color control word.

$$G_{\text{coef}} * G > B_{\text{coef}} * B + R_{\text{coef}} * R \leftarrow \text{to detect Green}$$

$$R_{\text{coef}} * R > B_{\text{coef}} * B + G_{\text{coef}} * G \leftarrow \text{to detect Red}$$

$$B_{\text{coef}} * B > G_{\text{coef}} * G + R_{\text{coef}} * R \leftarrow \text{to detect Blue}$$

As you can see, there are a total of 5 different sets of comparisons that we can do in our color detection logic (two for the specifying threshold method and three for the specifying coefficients method). The method that our color replacement logic decides to use is determined by bit 0 to 7 of the color control word.

### 3.8 Mem\_to\_IP Custom Logic

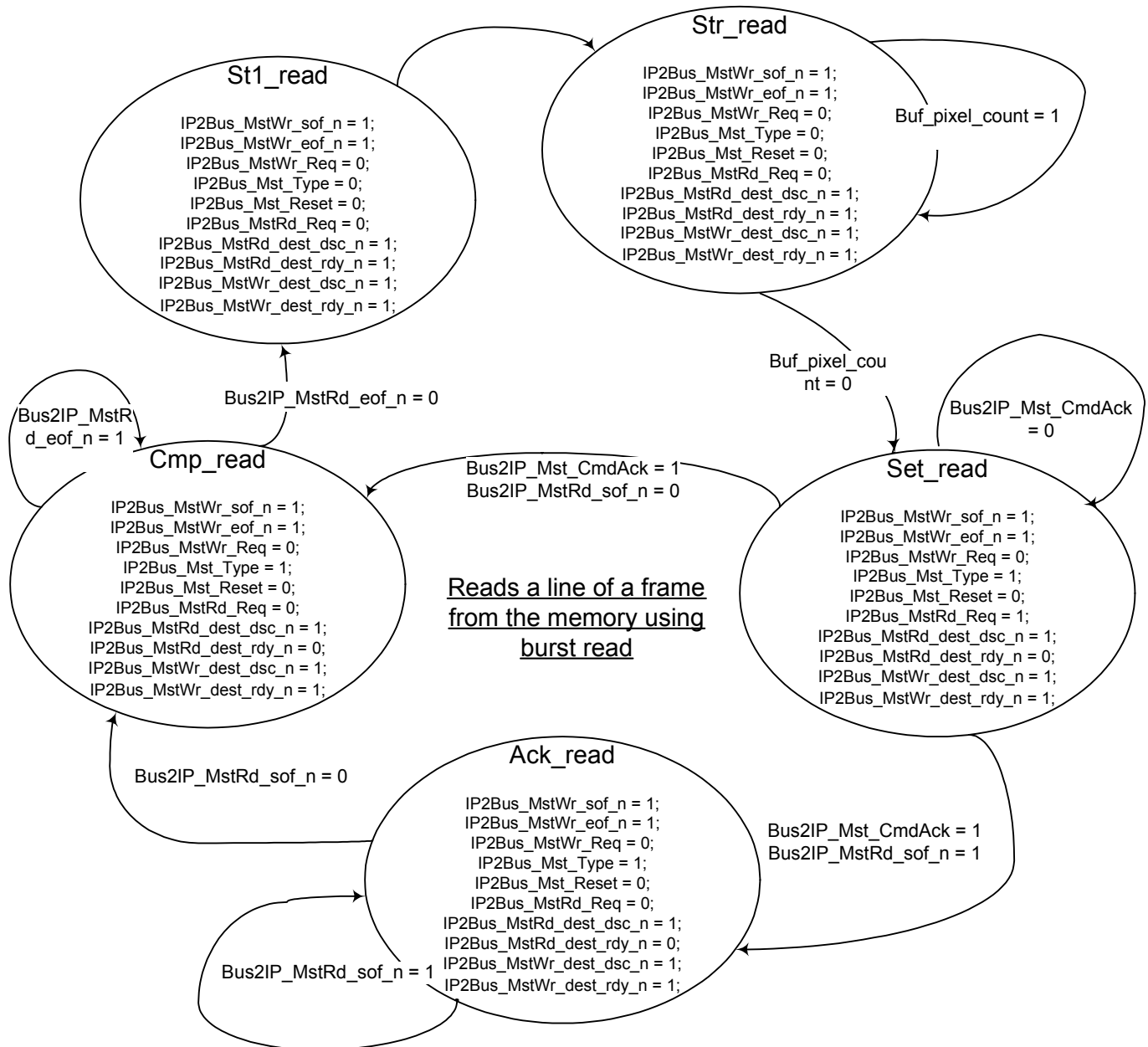
---

Mem\_to\_IP is responsible for fetching lines from the static memory region and making it available to IP\_to\_mem to perform colour replacements. The finite state machine used for burst reading from memory is as illustrated below. Timing diagrams are included as an appendix for reference.

| States   | Description  |
|----------|--|
| Str_Read | This is the first state upon reset. It waits until Buf_pixel_count=0 to synchronize the finite state machine with the IP_to_mem module.  |
| Set_Read | Prepares for burst read:<br>1.Set master type to bursting (IP2Bus_Mst_Type = 1)<br>2. Raise master read request<br>3.Indicate that the master is ready to read (IP2Bus_MstRd_dst_rdy_n= 0)         |
| Ack_Read | Transition to ack_read occurs when the bus responds with an acknowledge. In this state, we can drop the read request. It waits for Bus2IP_MstRd_sof_n to be asserted to begin reading in cmp_read. |
| Cmp_Read | This state reads one word per cycle (while (Bus2IP_MstRd_src_rdy_n= 0) and continues until   |



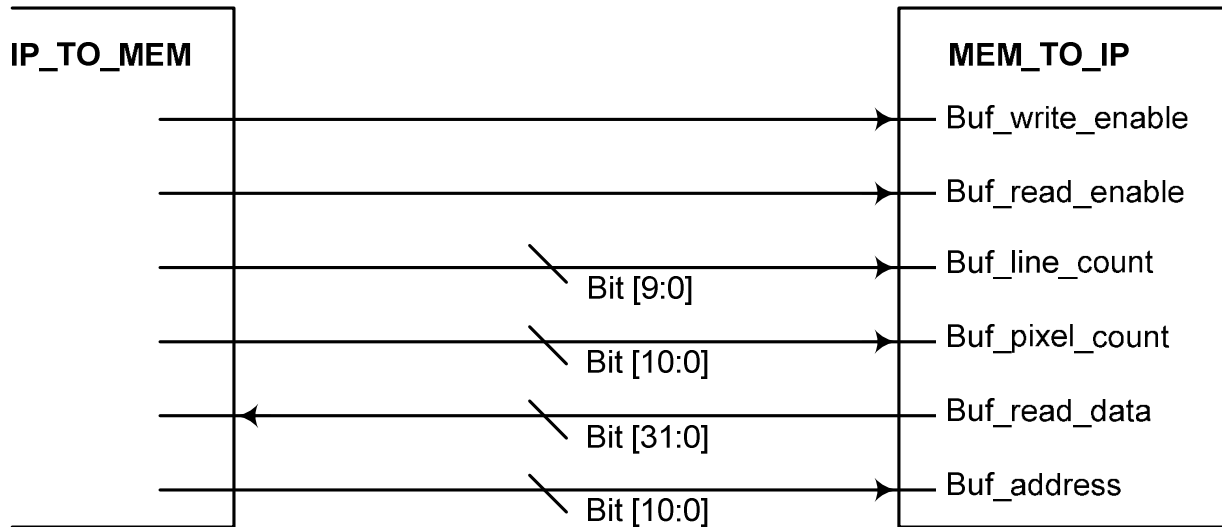
|     |  |
|-----|--|
|     | Bus2IP_MstRd_eof_n to be asserted.   |
| St1 | Intermediate state before transitioning back to str_read to begin reading again. |



### 3.9 Connection between IP\_to\_Mem and Mem\_to\_IP

---

The two custom modules are connected using internal connections inside the EDK system builder.



The ip\_to\_mem controls the mem\_to\_ip block through the signals shown in the above figure. Buf\_pixel\_count and Buf\_line\_count are the video input synchronization signals sent from the synch modules in the ip\_to\_mem block. These signals tell the mem\_to\_ip block which line and pixel is currently being streamed from the input. They are used to decide which line is currently being read from memory by mem\_to\_ip.

The Buf\_read\_enable and Buf\_write\_enable signals are used to decide which of the two line buffer in mem\_to\_ip is being written to, and which one is being read from. Since the buffer alternation is the same as in the ip\_to\_mem block, there was no reason to recreate this logic in mem\_to\_ip.

The Buf\_address signal is used to tell mem\_to\_ip which pixel to read from one of the buffers. When reading, it is always true that one line buffer is filled up with valid data. It is thus up to the ip\_to\_mem block to request which pixel it wants to read from this buffer. When it requests the pixel, the data corresponding to that pixel is placed on the Buf\_read\_data line.

## DESCRIPTION OF OUR DESIGN TREE

The following table illustrates the content of every folder that we have submitted regarding to our project. A README file has also been included at the top level of our directory for the same purpose.

| Folder Name                  | Description   |
|------------------------------|---|
| <b>System</b>                | <p>This contains the XPS project.</p> <p>The code for our custom pcore and its data files can be found in the \pcore dir</p> <p><u>key files include:</u></p> <ol style="list-style-type: none"><li>1. user_logic.v (for ip_to_mem) --&gt; \System\pcores\ip_to_mem_v1_00_a\hdl\verilog\user_logic.v</li><li>2. user_logic.v (for mem_to_ip) --&gt; \System\pcores\mem_to_ip_v1_00_a\hdl\verilog\user_logic.v</li><li>3. ddr_test.c --&gt; \System\DDR_in_10_1_02_v1\code\ddr_test.c</li></ol> <p>Both user_logic.v is the main code for our custom hardware, and they have been neatly commented for further review.</p> |
| <b>Doc</b>                   | <p>This includes all the documentation file for our project:</p> <p>Project Proposal: Project_proposal.doc</p> <p>Power point of our Presentation: Final_Presentation.pptx</p> <p>Group Report: Group_report.pdf</p> <p>Individual Report: Individual_report.pdf</p>  |
| <b>Demo_video</b>            | <p>A video showing our working project</p>  |
| <b>video_capture_rev_1_1</b> | <p>This is a copy of the example project that we metioned in section 3.3 of our final report. it is obtained from <a href="http://www.digilentinc.com/Products/Detail.cfm?NavTop=2&amp;NavSub=453&amp;Prod=XUPV2P">http://www.digilentinc.com/Products/Detail.cfm?NavTop=2&amp;NavSub=453&amp;Prod=XUPV2P</a></p>   |

## REFERENCES

1. 2005 Past Projects (Interactive Video Game),  
<<http://www.eecg.toronto.edu/~pc/courses/532/2009/pastprojectsnode2.html>>
2. Advanced Digital Systems Design (DDR memory reference), ENSC 452/894,  
<<http://www.ensc.sfu.ca/~lshannon/courses/ensc452/labinfo.html>>
3. How Chroma Keying Works, Schnarr, Bill, Sign Video Ltd.  
<<http://www.signvideo.com/chrom-ky-wks.htm>>
4. Question regarding the TFT – VGA Controller and MPMC, Xilinx User Forum,  
<<http://forums.xilinx.com/xlnx/board/message?board.id=EDK&thread.id=7339>>
5. VideoCapture (With Vdec 1), Digilent Inc,  
<<http://www.digilentinc.com/Products/Detail.cfm?Prod=XUPV2P>>
6. Video Capture example on XC2VP30 (XUP), Xilinx User Forum,  
<<http://forums.xilinx.com/xlnx/board/message?board.id=EDK&thread.id=7375>>

### 5.1PLB Burst Write Timing Diagram





