431 - Introduction to VLSI CAD
Video acquisition and display
Department of Electrical and Computer Engineering, McGill University
October 10, 2013
Montréal, Québec, Canada

1 Introduction

In this project, the sections designed for VGA and NIOS projects are integrated along with a DMA engine, a grab interface and a memory controller. The complete system is shown in Figure 1. The source code of the grab interface section is provided, as well as the SDRAM memory model which is needed to simulate the entire system. The memory controller is taken from the Qsys component library.

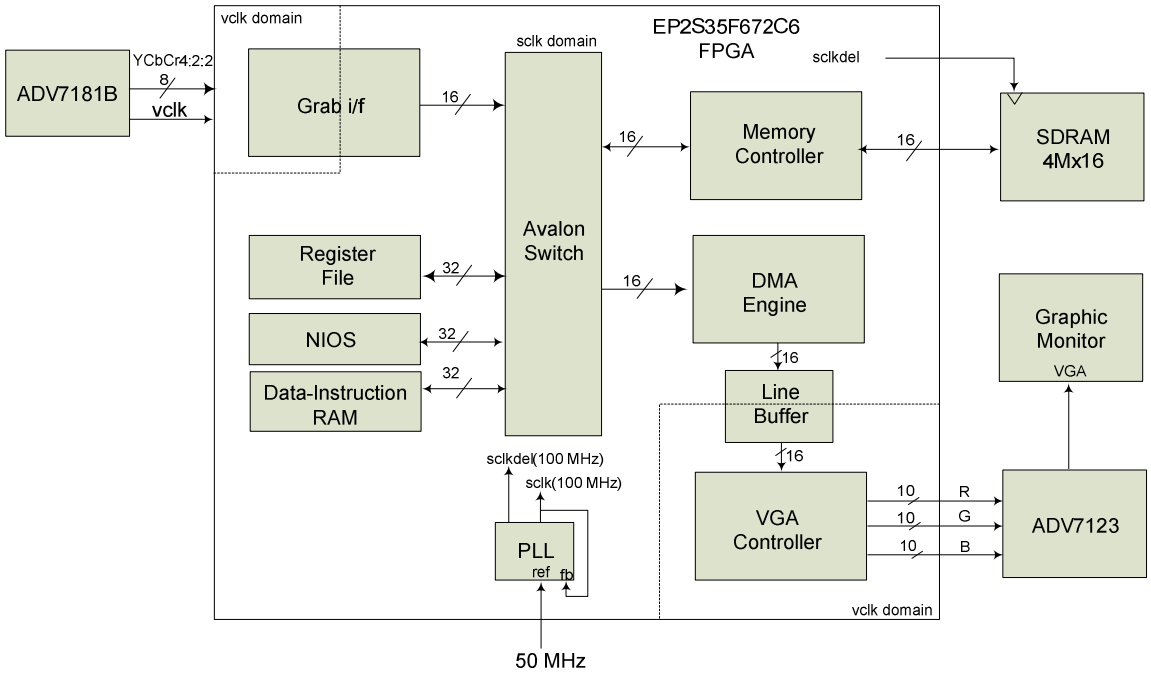


Figure 1. Video acquisition FPGA system block diagram.

The FPGA is used to display images coming from a standard NTSC signal through the video decoder on a graphic monitor. The image is temporarily stored into memory before being displayed. The grab interface writes the image into memory, the NIOS processes it, and the DMA engine reads it and feeds the VGA controller that displays it. The NIOS also executes the driver code that sequences the acquisition *field by field*. The Qsys system operates at 100MHz. Each step is reviewed in details in the following paragraphs.

2 NIOS

2.1 Acquisition

The NIOS is responsible to control the sequence of image acquisition through appropriate register programming. It controls the acquisition field by field by setting the appropriate registers for each of the fields being grabbed (ex: setting a different GFSTART register for each new field). The NIOS responds to interrupts generated by the grab interface and schedule the acquisition of the next field to be output by the video decoder. According to the design and functionality of the grab interface provided, it is suggested that you generate an End-Of-Frame (EOF) interrupt on the falling edge of GACTIVE (refer to Figure 3). A progressive frame (2 fields) is stored into memory before it is processed or displayed. In order to prevent

artifacts between frames being displayed and the ones being acquired or processed, a "multi-buffering" mechanism is used such as the frames being acquired or processed do not interfere with the frame being displayed (Figure 2). Once a new frame has been written to memory, the NIOS can perform an operation on it, but then needs to wait until the processing is completed before it can switch the source of the frame that is to be displayed to this new processed image. During that time (unless you are able to perform the processing pass in real-time!), the NIOS stops the acquisition of new fields otherwise it would overwrite the frame being displayed, or processed. When the NIOS does not do processing on incoming frames, it must be able to display all the frames in real-time.

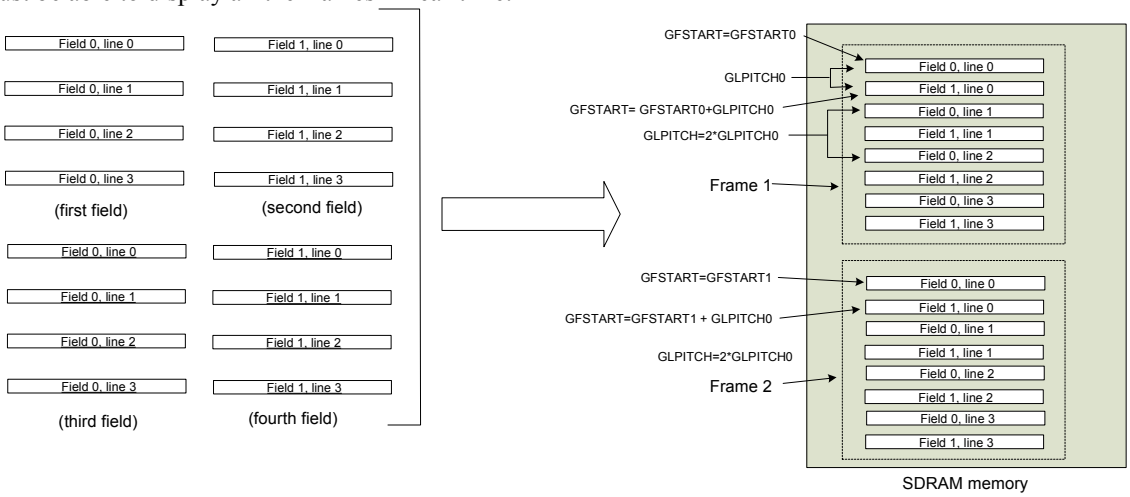
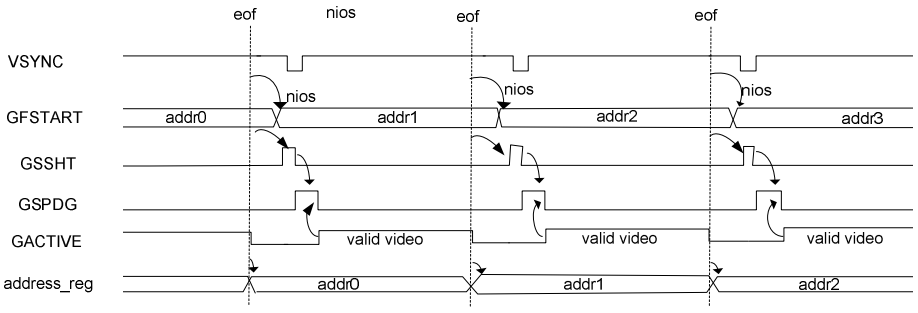


Figure 2. Multi(double)-buffering of frames into SDRAM memory.

The sequence of operations performs by the NIOS for displaying images (this can be extent to include a processing pass…) can look like the following:

a) Program initial register setting for first field (GFSTART, GLPITCH, GCTL.GMODE)
b) Enable interrupts on end-of-frame[1] (GINT.EOFIEN)
c) Arm acquisition (write '1' to GCTL.GSSHT (odd or even field depending on GCTL.GMODE))
d) Wait for EOF interrupt
e) Program memory addressing register for second field (GFSTART)
f) Arm acquisition (write 1 to GCTL.GSSHT, GCTL.GMODE)
g) Wait for end-of-field interrupt
h) Repeat c) to g) to grab the second frame positioned at another location into memory
i) Repeat c) to h)



[1] The term "frame" in "end-of-frame" is used as a general term. Depending on GMODE, and end-of-frame event actually occurs on every field.

Figure 3. Acquisition sequence (main registers and events).

The above sequence is related to the *acquisition of the images*, but the NIOS also has another role related to the *data being displayed*. The NIOS has to inform the DMA engine where to read the image from. It needs to insert write accesses to the DFSTART register to indicate where a stable image can be read by the DMA engine (refer to the DMA read engine section).

Validation note

The NIOS also has accessed to memory. This allows for the NIOS to initialize the SDRAM with an image that can be read by the DMA engine and displayed by VGA controller. This means that the hardware validation of DMA and VGA controller functionality can be done without being connected to a camera (once everything is working in simulation!).

2.2 Processing

Once a frame is stored into memory, the processor can perform many types of operation on the pixels, and could also add information to the image before it is displayed. Some manipulations can be done in real time, (i.e. without missing a frame), while other manipulations may take more than one frame to complete, The NIOS needs to be programmed to perform the following two operations (switches on the board should be used to individually enable each functionality).

2.2.1 Frame counter and frame rate

The NIOS keeps track of the amount of **distinct** frames that have been displayed during the last second (i.e. frame rate) and add this information to the image (2 decimal digits). It also keeps track of the total amount of distinct frames displayed since the beginning of the acquisition (4 digit is fine… enough for 5 minutes at 30 frames per second before it wraps around). Each character is displayed by overwriting a block of pixels with the font you have created The fonts can be created in real time by a NIOS function, or that information can be stored into SDRAM memory and retrieved according to the values you need to display. The frame count is displayed in the upper right corner of the screen, while the frame rate is displayed in the right low corner of the image.

2.2.2 Convolution

When this functionality is enabled, each progressive frame stored into SDRAM is processed by the NIOS processor. An interesting visual application for a convolution is with edge detection. Edge detection can be performed using 3x3 Sobel kernels. The Sobel kernels computes the gradient in both X (G_x)and Y(G_y) directions. The resulting gradient can be approximated by adding the absolute value of $|G_x| + |G_y|$. Please consult the web as there are many references on the Sobel convolution kernels. You may want to include a discussion in your report explaining the differences in the operating speed comparing whether the Sobel operation is performed with full accuracy or approximate (what would be the visual difference?). Also, if your algorithm uses 9 multiply between the kernel coefficients and the pixel values, you can exercise yourself with many difference kernels. The operations that are performed by the NIOS consist of reading the entire image, performing the multiplications and/or additions for each kernel and storing the result back into memory. This cannot be performed in real time by the soft processor, and you would notice a substantial degradation of the frame rate. Code would be more efficient making use of a data cache such as the NIOS use efficiently the pixels that are fetched from SDRAM, as the same pixel is used into three different output pixel computations. Also note that the resulting image decrease by two lines vertically and two pixels horizontally, therefore, something as to be done for the pixel values at the periphery of the image (overscan), the easiest would be to use “transparent overscan” (do nothing) or a constant value.

3 Grab interface

The grab interface provides a bridge between the video decoder and the Avalon switch fabric. The function of the grab interface is to write valid data into the “grab buffer”, and then empty the buffer by initiating write requests toward the memory controller while acting as an Avalon memory-mapped master interface. The grab buffer is separated into two sections that can simultaneously write and read data. When the write controller is writing into one half of the buffer, the read controller is reading the other half which has just

been written with valid data. The grab interface also includes a block of logic that is responsible to compute the SDRAM memory address at which the data is to be written to, as shown in Figure 4. The bandwidth toward SDRAM memory is almost 10 times (200 MB/s) the acquisition bandwidth (20.7 MB/s).

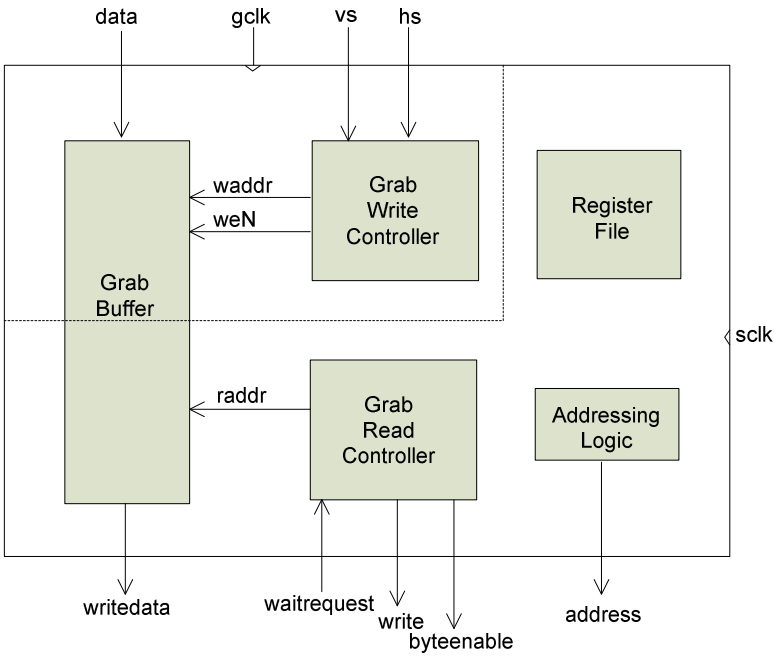


Figure 4. Grab interface block diagram.

The grab interface being the most complicated part of the system (beside the memory controller), a functional grab interface block of logic is provided (in the form of a VHDL source code). You have to create an Avalon component from the grab interface top level that is provided. The registers described in the register section associated with the grab interface control which fields are grabbed and where images are stored into memory. To improve bandwidth performance toward memory, a certain amount of data needs to be buffered (64, 128 or 256 bytes for instance) before the request is sent to the memory controller (that is the role of the grab buffer). Burst of 128 bytes (tbc) burst are implemented by the grab interface block of logic provided. Note that most of the registers can be hard coded for the purpose of simulations when the NIOS is not present. Registers would eventually be controlled by the NIOS processor. During the acquisition of fields, the grab start address has to be changed from field to field. One reason is that we are building a progressive frame into memory. For instance, if GFSTART is the start address of the first field to be grabbed, considering that the distance between each line (of the interlaced fields) into memory is GLPITCH, then the grab frame start address of the second field would have to be GFSTART + GLPITCH/2 in order to create a progressive frame into memory. It is this complete frame that is going to be read by the DMA engine or processed by the NIOS processor

Hardware designers are often required to correct/modify/optimize already existing codes. This exercise is done through the support of horizontal and vertical sub sampling for the grab interface. Sub sampling should occur at the **source** of the data, so we do not waste bandwidth writing/reading to/from memory. One pixel out of two, four or eight could be stored into SDRAM memory. The same principle applies for lines. Also, the GFMT bit of GCTRL should enable grabbing in both monochrome and color. Sub sampling and monochrome implementations require minor modifications to the grab interface, but provide major improvement in bandwidth usage. Combining horizontal and vertical sub sampling to monochrome storage would substantially increase the frame rate of the edge detection algorithm.

4 DMA engine

The DMA engine is an Avalon master read controller that writes the incoming data into a line buffer. The Avalon master controller implements the proper handshake with the Avalon switch and provides to the memory controller the correct memory locations where pixels are read. The incoming valid data is routed to the line buffer write data port along with line buffer's addresses and control to perform the writes at the right locations. When enabled, the DMA engine is controlled by the VGA controller that indicates to the DMA engine when it requires new lines to be fetched. The VGA controller could be emulated using at least two signals, SOF (start-of-frame) and SOL (start-of-line), each of those indicating that a new line has to be fetched. Note that SOF and SOL signal are provided as a guideline only, and depending on your implementation, you may required different and additional signals to achieve proper handshake between VGA controller and DMA engine. To achieve minimum memory access efficiency, the DMA engine must be able to initiate burst accesses of at least 64, 128 or 256 bytes at a time. The memory read latency is not fixed because arbitration on the memory bus is not guaranteed (grab interface may be writing, the SDRAM memory may be refreshed or the NIOS might be reading/writing to memory), so a buffer being able to hold two lines will quite eliminate the chances of artifacts caused for instance by the VGA controller catching up on the DMA line being read. The DMA buffer should always have one line in advance with regard to the VGA controller.

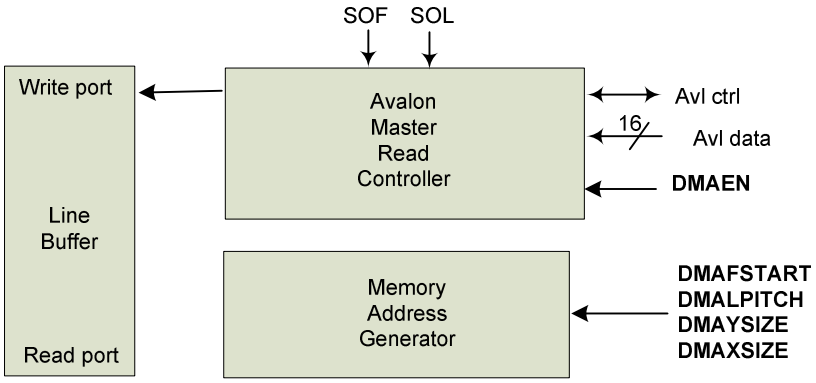


Figure 5. DMA engine block diagram.

During the acquisition sequence, the DMAFSTART register has to be modified by either the NIOS or through a hardware mechanism in order to guarantee that no display artifacts will be present on the monitor, therefore making sure that the DMA engine is not fetching a line/frame that is being modified by the grab interface or the NIOS. Refer to the register file section associated with the DMA engine to complete the description on its functionality. Note that one of the DMA control field allows reading a line in reverse order, therefore flipping the image horizontally. Line reversal is a standard feature often use when grabbing with a camera that has taps in different directions, or when the camera cannot be physically installed on its mounting bracket such as to provide an image in the usual direction.

5 VGA controller

The VGA controller was slaved to synchronization signals coming from the camera in part 1, but it now needs to generate its own synchronization signals. Writing the images to SDRAM memory allows removing the synchronization requirement with the video decoder. The DMA engine is slaved to the VGA controller, and indicates to the DMA engine when it can fetch new lines. Pay a particular attention to signals that are exchanged between the VGA controller and the DMA engine. They go through a clock domain crossing. VGA controller must have the ability to vertically and horizontally zoom the data.

The VGA controller can be completely isolated from the acquisition, allowing increasing the refresh rate of the display. It is important that the data being read by the DMA engine are stable. One way to guarantee an artifact free display is to create at least two frame buffers into memory, one where the acquired frame is written, and the other one where the previous frame is being read. This has the disadvantage of inserting a latency of one frame (compare to a latency of one line in part 1), which can be acceptable depending on the application. GFSTART and DMAFSTART are modified to switch between those (at least) two frame buffers. To more easily avoid displaying artifacts on the graphic monitor, the VGA controller can operate on the same clock as the acquisition clock coming from the video decoder. If that is done, it is therefore a good practice to synchronize the vsync of the VGA controller with the vsync of the video decoder.

Other register fields added the ability for the VGA controller to implement horizontal and vertical zoom. Horizontal zoom is achieved by providing the same pixel for two, four or eight clock cycles, while vertical zoom is achieved by reading the same lines two, four or eight times. Those are basic versions of hardware sub sampling and zoom functions as we could achieve more precise operation when the pixel's neighborhood is being considered in the operation. Complexity could be increased simply by also supporting odd zoom and fractional factors.

6 Register description

The register file is centralized and contains registers associated to all main modules. There is a gap of 256 bytes between each section, in order to facilitate the addition of further registers in different sections if required. It is suggested that all registers associated with a counter value be zero-based (a value of zero meaning one, for instance VSSYNC =0 means VSYNC activated on the first line).

6.1 Grab interface

GCTRL address 0x00 Grab ConTRoL

15	14	13	12	11	10	9	8
Reserved				GYSS(1:0)		GXSS(1)	
7	6	5	4	3	2	1	0
Reserved	GMODE		GFMT	GACTIVE	reserved	GSPDG	GSSHT

GYSS RW(1:0)	Grab Y SubSampling It indicates the factor to which lines are being subsampled. *Note: The grab interface code provided does not currently support vertical subsampling.*

Value	
0b00	No vertical subsampling
0b01	Vertical subsampling factor of 2
0b11	Vertical subsampling factor of 4
0b11	Vertical subsampling factor of 8

GXSS RW(1:0)	Grab X SubSampling It indicates the factor to which pixels are being subsampled. *Note: The grab interface code provided does not currently support horizontal subsampling.*

Value	
0b00	No horizontal subsampling
0b01	Horizontal subsampling factor of 2
0b11	Horizontal subsampling factor of 4
0b11	Horizontal subsampling factor of 8

GMODE RW	Grab MODE
	This bit indicates to the grab interface the conditions under which fields are being acquired. The next occurrence of a vertical synchronization signal when the selected condition is met, paired with an active GSPDG, will enable the image data to be stored by the grab interface and forwarded to memory. *NOTE: The grab interface code provided does not support all the modes. Only GMODE='10" and GMODE="11" were tested and verified*.

Value	Description
00b	Grab next two fields, starting with odd field.
01b	Grab next two fields, starting with even field
10	Grab next odd field
11	Grab next even field

GFMT RW	Grab ForMaT
	This bits sets the grab interface in monochrome (Y) or in color (YcrCb 4:2:2) mode format. *Note: The grab interface code provided does not currently support monochrome format.*

Value	Description
0b	Monochrome
1b	Color

GACTIVE RO	Grab ACTIVE
	This bit is set to one to indicate that a frame is being grabbed. It is set at the start-of-frame event if GSPDG is set. It is reset at the end-of-frame event. *Note: the grab interface provide is designed such as the sof event is the start of valid video while the eof event is the end of the valid video.*

Value	Description
0b	Grab is active
1b	Grab is inactive

GSPDG RO	Grab Snapshot PenDinG
	This bit indicates the status of the grab pending signal. This signal is activated between snapshot assertion and the start-of-frame event.

Value	Description
0b	No snapshot is pending
1b	A snapshot is pending

GSSHT WO	Grab SnapSHot
	This bit is used to enable the grab of 1 (one) field or frame (depending on the value of GMODE) . This bit resets to zero after being written to one. It sets the grab pending signal, which will allow grabbing of one frame on the next start-of-frame event.

Value	Description
0b	No effect.
1b	Set snapshot pending

GFSTART address 0x04 Grab Frame START address

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved	GFSTART (22:16)						
15	14	13	12	11	10	9	8
GFSTART (15:8)							
7	6	5	4	3	2	1	0
GFSTART (7:0)							

GFSTART RW(22:1) RO(0)	Grab Frame START address This register indicates the byte start address of the image to be stored into memory. The LSB should be ignored and read as zero.

Value	Any 23-bit value.

GLPITCH address 0x08 Grab Line PITCH

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved	GLPITCH(22:16)						
15	14	13	12	11	10	9	8
GLPITCH(15:8)							
7	6	5	4	3	2	1	0
GLPITCH(7:0)							

GLPITCH RW(22:1) RO(0)	Grab Line Pitch This register indicates the amount of bytes between two consecutive lines into memory. The LSB should be ignored and read as zero. Note that a negative line pitch could be used and therefore, 23-bit arithmetic has to be used by the addressing logic.

Value	Any 23-bit value.

GINT address 0x0C Grab INTerrupt

15	14	13	12	11	10	9	8
Reserved							
7	6	5	4	3	2	1	0
Reserved				EOFISTS	EOFIEN	SOFISTS	SOFIEN

SOFTIEN RW	Start-Of-Frame Interrupt ENable
	This bit enables or disables detection of the start-of-frame event. The term “frame” also refers to a “field”, according to the grab mode of acquisition selected. ***Note**: sof interrupt usage may be omitted in your implementation, providing that you can program the next field before the gactive comes back active again in the next field you need to program (which should be the case).*

Value	Description
0b	Start-of-frame interrupt is disabled.
1b	Start-of-frame interrupt is enabled.

SOFISTS RW2C	Start-Of-Frame Interrupt StaTuS
	This field indicates the status of the start-of-frame interrupt. . Writing 1 clears this bit if it is set. Writing 0 has not effect.

Value	Description
0b	Start-of-frame interrupt is inactive
1b	Start-of-frame interrupt is active

EOFIEN RW	End-Of-Frame Interrupt ENable
	This bit enables or disables detection of the end-of-frame event. The term “frame” also refers to a “field”, according to the grab mode of acquisition selected.

Value	Description
0b	End-of-frame interrupt is disabled.
1b	End-of-frame interrupt is enabled.

EOFISTS RW2C	End-Of-Frame Interrupt StaTuS
	This field indicates the status of the end-of-frame interrupt. . Writing 1 clears this bit if it is set. Writing 0 has not effect.

6.2 DMA engine

DMACTRL address 0x100 DMA ConTRoL

15	14	13	12	11	10	9	8
Reserved							
7	6	5	4	3	2	1	0
Reserved						DMALR	DMAEN

DMALR	DMA Line Reversal
RW	This bit controls the addressing direction of the DMA engine. When enabled, the DMA engine starts fetching pixels at the end of the line and decrements the address until beginning of line is reached.

Value	
0b	DMA line reversal is disabled
1b	DMA line reversal is enabled

DMAEN	DMA ENable
RW	This bit enables or disables the DMA engine to fetch lines from memory.

Value	
0b	DMA engine is disabled
1b	DMA engine is enabled

DMAFSTART address 0x104 DMa Frame START address

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved	DMAFSTART (22:16)						
15	14	13	12	11	10	9	8
DMAFSTART (15:8)							
7	6	5	4	3	2	1	0
DMAFSTART (7:0)							

DMAFSTART	DMA Frame START address
RW(22:1) RO(0)	It indicates the byte start address of the image to be read from memory. The LSB should be ignored and read as zero.

Value	Any 23-bit value.

DMALPITCH address 0x108 DMA Line PITCH

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved	DMALPITCH(22:16)						
15	14	13	12	11	10	9	8
DMALPITCH(15:8)							
7	6	5	4	3	2	1	0
DMALPITCH(7:0)							

DMALPITCH RW(22:1) RO(0)	DMA Line Pitch It indicates the amount of bytes between two consecutive lines into memory. The LSB should be ignored and read as zero. Note that a negative line pitch could be used and therefore, 23-bit arithmetic has to be used by the addressing logic.

Value	Any 23-bit value.

DMAXSIZE address 0x110

31	30	29	28	27	26	25	24
Reserved							
23	22	21	20	19	18	17	16
Reserved							
15	14	13	12	11	10	9	8
DMAXSIZE(15:8)							
7	6	5	4	3	2	1	0
DMAXSIZE(7:0)							

DMAXSIZE RW(15:0)	DMA X SIZE This register indicates the amount of bytes per line.

Value	Any 16-bit value.

6.3 VGA controller

NOTE: HTOTAL, HSSYNC, HESYNC, HSVALID, HEVALID, VTOTAL, VSSYNC, VESYNC, VSVALID and VEVALID can be hardcoded.

VGACTRL address 0x200 VGA ConTRoL

15	14	13	12	11	10	9	8
Reserved							
7	6	5	4	3	2	1	0
Reserved		PFMT		VGAVZOOM(1:0)		VGAHZOOM(1:0)	

PFMT RW	Pixel ForMaT This bits indicates the pixel transformation.

Value	Description
00b	Mono to RGB
01b	YCrCb to RGB
10b	YCrCb to mono to RGB
11b	Reserved

OPFMT RW	Output Pixel ForMaT This bits indicates the format for the output pixels

Value	Description
0b	Monochrome
1b	RGB (through color space converter)

VGAVZOOM(1:0) RW	VGA controller Vertical ZOOM This field controls the vertical zoom factor that is applied by the VGA controller to each valid line of the frame..

Value	
00b	No vertical zoom
01b	Vertical zoom by 2
10b	Vertical zoom by 4
11b	Reserved

VGAHZOOM(1:0) RW	VGA controller Horizontal ZOOM This field controls the horizontal zoom factor that is applied by the VGA controller to each valid pixel of the line.

Value	
00b	No horizontal zoom
01b	Horizontal zoom by 2
10b	Horizontal zoom by 4
11b	Reserved

HTOTAL address 0x204 Horizontal TOTAL

15	14	13	12	11	10	9	8
HTOTAL(15:8)							
7	6	5	4	3	2	1	0
HTOTAL(7:0)							

HTOTAL RW(15:0)	Horizontal TOTAL This register indicates the total amount of clock cycle per line (valid + blank pixels).

Value	Any 16-bit value.

HSSYNC address 0x208 Horizontal Start of SYNC

15	14	13	12	11	10	9	8
HSSYNC(15:8)							
7	6	5	4	3	2	1	0
HSSYNC(7:0)							

HSSYNC RW(15:0)	Horizontal Start SYNC This register indicates the horizontal position with regard to the horizontal counter value to which the horizontal sync is asserted.

Value	Any 16-bit value.

HESYNC address 0x20C Horizontal End of SYNC

15	14	13	12	11	10	9	8
HESYNC(15:8)							
7	6	5	4	3	2	1	0
HESYNC(7:0)							

HESYNC RW(15:0)	Horizontal End SYNC This register indicates the horizontal position with regard to the horizontal counter value at which the horizontal sync is de-asserted.

Value	Any 16-bit value.

HSVALID address 0x210 Horizontal Start VALID

15	14	13	12	11	10	9	8
HSVALID(15:8)							
7	6	5	4	3	2	1	0
HSVALID(7:0)							

HSVALID RW(15:0)	Horizontal Start VALID This register indicates the start of the horizontal valid window according to the horizontal counter value..

Value	Any 16-bit value.

HEVALID address 0x214 Horizontal End VALID

15	14	13	12	11	10	9	8
HEVALID(15:8)							
7	6	5	4	3	2	1	0
HEVALID(7:0)							

HEVALID RW(15:0)	Horizontal End VALID This register indicates the end of the horizontal valid window according to the horizontal counter value.

Value	Any 16-bit value.

VTOTAL address 0x218 Vertical TOTAL

15	14	13	12	11	10	9	8
VOTAL(15:8)							
7	6	5	4	3	2	1	0
VTOTAL(7:0)							

VTOTAL RW(15:0)	Vertical TOTAL This register indicates the amount of total lines per frame (valid + blank lines).

Value	Any 16-bit value.

VSSYNC address 0x21C Vertical Start of SYNC

15	14	13	12	11	10	9	8
VSSYNC(15:8)							
7	6	5	4	3	2	1	0
VSSYNC(7:0)							

VSSYNC RW(15:0)	Vertical Start SYNC This register indicates the vertical line position at which the vertical sync is asserted.

Value	Any 16-bit value.

VESYNC address 0x220 Vertical End of SYNC

15	14	13	12	11	10	9	8
VESYNC(15:8)							
7	6	5	4	3	2	1	0
VESYNC(7:0)							

VESYNC RW(15:0)	Vertical End SYNC This register indicates the vertical line position at which the vertical sync is de-asserted.

Value	Any 16-bit value.

VSVALID address 0x224 Vertical Start VALID

15	14	13	12	11	10	9	8
VSVALID(15:8)							
7	6	5	4	3	2	1	0
VSVALID(7:0)							

VSVALID RW(15:0)	Vertical Start VALID This register indicates the vertical line position associated with the start of the vertical valid window.

Value	Any 16-bit value.

VEVALID address 0x228 Vertical End VALID

15	14	13	12	11	10	9	8
VEVALID(15:8)							
7	6	5	4	3	2	1	0
VEVALID(7:0)							

VEVALID RW(15:0)	Vertical End VALID This register indicates the vertical line position associated with the end of the vertical valid window.

Value	Any 16-bit value.

Evaluation Criteria (listed by degree of importance)

- Functionality of the system (1- functional simulation with Modelsim and 2- actual hardware demonstration)
- Validation methodology
- Constraints and synthesis reports
- Quality of block diagrams
- Improvement proposal (design/architecture/scalability)

7 Report, simulation and synthesis

Image acquisition form NTSC camera with display to a monitor should be demonstrated, along with a functional simulation of the system with Modelsim. The design should meet the requirements as specified by the register file. The design should be able to run at 100 MHz. The constraint file along with the synthesis report summary must be given in the appendix. Key points in the demonstration/simulation should be pointed out in your demo and report. The demonstration should show sequence of image acquisition demonstrating all the features mentioned in this document. The sequence should vary color frame, monochrome frame, zoom and subsampling frames, line and frame reversal and edge detected frames.

The report is to be limited to 5 pages (single sided). Appendices are considered extra. Dual column format is recommended but not necessary. It should be clear and easy to read. A detailed block diagram must accompany all substantial piece of code. The report should include a short description of the design and any interesting or important elements encountered in the design process. Also, the interesting features of the design should be pointed out. You could include any state diagrams that indicate how your system operates as well as describe its overall operation.

Due date: Day of demo, to be held during the last week of class.