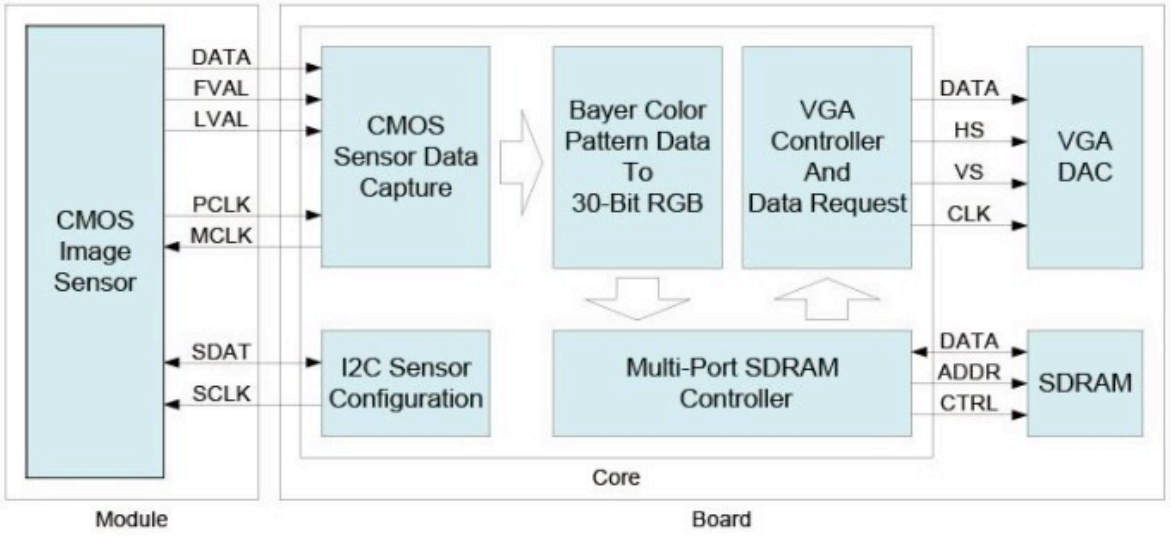
Camera to VGA Mini-Project

The purpose of this mini-project assignment is to gain experience with the DE1-SoC by developing and testing a Verilog module that interfaces with the D5M camera, SDRAM, and VGA. The module you develop will process captured images on the fly and display the processed images on the screen.

A reference design from Terasic (DE1_SoC_CAMERA.zip on Canvas) is provided that contains the D5M interface, SDRAM controller, and VGA driver. You should first download and run the reference demo to familiarize yourself with it. You will need to modify the reference design to implement the mini-project.

D5M Camera Reference Design

The reference system provides a link between the camera module and the VGA port of DE1-SoC board. The block diagram of the reference design is shown below.



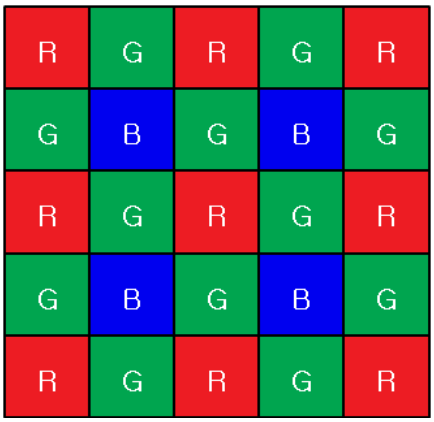
The reference system provides the following board functions:

Component	*Function Description*
KEY[0]	**Reset circuit**
KEY[1]	**Increment or decrement the exposure time (based on SW[0])**
KEY[2]	**Stop capturing images**
KEY[3]	**Resume capturing images**
SW[0]	**Off: Increase exposure time with KEY[1]** **On: Decrease exposure time with KEY[1]**
SW[9]	**On: ZOOM** **Off: normal**
HEX[7:0]	**Displays frame count since reset**

The D5M interfaces to the DE1-SoC GPIO1 port either directly or using a ribbon cable. Please be careful not to damage the boards by applying too much force when connecting the cameras. The D5M interface is as follows:

Pin	*Type*	*Description*
D5M_D	**Output**	**12 bit Data Signal**
D5M_FVAL	**Output**	**Frame Valid**
D5M_LVAL	**Output**	**Line Valid**
D5M_PIXLCLK	**Output**	**D5M PLL clock synchronized to D and *val signals**
D5M_RESET_N	**Input**	**Reset**
D5M_SCLK	**Input**	**I2C Serial Clock**
D5M_SDATA	**InOut**	**I2C Serial Data Bus**
D5M_STROBE	**Output**	**Active During Exposure**
D5M_TRIGGER	**Input**	**Triggers image capture when set to 1**
D5M_XCLKIN	**Input**	**Reference Clock**

The data, fval, and lval signals are synchronized to the PCLK output of the D5M PLL. The frame valid signal (fval) is asserted when the camera is transmitting an image frame and remains asserted until the frame is done. The line valid signal is asserted while the camera is transmitting a line of pixels within the frame. One 12-bit pixel arrives every cycle when both fval and lval are asserted. These valid signals include blanking intervals like the VGA hsync and vsync. The pixels are delivered row by row following a Bayer color pattern in which each pixel corresponds to a color as in the pattern shown.



The reference design includes a module to convert from the Bayer pattern to the RGB format used by VGA. Conversion to an RGB image format requires a process known as demosaicing where adjacent pixels in the Bayer pattern are combined into each RGB pixel. Better image quality can be achieved with more sophisticated interpolation techniques.

The number of pixels per row output by the camera is determined by registers set during camera configuration. The reference design generates 960 rows per frame, each containing 1280 pixels. By demosaicing at even rows and columns, the output image has a resolution of 640x480.

Each RGB frame is buffered in SDRAM before being read by the VGA controller. The VGA controller provides an interface to the DE1-SoC Video DAC. The VGA controller generates the sync and blank signals and synchronizes pixels to these signals according to the 640x480@60Hz VGA standard. The primary outputs to the VGA port are Red, Green, Blue, Hsync, and Vsync. The VGA standard uses sync signals to indicate active pixel transmission intervals. The monitor also uses these signals to calibrate to the proper display format. Figures 2&3 show the timing diagrams used for VGA H-Sync and V-Sync signals alongside the the active video pixel transmission intervals. For more information about the Video DAC, please reference the datasheet on the DE1-SoC system CD.

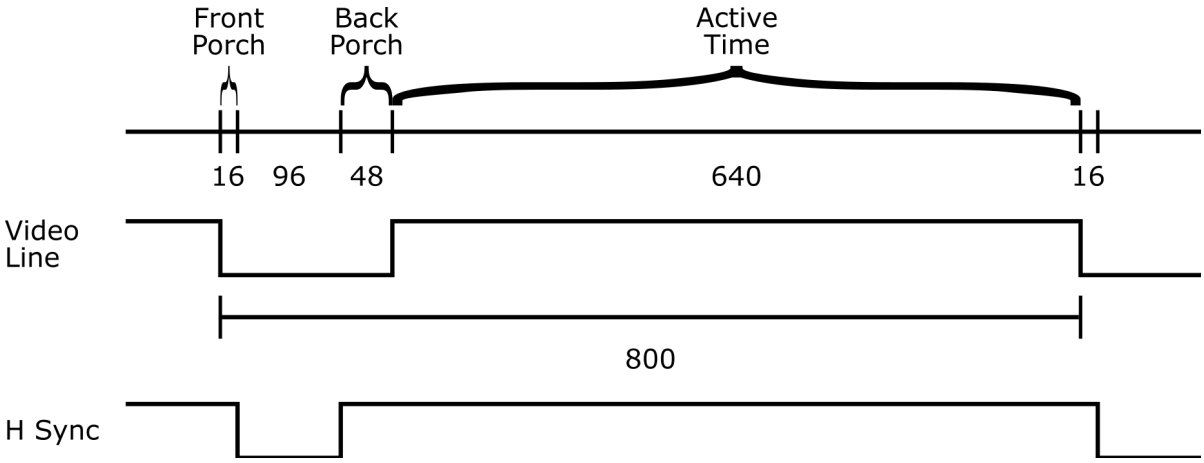


Figure 2: Horizontal Sync Timing (clock cycles)

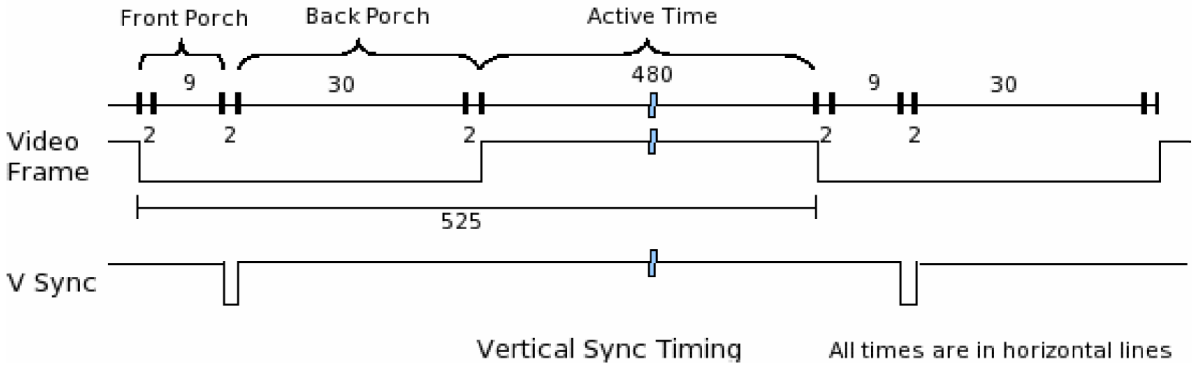
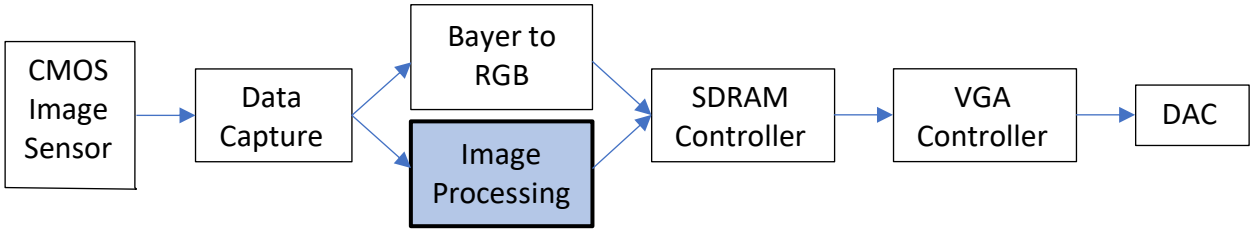


Figure 3: Vertical Sync Timing, © 2008, Mac A. Cody

Image Processing Assignment

For this mini-project you need to write Verilog for an image processing module and integrate it into the reference design to process the stream of pixels output by the camera before they are written to SDRAM. You may modify the reference design including the camera configuration. The output of image processing will be grey scale, so identical pixel values should be sent to the three VGA color channels.

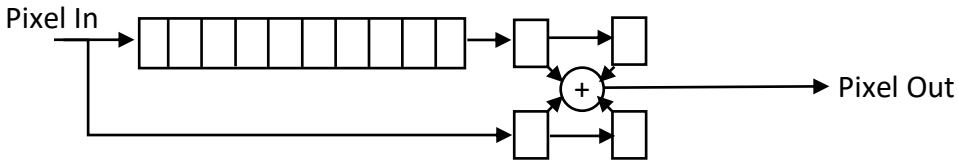


The image processing module should perform 2D convolution with a 3x3 filter. Convolution is used in many different image processing algorithms. The 2D-convolution operation takes an image (x) and a filter (h) as input and outputs a filtered image (y). The following equation describes convolution with a 3x3 filter:

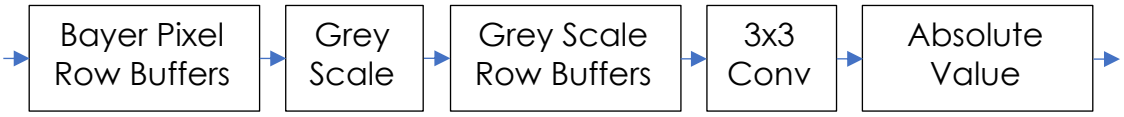
$$y[i][j] = \sum_{m=-1}^{1} \sum_{n=-1}^{1} x[i-m][j-n] \times h[m][n],$$

where each pixel in the output image is a weighted sum of the neighboring pixels of the input image.

The input image for the convolution only has one color channel, so a conversion to greyscale should be done by interpolation of the Bayer pattern before convolution is applied to the input image. The interpolation requires two rows of pixels from the camera, so Bayer pixels should be placed into a shift register with enough capacity for a full row from the camera.



The 3x3 convolution requires a second set of shift registers to maintain at least two rows of the grey pixels. For some filters, the output of convolution can be negative so it may be necessary to return the absolute value of the convolution from your 3x3 convolution module in order to display it on the screen. The block diagram is shown below.



For pixels at the edges of the screen, there aren't enough input pixels to apply the full convolution kernel. These pixels can be omitted or you can duplicate pixels at the edge to fill the filter inputs.

Using and Creating IP

There are two ways to generate IP in Quartus. The first is "Tools->IP Catalog" which provides a user interface for setting parameters of modules that can be instantiated in Verilog. Some useful IP accessible from the catalog are the Altera PLL and FIFO, which can be used for creating and crossing different clock domains. Some of the IP like shift registers and ROMs are more easily implemented directly as Verilog. Open a Verilog file in Quartus and check "Edit->Insert Template->Verilog HDL" for synthesizable constructs. The other way to generate IP is using QSYS (known as Platform Designer in recent releases of Quartus). The IP catalog in QSYS is a bit more extensive and includes an SDRAM controller, but the presets need to be set manually.

Testing

You need to write a testbench for your image processing module. The purpose of the test bench is to verify proper timing of filtered pixels and correctness of filter outputs. You should annotate the simulation logs in the report.

To simulate IP, the path to the altera megafunctions must be included in the modelsim "search libraries first" list on the libraries tab of the "start simulation" dialog. "C:/altera_lite/16.0/modelsim_ase/altera/verilog/altera.mf" is the path for Quartus 16.0.

For the demo, the filter coefficients should be setup to perform edge detection using the Sobel operators shown below.

Sobel

-1	0	1
-2	0	2
-1	0	1

Sobel

-1	-2	-1
0	0	0
1	2	1

Submission

Your submission should be a single zip file including –

1. Well commented Verilog code. Please clearly indicate which Verilog files are written by you, which files are generated, and which files are copied.
2. The Quartus flow summary exported to an rpt file. Make note of the utilization of ALMs, BRAM, and DSP blocks in the report.
3. Report describing your implementation, test bench, problems encountered and solutions for those problems (if you solved them).