

CprE 488 – Embedded Systems Design

MP-2: Digital Camera Design

Assigned: Monday of Week 6

Due: Monday of Week 8

Points: 100 + bonus for additional camera features

[Note: at this point in the semester you should be fairly comfortable with using the Xilinx VIVADO development environment, and so these directions will only expand upon the parts that are new. The goal of this Machine Problem is for your group to become more familiar with three different aspects of embedded system design:

1. *IP integration – you will work with several different IP cores that interface on the AXI Stream bus.*
2. *Digital image processing – you will gain exposure to some of the basic computational image processing kernels, and will build a digital camera by combining the individual components.*
3. *HW/SW tradeoffs – you will analyze the performance tradeoffs inherent in an embedded camera system as you first design software components and iteratively replace them with equivalent hardware IP cores.]*

1) Your Mission. You've had a great run as a Chemical Process Engineer at Eastman Kodak. While it's true that cameras using Polaroid and other film-based technology are not as popular as in their heyday, look on the bright side! You have a generous pension plan, great coworkers, and a modest 4 bedroom home in lovely Rochester, NY. With mere months to go until you can start enjoying your retirement, you're surprised by an impromptu meeting request by the CEO. The (one-sided) conversation starts a bit ominously, and proceeds at a rapid pace: "Sit down, we need to talk. They say you're my most capable engineer. Now I keep hearing about these so-called *digital* cameras. I don't know what that is, and frankly, new technology scares me. So I need a prototype on my desk in no more than 14 days – no excuses!"

It's time to get to work. You know a little bit about camera optics (and certainly HW-2 provided a quick refresher), but how to transform that to a useful digital output is well outside your comfort zone. Fortunately, you have a skeleton project that provides the basic framework. Your task is to use system design techniques to implement an image processing pipeline and other functionality commonly found in digital cameras.

2) Getting Started. The ZedBoard wasn't sufficiently complicated by itself, so we've coupled it with the Digilent FMC-Pcam adapter card. The FMC-Pcam connects via the FPGA Mezzanine Card (FMC) connector on the ZedBoard, and provides the following features:

- Video input from up to 4 Digilent PCAM 5C cameras
- I2C interface for PCAM 5C configuration
- GPIO interfaces for PCAM 5C management

The FMC connector's pins are directly routed to the Zynq FPGA on the ZedBoard, and so the I2C and other peripheral controllers need to be instantiated in our VIVADO design. **Make sure the ZedBoard is turned off while plugging in the FMC-IMAGEON.**

HDMI: We will use the Zedboard's HDMI interface to output video to an HDMI monitor.

Note: the HDMI output controller can become incorrectly configured when a new software application is downloaded, so when testing new software designs you will want to first re-download the FPGA bitstream and/or reboot the ZedBoard.

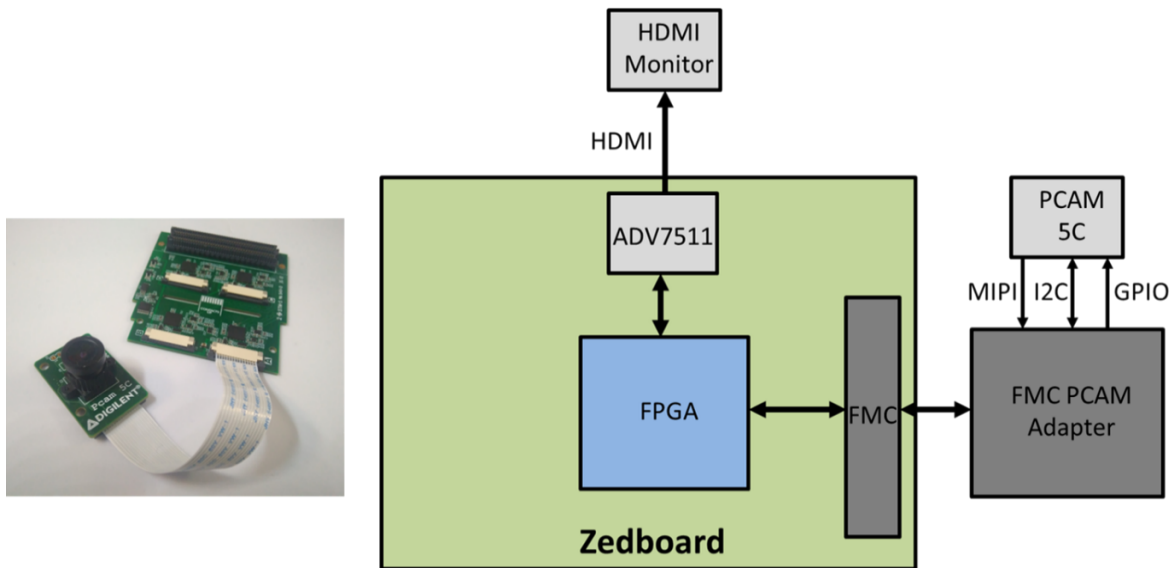


Figure: Pcam FMC Adapter and Pcam 5C camera (photo), and block diagram of MP2 system. Use care when attaching and un-attaching the FMC card. It is quite expensive so please don't break it. Pretty pretty please.

Given the complexity of this assignment, we have provided a starter VIVADO project that you can use as the baseline for implementing the digital camera functionality. Download the provided MP-2.zip file, unzip, and peruse the directory structure. You will find another zip file called mp2.xpr.zip. This contains a base project with an initial design for HDMI video output. Unzip this zip file and open the project in Vivado using Open Project, then find the xpr file within the project folder.

3) Design Test and Analysis. The initial design provides a Test Pattern Generator (TPG) that creates a 1080p image, which is streamed into DRAM via our old friend the Video Direct Memory Access (VDMA) module. A software loop performs some simple processing on the incoming video stream, and the VDMA takes the processed pixels and streams them to the HDMI controller chip.

The design is already 100% functional, so you just need to 1) Generate a bitstream, 2) Export HW (Include Bitstream), 3) Launch SDK – but don't walk away! While the system is building, analyze the design using the Block Diagram view, and Implementation view, as well as directly through the MP2.xcd file. **In your writeup provide the following:**

- A detailed system diagram that illustrates the interconnection between the various modules in the system, both at the IP core level (i.e. the components in your VIVADO design) as well as the board level (i.e. the various chips that work together to connect the output video to your monitor). The documents found in MP-2/docs/Camera will be of assistance in understanding the various components, and the IP core documentation is found in MP-2/docs/IP.
- A detailed description of how the hardware in the starter MP-2 design is intended to operate. Make sure to describe the role of the various I2C interfaces, how the Video Timing Controllers (VTCs) are being used, and what differentiates this VDMA from the version we used in MP-0. Also, explain the role of the various clocks in the system (be specific).

After bitfile generation completes [**Note:** OK if this bitfile has some timing violations], and you have Exported the Hardware and Launched Vitis, we need to set up the software side of the project. Use the follow steps:

- 1) Create a New Platform Project. Call it TPG, and associated it with your exported hardware (.xsa file)
- 2) Create a New Applicate Project. Call it MP2-TPG. Select the “**Empty Application (C++)**” Template
- 3) In the MP2 file structure you downloaded copy everything from the `sw/camera_app/src` directory and paste them into the MP2-TPG/src directory.
- 4) Rebuild the MP2-TPG project:
 - a) Right click and select “Clean Project”
 - b) Right click and select “Build Project”

Download the bitfile and `camera_app` executable to your board to ensure that the starter design is working correctly. Modify the test pattern that is being generated to demonstrate your understanding of the general `camera_app` structure. Provide at least two modifications: one which configures the TPG core directly (see the provided TPG datasheet for several examples of this), and one which uses the software processing loop in `camera_app.cc` to modify the incoming video stream. Describe in your writeup what changes you made, and save a copy of any files modified (presumably only `camera_app.cc`) during this process into a folder named `part3/`.

4) Grayscale Camera. Navigating back to VIVADO, it is time to interact with the Pcam 5C image sensor. The following gives detailed directions for how to connect to the Pcam 5C. Also a nice detailed PDF of what your system should look like after these steps is provided in the top level of the MP2 file structure (`Gray-Scale-PassThrough.pdf`).

1) Add the “MIPI CSI-2 Rx Subsystem” IP block

- i) Make the “`mipi_phy_if`” interface an external pin, then rename that external pin to “**MIPI_DPHY**”
- ii) AXI Lite Interface
 - Connect “`csirxss_s_axi`” to the system as an AXI lite bus
 - Connect “`lite_aclk`” to the AXI lite bus clock
 - Connect “`lite_aresetn`” to the AXI lite reset
- iii) Connect “`dphy_clk_200M`” to the `clk_wiz_0` “`clk_out3`”, this is a 200 MHz clock
- iv) Connect “`video_aclk`” to the `clk_wiz_0` “`clk_out2`”, this is a 150 MHz clock
- v) Connect “`video_aresetn`” to the `rst_system_150M` “`peripheral_aresetn`”
- vi) Configure this IP core as follows:
 - Pixel Format: RAW10
 - Serial Data Lines: 2
 - Include Video Format Bridge (VFB): Check
 - Line Rate (Mbps): 420
 - D-PHY Register Interface: Check
 - Calibration Mode: None
 - CSI2 Controller Register Interface: Check
 - Line Buffer Depth: 2048
 - Allowed VC: All
 - Pixels Per Clock: 1
 - TUSER Width: 1
 - Enable CRC: Check

2) Update the "axis_subset_converter_0" block

i) Configure "axis_subset_converter_0" as:

- Slave Interface Signal Properties: TDATA Width=2
- Extra Settings, TDATA Remap String= 8'b10000000,tdata[9:2]

3) Remove the TPG IP block

i) Disconnect each of the pins of the TPG IP block. Care not to delete the wires, or more things may be deleted than what you intended

ii) Once all the pins have been disconnected, delete the TPG IP block.

4) **Video_out:** Connect the mipi_csi2_rx_subsys0 "video_out" to slave input of the axis_subset_converter_0

5) **Update XDC file:** Check that all lines of the XDC file are uncommented, except for the one labeled as "phjones: UNUSED"

Questions:

1) Note that several of these ports in the XDC file are paired together, with one port ending in `_p` and the other ending in `_n`. In your writeup, briefly describe what this pairing of signals signifies, and what this configuration is typically used for.

2) We modified the 16-bit "mipi_csi2_rx" IP core output by keeping bits 9:2, and appending the 8-bit value "10000000" (see step 2.i), before passing this signal to the VDMA. Explain why this is an appropriate value to append, and why appending "00000000" would not make sense.

Confirm that your Block Design passed the validate check, and then follow the typical steps for building a bitfile (See MP-0 tutorial if you need a reminder on these steps), then Export the Hardware and Launch SDK. While you wait, meditate on what you have learned.

Things on the software side are not nearly as complicated. In `camera_app.cc`, uncomment the functionality for the vita receiver initialization code, and make sure you call *the function for enabling the camera* instead of *the TPG*. Remove any previous transformation code in `camera_loop()`, and test that your design works as expected. In your writeup, briefly explain why the camera at this stage is not outputting any color.

5) Color Conversion Software. As discussed in HW-2, we can colorize this grayscale camera by applying a Bayer filter.

A) Matlab prototype of Color Conversion Software:

- Use Matlab to create a “Bayer” image that emulates what the camera will provide your system. You can create such a test image by starting with a RGB 24-bit BMP color image (choose an image) and converting it to a “Bayer” image by just using a signal color component from each pixel in the color image based on the “Bayer Pattern” (as a quick estimate) or you could apply the reverse transformation that would be applied by the CFA.
- Now use Matlab to implement an algorithm to convert this “Bayer” image into a 4:2:2 Color image.
- Note: you will find the IP core documentation useful for details as you prototype your conversion algorithm.
- Provide your Matlab Prototype software and your original RGB image, corresponding Bayer image, and final output of your conversion algorithm in a folder named `part5/`

B) Create a software implementation for the Bayer color filter array in function `camera_loop()`, based on your Matlab algorithm prototype. Describe in your writeup what changes you made, and save a copy of any files modified (presumably only `camera_app.cc`) during this process into a folder named `part5/`. While you have already derived a pseudocode implementation for this operation, there are several complicating details:

- Although we are streaming 16-bit values to the VDMA (to be processed via software), based on our system configuration above, only every other byte represents the vita camera output.
- The output of your Bayer pattern will be an RGB image, presumably with a 24-bit pixel representation (since you can directly capture the 8-bit R, G, or B component). However, both the VDMA and the HDMI output is configured to use 16-bit pixels. Specifically, the HDMI is expecting 16-bit values in a 4:2:2 YCbCr pattern.
- YCbCr is a family of color spaces used as part of the color image pipeline in video and digital photography systems. The ‘Y’ component corresponds to the relative luminance, with ‘Cb’ and ‘Cr’ corresponding to the blue-difference and red-difference chroma components. Note that YCbCr is not a color space in the strict sense that RGB is; it is more of an encoding scheme for a color space (such as RGB). The matrix equation for this conversion is given as follows:

$$\begin{bmatrix} Y & Cb & Cr \end{bmatrix} = \begin{bmatrix} 0.183 & 0.614 & 0.062 \\ -0.101 & -0.338 & 0.439 \\ 0.439 & -0.399 & -0.040 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}$$

- The YCbCr 4:2:2 pattern is an example of an encoding scheme referred to as chroma subsampling: see http://en.wikipedia.org/wiki/Chroma_subsampling#4:2:2, and the following datasheets for useful details “Chroma Resampler” (`pg012_v_cresample.pdf`), and “AXI4-Stream Video IP and System Design” (`ug934_axi_videoIP.pdf`). Because the human visual system is less sensitive to the position and motion of color than it is to luminance, bandwidth can be optimized by storing more luminance detail than color detail. Look at the VDMA initialization code in function `fmc_imageon_enable()`, and infer from the Red, Green, and Blue examples how the 16-bit 4:2:2 YCbCr format is encoded. Briefly describe this in your writeup, and use this format as the output of your `camera_loop()` conversion pass.

In your writeup, describe the performance of your software-based color conversion (in terms of frames per second), and how you measured it. Overall this is a non-trivial piece of software, so put in a good faith effort for this part and in your writeup, describe your testing methodology. If you get really stuck, fork your project so that you can continue to work on the remaining system design parts.

6) Image Processing Pipeline. Although there are all sorts of software optimizations that can be applied to the color filter array in the previous section, the overall performance will likely remain insufficient for our digital camera needs. Fortunately for us, we can build a hardware image processing pipeline that should be able to keep up with the input and output throughput requirements. Comment out the relevant *camera_loop()* code, and switch back to VIVADO.

The three cores you will need to integrate to your project are: 1) Sensor Demosaic (*v_demosaic_0*) to convert the Bayer pattern pixel stream of the camera sensor into an RGB pixel stream, 2) Video Processing Subsystem IP configured for Color Space Conversion Only (*v_proc_ss0*) to convert the RGB pixel stream to 4:4:4 YCrCb pixels, and 3) Video Processing Subsystem IP configured for 422-444 Chroma Resampling Only (*v_proc_ss1*) to convert the 4:4:4 pixel stream to 4:2:2 pixels. Note the YCrCb format as opposed to YCbCr. At this point in the semester, you should be sufficiently experienced in VIVADO to be able to integrate these cores without a detailed walkthrough.

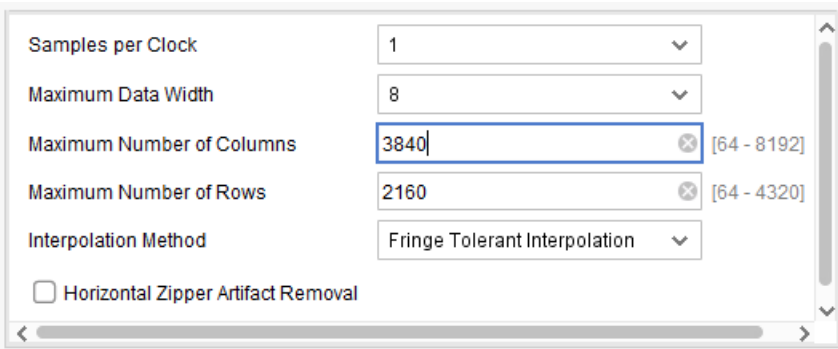
The image pipeline should proceed from Pcam -> MIPI CSI-2 Rx Subsystem -> *axis_subset_converter* (8-bit output) -> (8-bit input) *v_demosaic_0* -> *v_proc_ss0* (Color Conversion Only) -> *v_proc_ss1* (422-444 Chroma Resampling Only) -> *axis_subset_converter* -> *vdma_S2MM* -> *vdma_MM2S* -> *vid_out* -> *hdmi_out*. Provide a diagram for this awesome pipeline in your writeup, making sure to label the bit width of the relevant signals.

***Note 1:** The MIPI CSI-2 Rx Subsystem outputs RAW 10-bit sensor values, to keep things simple we will just pass the most significant 8-bits of the sensor data to the Color HW Pipeline. This is the purpose of the *axis_subset_converter* that is between the MIPI CSI-2 Rx Subsystem IP, and Demosaic IP

***Note 2:** You will need to update the *axis_subset_converter* (the one between the Resampling and VDMA) Slave Interface for TDATA to be 3, to match the width of the 422-444 Chroma Resampling Only IP block. Yes this is a bit odd, but since this IP block can be configured using software to convert both directions (i.e. 422->444, or 444->422) they have made the input and out interfaces fixed to 24-bits in size.

Here are screen shots of recommended settings for these three IP cores:

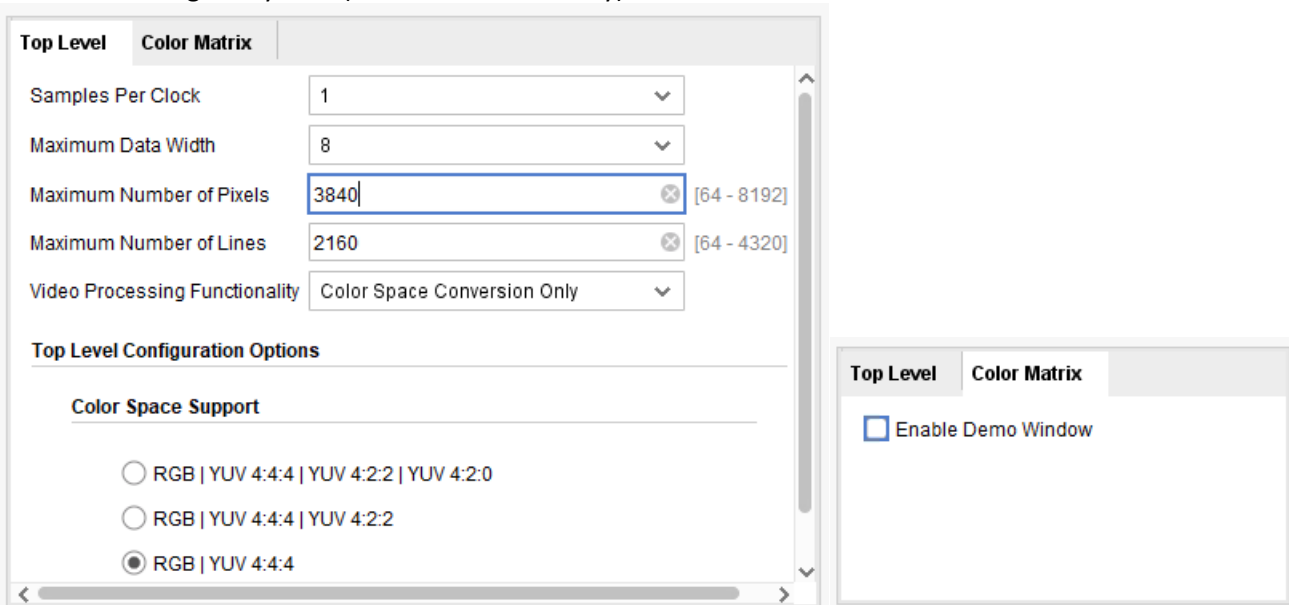
Demosaic:



Configuration window for Demosaic:

- Samples per Clock: 1
- Maximum Data Width: 8
- Maximum Number of Columns: 3840 [64 - 8192]
- Maximum Number of Rows: 2160 [64 - 4320]
- Interpolation Method: Fringe Tolerant Interpolation
- Horizontal Zipper Artifact Removal

Video Processing Subsystem (Color Conversion Only):



Configuration window for Video Processing Subsystem (Color Conversion Only):

Top Level | Color Matrix

- Samples Per Clock: 1
- Maximum Data Width: 8
- Maximum Number of Pixels: 3840 [64 - 8192]
- Maximum Number of Lines: 2160 [64 - 4320]
- Video Processing Functionality: Color Space Conversion Only

Top Level Configuration Options

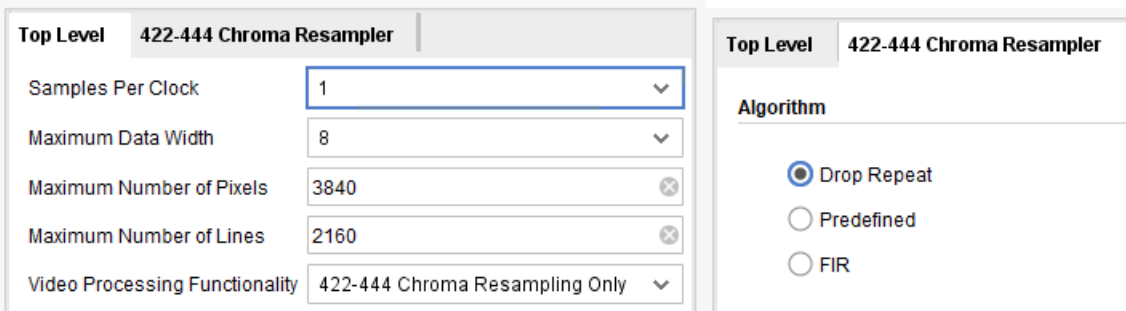
Color Space Support

- RGB | YUV 4:4:4 | YUV 4:2:2 | YUV 4:2:0
- RGB | YUV 4:4:4 | YUV 4:2:2
- RGB | YUV 4:4:4

Top Level | Color Matrix

- Enable Demo Window

Video Processing Subsystem (422-444 Chroma Resampling Only):



Configuration window for Video Processing Subsystem (422-444 Chroma Resampling Only):

Top Level | 422-444 Chroma Resampler

- Samples Per Clock: 1
- Maximum Data Width: 8
- Maximum Number of Pixels: 3840
- Maximum Number of Lines: 2160
- Video Processing Functionality: 422-444 Chroma Resampling Only

Top Level | 422-444 Chroma Resampler

Algorithm

- Drop Repeat
- Predefined
- FIR

After you rebuild your design, then Export Hardware, and Launch Vitis. Most of the image pipeline-related code is already provided for you in the `camera_app` project, so just uncomment out the headers, configuration data, and pipeline enable calls in `camera_app.cc`, and, `camera_app.h`. You will have to investigate the initialization code for the Demosaic, Chroma Resampler IP core. The Demosaic IP core should be straightforward to configure using its datasheet. The Video Processing Subsystem configured as Chroma Resampler Only is almost as straightforward, but you will need to dig through the Xilinx header files a bit to find appropriate values for configuring a couple of registers. In your writeup, describe the performance of your image processing pipeline (in terms of frames per second), and how you measured it.

7) Making the Camera. At this point you've put in a considerable amount of effort, but the current system only implements a video pass-through. Create a new function called `camera_interface()` which adds the following user interface functionality:

1. Pressing the middle button should capture the current frame as a raw image. Store up to 32 images (in memory), and when a new image is captured, it should be displayed on the screen for 2 seconds.
2. Have one of the switches activate playback mode. In this mode, the left and right buttons rotate through the previously captured images.

Provide a copy of any modified code for this section in a folder named `part7/`.

What to submit: a .zip file containing 1) A cleaned archive of your xpr project, 2) Your updated `design_1_wrapper.vhd` and `zedboard_fmc_imageon_gs.xdc` files, 3) your Matlab prototype for color conversion (including the image you converted), 4) your modified source files (the previously mentioned directories with changes to `camera_app.cc` and `camera_app.h`), and 5) your writeup in PDF format containing the highlighted sections of this document. In the Canvas submission, list each team member with a percentage of their overall effort on MP-2 (with percentages summing to 100%).

What to demo: at least one group member must be available to demo the current state of your implementation. A full demo score requires a system that can capture images using the completed hardware pipeline, but partial credit will be given for effort. Be prepared to briefly discuss your source code.

BONUS credit. MP-2 has two separate bonus point criteria. The first is for extra camera *features*. Consider what additional features a typical point-and-shoot camera has over our simple MP-2 implementation. Some possible examples (and their bonus point worth):

- A video mode, which records and can replay up to 5 seconds of 1080p video. (5 bonus points).
- A digital zoom mode, which uses the up and down buttons to zoom in and out of the current scene. (5 bonus points).
- Various analog and digital adjustments for the gain, exposure, and other common user-configurable digital camera settings. (2 bonus points each: up to 6pts)

The second MP-2 bonus point criterion is additional image processing *pipeline stages*. Similar to the steps followed in part 6), to be eligible for bonus points any new pipeline stage will need a comparison between a software and hardware-based implementation. For example, using filters for which VIVADO has an appropriate IP core. In general, a few commonly used filters for image processing are Sobel, and Discrete Laplace (http://en.wikipedia.org/wiki/Sobel_operator, http://en.wikipedia.org/wiki/Discrete_Laplace_operator) (8 bonus points).

Each group is limited to 100 bonus points for the entire semester.