Iowa State University
Electrical And Computer Engineering

CprE 488

Machine Problem 2

February 4, 2023

Group : S1-4

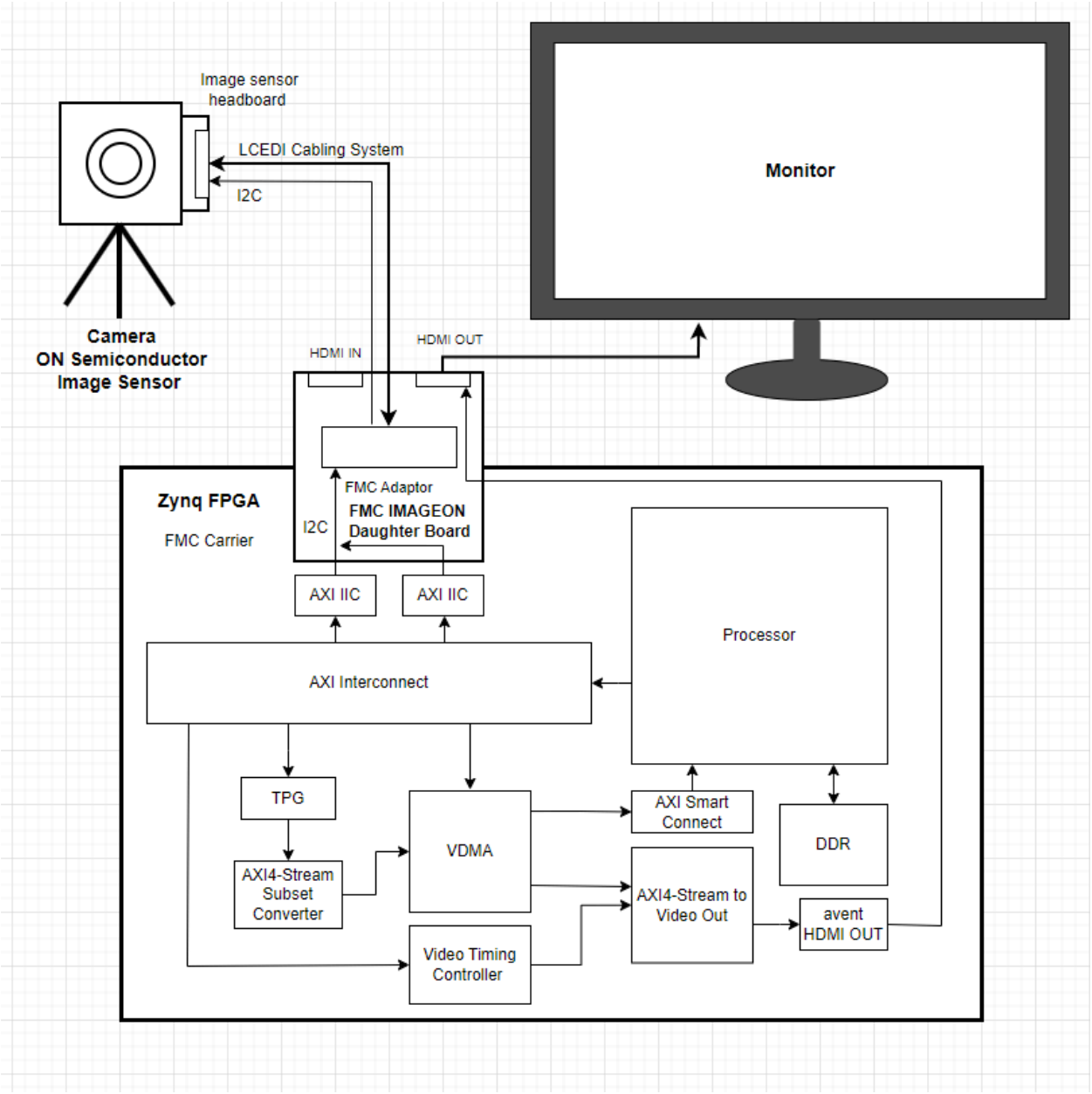
Member 1: Nathan Thoms

Member 2: Taylor Johnson

Member 3: Jack Cassidy

Member 4 : John Lavigne

1) *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).*



2) *A detailed description of how the hardware in the starter MP-2 design is intended to operate.*

The video for the digital camera design depicted above uses a ON semiconductor image sensor that captures a video input stream. The image sensor has a headboard that is connected to

the FMC-IMAGEON daughter board through a LCEDI cabling system. The cabling system connects to the FMC adapter. The daughter board then connects to the Zynq-7000 FPGA board via an FMC connector. Video data is transferred from the camera's image sensor to the AXI interface of the FPGA using I2C. In this case, the HDMI output is configured such that the test pattern generator (tpg) IP is the video source. When the tpg is configured to pass the active video from the image sensor, then the I2C protocol is used. The AXI interconnect IP then sends control signals to the tpg and video timing controller IPs. The tpg outputs a 24-bit RGB value that gets passed to the AXI4-Stream subset converter, which converts the RGB data to a 16-bit YCbCr value. The YCbCr data is passed to the VDMA IP. The VDMA then writes the frame to DDR memory via the AXI smart connect IP and Processing system. The VDMA is also responsible for reading from the DDR memory and sending the video data to the AXI4-Stream to Video Out IP. The AXI4-Stream to Video Out has another input connected to the Video Timing Controller, which is responsible for synchronization between the VDMA when a video frame is ready to be output to the display. The avent HDMI out IP connects to the HDMI output port. The video data gets sent out through the daughter board HDMI out port to the HDMI input display port on the monitor. The monitor then displays the video stream.

3) *Describe in your writeup what changes you made, and save a copy of any files modified (presumably only camera_app.c and fmc_imageon_utils.c) during this process into a folder named part3/.*

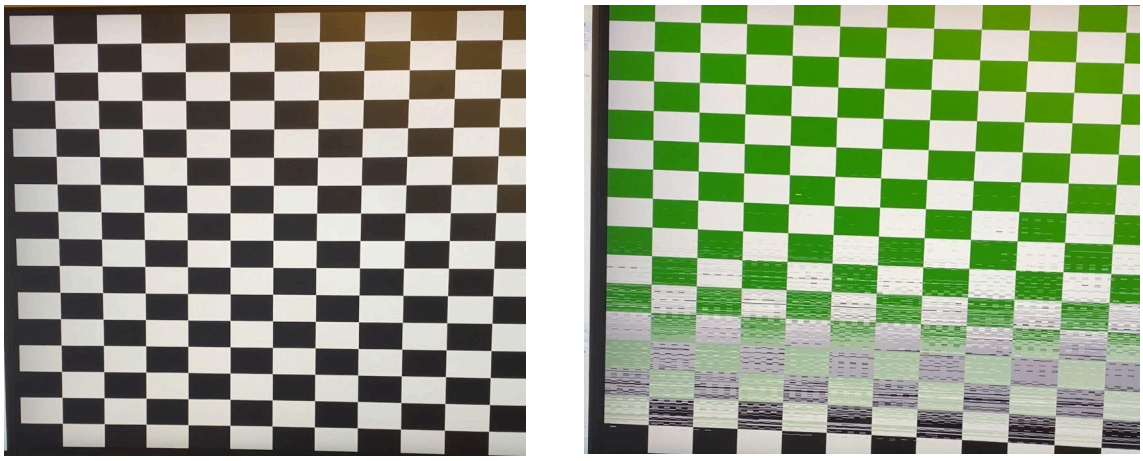


Figure 1 - *Altering TPG Output from camera_loop()*

We altered the pattern generated by the TPG through configuring a register in fmc_imageon_utils.c. Within the camera_app.c camera_loop() function we inverted the checker pattern and changed the black color to green every two frames then would revert back to the original after two altered frames.

4) *In your writeup, briefly describe what this pairing of signals signifies, and what this configuration is typically used for.*

The *_p* and *_n* denote positive and negative signal polarities respectively. Pairing positive and negative signals together like this is what is referred to as differential signaling, which is used to transmit data with reduced noise.

5) *Explain why this is an appropriate value to append, and why appending "00000000" would not make sense.*

Since we are converting to the color format, YCbCr, when we append an 8 bit value to the most significant bits of the output to the "Video In to AXI4-Stream" IP core, we are adding a Y component value. The Y component represents luminance/brightness. We append "10000000" because this is a middle range luminance that allows us to still interpret meaningful color data from the Cb and Cr components. If we were to append "00000000", this would mean the Y component gets set to zero or black. Then, we lose any meaningful color information from the chroma color components. The output would be black.

6) *In your writeup, briefly explain why the camera at this stage is not outputting any color.*

In its current state, the output lacks color because each pixel comprises only intensity information from a single color channel. Color representation stems from the combination of red, green, and blue color intensities overlapping each other. To produce a color image, we must interpolate the absent color channels for every pixel by leveraging surrounding pixel color data, thereby crafting a comprehensive RGB color representation.

7) *Describe in your writeup what changes you made, and save a copy of any files modified (presumably only camera_app.c) during this process into a folder named part5/*

Many of the details for the changes made to the camera_app.c camera loop() can be found in the provided sources files. The main difference between the MatLab implementation

and the C implementation was the need to create a means of handling border pixel transformations. The MatLab implementation used padarray() to create a border to prevent accessing uninitialized data. The second difference was the conversion from a single dimensional structure ranging from 0 to DISP_WIDTH*DISP_HEIGHT - 1 to an intuitive two dimensional indexing scheme for dealing with image data.

8) *Briefly describe this in your writeup, and use this format as the output of your camera_loop() conversion pass.*

VDMA Initialization Code :

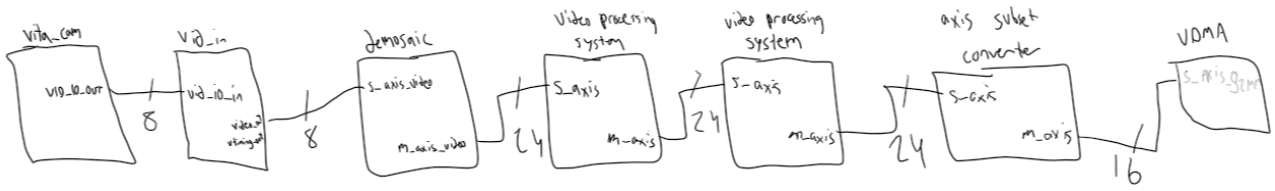
```
   // Frame #1 - Red pixels
   for (i = 0; i < storage_size / config->uNumFrames_HdmiFrameBuffer; i +=
4) {
     *pStorageMem++ = 0xF0525A52;  // Red
   }
   // Frame #2 - Green pixels
   for (i = 0; i < storage_size / config->uNumFrames_HdmiFrameBuffer; i +=
4) {
     *pStorageMem++ = 0x36912291; // Green
   }
   // Frame #3 - Blue pixels
   for (i = 0; i < storage_size / config->uNumFrames_HdmiFrameBuffer; i +=
4) {
     *pStorageMem++ = 0x6E29F029;  // Blue
   }
```

The output conversion pass of the camera_loop() function follows the same format as above; A compact way to store pixel data in the YCbCr 4:2:2 format. The main difference is that we stored the results in 16 bit memory locations versus the 32 bit locations above. The format above follows this format Cr[0]&Y[1]&Cb[0]&Y[0] - the luminescence is sampled every pixel and the Cb and Cr values every other.

9) *In your writeup, describe the performance of your software-based color conversion (in terms of frames per second), and how you measured it. In your writeup, describe your testing methodology.*

We got 1.2 fps for the software-based color conversion, to get this we used the XTime_GetTime from the xtime_l.h file which would get current time from the counter register. We would put the get time function in front of the for loop that ran each frame and at the end of it and then subtracted the end time by the start time and divide by COUNTS_PER_SECOND which is the amount of clock cycles that happen per second in the cpu.

10) *Provide a diagram for this awesome pipeline in your writeup, making sure to label the bit width of the relevant signals.*



11) *In your writeup, describe the performance of your image processing pipeline (in terms of frames per second), and how you measured it.*

We established a lower bound of roughly 10 fps by starting a timer and counting the frames that were copied within that time period. Dividing the number of frames by the duration resulted in that fps.
From using the testing procedure from the color conversion software, we got a fps of roughly 18 when we had to copy pixel data from one memory location to another. In reality the hardware pass through would have a greater throughput.

Maybe a more accurate way to quantify the fps would be to look at the throughput of say the demosaic IP and similar IPs within the pipeline. If it can convert a pixel into a clock cycle then you can find the time taken to convert an entire frame at a given clock rate to be roughly 71.37 fps at a clock rate of 148 MHz with frame resolution of 1080p.

Team Effort

Nathan Thoms - 25%
Taylor Johnson - 25%
Jack Cassidy - 25%
John Lavigne - 25%