Name: Joseph Ward, Yao Jiang Cheah, Raed Ibrahim, Malcolm Johnson

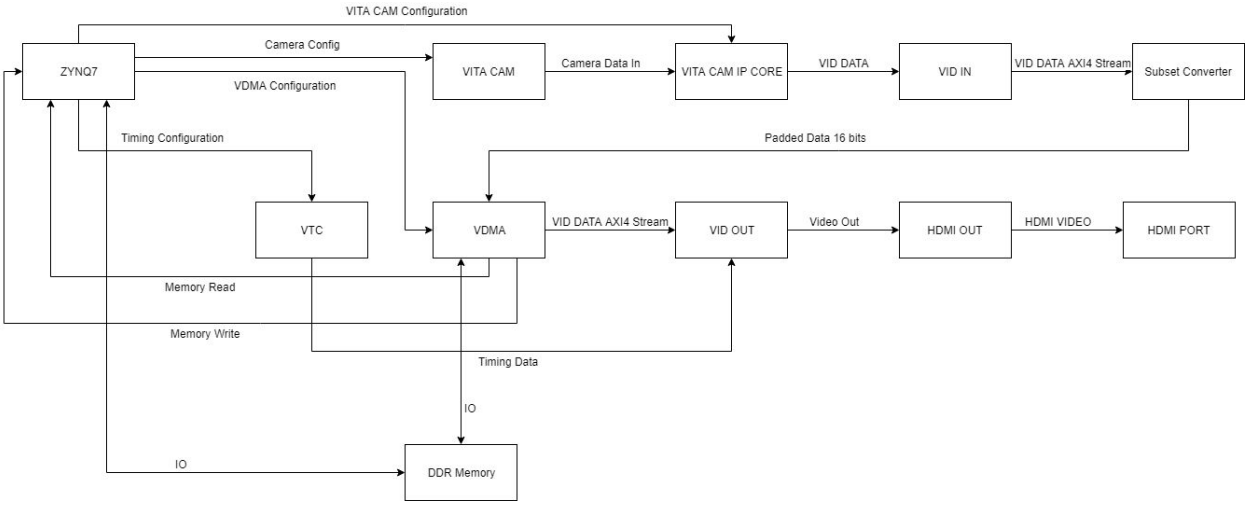
CprE 488 - Embedded Systems Design

MP-2: DIGITAL CAMERA DESIGN

1. **(Part 2: Getting Started)** Make sure the ZedBoard is turned off while plugging in the FMC-IMAGEON.

   DONE

2. **(Part 3: Design Test and Analysis)** Provide 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).



3. **(Part 3: Design Test and Analysis)** Provide a detailed description of how the hardware in the starter MP-2 design is intended to operate.

The raw data from the camera is fed into the VITA Cam Receiver IP core. This core processes the data and outputs the video data. The video data is then processed into the AXI4 stream from the VID_IN core and then written into memory via the VDMA. The VDMA then reads the data and outputs it to the VID_OUT core which utilized the timing data from the VTC and creates a video signal and then converts it into the proper HDMI signal via the Avnet HDMI Output core.

A key difference between the VDMA in this lab and the one in the first lab is that this one both writes and reads to the memory. In MP-0, the VDMA could only read from memory.

Name: Joseph Ward, Yao Jiang Cheah, Raed Ibrahim, Malcolm Johnson

VTC CLK: 148.5 MHz Used to generate timing signals for the Video Out. Also used for the Memory stream to and from the VDMA.

AXI BUS CLK: 76.9MHz Used for the AXI Bus which is used to configure the various ip cores

Memory Bus CLK: 142.85MHz Used by the VDMA M_AXI_MM2S and S_AXI_S2MM busses

VITA CLK: 200MHz Used by the VITA CAM IP core

4. **(Part 3: Design Test and Analysis)** 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/.

We Changed the fmc_imageon_utils and specifically registers in the fmc_imageon_enbale_tpg() function.

The background pattern register which is reg 0x20 we changed its value from 0xA the Zone plate output to 0x10 which is just a pseudo random pattern and I changed the motion speed register 0x38 from 0x4 to0x20 which increases the value to increment by frame but that should not do anything amazing there is no zoneplate now.

In camera_app, we enabled circular park mode before and after in reading the stream and supposedly that should increase the smoothness when outputting through the software mode.

5. 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 write up, briefly describe what this pairing of signals signifies, and what this configuration is typically used for.

   I think this pairing is a naming standard for positive and negative wiring probably used when you have a wire with negative and positive changing voltage.

6. We convert the 8-bit output of the “Video In to AXI4-Stream” IP core to 16-bits to be given to the VDMA by appending the 8-bit value “10000000” (see step 3.vii),. Explain why this is an appropriate value to append, and why appending “00000000” would not make sense.

   Setting the front 8-bit to value “10000000” is a way of defining that it is a grayscale image because the top 8-bit corresponds to Cb & Cr which when the red and blue difference is half of 255 (which is 128), it creates a grayscale image. If we set the values to “00000000” means no red and blue difference at all so the image will just be very green tinted.

7. Things on the software side are not nearly as complicated. In fmc_imageon_utils.c, uncomment the functionality for the vita receiver initialization code, and make sure you call fmc_imageon_enable_vita() instead of fmc_imageon_enable_tpg(). Note that since the appropriate libraries do not get included until the core is added to the project, you will also need to uncomment the vita-related code in camera_app.h and camera_app.c as well. Remove any previous transformation code in camera_loop(), and test that your design works as expected. In your write up, briefly explain why the camera at this stage is not outputting any color.

The colors are grayscale since only the luminance channel is being sent in the output and the Cr and Cb channels(which give the color) are padded to 10000000. Thus the monitor only receives the brightness of each pixel.

8. 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/

   Done

9. 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.c) during this process into a folder named part5/.

   When implementing the matlab code in c for the project, we had to change the arrays that we were reading and writing from/to as well as the format that we were outputting. Specifically the output had to be in 4:2:2 format.

10. The YCbCr 4:2:2 pattern is an example of an encoding scheme referred to as chroma subsampling: http://en.wikipedia.org/wiki/Chroma_subsampling#4:2:2. 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

    The format of the 4:2:2 YCbCr signal is 8 bits to store either Cr or Cb, and 8 bits to store luminance. Each pixel is two bytes so Cr and Cb is stored on alternating pixels.

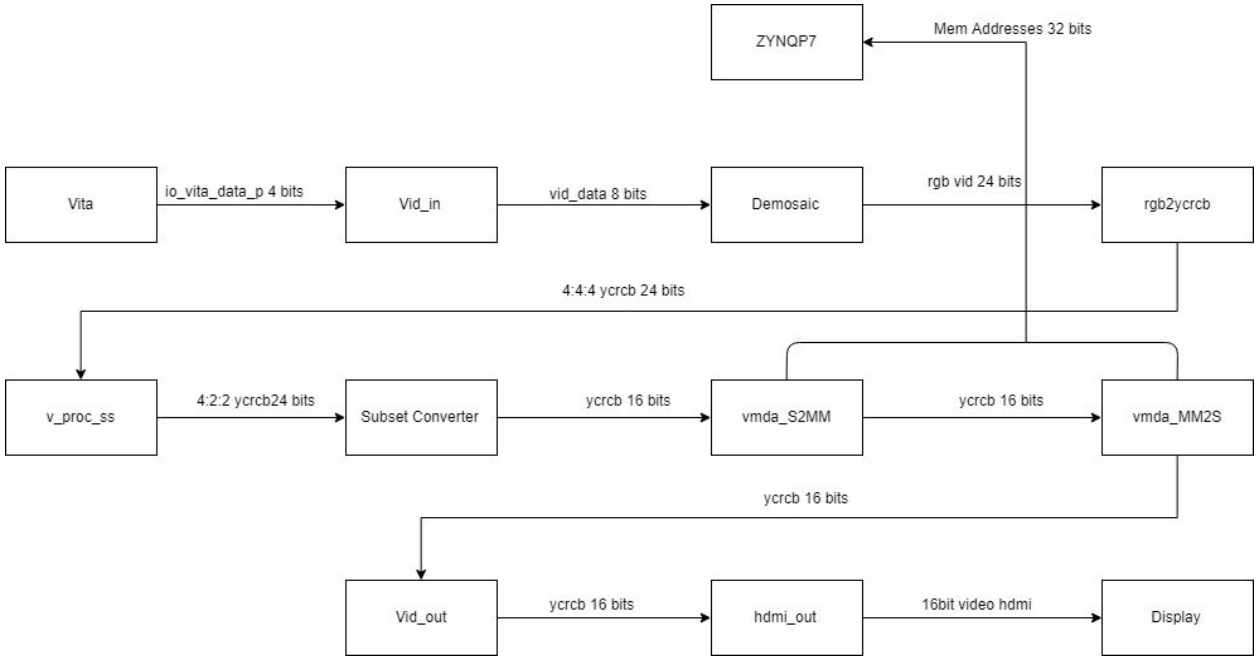
11. In your write up, describe the performance of your software-based color conversion (in terms of frames per second), and how you measured it. Overall this is a non-trivial piece of software, so put in a good faith effort for this part and in your write up, describe your testing methodology.

Most of the testing took place in implementing the prototype. This involved using solid images of red green and blue to ensure we were grabbing the right colors when creating a the rgb → bayer and bayer → rgb programs. Once we were sure that these conversions were correct we ported the code to c. The first attempt of running the new c code resulted in incorrect colors. To debug this we tried outputting constant color values to the screen inorder to figure out the HDMI ycrcb output format. To test the fps of the program, we first tried to use a timer on the chip to count the time it takes to output a few frames. This turned out to not be as simple as we thought so we changed our strategy to just use a the timer on our smartphones to time how long it took to display 200 frames. Putty was used to output when to begin and end timing. Once we had the time, calculating the fps is as simple as dividing the time by 200.

Time for 200 frames = 1:54 or 114 seconds

The FPS for software mode was: 1.75 FPS

12. The image pipeline should proceed from vita -> vid_in -> cfa -> rgb2ycrcb -> cresample -> 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.



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

The performance of the hardware pipeline is much greater than that of the software pipeline. We measured the fps the same way as in the software pipeline with the exception of measuring 1000 frames instead of

200 since it was so fast. The result is that it only took 13 seconds to output 1000 frames. This comes out to 78 fps which exceeds the refresh rate of the monitor

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

Done, provided in the project folder.