VmodCamera How-To Guide

This guide involves getting the VmodCamera peripheral setup up and running on a computer connected to a Virtex-5 FPGA board. It uses the VmodCamera.zip file as a starting point and refers to the online Digilent manual at http://www.digilentinc.com/Data/Products/VMOD-CAM/VmodCAM_rm.pdf.

Step 1: Download the VmodCamera.zip file onto your computer.

It should contain 4 folders: binaries, doc, IPRepository, and proj. The doc folder contains a PDF guide, similar to this one, on how to set up the VmodCamera. The binaries folder has a bit file that has been configured following the guide, so if you run it using Digilent Adept (http://www.digilentinc.com/Products/Detail.cfm?Prod=ADEPT2), it should display two screens for each camera. However, we did not find a way to configure that bit file for our own purposes, so we found it best to start from scratch and generate our own bit file. IPRepository holds the peripherals you need to add to the XPS project to connect the VmodCamera to the FPGA board. Finally, the folder proj holds several test projects to compile in Xilinx SDK software as well as system.xmp which will be our starting point.

Step 2: Open up proj/system.xmp in Xilinx XPS and check for the correct peripherals.

Go through the update if your files are out of date.

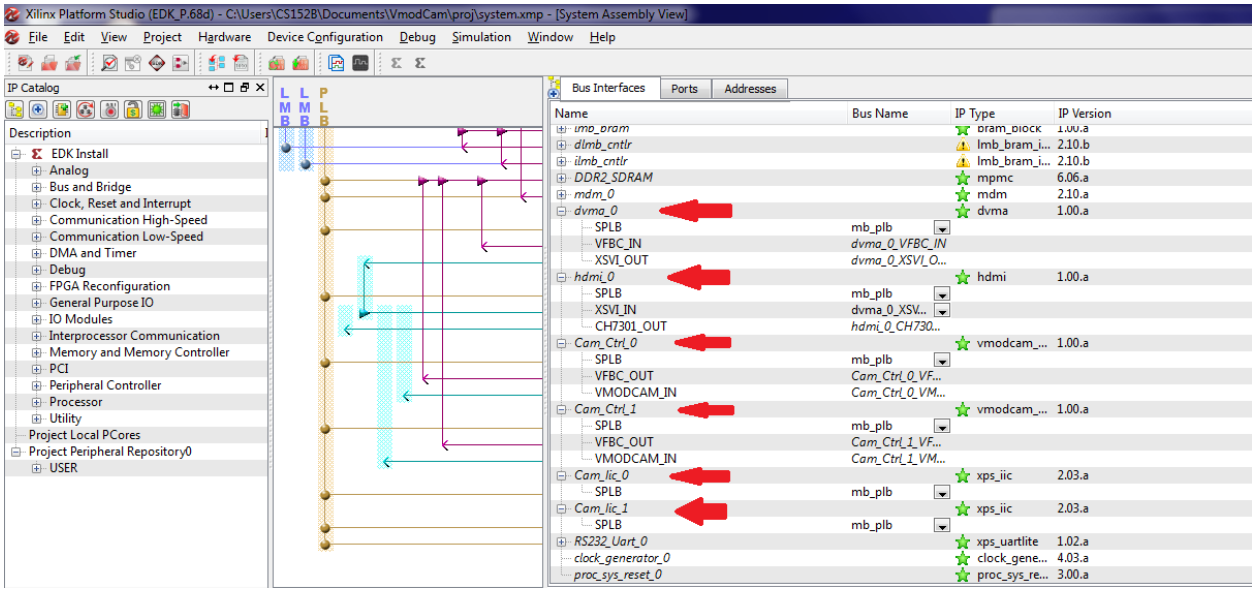

Figure 1: Peripheral setup on XPS

In the System Assembly View, you should already see a bunch of peripherals that have been added. If you're using the right xmp file, you should see CAM_Iic_0, CAM_Iic_1, Cam_Ctrl_0, Cam_Ctrl_1, hdmi_0, and dvma_0 already installed. These are all custom peripherals that the project was set up with already. This shows that the

FPGA will link correctly to the camera when programmed, and there's no need to tinker with the settings since everything has been preconfigured.

Step 3: Export the Design to the SDK.

Click on Project -> Export Hardware Design to SDK -> Export & Launch SDK. This may take up to 30 minutes so bring a book.

Step 4: Set Xilinx SDK workspace to the directory you unzipped VmodCam/proj. Create an empty application project ("HelloCamera" in this case) and import TestApp_VmodCam/src files into it.

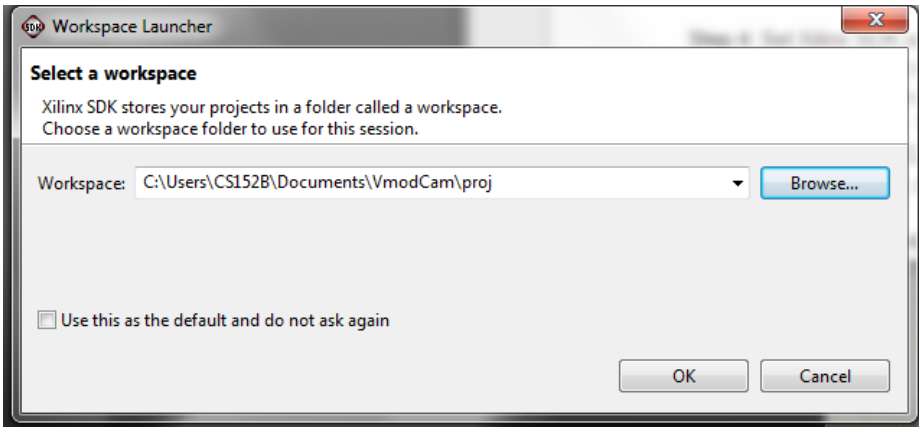

Figure 2: Xilinx SDK workspace path

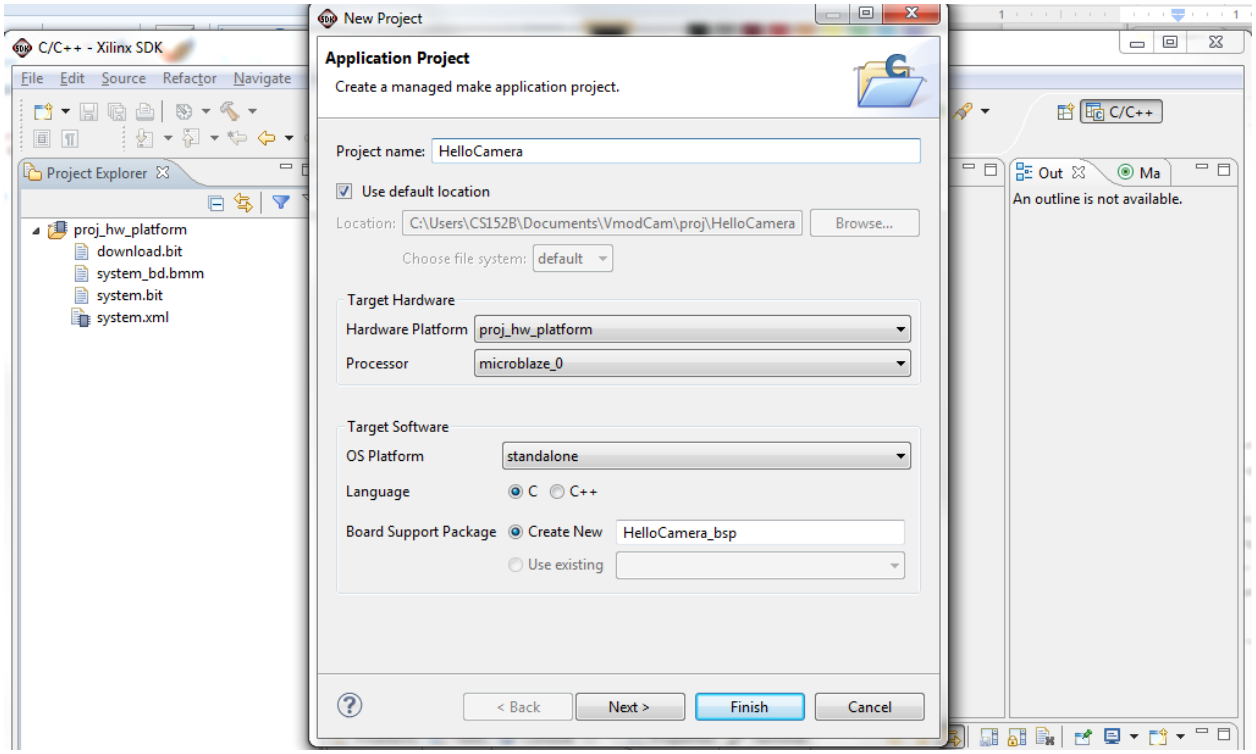

Figure 3: Create new application project

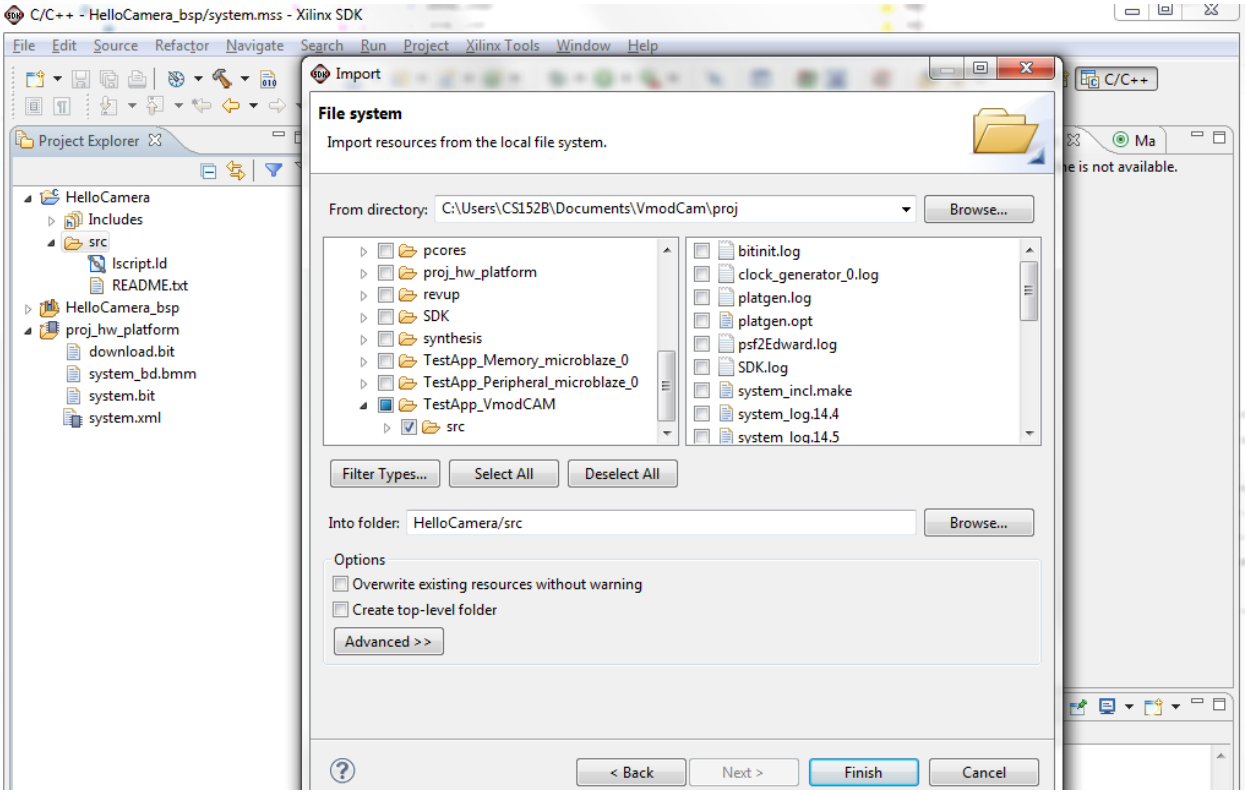

Figure 4: Import src files from TestApp_VmodCAM

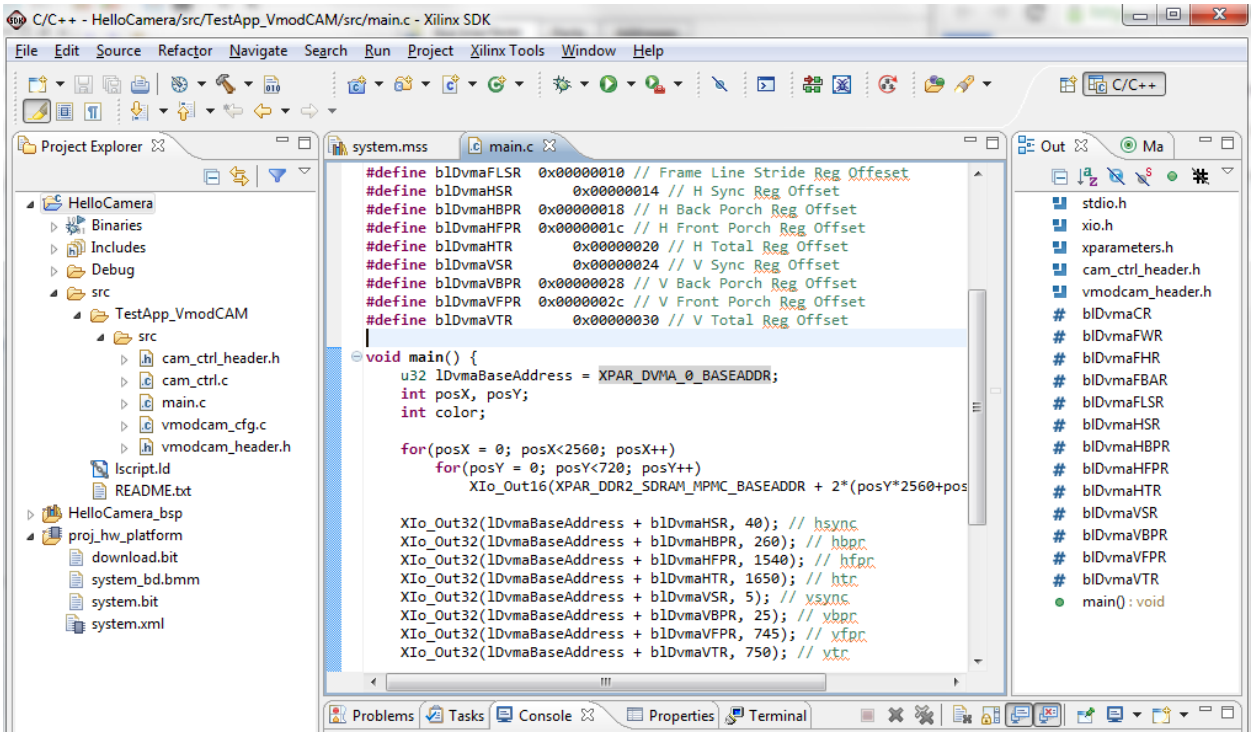

Figure 5: main.c from TestApp_VmodCAM

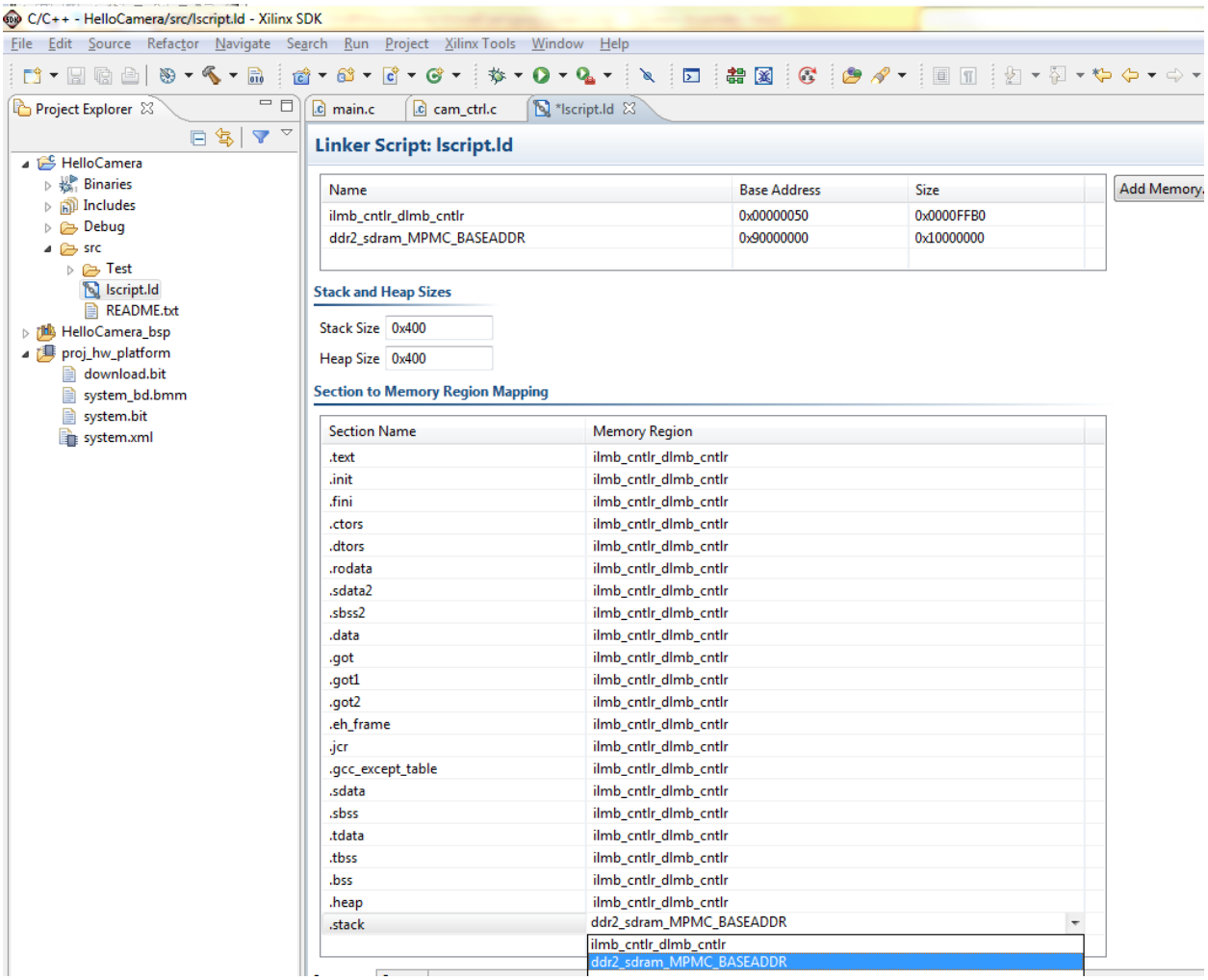

Figure 6: Change memory region mapping

If the workspace path has been set correctly, you should be able to easily import the three projects TestApp_Memory_microblaze_0, TestApp_Peripheral_microblaze_0, and TestApp_VmodCAM.

File->New Application Project ->Type in name of project (“HelloCamera” for example)->Next -> Empty Application -> Finish.

Right click new project folder ->Import -> General/File Systems -> Use path to VmodCam/proj and check TestApp_VmodCAM to import files into new project.

Go into the lscript.ld file and change the memory regions to ilmb_cntlr_dlmb_cntlr, or else when you try to run the program it runs out of memory.

Step 5: Setup your board, program the FPGA, and run the program

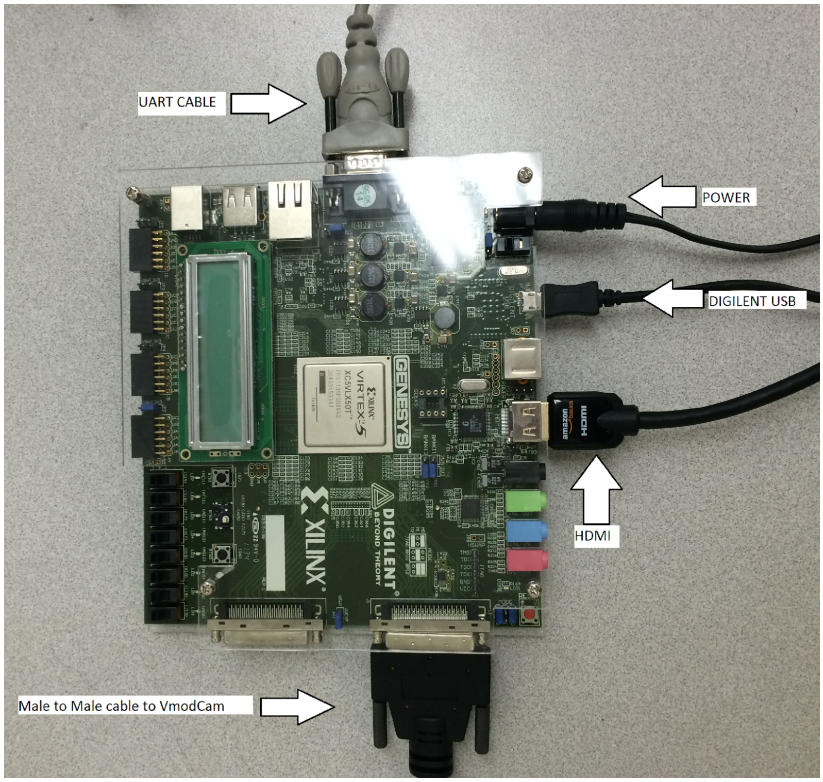

Figure 7: FPGA board setup

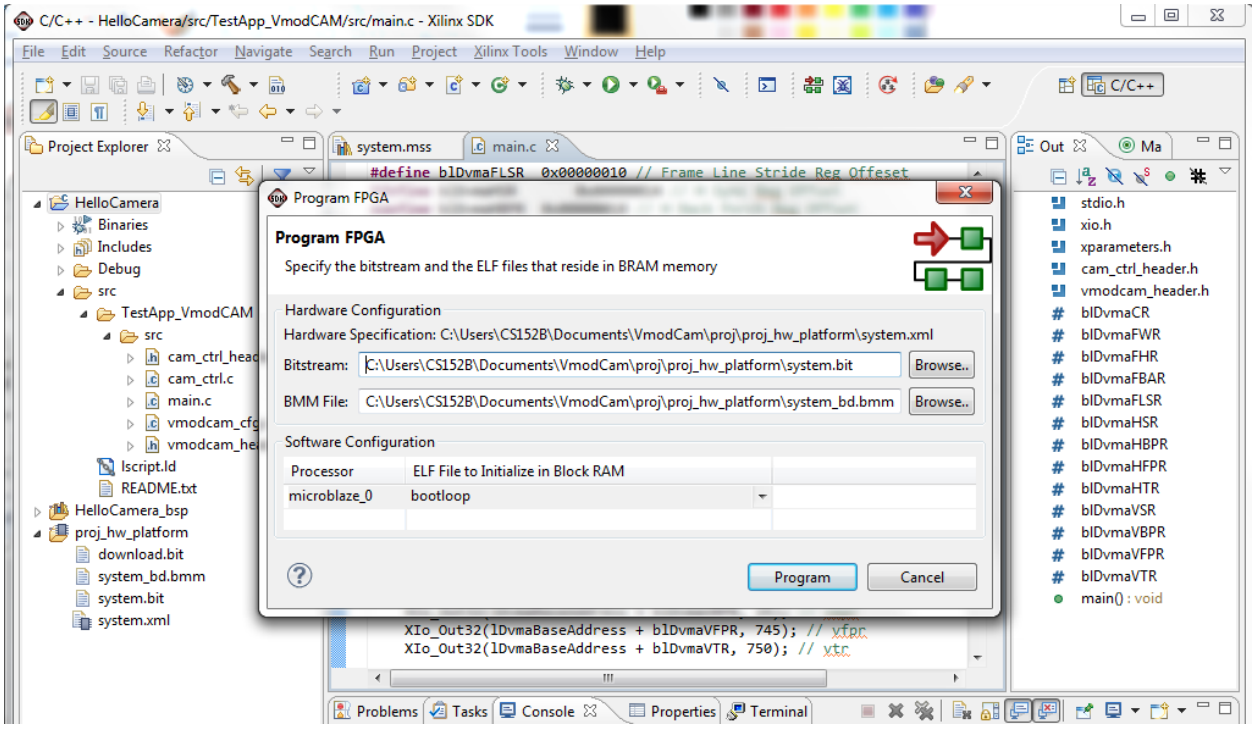

Figure 8: Program FPGA with XPS system bit and bmm files

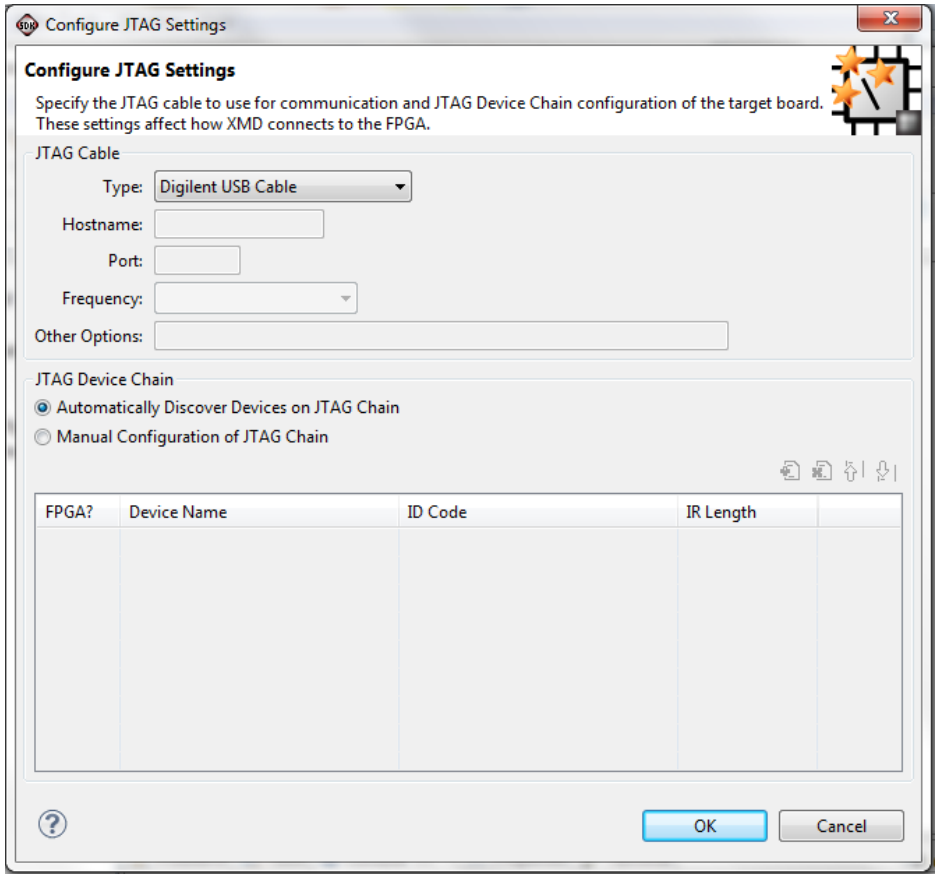

Figure 9: JTAG Settings Configuration

Your FPGA board should have a Digilent USB cable to program the board with our C code, as well as an HDMI-DVI cable to connect the camera output to an external display. Once all these cables are connected, and the FPGA board is on, run Xilinx Tools->Program FPGA with the system.bit and system.bmm files that were generated from step 3. If it says 'Failed to Open JTAG Cable' go to Xilinx Tools->Configure JTAG Settings -> Select Digilent USB Cable in the drop down menu.

Finally, click Run->Run on Hardware(GDB) and watch as the external monitor displays what is coming in from the camera lenses.

The whole process from pressing Run to having the images show up should take around 5 minutes. If you're still waiting after 7 min, something is probably wrong.

Figure 10: What the gradient pattern should look like when running

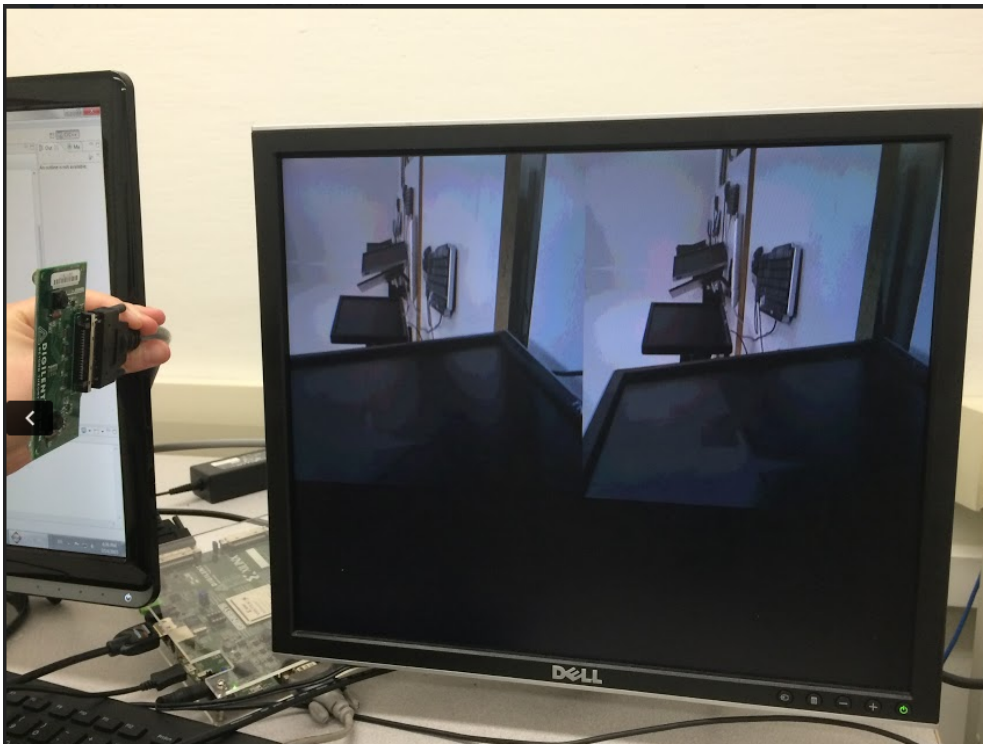

Figure 11: Views from Camera 0 and Camera 1

Source Code Details

The source code is pretty well commented, but here's a few pointers to hopefully make it more clear.

main.c

```
for(posX = 0; posX<2560; posX++)
    for(posY = 0; posY<720; posY++)
        XIo_Out16(XPAR_DDR2_SDRAM_MPMC_BASEADDR + 2*(posY*2560+posX), (posX/40)<<4);

XIo_Out32(lDvmaBaseAddress + blDvmaHSR, 40); // hsync
XIo_Out32(lDvmaBaseAddress + blDvmaHBPR, 260); // hbpr
XIo_Out32(lDvmaBaseAddress + blDvmaHFPR, 1540); // hfpr
XIo_Out32(lDvmaBaseAddress + blDvmaHTR, 1650); // htr
XIo_Out32(lDvmaBaseAddress + blDvmaVSR, 5); // vsync
XIo_Out32(lDvmaBaseAddress + blDvmaVBPR, 25); // vbpr
XIo_Out32(lDvmaBaseAddress + blDvmaVFPR, 745); // vfpr
XIo_Out32(lDvmaBaseAddress + blDvmaVTR, 750); // vtr

XIo_Out32(lDvmaBaseAddress + blDvmaFWR, 0x00000500); // frame width
XIo_Out32(lDvmaBaseAddress + blDvmaFHR, 0x000002D0); // frame height
XIo_Out32(lDvmaBaseAddress + blDvmaFBAR, XPAR_DDR2_SDRAM_MPMC_BASEADDR); // frame base addr
XIo_Out32(lDvmaBaseAddress + blDvmaFLSR, 0x00000A00); // frame line stride
XIo_Out32(lDvmaBaseAddress + blDvmaCR, 0x00000003); // dvma enable, dfl enable
```

The double for loop basically colors your external display. The first parameter in XIo_Out16() represents the position you're writing to, starting from the top left corner of the external monitor. If you wanted to collect pixel data from the image being displayed by the camera, you could use this double for loop with different height and widths corresponding to the dimensions of your image. The second parameter is the color, so changing this to 0 would make an entirely black monitor. You shouldn't need to change the rest of the code since its all camera setup, ignore it.

```
CamIicCfg(XPAR_CAM_IIC_0_BASEADDR, CAM_CFG_640x480P);
CamIicCfg(XPAR_CAM_IIC_1_BASEADDR, CAM_CFG_640x480P);
CamCtrlInit(XPAR_CAM_CTRL_0_BASEADDR, CAM_CFG_640x480P, 640*2);
CamCtrlInit(XPAR_CAM_CTRL_1_BASEADDR, CAM_CFG_640x480P, 0);
```

This code actually starts to power and configure the camera lens itself. The first parameter is the address of the camera and the second parameter is the configuration to produce 640x480 displays for each camera. The third parameter is the starting position to output the image, so the camera_0's image is displayed starting at position x=1280, y=0 on the external monitor and the other camera starts from x=0, y=0.

vmodcam_cfg.c

```
case CAM_CFG_640x480P:
    xil_printf("Current Settings: Context A, 640x480P ...\r\n");
    CamIicSingleWrite(lIicBaseAddress, 0x338C2702); // Context
    CamIicSingleWrite(lIicBaseAddress, 0x33900000); // A
    CamIicSingleWrite(lIicBaseAddress, 0x338CA103); // Sequencer Var: cmd
        (R0x0003)
    CamIicSingleWrite(lIicBaseAddress, 0x33900001); // Preview
    CamIicSingleWrite(lIicBaseAddress, 0x338C2703); // Output width context A
    CamIicSingleWrite(lIicBaseAddress, 0x33900280); // 640
    CamIicSingleWrite(lIicBaseAddress, 0x338C2705); // Output height context A
    CamIicSingleWrite(lIicBaseAddress, 0x339001E0); // 480
    CamIicSingleWrite(lIicBaseAddress, 0x338CA103); // Sequencer Var: cmd
        (R0x0003)
    CamIicSingleWrite(lIicBaseAddress, 0x33900006); // Refresh Mode
    break;
```

This specific code snippet is found in CamIicSetupPicFormat(), lines 210-222, though the following explanation applies to all the cam_cfg files in the project.

To set up the camera config, you have to write to certain addresses with valid values. For example, the address 0x338C2702 should actually be read as R[0x338C] = 2702. This way of setting registers is mentioned in the Digilent VmodCam reference manual, starting at page 3 where it states, “Driver variables can be accessed via two hardware registers, R[0x338C] and R[0x3390]. To access a variable, its address first needs to be written to R[0x338C], which is a standard two-wire register write. Then, reading register R[0x3390] reads the variable and writes a value to it, thus setting the variable to that value.”

In this way you can change the camera settings by first accessing the variable R[0x338C] = A103 or 0x338CA103, and then writing a value to it in the next command, R[0x3390] = 0006 or 0x33900006. This writes the refresh mode shown in the example, though if you changed the value to 0x33900003, the camera would go to standby mode and freeze.

xparameters.h

xparameters.h is a file with all the register addresses mapped to variables that is dynamically generated upon loading the bitstream onto the board. You’ll want to refer to this to control peripheral settings and determine where information is being stored. Remember, the addresses are automatically set by the values you inputted during the XPS program. If you arbitrarily change the addresses in the header file, your program will be screwed up because your FPGA board is programmed by the system.bit file which has different base addresses. So don’t touch it.

You can use the XIo_In16 function to read the information at any of the given addresses in xparameters. The function will return the information in a two-byte chunk unsigned variable. Use the VModCamera manual linked above to find the locations of certain key addresses such as the screen. Generally, the format of your request should be as follows: XIo_In16(some_base_address + offset).

Other things to keep in mind:

- If you change the code, but they do not seem to be having any effect, try reconfiguring the FPGA with the bitstream and then clicking Run on Hardware.
- Close other projects if you have multiple in the workspace because Run gets confused.
- Use xil_printf statements as debugging tools to check if the program has frozen or is just taking forever.
- To see the COM output, go to Run Configurations -> STDIO Connection -> Connect STDIO to Console -> Should automatically have correct port number based on USB connection and change BAUD rate to 115200 (115200 is set in the XPS, RS232_Uart_0 peripheral. Right click -> Configure IP -> Change baud rate if you want to play around)
- A good way to see if you have certain peripherals working is to simply go File->New Application Project -> Next -> Peripheral Tests in Xilinx SDK. This generates a project that has some test code for all your current peripherals, so you have somewhere to start before writing your own test cases.
- Bring books or other materials to occupy your time because running this project takes forever. D:

… well, good luck! :)