D8M Camera for the DE1-SoC Board Tutorial

Introduction

You can find a Quartus project for the camera on Canvas, called *Camera_Reduced.zip*. The project folder for the camera works without modification. You should plug it in and verify that everything is working correctly.

Connect your DE1 board to a monitor using the VGA output, just as you did in previous labs. Then, connect the camera module to the board, using the exposed 20-pin GPIO headers on the edge of the board, as shown in Figure 1. Your camera will come with a ribbon cable, which you can use to extend this connection if you would like.

Figure 1: *Image showing the D8M Camera properly connected to the DE1-SoC board*

Open *DE1_SOC_D8M_RTL.qpf* and compile the project. You might have a notice about upgrading your Altera IP; you can ignore this. Program the project onto the board. A live feed from the Camera should be displayed on the monitor.

There's a lot going on in this project. To start, you'll probably want to check out the top-level module, found in *DE1_SOC_D8M_RTL.v*. This module instantiates ~16 other modules and connects them to each other and to the inputs/outputs of the DE1 board. You should look through this yourself to get some familiarity with what's happening here.

In particular, observe the instance of the *Filter* module. The code for this is shown below:

```
// The signals from the system to the filter.
wire          pre_VGA_BLANK_N;
wire [7:0]    pre_VGA_B;
wire [7:0]    pre_VGA_G;
wire          pre_VGA_HS;
wire [7:0]    pre_VGA_R;
wire          pre_VGA_SYNC_N;
wire          pre_VGA_VS;
// The signals from the filter to the VGA
wire          post_VGA_BLANK_N;
wire [7:0]    post_VGA_B;
wire [7:0]    post_VGA_G;
wire          post_VGA_HS;
wire [7:0]    post_VGA_R;
wire          post_VGA_SYNC_N;
wire          post_VGA_VS;

Filter #(.WIDTH(640), .HEIGHT(480))
   filter (.VGA_CLK(VGA_CLK),
      .iVGA_B(pre_VGA_B), .iVGA_G(pre_VGA_G), .iVGA_R(pre_VGA_R),
      .iVGA_HS(pre_VGA_HS), .iVGA_VS(pre_VGA_VS),
      .iVGA_SYNC_N(pre_VGA_SYNC_N), .iVGA_BLANK_N(pre_VGA_BLANK_N),
      .oVGA_B(post_VGA_B), .oVGA_G(post_VGA_G), .oVGA_R(post_VGA_R),
      .oVGA_HS(post_VGA_HS), .oVGA_VS(post_VGA_VS),
      .oVGA_SYNC_N(post_VGA_SYNC_N), .oVGA_BLANK_N(post_VGA_BLANK_N),
      .HEX0(HEX0), .HEX1(HEX1), .HEX2(HEX2), .HEX3(HEX3), .HEX4(HEX4), .HEX5(HEX5),
      .LEDR(KEDR), .KEY(KEY[1:0]), .SW(SW[8:0]));

assign VGA_BLANK_N = post_VGA_BLANK_N;
assign VGA_B = post_VGA_B;
assign VGA_G = post_VGA_G;
assign VGA_HS = post_VGA_HS;
assign VGA_R = post_VGA_R;
assign VGA_SYNC_N = post_VGA_SYNC_N;
assign VGA_VS = post_VGA_VS;
```

This module is added **between** the camera modules and the output to the VGA monitor. Notice the seven `assign` statements at the end of this excerpt. These are the VGA outputs of the top-level module. You can add your own code to modify these to fit your needs. For a simple example, you could use one of the switches on your board to change the value of VGA_B from its current assignment to zero. The switch would then remove all of the blue from the image being displayed. Similarly, you could use a switch to swap between the camera output and your own graphics.

Recall that the old VGA driver you used only drew black and white pixels. That's because the driver set the red, green and blue outputs to be all either 0xFF or 0. These corresponded to drawing white pixels and black pixels, respectively. Here, if you want to create your own graphics, you'll have to set all these colors yourself. Check out the outputs of the *VGA_Controller* module if you want access to (x, y) coordinates.

With this, you should have enough information to get started using the camera to fit your individual needs. There's a lot going on, but you probably don't need to worry about most of it.