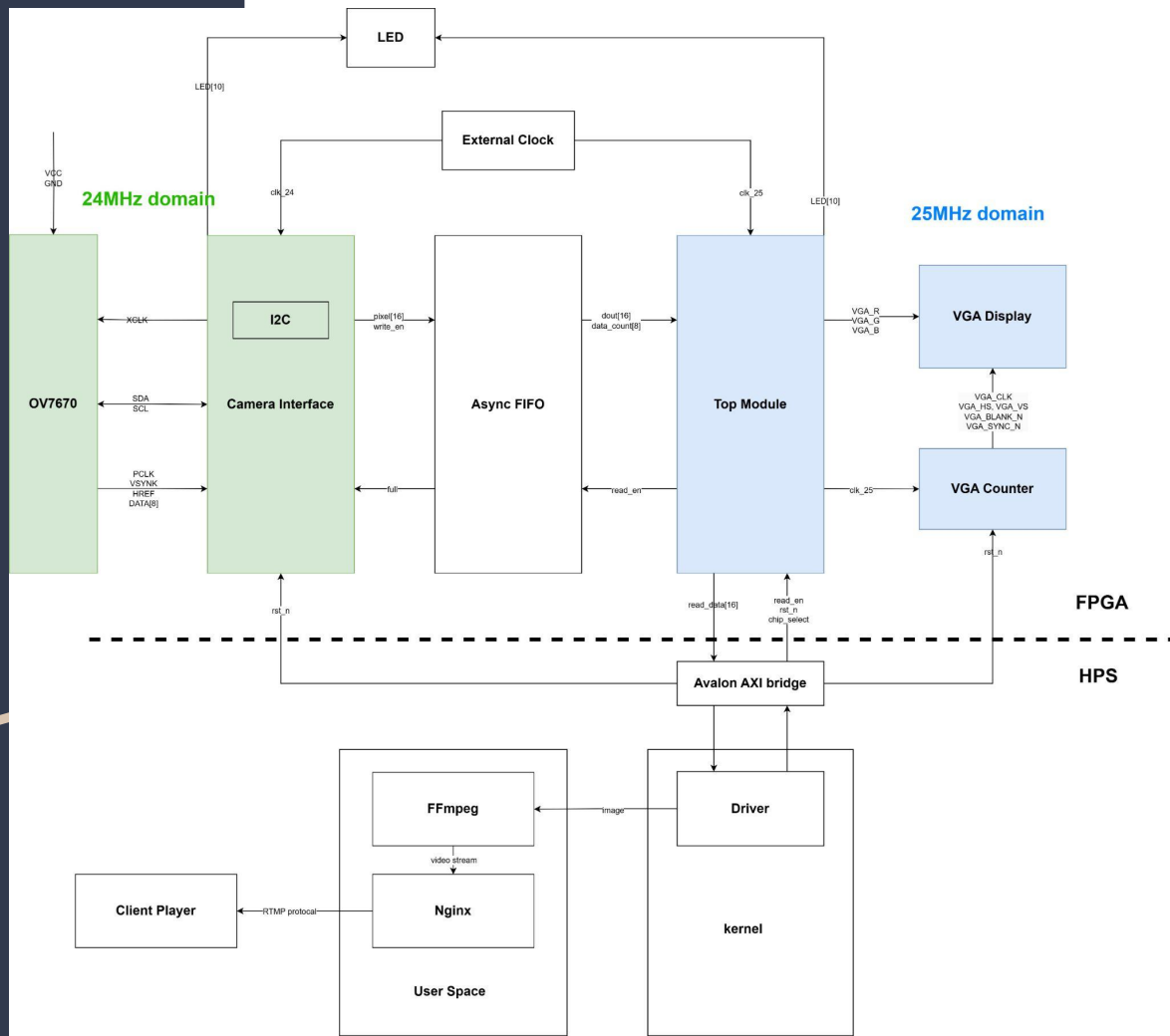


# Web Camera

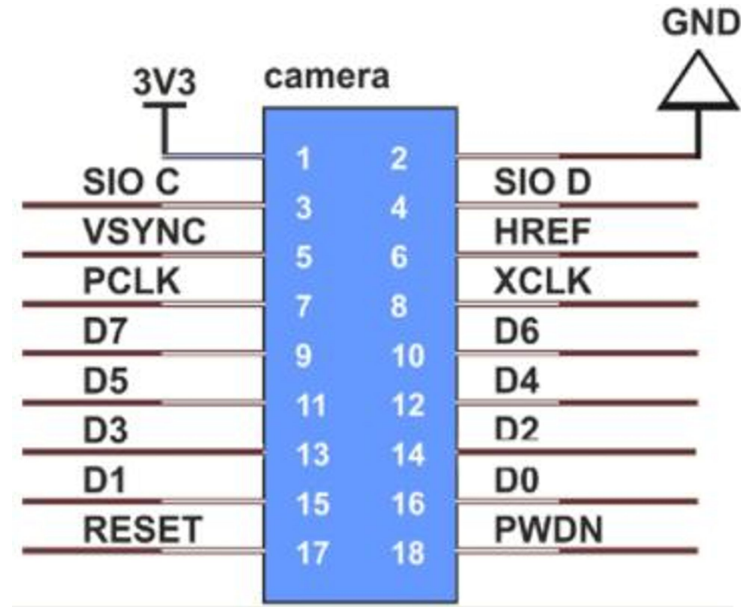
Final Presentation

# Block Diagram



# Connecting OV7670 Camera

- Communicate via SCCB protocol (subset of I2C)
- Send initialization signals to make it output RGB565 signal
- Read 2 consecutive cycles to data to get 16-bit pixel



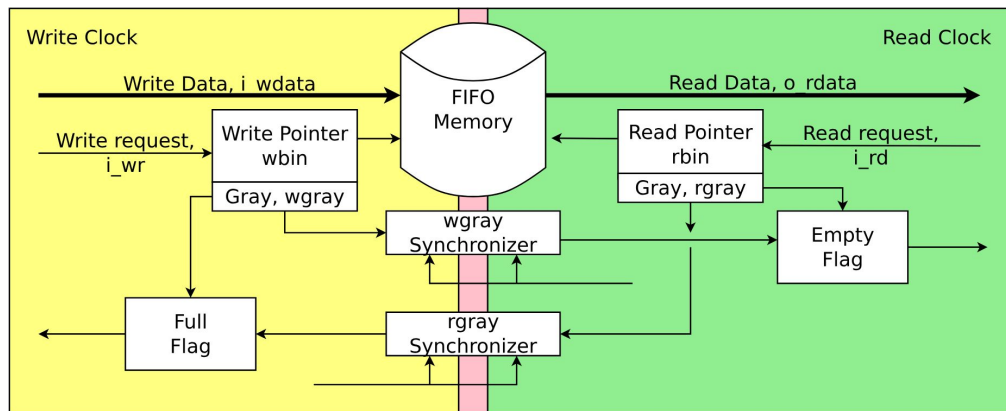
# Connecting 2 frequency domains

- FIFO needs two flags: **full** and **empty**
- Comb. logic uses write pointer and read pointer to judge the two flags
- This logic is not stable when write pointer and read pointers are clocked by two different clocks (in this case 24MHz and 25MHz)

```
assign empty = (w_ptr == r_ptr);  
assign full  = (w_ptr[ADDR_WIDTH] != r_ptr[ADDR_WIDTH])&&(w_ptr[ADDR_WIDTH-1:0]  
== r_ptr[ADDR_WIDTH-1:0])
```

# Connecting 2 frequency domains

- Idea 1: use gray counter so it will be more stable
- Idea 2: introduce middle signals, clocked by the other clock



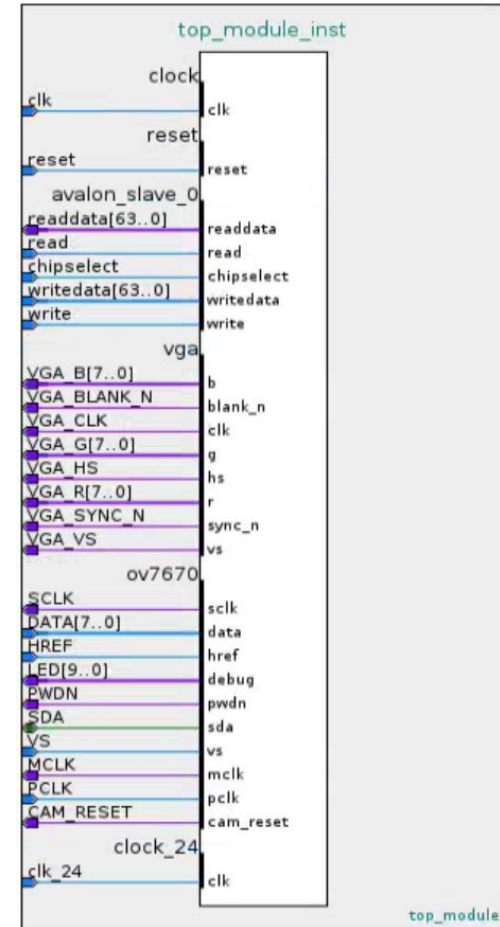
# Output to VGA (for testing)

- Inherit from lab3
- Bouncing ball still functioning (a bouncing ball stacking over the video)
- Use cascading to get RGB888 signal from RGB565 signal

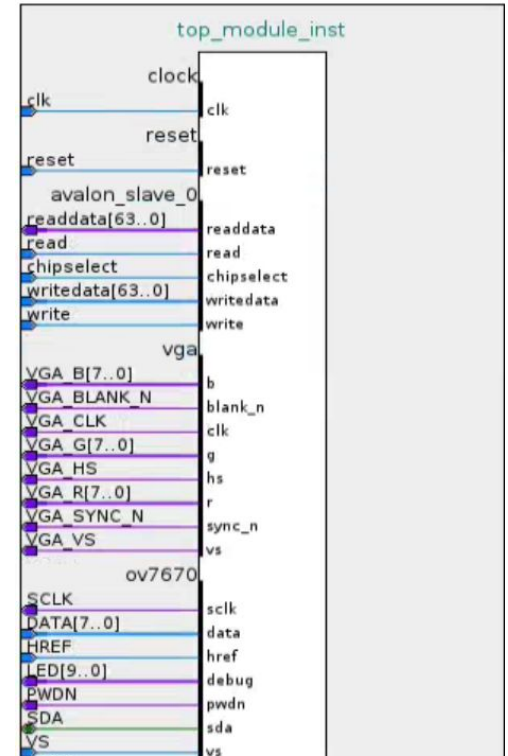
```
r_8 = {r_5, r_5[2:0]};  
g_8 = {g_6, g_6[1:0]};  
b_8 = {b_5, b_5[2:0]};
```

# Signal Interface and Component Connection

clk_reset	Reset Output	Double-click to		
hps_0	Arria V/Cyclone V Hard Proce...	Double-click to		
h2f_user1_clock	Clock Output	Double-click to	hps_0_h2f_user1_clock	
memory	Conduit	hps_0		
hps_io	Conduit	hps_0		
h2f_reset	Reset Output	Double-click to		
h2f_axi_clock	Clock Input	Double-click to	clk_0	
h2f_axi_master	AXI Master	Double-click to	[h2f_axi_clock]	
f2h_axi_clock	Clock Input	Double-click to	clk_0	
f2h_axi_slave	AXI Slave	Double-click to	[f2h_axi_clock]	in
h2f_lw_axi_clock	Clock Input	Double-click to	clk_0	
h2f_lw_axi_master	AXI Master	Double-click to	[h2f_lw_axi_clock]	
f2h_irq0	Interrupt Receiver	Double-click to		
f2h_irq1	Interrupt Receiver	Double-click to		
top_module_0	top_module	Double-click to	clk_0	
clock	Clock Input	Double-click to	[clock]	
reset	Reset Input	Double-click to	[clock]	in 0x0000_0000
avalon_slave_0	Avalon Memory Mapped Slave	Double-click to	[clock]	
vga	Conduit	Double-click to	clk_1	
ov7670	Conduit	Double-click to	clk_1	
clock_24	Clock Input	Double-click to		
clk_1	Clock Source	clk_0	exported	
clk_in	Clock Input	reset_0		
clk_in_reset	Reset Input	Double-click to		
clk	Clock Output	Double-click to	clk_1	
clk_reset	Reset Output	Double-click to		



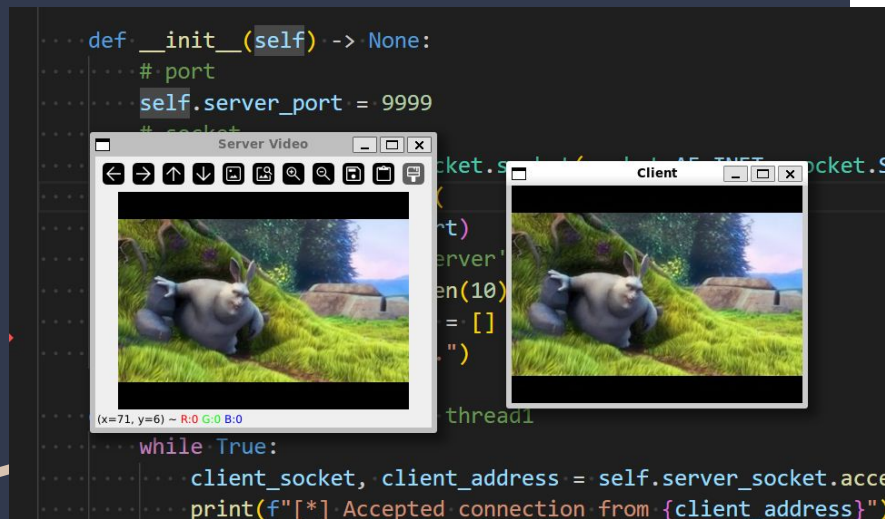
# Signal Interface and Component Connection



```
output logic [63:0] readdata, // {12'd0, column_num, row_num, endOfField, 15'd0, dout}
input logic read,
input logic [63:0] writedata, // {32'd0, x, y}
input logic write,
```



# Sending frames over the network



- Server
  - Running on DE1-SoC board (testing on Ubuntu 20.04)
  - Establish TCP connection with clients
  - Use cv2 to encode the image to JPEG and serialize
  - Send to client at 30fps (320x240 test video)
- Sender
  - Read pixel data using ioctl calls to the camera driver
  - Send frames to server
- Client
  - Running on WSL Ubuntu 20.04
  - Receive frames, decode and display with cv2

# Video Driver

For the camera reading, it utilize `ioread32` to fetch data from the camera

```
return ioread32((dev.virtbase));
```

In the sender:

**Frame Reading:** Utilizes a while loop to continuously read pixel data using `ioctl` calls to the camera driver, pixel by pixel, illustrating how raw image data is processed

**Frame Sending:** After capturing a frame, it immediately sends this data to a server using the established TCP connection, showcasing a real-time data streaming application.

```
// Extract fields from the info
hcount = (info >> 29) & 0x3FF; // Extract bits 29-38 (10 bits for hcount)
vcount = (info >> 19) & 0x3FF; // Extract bits 19-28 (10 bits for vcount)
pixel = info & 0xFF; // Extract bits 0-7 for the pixel value (dout)

// Calculate the address
address = vcount * 640 + hcount;

if (address < READ_PIXELS) {
    buf[address] = pixel;
}

// Check for end of field
if (info & (1 << 18)) {
    break; // Exit the loop when end of field is encountered
}
}
```

Demo