Design Document for Network Security Camera

Carlos D. Nunez-Huitron, James Phan, Michael Lee, Patricio Tapia, Kenny Martinez

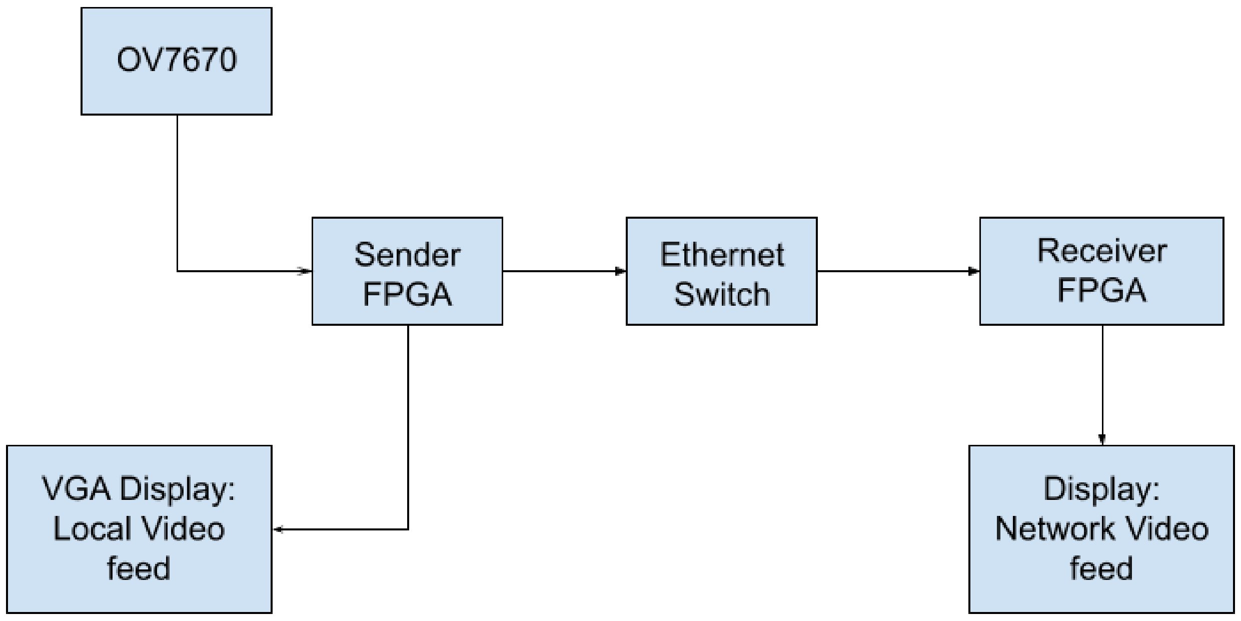
{cdn2128, bnp2113, ml4719, pet2119, km3734}@columbia.edu

Spring 2023

Contents

1. Introduction

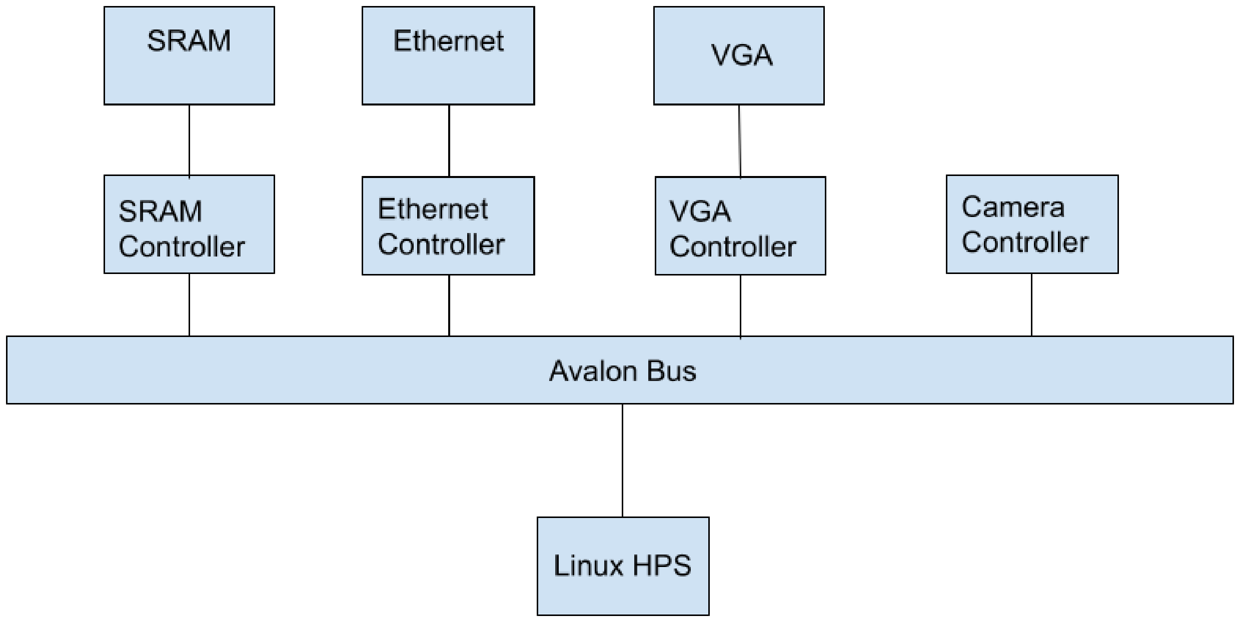
We would like to communicate a video feed taken using a OV7670 camera module from one DE1 FPGA to another DE1, displaying the video on both ends. We plan to use an ethernet connection to pass information from the first FPGA to the second. The figure below illustrates what we would like to accomplish on a high level:



The OV7670 camera module would interface with the “Sender” FPGA using the SCCB protocol - a cousin of I2C that lacks ACK bits. Video collected from the camera module is both passed to a VGA monitor for display and also sent via UDP packet through an Ethernet cable using IP to the “Reciever” FPGA.

2. System Block Diagram

The planned hardware architecture of the receiver FPGA is as follows:



The camera module produces pixels compatible to VGA format frames where each pixel is represented by a byte. Storing a frame buffer of the entire camera output (640 x 480 = 307,200 pixels equivalent to 307,200 bytes in our case) pushes us to the edge of the embedded memory (~500,000 bytes) but should be possible. This frame buffer can be passed along to the VGA controller to be displayed. We plan to use a version of the VGA controller similar to the one used in Lab 3 to handle the display. To do this, the camera controller will be connected to an SRAM controller in QSYS and set up to feed the memory controller one byte at a time. The SRAM controller will save this byte in its own slot in memory, effectively keeping track of which pixel it belongs to. When the camera gives a high VSYNC signal, the frame is completed and the Camera controller would produce a flag in another section of memory to notify our user program. At that point, the program grabs the entire frame data from memory for display (to the VGA controller) and distribution to the Reciever FPGA.

However, we must also consider the limits of our Ethernet connection. The largest packet size for UDP is 65,507 bytes yet we have 640 x 480 = 307,200 pixels to communicate (each pixel represented by a byte). In other words, the largest possible packet could only store approximately 20% of a full frame. Instead of sending an entire frame, one option is to split up a frame into five packets equal to 307,200/5 = 61,440, sending each as they are filled in SRAM. When a fifth of a frame has been collected, it can be transmitted to the reciever. Alternatively, the simpler solution is to chop up each frame into five sections and transmit each in turn but this is an approach that might well be inefficient. The Receiver FPGA will be similar in every way except it will not have the camera module. The Reciever is only concerned with recieving UDP packets and displaying those frames on its own VGA display.

3. Resource Budget

Our most tightly controlled resource will be memory. Saving a frame buffer consisting of every pixel will tightly constrain our embedded storage. Another concern is the bandwitdth of the Ethernet connection. If the Reciever FPGA is not able to grab, interpret and process data in the UDP packets fast enoough, there might be a situation where the Reciever fails to display packets sent resulting in choppy video. Because memory usage is nearly symmetric between both FPGA's, it might be difficult but not impossible to find space for additional buffering on the Recieving end to catch dropped packets.

4. The Hardware/Software Interface

The OV7670 module our team has acquired consists of 18 pins. Interface pins will be connected with jumper wire from the camera to the GPIO pins of the Transmitter board.

Additionally, 3.3V power and ground should also be given to the camera through lab power supplies. The camera controller is a peripheral and implements an SCCB protocol to talk with the OV7670 module. Part of the boot up sequence required for the OV7670 is a process to configure registers on the camera module for frame queries[1]. Registers can be addressed and written to on the SIO_D line as a combination of four hex digits. The first two denote the register to be written to and the last two hex values refer to the setting those registers should take on. This start up sequence needs to be implemented as a finite state machine that configures a new register, every clock cycle until they are all configured.

After configuration, data from the camera comes with a few flags courtesy of the HREF and VSYNC signals. When HREF is high, subsequent bytes sent through the 8 pins represent would represent RGB values of a row of pixels. A pulse of VSYNC signals that the camera is done transmitting a frame. The camera controller will store a positive flag to a slot in memory once it sees a VSYNC and promptly negate that flag once the camera moves on. The camera module passes on valid packets to the SRAM controller but does not track them. The SRAM controller will increment an internal address to a block of 307,200 bytes everytime it is given a new packet to store before storing said packet. After the VSYNC flag has been raised from the camera (passed on through the camera module), the internal address for the SRAM controller will be reset. The VGA controller will work the same way in Lab 3 . In the user space, a program will continuously poll the memory address assigned to the camera controller responsible for keeping track of VSYNC. If VSYNC is high, the program will read from the block memory containing the frame. After the frame has been read in, the program distributes this data to the VGA display and to a seperate function that sends the data via Ethernet.

[1] 11, “OV7670_2006.pdf”, http://web.mit.edu/6.111/www/f2016/tools/OV7670_2006.pdf

On the Recieving side, code will be written to process UDP packets from the Transmiter FPGA and call an identical VGA driver to display the image.

IGNORE BELOW - DO NOT PRINT but pls keep because of the comments that help me understand.

to GPIO pins seem to be addressable in System Verilog so we can