UNIVERSITY OF CALIFORNIA AT BERKELEY
COLLEGE OF ENGINEERING
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

CHECKPOINT 2.5
FOUR PORT ARBITER AND USER INTERFACE

1.0 MOTIVATION

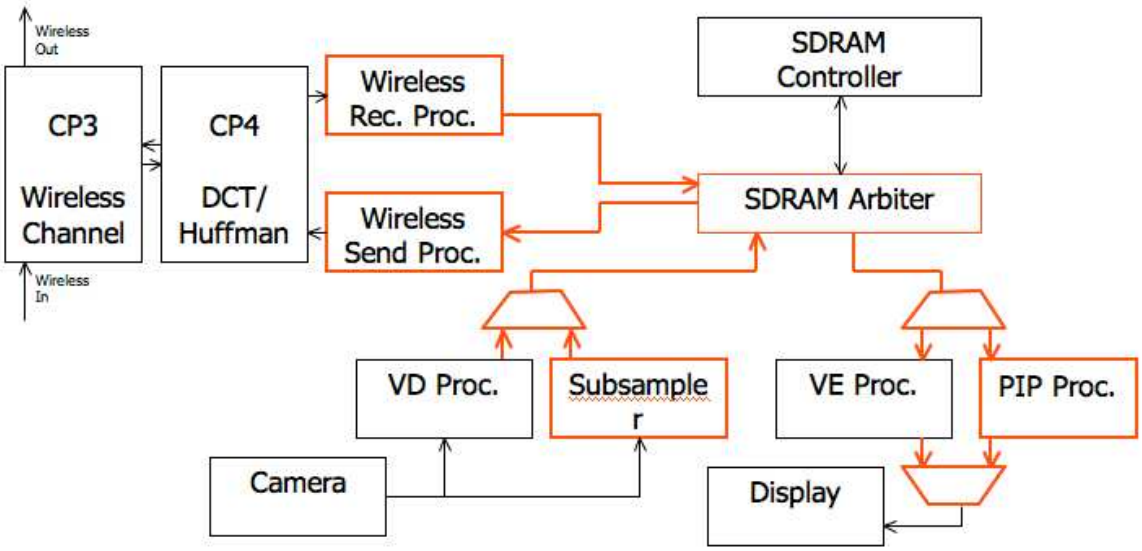
Please note that this checkpoint serves as an early start on Checkpoint 4. It will be worth 5% extra credit on Checkpoint 3 if turned in along with Checkpoint 3.

In this Checkpoint you will add onto your 2-port arbiter from Checkpoint 2 to add 2 additional ports for wireless sending and receiving. In addition, you will add on additional processors to your previous 2 ports to create an additional VD processor that subsamples the input frame and an additional VE processor for displaying the local subsampled frame and the remote received frame in the 2 top corners of the screen. By the end of this checkpoint, you will have a good portion of Checkpoint 4 completed, leaving you to focus on compression and protocol implementation in the last 2 weeks of the project.

2.0 INTRODUCTION

2.1 Four Port Arbitration

You will need to modify your FSM from Checkpoint 2 to account for any combination of 4 possible request signals from each of the ports. You will need to prioritize these signals in some fashion. Much like with the 2 ports you created in Checkpoint 2, you will need to create address counters and FIFOs at each port to buffer the input/output and ensure that requests are made in the necessary order. The diagram below shows the components needed for this checkpoint.

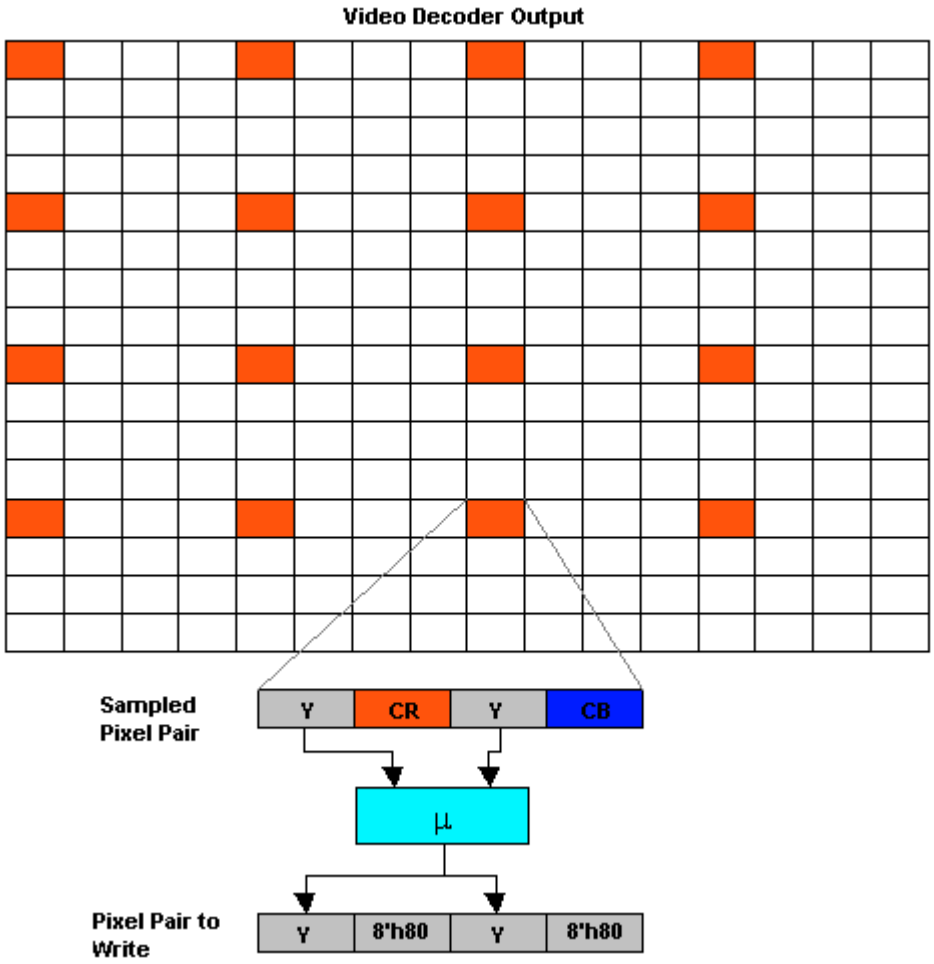


2.1.1 Subsampling Processor

The Subsampling processor is responsible for subsampling a full frame of active video and storing its contents into SDRAM. Because of the limited bandwidth imposed by the wireless transceiver, we do not want to be constantly updating our subsampled frame while the transceiver is in the process of sending that frame out. **Thus, we should only begin sampling our next frame when the wireless is done queuing up and sending the current frame.**

Upon the receipt of the WirelessReady signal, the Subsampling processor will subsample every 4th pixel pair from every 4th line of the video decoder data. In order for you to obtain a subsampled frame of 160 pixels (80 pairs) by 120 lines, you will need to sample an area the size of 640 pixels (320 pairs) by 480 lines from the video decoder. However, because of even and odd lines, this process is actually a little bit more complicated. **Instead of sampling every 4th line from lines 1-480, you should sample every 4th line from lines 1-240 and lines 255-494**.

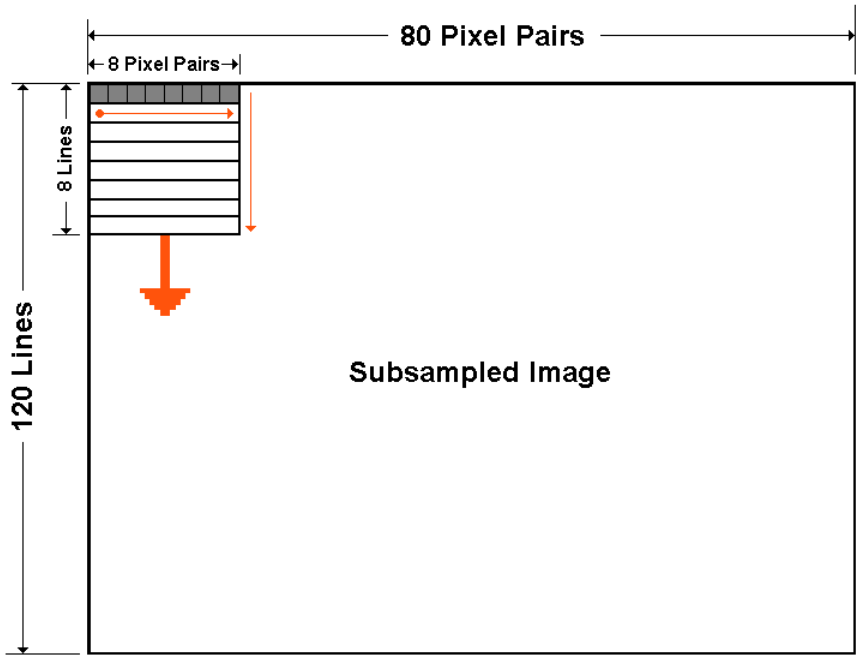
Additionally, we will be reducing the pixel quality to 8 bits of grayscale per pixel pair by extracting the averaged luminance value of every 4th pixel pair. **Each pixel pair you write to SDRAM will have the same luminance value for both the left and the right pixels.** You should also fill the CR and CB values of the pixel pair with 8'h80 (instead of 8'h00) in order to display grayscale instead of a greenish color. These pixel pairs make up your subsampled frame and will be stored in a separate bank in SDRAM that both the Wireless Sending Processor and PIP processor will read from.



2.1.2 Wireless Receiving Processor

This module is responsible for writing the data received over the wireless channel to SDRAM in the correct order.

Wireless packets will arrive as 8x8 squares (used by the DCT / Inverse DCT). In order to have cross compatible solutions in Checkpoint 4, you will need to write the data to SDRAM in the order as follows: first write line 1, pairs 1-8, followed by line 2 pairs 1-8, all the way to line 8 pairs 1-8. It then begins the next 8x8 block with line 9, pairs 1-8 to line 16, pairs 1-8. Once it reaches line 120, it begins the next column of blocks at line 1, pairs 9-16, then line 2, pairs 9-16 and etc. The subsampled frame will contain 160 pixels (80 pairs) by 120 lines. This corresponds to 150 8x8 blocks starting with the top left corner and terminating in the bottom right corner. It will write to its own bank of SDRAM from which the PIP processor will read.



2.1.3 Wireless Sending Processor

Wireless data will also be sent in the same order described in section 2.1.2. The same address counter can be used for both modules. It will pull data from the bank the Subsampling processor writes to. This module is also responsible for telling the subsampler that it has just finished sending out a frame.

2.1.4 PIP Processor

The Picture in Picture processor will need to be added to handle displaying of the small frames to be displayed in the top corners of the screen. It will decide what to display based on the InRequest, InRequestLine and InRequestPair outputs of the video encoder.

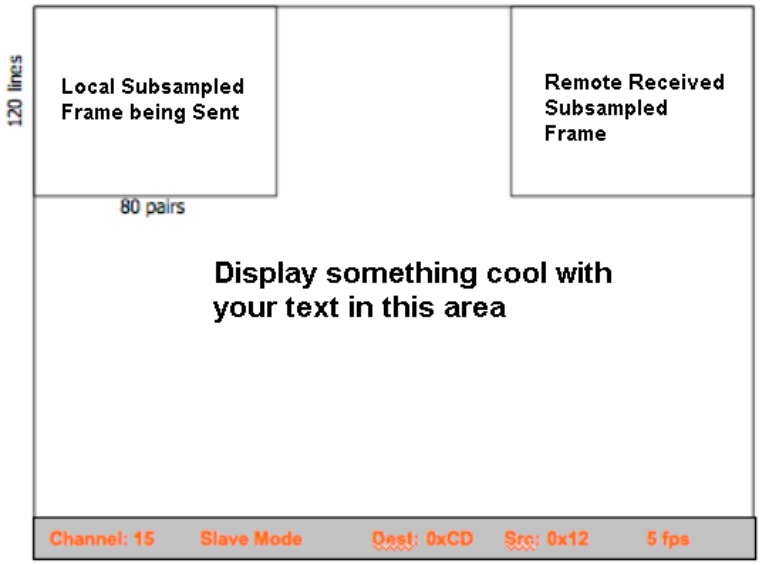
Note: When the InRequest signal is not asserted, the InRequestLine and InRequestPair signals could be anything.

2.2 User Interface and Display

You will have to modify the VE Processor port of the arbiter as shown above to generate data for the Video Encoder that displays the 2 smaller frames in the corners of the screen. In addition you must display the following information:

- Wireless channel number (4 bit)
- Source address (4 bits)
- Destination address (4 bits)
- Operation mode (1 bit - Master / Slave)
- Anything else you think may be useful (ie frames per second)

If you are unsure how these values will be generated, simply make them input ports with the specified number of bits to your graphics / UI module. The screen should look thus something like this at the end of this checkpoint:



Make sure that your method for writing text is flexible as well as reliable, as you will likely be writing lots of text in Checkpoint 4.

2.2.1 Character Display

A character ROM has been given to you in CharROM.v. The module accepts an 8-bit input as an ASCII character code and outputs an 8x8 bit Dot Matrix which can be mapped to a 8 pair x 16 line block of the screen to draw the desired character. It is up to you to design the interface to this module and use it to draw the desired text to the screen. An SRAM module will also be provided in case you want to use it. Feel free to change its size.

2.2.2 Double Buffering

Because of the slow transmission rate of the wireless, you may not want to see the received frame as it is being updated and only update the received frame after it is completely written. You can do this by having the wireless receive and video encoder read/write from alternating places in memory.

3.0 PRELAB

1. Examine the Verilog skeleton provided for this checkpoint.
2. Start your design ahead of time.
    a. Begin with schematics and bubble-and-arc diagrams.
    b. Come prepared for your design review. Make sure you understand how to use the Character ROM and how to draw to different regions of the screen.
    c. The entire checkpoint will require significant debugging. Make sure to at least write a draft of it ahead of time

4.0 LAB PROCEDURE

The best approach to designing a complex digital system is to abstract away details into many layers of operation. This checkpoint will involve a lot of design on your own part and thus it is important that you figure out a clean way to do things before you begin. **Writing "quick-fixes" to make your project work will cause you major problems in the future!** This "mini-checkpoint" will be significantly harder than Checkpoints 1 and 2 combined, so don't put it off until checkpoint 4.

The following tables are **SUGGESTIONS** for input/output port specifications of a particular implementation of the checkpoint

4.1 Graphics.v (modified from CP2)

Signal	Width	Dir.	Description
Video Encoder			
InRequest	1	I	Pulse to request from Video Encoder for the next piece of data
InRequestPair	9	I	Pair requested from Video Encoder (actually used unlike in CP2)
InRequestLine	9	I	Line requested from Video Encoder (also used in this CP)
DataIn	32	O	The data to send to the Video Encoder
Video Decoder			
Valid	1	I	Indicates the Video Decoder DataOut signal has valid data
Line	9	I	Indicates the line number of the data coming from the decoder
Pair	9	I	Pair number from the decoder
DataOut	32	I	The data coming from the video decoder
Wireless Out Processor			
DCTRequest	1	I	Request from checkpoint 4 for the next piece of data.
WirelessProcOut	7	O	The next 7-bit sample to be sent to the DCT
BlockOutIndex	9	O	Index of the 8x8 block currently being sent
BlockOutRow	3	O	The row number within the 8x8 block.
BlockOutCol	3	O	The column number within the 8x8 block
Wireless In Processor			
DCTValid	1	I	Indicates that the data from the DCT module is valid.
BlockInRow	3	I	The row within the 8x8 block from the DCT
BlockInCol	3	I	The column within the 8x8 block from the DCT
BlockInIndex	9	I	The index of the 8x8 block from the DCT
WirelessProcIn	7	I	The data coming in from the DCT.

5.0 HINTS AND TIPS

1. Debug your address counters in Modelsim extensively in order to ensure its functionality. Much of this checkpoint involves designing counters that work properly
2. As you may have learned from Checkpoint 2, looking at camera video data on Chipscope is kind of pointless. Try to come up with "dummy" inputs to feed into your subsampler that will help you debug
3. Depending on your implementation of the SDRAM arbiter and the various processors, you may exceed your SDRAM bandwidth. **Thus, you will NOT be required to display uncompressed full resolution local video for this or any further checkpoints.**
4. It is highly recommended that you test by looping the wireless sending processor directly to the wireless receiving processor. You should also try to artificially slow down frame rates to see how your system behaves under those conditions.
5. As mentioned, many parts of this checkpoint are actually part of checkpoint 4. Some details may be changed in the final project specification for checkpoint 4.