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

Fall 2003 Project
Checkpoint 4, Dummy Vector Overlay

1. Motivation

This is the fourth checkpoint for the Fall 2003 project.

- You will get used to our vector overlay unit
- You will display some dummy vectors on the screen

2. Objective

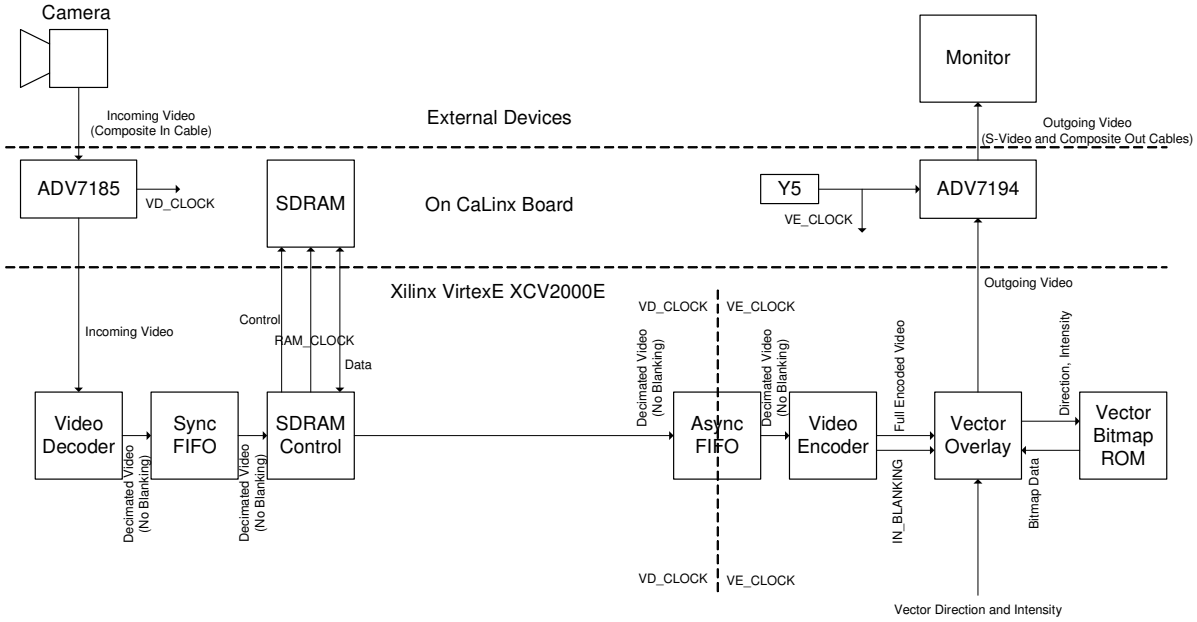
Use our VectorBlend module to display a 39 column by 29 row grid of dummy vectors on top of your full motion video from checkpoint 2. You will need to generate a handful of signals to control the VectorBlend unit properly, as well as connecting a few dipswitches and/or a counter to generate a pattern of dummy vectors.

For the demo you will need to make two modes:

1. Display a 29x39 grid of identical vectors, the direction and intensity should be selectable by dipswitches
2. Display a non-uniform grid of vectors. You should use some kind of counters to display all possible combinations of direction and intensity (some may be duplicated). Note that the pattern should not change from frame to frame.

3. Methodology

3.0 Block Diagram



Given Modules:

- The VectorBlend vector blending module (and a few others in the same file) which will do all the real work of this checkpoint
- Two bitmap ROMs made from blockRAM which contain the vector images
-

Modules You'll Need:

- Everything from checkpoint3
- A counter to generate a test pattern of vectors

You will need to assemble these modules in some kind of sensible fashion according to the block diagram above. Please read the module descriptions, and think carefully before you begin to write verilog. This checkpoint can be completed in about 30min with 8 lines of verilog if you are lucky and remember how your encoder works. Otherwise it should be easily doable during your lab section.

3.1 Vector Blender

Below is a table filled with the input and output signals from the vector blending unit:

Group	Signal	Bits	Description
Vector to Display	Direction	4	Direction of Vector to Display
	Intensity	3	Intensity of Vector to Display
Input from Encoder	Row	4	Row of current vector
	Col	3	Column of current vector (pixel pair)
	Blanking	1	In blanking?
Video	VideoIn	8	Video from encoder
	VideoOut	8	Video to ADV7194
General	Clock	1	VE_CLOCK_

The Direction and Intensity signals are really quite simple, they select the direction of the arrow to display and the intensity to display it at. There are 16 possible directions, some experimentation will show you which is which. The arrows point in all directions with a 22.5° change between two neighboring arrows. The intensity is a little more complex, there are 5 intensities (0, 1, 2, 3, 4 and 5. 6, 7, and 8 are all treated the same as 5). As the intensity increases the arrow will fade in from nothing. That is when intensity is 0, the arrow will not be visible, only the video, at 3 the output will consist of exactly half arrow and half video and at 5 the video will be completely obscured by the arrow. This complicated alpha blending is the reason we are giving you the VectorBlend module.

The VideoIn and VideoOut busses should be self explanatory. The VideoIn should come from your encoder and the VideoOut should go straight to the ADV7194 video encoder chip.

Clock should be equally obvious.

The Blanking signal should be used to disable the VectorBlend module. When Blanking is asserted the VectorBlend module will not modify the incoming video stream (though it will delay it by a few cycles). This should be used to prevent the VectorBlend module from destroying the sync data

(FF,00,00,XX) that must be in the data stream. Without this signal the vector blend module might mix a vector into a sync signal thereby corrupting the video stream.

Note: Blanking will also be needed to disable the VectorBlend module while the borders of the image are being transmitted. Because all of the 8x8 block motion kernels are slightly offset from the edge of the screen (by 4 pixels @ 320x240) the VectorBlend unit should be disabled during these borders so as not to display useless or random vectors.

The final two signals: Row and Col are the most complicated. These two signals tell the VectorBlend unit which pixel pair of the current vector should be displayed RIGHT NOW. Notice that these values are in terms of actual video pixels NOT decimated 320x240 pixels. This means that there are 16 values for Row, even though the vectors are the same size as a kernel, because the kernels are 8x8 at 320x240 and the vectors are 16x16 at 640x480.

Caveats of Row and Col:

- Row must take the alternating fields into account, this means the lowest bit of the row will depend on which field we are in.
- Row should depend primarily on VCOUNT (perhaps with an offset because VCOUNT might not be 0 where we want Row to be 0)
- Remember to take blanking/borders into account properly.
- Col has only 3 bits, it corresponds to PIXEL PAIRS! A column corresponds to one 32bit pixel pair of chroma and luma.
- Col should depend primarily on HCOUNT (perhaps with an offset because HCOUNT might not be 0 where we want Col to be 0)
- Remember, both Row and Col might not use the least significant bits of HCOUNT and VCOUNT. For example, the Col should not change every time a HCOUNT goes up by a byte.
- Remember that the vectors are not sequential! You will need to do the first row of pixels in the first row of vectors, then the second row of pixels in the second row of vectors and so on.

3.2 Video Encoder

You will need to modify your video encoder SLIGHTLY. Changes:

- You will need to extract Row, Col and Blanking for use by the VectorBlend unit.

4 Where to Begin

Begin with your video encoder, you'll need to modify this slightly to extract the needed signals. Notice that even if the signals are not exactly right at first you should be able to see what you did wrong, since you'll still have video output to examine.

You will need to instantiate the VectorBlend module somewhere. You may put it inside your encoder or in FPGA_Top, we don't care.

Start by connection Direction and Intensity directly to the dipswitches. Once you can get a static pattern of 29x39 identical vectors to display on the video and you can change the vectors with dipswitches, then switch to some kind of counter based scheme. This should not be needlessly complex, we just want to see a 29x39 grid vectors with all 65 possible vectors showing in some kind of pattern so that we can see you didn't somehow hardcode the solution, and check that you can actually display different vectors at the same time.

5 Acknowledgements

Original Lab by Greg Gibeling

Checkpoint 4 Check-offs

Name: _____

Name:_____

Lab Section: _____

1. Testbench and Simulation	_____	(20%)
2. Demo		
Uniform grid of 29x39 vectors	_____	(40%)
Counter-Based vector pattern	_____	(40%)
Total	_____	(%)