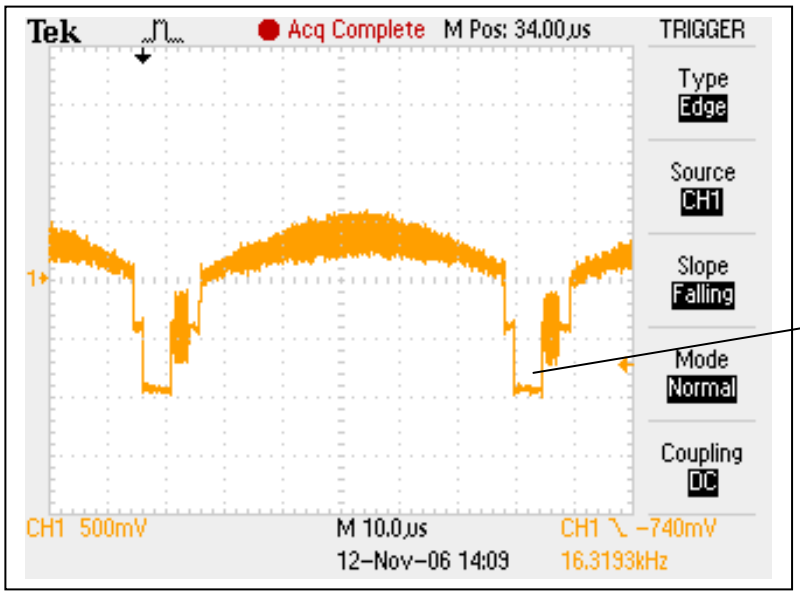
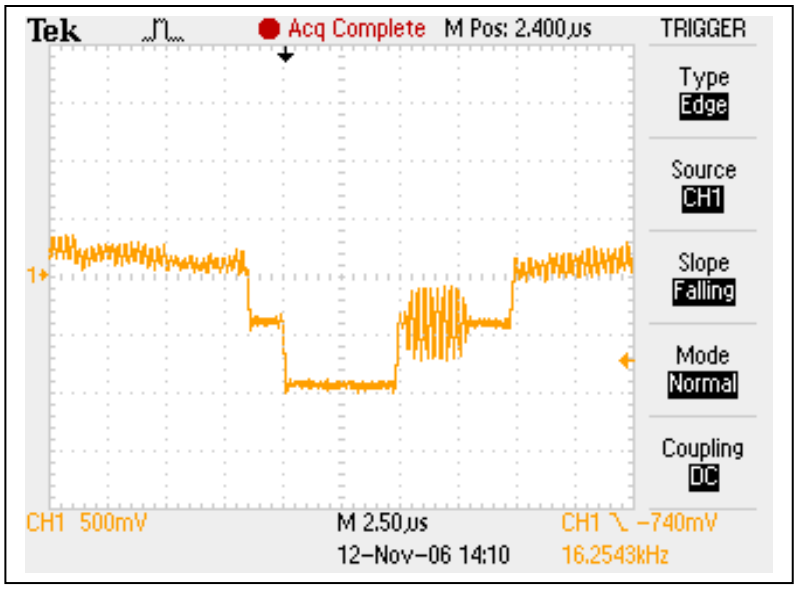
The video cameras used in the 6.111 lab outputs a NTSC (National Television Standards Committee) analog video signal. This signal is used in North America. In Europe PAL (Phase Alternating Line) is used. The video signal contains color (chrominance), intensity (luminance) and synchronization information. For black and white TV's, only the intensity and synchronization information is used.

One horizontal line of NTSC data is shown in the attached oscilloscope image followed by a more detailed display of the horizontal sync pulse



Horizontal Sync



Unlike a PC VGA type display, the NTSC signal is an interlaced signal with 525 scan lines, in two fields of 262.5 lines each. Only 480 lines are visible. Lines 248-263 and 511-525 are blanked on most displays to allow beam to return to the upper left hand corner for the next scan.

The NTSC (analog signal) is converted into digital format with an Analog Devices ADV7185. There are three key timing/information signals:

1. F (Field): 1 indicates even field, 0 indicates an odd field. Recall that the display is interlaced.
2. V: related to vertical sync signal indicating the start of a new frame (note that the frames are even field and old field).
3. H: related to horizontal sync signal indicating the start of a new horizontal line.

See http://www-mtl.mit.edu/Courses/6.111/labkit/appnotes/xapp286_04.pdf for the gruesome details.

Because of bandwidth limitations, there is more luminance data transmitted that chrominance data. The data stream from the ADV7185 is as follows:

```
// The data stream looks as follows
// SAV_FF | SAV_00 | SAV_00 | SAV_XY | Cb0 | Y0 | Cr1 | Y1 | Cb2 | Y2
            | ... | EAV sequence
```

Note that Cr and Cb alternates – ie you get a new Cr and Cb value for everyone other pixel (downsampled). SAV is Start of Active Video, EAV is End of Active Video.

In order to process the NTSC video data, the bits needs to be stored in memory. The ZBT memory on the labkit can be used. The student is encouraged to implement their own Verilog for NTSC to ZBT to VGA display. Because of the complexity of the implementation, the 6.111 staff (Prof Ike Chuang, Javier Castro and Nathan Ickes) have written a sample Verilog which takes b&w NTSC video, stores it in ZBT memory and then displays the video in a 1024x768 window on the monitor.

http://web.mit.edu/6.111/www/f2011/tools/ntsc_zbtfix.zip

This sample code stores the Y value for four pixels (*vr_pixel*) in each ZBT location – hence the grayscale display. To display color, you will need to modify the sample code to store the YCrCb rather than just black and white intensity information to ZBT. The *ntsc_decode* module provides a 30 bit YCrCb. Since each ZBT memory location is 36 bits wide, you can store two pixels worth of data in each location with each pixel consisting of 18 bits of YCrCb data – so extract 18 bits from *ntsc_decode* and modify *ntsc2zbt* to write two 18 bit values. Obviously *vr_pixel* is now 18 bits. To store four 8 bit pixels, memory is written when hcount[1:0]=2. To store two 18 bit pixels, assert write enable when hcount[0]=1

In *ntsc2zbt* each pixel data is now 18 bits wide rather than 8 bits wide. Each memory word is created by shifting in a 18 bit *data1*. The ten bit wide variable ***x*** is the count of each

pixel and used a part of the memory address. Since the lower order two bits are not used, memory address changes for every four pixels. Since each memory location contains only two pixels, memory addressing changes twice as fast. Therefore only the lsb is not used. ***myaddr2*** must now address twice as memory location and should be modified accordingly.

Obviously, *vram_display*, module supply pixels to the VGA display, needs to be be changed accordingly. [For b/w, the memory address is created by ignore the low order two bits of hcount. Why?]

Alternatively, you can convert YCrCb to RGB and send 18bits of RGB per pixel to *ntsc2zbt*. The choice depends on what you want to do with the information. Very important – there are two clock domains in the system, the FPGA system clock and the NTSC camera clock (*tv_in_line_clock1*). When you are processing data, think carefully about which clock to use.

So what else has to be changed to display the image? In the main file, where we previously store four 8 bits of Y in each memory location, we now store two 18 bits of either YCrCb or RGB in each memory location. So vr_pixel must be changed accordingly. Also the ZBT memory address now changes on every other hcount rather than every fourth hcount. Similarly in *ntsc2zbt*, the same changes must be made.

Another bit of complexity is that NTSC luminance and chrominance YCrCb data must be converted over to RGB. In very simplistic terms, RGB is a just a linear transformation of YCrCb. See http://web.mit.edu/6.111/www/f2007/handouts/labs/lab5.html for an overview. Another level of complexity is the timing. Many of the modules have latency due to pipelining. The video signals should be appropriately pipelined.

The actual mathematics is described in detail in http://www-mtl.mit.edu/Courses/6.111/labkit/appnotes/xapp283.pdf. There is a Xilinx Verilog module ycrcb2rgb.v (http://web.mit.edu/6.111/www/f2011/tools/ycrcb2rgb.v) that implements this conversion. Keep in mind that this conversion has a five clock cycle (need to verify) latency.