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

Fall 2003 Project
Checkpoint 3, Full Motion Video

1. Motivation

This is the third checkpoint for the Fall 2003 project.

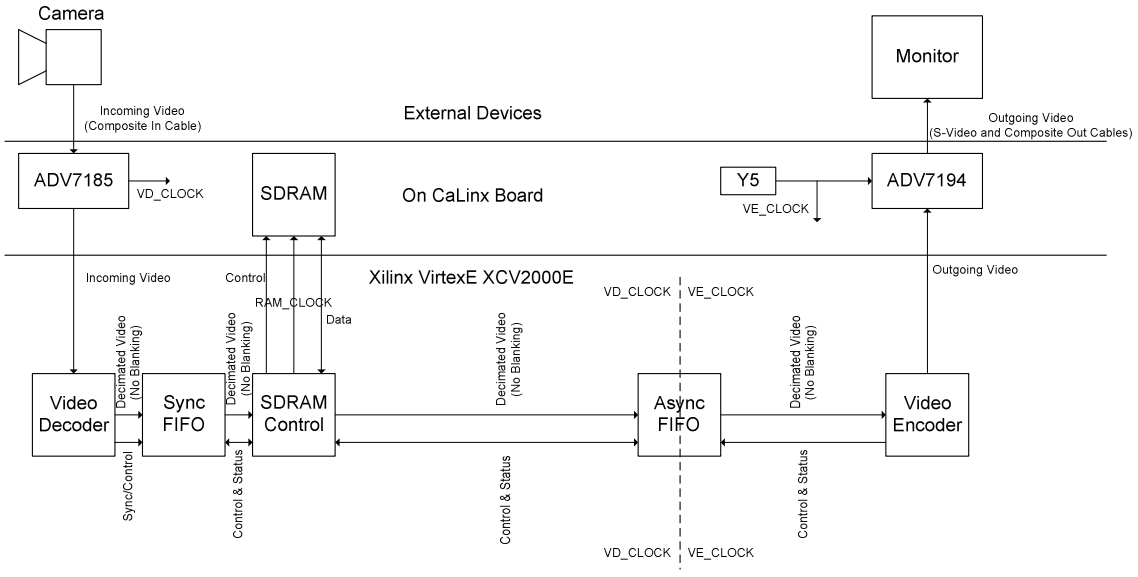
- You will familiarize yourself with digital video encoding and decoding standards
- You will familiarize yourself with NTSC standard
- You will use an asynchronous and a synchronous FIFO
- You will make good use of your SDRAM controller
- You will be able to actually see something impressive

2. Objective

Use a synchronous FIFO, the micron SDRAM chips on the CaLinx board, your SDRAM controller, an asynchronous FIFO and your video encoder to display full motion black and white 320x240 30frame/30field per second video on the TV monitors.
In this checkpoint you will combine the previous two checkpoints with a minimum of logic to get video running.

3. Methodology

3.0 Block Diagram



Given Modules:

- Video Decoder "**vid_dec**", this is the same as in checkpoint 2
- An asynchronous FIFO "**async_fifo**" with 32 bit I/O for use in crossing clock domains
- An synchronous FIFO "**sync_fifo**" with 32 bit I/O for use in buffering data to the SDRAM
- A skeleton **FPGA_TOP**, you must add to this module
- Two files of black boxes "**black.v**" and "**util.v**"

Modules You'll Need:

- Your SDRAM top and control modules.
  - You wont need the PN generators
  - Remember to use the new FIFOs, (they have counters!)
- Your video encoder module

You will need to assemble these modules in some kind of sensible fashion according to the block diagram above. Please read the module descriptions, look at the data sheets for the FIFOs and think carefully before you begin to write verilog.

3.1 Video Encoder

You will need to modify your video encoder SLIGHTLY:

```
output [9:0]  P
output        PAL_NTSC;
output        RST_OUT_;
output        HSYNC_;
output        VSYNC_;
output        BLANK_;
output        SCRESET;
input         CLK;
```

CLK will be VE_CLOCK will be the 27MHz clock from the encoder, which will be provided to you to work with. P is the 10bits output on which you have to send your video and control data in the upper 8. It doesn't matter what you send in the lowest 2 bits. Ignore them.

Name	Description
P	10 bits for 8-bit parallel ITU 601/656 stream
PAL_NTSC	keep it to 0 for NTSC transmissions
RST_OUT_	use it to RESET the encoder (Check datasheet)
HSYNC_	not needed for our mode of operation. Tie to logic 1
VSYNC_	not needed for our mode of operation. Tie to logic 1
BLANK_	not needed for our mode of operation. Tie to logic 1
SCRESET	not needed. Tie to 0

Remember that the output from this module MUST be standard video as describe in lab lecture and in the Video In a Nutshell handout. You will need to make sure your encoder transmits the proper number of lines and bytes per line. This means it will have to duplicate pixels and fields to convert from 320x240 to 640x480, and it will need to add some black pixels at the end of each line and some black lines at the end of each field.

Changes:

- You might need control signals to and from the asynchronous FIFO or SDRAM.
- You will need to convert between the 32-bit 4-pixel value from the FIFO to actual video. Remember each byte of that 32-bit value is a luma for a new pixel at 320x240

3.2 SDRAM

This should be the simplest module to modify. Mostly you will need to change the addressing scheme and eliminate the stop state.

Changes:

- You should have a switch from the board (one of the DIP switches) which will freeze the video by disabling writes to SDRAM so that reads will keep displaying the same frame.
- You will normally need to continuously read and write SDRAM (rather than stopping after one read and write as in checkpoint 1)
- Your addressing scheme should allow you to easy access a given row and column of video.
    - **Think about how this will work with your project!!!**
- Think about the clocking. With an async FIFO (you can choose its clock) do you still need that negedge/RAM_CLK register between the SDRAM and the FIFO?

3.3 FIFOs

These FIFOs are very similar to the other FIFOs you've seen. These are both 32-bit FIFOs.

Why 32-bit? Think about the SDRAM interface…

In fact the 32bit interface is the whole reason for the new FIFOs, otherwise we'd have made life easier and let you use 8bit like last time.

In addition to the signals that FIFOs have had in the past these two have a single bit data count. The synchronous FIFO has a single bit output of data count which will indicate whether the FIFO is half empty or half full. The asynchronous FIFO has two separate counters one for each clock domain. Please read the datasheets (**async_fifo.pdf** and **sync_fifo.pdf**) for more information.

Tips: (If you don't follow these you might end up with an impossible design)

- You **CANNOT** put sync or blanking data into the FIFOs. Only valid, active video data can cross FIFOs. If you try to put blanking data into the FIFOs you wont have enough time and bandwidth to match data rates between the encoder and decoder.
- You may **NOT** have any sync signals (VALID, SOF, EOF, etc) that bypass the FIFOs
    - These would need to be synchronized somehow, and that's much too hard to debug
    - You will need to ensure that your modules all reset properly to keep things running.
    - Each separate module, the encoder, SDRAM and decoder must read/write the correct amount of data from/to the FIFOs
    - Each module will need some arrangement of counters so that it knows which pixels it is pulling from or stuffing into the FIFO
- Your FIFOs should not become full or empty, this is important, your solution wont work if this happens. For example if the async FIFO becomes full in the middle of a read burst from SDRAM, what happens? Or if it becomes empty right when your encoder enters the active portion of a field and needs data what happens?

4 Video Output

Once the I2C data is sent, you can start sending the video date to the encoder. See the class information and the data sheets (page 22) and the VideoNutshell.doc for specific details on the 8-bit parallel bit stream to send.

The circuit has two main states – one when it is sending the I2C data, and one when transmitting video data. While transmitting video, we have to send a new byte every clock cycle, so it would be convenient to just have a couple of counters (HCOUNT and VCOUNT for example) running and let various operations like sending EAV, blanking data, SAV, video data take place depending on the value of the counter. Since counters are expensive, think of a way to share your video counters with the I2C logic (we've pretty much given this to you)

Whenever reset (switch 1) is pressed, restart from the beginning, resending the I2C data and then start

transmitting video data.

Notes:

- It's not necessary to start transmitting your EAV immediately after you finish transmitting the I2C sequence. The video encoder will wait until it gets an EAV (ff, 00, 00, code) before it begins decoding your data. Just make sure you have your sequence exactly the way it should be. Once you begin sending, you have to maintain the sequence of EAV, HB, SAV, Video *exactly* with the correct number of cycles.
- If you get a shaky picture or messy, then you probably have too few or too many cycles somewhere (wrong number of pixels per line, wrong number of lines per field, etc). Check your state machine/counter logic. An easy way to check you are doing the right thing is to write the output of your 656/601 stream to a file and actually count pixels and check the sync codes.
- If you are getting weird colors, you probably are not sending data in the correct order (i.e., Cb first, Y next, Cr, and then Y).
- Wrong values of Y,Cb and Cr could also result in the monitor losing sync! Therefore, if you want to play around with values start with black (0x80 for Chroma and 0x10 for Luma)
- We will always use 0x80 for chroma to keep things in black and white

5 Where to Begin

Begin with FPGA_Top, you'll need to modify this extensively since the version we have given you is blank. You will need to integrate components from checkpoints 1 and 2 into this module. Once you have a vague idea of what signals will go where you should be able to start modifying your submodules.

Remember we're using more advances FIFOs for this checkpoint. They are 32-bit and one is asynchronous. You'll need to ensure they're never full or empty by correctly connecting it them to your SDRAM controller. You will also need to make sure that your memory addressing scheme will work with your block motion estimation so that you don't need to modify things later.

You encoder should be mostly unchanged. But the decoder will need to enter more data into the FIFO, thankfully this should be significantly easier than in checkpoint2.

6 Acknowledgements

Original Lab by Greg Gibeling
Edited Prof Ron Fearing
Based on labs by Prof.John Wawryznek

Checkpoint 3 Check-offs

Name: _____

Name:_____

Lab Section: _____

1. Testbench and Simulation _____ (25%)

2. FIFOs never empty or full _____ (12.5%)

3. Memory Addressing works for Project _____ (12.5%)

4. Demo

    Static Display (Capture one frame) _____ (25%)

    Dynamic Display (Moving Video) _____ (25%)

Total _____ (%)