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

Checkpoint 2
Local Video System

1.0 Motivation

This checkpoint serves three purposes:

1. To acquaint you with the arbitration of SDRAM read/write requests
2. To give you experience piecing together sub-modules to accomplish a larger goal.
3. To get local video working, which is a major component of your final project.

This will be the first project checkpoint that requires a significant about of design to be done by you – this checkpoint is, for the most part, very open-ended. Just make it work!

-Good Luck!!

"BECAUSE YOU WILL BE KEEPING AND RELYING ON THIS CODE FOR MONTHS, IT WILL ACTUALLY SAVE YOU MANY STRESSFUL HOURS TO ENSURE IT WORKS WELL NOW, RATHER THAN WHEN YOU ARE ABOUT TO FINISH THE PROJECT"

2.0 Introduction

In this module you will be building the "glue" that will hold together the entire video system. You have already built the SDRAM controller, now you just need to create a reliable interface such that several data-hungry components can talk to it without conflicts.

Your goals for this checkpoint will be:

1. To design and implement an arbitration scheme for the SDRAM control.
    a. You will need to be able to handle simultaneous read requests and write requests.
2. To implement local video
    a. To capture data from the provided video decoder and feed it reliably into the video encoder to display local video
    b. You will need to modify your address counter to correctly access the data needed by the video encoder and to write the data correctly for your video decoder.
3. To establish a reliable interface between components with different data rates

a. Using FIFOs as buffers, create an interface to transfer data between the SDRAM arbiter and various connected components.
b. You must ensure that the FIFO does not overflow or underflow

2.1 Video Format

The video format for the video encoder is the ITU 656 and the format of the video decoder is the ITU 601. As far as this checkpoint is concerned, the only difference between the two is the number of active video lines (more on this later). Both formats are based on the luminance and chrominance used in television instead of the standard RGB format found in computer monitors.

The video data is in 32-bit words, each word representing a pair of pixels side-by-side on the screen. There are two 8-bit luminance (or "brightness") values per pixel pair: one for the left pixel and one for the right pixel. There is also an 8-bit blue chrominance value and an 8-bit red chrominance value containing the color information of the pixel pair, making a total of 32 bits per pixel pair. This results in the use of **the same chrominance for each pixel pair but different luminance values for each individual pixel**. The reason for this is that your eyes are more sensitive to brightness (luma) rather than color (chroma).

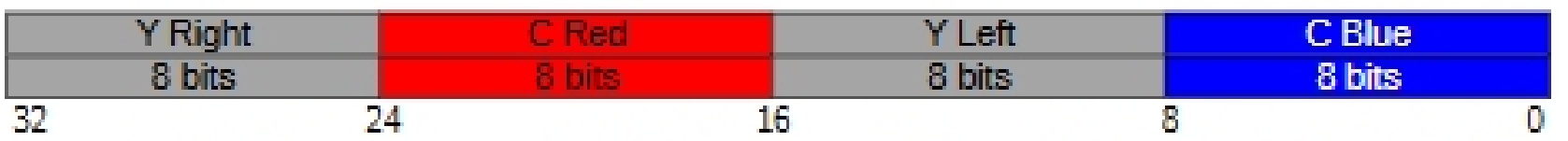

Figure 1: Pixel Pair Representation (Y is luminance, left and right is the pixel in the pair)

These pixel pairs are arranged in a row/column format like a square matrix that make up the screen, with each row corresponding to a row of active video.
The provided video encoder takes in this exact format via its 32-bit *DIn* input. If you wish to read more about the video format, read the specification about the video encoder from Spring 2007.

2.2 Local Video in SDRAM

Of course the point of this checkpoint is to store video in SDRAM. This section details the format of the video data in memory.

Since the video data is organized into 32bit words, this motivated the design of a 32-bit wide SDRAM interface in Checkpoint 1. The organization of pixels into words and words into bursts is shown in figure 2 below.

Figure 2: Video Memory Organization

This simply shows the organization of the pixels within a burst. In addition, we must decide on a format for how to organize the bursts. The scheme we will use is the following:

```
RowAddress = {pad with 0's, PixelRow}
ColumnAddress = {BurstColumn, 3'h0}
BankAddress = 2'b00
```

This addressing scheme should be pretty straightforward; **the row address is the row of active video you are accessing and the column address is what pixel pair to access within that row**. If this sounds confusing, just remember that all you are doing is copying the contents of a square matrix ([pixel row][pixelpair column]) into a bigger square matrix ([row address][column address]).

Since the ITU 601 standard has only 487 lines of active video and the ITU 656 standard has only 487 has 507 lines of active video, you must **pad the top of the row address with 4*1'b0s**, to make it total a 13bit for a row address.

The **bank address is fixed at 2'b00** for local video.

The **column address is based on the video pixel pair column**. Notice that in this case the `BurstColumn` indicates which BURST you wish to access. The `3'h0` on the right ensures that you will always **read from a column address that is a multiple of 8**, since you read out 8 pixel pairs per read/write request.