CS-476: Embedded System Design

Practical work 6

Streaming interface

Version:
1.0

Contents

1 Introduction

In practical work 2 we have seen how we can accelerate the grayscale conversion by using a custom instruction. However, this acceleration is not very effective, as we have to transfer the RGB-565 pixels to the μC, and we have to write back the grayscale value. As the data-port of the μC does for each read and each write a single-bus transfer, the bus-communication overhead is quite big. In this practical work we are going to look how we can reduce this bus-overhead by:

1. Using the CI-attached memory with DMA, developed in PW4. As the DMA-controller has the ability to transfer the pixels in burst-transfers, and we can overlap transfer with calculation, this should drastically reduce the bus-communication overhead.
2. As we do not really need the color-picture produced by the camera, and the camera does not support grayscale mode, we can use a so-called streaming accelerator. This accelerator will be build into the camera interface, reducing completely the bus-communication overhead, and of course also the calculation part.

2 Exercises

2.1 Prerequisites

For this PW you are allowed to use your own results of PW2 and PW4, or to use the sample solutions that you can find on moodle. Furthermore, for the second part of this PW I put the data-sheets of the camera-chip (OV7670) on moodle. Please study *Figure 6*, describing the VGA-Frame-Timing, and *Figure 11* describing the RGB-565-Output timing. Furthermore, we are going to modify the camera interface found in:
`virtual_prototype/modules/camera/verilog/camera.v`
Study this interface and make sure that you understand how it works.

2.2 Task 1: Overlap transfer with calculation (6 Points)

The basis for this task is the grayscale custom instruction as result of PW2. We will take the version that converts 4 pixels at a time to grayscale (see the sample solution on moodle). Furthermore, we are going to use the result of PW4, the custom instruction memory with DMA.
As preparation for this task we are going to profile again the system made in PW2 and we are going to note the bus-idle cycles and the run-time to convert one RGB565 image into an RGB-image.
Next we are going to modify the program, such that we use the DMA to transfer the RGB565-pixels into the CI-memory, convert those pixels to Grayscale with the grayscale custom instruction, and transfer the grayscale pixels back with DMA to the grayscale screen-buffer.
To be able to use the overlap of transfer and calculation we have to use the method of *ping-pong (double) buffering*. For this we divide our 2 kByte memory in two parts of 1 kByte where one is used to transfer the new set of pixels, and the other for calculating the grayscale values. This means that we can handle $\frac{1024}{2} = 512$ RGB565-pixels at a time. As our screen consists of $640 * 480 = 307200$ pixels, we need $\lceil\frac{307200}{512}\rceil = 600$ iterations.
The sequence in which we do the transformation is:

1. For the first 512 RGB565-pixels we perform a DMA-transfer to the first buffer and we wait until the DMA-transfer has finished.
2. For 599 iterations we initialize a DMA-transfer to the other buffer, after that we calculate the grayscale values in the buffer that contains already the RGB565-pixels, and we "overwrite" them with the grayscale pixels. After this calculation is done we check that the DMA-transfer to the other buffer has finished. We now transfer the calculated grayscale pixels to the grayscale screen buffer with DMA, and wait until this DMA-transfer has finished.
3. For the last iteration we do the same as for the previous iteration, with the exception that we do not initialize a DMA-transfer of 512 RGB565-pixels, as we already have them in our buffer.

Hint: The size in the CI-memory of grayscale pixels is half the size of the RGB565-pixels, as the grayscale pixels only take 1 byte, whilst an RGB-565 occupies 2 bytes.
Implement this algorithm and profile it. What can you observe?

2.3 Task 2: Simple streaming interface (4.5 Points)

Before starting this task, download the file `streaming.zip` from moodle and extract it in your virtual_prototype-directory. Compile and download this program to your virtual prototype. When all went well you should see the color picture on your screen with 15 frames-per-second.

We are now going to modify the file:
`virtual_prototype/modules/camera/verilog/camera.v`
in such a way that we convert the RGB565-color pixels into RGB565-grayscale pixels. To do so, you need to:

- The signal `s_pixelWord` contains two RGB565 color pixels. These are directly written into the `lineBuffer`. This we are going to change by:
- Define a signal `s_grayscalePixelWord` that will contain two RGB565-grayscale pixels, this will now be written into the `lineBuffer`, hence replace the line `.dataIn1(s_pixelWord)` by `.dataIn1(s_grayscalePixelWord)`.
- Perform the RGB565 to grayscale conversion on the two pixels contained in the signal `s_pixelWord` as done in PW2. You will now have two 8-bit based grayscale pixels.
- Perform the RGB565 grayscale pixel reconstruction in the signal `s_grayscalePixelWord` by using the luminance of the grayscale pixel and assigning the same luminance to the red, green, and blue-channel, e.g. `red[4..0]=grayscale[7..3]`, `red[5..0]=grayscale[7..2]`, and `blue[4..0]=grayscale[7..3]`.

Build the new virtual prototype and upload the program used before to the virtual prototype. Note that you should not change the program. You will now see a grayscale image at 15 frames per second. *Info:* The camera interface reads one line of pixels (640) in the `lineBuffer` during the time the `hsync` (noted in the datasheets as `HREF`) is active. In the time the `hsync` signal is inactive, the camera interface writes these pixels to the screen-buffer with a burst-size of 16 words per burst.

2.4 Task 3: Optimizing the streaming interface (7 Points)

Of course, the Sobel algorithm does not work on the RGB565-grayscale pixels, but on 8-bit based grayscale pixels. By transforming the RGB565-color pixels directly in 8-bit based grayscale pixels also has the advantage that we reduce the bus-occupation by a factor of 2, as we only have to transfer 1 byte per pixel instead of 2 bytes per pixel. In this exercise we are going to implement this optimized streaming interface. Make all changes required to implement this functionality. *Note:* As we now use a grayscale frame buffer, you have to un comment the line `#define __RGB565__` in the file `camera.v`.

2.5 Handing in:

To be able to correctly verify your solutions, please hand in a separate zip-file with your complete virtual prototype for each task. Also submit one `readme.md`-file where you answer the questions and describe the content of the different zip-files.