CSCE 313: Embedded Systems

Video Out and Image Transformation

Instructor: Jason D. Bakos
Annoucements

- Demo your Lab 1 to instructor/TA on Monday
- Save your Lab 1 in a separate directory
  - Ex. `cp -a ~/lights ~/lights_lab1`
- Create new Eclipse project
Video on DE2 Board

- VGA is a simple video standard from the late 1980’s
  - Uses a 15-pin connector, but only 5 pins are needed:
    - 3 analog pins: red, green, blue using amplitude modulation
    - 2 digital pins: horizontal sync, vertical sync

- DE2-115 has an off-chip video chip
  - Mostly just an digital-to-analog converter connected to VGA output
  - VGA contains analog reg, green, blue intensity and digital synchronization signals
Video on DE2 Board

- VGA controller (in SOPC Builder) sends timed values to the DAC
  - Natively set to 640x480 resolution but this can be changed in the Verilog

- Images are transmitted to the DAC as “row-major” (line-by-line) array of pixels
  - Each pixel has three components: red, green, blue
  - All 0’s is black, all 1’s is white
  - Each pixel is 10 x 3 bits for DE2, 8 x 3 bits for DE2-115

8 bits 8 bits 8 bits

RED INTENSITY  GREEN INTENSITY  BLUE INTENSITY
Video on DE2 Board

- Need to establish a “frame buffer” in memory to hold a picture to display that the CPU can manipulate
  - Use the on-board SRAM as our frame buffer (“pixel memory”)
  - A 640x480x24 image requires ~1 MB to store
  - Initially, scale down the image to 320 x 240 x 24 bits = 225 KB

- The Altera University Program contains cores to perform color-space and resolution re-sampling (scaling/conversion) in hardware

- SOPC Builder:
  - First task: edit your clocks component to add vga_clock to your design
Video on DE2 Board

- **Frame layout:**
  - Consecutive addressing, each row stored consecutively
  - X-Y addressing, pad each row to make it a power of 2

- Pixel \( x,y \) has offset \((y \times \text{row siz} + x)\) in pixels from base address
- Pixel \( x,y \) has offset \((y << 10 \mid x)\) in pixels from base address
- Wastes 184 KB
Image Representation

(cols pixels, col x 3 bytes)

Byte offset = 0
(src_image)

Byte offset = cols x 3

Byte offset = (rows - 1) x cols x 3

Byte offset = cols x 3 - 1

Byte offset = 3

Byte offset = cols x 3 - 3
System Design for Video (DE2/DE2-115)

CPU

SRAM interface

Specify the base address of the SRAM as front and back buffer addresses

data master

data

SRAM chip

320x240x24 image

on-chip

off-chip

(From Univ. Program)

Pixel Buffer for DMA

RGB Resampler

24 -> 30

stream 24 bit color
320x240 sys_clk

10 bits/channel
3 channels

Video character buffer with DMA

Enable transparency

data/control

video stream

30 bit color
640x480 sys_clk

Simple mode

VGA controller

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

VGA controller

Dual-clock FIFO

sys_clk

stream 30 bit color
640x480 vga_clk

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.

DE-115 D-sub

off-chip

on-chip

VGA DAC

DE-115 D-sub

off-chip

on-chip

Setting:
DE2/DE2-115

specify the base address of the SRAM as front and back buffer addresses.
Verilog Modifications

- Add to your top-level module definition:

```
//------------------------------- SRAM Interface ----------------------------- //
inout [15:0] SRAM_DQ, // SRAM Data bus 16 Bits
output [19:0] SRAM_ADDR, // SRAM Address bus 20 Bits
output SRAM_ADDR // SRAM High-byte Data Mask
output SRAM_ADDR // SRAM Low-byte Data Mask
output SRAM_ADDR // SRAM Write Enable
output SRAM_ADDR // SRAM Chip Enable
output SRAM_ADDR // SRAM Output Enable

//------------------------------- VGA --------------------------------------- //
output VGA_CLK, // VGA Clock
output VGA_HS, // VGA H_SYNC
output VGA_VS, // VGA V_SYNC
output VGA_BLANK_N, // VGA BLANK
output VGA_SYNC_N, // VGA SYNC
output [7:0] VGA_R, // VGA Red
output [7:0] VGA_G, // VGA Green
output [7:0] VGA_B, // VGA Blue
```
Verilog Modifications

- Add to module instantiation for nios_system:

```
.SRAM_ADDR_from_the_sram_0 (SRAM_ADDR),
.SRAM_CE_N_from_the_sram_0 (SRAM_CE_N),
.SRAM_DQ_to_and_from_the_sram_0 (SRAM_DQ),
.SRAM_LB_N_from_the_sram_0 (SRAM_LB_N),
.SRAM_OE_N_from_the_sram_0 (SRAM_OE_N),
.SRAM_UB_N_from_the_sram_0 (SRAM_UB_N),
.SRAM_WE_N_from_the_sram_0 (SRAM_WE_N),

.VGA_BLANK_from_the_video_vga_controller_0 (VGA_BLANK_N),
.VGA_B_from_the_video_vga_controller_0 (VGA_B),
.VGA_CLK_from_the_video_vga_controller_0 (VGA_CLK),
.VGA_G_from_the_video_vga_controller_0 (VGA_G),
.VGA_HS_from_the_video_vga_controller_0 (VGA_HS),
.VGA_R_from_the_video_vga_controller_0 (VGA_R),
.VGA_SYNC_from_the_video_vga_controller_0 (VGA_SYNC_N),
.VGA_VS_from_the_video_vga_controller_0 (VGA_VS),
```
Storing and Accessing an Image on the DE2

- Altera has designed a read-only flash-based file system that we can use to store data files.

- Instead of a traditional file system (i.e. NTFS, FAT32, ext3, reiserfs), Altera uses the internal structure of an uncompressed ZIP file to store one or more files.

- To use it, you need to add an interface for the 8 MB CFI Flash memory to your system design, along with an Avalon tri-state bridge so the flash can be initialized externally.

```
CPU
    ▼
     data master
        ▼
          slave
              ▼
                registered

Avalon-MM
   tristate
      bridge

    ▼
        master

CFI Flash Interface

addr=23 bits
data=8 bits
setup 0 ns,
wait 100 ns,
hold 0 ns
```
Verilog Modifications

- Add to your top-level module declaration:

```verilog
• Add somewhere in the top-level module:
  assign FL_RST_N = 1'b1;
  assign FL_WP_N = 1'b1;

• Add to module instantiation for nios_system:
  .address_to_the_cfi_flash_0 (FL_ADDR),
  .data_to_and_from_the_cfi_flash_0 (FL_DQ),
  .read_n_to_the_cfi_flash_0 (FL_OE_N),
  .select_n_to_the_cfi_flash_0 (FL_CE_N),
  .write_n_to_the_cfi_flash_0 (FL_WE_N),
```
BSP Modifications

- In the BSP Editor, make the following changes:
  - This must match base address in SOPC builder.
  - Identifier.
  - Make this 0 to use all of Flash memory.
To load an image into the DE2, I have written a MATLAB script that can:
- read image file with size 320x240 or smaller
- add a black border around the image if smaller than 320x240
- write the image in 24-bit color into a RAW image file
- displays the original and bordered images

To use it:
- download it from the course webpage
- open MATLAB (command: “octave”)
- change current folder to where you downloaded it
- type: `convert_image_to_data_file('<filename>');`
  - You may use my image, lumcat.jpg or use your own
- this will generate myfile.dat and myfile.zip
Programming the Flash Memory

- To program Flash memory, prior to running your program, in Eclipse, go to Nios II | Flash Programmer

- Then, do File | New
  - Get settings from BSP settings file
  - Browse for your BSP settings file
    - Under `<project name>/software/<eclipse project>_bsp`
  - Add myfile.zip and click Start

Make sure this matches BSP
Pointers

• In Java, all object “handles” are pointers (references)
• In C/C++, object handles can be either actual or pointers:
  – int a; (integer)
  – int *b; (pointer to an integer)
  – b = &a (address of a)
  – *b = 2; (assign contents of b)

• Arrays are pointers:
  – int a[100];
  – a[0] = 2; \Leftrightarrow \phantom{1} * (a) = 2;
  – a[5] = 5; \Leftrightarrow \phantom{1} * (a+5) = 5;

• 2-dimensional arrays can be “superimposed” over one dimensional:
  – a[i * (2^{nd} dimension size) + j]

• 3-dimensional arrays can be “superimposed” over one dimensional:
  – a[i * (2^{nd} dimension size) * (3^{nd} dimension size) + j * (3^{nd} dimension size) + k]
Typecasting

• In lab 2, you will need to make use of floats and convert to integers

• Examples:
  
  ```c
  float a;
  alt_u16 b;
  
  a = sin(2.5);
  b = (alt_u16)roundf(a);
  ```
Allocating Heap (Dynamic) Memory in C

- Use the `malloc()` system call
- `malloc()` returns a void pointer, so you must cast the return value to match the type for which you’re allocating memory
- The only parameter is the number of bytes to allocate
- For arrays (almost always the case), the number should be a multiple of the size of each element

Example:
```c
alt_u8 *my_image;
...
my_image=(alt_u8 *)malloc(320*240*3);
...
free (my_image);
```
Accessing the RO File System from SW

• Declare C-style file pointer:
  FILE *myfile;

• Open the file:
  myfile=fopen("my_fs/myfile.dat","rb");
  if (myfile==NULL) perror ("error opening datafile");

• Note: path above must match one in BSP!

• Allocate memory and read the image data:
  my_image=(alt_u8 *)malloc(320*240*3);
  fread(my_image,sizeof(alt_u8),320*240*3,myfile);
Accessing the Source Image

- We’re using consecutive mode for the pixel memory, so pixels are stored consecutively

\[
\begin{array}{c}
\text{row 0, 320 pixels} \\
\text{row 1, 320 pixels} \\
\text{row 2, 320 pixels} \\
\ldots \\
\text{row 239, 320 pixels}
\end{array}
\]

low addresses

high addresses

- Each pixel is 3-byte value

\[
\begin{array}{c|c|c}
\text{RED} & \text{GREEN} & \text{BLUE} \\
23 & 16 & 15 \\
8 & 7 & 0
\end{array}
\]

- To access pixel at row=100, col=200:
  - my_image[100*320*3+200*3+0] (red)
  - my_image[100*320*3+200*3+1] (green)
  - my_image[100*320*3+200*3+2] (blue)
New Header Files

• Add:

```
#include <altera_up_avalon_video_character_buffer_with_dma.h>  // to write characters to video
#include <altera_up_avalon_video_pixel_buffer_dma.h>  // to swap front and back buffer
#include <math.h>  // for trigonometry functions
#include <stdlib.h>  // for file I/O
```
The Pixel Buffer

- To use:
  - Declare global variable:
    ```
    alt_up_pixel_buffer_dma_dev *my_pixel_buffer;
    ```
  - Assign it:
    ```
    my_pixel_buffer =
    alt_up_pixel_buffer_dma_open_dev("/dev/video_pixel_buffer_dma_0");
    ```
  - To clear screen:
    ```
    alt_up_pixel_buffer_dma_clear_screen(my_pixel_buffer, 0);
    ```
  - To draw pixel:
    ```
    alt_up_pixel_buffer_dma_draw(my_pixel_buffer,
    (my_image[(i*320*3+j*3+2)]) +
    (my_image[(i*320*3+j*3+1)]<<8) +
    (my_image[(i*320*3+j*3+0)]<<16), j, i);
    ```
Using the Character Buffer

• Use:

```c
alt_up_char_buffer_dev *my_char_buffer;
...
my_char_buffer=alt_up_char_buffer_open_dev("/dev/video_character_buffer_with_dma_0");
if (!my_char_buffer) printf ("error opening character buffer\n");
alt_up_char_buffer_clear(my_char_buffer);
alt_up_char_buffer_string(my_char_buffer,"Video works!",0,59);
```

• Allows you to superimpose text on the screen at (col,row)
  - 80 cols x 60 rows
Image Transformation Matrices

- Simple image transformation matrix can be used to...
  - rotate, scale, shear, reflect, and orthogonal projection

- For Lab 2, we want to perform rotation and scaling

- The matrices we use are 2x2 and used to determine how to
  move each pixel from the original image to the new image
  in order to perform the transformation

- Consider:
  - source pixels (row,col) of original image
  - destination pixels (row’,col’) of transformed image
Image Transformation Matrices

• Clockwise rotation:

\[
\begin{bmatrix}
row' \\
col'
\end{bmatrix} = \begin{bmatrix}
\cos \theta & \sin \theta \\
-\sin \theta & \cos \theta
\end{bmatrix} \begin{bmatrix}
row \\
col
\end{bmatrix}
\]

\[
row' = row \cdot \cos \theta + col \cdot \sin \theta \\
col' = -row \cdot \sin \theta + col \cdot \cos \theta
\]

• Counterclockwise rotation:

\[
\begin{bmatrix}
row' \\
col'
\end{bmatrix} = \begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{bmatrix} \begin{bmatrix}
row \\
col
\end{bmatrix}
\]

\[
row' = row \cdot \cos \theta - col \cdot \sin \theta \\
col' = row \cdot \sin \theta + col \cdot \cos \theta
\]

• Scaling (factor \( s \)):

\[
\begin{bmatrix}
row' \\
col'
\end{bmatrix} = \begin{bmatrix}
s_x & 0 \\
0 & s_y
\end{bmatrix} \begin{bmatrix}
row \\
col
\end{bmatrix}
\]

\[
row' = row \cdot s_x \\
col' = col \cdot s_y
\]
Issues to Resolve

- Using these algorithms directly, the rotation and scaling occur about the origin (0,0)
Issues to Resolve

- We want it to occur about the center of the image
Issues to Resolve

• To fix this:
  – subtract 320/2 from the column
  – subtract 240/2 from the row

...before you multiply against the transformation matrix, then add these values back after your multiply
Issues to Resolve

• Second problem: pixels aliasing to same location, causing unfilled pixels in destination image
Issues to Resolve

- To solve this, iterate over all destination image pixels and calculate reverse transform
  - Counterclockwise rotation
  - Scale factor 1/s
Issues to Resolve

- Assume destination pixel (10,20) maps to source pixel (87.4, 98.6)
- Must interpolate the value of this “virtual” source pixel

\[
\text{weight}(i_{\text{int}}, j_{\text{int}}) = (1 - i_{\text{frac}}) \cdot (1 - j_{\text{frac}})
\]

\[
\text{weight}(i_{\text{int}}, j_{\text{int}} + 1) = (1 - i_{\text{frac}}) \cdot j_{\text{frac}}
\]

\[
\text{weight}(i_{\text{int}} + 1, j_{\text{int}}) = i_{\text{frac}} \cdot (1 - j_{\text{frac}})
\]

\[
\text{weight}(i_{\text{int}} + 1, j_{\text{int}} + 1) = i_{\text{frac}} \cdot j_{\text{frac}}
\]
Bilinear Interpolation

- Example: Set destination pixels (10,20) as a mixture of pixels:
  - (87,98), (88,98), (87,99), (88,99)
  - Must separate color channels in code
Issues to Resolve

• Make sure you...
  – use rounding and type casting for the transformation matrix (float and alt_u16)
  – disregard output coordinates that fall outside the frame
  – always transform against the original image
  – initialize the output image to black before transforming