CSCE 313: Embedded Systems

Video Out and Image Transformation

Instructor: Jason D. Bakos
Video on DE2 Board

- DE2 has a VGA DAC (digital-analog converter) and VGA output

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

- Images are transmitted to VGA controller as row-major array of RGB pixels
  - 10 bits of RED, GREEN, BLUE intensities
  - All 0’s is black, all 1’s is white

<table>
<thead>
<tr>
<th>bits 29:20</th>
<th>bits 19:10</th>
<th>bits 9:0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RED INTENSITY</td>
<td>GREEN INTENSITY</td>
<td>BLUE INTENSITY</td>
</tr>
</tbody>
</table>
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
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 640x480x30 image would require ~1.1 MB to store
  - The DE2 SRAM is only 512 KB
  - 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
System Design for Video

CPU

SRAM interface

Pixel Buffer for DMA

RGB Resampler 24 -> 30

Video character buffer with DMA

Video alpha blender

dual-clock FIFO sys->vga

VGA controller

VGA DAC

DE-15F D-sub

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

SRAM chip

320x240x24 image

(From Univ. Program)

on-chip

off-chip

(From Univ. Program)

data master

control

data

consecutive mode

data

data/control

stream 24 bit color 320x240 sys_clk

stream 30 bit color 320x240 sys_clk

stream 30 bit color 640x480 sys_clk

stream 30 bit color 640x480 vga_clk

simple mode

Enable transparency

3 channels

10 bits/channel

sys_clk

sys_clk

sys_clk

sys_clk

sys_clk

sys_clk

sys_clk

sys_clk

sys_clk

320x240x24 image

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

- Add to your top-level module definition:

```verilog

SRAM Interface

inout [15:0] SRAM_DQ, // SRAM Data bus 16 Bits
output [17:0] SRAM_ADDR, // SRAM Address bus 18 Bits
output SRAM_ADDR_LSB, // SRAM High-byte Data Mask
output SRAM_ADDR_MSB, // SRAM Low-byte Data Mask
output SRAM_WE_N, // SRAM Write Enable
output SRAM_CE_N, // SRAM Chip Enable
output SRAM_OE_N, // SRAM Output Enable

VGA

output VGA_CLK, // VGA Clock
output VGA_HS, // VGA H_SYNC
output VGA_VS, // VGA V_SYNC
output VGA_BLANK, // VGA BLANK
output VGA_SYNC, // VGA SYNC
output [9:0] VGA_B, // VGA Blue[9:0]
```
Verilog Modifications

- Add to module instantiation for nios_system:

```verilog
.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),
.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),
.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 4 MB CFI Flash memory to your system design, along with an Avalon tri-state bridge so the flash can be initialized externally.

![Diagram showing CPU connected to Avalon-MM tristate bridge, which in turn connects to CFI Flash Interface. Details: addr=22 bits, data=8 bits, setup 0 ns, wait 100 ns, hold 0 ns.]
Verilog Modifications

- Add to your top-level module declaration:

```verilog
Thursday 6-20-2013 12:44 PM
CU-12 Vandenbosch Lab

Verilog Modifications

- Add to your top-level module declaration:

```verilog
```Verilog Modifications

```verilog
inout [7:0] FL_DQ, // FLASH Data bus 8 Bits
output [21:0] FL_ADDR, // FLASH Address bus 22 Bits
output FL_WE_N, // FLASH Write Enable
output FL_RST_N, // FLASH Reset
output FL_OE_N, // FLASH Output Enable
output FL_CE_N, // FLASH Chip Enable
```

- Add somewhere in the top-level module:

```verilog
assign FL_RST_N = 1'b1;
```

- Add to module instantiation for nios_system:

```verilog
.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: “matlab”)
- 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;` ⇔ `*(a) = 2;`
  - `a[5] = 5;` ⇔ `*(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^{rd} dimension size) + j * (3^{rd} dimension size) + k]`
Typecasting

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

• Examples:
  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:

    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:
  ```c
  FILE *myfile;
  ```

- Open the file:
  ```c
  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:
  ```c
  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

- Each pixel is 3-byte value

- 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:

```c
#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:
    ```c
    alt_up_pixel_buffer_dma_dev *my_pixel_buffer;
    ```

  - Assign it:
    ```c
    my_pixel_buffer= 
    alt_up_pixel_buffer_dma_open_dev("/dev/video_pixel_buffer_dma_0");
    ```

  - To clear screen:
    ```c
    alt_up_pixel_buffer_dma_clear_screen(my_pixel_buffer,0);
    ```

  - To draw pixel:
    ```c
    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

\[
\begin{align*}
\text{weight}(i_{\text{int}}, j_{\text{int}}) &= (1 - i_{\text{frac}}) \cdot (1 - j_{\text{frac}}) \\
\text{weight}(i_{\text{int}}, j_{\text{int}} + 1) &= (1 - i_{\text{frac}}) \cdot j_{\text{frac}} \\
\text{weight}(i_{\text{int}} + 1, j_{\text{int}}) &= i_{\text{frac}} \cdot (1 - j_{\text{frac}}) \\
\text{weight}(i_{\text{int}} + 1, j_{\text{int}} + 1) &= i_{\text{frac}} \cdot j_{\text{frac}}
\end{align*}
\]
Bilinear Interpolation

- Example: Set destination pixels (10,20) as a mixture of pixels:
  - \((87,98), (88,98), (87,99), (88,99)\)
  
  - \(\text{dest}[10,20] = (1-.4)(1-.6)\text{src}[87,98] + (.4)(1-.6)\text{src}[88,98] + (1-.4)(.6)\text{src}[87,99] + (.4)(.6)\text{src}[88,98]\)
  
  - 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
  – always transform against the original image