CSCE 313: Embedded Systems

Final Project

Instructor: Jason D. Bakos
Final Project

- Objective:
  - Begin with Lab 5
  - Convert all floating-point computations to fixed-point
  - Increase resolution to 1024 x 768

- Use fixed-point representation for:
  - z
  - c
  - min_x, max_x, min_y, max_y
  - $x^2 + y^2$ (for checking for divergence)
Final Project

• Both partners must demo their group’s final project
  
  – Each group must still submit code, designs, and report on Dropbox
  
  – Perform demo by May 3
    • Until and including April 28, schedule with me
Fixed-Point Review

• Recall: fixed-point has fixed range
  – Recall: Range of non-fixed-point n-bit integer:
    • \(-2^{n-1} \leq val \leq 2^{n-1}-1\)
  
  – Range of signed (n,m) value:
    • \(-2^{n-m-1} \leq val \leq 2^{n-m-1} - 2^{-m}\)

  – Need to decide where to set decimal point
    • For \(c, [\text{min}|\text{max}]_c[x|y]\):
      – Need to represent values from -2 to 2
      – \(z\) should cover the worst case for a diverged pixel
    • \(x^2+y^2\) should cover the worst case for the \(r^2\) of a diverged pixel
Worst Case Analysis

\[
P(z) = z^2 + y^2
\]

<table>
<thead>
<tr>
<th>( z )</th>
<th>( z^2 )</th>
<th>( c )</th>
<th>( P(z) )</th>
<th>( x^2+y^2 )</th>
<th>LH bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,2)</td>
<td>(4,0)</td>
<td>(2,2)</td>
<td>(6,2)</td>
<td>40</td>
<td>6</td>
</tr>
<tr>
<td>(0,2)</td>
<td>(4,0)</td>
<td>(2,2)</td>
<td>(6,2)</td>
<td>40</td>
<td>6</td>
</tr>
<tr>
<td>(2,0)</td>
<td>(4,0)</td>
<td>(2,2)</td>
<td>(6,2)</td>
<td>40</td>
<td>6</td>
</tr>
<tr>
<td>(2,0)</td>
<td>(4,0)</td>
<td>(2,2)</td>
<td>(6,2)</td>
<td>40</td>
<td>6</td>
</tr>
<tr>
<td>(1.4,1.4)</td>
<td>(0.3.9)</td>
<td>(2,2)</td>
<td>(2,5.9)</td>
<td>38.8</td>
<td>6</td>
</tr>
</tbody>
</table>
Range and Precision

• Problem:
  – Multiply \((32,n)\) \(\text{val}\) with \((32,m)\) \(\text{val} \Rightarrow (64,n+m)\) \(\text{val}\)

• Need a way to get 64-bit product, or we lose the upper 32 bits
Multiply

- Assume 4-bit registers, need 8 bit product:

\[
\begin{align*}
\text{A1} & \quad 10 \quad 11 \quad \text{A0} \\
\text{B1} & \quad 11 \quad 10 \quad \text{B0} \\
\hline
\text{11} & \quad 14 \\
\text{154} & \\
\end{align*}
\]

\[
\begin{align*}
0000 & \quad \text{A0*B0 (4b)} \\
00 & \quad \text{A0*B1 (4b)} \\
00 & \quad \text{A1*B0 (4b)} \\
\text{A1*B1 (4b)} & \quad 0000 \\
\hline
3*2=6 & \\
3*3=9 \rightarrow 9*4=36 \\
2*2=4 \rightarrow 4*4=16 \\
2*3=6 \rightarrow 6*16=96 \\
\hline
154 & \\
\end{align*}
\]
Example Fixed Point Multiply

- Multiply $A = (32,n)$ val and $B = (32,n)$ val, need $C = (32,n)$ product:
  - Declare $A$ and $B$ as “long”
  - Declare $C$ as “long long”
  - Cast $A$ and $B$ as “long long”,

- $C = (\text{long long})A \times (\text{long long})B$;

- Convert $C$ from $(64,2n)$ to $(32,n)$:
  - Shift $C$ $n$ bits to the right

- Return $C$ as int
High Resolution

• Goal: Increase resolution from 320x240 to 1024x768

• Problems:
  – Native resolution of VGA Controller is 640x480, so hardware modification is needed
  – SRAM is only 512 KB, not large enough to store a higher resolution frame, so we need to move pixel buffer to SDRAM
Hardware Modification

- Remove SRAM interface, connect DMA controller master interface directly to SDRAM
- Remove rescaler (no longer needed)
- Change DMA controller settings to 1024x768
- Each time you generate in SOPC Builder, must make edits to two generated Verilog files
Hardware Modification

- **VGA_Controller_0.v, line 78 (after each generation):**

  ```
  parameter CW = 9;
  parameter DW = 29;
  parameter R_UI = 29;
  parameter R_LI = 20;
  parameter G_UI = 19;
  parameter G_LI = 10;
  parameter B_UI = 9;
  parameter B_LI = 0;

  /* Number of pixels */
  parameter H_ACTIVE = 640;
  parameter H_FRONT_PORCH = 16;
  parameter H_SYNC = 96;
  parameter H_BACK_PORCH = 48;
  parameter H_TOTAL = 800;

  /* Number of lines */
  parameter V_ACTIVE = 480;
  parameter V_FRONT_PORCH = 10;
  parameter V_SYNC = 2;
  parameter V_BACK_PORCH = 33;
  parameter V_TOTAL = 525;

  parameter NUMBER_OF_BITS_FOR_LINES = 10;
  parameter LINE_COUNTER_INCREMENT = 10'h001;
  parameter NUMBER_OF_BITS_FOR_PIXELS = 10;
  parameter PIXEL_COUNTER_INCREMENT = 10'h001;
  ```
Hardware Modification

- clocks_0.v, line 69 (after each generation):
  parameter SYS_CLK_MULT = 1;

- Change to:
  parameter SYS_CLK_MULT = 2;

- Line 174:
  DE_Clock_Generator_System.clk2_divide_by = 2,
  DE_Clock_Generator_System.clk2_duty_cycle = 50,
  DE_Clock_Generator_System.clk2_multiply_by = 1,

- Change to:
  DE_Clock_Generator_System.clk2_divide_by = 7,
  DE_Clock_Generator_System.clk2_duty_cycle = 50,
  DE_Clock_Generator_System.clk2_multiply_by = 9,
Software Modification

- New pixel buffer requires $1024 \times 768 \times 3 = 2359296$ bytes
  - Allocate in the last (highest-addressed) memory:
    - DE2 has 8MB, allocate at address $\text{DRAM\_BASE} + 8 \times 2^{20} - 2359296 = \text{DRAM\_BASE} + 0x5C0000$
      - Leaves only 6029312 bytes for the CPUs, so allocate 1.25 MB ($0x140000$) to each CPU
    - DE2-115 has 128 MB, allocated at address $\text{DRAM\_BASE} + 128 \times 2^{20} - 2359296 = \text{DRAM\_BASE} + 0x7DC0000$
More Tips

• Avoid divide: add a fixed-point constant for \(1/768\) and \(1/1024\) and multiply these

• In Quartus, create a new file called lights.sdc with the following contents and add it to your project:
  ```
  create_clock CLOCK_50 -period 20
  derive_pll_clocks -create_base_clocks
  ```