FPGA System Specification

Revision 1 — Sørensen, Jonsterhaug, September 24, 2024

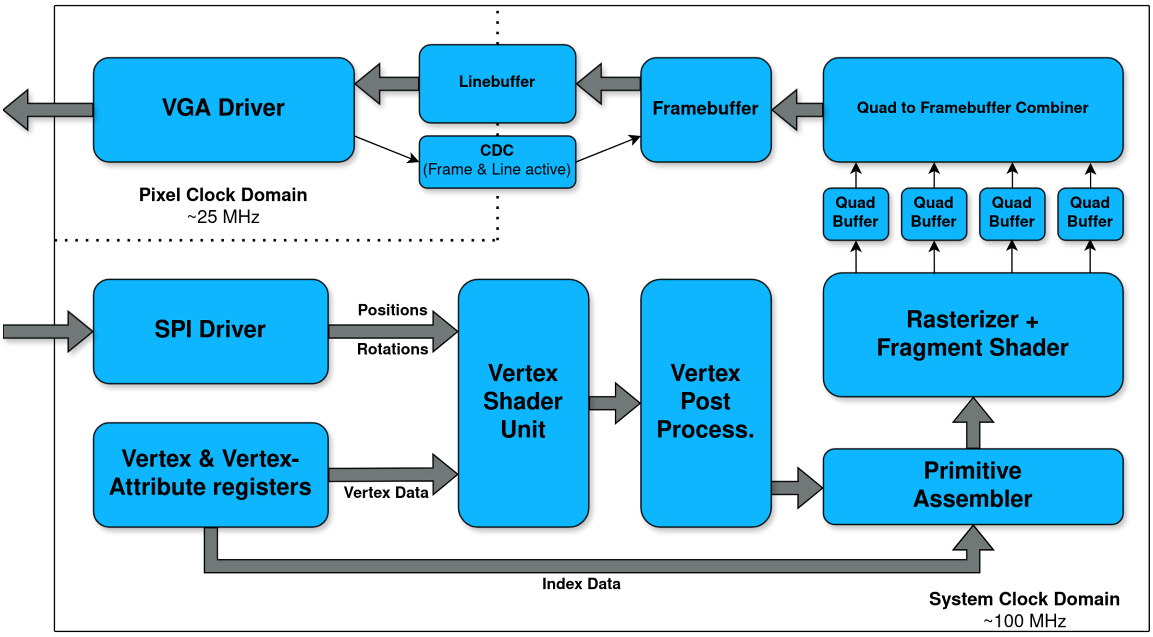
Contents

1 System Overview

The system consists of four main parts:

1. The communications system between the FPGA and the MCU
2. The Framebuffer
3. The Display Driver
4. The Render Pipeline

The following is a diagram of the entire system, it's parts, and the dataflow between modules.



The system has two main clock domains: the pixel clock domain for controlling the display at around $25\,\mathrm{MHz}$, and the system clock domain for the rest of the system at around $100\,\mathrm{MHz}$.

The render pipeline is the main part of the system. It's what does the graphics processing, taking in mesh data and outputting pixels.

2 FPGA-MCU Communication

2.1 SPI Driver

2.2 Data Protocol

The data protocol for the communication between the MCU and the FPGA is shown in figure 1.

[[FIGURE SHOWING PROTOCOL]]

Figure 1: Data protocol

First byte represents the number of entities that is to be rendered to the screen, called *NUM_ENTETIES*. The following 3 bytes represents the camera yaw and pitch angles in relation to the y and x axis repsectively. The three bytes are split into two 12-bit fixed-point numbers of type Q1.11, where the MSB is the sign bit. This means that each angle represents a number in the range [-1, 0.99951171875], which maps to the range [-π, π].

After that follows *NUM_ENTETIES* accounts of entity data to be rendered, where the first entity is the player. The entity data encoding is as follows:

1. **Byte 1 – 8:** The 10 MSB bits are flags for each entity (TBD), then follows the *x, y* and *z* position of the entity, each of which are 18-bit fixed-point numbers on the form Q7.11.
2. **Byte 9 – 11:** Entity rotation in pitch, yaw, and roll, each represented with an 8-bit fixed-point number in a Q1.7 format (again MSB is sign).

The flag bits can be decoded as follows:

Figure 2: Flag bit decoding

3 Framebuffer

4 Display Driver

4.1 VGA Signals

5 Mesh and Attribute Storage

6 Render Pipeline

6.1 Vertex Shader

Algorithm 1 Vertex Shader

$vertices, mvp \leftarrow$ input ▷ Model-View-Projection matrix
for all $v \in vertices$ **do**
  $out\_vertex \leftarrow mvp \cdot \text{vec4}(v, 1)$

6.2 Vertex Post-Processor

Algorithm 2 Vertex Post-Processor

$vertices \leftarrow$ input
for all $v \in vertices$ **do**
  $out\_vertex \leftarrow v/v.w$
  $out\_vertex.x \leftarrow (out\_vertex.x + 1) \cdot \text{screen\_width}/2.0$
  $out\_vertex.y \leftarrow (1 - out\_vertex.y) \cdot \text{screen\_height}/2.0$
  $out\_vertex.z \leftarrow out\_vertex.z/(\text{zfar} - \text{znear})$

6.3 Primitive Assembler

Algorithm 3 Primitive Assembler

$vertices, indices \leftarrow$ input

for all $i \in [0, \text{len}(indices), 3]$ **do**
  $a \leftarrow vertices[indices[i]]$
  $b \leftarrow vertices[indices[i+1]]$
  $c \leftarrow vertices[indices[i+2]]$

  $normal \leftarrow (v1 - v0) \times (v2 - v0)$

  $v0 \leftarrow (a + b + c)/3$
  $cam\_to\_v0 \leftarrow cameraPos - v0$
  $dot \leftarrow cam\_to\_v0 \cdot normal$

  $triangle \leftarrow Triangle()$ ▷ Output triangle
  if $dot < 0$ **then**
    $triangle.valid \leftarrow$ **false**
    continue
  end if
  $triangle.valid \leftarrow$ **true**

  $triangle.v0 \leftarrow a$
  $triangle.v1 \leftarrow b$
  $triangle.v2 \leftarrow c$

  $triangle.normal \leftarrow normal$

  $triangle.BB \leftarrow$ ComputeBoundingBox(triangle)

  yield triangle

6.4 Rasterizer & Fragment Shader

For the rasterizer, each tile has its own buffer of binned triangles. For each triangle, it goes over the pixels in the bounding box of the triangle. For each pixel it computes the barycentric coordinate β as follows:

Algorithm 4 Barycentric coordinate computation

$v0, v1, v2, p \leftarrow$ input

$v10 \leftarrow v1 - v0$
$v20 \leftarrow v2 - v0$
$v0p \leftarrow p - v0$

$d00 \leftarrow v10 \cdot v10$
$d01 \leftarrow v10 \cdot v20$
$d11 \leftarrow v20 \cdot v20$
$d20 \leftarrow v0p \cdot v10$
$d21 \leftarrow v0p \cdot v20$

$det \leftarrow d00 \cdot d11 - d01 \cdot d01$
$beta.x \leftarrow (d11 \cdot d20 - d01 \cdot d21)/det$
$beta.y \leftarrow (d00 \cdot d21 - d01 \cdot d20)/det$
$beta.z \leftarrow 1 - beta.x - beta.y$

return $beta$ =0

Algorithm 5 Rasterizer + Fragment Shader

$triangle \leftarrow$ input
for all pixel in $triangle.BB$ **do**
    $beta \leftarrow$ ComputeBarycentricCoordinates(triangle, pixel)

    if $beta.x \geq 0$ and $beta.y \geq 0$ and $beta.z \geq 0$ **then**
        $depth \leftarrow beta.x \cdot v0.z + beta.y \cdot v1.z + beta.z \cdot v2.z$

        if $depth < depthBuffer[pixel]$ **then**
            $depthBuffer[pixel] \leftarrow depth$
            $color \leftarrow beta.x \cdot v0.color + beta.y \cdot v1.color + beta.z \cdot v2.color$
            $framebuffer[pixel] \leftarrow color$
        end if
    end if