FPGA GPU Documentation

-James Guo

Introduction

This project is a Continuation of my previous project FPGA-PPU, where I built a Pixel Processing Unit (PPU) on an FPGA board. The PPU is used by the Nintendo Entertainment System for graphic processing and was revolutionary at its time. This project aims to create hardware that can do real-time graphic processing instead of drawing sprites, at the end goal of modularizing the hardware and creating 3D graphics using modern computer graphic principles.

The project is still a work in progress. I currently implemented a triangle drawing algorithm, and am working on implementing the 3D graphics calculation/rendering.

Top-level Module Overview

As shown below, Figure 1 is the top level view of the GPU. There are 3 layers that processes the pixels and eventually goes to the LCD display to be visible to the user. The detailed implementation of the submodules are discussed in the section below.

This is a work in progress and the top level module will be subjected to change.

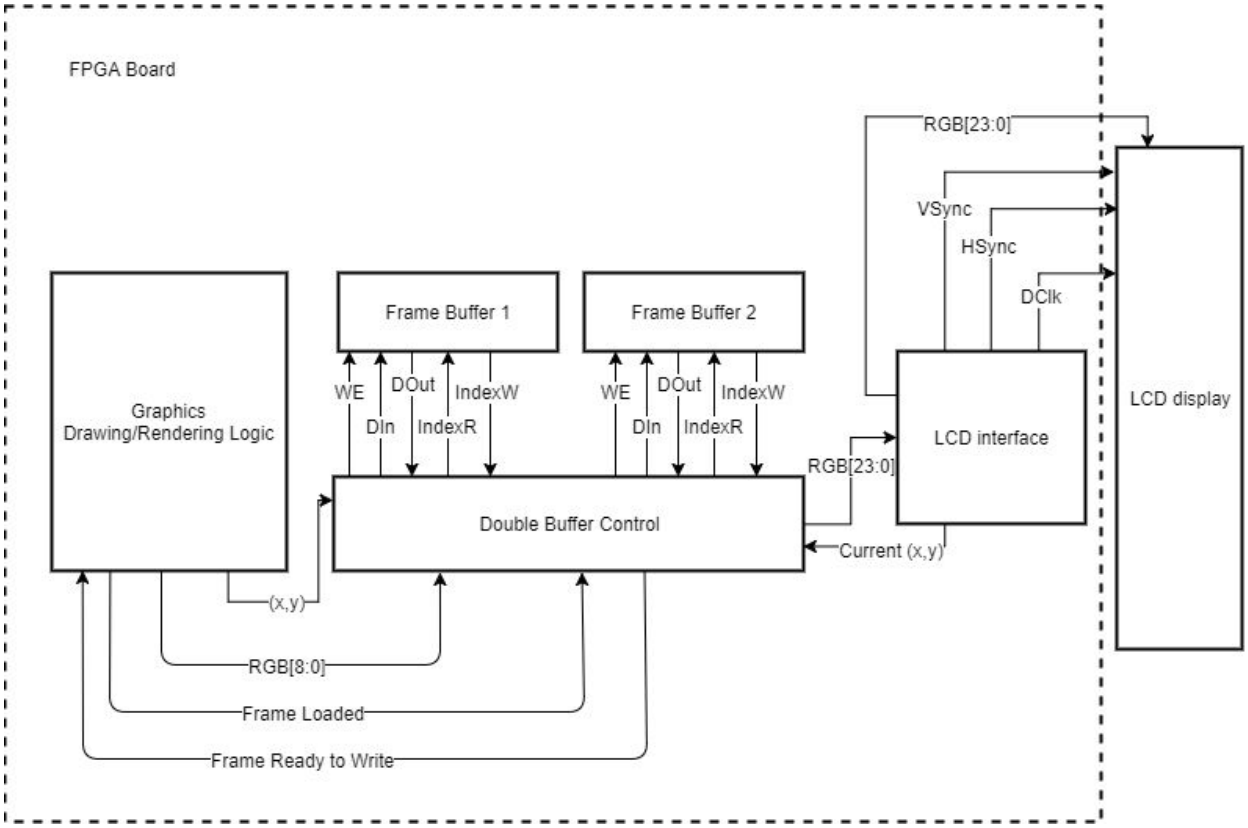

Figure 1. The top level modules of the GPU.

3. Submodules

I.LCD interface

The LCD that came with the MTL2 board is already wired up to the GPIO_1 module of the De1-SoC FPGA. The screen has 800*480 pixel resolution. The pin diagram is shown in the figure below. All the pins are declared accordingly in the LCD interface top level module.

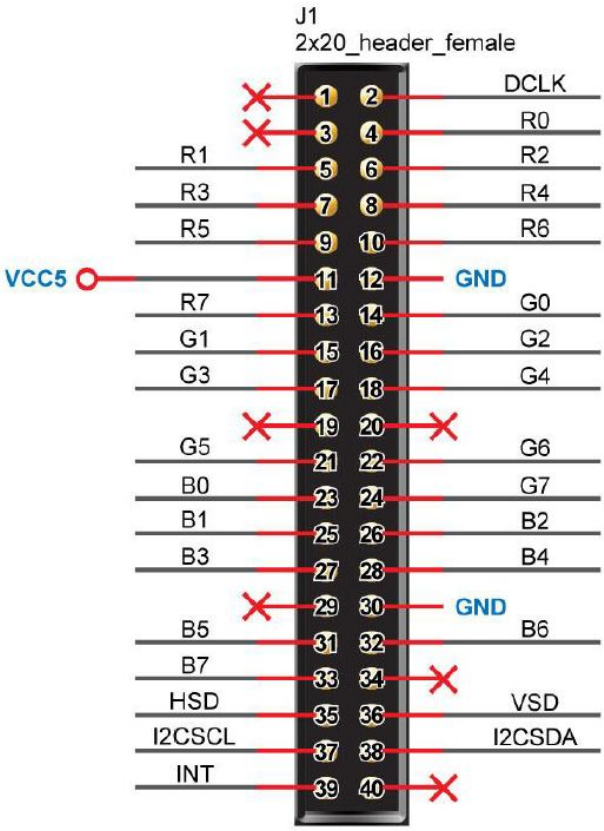

Figure 2. The pin layout for the LCD screen connecting to the FPGA board

The LCD uses a VGA protocol where DClk, Vsync, Hsync pulses are provided to determine and time the current rendering pixel and frame. The Dclk determines the rate of writing individual pixels, the Hsync determine horizontal line transitions and the Vsync determines frame-to-frame transition of the LCD screen. The LCD screen receives and draws pixels starting from top right, going down line-by-line until reaching the bottom-right of the screen. For convenience I set the top-left pixel to be (0,0) and the bottom-right pixel to be (799,479) with cartesian coordinates.

The waveform figure shown in the figure below from the datasheet shows how the signals relate to each other in this protocol, and the Tables shown below shows the timing specifications of each signal. These resources can be found in the De1-SoC-MTL2 datasheet.

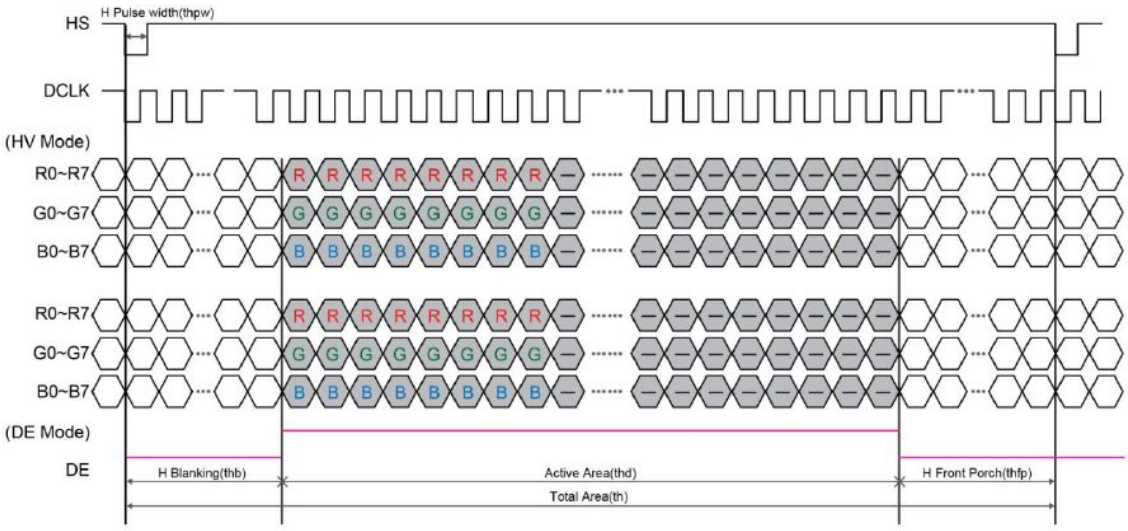

Figure 3-2 Horizontal input timing waveform

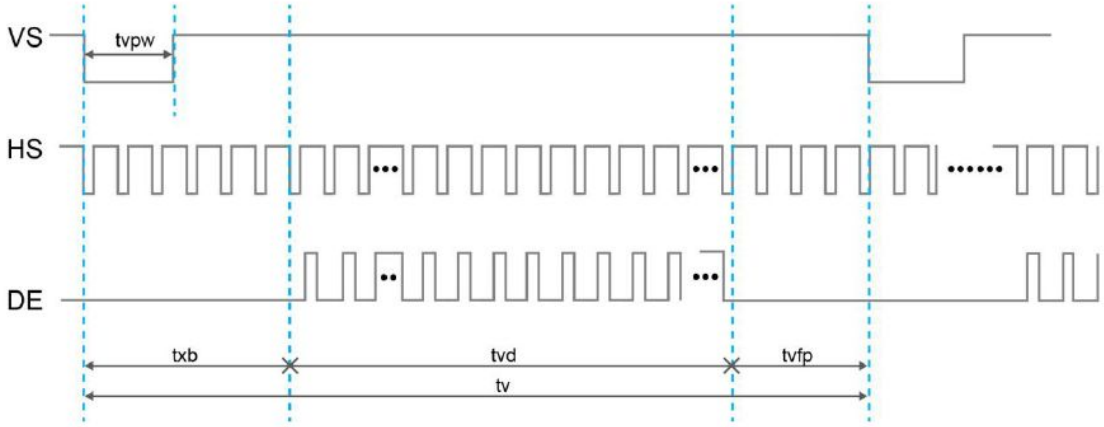

Figure 3-3 Vertical input timing waveform

Table 3-2 LCD Horizontal Timing Specifications

Item	Symbol	Typical Value			Unit
		Min.	Typ.	Max.	
Horizontal Display Area	thd	-	800	-	DCLK
DCLK Frequency	fclk	26.4	33.3	46.8	MHz
One Horizontal Line	th	862	1056	1200	DCLK
HS pulse width	thpw	1		40	DCLK
HS Blanking	thb	46	46	46	DCLK
HS Front Porch	thfp	16	210	354	DCLK

Table 3-3 LCD Vertical Timing Specifications

Item	Symbol	Typical Value			Unit
		Min.	Typ.	Max.	
Vertical Display Area	tvd	-	480	-	TH
VS period time	tv	510	525	650	TH
VS pulse width	tvpw	1	-	20	TH
VS Blanking	tvb	23	23	23	TH
HS Front Porch	tvfp	7	22	147	TH

The LCD interface sends RGB values of pixels to the screen. RGB values are simply 24 bits of data that determines the amount of red, green and blue of a certain color. The format is shown in the figure below:

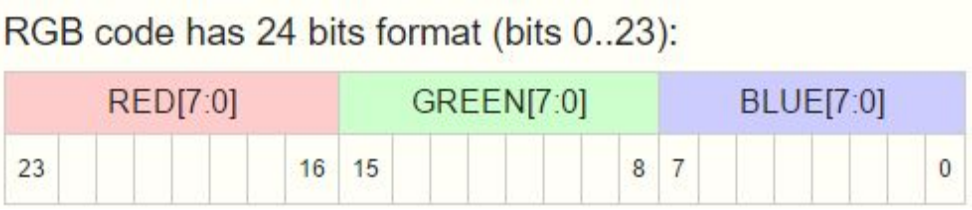

Figure 4. The RGB code format.

During the process of feeding the pixels to the screen, the LCD interface keeps track of the coordinate of the current processing pixel and outputs it for other modules to use.

II.Double Frame buffer

In modern graphic systems, multiple buffering is used for frame rendering. This means that there are multiple pieces of memory storing different frames to send to the screen. This allows one frame to be displayed on the screen while data is written to the other frames. A timing diagram for some examples of multiple buffering is shown in the figure below, which is taken from wikipedia's page "multiple buffering". (Credit: By Cmglee - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=20161108)

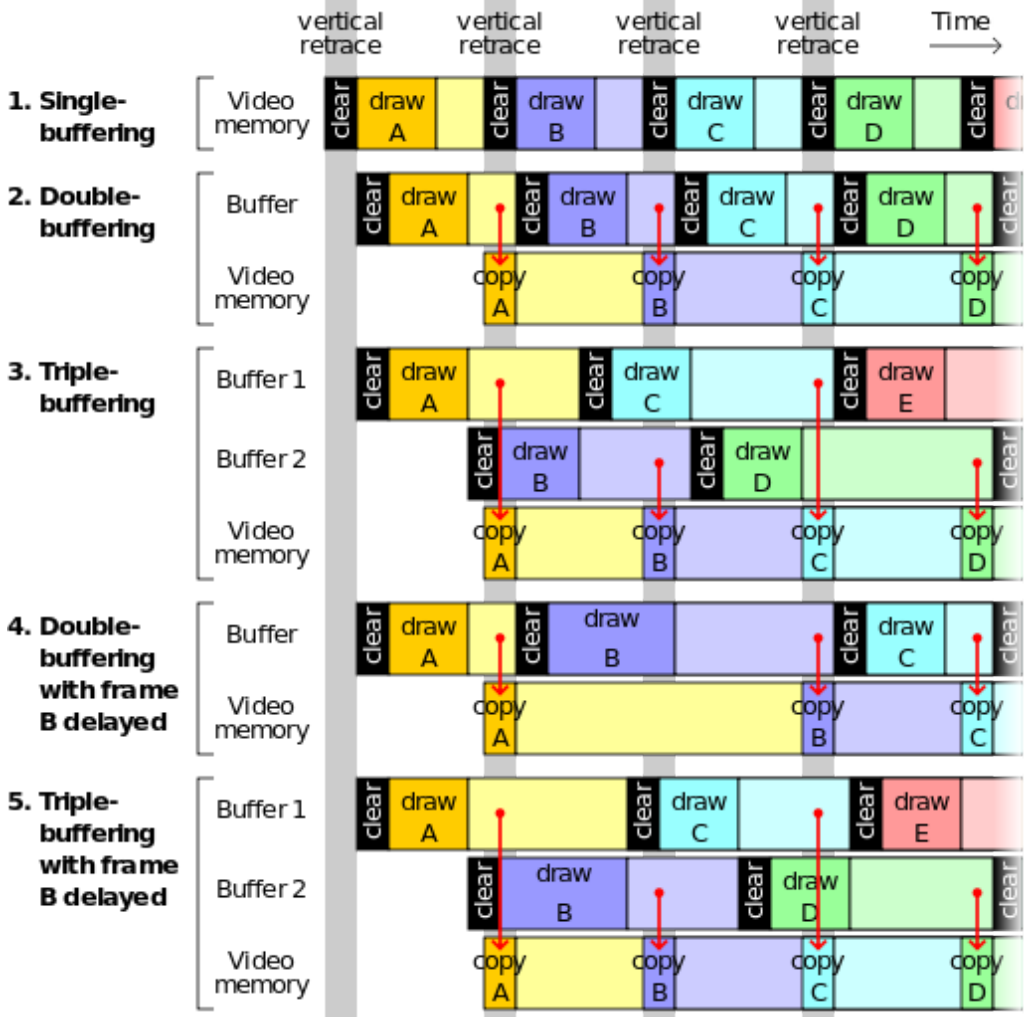

Figure 4. Process of multiple buffering for different frames.

The module in this project uses double frame buffers with frame delay support, which corresponds to structure 4 in Figure 4 shown above. When one frame is displaying on the screen, the other frame is being drawn. Delay of altering the frame when the screen refreshes is supported.

This structure ensures two benefits:
1.At any time, the frame displayed on the screen is always a completely rendered screen.
2.Frame rate can be sacrificed by delaying the frame buffers to switch between read and write mode.

The module takes data from the GPU rendering logic to draw corresponding pixels to the frame buffers. The GPU rendering logic sends a “frame loaded” signal to the double buffer control unit when the frame rendering is complete, and the frame buffer control unit send a “ready to write” signal when it cleared the writing frame and is ready to receive data.

III. Graphics drawing/rendering logic

This is how far I have gotten in my project. Subsections of different algorithms will be added once complete.

Triangle Drawing
Modern computer graphics is based on drawing numerous triangles. I have implemented an algorithm that given any 3 points/vertices, the area of the triangle is defined.

The figure shown below is an example of a triangle that needs to be drawn to the algorithm. Vertex A,B and C’s coordinates are known.

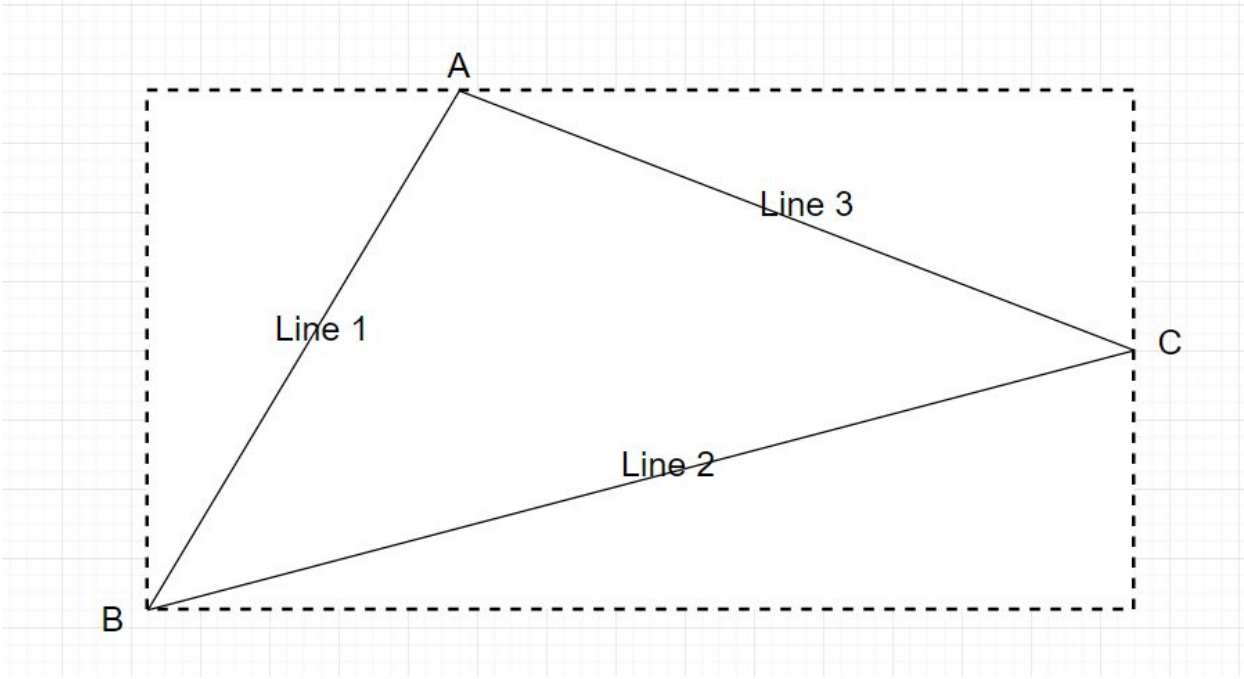

Figure 5. A triangle to be drawn on the screen.

Knowing the coordinates of A,B and C, we can determine line 1,2 and 3 in the form of y = ax +b, where a is the slope and b is the offset of the line, as defined in basic geometry.

The next step is to narrow the area we perform the algorithm to the rectangle shown in Figure 5 above, which is the smallest rectangle that contains the triangle. This can easily be done by finding the minimum and maximum value of x and y for all 3 vertices.

Within this rectangle, any pixel within the triangle is in between any 2 out of the 3 lines that make the triangle. Thus we can determine the area within the triangle by performing a pixel by pixel “for-loop” throughout the rectangle.

Using this algorithm, we can draw and define a triangle using only 3 vertices. This is very memory and resource efficient for the more complex algorithms that are based on drawing triangles.