Course Project
Design of Embedded Sytems

Hugo Bueno Buhigas
Juan Granja Garijo
Emilio Pérez del Río
Héctor Rodríguez Rodríguez

Professor: Jose Andrés Otero Marnotes

October-December 2023

Table of contents

Introduction

This project aims to design a system that displays vector images in a VGA monitor, to be implemented in a Pynq board with a Zynq 7020 System-On-Chip. This has two main components, the Processing System, with two ARM cores; and the programmable logic, the FPGA fabric.

There are several tasks to be accomplished:

- VGA signal generation to drive a monitor from a video RAM.
- Reception of list of lines to be displayed from a PC via serial port.
- Rendering of said lines into the video RAM buffer using the Bresenham's line drawing algorithm, both in software and hardware.
- Auxiliary functionality needed to feed the hardware Bresenham implementation with the endpoints of the lines to be drawn and to copy the output pixels to the video RAM.
- Additionally, rotate points to be displayed in 3D and project them in 2D in order to be able to rotate the figure being displayed.

A block diagram of the system, showing its main components is shown in Figure 1.

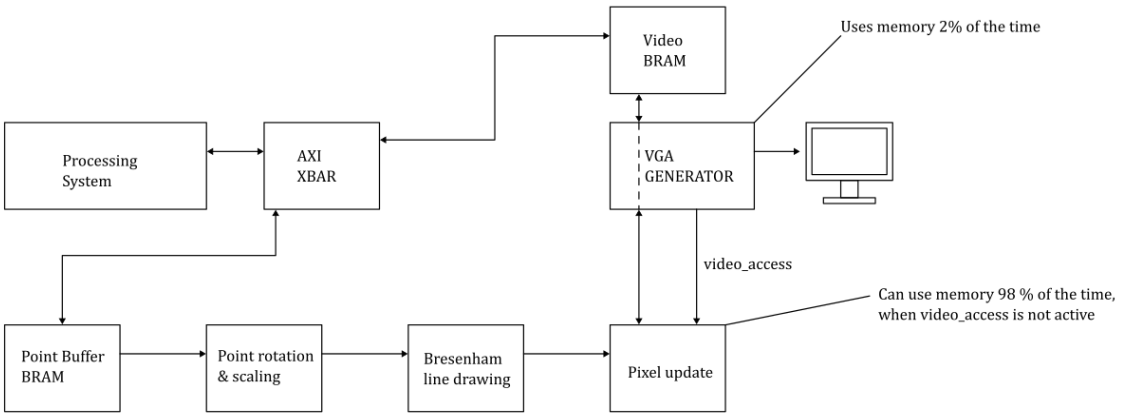


Figure 1 - System block diagram, showing its main components.

The main components are:

- **Video BRAM:** Holds the frame buffer, the image to be displayed. It is a dual-port RAM, which can be accessed both from the Processing System and from the VGA Generator IP.
- **VGA Generator:** Drives the display from the frame buffer and allows the Pixel Update IP to access said buffer whenever it is not busy.
- **Point Buffer BRAM:** Holds the endpoints of each line to be drawn.
- **Point rotation & scaling:** Reads the endpoints and feeds them to the Bresenham IP, with optional rotation and scaling, which have not been yet implemented in hardware. It is an AXI-Lite peripheral, with registers to start line drawing, status and rotation and scaling parameters.
- **Bresenham line drawing:** Hardware implementation of the Bresenham line drawing algorithm, using only integer arithmetic. Once it receives the endpoints, outputs each pixel of the corresponding line.
- **Pixel update:** Receives pixels to be written and sets the appropriate bit in the appropriate word on the frame buffer.

Communication between IPs is done in a similar way to AXI, the source IP provides data and a “valid” signal, while the destination IP receives the data and provides a “ready” signal. The transaction takes place once both “ready” and “valid” are active.

The whole hardware processing chain is limited by the speed with which pixels can be written into video RAM, which takes two clock cycles (read and write) at a clock frequency of 100 MHz.

Two solutions are explored, using AXI-Lite and DMA for data transfer. A timing comparison is performed between them.

Solution I: Memory transfers with AXI-LITE

Figure 2 Shows the block diagram of the implementation in Vivado, in which the mentioned IPs and their interfaces can be seen.

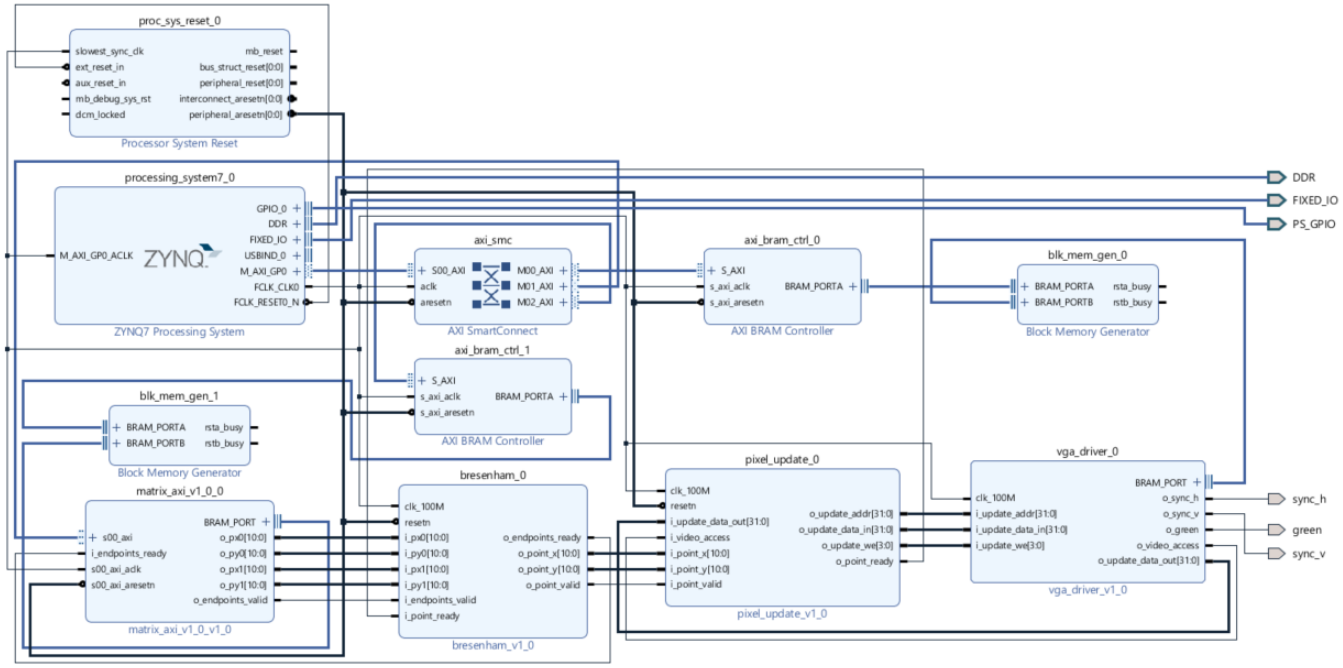


Figure 2 - Block diagram of the implementation in Vivado.

Names related to the block diagram of Figure 1 are shown below.

- **blk_mem_gen_0:** Video RAM, frame buffer
- **blk_mem_gen_1:** Point buffer RAM
- **matrix_axi :** Point Rotation & Scaling
- **bresenham:** Bresenham line drawing.
- **pixel_update:** Pixel Update
- **vga_driver:** VGA generator

Figure 3 shows the address editor, both BRAMS (endpoints and frame buffer) and the "matrix" (point rotation & scaling) IP, are all accessible from the PS.

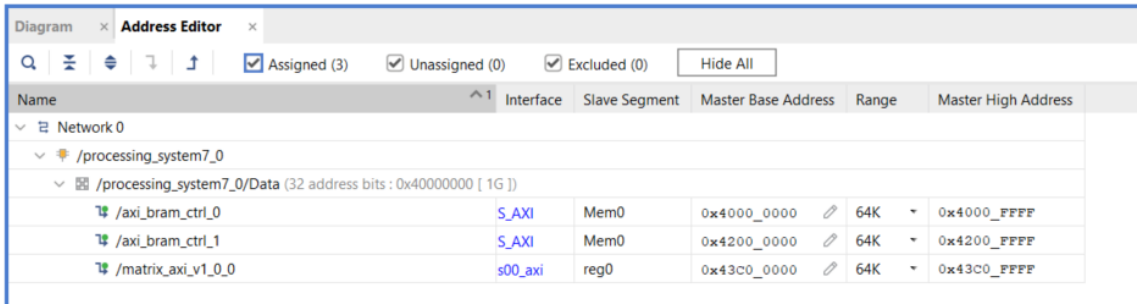


Figure 3 - Address editor, showing both BRAMS (endpoints and frame buffer) and the "matrix" (point rotation & scaling) IP, all accessible from the PS.

Solution II: Memory Transfers with DMA

Figure 4 builds on the block diagram of Figure 2, by adding CDMA blocks, which enable DMA transfers from PS's DRAM to the block RAMs in the PL. An Integrated Logic Analyzer per CDMA is added too, aiming to monitor what happens in each of them.

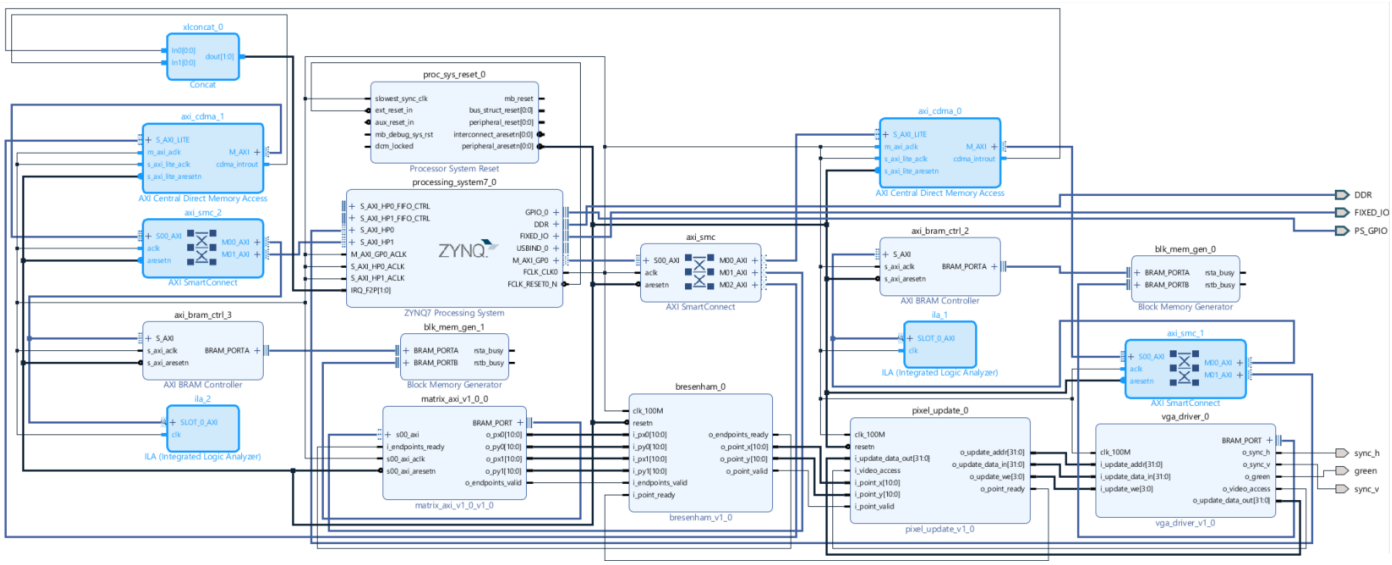


Figure 4 - Block diagram of the implementation in Vivado. Blocks added for DMA are highlighted in blue.

Figure 5 shows the new address editor view, both BRAMS (endpoints and frame buffer) are now accessible from the CDMA IPs and the CDMA configuration and "matrix" IP, are accessible from the PS.

Diagram | Address Editor

Assigned (7) | Unassigned (0) | Excluded (0) | Hide All

Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0					
/axi_cdma_0					
/axi_cdma_0/Data (32 address bits : 4G)					
/axi_bram_ctrl_2	S_AXI	Mem0	0xC000_0000	64K	0xC000_FFFF
/processing_system7_0	S_AXI_HP0	HP0_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF
/axi_cdma_1					
/axi_cdma_1/Data (32 address bits : 4G)					
/axi_bram_ctrl_3	S_AXI	Mem0	0xC000_0000	64K	0xC000_FFFF
/processing_system7_0	S_AXI_HP1	HP1_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF
/processing_system7_0					
/processing_system7_0/Data (32 address bits : 0x40000000 [1G])					
/axi_cdma_0	S_AXI_LITE	Reg	0x7E20_0000	64K	0x7E20_FFFF
/axi_cdma_1	S_AXI_LITE	Reg	0x7E21_0000	64K	0x7E21_FFFF
/matrix_axi_v1_0_0	s00_axi	reg0	0x43C0_0000	64K	0x43C0_FFFF

Figure 5 - Address editor, showing both BRAMS (endpoints and frame buffer) accessible from the CDMA IPs and the CDMA configuration and "matrix" IP, accessible from the PS.

Example executions

A cube figure is hard-coded in the C code that runs in the Processing System. However, any figure can be loaded through the serial port by using a C utility written for this purpose. The utility reads STL files, a common file for 3D object representation, based on triangles. The number of triangles is limited to 910, by the 64 KB buffer allocated for this task[1]. An example execution is shown in Figure 6.

```
 .\load_stl.exe COM9 .\Teapot_mr.stl
Num triangles: 800
Sending triangles...
Done
Received:
Number triangles: 800
```

Figure 6 - Loading an .stl file into the PYNQ board through the serial port.

Figure 7 shows the teapot figure just received via the serial port being drawn on the monitor. The PYNQ board is connected to the VGA cable using a custom adapter. The use of the hardware or software implementation can be chosen via a switch on the board, and the figure can be rotated in three axis using the pushbuttons, being redrawn each time.

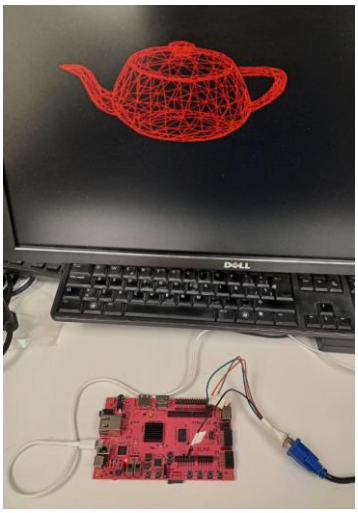
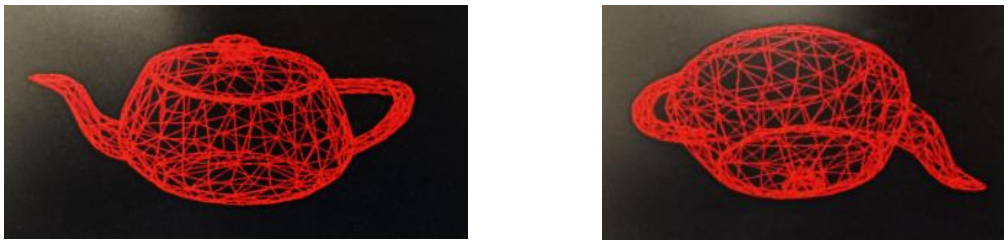


Figure 7 - Example execution, a teapot is drawn on the monitor. The PYNQ board is connected to the VGA cable using a custom adapter.

Figure 8 shows the teapot in two orientations.



a. b.

Figure 8 - Teapot MR (for Medium Resolution) in two orientations.

[1] Each triangle has three edges, each edge has two vertices which take three floating point coordinates each to describe. Each coordinate takes four bytes. Vertices common between edges are duplicated for simplicity.

Figure 9 shows the serial port output during execution, displaying the timing information for each execution, and whether it was running the hardware and software implementation or not.

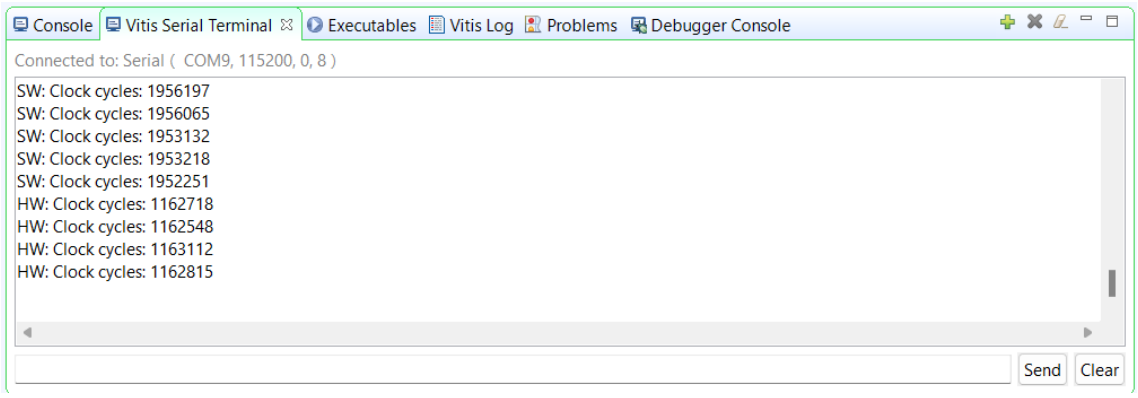


Figure 9 - Serial terminal in Vitis, the clock cycles taken for each execution are shown.

Figure 10 shows two additional figures, a lower resolution teapot and a cube.

a. b.

Figure 10 - a) Teapot LR (for Low Resolution). b) Cube.

The teapot STL was obtained from [1] and its number of triangles reduced using FreeCAD.

Timing analysis

Figure 11 shows an execution time comparison between hardware and software implementations, with and without the use of DMA, for three figures with an increasing number of triangles.

It can be seen that the hardware implementation provides a significant speedup. After the introduction of DMA, it is apparent that most of that speedup is due to the slow memory accesses on a per-pixel basis that the software solution has to perform. Hardware has direct control of the frame buffer, so it can access it every cycle.

The use of DMA to copy endpoints from DRAM to the PL BRAM slows down the hardware implementation for a reduced number of lines (Cube figure) due to its overhead. This overhead becomes less significant as the number of lines increases with the teapot figure.

For the most complex figure, the medium resolution teapot, execution time is improved both by using the hardware implementation and DMA.

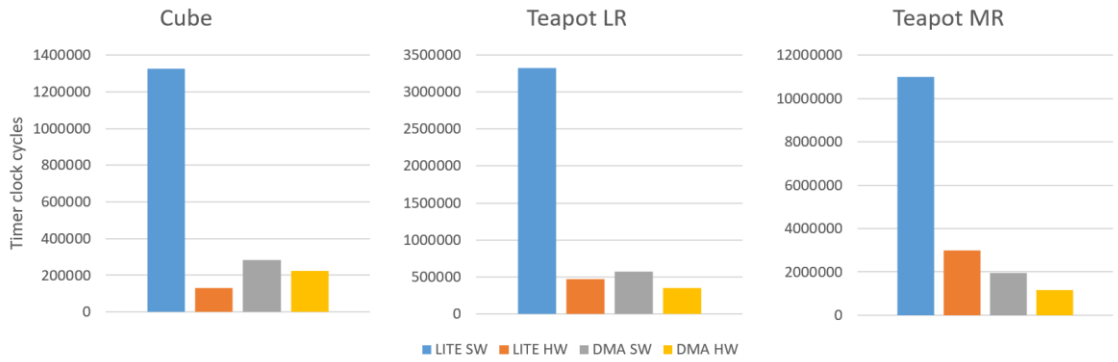


Figure 11 - Execution time comparison between hardware and software implementations, with and without the use of DMA.

Details on VGA

- **Implemented:** 800x600, 72 Hz, monochrome
- **Hardware details:** Signals driven from pins configured for 3.3V, with a 330 Ω series resistor for “red” and an 82 Ω resistor in series for HSYNC, VSYNC.

The pinout of the connector is shown in Figure 12.

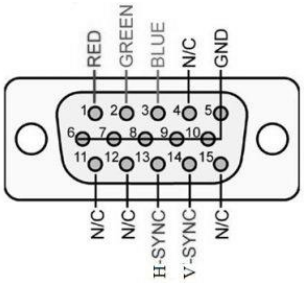


Figure 12 - VGA connector, showing R-G-B signals and H-SYNC, V-SYNC.

Sources: [2] [3] [4].

Serial port protocol for sending triangles

Newfile:

0xAA	0x10	4	Number of triangles	(Not implemented)	0x85
Header (1 byte)	Type (1 byte)	Size (1 byte)	Data (4 bytes) Uint32, little endian.	Checksum (1 byte)	Footer

Triangle:

0xAA	0x20	36	3x3verticesx4bytes	(Not implemented)	0x85
Header (1 byte)	Type (1 byte)	Size (1 byte)	Data (36 bytes) IEEE-754 float	Checksum (1 byte)	Footer

Sources

[1] Utah Teapot STL. https://en.m.wikipedia.org/wiki/File:Utah_teapot_(solid).stl

[2] VGA 800x600 timing http://tinyvga.com/vga-timing/800x600@72Hz

[3] VGA electrical characteristics http://tinyvga.com/faq/electrical

[4] VGA connector pinout https://www.elprocus.com/vga-connector/

[5] STL File Format https://en.wikipedia.org/wiki/STL_(file_format)