VGA On Zedboard

Jaeho Cho

December 17, 2024

1 VGA Peripheral

1.1 Display

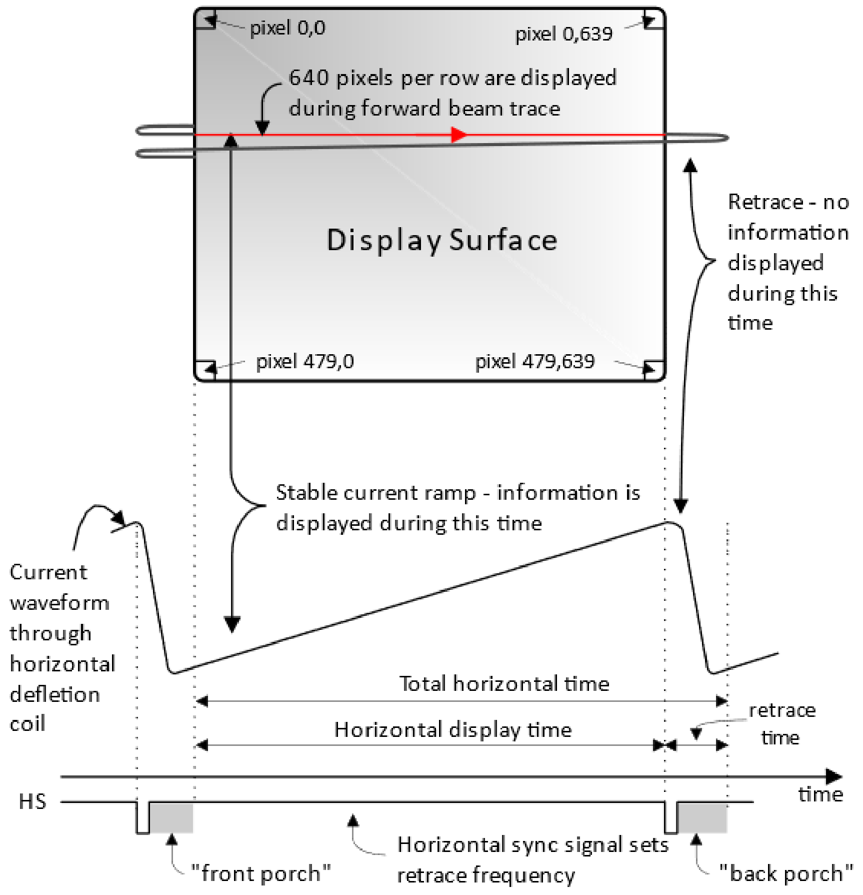
pixel 0,0
pixel 0,639
640 pixels per row are displayed during forward beam trace
Display Surface
Retrace - no information displayed during this time
pixel 479,0
pixel 479,639
Stable current ramp - information is displayed during this time
Current waveform through horizontal defletion coil
Total horizontal time
Horizontal display time
retrace time
HS
time
"front porch"
Horizontal sync signal sets retrace frequency
"back porch"

Figure 1: CRT Operation and Timing

The VGA display operates on a 640 x 480 pixel grid where pixel data is received serially as analog input, with timing signals controlling the rendering of the image. Figure **??** illustrates the timing in CRT displays, this involves a forward beam trace that lights up each pixel line-by-line, with synchronization provided by horizontal and vertical sync pulses. Key timing elements include the front porch, the interval between the end of active video and the start of the sync pulse, and the back porch, the time between the end of the sync pulse and the start of the next active video. Modern LCDs replace the beam trace with direct pixel addressing while maintaining the same VGA timing principles. The synchronization signals still play a critical role, ensuring pixel data is properly sequenced and mapped to the display grid without artifacts, even though the scanning process is no longer physical. This compatibility enables smooth rendering of video on both legacy and modern displays.

1.2 Protocol

The VGA controller protocol consists of three primary components: the clock generator, the timing generator, and the color output logic. The clock generator produces a pixel clock, synchronized to the display's refresh rate, using a clocking wizard or divider circuit. The timing generator includes horizontal and vertical counters that create the synchronization signals (HSYNC and VSYNC) based on the VGA standard. These counters manage the active video region (visible pixels) and blanking intervals, ensuring proper display refresh. The color output logic controls the RGB signals, outputting digital color values during the active video region to produce the desired image on the screen.

1.3 Connector

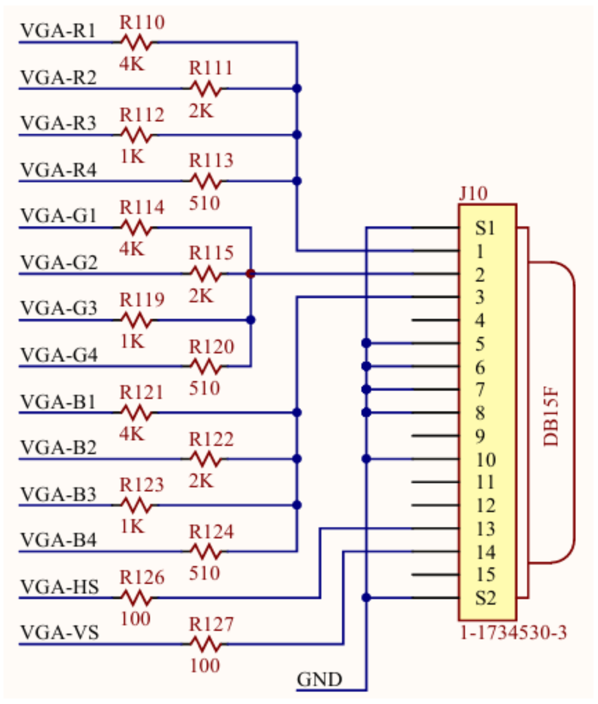

Figure 2: VGA Interface Schematic[1]

This schematic represents the Zedboard's VGA interface, connecting digital RGB (`VGA-R`, `VGA-G`, `VGA-B`) and synchronization signals (`VGA-HS`, `VGA-VS`) to a `DB15` VGA connector for output to a monitor. The RGB signals pass through a resistor ladder (`R110-R124`), converting digital signals into analog voltage levels (0–0.7 V). The Hsync and Vsync signals are set low (active state) during the synchronization pulse period, indicating the start of a new line (for Hsync) or a new frame (for Vsync). They return to high (inactive state) during the rest of the line or frame to allow the display to continue scanning and drawing the image.

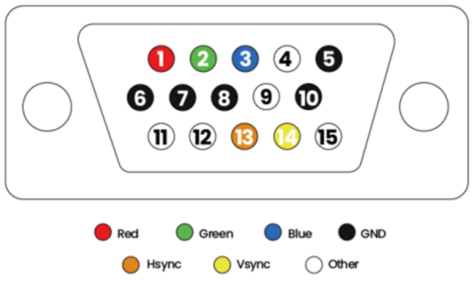

Figure 3: VGA Connector Pins

2 Implementation

2.1 Block Diagram

Figure 4: System Block Diagram

The top-level Vivado block diagram [2][3] for the Zedboard's VGA interface is designed to handle video data transfer, formatting, and timing generation. Central to this design is the ZYNQ7 Processing System (PS), which provides the high-level control and data management. A Clocking Wizard is used to generate stable, phase-aligned clock signals necessary for synchronous operation of all the video-related IP cores. To ensure proper system initialization, the Processor System Reset block manages resets for the entire design, thereby guaranteeing that all modules start in a known, consistent state.

At the heart of video data handling is the AXI Video Direct Memory Access (VDMA) core. The VDMA fetches frames from memory and converts them into a streaming video data format. This video stream is then processed by the Video Timing Controller, which provides all the necessary timing signals (horizontal and vertical sync, blanking) required by standard video interfaces. The combination of VDMA and VTC ensures that pixel data arrives at the correct time and in the proper sequence.

Once the properly timed video stream is available, the AXI4-Stream to Video Out IP core converts the AXI4-Stream video data into analog-friendly RGB signals that comply with the VGA protocol requirements [**??**]Before these signals are presented at the VGA output pins, a multiplexer (mux[**??**]) and Slice IP cores are employed to select and route the desired portions of the data word to the `VGA_R`, `VGA_G`, and `VGA_B`) lines. This final stage ensures that the correct pixel intensities reach the display.

Listing 1: mux.v

Together, these carefully orchestrated IP blocks create a streamlined, modular, and maintainable VGA output pipeline. Using standardized AXI interfaces and Xilinx-provided video IP cores, the design ensures compatibility, scalability, and simplicity with the VGA peripheral capabilities of the Zedboard.

2.2 Program

The Xilinx SDK was used to develop the software application [2] for using the VDMA IP and testing on the Zedboard.

2.2.1 VDMA

This section of the program is responsible for initializing the VDMA. `XAxiVdma_LookupConfig` retrieves the DMA configuration based on the device ID and `XAxiVdma_CfgInitialize` configures the VDMA hardware with this configuration.

This block sets the read channel parameters: `VertSizeInput` and `HoriSizeInput` specifies the vertical and horizontal size of the frame; `Stride` indicates the number of bytes in one row;

`EnableCircularBuf` ensures that playback loops over the frames continuously; `EnableSync` aligns the VDMA operations with the hardware.

2.2.2 Frame Buffers Setup

This block sets up the frame buffers: each frame buffer's base address is set using the `Buffer` array and `FrameSize` (1080p frame size) is added to `Addr` to calculate the next buffer's start address. These addresses are assigned to the VDMA using `XAxiVdma_DmaSetBufferAddr`.

2.2.3 Main Playback Loop

This is the core playback loop where `drawImage` Loads the current image into the buffer, centering it on the display. The `imageIndex` is updated each loop to show the next image and loops back to the first image after the last. `usleep(66666)` adds a delay (˜66 ms) to achieve approximately 15 frames per second.

2.2.4 Drawing an Image

The `drawImage` function overlays an image onto a display buffer. It iterates over each pixel of the display, checking if it falls within the bounds of the image (based on `hOffset` and `vOffset`. If so, it maps the corresponding pixel from the image data, calculates its RGB values, and writes them into the `Buffer` (global display memory), downscaling the color intensity by dividing each channel by 16 to match the 4 bit widths of the VGA color pins. The function ensures memory coherence by flushing the cache to physical memory using `Xil_DCacheFlushRange`.

2.2.5 Interrupt Setup

The code configures interrupt handling for the AXI VDMA (Video Direct Memory Access). It connects the interrupt controller to the VDMA's read interrupt using `XScuGic_Connect`, linking it to the `XAxiVdma_ReadIntrHandler` function and passing the VDMA instance context. If the connection fails, an error message is displayed, and the function exits. It also sets up two callback functions using `XAxiVdma_SetCallBack`: one (`ReadCallBack`) for general events like transfer completion and another (`ReadErrorCallBack`) for handling errors. This setup ensures the system can respond appropriately to VDMA interrupts during operation.

2.2.6 Buffer Generation

Python was used to process and generate the `video_buffers.h` file from an mp4 video [**??** **??**] Generating a `video_buffers.h` header file

Each image is stored in flat arrays in row-major order, where every pixel is represented with 3 bytes (R, G, B). An array `images` stores pointers to these buffers, allowing for easy access to each frame.

2.3 Result

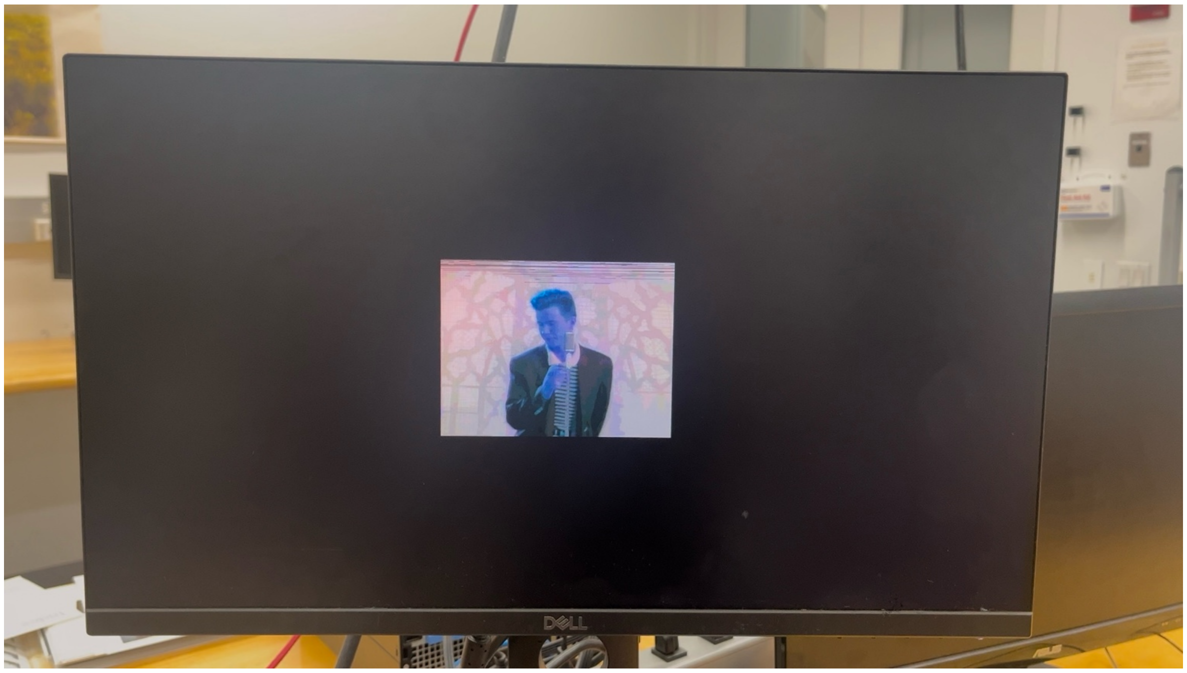

Figure 5: Video on Monitor [4]

3 Reflection

What did you enjoy about this project? What did you not like? What was the hardest part? What did you learn?

I enjoyed strengthening my understanding of the AXI interface from the previous OLED project. This project required less hand-written verilog code as most of the IP cores that I needed were already developed and on the IP catalog. This made developing the block diagram both easier and harder as I didn't fully understand what was going on in the background of each IP core, but the high level abstraction made it much faster to get to the final product. As I mentioned in the reflection for the OLED project, I wanted to rely less on the Vipin tutorial for this project, and put more time into reading documentation directly from AMD, however, in the end I still developed the majority of this project with Vipin. As most of the VGA implementation was guided with Vipin, the most difficult part of this project was implementing the color video aspect. I used python to generate the video buffers in this project, but in the future I would like to use C code to extract the video buffers directly from an mp4 file.

4 Appendix

Listing 2: display.c

Listing 3: video_process.py

Listing 4: video_buffer.py

References

[1] *ZedBoard — Avnet Boards.* URL: https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/zedboard/ (visited on 12/16/2024).

[2] *vipinkmenon/vga: Tutorial*. URL: https://github.com/vipinkmenon/vga/tree/master (visited on 12/16/2024).

[3] *Using the Zynq SoC Processing System — Embedded Design Tutorials 2023.1 documentation*. URL: https://xilinx.github.io/Embedded-Design-Tutorials/docs/2023.1/build/html/docs/Introduction/Zynq7000-EDT/2-using-zynq.html (visited on 12/11/2024).

[4] Rick Astley. *Rick Astley - Never Gonna Give You Up (Official Music Video)*. Oct. 25, 2009. URL: https://www.youtube.com/watch?v=dQw4w9WgXcQ (visited on 12/16/2024).