Visual Output: VGA

Seeing as the inspiration for this project was to create a reusable chip for gaming applications, it was critical to design a visual output for the user. The visual output would therefore confirm user input to the system and allow them to interact intelligently with the system.

The output that was chosen for this project was a video graphics array (VGA) protocol. VGA is a protocol that has been around for several decades, and has been known to be very forgiving on older cathode ray tube (CRT) monitors. However, newer digital monitors are much less forgiving in their timing constraints, so the design of the VGA protocol for the ASIC was well engineered before any implementation began. The engineering process included the following steps:

1. System clock rate for screen resolution
2. Graphics memory and timing requirements
    a. Scheduled datapath use between VGA controller and application instructions.

12.5MHz System Clock

Standard VGA protocol for a 25MHz system clock with 60Hz refresh rate on a 640x480 screen resolution is very well known. The timing details can be found online with a simple search. Originally, the overall system design was based on this protocol and was predetermined to run on a 25MHz external clock. However, it was brought to our attention that 25MHz is a rather fast frequency for student designed cells in a .6u fabrication process. Therefore, to be conservative and ensure proper functionality, a 16MHz clock was chosen instead, due to the lower clock rate and standard availability.

Designing the VGA protocol around the 16MHz clock became very trick and very limiting according to the calculations. We determined that the simplest and most efficient design would allow each pixel to last for one clock cycle. However, with the 16MHz clock, the calculations yielded that the most reasonable screen resolution for this design was 256x256. Such a low screen resolution, compared to 640x480, would be extremely limiting in applications. Thus another option was explored.

We realized that we could achieve a 320x240 screen resolution by simply dividing the system clock in half, and running the VGA protocol at 12.5MHz. The largest concern for this design was the availability of a 12.5MHz oscillator to drive the system. After exploring a few options, we determined it could be done either by using a programmable digital clock chip with an internal PLL, or to use a 100MHz crystal oscillator, pipe it through a clock divider that would output a digital clock signal every 8 clock periods (divide by 8), to yield the 12.5MHz clock. We determined the additional latency due to pipe between the clock signal and our ASIC to be negligible because all of our computations are relative to the signal that arrives at the ASIC, rather than the actual 100MHz clock. Therefore, due to simplicity, and lower overall expense vs. effort, the later design with the clock divider was chosen, which allowed for a 320x240 screen resolution.

Graphics Memory and Timing Requirements

There are three main types of representing graphics in a VGA display: bitmapped displays, glyph based displays, and direct graphics displays. The first two methodologies relate to storing pixel information within memory and accessing the pixel information based on the current pixel location being displayed. The later technique is a hardware implementation that determines pixel information based on either

current pixel location, or regions of pixels. While the hardware implementation requires no memory interface, it also limits the design of the VGA to a single application, and does not yield itself to versatility. Therefore, in order to maximize memory efficiency and flexibility with timing, we decided to implement a glyph based VGA controller. An 8-pixel x 8-pixel glyph interface was chosen, and essentially divides the screen into 40x30 glyph resolution.

In order to implement a glyph based VGA controller, two things are required in memory: a frame buffer that represents the 40x30 glyphs of the screen, and a glyph library that houses the pixel information for each glyph. The frame buffer would reside in a writable memory space (in our case, on the SRAM chip), and the glyph library, as to not be overwritten, would reside in a read only memory space (ROM chip). As previously discussed, the memory of the system is to be stored entirely off-chip. This meant that we had to ensure fetching glyph/pixel information would fit within the timing bounds allotted for the off-chip memories, and several compromises were made. By using 8x8 glylphs, we allow storage of two glylphs per word (16-bit) in memory, yielding a total of 256 possible glyphs to the application. This also means that we only need to fetch from the SRAM chip every 16 pixels, or every 16^{th} clock cycle. We also decided to use just 2-bit color, allowing us to store an entire horizontal row of pixels for a single glyp h in just one word of memory. This means that the glyph library only needs to be accessed every 8 pixels, or every 8^{th} clock cycle. Such an interface would then allow for utilization of the processor by the application during the other clock cycles not required by the VGA controller, and thus the schedule of the processor use was born and controlled through an arbiter.

The arbiter simply counts 8 clock cycles, and based on those 8 clock cycles, the first is scheduled to access the frame buffer in the SRAM, the second is used to access the glyph library in the ROM, and the remaining six are used by the application to perform whatever instructions are in instruction memory. This eliminates discrepancies in datapath usage.

Based on these number, we then allotted the first 8x8x2x256 bits of memory in the ROM to the glyph library (first 2048 addresses in the ROM chip). The frame buffer was allotted the top most 40x30/2 = 1200/2 = 600 addresses in the SRAM (really we allotted the top most 1024 address to simplify memory partitioning).

Based on these design decisions, the VGA controller was then designed and tested for correctness and functionality. A complex simulation was performed to test the VGA controller, but we would like to implement the design onto an FPGA before chip fabrication to confirm that the design passes the “eye test.”

The VGA controller was written entirely in Behavioral Verilog and includes no custom hardware layout beyond the standard cell library.