G9 Impulse-NG

Se-Joon Chung   Wenxun Huang   Chun Yang
TA: Zuofu Cheng

Fall 2010

ECE 395: Advanced Digital Projects Lab
University of Illinois at Urbana-Champaign

Abstract

Project G9 Impulse-NG is a derivation from the G9 Impulse: Video Game System project by Zuofu Cheng, James Cavanaugh, Eric Sands, Sean Bires, and Chris Schmich during the Fall 2005 and Spring 2006 semesters. As the original gaming system was a project from almost 5 years ago, this projects aim was to import the whole system to more modern hardware platform and extend its capabilities. However, due to technical difficulties, G9 Impulse-NG currently only supports drawing a single frame with transparent colors for drawing sprites.

Contents

1 Project Description and Overview

1.1 G9 Impulse (The Original System)

The original G9 Impulse was an 8-bit, 2D gaming system. It takes NES controller or custom arcade interface for its inputs, and the game is displayed on a CRT monitor at a 320x240 resolution using interlacing mode. It has 2-bit DAC for each of the red, blue, green colors, allowing a total of 64 colors. Audio is provided through an MP3 player via standard 3.5mm audio jack. For further details, please take a look at the original G9 Impulse report at `https://courses.engr.illinois.edu/ece395/projects/spring2006/project9_final_paper.doc`.

1.2 G9 Impulse-NG

The G9 Impulse-NG is still an 8-bit, 2D gaming system, but with extended hardware capabilities. It uses the Altera DE2 FPGA board which supports 10-bit DAC video out for each of the red, blue, green colors as opposed to the 2-bit DAC on the original system. Also, it is designed to output to an LCD monitor without interlacing at a 640x480 resolution. It does not support any audio.

1.3 Tools Used

The primary tool used in this project was the Quartus II 9.1/10.0 which can be downloaded on the Altera website. It was used to compile the VHDL code and to program the Altera DE2 board. Also, the MegaWizard Plug-In Manager and NIOS II SOPC Builder under the tools tab in Quartus II were used to generate some of the required entities. ModelSim-Altera 6.5b Starter Edition, which can also be downloaded on the Altera website, was used to simulate the entities and aid in debugging.

In terms of hardware, the project utilized the Altera DE2 Development and Education Board for the GPU and the PIC18F4620 as the CPU.

1.4 Description of GPU Components

Most of the project was done on the Altera DE2 FPGA board which is the graphics processing unit of the game system. The following are the components inside the GPU chip on the Altera DE2 board.

View (view.vhd)

The view entity handles outputs to the LCD monitor. It generates the vertical sync, horizontal sync, and blank signals for the LCD monitor to use and the appropriate RGB values that go with the current pixel position on the screen. The view entity uses the onboard asynchronous SRAM as the video buffer for fast access and accepts writes to the SRAM when the SRAM is not

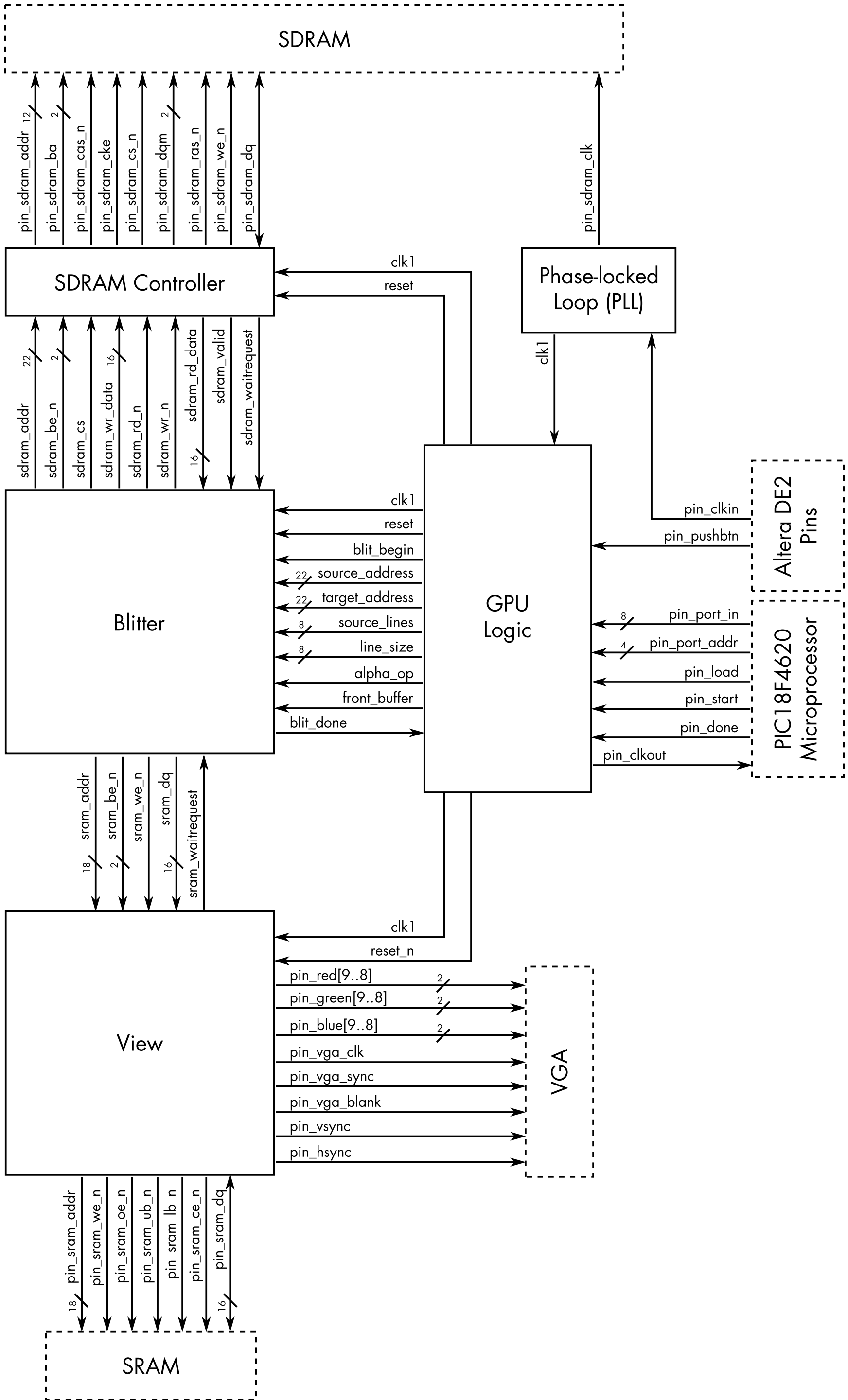

Figure 1: Block diagram of G9 Impulse-NG

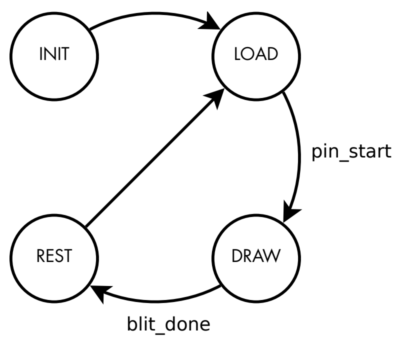

Figure 2: GPU State Diagram

being read. The subcomponents of the view entity include the VGA controller (VGA_controller.vhd) and Color Mapper (Color_Mapper.vhd).

VGA Controller (VGA_controller.vhd)

The vga_controller entity takes a 50 MHz clock as its input and generates a 25 MHz clock which acts as the pixel clock and the appropriate horizontal and vertical sync pulses. It also outputs the current coordinates on the screen as DrawX and DrawY. The code is sourced from Lab 9 of the ECE 385 (Digital Systems Laboratory) course at the University of Illinois at Urbana-Champaign.

Color Mapper (Color_Mapper.vhd)

The Color Mapper entity takes the current coordinates on the screen and decides which color to draw on the screen. Currently, the G9 Impulse-NG outputs at a 640x480 resolution, it only uses the upper left quarter of the screen (320x240) for actually outputting the game, virtually keeping the same resolution as in the original system. The Color Mapper outputs the actual game if the current pixel is in the upper left quarter. Otherwise, it just outputs a blue color.

Blitter FIFO (blitter_fifo.vhd)

The blitter fifo entity is a 512-byte, First-In First-Out buffer created by the Altera MegaWizard tool. It is implemented on the M4K memory block on the FPGA board. The buffer keeps track of current level of usage. Using this information, it blocks read when it is empty and blocks write when it is full.

SDRAM Controller (sdram_0.vhd)

The sdram_0 entity is the SDRAM controller generated by the NIOS II SOPC Builder in Quartus II. It follows the Avalon Memory-Mapped Interfaces specification as outlined in `http://www.altera.com/literature/manual/mnl_avalon_` `spec.pdf`. Out of the 22-bit address it takes as input, it uses bits 21 and 8 for the bank address, bits 20 down to 9 as the row address, and bits 7 down to 0 as the column address. One should note that this format for decoding address is different from the one they use in the DE2 Control Panel program which can be used to write data into the SDRAM. The DE2 Control Panel uses bits 21 down to 20 as the bank address, bits 19 down to 8 as the row address, and bits 7 down to 0 as the column address.

Phase-Locked Loop (sdram_pll.vhd)

The sdram_pll is the phase-locked loop entity to ensure proper delay between the system clock and the SDRAM clock. The SDRAM clock needs to be advanced by 3ns with respect to the system clock for it to function properly. The PLL entity was generated by the Altera MegaWizard tool.

Hex Display Driver (HexDriver.vhd)

The HexDriver entity takes 4-bit number as its input and outputs a 7-bit vector that can drive the onboard 7-segment display. The code was sourced from ECE 385 class. This entity is not an integral part of the GPU and was used only for debugging purposes.

Dualport RAM (ram2port.vhd)

The ram2port entity is a 512-byte, 2-port memory block created by the Altera MegaWizard tool. It uses the onboard M4K memory and supports simultaneous reads and writes. It is used inside the fifo_cc entity to implement a First-In, First-Out buffer.

fifo_cc (fifo_cc.vhd)

The fifo_cc entity is a 512-byte, First-In First-Out buffer that uses the ram2port entity as its component. It keeps track of current level of usage. Using this information, it blocks read when it is empty and blocks write when it is full.

Blitter

The blitter facilitates transferring large amounts of data in memory, from the sprite sheet to the framebuffer. Operands for the blitter are provided by registered inputs from the GPU, which ultimately originate from the PIC. The blitter uses the operands and reads the appropriate bytes from the sprite sheet and blits them onto the framebuffer. The blitter supports alpha blending which enables it to only blit certain pixels of a square sprite region onto the framebuffer.

The operands required to operate the blitter are given in table 1.

The architecture of the G9 Impulse-NG was redesigned and resembles the original G9 Impulse blitter only in function and I/O. Originally, the sprite sheet and framebuffer both existed in the SDRAM. However, due to complications in

Operand	Description
Source Address	Address in sprite sheet to blit from
Target Address	Address in framebuffer to blit to
Source Lines	Number of rows to blit
Line Width in pixels/2	Width of each row (divided by two because each address in the SDRAM stores data for two pixels)
Alpha Op	Whether or not to enable alpha blending
Front Buffer	Specifies which framebuffer to draw to

Table 1: Blitter Operands

reading and writing to the SDRAM due to the lack of a dualport module for the Altera SDRAM controller, we decided to use the SRAM to hold the framebuffer. Using the SRAM also has the advantage of simplicity and speed, as writes to the SRAM can be completed within one clock cycle. The SRAM must also be used by the view entity to calculate the appropriate colors to output to the VGA device, so access to the SRAM must be arbitrated. To make sure there are no conflicts in using the SRAM, the view entity sends a sram_waitrequest to block writes when the screen is being drawn. Particularly, since we are only drawing the framebuffer to the upper left quadrant of the screen, the blitter is allowed to write when the VGA is drawing the bottom half of the screen. Figure 3 shows when the blitter is allowed to write to SRAM.

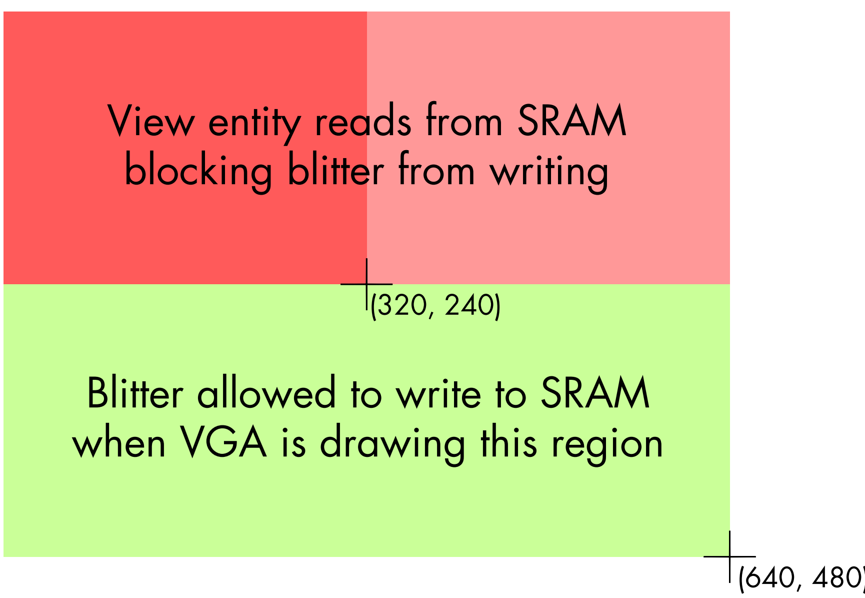

Figure 3: How SRAM access from blitter and view is divided

The blitter is comprised of two processes and a queue. One of the processes reads sprites from the sprite sheet and puts the data into the queue. The other process reads from the queue and writes the data into the framebuffer when the blitter is given permission to do so. A diagram of the interaction is given in figure 4

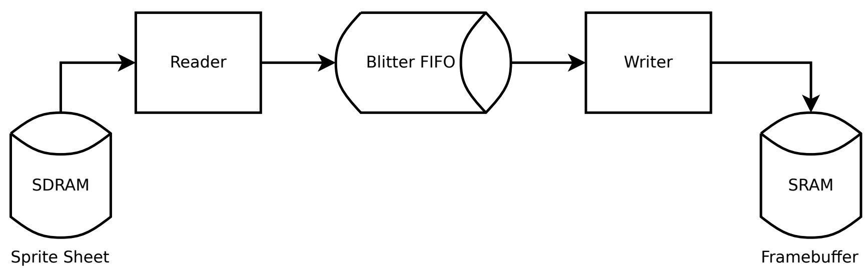

Figure 4: Diagram of how the blitter works

The reason this design was chosen was to take advantage of being able to read from the sprite sheet and write to the framebuffer simutaneously. In the original G9 Impulse, this was not possible because both operations would involve accessing the SDRAM, but since the G9 Impulse-NG uses the SRAM for the framebuffer, using this architecture results in a slight speed increase because reads and writes can occur simultaneously.

Since reads from the SDRAM are pipelined, data is available several seconds after a read is issued. Using the queue allows the blitter to catch all the data read from the SDRAM independent of whether the writer is able to write the data immediately. When the VGA is drawing the screen, the writer will be blocked from accessing the framebuffer. With the queue in place, though, the reader will still be able to read from the sprite sheet until the queue is full.

Again, because of pipelining, data may arrive even after read is deasserted. Thus, the queue must always have enough room to accept the data. Therefore, an 'almost full' signal determines whether or not reads are allowed.

State diagrams for the blitter is given in figure 5.

The processes behave almost identically except for the direction of data flow and the conditions which allow the process to operate. When in the STANDBY state, the two processes wait for the blit_begin signal to begin processesing. Upon receiving the signal, various registers are loaded/reset with information about the source address, target address, current row being read/written, and current position being read/written in the current row. During the READ and WRITE states, the two processes respectively read from the sprite sheet and write to the framebuffer. Reads occur when the queue is not full and the SDRAM waitrequest is not asserted. Writes occur when the queue is not empty and the SRAM waitrequest is not asserted. Both processes finish when the number of lines read/written is equal to the number of source lines.

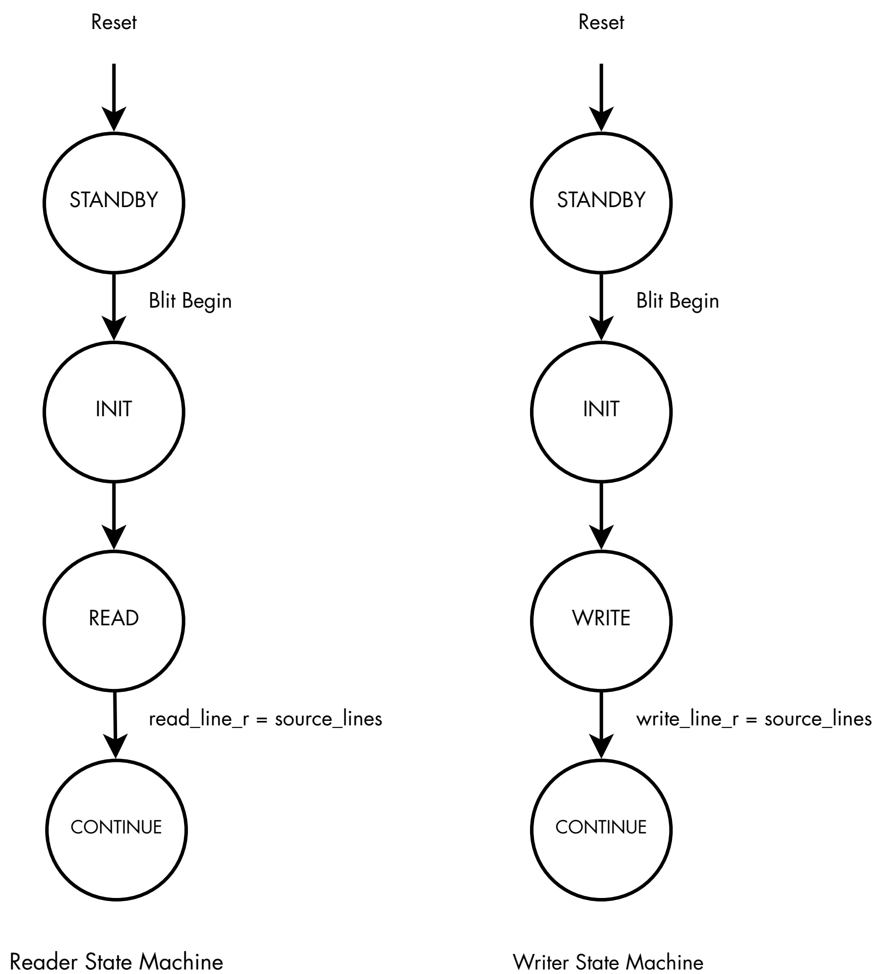

Figure 5: Blitter state diagrams

Due to two pixels being stored in each sprite sheet address, the original G9 Impulse implemented alpha blending by reading both from the framebuffer and from the sprite sheet and blitting one or both pixels based on the color and the value of the alpha op operand. The Xilinx SDRAM controller used by the G9 Impulse did not support writing and reading to only one byte. The Altera SDRAM controller used by the G9 Impulse-NG however does support a byte enable signal. Thus, the G9 Impulse-NG blitter is able to perform alpha blending without reading from the framebuffer. The result of this is that when blitting large sprites with alpha blending on, the new blitter will be faster because there will be less memory access.

When finished blitting, the blitter asserts the blit_done signal. To blit an-other sprite, the blitter must be reset. The logic inside the GPU Chip takes care of handling the blit_done signal and resetting the blitter between operations.

GPU Chip (gpuchip.vhd)

The gpuchip is the top level entity inside the Altera DE2 board. It takes source address, target address, source lines, source width, and alpha blending as its inputs from the external PIC18F4620 microprocessor and passes this information onto the blitter. When the blitter operation is done, the gpuchip also signals the PIC18 that the operation has been completed. More specifically, the pin_port_addr should contain the address to the registers which hold the values and the pin_port_in should contain the data to the register when the pin_load is asserted high. When this happens, the gpuchip loads the corresponding registers with these values. After all the information has been updated, the CPU should then assert pin_start to indicate that the blitter operations can now begin. However, due to technical difficulties in the hardware wiring from the DE2 board to external devices, the current system does not utilize any external inputs outside of the FPGA board. The instruction set of the GPU is shown in table 2.

Address	Instruction
0000-0010	Source Address MSB-LSB
0011-0101	Target Address MSB-LSB
0110	Source Lines
0111	Line Width in pixels/2
1000	Alpha Op Register
1001	Double Buffer Enable
1010	Front Buffer Register

Table 2: GPU Instruction Set (from G9 Impulse Final Documentation)

1.5 Description of Software Components (Starcell XF-1 Game)

The original G9 Impulse came with vertical scrolling shooter arcade game written for the system titled Starcell XF-1. The game code runs on the PIC18F4620 microprocessor which is the main CPU of the system. The game code was not actually utilized because of technical difficulties in wiring the DE2 board with the PIC. More specifically, the ribbon cable we used to connect the 40-pin expansion header on the DE2 board to the protoboard we used to put the PIC18 and other related components seemed to distort the signals. However, images from the game was converted into a binary file and used extensively in testing functionalities of gpuchip.

2 Results

Most of the time devoted to the project was spent on figuring out how to use the SDRAM on the Altera DE2 board. There were no SDRAM controllers in VHDL provided by the manufacturer or good documentation on what the NIOS II generated SDRAM controller does, so there had been a lot of guesswork going into how to use the SDRAM controller. There had also been an attempt to use the Xilinx XSB SDRAM controller on the DE2 board, but this approach was dropped after realizing that the Altera DE2 board does not have hardware support for delay-locked loops. Using ModelSim to simulate these controllers aided greatly in having breakthroughs towards the end of the semester. After getting the NIOS II controller working, it was noticed that the screen was scrambled. The problem was found to be the way the address is decoded inside the DE2 Control Panel which is used to initialize contents of the SDRAM is different from the way the address is decoded inside the SDRAM controller which reads out of the memory. In the end, the vga entity and the blitter was redesigned to integrate the correct SDRAM controller, but there was not enough time left for debugging the glitches in the synchronization between components. Also, the PIC18F4620 hardware had to be left out of the picture because of bad connection cable. The current G9 Impulse-NG hardware can display a static frame with alpha blending.

3 Future Development

As it has been discovered how to access the SDRAM and other required components inside the gpuchip, if the synchronization issue between these components can be resolved, it should be relatively easy to display a scrolling background and draw sprites that move around. Audio may be added using the onboard support for an audio output. However, it has not been tested if the expansion headers can be used correctly to receive external inputs into the DE2 board and there still may be some difficulty in interfacing the board with a PIC18. It is advised that a future developer continuing this project consider using a Xilinx FPGA board rather than the Altera board, as Xilinx provides more resources and documentation for developers.

Bibliography

[1] Using the sdram memory on altera's de2 board with vhdl design. http://www.cs.columbia.edu/~sedwards/classes/2010/4840/tut_DE2_sdram_vhdl.pdf, 2008.

[2] Zuofu Cheng, James Cavanaugh, Eric Sands, Sean Bires, and Chris Schmich. G9 impulse: 2d video game system. https://courses.engr.illinois.edu/ece395/projects/spring2006/project9_final_paper.doc, Spring 2006.

[3] Altera Corp. De2 development and education board user manual. ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf, 2006.

[4] Altera Corp. Avalon interface specifications. http://www.altera.com/literature/manual/mnl_avalon_spec.pdf, August 2010.

[5] Altera Corp. Phase-locked loop (altpll) megafunction user guide. http://www.altera.com/literature/ug/ug_altpll.pdf, November 2010.

[6] XESS Corp. Dualport module for the sdram controller. http://www.xess.com/appnotes/an-071205-dualport.pdf, July 2005.

[7] XESS Corp. Xsa board sdram controller. http://www.xess.com/appnotes/an-071205-xsasdramcntl.pdf, July 2005.

[8] XESS Corp. Xsa board sdram controller. http://www.xess.com/appnotes/an-071205-xsbsdramcntl.pdf, July 2005.