G9 Impulse: 2D Video Game System

By

Zoufu Cheng
James Cavenaugh
Eric Sands

ECE 395, Advanced Digital Projects

FALL 2005

TA: Mark Dykstra

Dec 13, 2005

ABSTRACT

The Impulse Video Game System is an 8-bit two dimensional platform. It has a custom built graphics card. The CPU is hard-coded with a release title. Audio is supplied through an MP3 player.

Table of Contents:

1. INTRODUCTION

1.1 Overview

The G9 Impulse is an 8-bit, 2-dimensional gaming system. The input can be any NES controller and the game is displayed on a CRT computer monitor at a 320x240 resolution with 64 colors. Audio is provided through a standard digital audio jack.

1.1.1 Hardware

This system is comprised of a custom-built graphics processor. The graphics processing and VGA interface are implemented on an XESS XSA-50 board. This board provides an FPGA, SDRAM, and VGA output that all contribute to the design. The CPU is an 8-bit PIC18F4620. This processor handles the game logic and I/O. Currently the audio processor is also a PICF18F4620. The input module is a PIC16F628A. It serves as a buffer between the CPU and the NES controller.

For programming and debugging purposes, MPLAB ICD 2 is utilized. This is an in-circuit serial programmer available at http://www.microchip.com.

The system is housed in a Plexiglas pyramid. This was chosen because other gaming consoles are box-like, i.e. XBOX and Game Cube. Illuminative wires line the inside of the casing to give the system an even more unique look. These wires are called EL wire (electroluminescent) and can be found at http://www.elwirecheap.com.

1.1.2 Software Tools Used

The XSA-50/100 boards were programmed using Xilinx Project Navigator in VHDL. All code for the microcontrollers was written in C. The compiler used is SourceBoost IDE and is available at http://www.picant.com/c2c/c.html. This compiler was utilized in MPLAB for programming purposes.

2. DESIGN ALTERNATIVES

2.1 I/O

For the I/O of the console, we were debating if we were going to use the arcade machine as our controls, or if we were going to go with a classic NES controller, as well as a PS controller, or a serial input from the computer. With the exception of the PS controller, all of these would be easily implemented. We had the serial input and the NES controller working. Once we got the NES controller working, we focused our time on other aspects of the design and stopped looking at alternatives to input.

2.2 Audio

As for audio, we were originally planning on using a MIDI synthesizer which would get the midi info from the SPU. The SPU would take the midi information from an EEPROM which would hold the midi stream. We got the EEPROM to work, and we had a MIDI synthesizer working with it, but there was not enough room to play entire songs with the EEPROM's available memory. We chose instead to use an existing MP3 player as our audio. We did this for ease of control. Instead of dealing with the actual streaming audio signal, we only had to control the MP3 player's buttons, i.e. play, stop, forward, reverse. Another reason we decided to go with the MP3 was because our sound designer did not want to work with midi.

2.3 Memory

We looked into using an external RAM, to reduce ram accesses from the SRAM on the FPGA. The reason we did not choose to go with this method is because we did not have enough pins on the FPGA to implement it easily. This device also drew a lot of power which caused our voltage regulator to become hot. In addition to the reduced access, we needed a memory chip to store the graphical data for the game because the ram on the FPGA was volatile.

3. DESIGN DETAILS

3.1 GPU

The GPU essentially takes graphics from the SDRAM and places them on a CRT monitor according to the instructions from the GPU processor. A blitter module takes care of adding sprites from the ram to the background. The VGA interface generates the video signal that is sent to the CRT monitor.

3.1.2 SDRAM Interface

The SDRAM is 8MB and is word addressable (1word = 2 bytes). The interface has two ports that can be simultaneously accessed. This is helpful as we are accessing the ram to read sprites, read the VRAM, and write to the VRAM.

3.1.3 Blitter

The blitter receives instructions from the CPU pertaining to sprite operations. The blitter must then read the sprite from memory line by line and write it back into the VRAM. This is accomplished by a sophisticated state machine and two FIFO buffers.

The instructions issued to the GPU are indicated in Table 3.1. Because the CPU is only 8 bits wide, the sprite source address and target address must be sent in three consecutive clock cycles. The source address gives the beginning address of the sprite in memory while the target address indicates where the sprite is to appear in the VRAM. The CPU must then indicate the height and width of the sprite by sending the number of source lines and the line width respectively. The line width is actually the number of pixels in the width divided by two because two pixels are stored in a single SDRAM word.

Table 3.1 GPU Instuction Set

Address	Instruction
0000 - 0010	Source Address MSB - LSB
0011 - 0101	Target Address MSB - LSB
0110	Source Lines
0111	Line Width in words Div/2
1000	AlphaOp Register
1001	Double Buffer Enable
1010	Front Buffer Register
1111	To SPU*

* Indicates Instruction used for Sound Processor Only

The blitter then uses this information to extract the sprite one line at a time into the FIFO buffer. Once an entire line has been read from the SDRAM, the blitter then decides whether or not to implement alpha blending. If alpha blending is not indicated by the CPU in the AlphaOp instruction, the line is written directly to the background with no alterations. If the CPU indicates an AlphaOp, the blitter must not write pixels that have the color hexadecimal coding 'EF'. This operation is complicated by the word size in SDRAM. Because only an entire word (2 pixels) can be read or written to memory, a pixel that should have been drawn to the background might not be or vice versa. The solution to this is to read the current background at the target address into another FIFO buffer. By reading both FIFO buffers in parallel, each pixel can be inspected and the correct one chosen to be written to the background.

Once an entire line has been written to the background memory, the next source address is computed by adding the background width to the source address. The next target address is computed in a similar way and the next line is read. This process continues until the last line is reached. Once this occurs, the blitter asserts to the CPU that the operation is complete and waits for a reset. The blitter must be reset before another sprite can be drawn.

The blitter state machine is shown in Figure 3.1. It describes the operations every time a new sprite is written to the VRAM. Once the blitter is reset, it waits for the information from the CPU to be read. Once this information is available, it proceeds with drawing the sprite.

The initialization process is completed in states INIT1 and INIT2. During these two processes, the target buffer in VRAM is determined and counters are cleared.

The read states, READ and READ_B, both read from the SDRAM and write to their corresponding FIFO buffers. They read an entire line from SDRAM one word at a time.

The Empty pipe states both wait for the SDRAM to finish reading. Once the SDRAM is done, this state initializes the registers to write to VRAM. The first empty pipe state looks at the AlphaOp instruction to determine whether to go to READ_B state.

The write states read from the FIFO buffers then decide which pixel to write to the VRAM. Once the FIFO buffers are empty, the write is done and the state is changed to the STOP state where a 'done' signal is sent to the CPU.

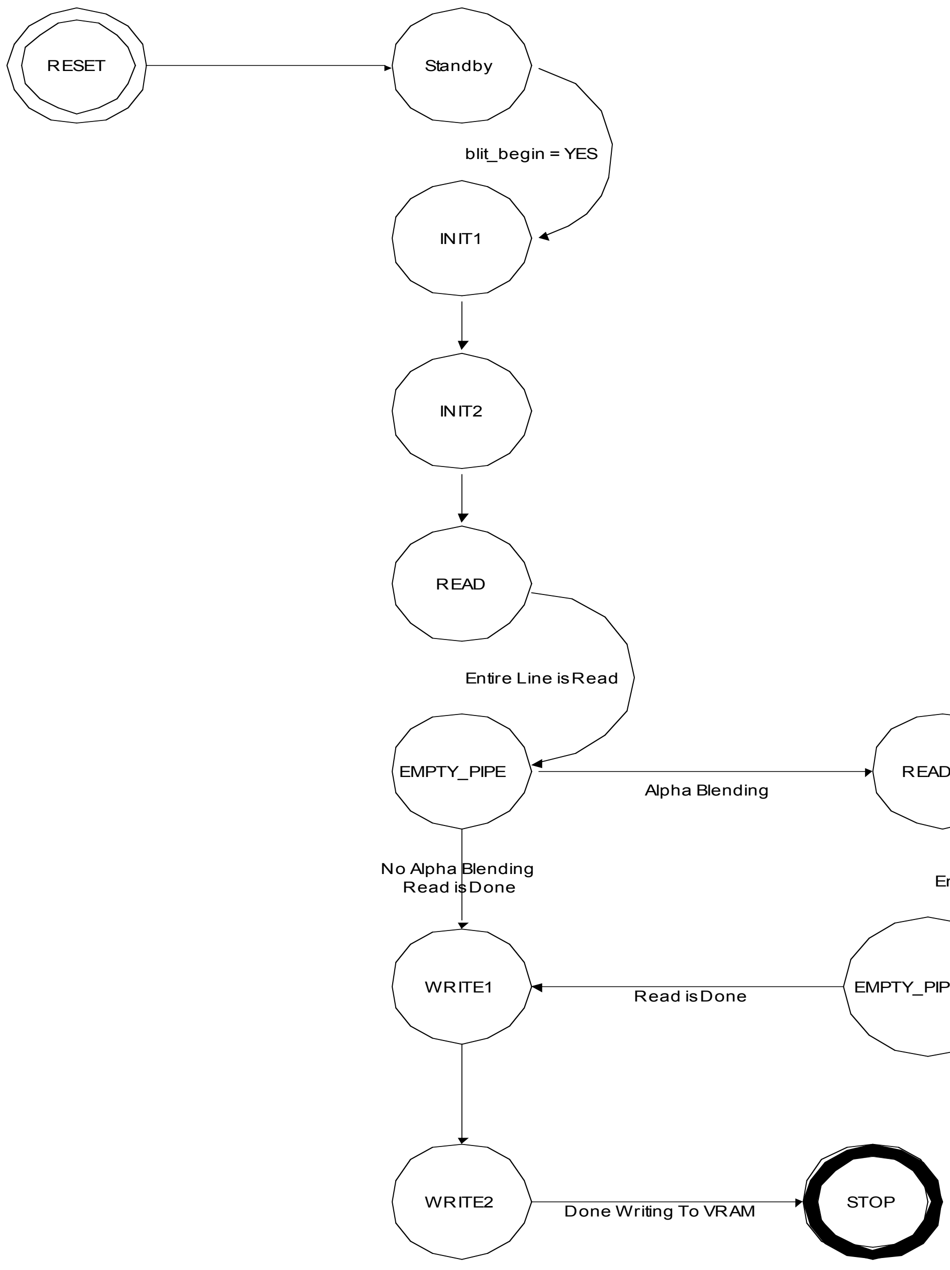

Figure 3.1 Flow Diagram for Blitter State Machine

3.1.4 VGA Interface

The VGA interface was heavily based off of XESS sample code. It was modified to output to a 320x240 screen. This given code writes to a CRT monitor the contents of VRAM whose address is dictated by the GPU. The VGA was modified to only write every other screen in order to access the SDRAM for other purposes. This created a problem in that the last pixel on the screen was written to the entire unused screen. This would make the screen seem tinted. The problem was corrected by always making this last pixel black. This had the effect of placing a blank screen in between frames which is not noticeable to the naked eye.

3.2 CPU (GPU Processor)

The CPU uses DMA architecture to communicate with the GPU registers. The instruction set is described in Table 3.1. Because the PIC18F4620 has an 8-bit architecture, the GPU addresses must be sent in three separate cycles.

This processor dictates what and where to draw. It first tells the GPU to draw the background. It does this by giving the target address, source address, source lines, source width, and no alpha blending. The CPU draws sprites in the same way except that alpha blending is turned on.

The CPU also keeps track of and controls two screen buffers. By alternating these buffers, one buffer can be written while the other can be drawn to the screen.

By keeping track of the target address of each sprite, the CPU can take input from the controller and move that address. The controller input is read asynchronously by the USART receiver. This module generates an interrupt where the information is read and appropriate actions are taken.

3.3 Audio Processor (SPU)

The SPU receives information from the CPU as to which song it needs to play. This processor controls the MP3 player by sending signals that "press" the buttons, i.e. play, stop, forward, and reverse. The audio processor keeps track of which song it is on and from that information can calculate how many forward or reverse button presses it needs to perform in order to get to the required track.

In addition, the MP3 player needs less than 5V, i.e. 3.3V, making it dangerous for our audio processor to send it 5V signals. The FPGA can generate a 3.3V output. In order to provide signals at this voltage, the audio processor is powered by 3.3V instead of 5V. By powering the PIC with this voltage, it is able to turn on and off the MP3 player by controlling the power to the device.

3.4 Input Processor

The Input processor is compatible with all NES controllers. It was discovered that the 50MHz clock made it impossible for the USART module on the PIC to serially communicate with the NES controller. The PIC16F628A was used as a buffer between the controller and the CPU.

The NES controller is essentially a shift register with a load capability. The input processor polls the NES controller once every 36ms. It does this by asserting the strobe signal for 1ms. This loads the NES shift register with the button data. The input processor then asserts 8 clock pulses to read each button sequentially. The processor then inverts the 8-bit controller data for transmission to the CPU. Data from the controller is asynchronously serially transmitted to the CPU via the on-board USART module at 9600 baud.

4. FUTURE DEVELOPMENT

4.1 16-32 bit processing

Possible future processing will be done on an ARM or Power PC. This would enable 3D graphics.

4.2 FPGA

The FPGA would be made more stable. This means that when the device is powered up, the device would be automatically programmed, and the graphics will be loaded onto the on-board RAM.

APPENDIX A

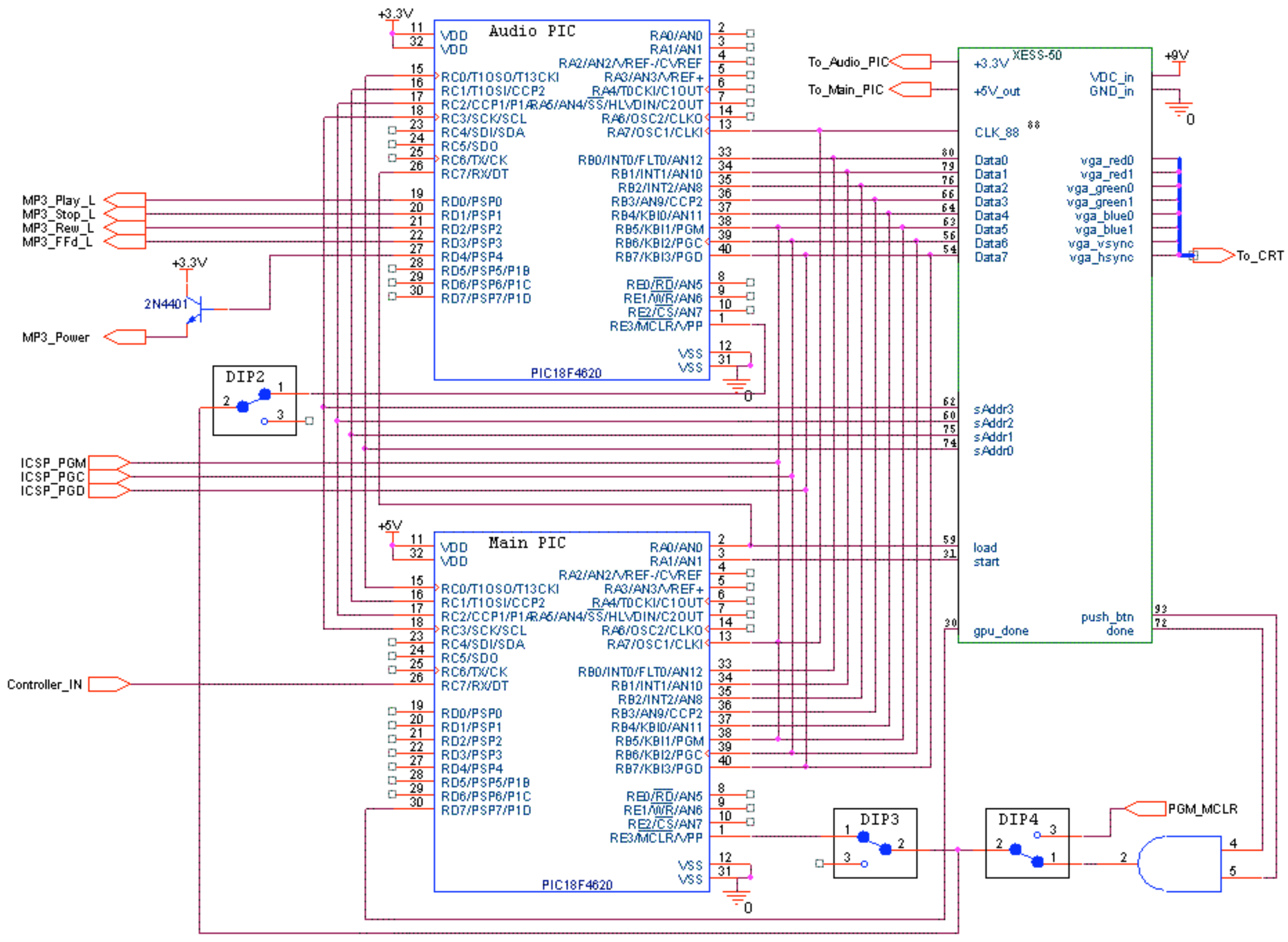

Figure A.1 Circuit Schematic

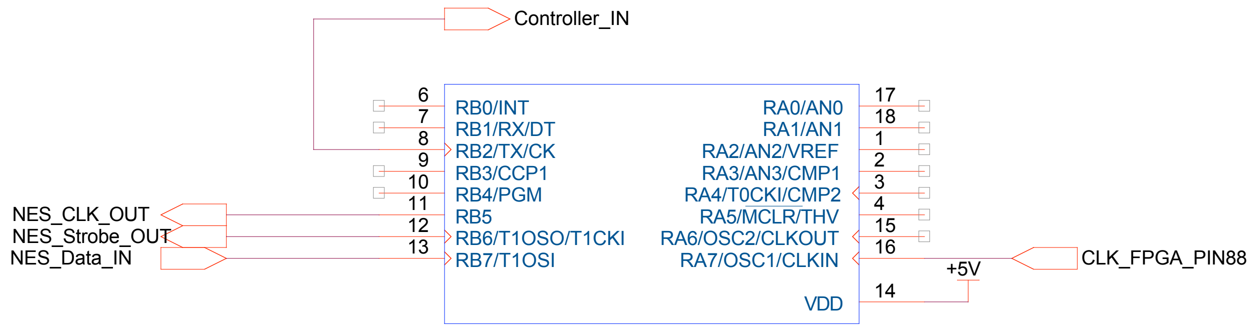

Figure A.2 Controller input Schematic