Sega Game Gear on a Chip

Design Report

Max Thrun — Samir Silbak

Fall 2012

Contents

1 Abstract

The objective of this project is to reimplement the hardware of the Sega Game Gear (GG) in a Field Programmable Gate Array (FPGA). To do this each chip in the system will be reimplemented in Verilog based on official and 3rd party descriptions and specifications on how they operate. This document details the design functionality and testing metrics that will be followed during the implementation and execution of our project.

2 System Overview

The Sega Game Gear, released in 1991, was Sega's first attempt at a handheld gaming console. It featured a Zilog Z80 clocked at 3.58MHz, 8KB of system RAM, 16KB of video ram, and a 160x144 pixel resolution screen.

Figure 1: Sega Game Gear [1]

The image below shows a picture of internal PCB of the Game Gear and it's major components

gamegear_pcb.png

Figure 2: Sega Game Gear PCB

1	Zilog Z80 CPU	4	16K VRAM
2	Video Display Processor	5	Sega BIOS ROM
3	8K RAM	6	Controller IO

2.1 Functional Diagrams

At a high level the Game Gear can be viewed simply as a system which takes input from a game controller and produces video and audio has an output. This description is shown in Figure 3

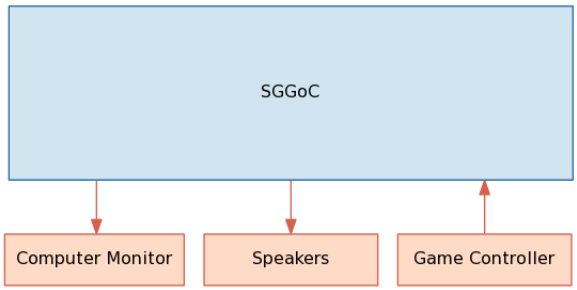

Figure 3: Black Box Diagram

Internally the functionality can be easily modularized based on the actual physical components. Additional components will be needed such as an IO controller and a memory management unit (MMU) to account for the lack of tri-state buses in our design (explained later). Figure 4 shows the major functional components in our design and their interaction with each other.

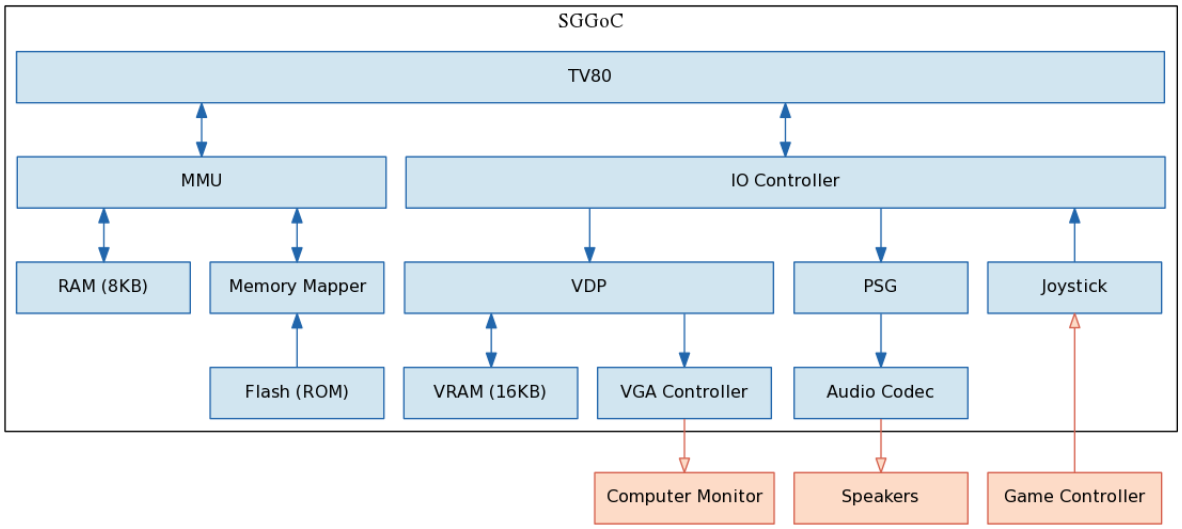

Figure 4: Internal Functional Diagram

3 Cartridges and Memory Mapping

The Z80 CPU only has 64KB of address space and only 48KB of it is dedicated to the game cartridge. A "mapper" is used to allow the use of larger ROMs (as well as on-catridge RAM). Figure 5 shows a standard GG cartridge and Figure 6 shows the internal PCB. The top chip is the Memory Mapper and the bottom chip is the actual game ROM.

Figure 5: GG Cartridge[2]

Figure 6: Cartridge PCB [3]

3.1 Sega Mapper

There are several different mappers in existence but we will only focus on implementing the "Sega Mapper" as it is the most popular. The Sega mapper defines 3 16KB slots in the Z80 memory map. Any 16KB bank of ROM can be mapped into any of these 3 slots. The mapping is controlled by memory addresses 0xFFFC-0xFFFF.

Control Register	Slot	Maps to Z80 Address
0xFFFC	Control Bits	-
0xFFFD	0	0x0000-0x3FFF
0xFFFE	1	0x4000-0x7FFF
0xFFFF	2	0x8000-0xBFFF

Table 1: Mapper control registers [4]

The ROM is viewed as an array of 16KB banks. The value written into the control register determines which bank is selected for that slot. Figure 7 shows an example of how the address space gets mapped to different ROM banks.

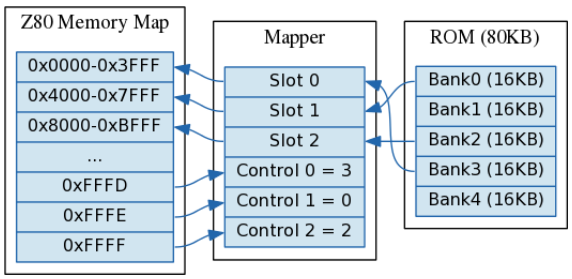

Figure 7: Mapping Diagram

3.2 Cartridge Alternatives

Since we are targeting a standard FPGA development board we cannot easily use the actual game cartridge without some kind of hardware adaptor. Additionally, using the actual game cartridge would go against what we are trying to accomplish with this project which is eliminating the need for any of the original hardware. Lucky, virtually every Game Gear cartridge has been dumped and are available online. Unfortunately, the ROMs range in size all the way up to 4MB and being that most affordable FPGAs do not have that much block RAM it would be impossible for us to pack the ROM with the bitstream. A more ideal solution would be to use a SD card which can be loaded with numerous ROM files and then write a bootloader to select which one to play. The complexity of that strategy, however, is not worth the initial effort. A simpler solution which would allow us to quickly start on the rest of the project is to load a single ROM file onto the Flash chip that comes with our DE-1 development board.

3.3 ROM Flasher

The technical requirements of interfacing with the flash chip on our board are outside the scope of our project and as such we will be using a core provided by Altera [5] to simplify the process. Figure 8 shows the black box diagram of the flash core provided by Altera.

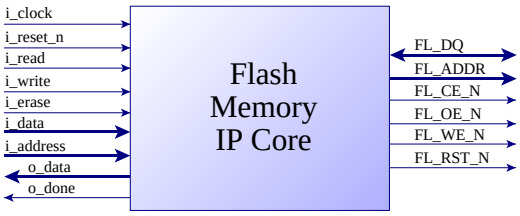

Figure 8: Altera Flash Core

A "ROM Flasher" project, completely separate from the GG project, will be used to initially load a single ROM file into the flash chip. On the computer a Python program will read a ROM file byte by byte and send it to the FPGA over the serial port. The FPGA will send back the byte it just recieved as an acknowledgement. Another Python program will be used to read back and verify that the flash contents match the ROM file. Figures 9 and 10 illustrate the process of writing and reading a ROM to and from the flash memory.

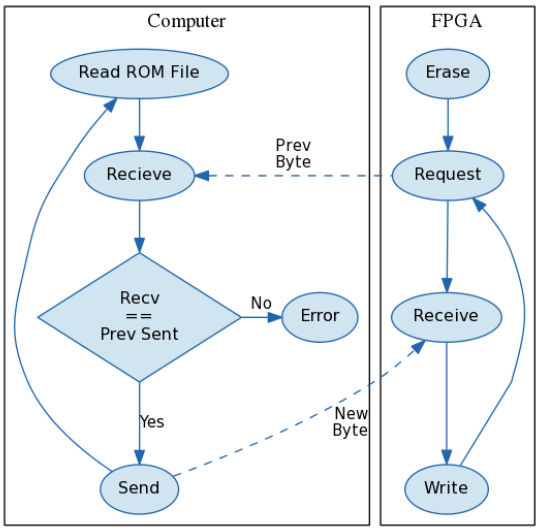

Figure 9: ROM Flasher - Write

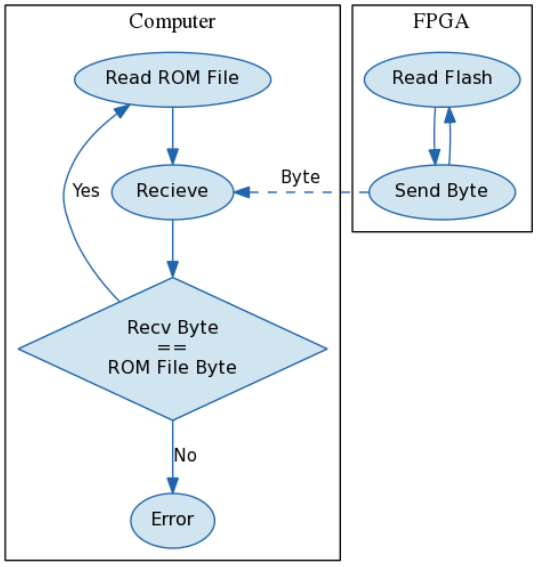

Figure 10: ROM Flasher - Read

4 Zilog Z80

The Game Gear uses the classic Zilog Z80 CPU running at 3.58MHz as it's main processor. The re-implementation of the Z80 is completely outside the scope of this project and as such we will be using the popular TV80 [6] CPU which is a proven, open source, implementation of the Z80 written in Verilog. The interface of the TV80 exactly matches that of the original Z80, shown in Figure 11, with the only exception being the data bus. The original Z80, as well as the whole Game Gear memory system, relies on tri-state buses where as the TV80 has a separate bus for data in and data out.

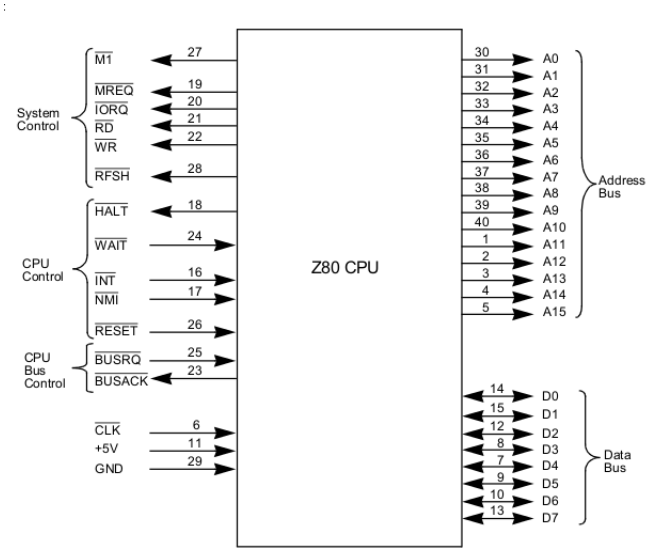

Figure 11: Zilog Z80

5 Memory Management Unit (MMU)

Since our design does not use tri-state buses we cannot simply connect the data lines from each component together. Instead, we need to have some kind of memory manager that can multiplex the data lines to whichever device the address is pointing to according the system memory map. The full memory map for the Game Gear is shown below

Address	Device
0x0000-0x03FF	ROM (unpaged)
0x0400-0x3FFF	ROM mapper slot 0
0x4000-0x7FFF	ROM mapper slot 1
0x8000-0xBFFF	ROM mapper slot 2 - OR - SaveRAM
0xC000-0xDFFF	System RAM
0xE000-0xFFFF	System RAM (mirror)
0xFFFC	SaveRAM mapper control
0xFFFD	Mapper slot 0 control
0xFFFE	Mapper slot 1 control
0xFFFF	Mapper slot 2 control

Table 2: Z80 Memory Map [7]

Since the cartridge memory mapper handles the slot mapping itself we are really only multiplexing between the cartridge and the system RAM. The system RAM happens to start at a nice edge being that 0xC000 = 0b1100000000000000. To see if we are accessing RAM we only need to check to see if bits 15 and 16 are high. If so we can simply use the 13 LSBs of the Z80 address as the index into the RAM. This method also accounts for the RAM mirror since the 13 LSBs repeat starting at 0xE000.

6 IO Controller

7 Video Display Processor (VDP)

7.1 Ports

7.2 Tiles

7.3 CRAM

7.4 Layers

7.5 Interrupts

7.6 Display Timing

8 Programmable Sound Generator (PSG)

9 VGA Controller

References

[1] http://mo5.com/musee-machines-gamegear.html

[2] http://www.magisterrex.com/prodimages/EccoDolphinGameGear-h450.png

[3] http://www.smspower.org/Development/SMSPagingChips

[4] http://www.smspower.org/Development/Mappers

[5] ftp://ftp.altera.com/up/pub/flash/altera_up_flash_memory.zip

[6] http://opencores.org/project,tv80

[7] http://code.google.com/p/bizhawk/source/browse/trunk/BizHawk.Emulation/Consoles/Sega/SMS/MemoryMap.Sega.cs