Software Sega CD Simulator

by

Jeff Quinn

Submitted to:

Dr. Donald Schertz

EE 452 Senior Project

May 9, 2001

Table of Contents

Abstract:

The project is a piece of software named “AGES.” It is the development of a simulator of a Sega CD, a peripheral add-on to a Sega Genesis that includes a CD-drive, a M68000, a custom VDP (video display processor) chip, a PCM (pulse code modulation) sound chip, and additional RAM. It will be added to a set of already existing code that simulates the Sega Genesis and another add-on, the Sega 32X.

Objective:

AGES is a learning project. Simulation of the Sega CD was chosen for the challenge it presents. Despite four years of effort by numerous individuals, no Sega CD simulator has been successfully written. This is primarily due to the lack of detailed information about the CD unit. Furthermore, the system, even without the CD unit, is extraordinarily complex. It has elements from older low-level-based and modern high-level-based hardware design styles. This project is sufficiently difficult though not intractable, as evidenced by the fact that AGES was the first software to successfully simulate the Sega 32X. This project is an indispensable learning experience in the areas of reverse engineering, hardware architecture, programming, and good methodology.

AGES was written to run on a Windows machine. The interface with Windows has been written in MFC/C++ but amounts to very little of the actual code. The majority of AGES was written in optimized Pentium-MMX assembly. The design goal of AGES is accuracy with optimum speed. This means that trade-offs between speed and memory usage have favored speed. Trade-offs between executable size and start-up time have favored executable size. Trade-offs between accuracy of emulation and speed have favored accuracy.

The specifications set at the beginning of the project were as follows. The software must be able to run 95% of Sega CD software perfectly, and the software must be able to run full-speed on a Pentium II 450 Mhz, which is representative of today's low-end PC market.

Inputs and Outputs of the project:

The general inputs and outputs for AGES are listed below. Refer to Fig. 1 for a graphical representation.

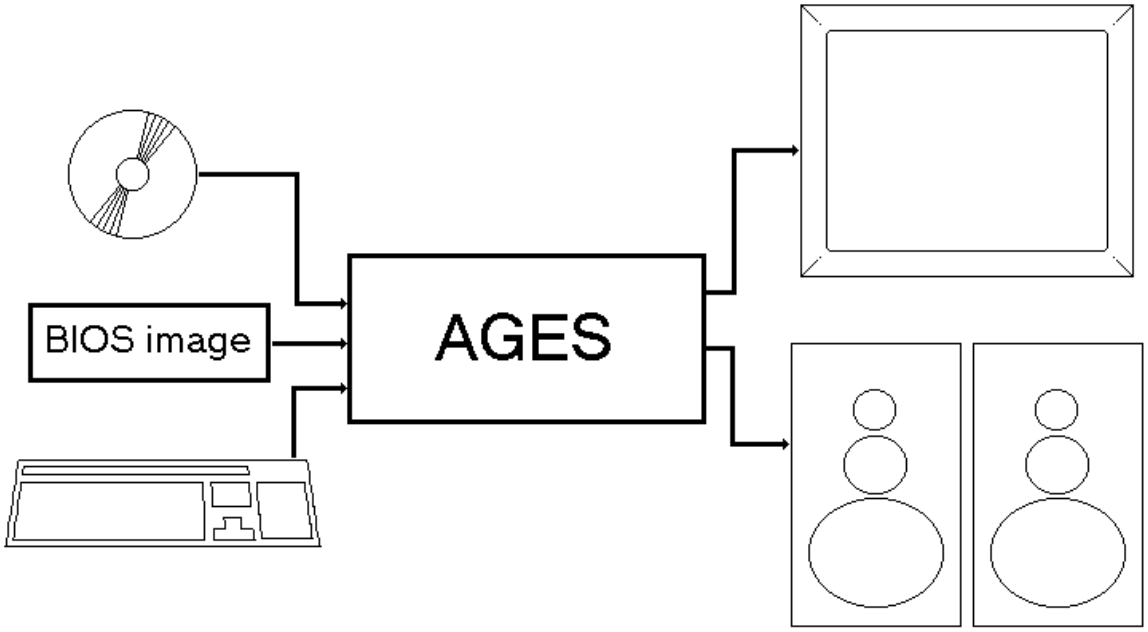
Inputs:

- Game control inputs
- Actual game CD
- One binary image of the Sega CD BIOS software

Outputs:

- Game visuals
- Game sounds/music

Fig. 1: AGES inputs and outputs.

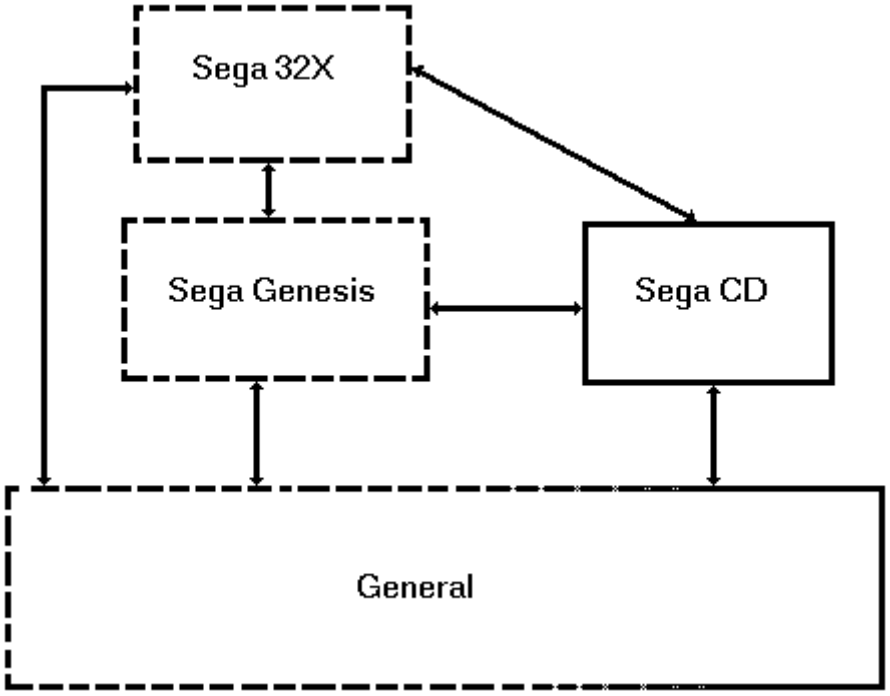


The user can place a Sega CD game disc into the CD-drive of his/her computer, after retrieving a binary image of the Sega CD BIOS software, and interact with the game software in the same way that he/she would by using a real Sega CD machine. This means that the picture is displayed in its entirety on the computer monitor (instead of television), the sound is reproduced in its entirety via the computer's sound card and speakers (instead of television speakers), and the input is received via the keyboard or an attached gaming device (joystick) at the user's discretion instead of the gaming device (joystick) native to the Sega CD machine. Aside from these slight differences, the behavior of the software attempts to duplicate the Sega CD's behavior exactly. One notable exception is the speed of the drive. A real Sega CD's drive was "single speed," however, the simulation software, where possible, utilizes the speed of the PC CD-drive to provide faster performance (E.G. shorter "load times.")

Software Modules of the project:

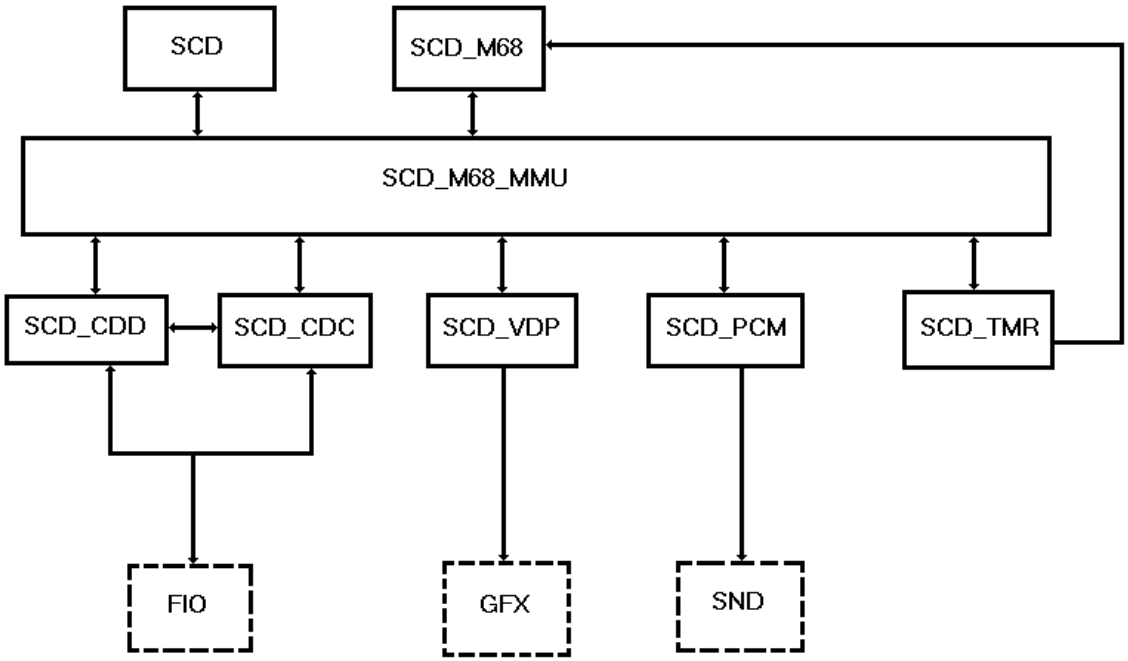
AGES is divided into four different blocks, the Sega Genesis Block, the Sega 32X Block, the Sega CD Block, and the General Block. The first two blocks were written prior to the beginning of the project and are beyond its scope. The General Block was also previously written, but not beyond the project's scope as it required additions to be made to it to accommodate new functionality needed by the Sega CD Block. The primary focus of the project was the Sega CD Block itself. Refer to Fig. 2 for an illustration of the blocks. The dashes indicate what was previously completed and beyond the scope of the project.

Fig. 2: AGES Block Diagram.



AGES is further broken down into two types of modules, “chip modules” and “engine modules.” Each chip module is a subset of the simulation containing all code pertaining to a single chip and a small amount of code that handles the interaction of said chip with other chips. Engine modules handle interaction with a specific part of the PC hardware. The first three blocks contain only chip modules. The General Block contains only engine modules. AGES already has fully written code for all chips located in the Sega Genesis and Sega 32X and, as previously stated, these are beyond the scope of the project. AGES also has a complete set of fully written engine modules. This project is concerned only with the chip modules of the Sega CD and adding code to the engine modules as needed. No new engine modules were created. Refer to Fig. 3 for an illustration of all the modules pertaining to the project and how they relate to each other.

Fig. 3: AGES modules in project scope.



Sega CD chip modules:

M68_EMU	This module simulates the Motorola 68000 CPU chip. It is not strictly a Sega CD module, but rather is a general module that is instantiated twice, once as GEN_M68, and once as SCD_M68. The SCD_M68 has three inputs and three outputs for accessing memory. The inputs are data reads in by byte, word, and dword sizes. The outputs are data writes in byte, word, and dword sizes. Additionally, it has one input for IRQ level. This module has already been written and debugged.
SCD	This simulates the system registers of the Sega CD, and contains basic driver information specific to the Sega CD. The inputs and outputs of this module are the data stored in the registers. This module is completed and debugged.
SCD_M68_MMU	This simulates the memory map of the M68000 located in the Sega CD. This module takes inputs and outputs from each other module in the Sega CD block. This module is completed and debugged.
SCD_CDC	This simulates the behavior CD controller in the Sega CD. The controller is used for reading data from CD data tracks. This module inputs and outputs the data in the sixteen 8-bit CDC registers. Additionally, this module has an input from the

SCD_CDD. The input is a periodic "read" signal asserted when the CDD is playing a data track to be decoded and error corrected. This module is implemented and as far as is known, needs no further work, though, it is always possible that in the future, a non-working piece of software, which uses some previously unknown unimplemented feature or quirk of this piece of hardware, may be encountered.

SCD_CDD — This simulates the behavior of the physical CD drive in the Sega CD. The drive is used for activating the spindle motor, reading TOC info, playing tracks, etc. Its inputs are data in the ten 4-bit Transmit Command Registers. Its outputs are data in the ten 4-bit Receive Status Registers and a periodic "read" signal (sent to the SCD_CDC when a data track is being played). This module is implemented and as far as is known, it needs no further work, though, it is always possible that in the future, one may encounter a non-working piece of software which uses some previously unknown unimplemented feature or quirk of this piece of hardware.

SCD_PCM — This simulates the behavior of the PCM (sound) chip located in the Sega CD. Its inputs and outputs are the data in the PCM registers. This module is implemented, but not wholly debugged as the SND engine additions are not completed (as of April 20, 2001).

SCD_VDP — This simulates the behavior of the VDP (graphics) chip located in the Sega CD. Its inputs and outputs are the data in the VDP registers. This module is completed and debugged with the exception of the "fullscreen" graphics feature, as a piece of software that actually uses it has not yet found.

SCD_TMR — This simulates all the timer chips located in the Sega CD. Its inputs and outputs are the data in the timer registers and an IRQ level. This module is completed and debugged.

Engine modules:

GFX — This handles all interfacing with the host PC's graphics hardware. This module inputs raw[1] graphics data from all VDP simulation modules and converts it to a form useable by DirectX for outputting to the monitor. The GFX module required additional code to support the needs of the Sega CD simulation modules. This work has been completed and debugged.

[1] In whatever format is native to the chip delivering the data.

SND — This handles all interfacing with the host PC's sound hardware. This module inputs raw[2] sound data from all sound simulation modules and converts it to a form useable by DirectX for outputting to the soundcard and speakers. The SND module required additional code to support the needs of the Sega CD simulation modules. This work is not yet completed.

FIO — This handles all interfacing with the host PC's file/IO hardware (E.G. image files off the harddrive or direct use of the CD drive). It inputs raw[3] file/IO data and converts it to a program-wide consistent binary format for outputting to the various modules. The FIO module required additional code to support the needs of the Sega CD simulation modules. This work is completed and debugged.

Conclusion:

As of the time of this report, the project is not completed nor were all goals/specifications reached. The reproduction of the Sega CD PCM sound has not been implemented and the "fullscreen" graphics have not been implemented. The program is not able to run full-speed on a Pentium II 450 Mhz, though with a feature called "frame-skipping", it can give the *appearance* of operating at full-speed, with a negligible amount of user-perceivable choppiness. The software is also not capable of meeting the 95% compatibility figure. At present estimates, it can run about 70% of software perfectly, and an additional 15% of software with reasonable accuracy, though glitches are present.

Naturally, the project will continue development even beyond the end of the semester. Some possible additions are improved performance for the CD-ROM drive code, improved efficiency and accuracy, and various presentational improvements over the real machine such as bi-linear filtering for the graphics and stereo output for the sound.

Though the project may seem to be frivolous at first impression, it most certainly is not. Writing a simulator has provided a unique understanding of microprocessors and machine architecture. The project was an excellent exercise in gaining valuable reverse engineering experience. Finally, the project has provided the opportunity to develop creative ways to optimize the performance of the simulator. These skills were given the great deal of attention they so rarely receive and will be extremely valuable in future endeavors.

[2] In whatever format is native to the chip delivering the data.
[3] In whatever format is native to the chip delivering the data.