Sega Game Gear on a Chip

Max Thrun | Samir Silbak

University of Cincinnati

Fall 2012

Agenda

- Problem Description
- Design Process
- Requirements/Assessment Metrics/Test Plan
- Design Overview
- Project Limitations
- Demonstration

Problem Description

Reimplement all the digital components of a legacy computer system in a FPGA

Problem Description

Why?

- **Maintainability** - You can no longer buy parts to service legacy computer systems
- **Upgradability** - Reimplementation gives an opportunity to add additional features
- **Portability** - Do not need all the original big clunky hardware. Reimplementation can be embedded in new designs

Problem Description

Why?

- **Maintainability** - You can no longer buy parts to service legacy computer systems
- **Upgradability** - Reimplementation gives an opportunity to add additional features
- **Portability** - Do not need all the original big clunky hardware. Reimplementation can be embedded in new designs

Problem Description

Why?

- **Maintainability** - You can no longer buy parts to service legacy computer systems
- **Upgradability** - Reimplementation gives an opportunity to add additional features
- **Portability** - Do not need all the original big clunky hardware. Reimplementation can be embedded in new designs

Problem Description

Why?

- **Maintainability** - You can no longer buy parts to service legacy computer systems
- **Upgradability** - Reimplementation gives an opportunity to add additional features
- **Portability** - Do not need all the original big clunky hardware. Reimplementation can be embedded in new designs

Design Process

1. Break down system components according to the original system architecture
2. Implement each component in Verilog matching the original functionality described by official and non official documents
3. Simulate each components functionality and compare it against the actual hardware (in our case an emulator)
4. Tie components together in a way that is better suited toward FPGA technology (*E.g.* avoid tri-state buses)

Design Process

1. Break down system components according to the original system architecture
2. Implement each component in Verilog matching the original functionality described by official and non official documents
3. Simulate each components functionality and compare it against the actual hardware (in our case an emulator)
4. Tie components together in a way that is better suited toward FPGA technology (*E.g.* avoid tri-state buses)

Design Process

1. Break down system components according to the original system architecture
2. Implement each component in Verilog matching the original functionality described by official and non official documents
3. Simulate each components functionality and compare it against the actual hardware (in our case an emulator)
4. Tie components together in a way that is better suited toward FPGA technology (*E.g.* avoid tri-state buses)

Design Process

1. Break down system components according to the original system architecture
2. Implement each component in Verilog matching the original functionality described by official and non official documents
3. Simulate each components functionality and compare it against the actual hardware (in our case an emulator)
4. Tie components together in a way that is better suited toward FPGA technology (*E.g.* avoid tri-state buses)

Requirements

1. VGA Output
2. Accurate gameplay
3. asdf

Requirements

1. VGA Output
2. Accurate gameplay
3. asdf

Requirements

1. VGA Output
2. Accurate gameplay
3. asdf

Assesment Metrics

Function	Requirement Specification	Design Verified	Device Validated
VGA Output	Design must output video at 640x480 to a VGA monitor	Yes	Yes
Sega Mapper	Design must implement the Sega Memory Mapper	Yes	Yes
Video Display Processor	Design must implement the TMS9918	Partly	Partly
Game ROM stored in flash	Design must be able to load game ROMs from flash	Yes	Yes
Controller Input	Design must implement a single controller	No	No
Game Gear system functionality	Design must implement the same functionality as the original Game Gear	Partly	Partly

Requirements/Ass. Metrics/Test Plan

In order to test the operation of the z80, and basic functionality of the VDP, custom ROMs were developed. Using the Small Device C Compiler (SDCC) [2] we are able to write programs that exercises various functionality of the system.
We found SDCC extremely easy to setup and get working. The only thing we needed to modify was the stack pointer location in the C Runtime file (crt0.s). The default stack pointer location is set to `0xFFFF` but the top most RAM address on the Game Gear is `0xDFFF`. Setting up IO is as easy as specifying a special function register at a given IO port. An example showing how to write data to the VDP data port (0xBE) is shown below.

Requirements/Ass. Metrics/Test Plan

With this library in place it is easy to to perform operations such as setting up the color palette:

Requirements/Ass. Metrics/Test Plan

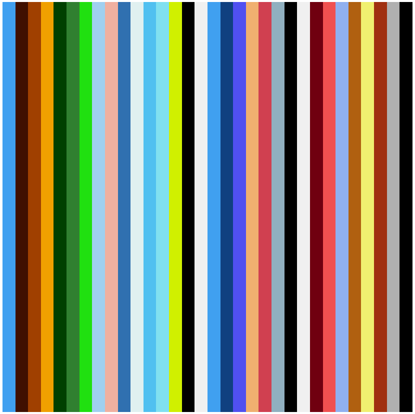


Figure : Color Palette

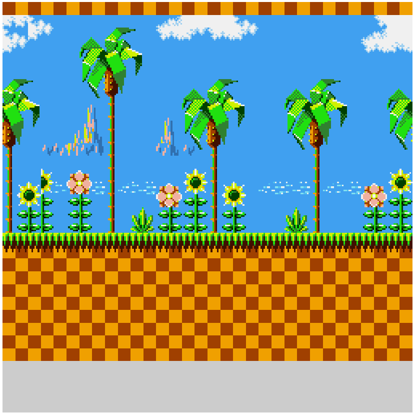


Figure : Complete Screen Render

Requirements/Ass. Metrics/Test Plan

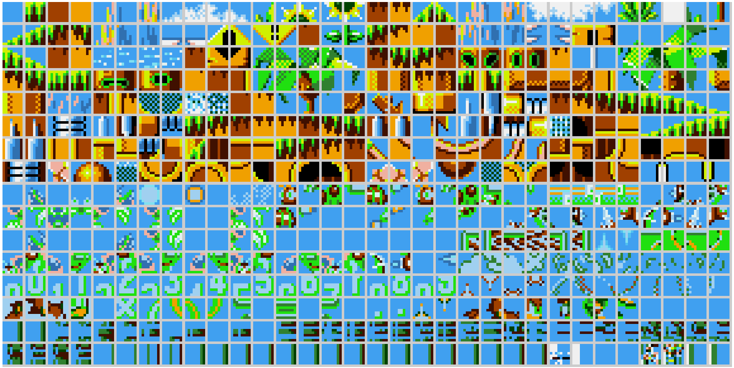
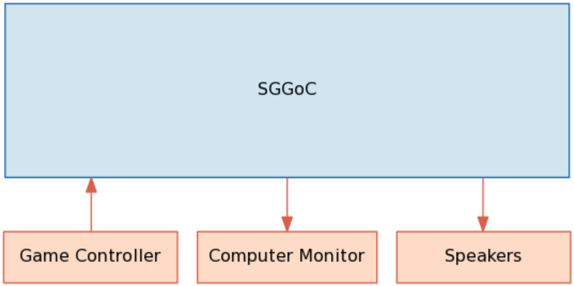


Figure : All 512 Tiles

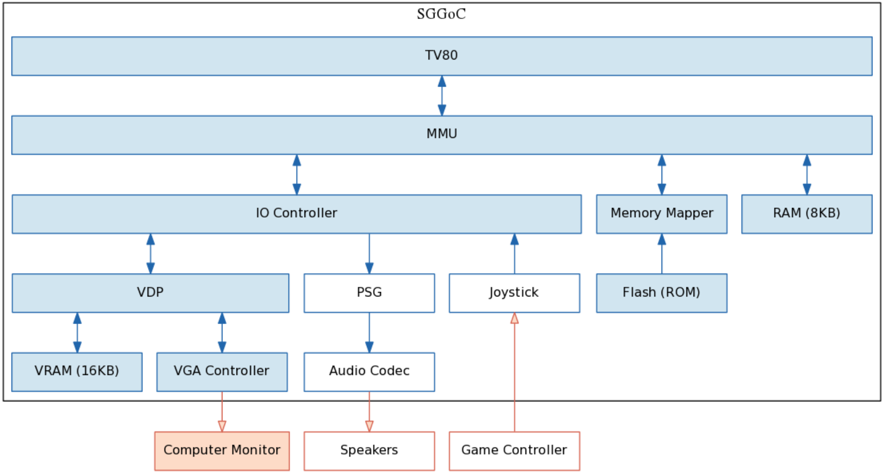
Design Overview

Black Box Diagram

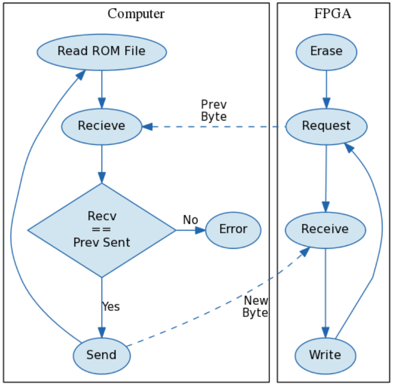


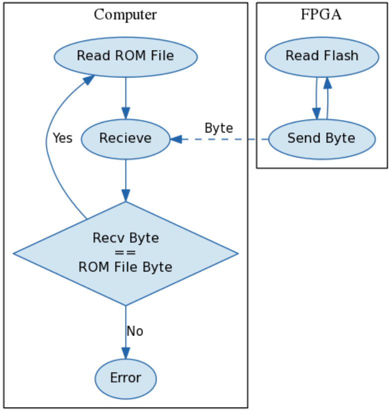
Design Overview

Internal Functional Diagram



Design Overview





Project Limitations

Game Gear is hard to test/verify

- Its only output is video (no UART, JTAG, etc..)
- Most documentation is 3rd party

Our strategy:

1. Use an emulator to watch memory fetches and get memory dumps
2. Initialize our RAMs with these dumps and verify we achieve the same visual output
3. Watch instruction fetches with a logic analyzer and see if they match the emulator

Could replace emulator with a logic/bus analyzer connected to the original hardware

Project Limitations

Game Gear is hard to test/verify

- Its only output is video (no UART, JTAG, etc..)
- Most documentation is 3rd party

Our strategy:

1. Use an emulator to watch memory fetches and get memory dumps
2. Initialize our RAMs with these dumps and verify we achieve the same visual output
3. Watch instruction fetches with a logic analyzer and see if they match the emulator

Could replace emulator with a logic/bus analyzer connected to the original hardware

Project Limitations

Game Gear is hard to test/verify

- Its only output is video (no UART, JTAG, etc..)
- Most documentation is 3rd party

Our strategy:

1. Use an emulator to watch memory fetches and get memory dumps
2. Initialize our RAMs with these dumps and verify we achieve the same visual output
3. Watch instruction fetches with a logic analyzer and see if they match the emulator

Could replace emulator with a logic/bus analyzer connected to the original hardware

Project Limitations

Game Gear is hard to test/verify

- Its only output is video (no UART, JTAG, etc..)
- Most documentation is 3rd party

Our strategy:

1. Use an emulator to watch memory fetches and get memory dumps
2. Initialize our RAMs with these dumps and verify we achieve the same visual output
3. Watch instruction fetches with a logic analyzer and see if they match the emulator

Could replace emulator with a logic/bus analyzer connected to the original hardware

Project Limitations

Game Gear is hard to test/verify

- Its only output is video (no UART, JTAG, etc..)
- Most documentation is 3rd party

Our strategy:

1. Use an emulator to watch memory fetches and get memory dumps
2. Initialize our RAMs with these dumps and verify we achieve the same visual output
3. Watch instruction fetches with a logic analyzer and see if they match the emulator

Could replace emulator with a logic/bus analyzer connected to the original hardware

Project Limitations

Game Gear is hard to test/verify

- Its only output is video (no UART, JTAG, etc..)
- Most documentation is 3rd party

Our strategy:

1. Use an emulator to watch memory fetches and get memory dumps
2. Initialize our RAMs with these dumps and verify we achieve the same visual output
3. Watch instruction fetches with a logic analyzer and see if they match the emulator

Could replace emulator with a logic/bus analyzer connected to the original hardware

Project Limitations

Game Gear is hard to test/verify

- Its only output is video (no UART, JTAG, etc..)
- Most documentation is 3rd party

Our strategy:

1. Use an emulator to watch memory fetches and get memory dumps
2. Initialize our RAMs with these dumps and verify we achieve the same visual output
3. Watch instruction fetches with a logic analyzer and see if they match the emulator

Could replace emulator with a logic/bus analyzer connected to the original hardware

Project Limitations

Game Gear is hard to test/verify

- Its only output is video (no UART, JTAG, etc..)
- Most documentation is 3rd party

Our strategy:

1. Use an emulator to watch memory fetches and get memory dumps
2. Initialize our RAMs with these dumps and verify we achieve the same visual output
3. Watch instruction fetches with a logic analyzer and see if they match the emulator

Could replace emulator with a logic/bus analyzer connected to the original hardware

Demonstration

Questions?

References

- http://bcz.asterope.fr/osmose.htm
- http://sdcc.sourceforge.net/