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)

Implementation Priority

1. Flash Memory Interface
2. Memory Management Unit (MMU)
3. Z80 + System RAM
4. Cartridge Mapper
5. VGA Driver
6. Background Tiles + VRAM
7. Video Display Processor Logic (VDP)
8. Sprites
9. Controller Input
10. Sound

Requirements

1. **VGA Output** - Need to output video via VGA which is the most common video interface found on FPGA development boards.
2. **ROM Loading** - Need to be able to easily load in different game and test ROMs into the Flash memory chip on the dev board.
3. **Accurate Re-implementation** - Overall goal of this project is to re-implement the Sega Game Gear as accurately as possible. User should not be able to tell its not the original hardware.
4. **Accurate Architecture** - Overall system architecture should be as clean as possible and closely match that of the original Game Gear.

Requirements

1. **VGA Output** - Need to output video via VGA which is the most common video interface found on FPGA development boards.
2. **ROM Loading** - Need to be able to easily load in different game and test ROMs into the Flash memory chip on the dev board.
3. **Accurate Re-implementation** - Overall goal of this project is to re-implement the Sega Game Gear as accurately as possible. User should not be able to tell its not the original hardware.
4. **Accurate Architecture** - Overall system architecture should be as clean as possible and closely match that of the original Game Gear.

Requirements

1. **VGA Output** - Need to output video via VGA which is the most common video interface found on FPGA development boards.
2. **ROM Loading** - Need to be able to easily load in different game and test ROMs into the Flash memory chip on the dev board.
3. **Accurate Re-implementation** - Overall goal of this project is to re-implement the Sega Game Gear as accurately as possible. User should not be able to tell its not the original hardware.
4. **Accurate Architecture** - Overall system architecture should be as clean as possible and closely match that of the original Game Gear.

Requirements

1. **VGA Output** - Need to output video via VGA which is the most common video interface found on FPGA development boards.

2. **ROM Loading** - Need to be able to easily load in different game and test ROMs into the Flash memory chip on the dev board.

3. **Accurate Re-implementation** - Overall goal of this project is to re-implement the Sega Game Gear as accurately as possible. User should not be able to tell its not the original hardware.

4. **Accurate Architecture** - Overall system architecture should be as clean as possible and closely match that of the original Game Gear.

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

Test Plan

Test plan split up into 3 areas

- Z80, RAM, Cartridge
- VDP Background Tile Rendering
- System Simulation vs Emulator

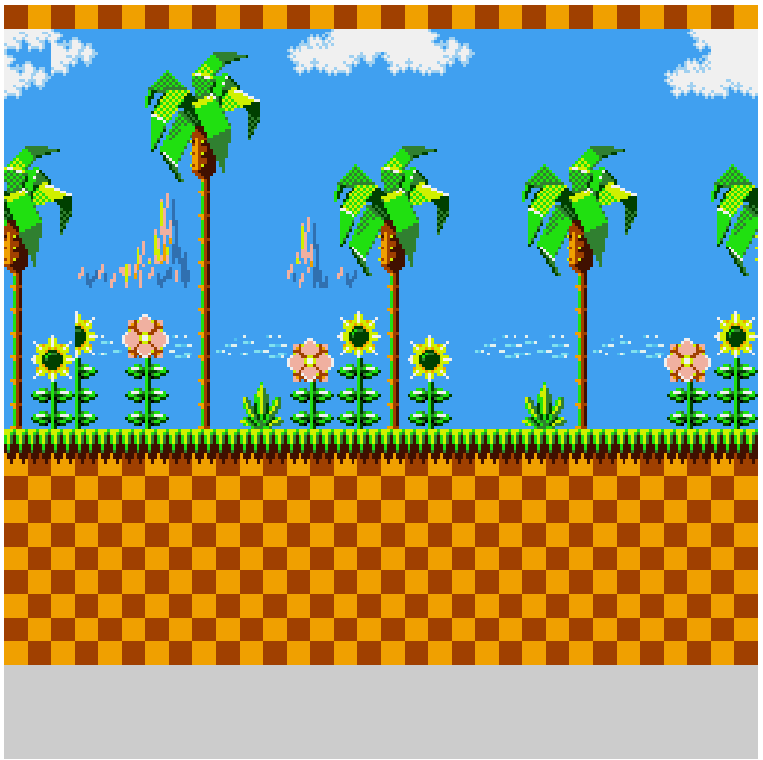
Z80, RAM, Cartridge Test

1. Connect 8 LEDs to Z80 IO port 1
2. Write simple Z80 test program that increments IO port 1
3. Load test program into flash
4. Divide Z80 clock down to 10Hz
5. Verify LEDs incrementing on development board

```
1 __sfr __at (0x01) debug;
2 int main() {
3     debug = 0x38;
4     while (1) {}
5     return 0;
6 }
```

VDP Background Tile Rendering

1. Obtain VRAM dump from the Osmose emulator
2. Verify dump integrity using rendering program on computer
3. Transfer dump to FPGA VRAM via the 'MemSend' tool
4. Verify that the resulting image on screen matches the emulator



System Simulation vs Emulator

1. Insert print statements on all memory and IO accesses in emulator
2. Do the same for the Verilog system level simulation
3. Run emulator and simulation on same ROM
4. Diff the print logs
5. Verify that the memory and IO accesses match between the emulator and simulation

System Simulation vs Emulator

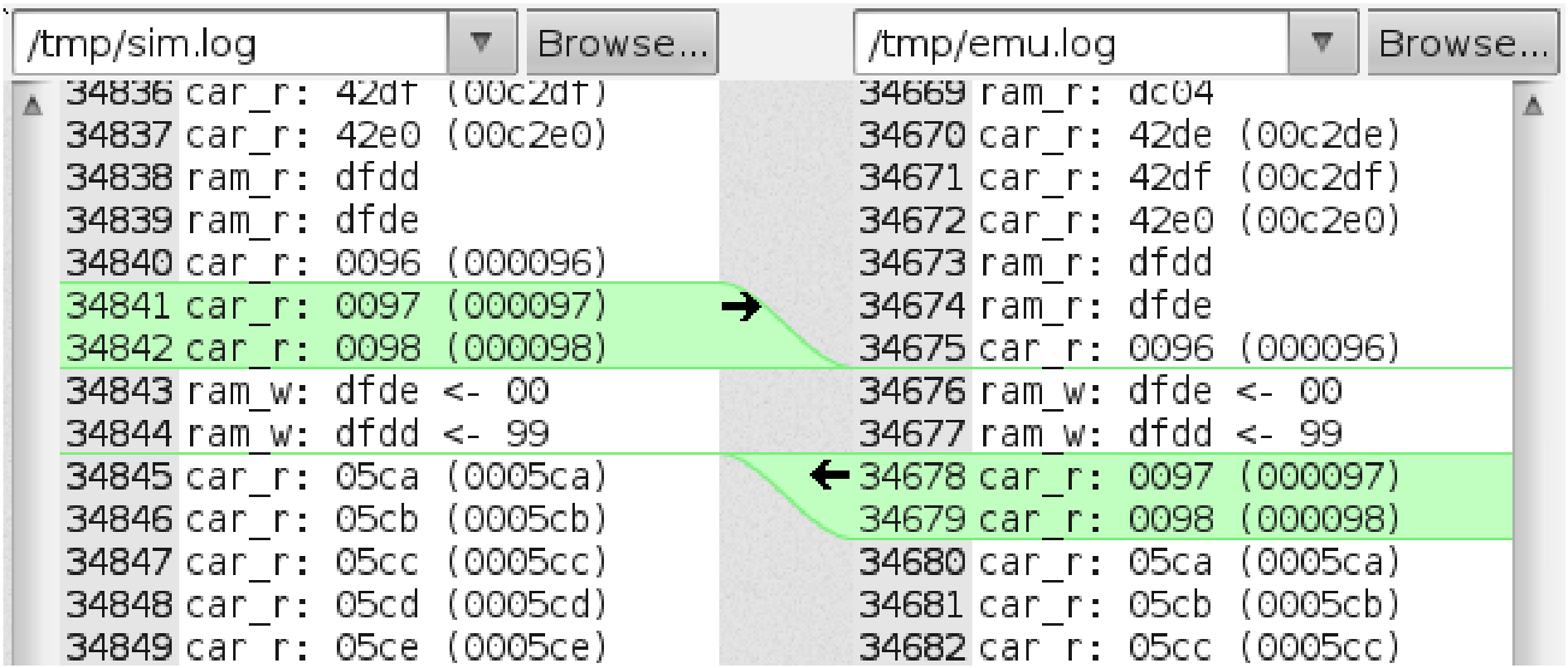
/tmp/sim.log Browse...

```
26 car_r: 0267 (000267)
27 ram_w: fffd <- 00
28 [mem] Bank 0 set to 00
29 car_r: 0268 (000268)
30 car_r: 0269 (000269)
31 car_r: 026a (00026a)
32 car_r: 026b (00026b)
33 car_r: 026c (00026c)
34 ram_w: fffe <- 01
35 [mem] Bank 1 set to 01
36 car_r: 026d (00026d)
37 car_r: 026e (00026e)
38 car_r: 026f (00026f)
39 car_r: 0270 (000270)
40 car_r: 0271 (000271)
41 ram_w: ffff <- 02
42 [mem] Bank 2 set to 02
43 car_r: 0272 (000272)
44 car_r: 0273 (000273)
45 car_r: 0274 (000274)
```

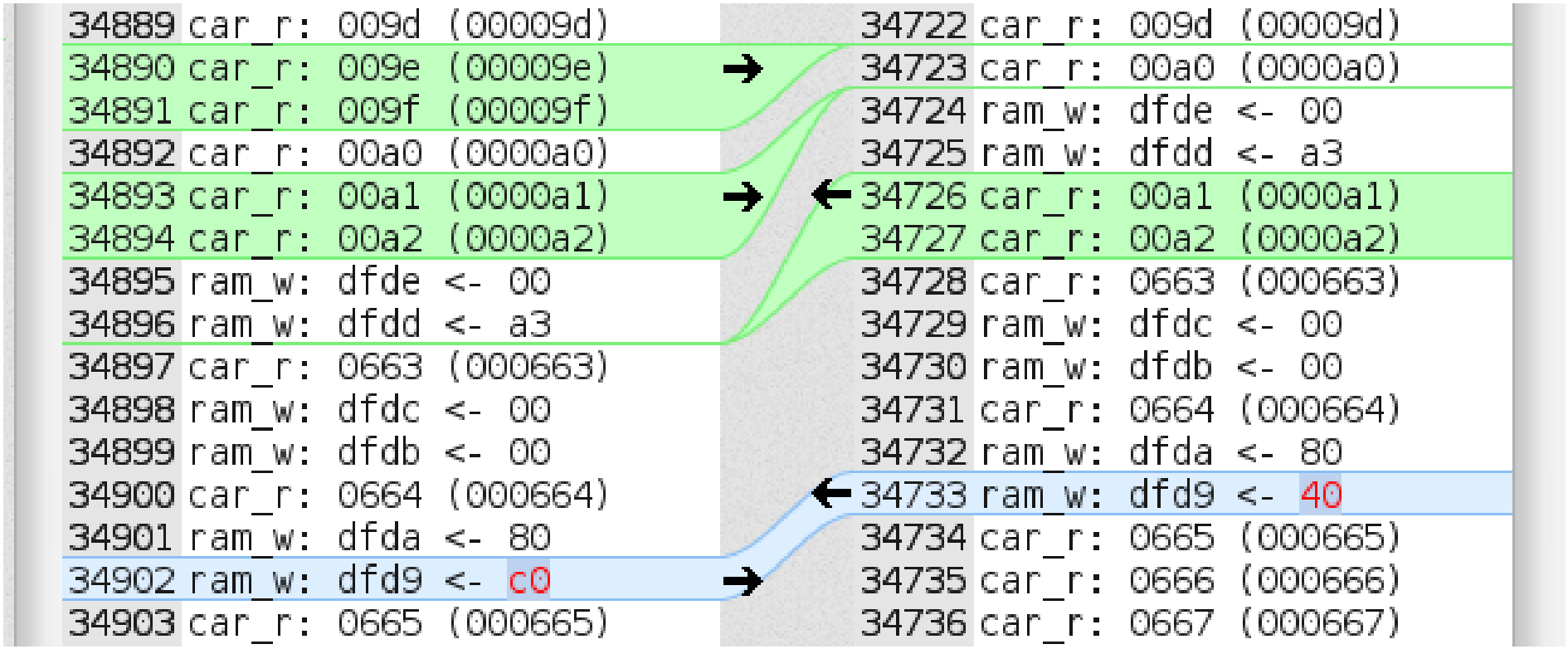
/tmp/emu.log Browse...

```
26 car_r: 0267 (000267)
27 ram_w: fffd <- 00
28 [mem] Bank 0 set to 00
29 car_r: 0268 (000268)
30 car_r: 0269 (000269)
31 car_r: 026a (00026a)
32 car_r: 026b (00026b)
33 car_r: 026c (00026c)
34 ram_w: fffe <- 01
35 [mem] Bank 1 set to 01
36 car_r: 026d (00026d)
37 car_r: 026e (00026e)
38 car_r: 026f (00026f)
39 car_r: 0270 (000270)
40 car_r: 0271 (000271)
41 ram_w: ffff <- 02
42 [mem] Bank 2 set to 02
43 car_r: 0272 (000272)
44 car_r: 0273 (000273)
45 car_r: 0274 (000274)
```

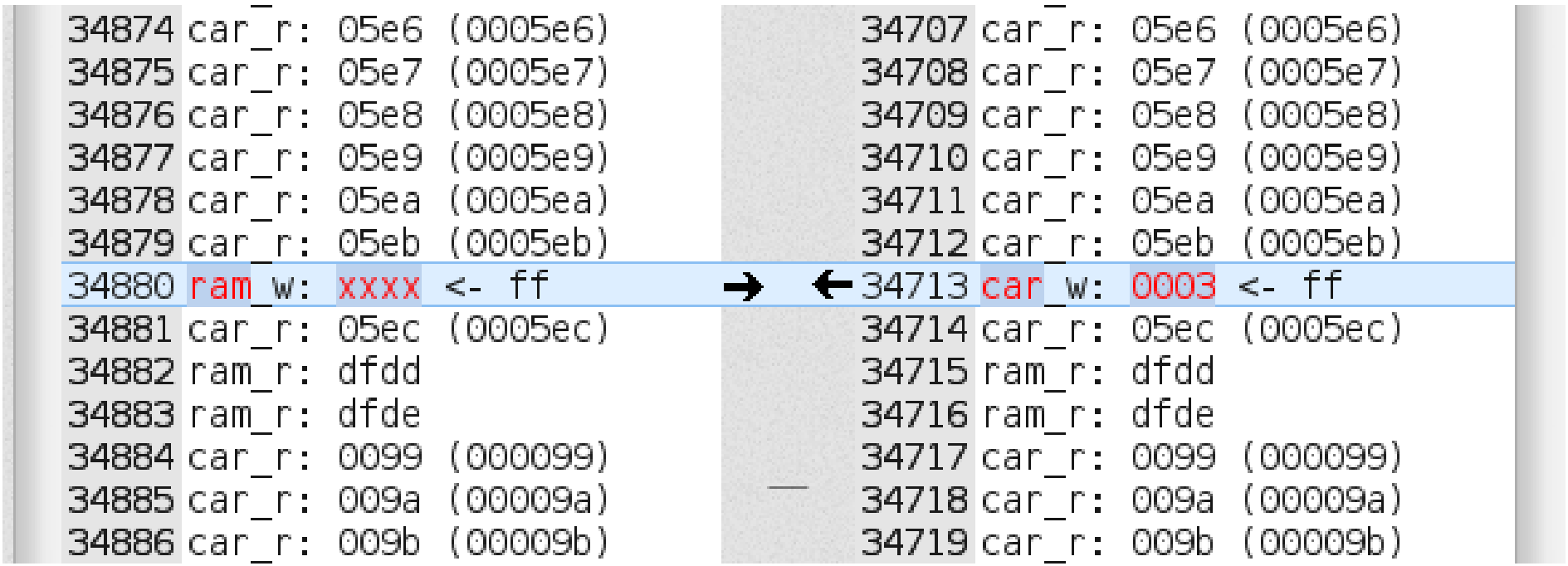
System Simulation vs Emulator



System Simulation vs Emulator



System Simulation vs Emulator



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

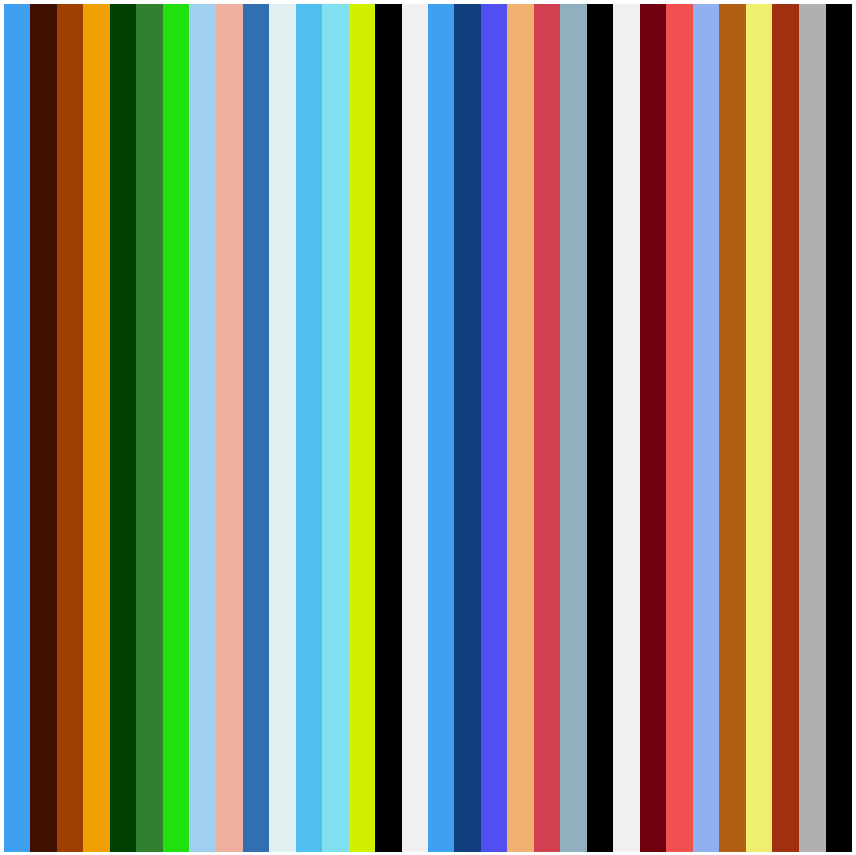


Figure : Color Palette

Figure : Complete Screen Render

Requirements/Ass. Metrics/Test Plan

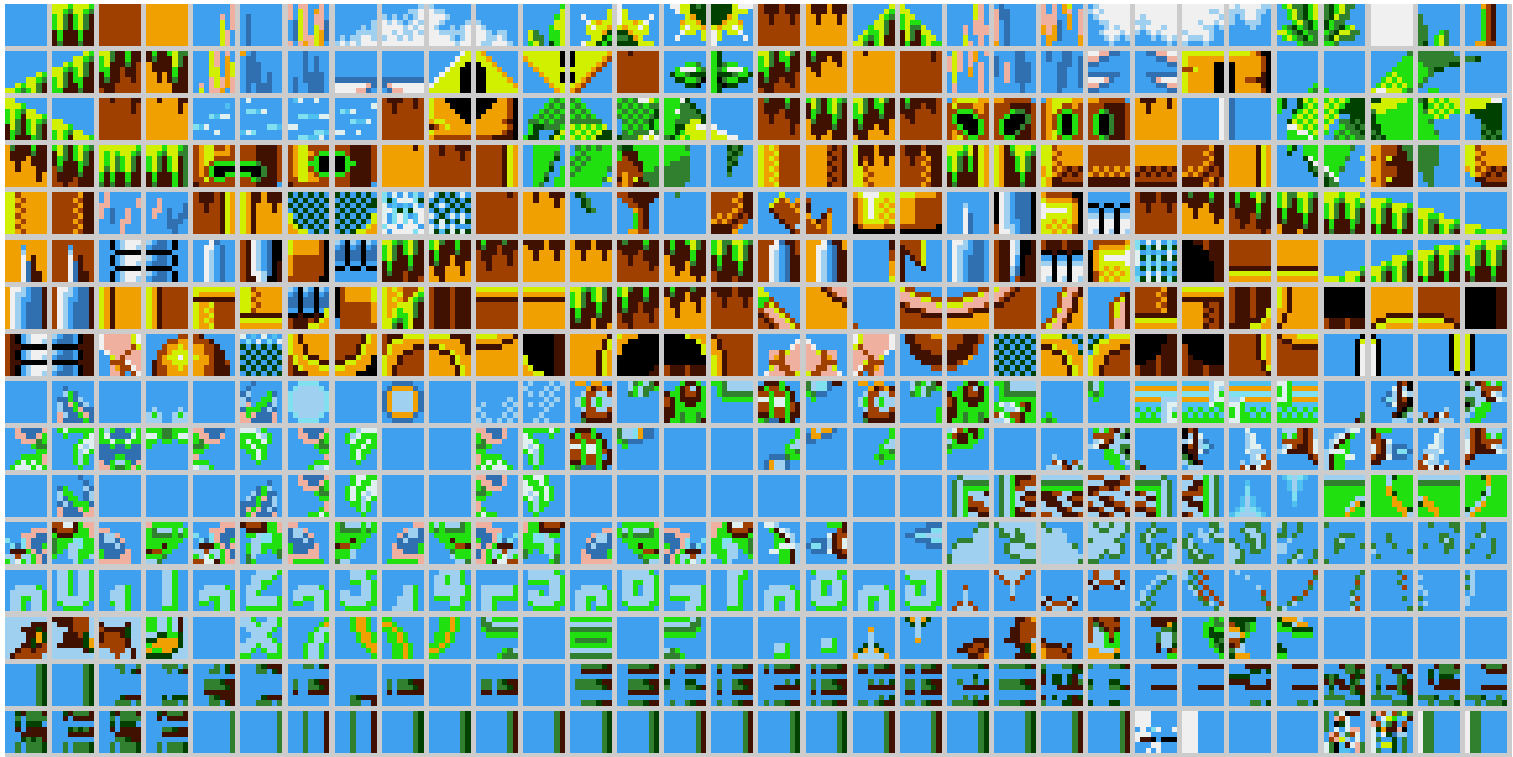
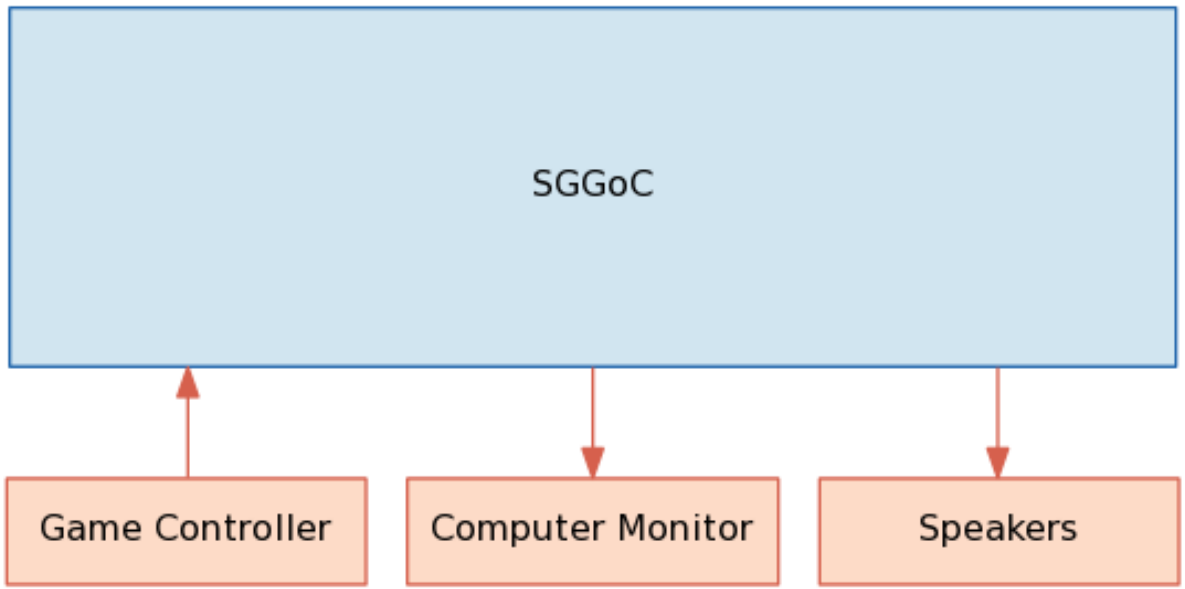


Figure : All 512 Tiles

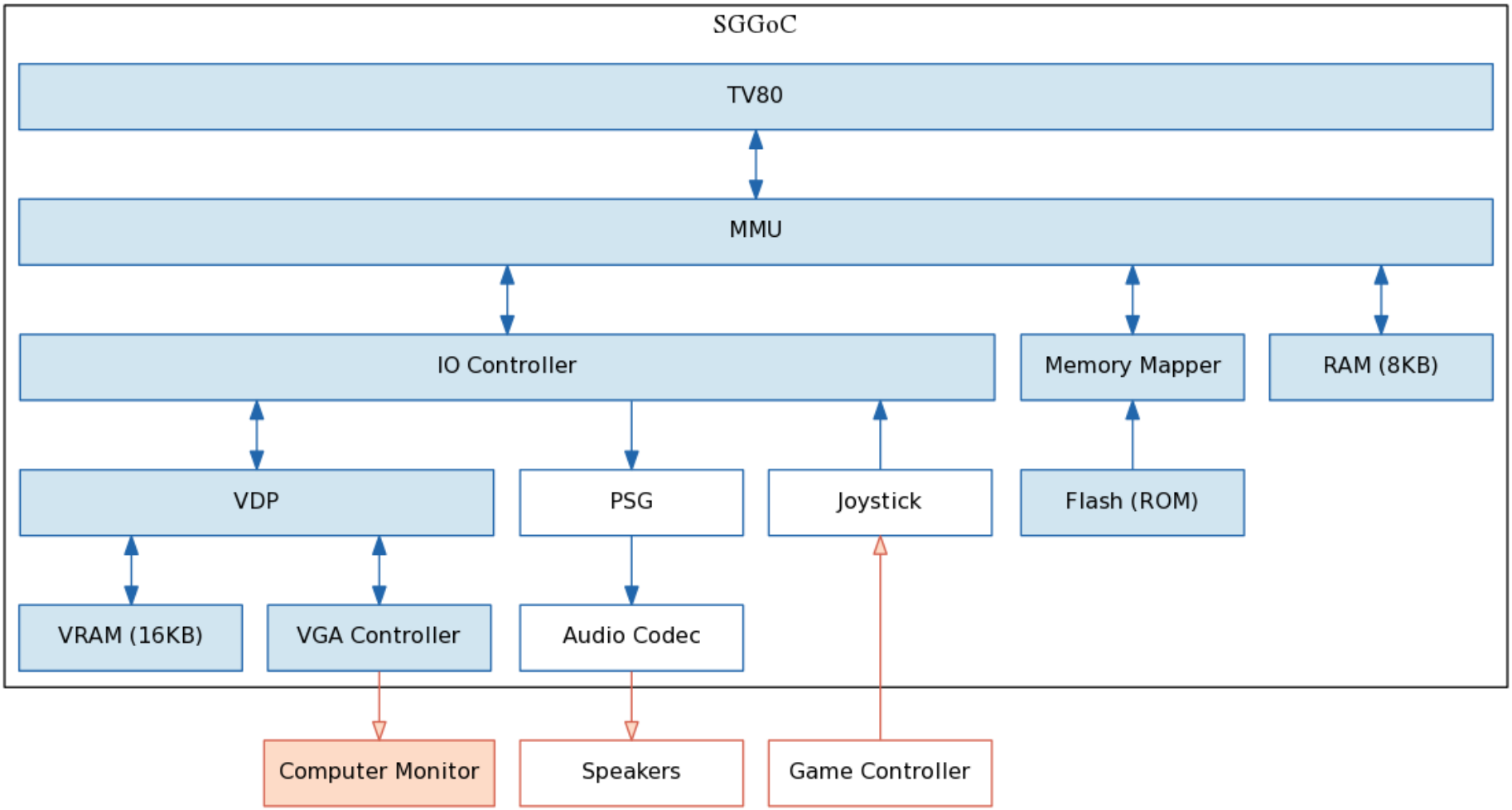
Design Overview

Black Box Diagram

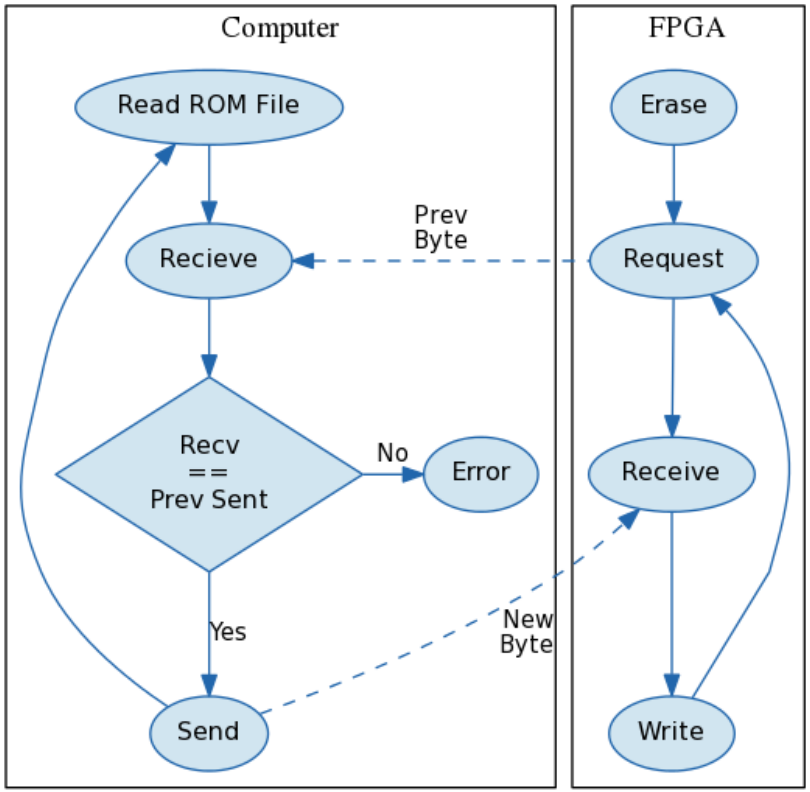


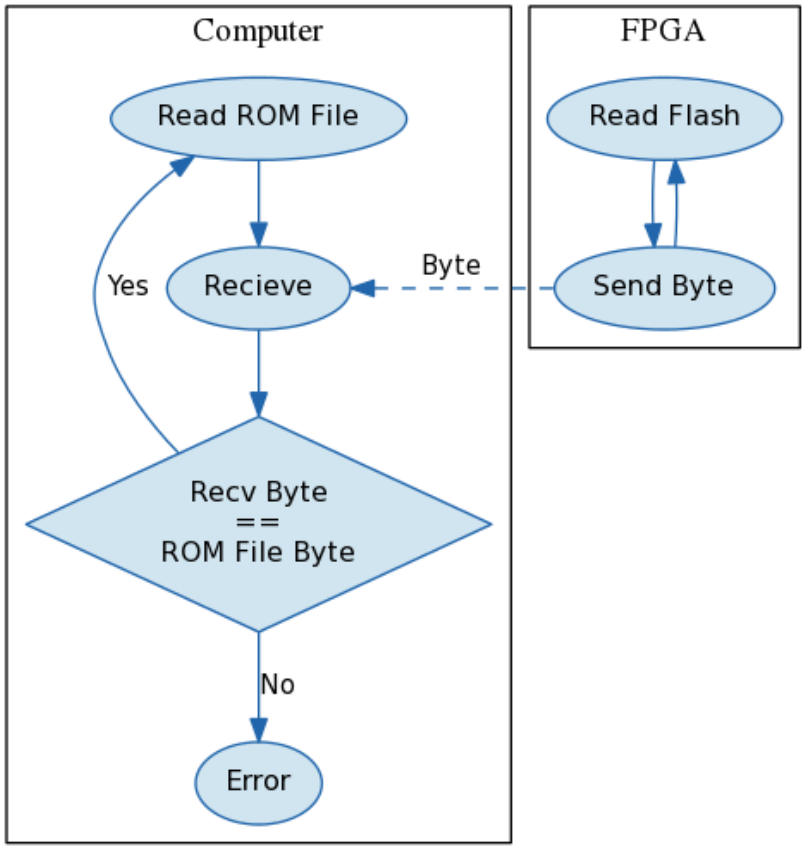
Design Overview

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/