Sega Game Gear on a Chip

Max Thrun — Samir Silbak

University of Cincinnati

Fall 2012

Underlying Goal

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

Purpose

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.

Purpose

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.

Purpose

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.

Purpose

Why?

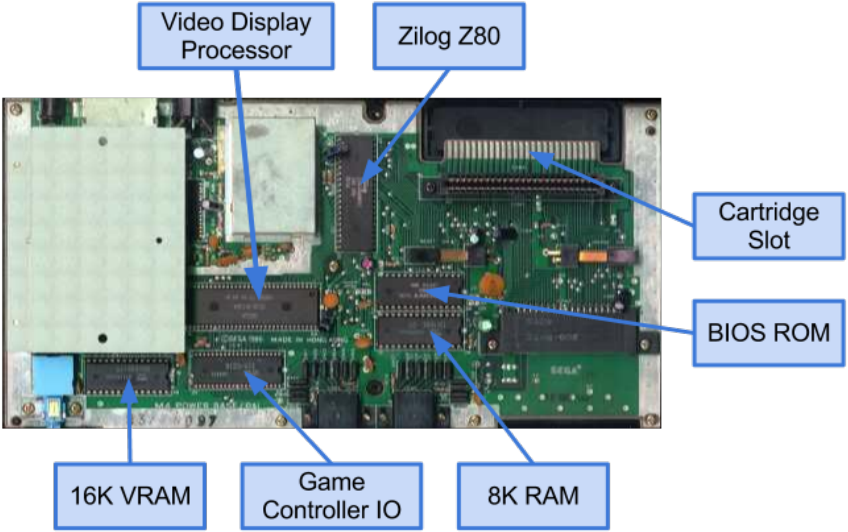
- **Maintainability** - You can no longer buy parts to service legacy computer systems
- **Upgradability** - Reimplementation gives an opportunity to add additional features
- **Portability** - Do not need all the original big clunky hardware. Reimplementation can be embedded in new designs.

Sega Game Gear

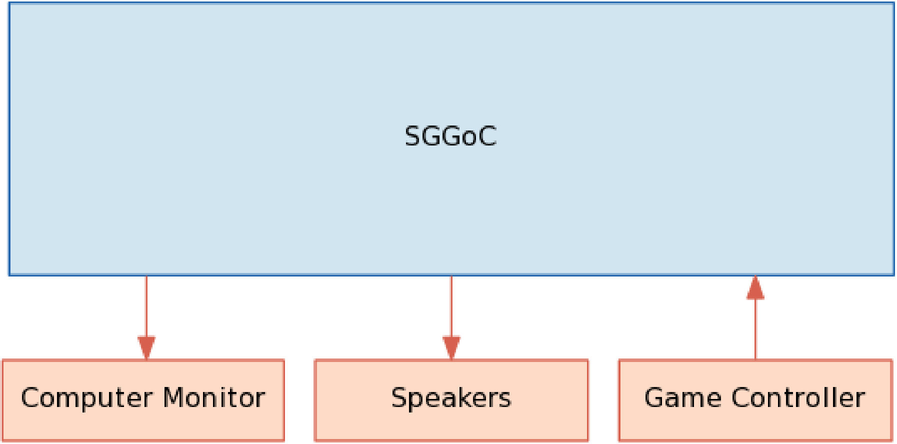
Sega Game Gear

- Released April 1990
- Mobile version of the Sega Master System (functionally identical)
- Standard system architecture for the time (Z80 CPU, tri-state buses, etc...)

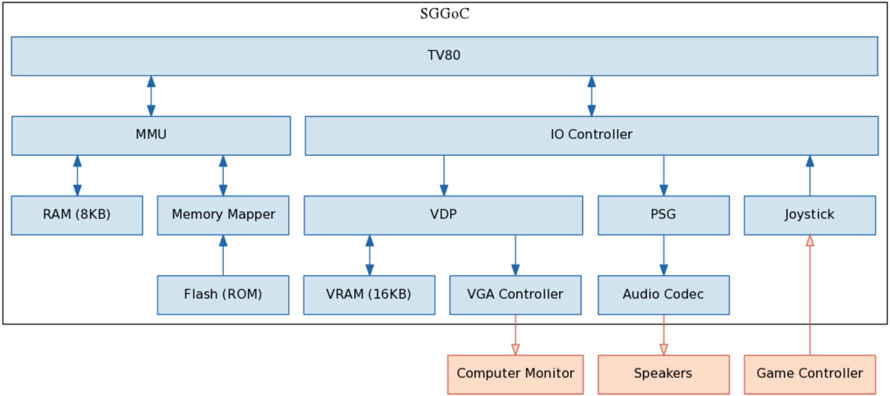
Sega Master System PCB



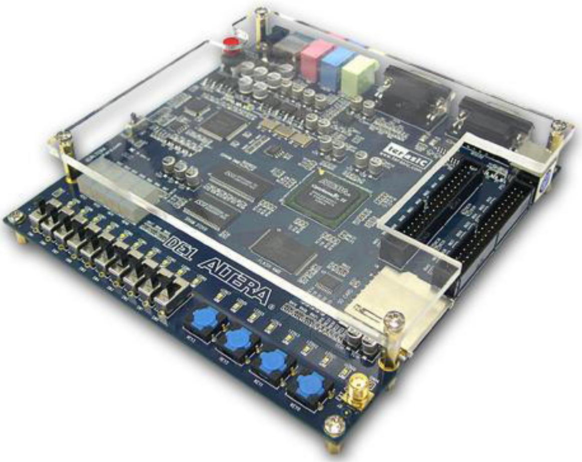
Black Box Diagram



Transparent Box Diagram

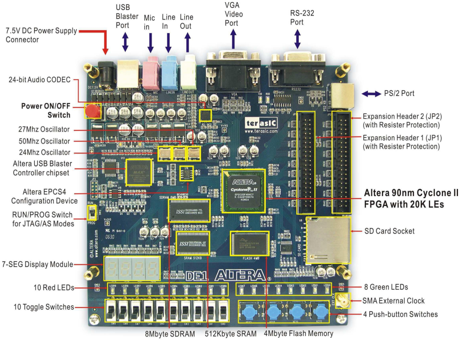


FPGA Development Board



Altera DE1

- Cyclone II EP2C20F484C7
- VGA, Audio, SD Card, 4 MB Flash
- Supports a familiar command line development environment
- Extremely good documentation



Design Strategy

1. Break down system components according to the original system architecture
2. Implement each component to match the original functionality described by official and non official documents
3. Test each components functionality 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 Strategy

1. Break down system components according to the original system architecture
2. Implement each component to match the original functionality described by official and non official documents
3. Test each components functionality 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)

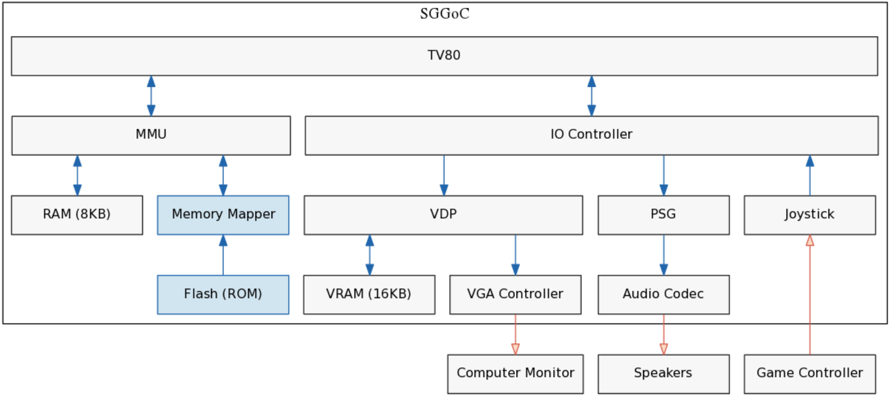
Design Strategy

1. Break down system components according to the original system architecture
2. Implement each component to match the original functionality described by official and non official documents
3. Test each components functionality 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 Strategy

1. Break down system components according to the original system architecture
2. Implement each component to match the original functionality described by official and non official documents
3. Test each components functionality 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)

Game Cartridge



Game Cartridge

Game Cartridge

Each game cartridge made up of at least two components:

- Game data ROM
- Memory Mapper

Practically every game ROM has been dumped as is available online

Game Cartridge

Each game cartridge made up of at least two components:

- Game data ROM
- Memory Mapper

Practically every game ROM has been dumped as is available online

Game Cartridge

Each game cartridge made up of at least two components:

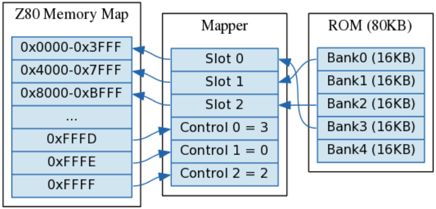
- Game data ROM
- Memory Mapper

Practically every game ROM has been dumped as is available online

Game Cartridge

Each game cartridge made up of at least two components:

- Game data ROM
- Memory Mapper

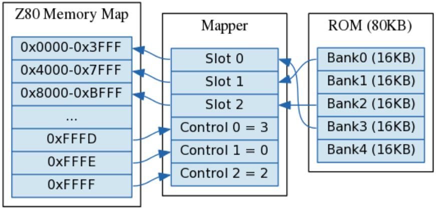


Practically every game ROM has been dumped as is available online

Game Cartridge

Each game cartridge made up of at least two components:

- Game data ROM
- Memory Mapper



Practically every game ROM has been dumped as is available online

Storing Game ROMs

A few options to store game ROMs:

1. Hookup the actual cartridge
    - Straight forward
    - Don't have to re-implement the memory mappers
    - Defeats most the point of the project
2. Store them on a SD card
    - Extremely portable / convenient
    - Complicated interface
3. Store them on the 4MB flash chip on the DE1
    - Fairly straightforward
    - Extremely non-portable
    - Flash chip looks just like original ROM chips

Storing Game ROMs

A few options to store game ROMs:

1. Hookup the actual cartridge
   - Straight forward
   - Don't have to re-implement the memory mappers
   - Defeats most the point of the project
2. Store them on a SD card
   - Extremely portable / convenient
   - Complicated interface
3. Store them on the 4MB flash chip on the DE1
   - Fairly straightforward
   - Extremely non-portable
   - Flash chip looks just like original ROM chips

Storing Game ROMs

A few options to store game ROMs:

1. Hookup the actual cartridge
    - Straight forward
    - Don't have to re-implement the memory mappers
    - Defeats most the point of the project
2. Store them on a SD card
    - Extremely portable / convenient
    - Complicated interface
3. Store them on the 4MB flash chip on the DE1
    - Fairly straightforward
    - Extremely non-portable
    - Flash chip looks just like original ROM chips

Storing Game ROMs

A few options to store game ROMs:

1. Hookup the actual cartridge
   - Straight forward
   - Don't have to re-implement the memory mappers
   - Defeats most the point of the project
2. Store them on a SD card
   - Extremely portable / convenient
   - Complicated interface
3. Store them on the 4MB flash chip on the DE1
   - Fairly straightforward
   - Extremely non-portable
   - Flash chip looks just like original ROM chips

Storing Game ROMs

A few options to store game ROMs:

1. Hookup the actual cartridge
   - Straight forward
   - Don't have to re-implement the memory mappers
   - Defeats most the point of the project
2. Store them on a SD card
   - Extremely portable / convenient
   - Complicated interface
3. Store them on the 4MB flash chip on the DE1
   - Fairly straightforward
   - Extremely non-portable
   - Flash chip looks just like original ROM chips

Storing Game ROMs

A few options to store game ROMs:

1. Hookup the actual cartridge
    - Straight forward
    - Don't have to re-implement the memory mappers
    - Defeats most the point of the project
2. Store them on a SD card
    - Extremely portable / convenient
    - Complicated interface
3. Store them on the 4MB flash chip on the DE1
    - Fairly straightforward
    - Extremely non-portable
    - Flash chip looks just like original ROM chips

Storing Game ROMs

A few options to store game ROMs:

1. Hookup the actual cartridge
    - Straight forward
    - Don't have to re-implement the memory mappers
    - Defeats most the point of the project
2. Store them on a SD card
    - Extremely portable / convenient
    - Complicated interface
3. Store them on the 4MB flash chip on the DE1
    - Fairly straightforward
    - Extremely non-portable
    - Flash chip looks just like original ROM chips

Storing Game ROMs

A few options to store game ROMs:

1. Hookup the actual cartridge
    - Straight forward
    - Don't have to re-implement the memory mappers
    - Defeats most the point of the project
2. Store them on a SD card
    - Extremely portable / convenient
    - Complicated interface
3. Store them on the 4MB flash chip on the DE1
    - Fairly straightforward
    - Extremely non-portable
    - Flash chip looks just like original ROM chips

Storing Game ROMs

A few options to store game ROMs:

1. Hookup the actual cartridge
   - Straight forward
   - Don't have to re-implement the memory mappers
   - Defeats most the point of the project
2. Store them on a SD card
   - Extremely portable / convenient
   - Complicated interface
3. Store them on the 4MB flash chip on the DE1
   - Fairly straightforward
   - Extremely non-portable
   - Flash chip looks just like original ROM chips

Storing Game ROMs

A few options to store game ROMs:

1. Hookup the actual cartridge
   - Straight forward
   - Don't have to re-implement the memory mappers
   - Defeats most the point of the project
2. Store them on a SD card
   - Extremely portable / convenient
   - Complicated interface
3. Store them on the 4MB flash chip on the DE1
   - Fairly straightforward
   - Extremely non-portable
   - Flash chip looks just like original ROM chips

Storing Game ROMs

A few options to store game ROMs:

1. Hookup the actual cartridge
    - Straight forward
    - Don't have to re-implement the memory mappers
    - Defeats most the point of the project
2. Store them on a SD card
    - Extremely portable / convenient
    - Complicated interface
3. Store them on the 4MB flash chip on the DE1
    - Fairly straightforward
    - Extremely non-portable
    - Flash chip looks just like original ROM chips

Storing Game ROMs

A few options to store game ROMs:

1. Hookup the actual cartridge
   - Straight forward
   - Don't have to re-implement the memory mappers
   - Defeats most the point of the project
2. Store them on a SD card
   - Extremely portable / convenient
   - Complicated interface
3. Store them on the 4MB flash chip on the DE1
   - Fairly straightforward
   - Extremely non-portable
   - Flash chip looks just like original ROM chips

Storing Game ROMs

A few options to store game ROMs:

1. Hookup the actual cartridge
   - Straight forward
   - Don't have to re-implement the memory mappers
   - Defeats most the point of the project
2. Store them on a SD card
   - Extremely portable / convenient
   - Complicated interface
3. Store them on the 4MB flash chip on the DE1
   - Fairly straightforward
   - Extremely non-portable
   - Flash chip looks just like original ROM chips

Storing Game ROMs

A few options to store game ROMs:

1. Hookup the actual cartridge
    - Straight forward
    - Don't have to re-implement the memory mappers
    - Defeats most the point of the project
2. Store them on a SD card
    - Extremely portable / convenient
    - Complicated interface
3. Store them on the 4MB flash chip on the DE1
    - Fairly straightforward
    - Extremely non-portable
    - Flash chip looks just like original ROM chips

ROM Flasher Tool

Need a tool to load a ROM file into the flash chip from the PC

1. Load RS232-to-Flash bridge into the FPGA
2. PC waits for FPGA to request a byte
3. PC send the next byte of ROM file
4. FPGA writes byte to flash
5. Go back to 2

Can also do the reverse to read back and verify the flash contents against the ROM file

ROM Flasher Tool

Need a tool to load a ROM file into the flash chip from the PC

1. Load RS232-to-Flash bridge into the FPGA
2. PC waits for FPGA to request a byte
3. PC send the next byte of ROM file
4. FPGA writes byte to flash
5. Go back to 2

Can also do the reverse to read back and verify the flash contents against the ROM file

ROM Flasher Tool

Need a tool to load a ROM file into the flash chip from the PC

1. Load RS232-to-Flash bridge into the FPGA
2. PC waits for FPGA to request a byte
3. PC send the next byte of ROM file
4. FPGA writes byte to flash
5. Go back to 2

Can also do the reverse to read back and verify the flash contents against the ROM file

ROM Flasher Tool

Need a tool to load a ROM file into the flash chip from the PC

1. Load RS232-to-Flash bridge into the FPGA
2. PC waits for FPGA to request a byte
3. PC send the next byte of ROM file
4. FPGA writes byte to flash
5. Go back to 2

Can also do the reverse to read back and verify the flash contents against the ROM file

ROM Flasher Tool

Need a tool to load a ROM file into the flash chip from the PC

1. Load RS232-to-Flash bridge into the FPGA
2. PC waits for FPGA to request a byte
3. PC send the next byte of ROM file
4. FPGA writes byte to flash
5. Go back to 2

Can also do the reverse to read back and verify the flash contents against the ROM file

ROM Flasher Tool

Need a tool to load a ROM file into the flash chip from the PC

1. Load RS232-to-Flash bridge into the FPGA
2. PC waits for FPGA to request a byte
3. PC send the next byte of ROM file
4. FPGA writes byte to flash
5. Go back to 2

Can also do the reverse to read back and verify the flash contents against the ROM file

ROM Flasher Tool

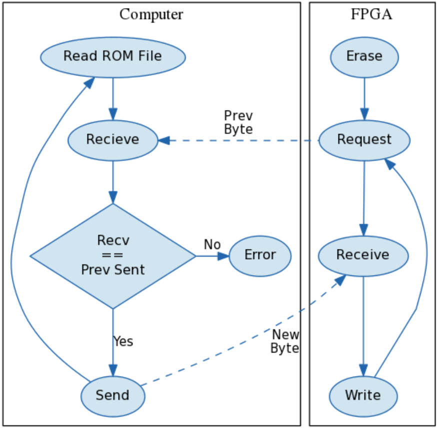
Need a tool to load a ROM file into the flash chip from the PC

1. Load RS232-to-Flash bridge into the FPGA
2. PC waits for FPGA to request a byte
3. PC send the next byte of ROM file
4. FPGA writes byte to flash
5. Go back to 2

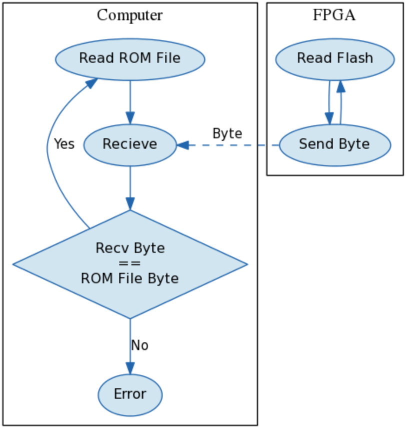
Can also do the reverse to read back and verify the flash contents against the ROM file

ROM Flasher Tool

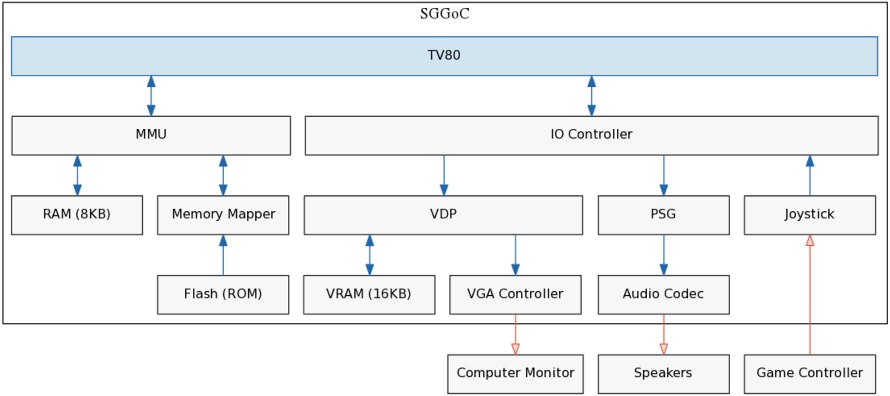
Writing



Reading

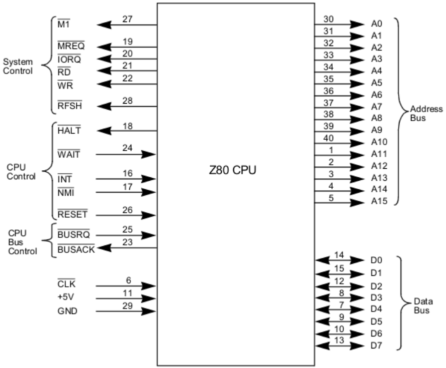


Zilog Z80


SGGoC
TV80
MMU
IO Controller
RAM (8KB)
Memory Mapper
VDP
PSG
Joystick
Flash (ROM)
VRAM (16KB)
VGA Controller
Audio Codec
Computer Monitor
Speakers
Game Controller

Zilog Z80

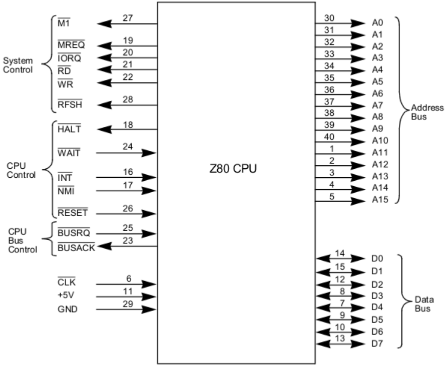
- Implementing a Zilog Z80 is totally outside the scope of this project
- Numerous implementations available online for free
- TV80 is an open source, proven, cycle accurate Z80 implementation written in Verilog



http://opencores.org/project,tv80

Zilog Z80

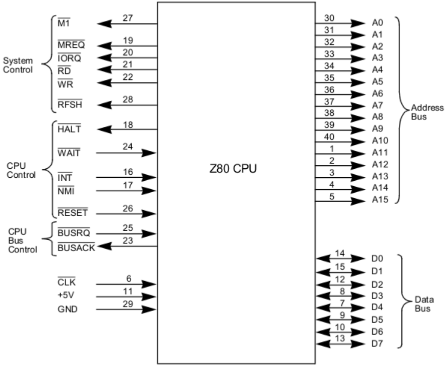
- Implementing a Zilog Z80 is totally outside the scope of this project
- Numerous implementations available online for free
- TV80 is an open source, proven, cycle accurate Z80 implementation written in Verilog



http://opencores.org/project,tv80

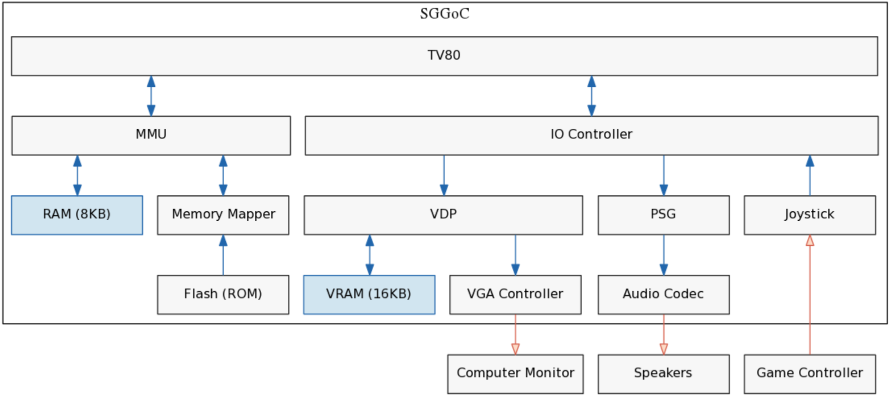
Zilog Z80

- Implementing a Zilog Z80 is totally outside the scope of this project
- Numerous implementations available online for free
- TV80 is an open source, proven, cycle accurate Z80 implementation written in Verilog



http://opencores.org/project,tv80

System and Video RAM



System and Video RAM

- Most modern FPGAs have more internal RAM than legacy systems
- Cyclon II has enough internal block RAM (30KB) to fit both system and video RAM
- *Design Strategy:* Write code that implies generic block RAM as opposed to device specific primitives to increase portability of codebase

http://danstrother.com/2010/09/11/inferring-rams-in-fpgas/

System and Video RAM

- Most modern FPGAs have more internal RAM than legacy systems
- Cyclon II has enough internal block RAM (30KB) to fit both system and video RAM
- *Design Strategy:* Write code that implies generic block RAM as opposed to device specific primitives to increase portability of codebase

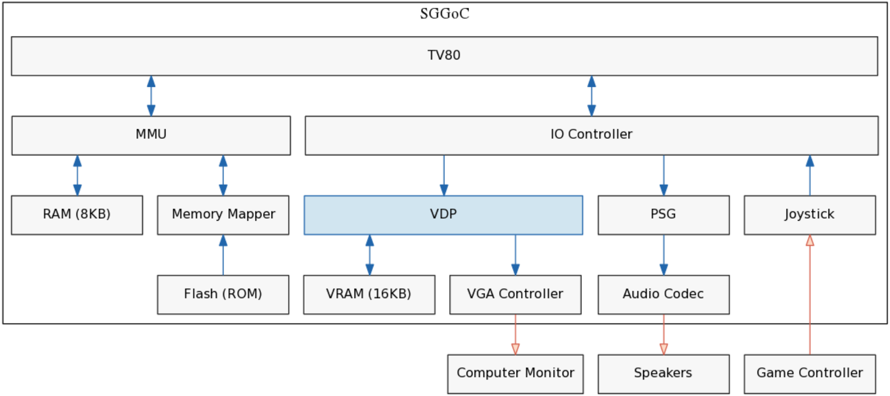
http://danstrother.com/2010/09/11/inferring-rams-in-fpgas/

System and Video RAM

- Most modern FPGAs have more internal RAM than legacy systems
- Cyclon II has enough internal block RAM (30KB) to fit both system and video RAM
- *Design Strategy:* Write code that implies generic block RAM as opposed to device specific primitives to increase portability of codebase

`http://danstrother.com/2010/09/11/inferring-rams-in-fpgas/`

Video Display Processor (VDP)



Video Display Processor (VDP)

- Texas Instruments TMS9918a
- Used in a number of legacy systems but no good HDL implementation exists online
- Meant to drive CRTs so functionality maps perfectly to the VGA interface on the DE1
- Communicates with the Z80 as an IO device

http://en.wikipedia.org/wiki/Texas_Instruments_TMS9918

Video Display Processor (VDP)

- Texas Instruments TMS9918a
- Used in a number of legacy systems but no good HDL implementation exists online
- Meant to drive CRTs so functionality maps perfectly to the VGA interface on the DE1
- Communicates with the Z80 as an IO device

`http://en.wikipedia.org/wiki/Texas_Instruments_TMS9918`

Video Display Processor (VDP)

- Texas Instruments TMS9918a
- Used in a number of legacy systems but no good HDL implementation exists online
- Meant to drive CRTs so functionality maps perfectly to the VGA interface on the DE1
- Communicates with the Z80 as an IO device

`http://en.wikipedia.org/wiki/Texas_Instruments_TMS9918`

Video Display Processor (VDP)

- Texas Instruments TMS9918a
- Used in a number of legacy systems but no good HDL implementation exists online
- Meant to drive CRTs so functionality maps perfectly to the VGA interface on the DE1
- Communicates with the Z80 as an IO device

`http://en.wikipedia.org/wiki/Texas_Instruments_TMS9918`

Testing Strategy

Game Gear is hard to test/verify

- Its only output is video
- Most documentation is 3rd party

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 logic analyzer on the real hardware

Testing Strategy

Game Gear is hard to test/verify

- Its only output is video
- Most documentation is 3rd party

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 logic analyzer on the real hardware

Testing Strategy

Game Gear is hard to test/verify

- Its only output is video
- Most documentation is 3rd party

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 logic analyzer on the real hardware

Testing Strategy

Game Gear is hard to test/verify

- Its only output is video
- Most documentation is 3rd party

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 logic analyzer on the real hardware

Testing Strategy

Game Gear is hard to test/verify

- Its only output is video
- Most documentation is 3rd party

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 logic analyzer on the real hardware

Testing Strategy

Game Gear is hard to test/verify

- Its only output is video
- Most documentation is 3rd party

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 logic analyzer on the real hardware

Testing Strategy

Game Gear is hard to test/verify

- Its only output is video
- Most documentation is 3rd party

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 logic analyzer on the real hardware