BameGoy

Nintendo GameBoy Hardware Emulator

Donovan Sproule (das2313), Nicolas Alarcon (na2946), Claire Cizdziel (ctc2156)

GameBoy Specs

SoC: Nintendo DMG-CPU (Sharp LR35902)

CPU: 4.194304 MHz Sharp SM83

Screen: 45.5mm x 41.5mm

Memory: 64 KB

- On SoC: 256 B "bootstrap" ROM, 127 B High RAM
- Internal: 8 KB RAM, 8 KB Video RAM
- External: (in cartridge) up to 1 MB ROM, up to 128 KB RAM

Resolution: 160 (w) × 144 (h) pixels (10:9 aspect ratio)

- 4 possible colors of gray

Power: 70–80 mAh, 4 x AA batteries

I/O: joypad, audio, graphics, LCD

(https://en.wikipedia.org/wiki/Game_Boy)

GB-Z80 Specs

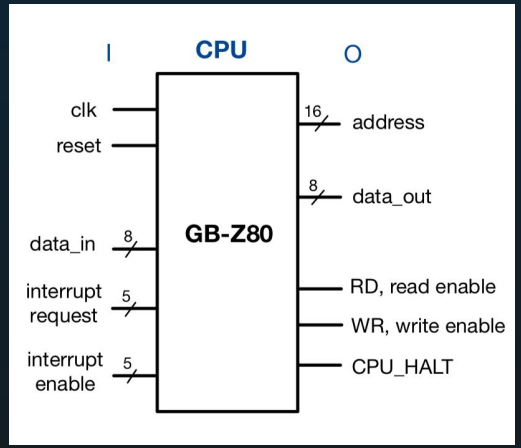
GB-Z80: Using modified open-source Z80, also operating at 4.19 MHz

Timing: Instructions in multiples of 4 cycles

Interrupts: V-Blank, LCD controller, timer, serial, and joypad

I/O: performed through memory load/store instructions

Modifications: for this specific usage, there were modifications, shown in the table on the right.



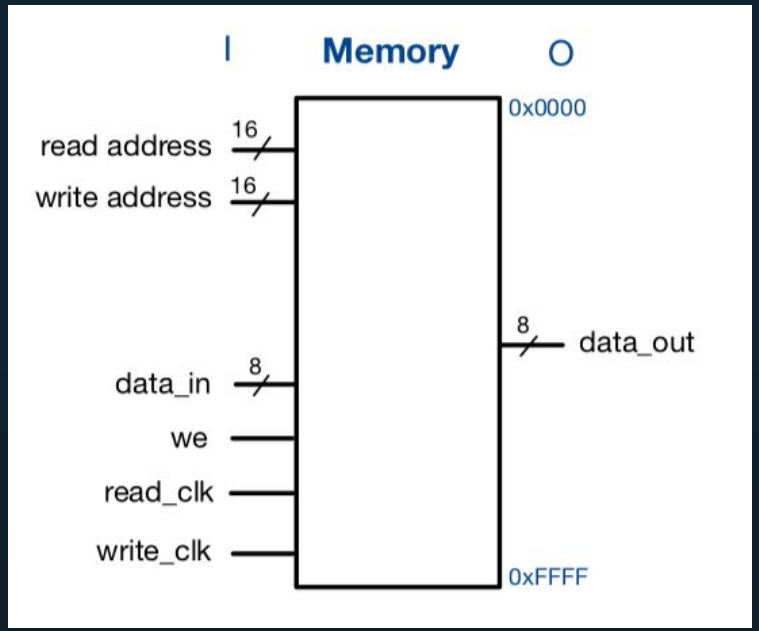
Opcode	Z80	GB CPU
08	EX AF,AF	LD (nn),SP
10	DJNZ PC+dd	STOP
22	LD (nn),HL	LDI (HL),A
2A	LD HL,(nn)	LDI A,(HL)
32	LD (nn),A	LDD (HL),A
3A	LD A,(nn)	LDD A,(HL)
D3	OUT (n),A	-
D9	EXX	RETI
DB	IN A,(n)	-
DD	<IX> prefix	-
E0	RET PO	LD (FF00+n),A
E2	JP PO,nn	LD (FF00+C),A
E3	EX (SP),HL	-
E4	CALL P0,nn	-
E8	RET PE	ADD SP,dd
EA	JP PE,nn	LD (nn),A
EB	EX DE,HL	-
EC	CALL PE,nn	-
ED	<prefix>	-
F0	RET P	LD A,(FF00+n)
F2	JP P,nn	LD A,(FF00+C)
F4	CALL P,nn	-
F8	RET M	LD HL,SP+dd
FA	JP M,nn	LD A,(nn)
FC	CALL M,nn	-
FD	<IY> prefix	-
CB 3X	SLL r/(HL)	SWAP r/(HL)

Memory

64 KiB Memory allocated as follows:

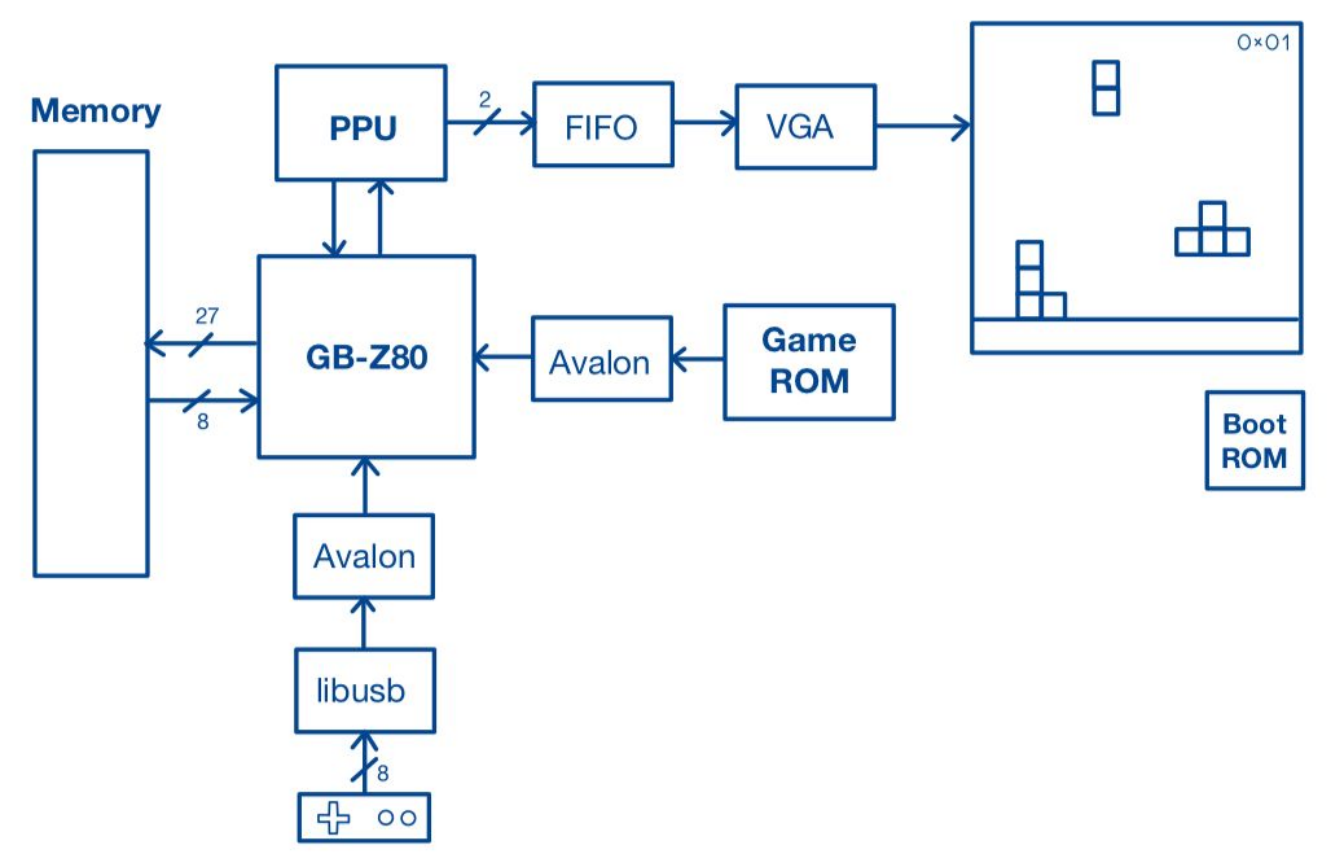
GameBoy Memory Areas

$FFFF	Interrupt Enable Flag
$FF80-$FFFE	Zero Page - 127 bytes
$FF00-$FF7F	Hardware I/O Registers
$FEA0-$FEFF	Unusable Memory
$FE00-$FE9F	OAM - Object Attribute Memory
$E000-$FDFF	Echo RAM - Reserved, Do Not Use
$D000-$DFFF	Internal RAM - Bank 1-7 (switchable - CGB only)
$C000-$CFFF	Internal RAM - Bank 0 (fixed)
$A000-$BFFF	Cartridge RAM (If Available)
$9C00-$9FFF	BG Map Data 2
$9800-$9BFF	BG Map Data 1
$8000-$97FF	Character RAM
$4000-$7FFF	Cartridge ROM - Switchable Banks 1-xx
$0150-$3FFF	Cartridge ROM - Bank 0 (fixed)
$0100-$014F	Cartridge Header Area
$0000-$00FF	Restart and Interrupt Vectors

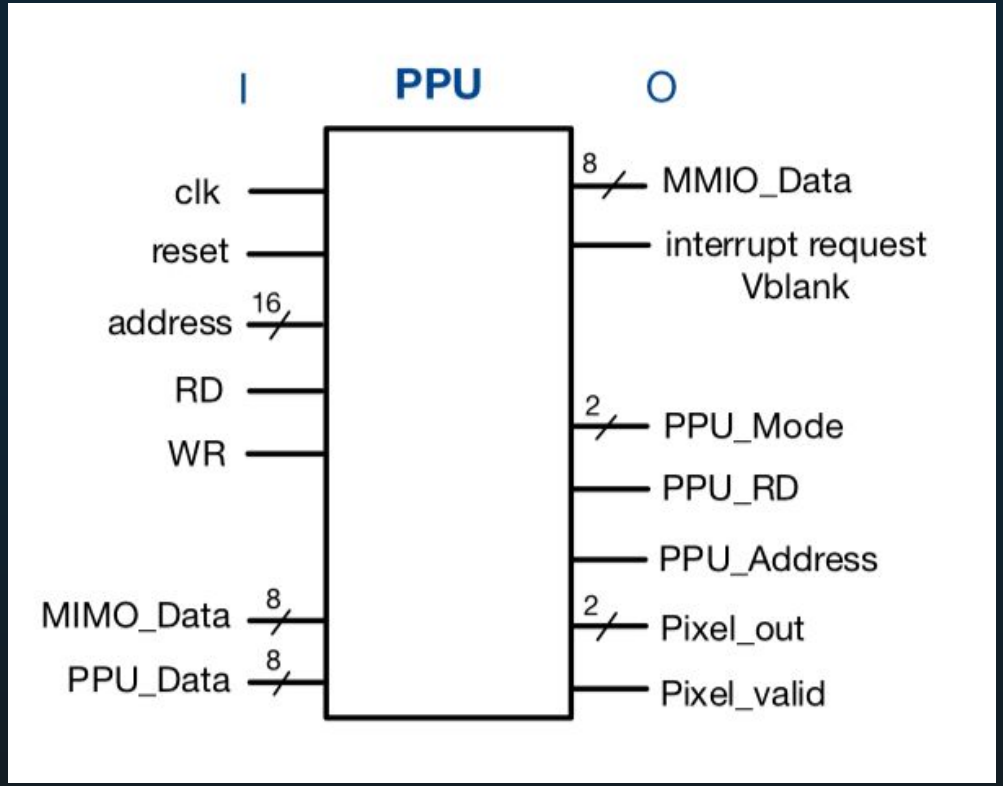


(https://crocidb.com/post/gameboy-emulator1/)

System Block Diagram



PPU


I
PPU
O
clk
reset
address
16
RD
WR
MIMO_Data
8
PPU_Data
8
8
MMIO_Data
interrupt request
Vblank
2
PPU_Mode
PPU_RD
PPU_Address
2
Pixel_out
Pixel_valid

PPU Modes

During a scanline, PPU enters specific modes with distinct functions:

Mode 2: OAM Scan

- PPU searches OAM memory for sprites and stores in buffer
- 80 T-cycles (PPU checks new OAM every 2 T-cycles)

Mode 3: Drawing

- PPU transfers pixels to LCD

Mode 0: H-Blank

- “Padding” for remainder of scanline - til reaches total of 456 T-cycles
- PPU “paused”

Mode 1: V-Blank

- “Padding” similar to H-Blank
- Takes place at the end of every frame for longer duration

PPU Modes & Timing

Object Attribute Memory (OAM)

GameBoy can display up to **40** moveable objects/sprites

- Maximum of **10** objects per scanline
- Each object consist of 4 bytes
- PPU compares (LCDC bit 2) to determine sprite height

Object attributes in memory at **$FE00-$FE9F**

Writing to OAM:

- Data is written to a buffer in WRAM
- WRAM copied to OAM
- Direct OAM writing only works during HBlank or VBlank

OAM Sprite Memory **$FE00 – $FE9F**

Byte 0: Y-Position

Actual Y position on screen = Byte 0 - 16

Byte 1 - X-Position:

Actual X position on screen = Byte 1 - 8

Byte 2 - Tile Number:

Unsigned 8 bit integer used for Sprite fetching.

Byte 3 - Sprite Flags:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OBJ-to-BG Priority	Y-Flip	X-Flip	Palette Number	Mode 0 STAT Interrupt Enable	Coincidence Flag	PPU Mode	PPU Mode

Bit 7: OBJ-to-BG Priority

0 = Sprite is always rendered above background

1 = Background colors 1-3 overlay sprite, sprite is still rendered above color 0

Bit 6: Y-Flip

If set to 1 the sprite is flipped vertically, otherwise rendered as normal

Bit 5: X-Flip

If set to 1 the sprite is flipped horizontally, otherwise rendered as normal

Bit 4: Palette Number

If set to 0, the OBP0 register is used as the palette, otherwise OBP1

Bit 3-0: CGB-Only flags

Object Attribute Memory (OAM)

OAM DMA Transfer:

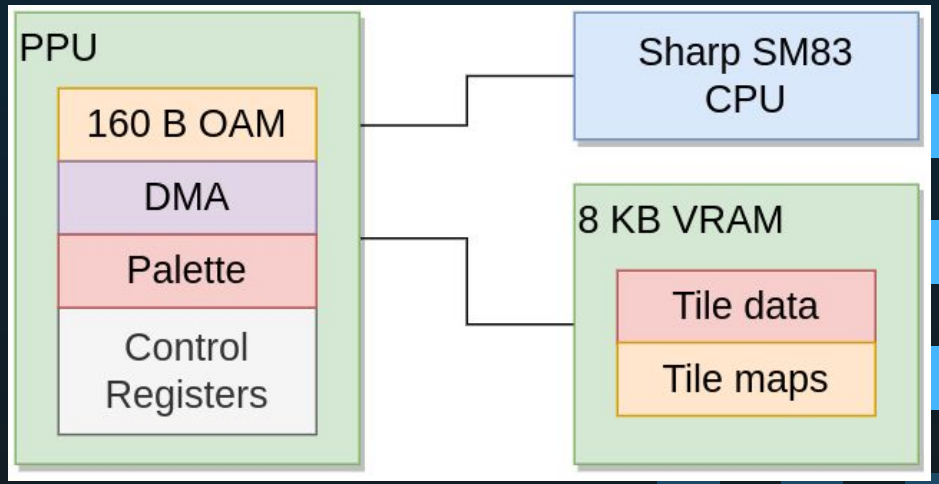
- Writing to **$FF46** – DMA initiates DMA transfer from **WRAM to OAM**

Selection priority:

- Only selects first (up-to) 10 objects to be drawn; can apply to off-screen objects
- Setting Y=0 would therefore "hide" sprite

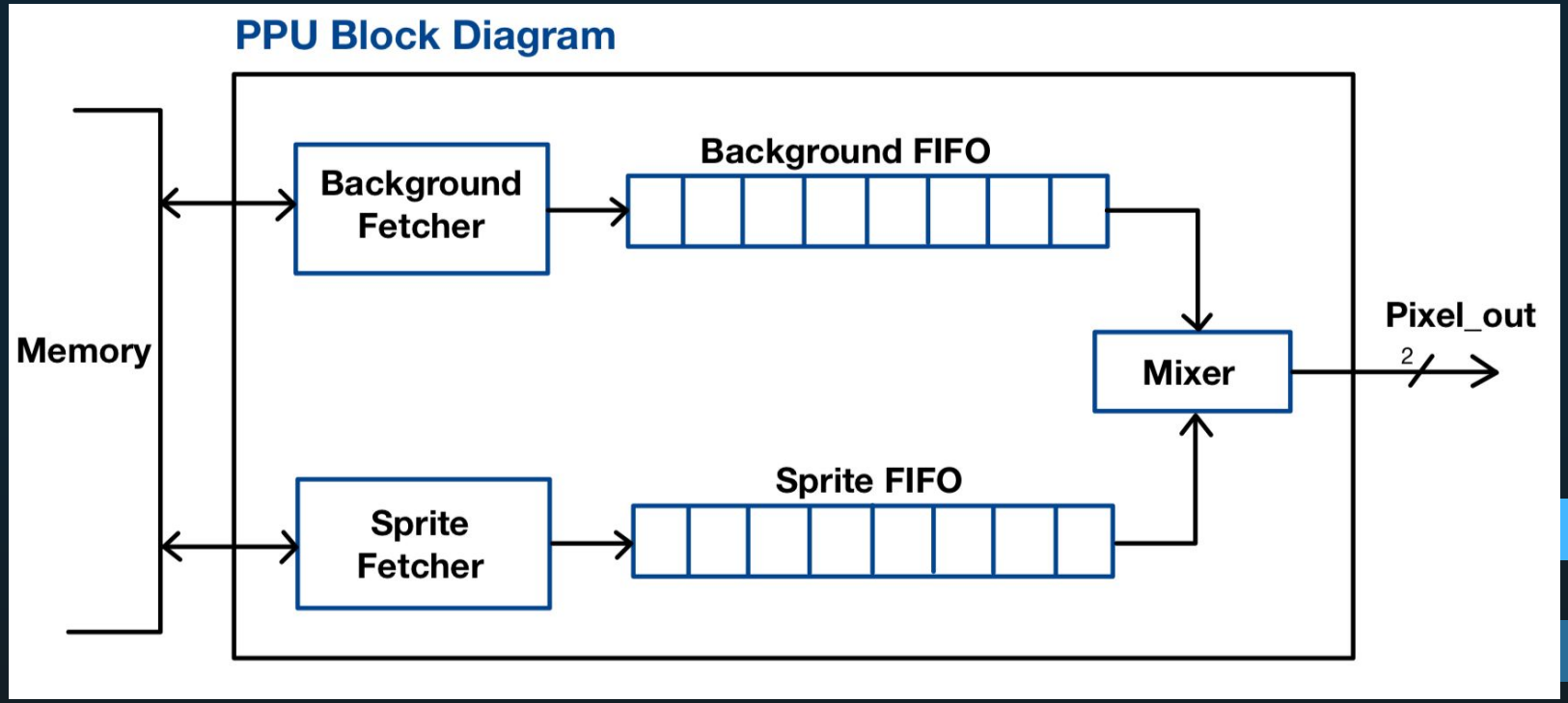
Drawing priority:

- Priority given to smaller X coordinate
- If X identical, sprite located first in OAM has higher priority



(https://www.copetti.org/writings/consoles/game-boy/)

PPU Block Diagram



Background Fetching

Fetching pixels takes 2 T-Cycles to complete, with the process:

1. Fetch Tile No.
    a. For background pixel - offsets tile data by 32 * (((LY + SCY) & 0xFF) / 8)
2. Fetch Tile Data (Low)
    a. Fetches first byte of tile data - with offset of 2 * ((LY + SCY) mod8)
3. Fetch Tile Data (High)
    a. Same as Low, except next byte is read and stored
4. Push to FIFO
    a. Only executes if FIFO is fully empty
    b. Step 4 usually restarts twice before pushing - Steps 1-3 take 6 T-cycles, PPU takes 8 T-cycles to shift out all 8 pixels

Pixel FIFO

Individual pixels are pushed to the LCD one by one

- Each pixel holds the information of color, palette, sprite priority, and background priority
- The **pixel fetcher** is responsible for loading the FIFO registers

Background and Pixel FIFO are merged when pushed to LCD

(https://hacktix.github.io/GBEDG/ppu/#the-concept-of-ppu-modes)

Background Scrolling

SCX register allows for scrolling background on a per-pixel basis

- While shifting pixels out of background FIFO, start of scanline **SCX mod 8** pixels are discarded
- Simultaneously, per-tile horizontal scrolling is handled with fetching process
- Results in PPU Mode 3 extending by SCX mod 8 cycles

(https://hacktix.github.io/GBEDG/ppu/#the-concept-of-ppu-modes)

Sprite Fetching

The Sprite Fetcher works very similarly to the background fetcher:

1. Fetch Tile No.
2. Fetch Tile Data (Low)
3. Fetch Tile Data (High)
4. Push to FIFO

- Tiles taken from **$8000-$87FF** and unsigned numbering

If (X-Position of any sprite in buffer) ≤ (current Pixel-X-Position + 8), sprite fetch is initiated:

- resets the Background Fetcher to step 1 and temporarily pauses it
- pixel shifter to the LCD is also suspended

(https://gbdev.gg8.se/wiki/articles/OAM_DMA_tutorial)

Sprite Timing

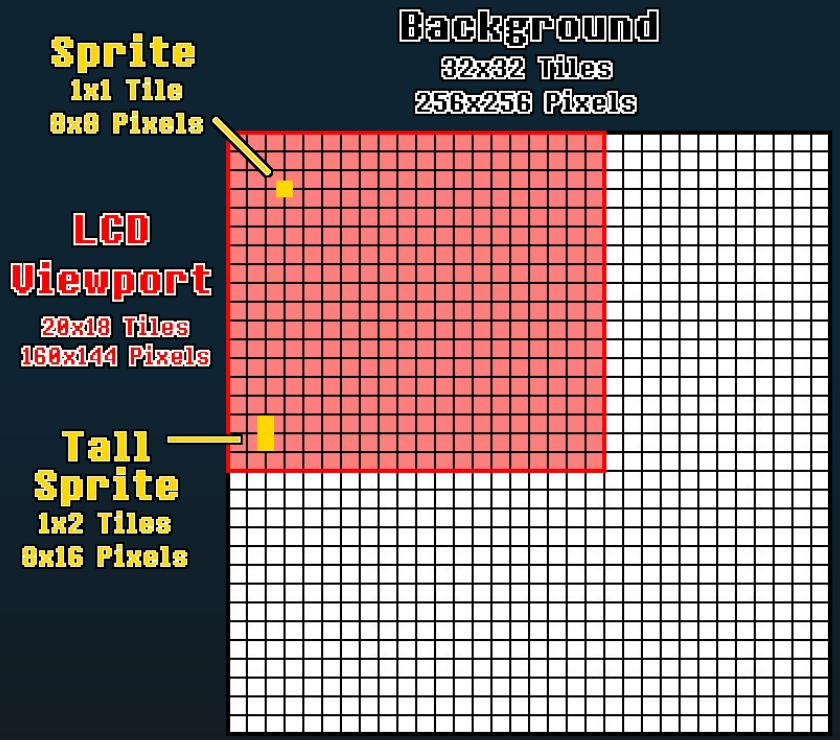
Once sprite fetch is completed

- PPU starts pushing pixels to LCD
- Background fetcher is restarted
- Delay occurs if <6 pixels remaining in Background FIFO (delay=*6 - REMAINING_PIXEL_COUNT*)

Sample timing diagram:

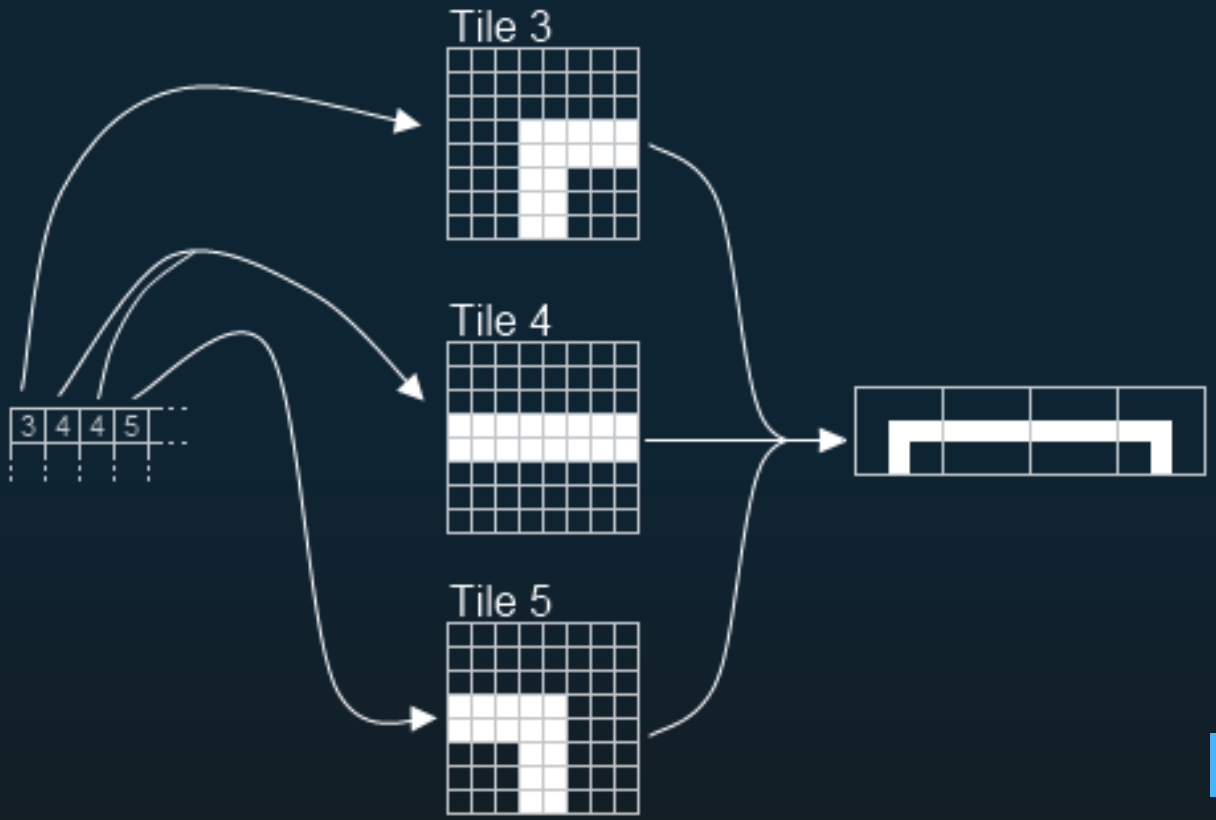
(https://hacktix.github.io/GBEDG/ppu/#the-concept-of-ppu-modes)

Sprite Timing



(https://hacktix.github.io/GBEDG/ppu/#the-concept-of-ppu-modes)

Example of tiles creating background map:



(https://imrannazar.com/GameBoy-Emulation-in-JavaScript:-Graphics)

Tile Data

Tile data stored in **VRAM** in **$8000-$97FF**

Tiles can be displayed as part of **Background/Window** and/or moveable **Sprites**

$8000 Method:

- $8000: base pointer
- TILE_NUMBER: unsigned 8-bit integer
- Add $8000 to (TILE_NUMBER * 16)

$8800 Method:

- $9000: base pointer
- SIGNED_TILE_NUMBER: signed 8-bit integer
- Add $9000 to (SIGNED_TILE_NUMBER * 16)

Sample tile data

(https://gbdev.io/pandocs/Tile_Data.html)

Tile Map

Game Boy contains two 32×32 tile maps in VRAM at **$9800-$9BFF** and **$9C00-$9FFF**

Each tile map has **1 byte** indexes of tiles to be displayed

- Tiles obtained through address from VRAM selected via LCDC register

Background-to-Object Priority

- BG/OBJ priority declared in **3** places: BG Map bit 7, LCDC bit 0, OAM bit 7
- Sample priority table shown below:

LCDC bit 0	OAM attr bit 7	BG attr bit 7	Priority
0	0	0	OBJ
0	0	1	OBJ
0	1	0	OBJ
0	1	1	OBJ
1	0	0	OBJ
1	0	1	BG color 1–3, otherwise OBJ
1	1	0	BG color 1–3, otherwise OBJ
1	1	1	BG color 1–3, otherwise OBJ

(https://gbdev.io/pandocs/Tile_Data.html)

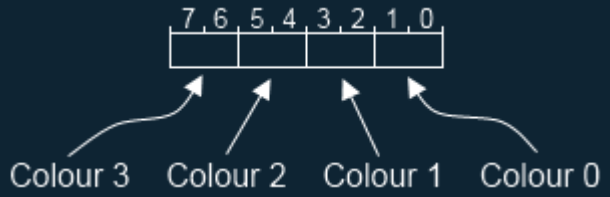
Interrupt Handling

VBlank	INT $40	Requested every time Game Boy enters VBlank
STAT	INT $48	Can be triggered by various sources
Timer	INT $50	Requested every time the timer overflows
Serial	INT $58	Requested upon completion of a serial data transfer (8 serial clock cycles after starting transfer)
Joypad	INT $60	Requested when a button is pressed

Color Rendering

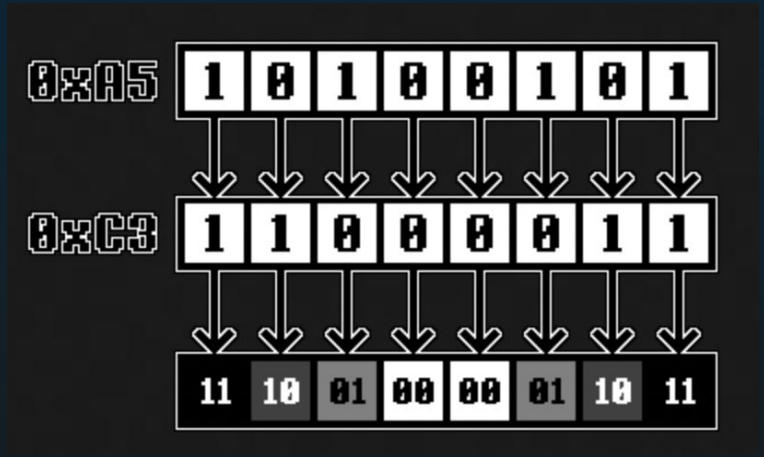
The GameBoy is capable of **4** colors

- Each color takes **2 bits**
- Each tile in tile data set is held in 16 bytes



Each color is achieved with the equation:

- (Each bit of first byte) + (bit in same position on second byte) => calculates color number

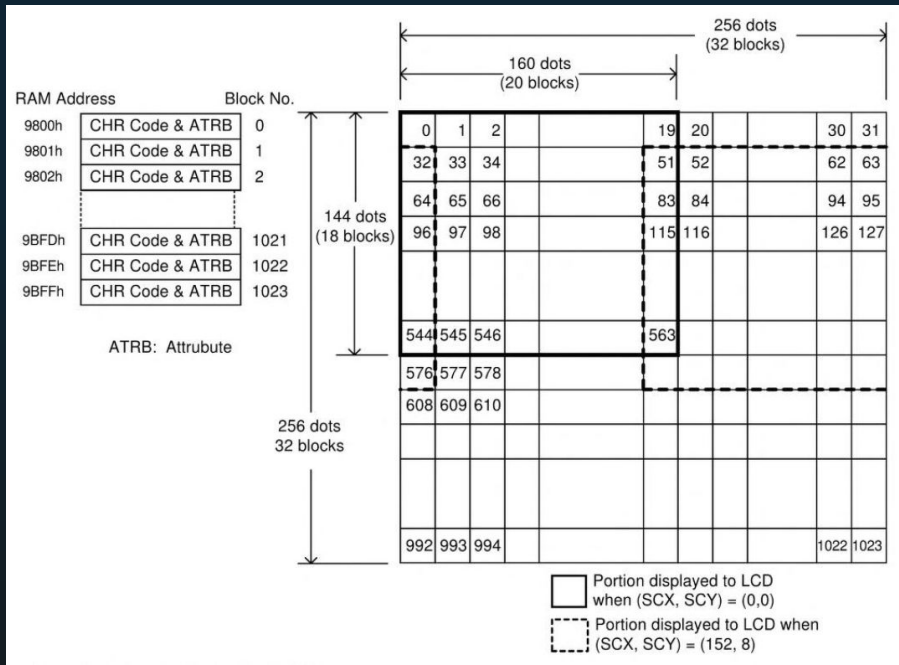


(https://imrannazar.com/GameBoy-Emulation-in-JavaScript:-Graphics)

Scrolling

GameBoy scrolling is 160x144 pixels, but background map is 256x256 pixels → **scrolling**

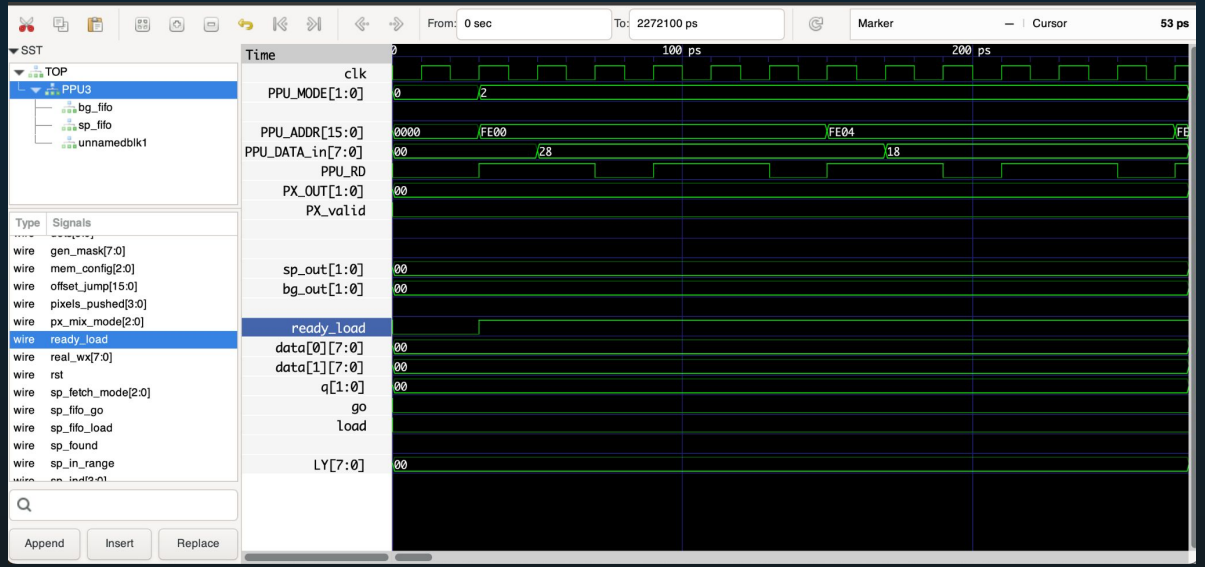
- Background is defined at the top-left of screen
- By moving this point between frames, the background can scroll
- Top-left is defined by registers **Scroll X**, **Scroll Y**



Timing

Proper timing ensure proper synchronization with the CPU, frame rate and refresh rate, memory access, and rendering pipelines (For testing)

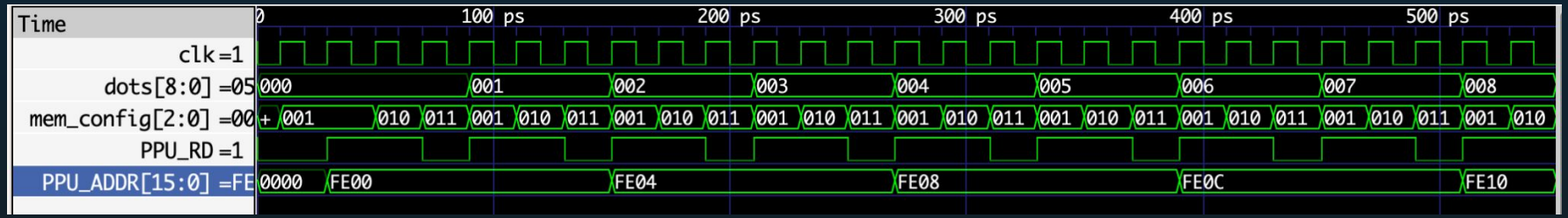
- Calibrated clock cycles using gtkwave
- Example final timing diagram of PPU



Timing

Example timing diagram of memory mechanism

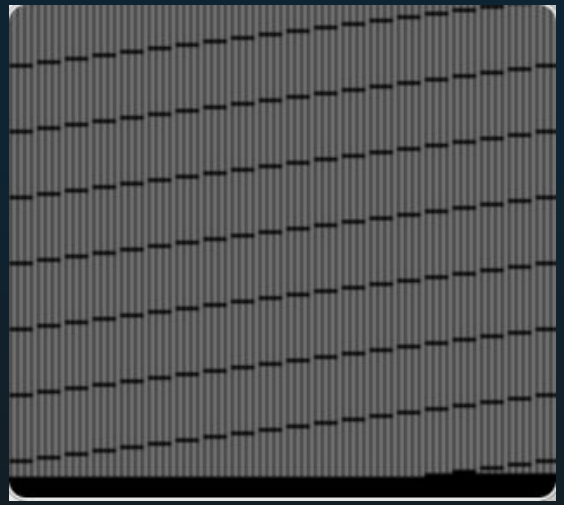
- “dots [8:0]” represents instructions processed by the PPU, for the sake of testing
- System stalls during testing but not deployment
- “dot” increments, then moves onto the next instruction.



PPU Testing

Using interactive test benches, gradually built up capabilities of PPU

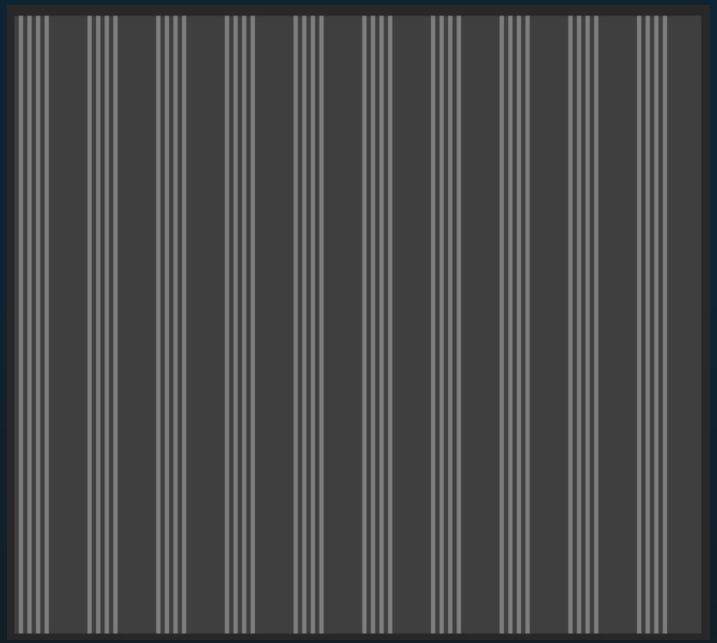
- Example images of initial PPU testing
- Found issues with timing and synchronization



PPU Testing

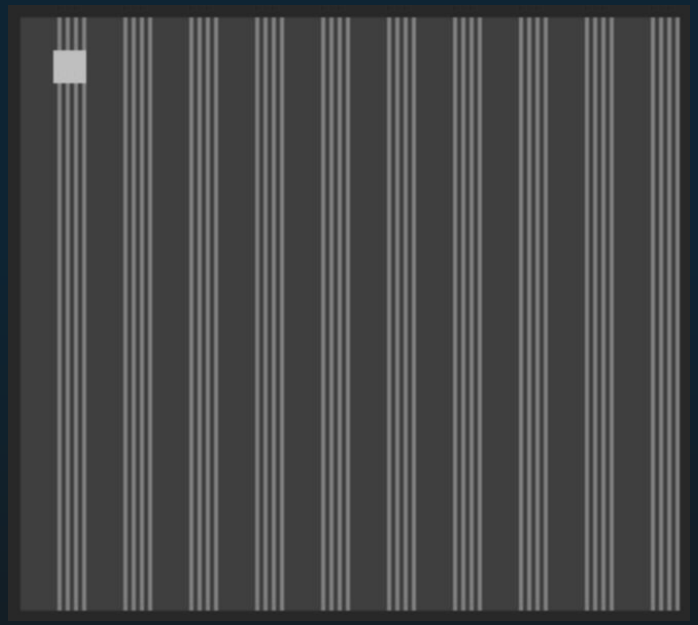
Tile map drawn from one tile

Tile map drawn from multiple tiles

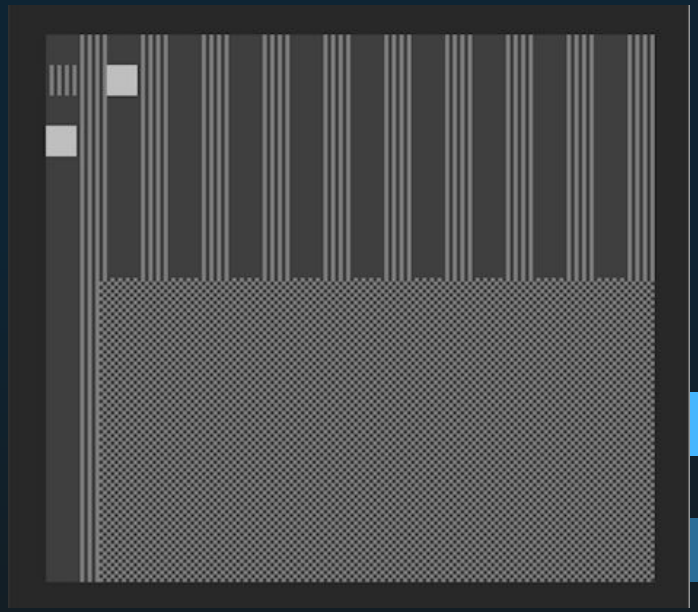


PPU Testing

Sprites implemented

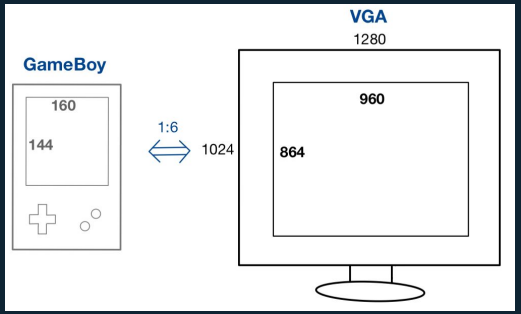


Sprites implemented and handles LCDC flags, window, multiple BG maps, and scrolling



DONE!

Video



GameBoy Video: 160x144 pixels

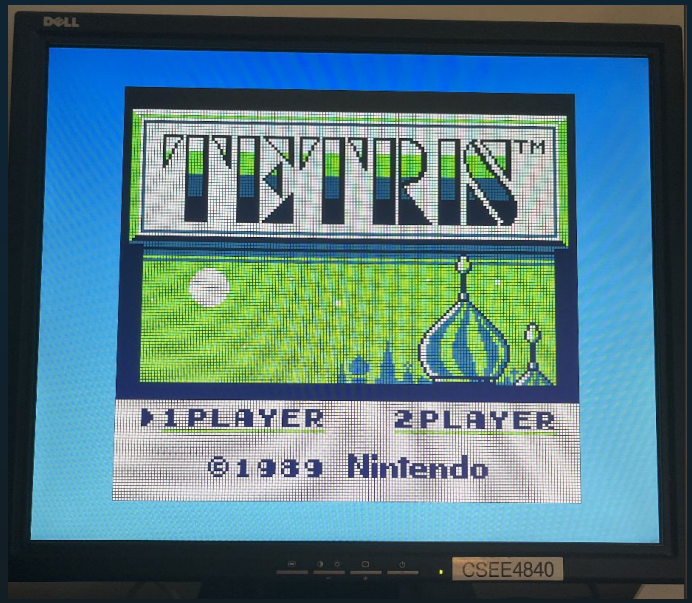
VGA: 1280 x 1024 pixels @ 60Hz

VGA Game Window: **960 x 864 pixels** (centered on screen)

- GameBoy pixel data written into framebuffer @ 4MHz
- VGA reads framebuffer with pixel frequency @ 108 MHz
  - One pixel every cycle
  - Referenced Lab 3 as a VGA scaling resource
- Background set to Pantone 292!

General timing

Screen refresh rate	60 Hz
Vertical refresh	63.981042654028 kHz
Pixel freq.	108.0 MHz

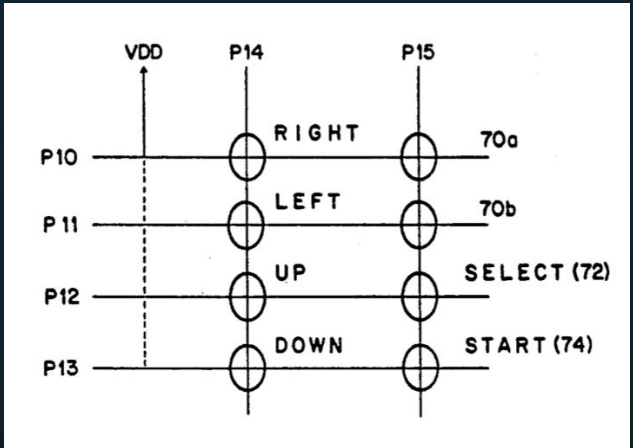


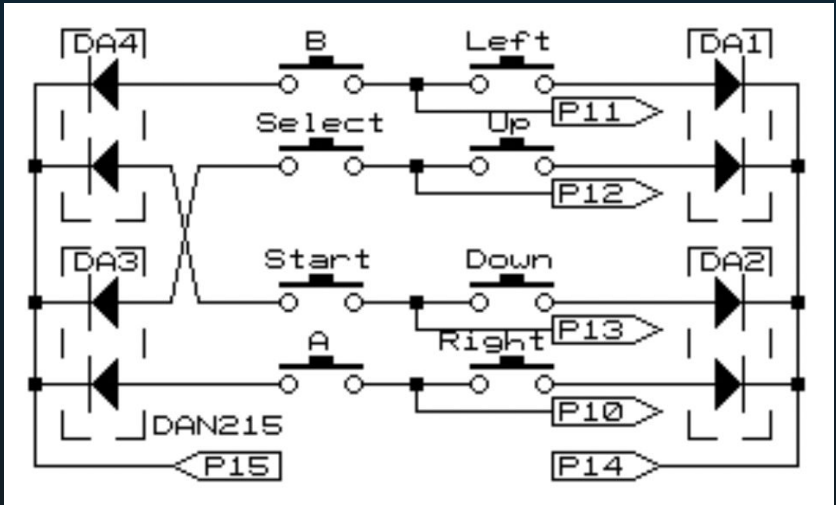
(http://www.tinyvga.com/vga-timing/1280x1024@60Hz)

Joypad

NES Controller

- Based on states of output pins (P14-15) and input pins (P10-13), CPU can identify a button press
- Configured to our CPU using libusb (referenced lab2)





(https://gbdev.gg8.se/wiki/articles/DMG_Schematics)

Cartridge

Obtained Tetris cartridge from online open source

Cartridge header (**$0100-$014F**) provides the information for running a game, namely:

- 0147 - Cartridge type
- 0148 - ROM Size
- 0149 - RAM Size

Cartridge ROM loaded to On-Chip RAM on DE1-SoC

(https://www.gamestop.com/game-boy-2/products/tetris---game-boy)

PPU Compilation

Analysis and Synthesis Resource Utilization of PPU

Analysis & Synthesis Resource Utilization by Entity

<<Filter>>

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers
2	|PPU3:GB_PPU|	734 (729)	488 (456)
1	|PPU_SHIFT_REG:bg_fifo|	3 (3)	16 (16)
2	|PPU_SHIFT_REG:sp_fifo|	2 (2)	16 (16)

Flow Summary

<<Filter>>

Flow Status	Successful - Sat Aug 3 15:18:00 2024
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	soc_system
Top-level Entity Name	soc_system_top
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	3,576 / 32,070 (11 %)
Total registers	2493
Total pins	356 / 457 (78 %)
Total virtual pins	0
Total block memory bits	705,792 / 4,065,280 (17 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	1 / 6 (17 %)
Total DLLs	1 / 4 (25 %)

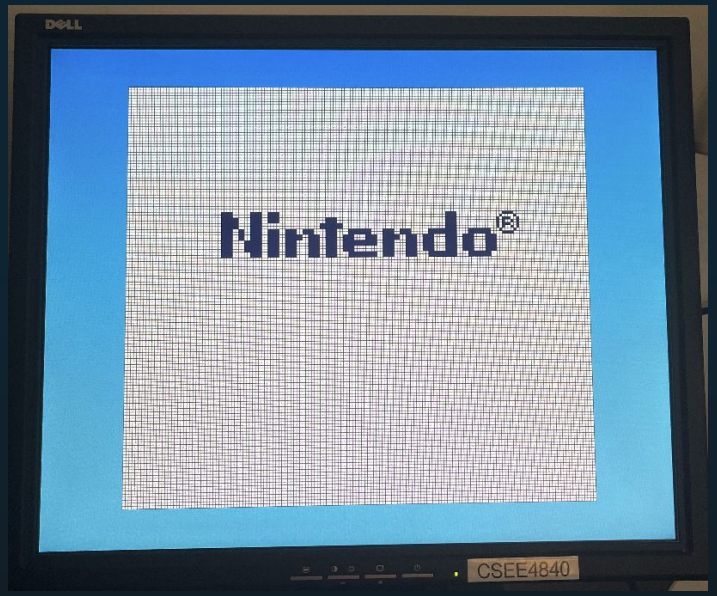
Analysis & Synthesis Resource Usage Summary

<<Filter>>

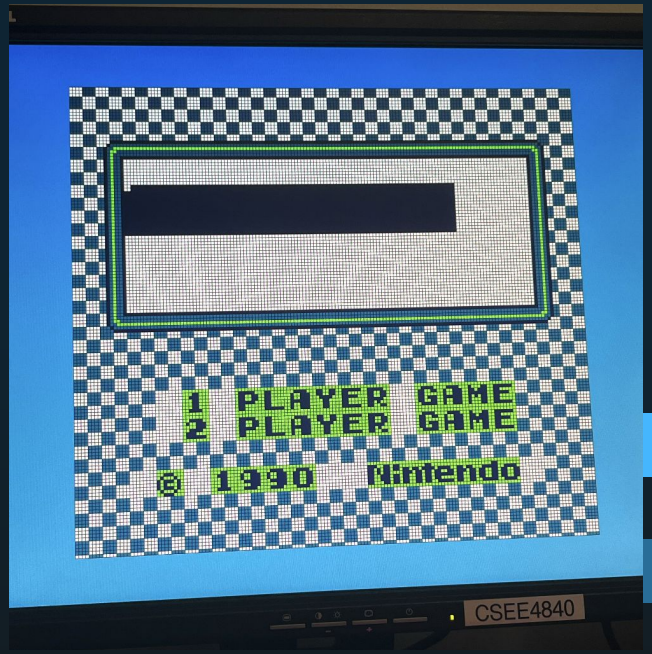
	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	3694
2		
3	Combinational ALUT usage for logic	5620
1	-- 7 input functions	115
2	-- 6 input functions	1518
3	-- 5 input functions	1256
4	-- 4 input functions	1102
5	-- <=3 input functions	1629
4		
5	Dedicated logic registers	2211
6		
7	I/O pins	356
8	I/O registers	186
9	Total MLAB memory bits	0
10	Total block memory bits	705792
11		
12	Total DSP Blocks	0
13		
14	Total PLLs	4
1	-- PLLs	4
15		
16	Total DLLs	1
17	Maximum fan-out node	soc_system:soc_system0|soc_system_Main_PL...ll|altera_pll:altera_pll_i|outclk_wire[1]
18	Maximum fan-out	903
19	Total fan-out	37130

Game Play Testing with PPU

Game Boy boot screen



Dr. Mario start screen



Open Source Emulator

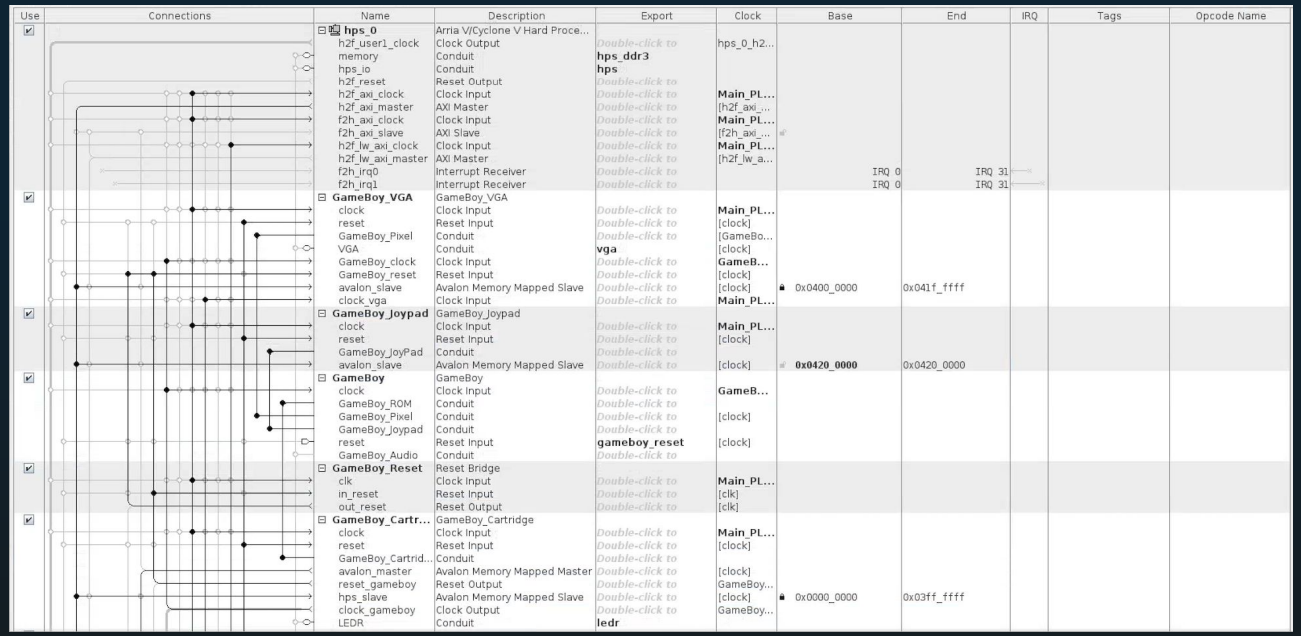
We utilized an open source emulator to base our systems on

- Used functional CPU in conjunction with our **PPU**, **hardware/software interface**, and **peripherals (joypad)**
- Independently tested our PPU with testbenches, performing successfully in simulation
    - Ran into issues with integration into larger emulator
    - Confirmed PPU issues by successfully running open source game play

Qsys File

Tested open source CPU to ensure functionality

- Generated Qsys file shown below
- Managed interconnects and removed SDRAM (used utilized on-chip RAM instead)

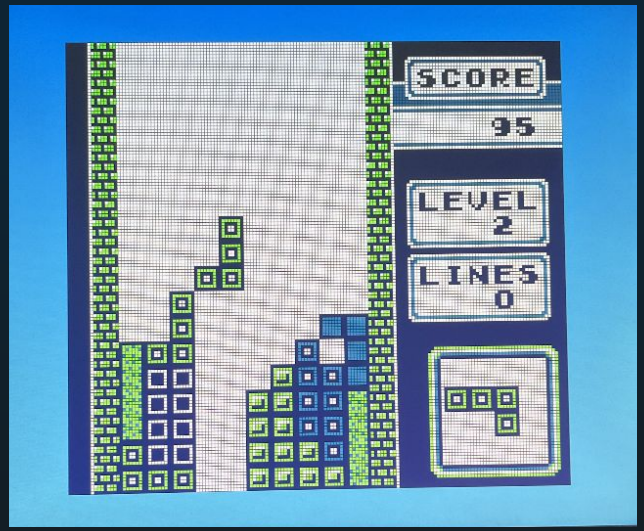


Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags	Opcode Name
✔		hps_0	Arria V/Cyclone V Hard Proce...							
		h2f_user1_clock	Clock Output	Double-click to	hps_0_h2...					
		memory	Conduit	hps_ddr3						
		hps_io	Conduit	hps						
		h2f_reset	Reset Output	Double-click to						
		h2f_axi_clock	Clock Input	Double-click to	Main_PL...					
		h2f_axi_master	AXI Master	Double-click to	[h2f_axi_...					
		f2h_axi_clock	Clock Input	Double-click to	Main_PL...					
		f2h_axi_slave	AXI Slave	Double-click to	[f2h_axi_...					
		h2f_lw_axi_clock	Clock Input	Double-click to	Main_PL...					
		h2f_lw_axi_master	AXI Master	Double-click to	[h2f_lw_a...					
		f2h_irq0	Interrupt Receiver	Double-click to		IRQ 0	IRQ 31			
		f2h_irq1	Interrupt Receiver	Double-click to		IRQ 0	IRQ 31			
✔		GameBoy_VGA	GameBoy_VGA							
		clock	Clock Input	Double-click to	Main_PL...					
		reset	Reset Input	Double-click to	[clock]					
		GameBoy_Pixel	Conduit	Double-click to	[GameBo...					
		VGA	Conduit	vga	[clock]					
		GameBoy_clock	Clock Input	Double-click to	GameB...					
		GameBoy_reset	Reset Input	Double-click to	[clock]					
		avalon_slave	Avalon Memory Mapped Slave	Double-click to	[clock]	0x0400_0000	0x041f_ffff			
		clock_vga	Clock Input	Double-click to	Main_PL...					
✔		GameBoy_Joypad	GameBoy_Joypad							
		clock	Clock Input	Double-click to	Main_PL...					
		reset	Reset Input	Double-click to	[clock]					
		GameBoy_JoyPad	Conduit	Double-click to						
		avalon_slave	Avalon Memory Mapped Slave	Double-click to	[clock]	0x0420_0000	0x0420_0000			
✔		GameBoy	GameBoy							
		clock	Clock Input	Double-click to	GameB...					
		GameBoy_ROM	Conduit	Double-click to						
		GameBoy_Pixel	Conduit	Double-click to	[clock]					
		GameBoy_Joypad	Conduit	Double-click to						
		reset	Reset Input	gameboy_reset	[clock]					
		GameBoy_Audio	Conduit	Double-click to						
✔		GameBoy_Reset	Reset Bridge							
		clk	Clock Input	Double-click to	Main_PL...					
		in_reset	Reset Input	Double-click to	[clk]					
		out_reset	Reset Output	Double-click to	[clk]					
✔		GameBoy_Cartr...	GameBoy_Cartridge							
		clock	Clock Input	Double-click to	Main_PL...					
		reset	Reset Input	Double-click to	[clock]					
		GameBoy_Cartrid...	Conduit	Double-click to						
		avalon_master	Avalon Memory Mapped Master	Double-click to	[clock]					
		reset_gameboy	Reset Output	Double-click to	GameBoy...					
		hps_slave	Avalon Memory Mapped Slave	Double-click to	[clock]	0x0000_0000	0x03ff_ffff			
		clock_gameboy	Clock Output	Double-click to	GameBoy...					
		LEDR	Conduit	ledr						

Open Source Game Play

Confirmed the function of our peripherals and hardware/software interface through integration with open source emulator, shown below

Tetris game play



Dr. Mario game play

Demo!