CSEE4840 Embedded Systems

BameGoy Project Report

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

Columbia University | Spring 2024

Introduction

The original GameBoy was launched in Japan in 1989 (Figure 1). It is a compact video game system for portable hand-held video action involving interchangeable game packs. Since its release, it has sold over 100 million units. The versatile capabilities of the console, and its revolutionary portability, led it to skyrocket into worldwide popular culture and gaming.

Figure 1: GameBoy and cartridge [13]

The portable GameBoy system contains an 8-bit Sharp LR35902 SoC. The main processor is a Z80 running at 4.19 MHz. It is sized at 148 x 90 x 32 mm with a power consumption of 70-80 mA hr, resulting in a playing time of up to 80 hours. The gameplay is observed on an LCD screen with 160 x 144 pixel resolution and a possible 4 shades of gray. The console is also capable of noise, with a loudspeaker and headphone connection. Within the 8-bit data bus and 16-bit address bus, there are up to 64 KB of memory to be addressed. This is composed of the game cartridge space, the RAM (8kB WRAM and 8kB VRAM), I/O (joypad, audio, graphics, and LCD), and interrupt controls.

We aim to use a DE1-SoC development board to emulate this classic GameBoy capable of playing a game through a ROM file uploaded via a User Interface on the monitor. We used a video graphics array (VGA) connector for the picture processing unit (PPU) to translate gameplay to a monitor. In place of GameBoy game cartridges, we utilized online ROM files to upload a game into one-chip RAM. Lastly, we used a NES controller for user inputs to mimic the original controls that were available on the GameBoy.

We used an open-source GameBoy emulator, created by Zheng and Hu [1], to base our system on. We used Zheng and Hu's functional CPU in conjunction with our PPU, hardware/software interface, and peripherals (joypad)

Design and Implementation

We referenced the block diagram from the original GameBoy patent, Figure 2, was referenced for understanding and mapping of electronic components.

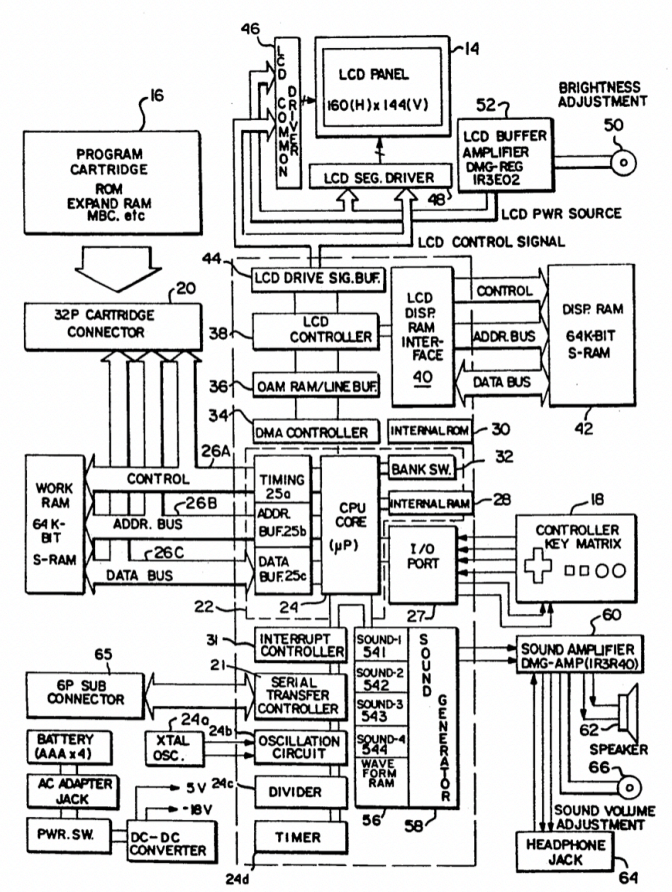

Figure 2 : Block diagram of electronic components from original GameBoy patent [14]

System Overview

The BameGoy system makes use of the Z80 core to write and read from the 16-bit address space. The Game ROM will be loaded into memory via software. The PPU will send addresses and data to the Z80 to write and read from RAM to access VRAM and OAM, as well as the game information for X and Y position of sprites and tiles. The PPU will output 2 bits per pixel, which will be stored in a framebuffer and sent over VGA to the monitor. The controller

interrupts and data will be read via libusb and sent over the Avalon bus via ioctl (input/output control) to be written to $FF00. We will be testing the system with Tetris and Dr. Mario game ROMs, which do not require a Memory Bank Controller to operate. We will not be implementing audio support or serial connection.

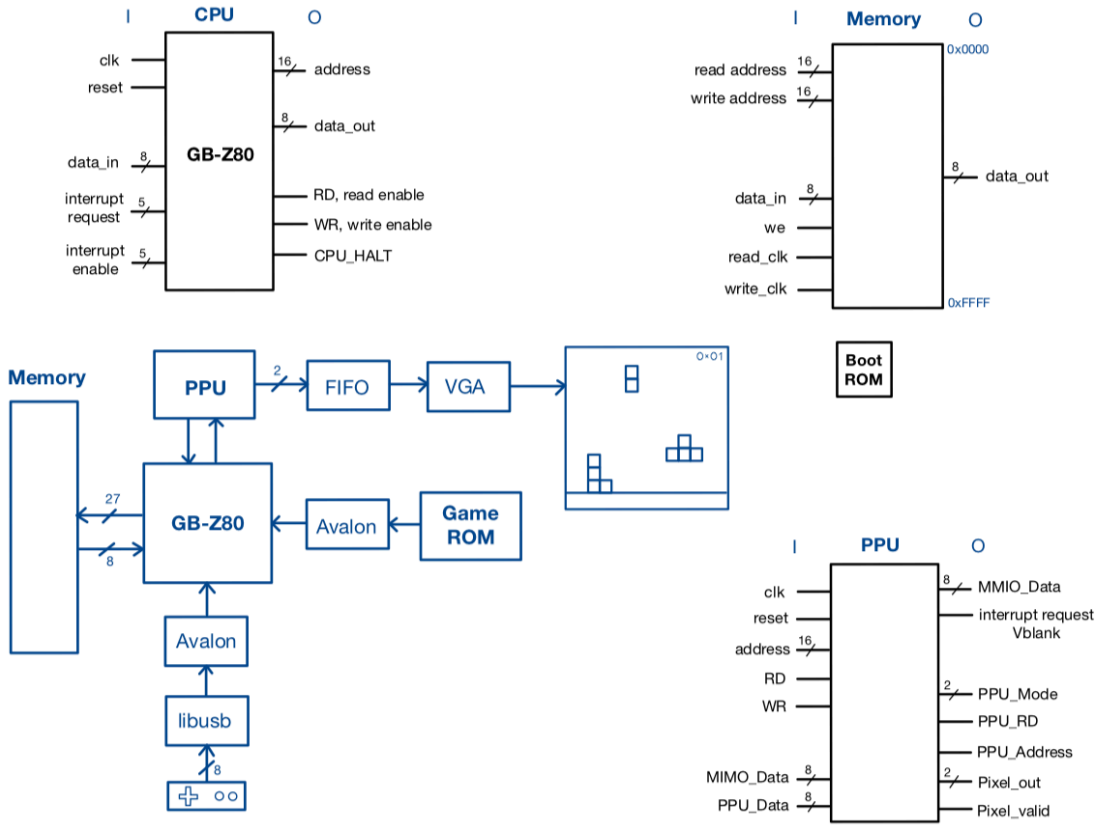

Figure 3 : Top-level diagram of BameGoy system

This will be implemented with SystemVerilog on the FPGA. The basic modules of the BameGoy are the GB-Z80 for the CPU, the Pixel Processing Unit (PPU) to generate graphics, the Game ROM to load in distinct game play cartridges, a game controller for button game play, and a 64 KiB Memory. This is displayed in Figure 3.

The three main data buses on the memory handle work RAM (WRAM), video RAM (VRAM), and OAM. Only one master device can write to a bus at any given time to avoid

conflicts and ensure data integrity. In order to avoid any conflicts, the source code CPU we utilized results in reading 8'hFF if simultaneous read/write occurs, in addition to separate data lines for reading and writing. The overhead functionality of the CPU is shown in Figure 3. The CPU addresses peripherals with Address, Read/Write, and Data requests. Data is prioritized with the MMU, which handles enabling, placing into data buses, simultaneous addressing, and memory access.

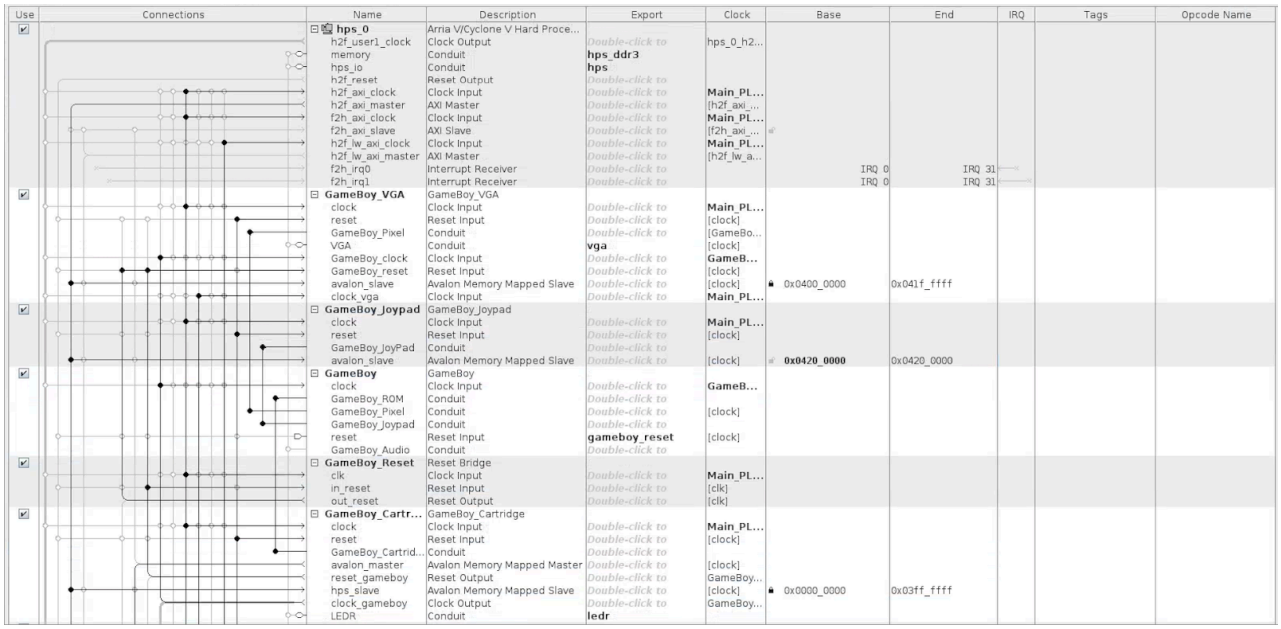

Figure 4 : Qsys file generated from open source emulator

In ensuring we began with a functional open-source CPU, we conducted testing. This included generating a Qsys file and analyzing its interconnects, shown in Figure 4. We discovered an incompatibility with the software version Quartus was running (21.1), versus the one the CPU was designed for (18.1). There was an error with SDRAM in the Qsys file. As a solution, we utilized on-chip RAM instead. Another modification from the original, was the removal of memory bank switching capabilities. The DE1-SoC board has enough memory on

board to permit normal gameplay for any ROM-only cartridge. We also ensured the interface was accurate to the GameBoy with 32 KiB in Cartridge ROM.

Resource Budgets

64 KiB Memory

ROM Bank	16KiB
Switchable ROM	16KiB
VRAM	8KiB
External RAM	8KiB
WRAM	8KiB
OAM	160 bytes
High RAM	127 bytes
IE	1 bytes

Hardware/Software Interface

Byte usage

$0000 – $7FFF : Cartridge ROM	Includes game data and internal work RAM, VRAM	32KiB

IE: Interrupts enabled 5 bits

IF: Interrupts asserted	5 bits
Joypad input	4 bits
SB: Serial Byte	8 bits
SC: Serial Control	8 bits
TAC: Timer Control	3 bits
LCDC: LCD control	8 bits
STAT: LCD status	8 bits
SCY: Background vert. scroll	8 bits
SCX: Background horiz. scroll	8 bits
BGP: Background palette	8 bits
OAM: Object Attribute Memory	8 bits

Hardware Overview

GB-Z80 CPU

The original GameBoy CPU was a Sharp SM83 CPU, however, we used a modified open-source Z80 that is compatible with the GameBoy instruction set. The GameBoy operates as fast as a 4MHz Z80, and all instructions take multiples of 4 cycles to complete. For a list of the GB-Z80 ISA refer to [1].

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)

Figure 5: Moved, removed, and added opcodes [4]

This open-source Z80 reproduced the CPU with a simple 2-stage reduced instruction set computer CPU with a front-end decoder to perform the equivalent operations. The 5 interrupt lines to the CPU are V-Blank interrupt, LCD controller interrupt, timer interrupt, serial interrupt and joypad interrupt. I/O is performed through memory load/store instructions. The sign, parity/overflow flags have been removed, as have the instructions dependent on them. All DD-, ED-, and FD- prefixed instructions, as well as the IX- and IY- registers were removed. Block instructions as well as most 16 bit operations were removed, but auto incrementing HL accesses were added. Edited opcodes are shown in Figure 5.

Pixel Processing Unit

The GameBoy's screen output data is handled almost exclusively by the Pixel Processing Unit, or PPU (Figure 6). It uses a 32x32 tile system featuring two overlays, the screen and the window. The window is 32x32 tiles but the screen which is what actually gets displayed at a given time is 20x18.

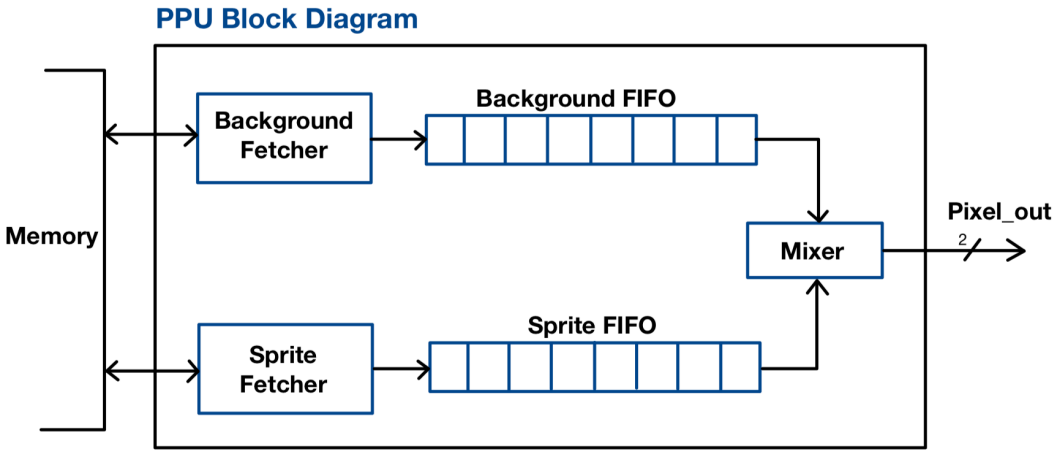

Figure 6: PPU block diagram

Tiles

Each tile is 8x8 pixels making the total resolution of the original GameBoy 160x144. The tiles are stored in memory in registers ($8000-$97FF) as a series of rows, each row being 2 bytes long. The two bytes are merged together and the binary number value of the corresponding index encodes the color, allowing for the ability to use 4 separate colors.

Operation

At a very high level, the PPU operates using scanlines where it draws a row at a time starting at the top left corner until it fills up the screen; the current scanline is stored in the register LY. It takes these pixels from 2 shift registers, the Background and Pixel FIFO. Each register can only hold data about 8 pixels at a time, and are populated using their respective fetching routines. The process runs in the background during the 3 modes of operation of the PPU.

Modes

Mode 2: The first mode the PPU does is the OAM scan. Games hold sprite graphics in tiles, as a sprite becomes relevant to the scene it is placed in the Object Attribute Memory (OAM) ($FE00-$FE9F). These registers can only hold up to 40 sprites, only 10 of which can be drawn at a time. Various information about location, the tile number for graphics and other flags are held here (see OAM Sprite Memory) . During the OAM scan the PPU will traverse a scanline looking for collisions with a sprite in the OAM. If it finds a sprite it marks it and holds it in another special register.

Mode 3: The next mode is for drawing. This is where the PPU runs the fetching steps, pulls from the shift registers and draws output to the screen.

Mode 0: This stage is called the h-blank. It pads the end of lines to use up extra clock cycles and the PPU effectively pauses during blank modes.

Mode 1: Similar to the stage above but this pads the PPU in the vertical direction for 10 pseudo-scanlines.

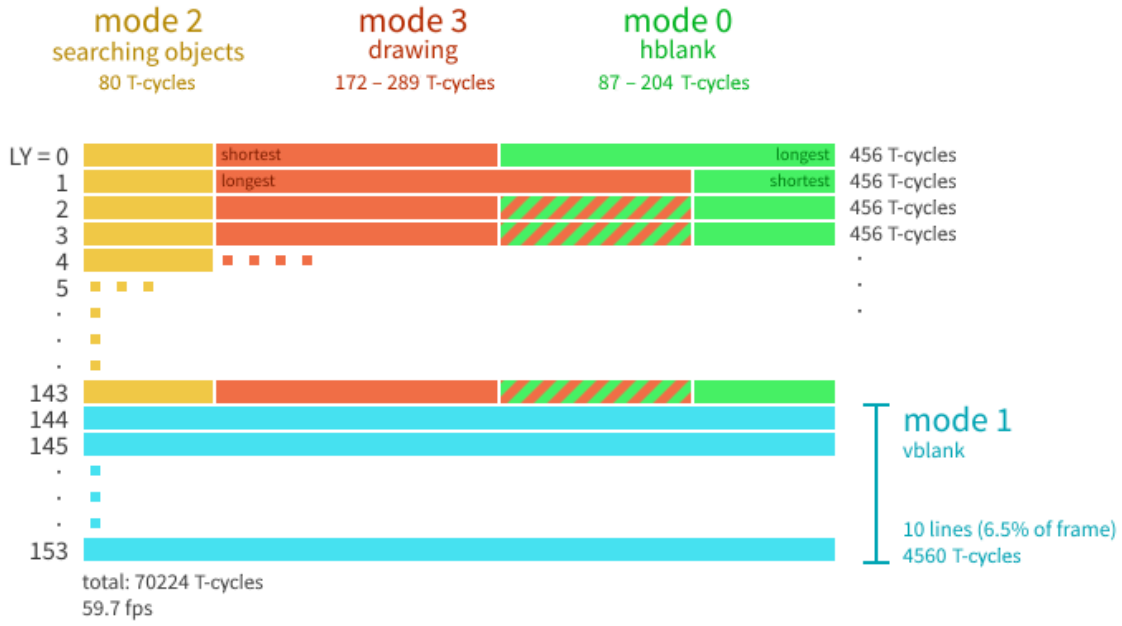

Figure 7: Simplified timing diagram for the PPU [4]

Fetching

As stated before, each FIFO can hold up to 8 pixels at a time, storing information about color, palette, priority and background priority. These are populated as a queue, per scan line. One FIFO is used for information on the sprites and another on the background. The basic

operation for populating each FIFO is the fetch: The PPU will find the corresponding tile number for the graphic from the map data or sprite data, grab the tile data and put it into the FIFO. In order to set which tiles should be displayed in the Background/Window grids, background maps are used. The VRAM sections $9800-$9BFF and $9C00-$9FFF each contain one of these background maps. A background map consists of 32x32 bytes representing tile numbers organized row by row.

This process consists of 4 steps each consisting of 2 cycles: fetching tile number, fetch first byte of tile data, fetch second byte of tile data, and finally push to FIFO. The background fifo can only be written to if it is empty. The tile number is fetched from VRAM or OAM, depending on whether it is background or a sprite, the address of which is taken from the current position on the screen. This timing can also be visualized in Figure 7.

Pixel Mixing

During every cycle, the PPU attempts to push to the LCD. The general approach to PPU operation is: if there is a pixel in the background FIFO, it will push to the LCD; If not, it will wait. If both FIFOs have pixels in them when rendering data, both will be shifted out. The background pixel will be output to the screen if the sprite color is 0 or if the background to object (bg-to-obj) priority is high and the background pixel color is not equal to 0. Otherwise, the sprite pixel is pushed. The SCX register is defined for background scrolling. The first SCX mod 8 pixels at the start of a scanline will be discarded, extending PPU mode 3 by SCX mod 8 cycles. This is handled through the board's memory bank in Figure 8.

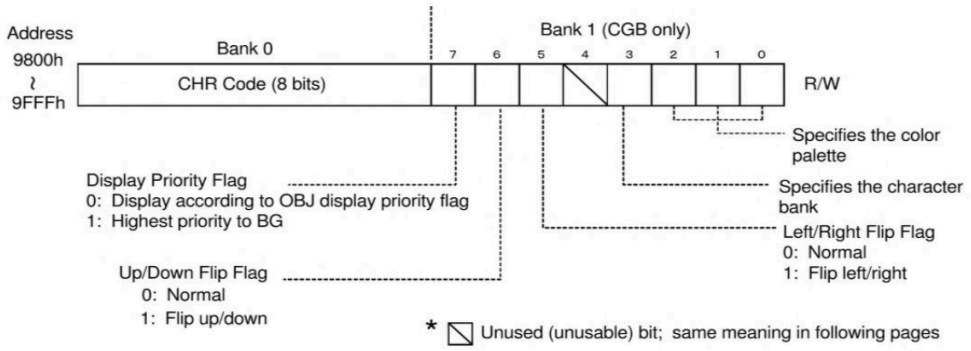

Figure 8: DE1-SoC Board memory bank [2]

Data for 32 x 32 character codes (256 x 256 dots) can be specified from 9800h or 9C00h as background (BG) display data. Of these, data for 20 x 18 character codes (160 x 144 pixels) are displayed to the original GameBoy LCD screen, Figure 9. This data is sent to a FIFO framebuffer on the VGA module. which has a resolution of 1280 x 1024 pixels, for an active LCD viewport of 960 x 864 pixels centered on the monitor. For this, we modified the VGA counters module used in Lab 3. For further technical data, refer to Technical Data portion.

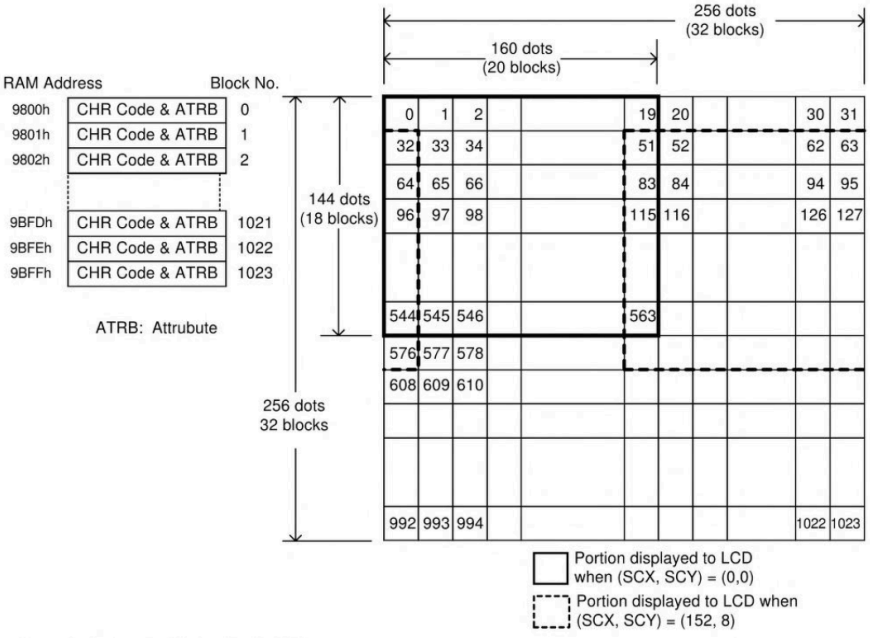

Figure 9: Video Output [2]

Timing

Timing is crucial in building the PPU in order to ensure proper synchronization with the CPU, frame rate and refresh rate, memory access, and rendering pipelines. The figures below show the timing diagrams used to build, debug, and iterate the PPU. Figure 10 shows the clock cycles of the OAM Search function, which is responsible for determining which sprites to display during each scanline. The ready_load timing in the figure indicates when the PPU is ready to load in more pixels.

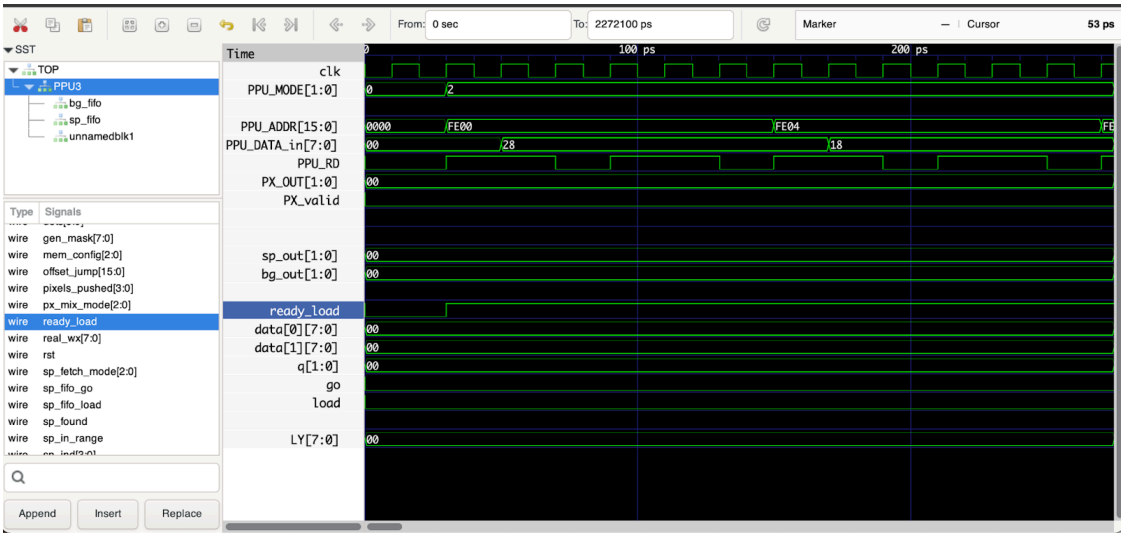

Figure 10: OAM Search

Proper timing ensures that the PPU can access and process data without causing issues with CPU instructions and memory. An example diagram of PPU functioning memory access is shown below.

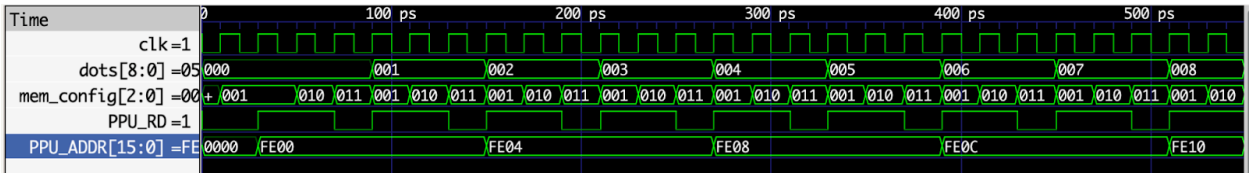

Figure 11: Example timing diagram of memory mechanism

In Figure 11, it is important to note that each memory request takes 3 clock cycles. This differs from the actual GameBoy design, where each request is processed in just one cycle. Due to the PPU operating at a slower clock rate compared to the memory, the PPU can issue a request and see the results on the following PPU cycle. For testing purposes, we introduced a stalling mechanism to simulate this behavior. The “dots [8:0]” represent instructions processed by the PPU, for the sake of testing. The system stalls during testing but not deployment. The “dot” then increments and moves onto the next instruction. This mechanism allows the system to mimic the delay caused by memory access using output signals from the PPU itself, ensuring that information is processed with a simulated delay.

Lastly, Figure 12 shows a comprehensive diagram of the final PPU clock cycles. The diagram illustrates how the PPU modes and memory address handle loading data and outputting pixels to video. PPU read occurs when a new address is made available. This allows for proper rendering and memory access of graphical data. Additionally, the clock cycles are separated by Modes and consist of waiting periods in between for memory addressing and synchronicity purposes.

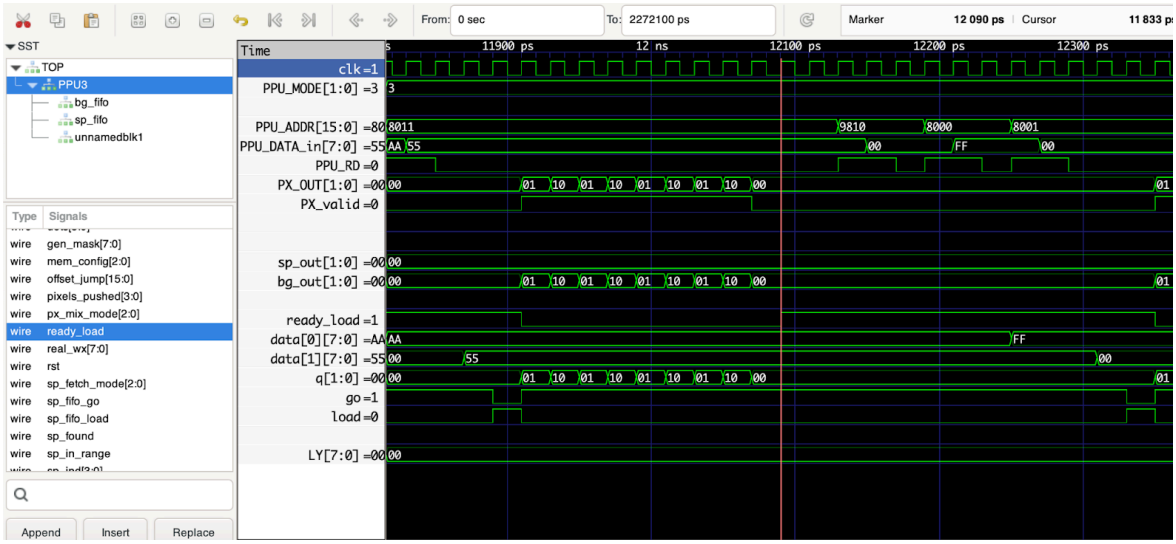

Figure 12: Final PPU timing diagram

Testing

In the case of CPU and PPU requesting memory at the same time, there may be an issue with priority. However, the FPGA with dual-port memory allows for simultaneous access from multiple components such as a CPU and a PPU. Each port has its own independent access path to the memory array. Therefore, we were able to disregard priority.

With regards to pipelining in loading sprites, we follow the process of fetching, processing, and rendering. The fetching, as described previously, initiates the request from memory to fetch sprite data. The processing is where the sprites are being pushed through the Pixel FIFO in Figure 13. Rendering then refers to the last step of outputting the pixel and background to the LCD. While the FIFO can process through multiple sprites before reaching the mixer, the fetching will occur based on priority. This is relevant with multiple sprites on top of each other, which will rely on accurate timing and system order. Whichever sprite is last in the OAM memory will be prioritized and drawn on top. This is because the sprites are processed from left to right.

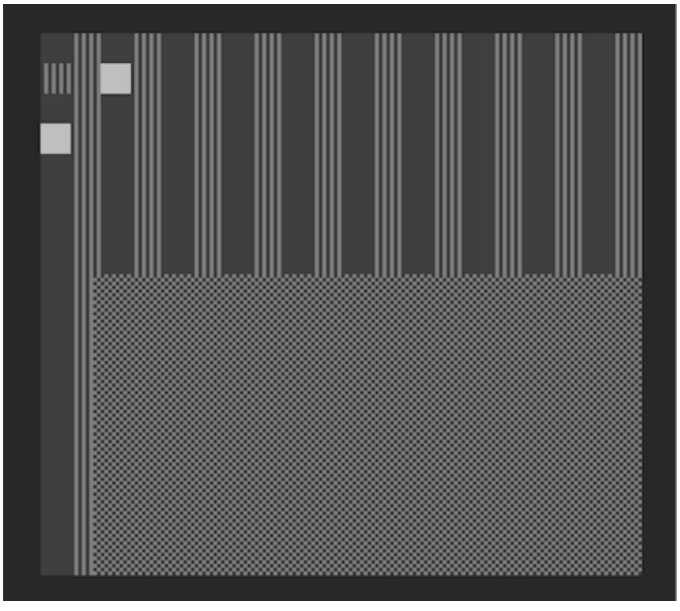

Figure 13: PPU Testbench Output

To test the PPU we incrementally set up verilator testbenches. The testbenches instructed the PPU to draw a tile, then access a tile map, execute sprites, handle background scrolling and window coordinates (WX, WY, SCX, SCY), and implement LCDC. Figure 13 shows a successful output of the testbench. The functional PPU can implement sprites, handle the LCDC flags, create the window, operate multiple BG maps, and control background scrolling, all shown on the testbench output.

In order to coordinate accessing memory for the pixels, we had to implement a stalling mechanism. This would take multiple clock cycles to serve as a "wait" because memory access takes multiple clock cycles. This caused issues as it was addressed after the development of other PPU functions, in which we had to reorganize the clock cycles of Modes in order to achieve full function. Additionally, the shift registers required an asynchronous mechanism as they filled and waited on different cycles.

Video

The GameBoy has a resolution of 160 x 144 pixels, a 10:9 aspect ratio. The VGA has a resolution of 1280 x 1024 @ 60 Hz. We opted to only use a portion of the VGA screen in order to maintain the original aspect ratio. Therefore, we established an active region of 160 to 1120 pixels horizontally and 80 to 944 pixels vertically. The resulting resolution of the active region was 960 x 864 pixels, see Figure 14. The GameBoy image was therefore scaled by 6 to fit the VGA output, with a colored background around the game screen.

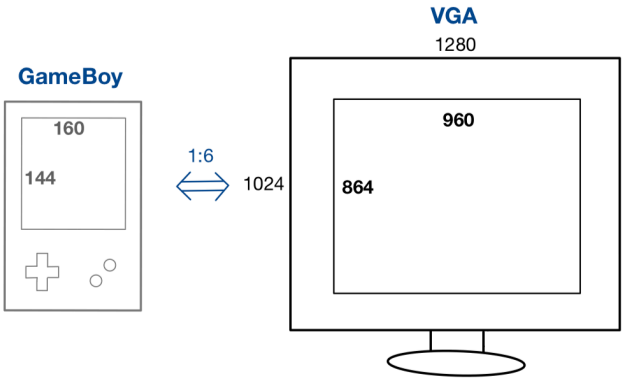

Figure 14: GameBoy to VGA resolution scaling

To display game play, the GameBoy writes at its own clock speed and the VGA reads pixels out at its own clock speed. GameBoy pixel data is written into the framebuffer using its 4MHz clock. The VGA reads the framebuffer using its pixel frequency of 108 MHz to allow for one pixel every cycle (Figure 15) . Address counters and scaling logic ensure the pixels are mapped correctly to the VGA's resolution. This maintains synchronization without noticeable jitter. By using dual-port RAM and the counters, the video output can be displayed correctly across the two different clock domains.

General timing

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

Figure 15: VESA Signal 1280 x 1024 @ 60 Hz timing [11]

The scaling of the pixels is performed by selectively reading from the framebuffer. Pixels outside the specified X and Y window ranges are not read from the framebuffer, effectively blanking those areas on the display. The framebuffer allows for easy implementation of scaling by manipulating the addresses of X and Y, specifically horizontal scaling by 6 and vertical by 6.

Scaling calculations shown below:

Horizontal Scaling:

GameBoy width: 160 pixels

VGA display width: 960 pixels

Scaling ratio (horizontal): 960 / 160 = **6**

Vertical Scaling:

GameBoy height: 144 pixels

VGA display height: 864 pixels

Scaling ratio (vertical): 864 / 144 = **6**

As an additional aesthetic feature, we also altered the color palette of the background to be Pantone 292 using a clock at 67.1 MHz, which is the clock that runs the SOC_System reset and on chip RAM. This outputs game play like in Figure 16.

Figure 16: VGA Screen of Tetris Home Screen

Joypad

Figure 17 : USB Wired Controller [13]

The controller is a USB Wired Controller, Figure 17 . The interrupts and data is read via libusb. The data is then sent over the Avalon bus via ioctl to be written to $FF00, which is the memory mapped I/O for joypad input. The joypad ioctl is an 8 bit ioctl write. The bit order is shown in the following table.

Start	Sel	B	A	Down	Up	Left	Right

Left is the most significant bit and right is the least. Each bit corresponds to a button pressed and also allows for multiple button presses.

The original GameBoy joypad has 8 keys in a 2x4 matrix, Figure 18. The CPU scans through the outputs P14-15 by pulling low. Each time this occurs, the input pins P10-13 are sampled. These input pins are connected to a voltage through a pull up resistor, resulting in a default high state. The combination of the input and output pin states determine if the CPU registers a button press. Since the controller we are using is an NES controller, it is compatible with this GameBoy instruction for joypad functionality.

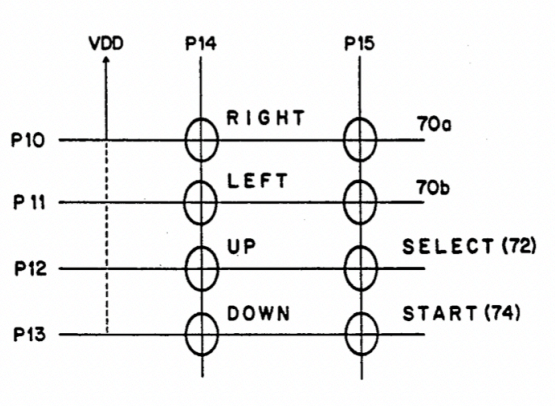

Figure 18: Key-matrix for detecting joypad input [15]

Cartridge

With real GameBoy cartridges, Memory Bank Controllers (MBCs) and additional RAM were utilized to manage larger game sizes, allowing for games beyond the initial 32 kiB of ROM. However, since we chose not to implement later generation cartridges that include MBCs, our system is limited to loading up to 32 kiB of ROM. We used the test ROM cartridge for Tetris [8] and Dr. Mario for gameplay. The ROM, Read Only Memory, file is a digital copy of a game's software. We obtained the ROM files used from online documentation. The system is also compatible with a variety of other cartridges. They are listed in entirety in the CartridgeList Github [12], working with the cartridges listed with the ROM-only type. The cartridge headers are located at $0100 – $014F. With the game cartridge loaded, the VGA screen displays the start screen of the game, like in Figure 19.

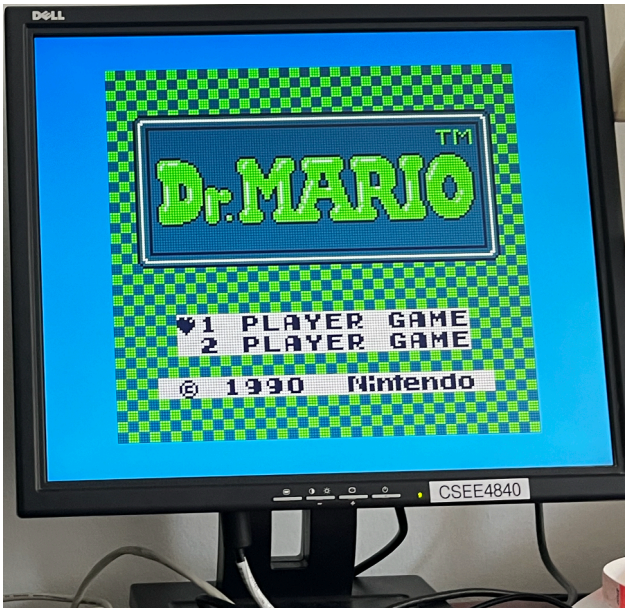

Figure 19: Dr. Mario start screen

Software Overview

We use the software to read in controller input from our USB NES controller. On an interrupt, the controllers send a singular byte mapping each of its 8 possible inputs to a bit. When we receive this, we will send an interrupt flag to the BameGoy's CPU and update the joypad register ($FF00) with the appropriate values.

The general overview of the software will function in the following rough procedure:

1. Load in USB NES controller
2. Load in ROM game cartridge to On-Chip RAM with a UNIX *mmap*
3. Wait for controller inputs
4. Take controller inputs

In conjunction with the hardware/software interface, this procedure will output game play onto the VGA screen. This is shown in Figure 20. Gameplay function can be referenced in the

ReadMe. Importantly, exiting game play is achieved with “Ctrl + C”. Additionally, we included support for double speed with command line arguments.

Figure 20 : Dr. Mario game play screen

Technical Data

CPU: GB-Z80

The CPU registers can be accessed as a 16 bit register or as two 8 bit registers.

Registers:

16-bit	Hi	Lo	Name
AF	A	Flags	Accumulator & Flags
BC	B	C	BC
DE	D	E	DE
H	H	L	HL
SP	-	-	Stack Pointer
PC	-	-	Program Counter

Flags Register: RAM: 8kiB (DMG), 32 kiB (CGB)

Bit	Name	Description
7	z	Zero flag
6	n	Subtraction flag (BCD)
5	h	Half Carry flag (BCD)
4	c	Carry flag

Video

- 160 x 144 pixels (20 x 18 tiles) → **960 x 864** active region on VGA
- 2 Backgrounds: Main and Window (for scrolling)
- **Max sprites: 40 → max 10 per scanline**
- 8×8 pixel tiles
- 456 cycles per line
- 70224 cycles per frame
- VDraw: 144 lines, 65664 cycles
- VBlank: 10 lines, 4560 cycles
- Clock Speed: 4.194304 MHz (2^22 Hz)
- Horizontal Sync: 9198 KHz
- Vertical Sync: 59.73 Hz

Pixel Data

- 2BPP format: 2 bits per pixel (2 bits to store color data)
- 2 bytes → 8 pixels
- (Each bit of first byte) + (bit in same position on second byte) => calculates color number, Figure 21

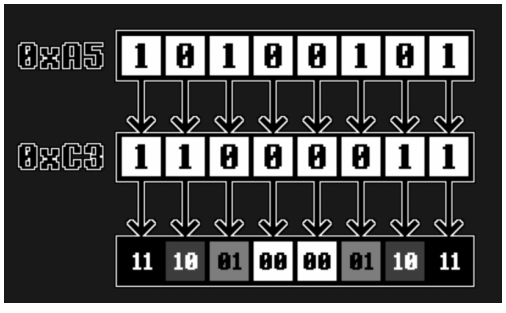

Figure 21: Calculating color number [4]

Tile Data

- Graphics encoded in 2BPP format, stored in VRAM $8000-$97FF
- "Tile Numbers" 16-byte-block in section of VRAM
- Addressing methods: 8000 and 8800
- 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 (TILE_NUMBER * 16)

Cartridge Header

Cartridge headers located at **$0100 – $014F:**

Entry point	**$0100 – $0103**
Nintendo logo	**$0104 – $0133**
Title	**$0134 – $0143**
Manufacturer code	**$013F – $0142**
CGB	**$0143**

- Enable color mode or monochrome compatibility mode
- $80: game supports CGB enhancements, backwards compatible with monochrome

New licensee code	**$0144 – $0145**

- Only meaningful if $33 (which our game will be)

SGB flag **$0146**

- Ignored if byte set to value other than $03

Cartridge type **$0147**

- Byte indicates the kind of hardware present on the cartridge - esp. Its mapper

ROM size **$0148**

- Typically given by 32 KiB × (1 << <value>)

RAM size **$0149**

- If cartridge type doesn't include “RAM” in name, set to 0

Destination code **$014A**

- $00: Japan
- $01: Overseas only

Old licensee code **$014B**

- Not consider – our new licensee code $33 will override

Mask ROM version num **$014C**

Header checksum **$014D**

- If byte at $014D doesn’t match the lower 8 bits of checksum, boot ROM will lock and the program in the cartridge won’t run

Global checksum **$014E –$014F**

Memory Map

(64 KiB resource budget)

IE register, interrupt enabler	**$FFFF**
External bus (2x 16 KiB ROM regions)	**$0000 – $7FFF**
VRAM (8 KiB)	**$8000 – $9FFF**
External bus (8 KiB RAM region)	**$A000 – $BFFF**
WRAM (4 KiB Work RAM)	**$C000 – $DFFF**
ECHO (WRAM secondary mapping, **not emulated**)	**$E000 – $FDFF**
Object Attribute Memory (OAM), Sprite use	**$FE00 – $FE9F**
High RAM (HRAM)	**$FF80 – $FFFE**
Memory mapped I/O	**$FF00 – $FF7F**
Joypad input	$FF00
SB: Serial Byte	**$FF01**
SC: Serial Control	**$FF02**
DIV: Clock Divider	**$FF04**
TIMA: Timer Value	**$FF05**
TMA: Timer Reload	**$FF06**
TAC: Timer Control	**$FF07**
IF: Interrupts asserted	**$FF0F**
Audio channels - unused for this project	**$FF10 – $FF26**
LCDC: LCD control	**$FF40**
STAT: LCD status	**$FF41**
SCY: Background vert. scroll	**$FF42**

SCX: Background horiz. scroll	**$FF43**
LY: LCD Y coordinate	**$FF44**
LYC: LCD Y compare	**$FF45**
DMA: OAM DMA source address	**$FF46**
BGP: Background palette	**$FF47**
OBP0: OBJ palette 0	**$FF48**
OBP1: OBJ palette 1	**$FF49**
WY: Window Y coord	**$FF4A**
WX: Window X coord	**$FF4B**
Boot ROM control	**$FF50**

Mapping Key

- 1: Unmapped (hi-Z), always reads 1
- I: Input only (readable by CPU)
- O: Output only (writable by CPU)
- B: Bidirectional (read/write by CPU)

Memory Map - Extended Descriptions

IE: Interrupt Enabler $FFFF

0: disable

1: enable

Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Transition from high to low of Pin #P10-P13	Serial I/O transfer complete	Timer overflow	LCDC	V-Blank

Joypad input $FF00

Keys pulled low (to 0) when press and high (to 1) when unpressed

- If bits 4 and 5 both low, then combination of D-pad and face buttons ANDed
- If neither bit 4 or 5 is selected, then all keys read 1

Bit	**4 low**	**5 low**
0	Right	A
1	Left	B
2	Up	Select
3	Down	Select

SB: Serial Byte $FF01

Serial transfer data (R/W)

8 Bits of data to be read/written

SC: Serial Control $FF02

SIO Control (RW)

Bit 7 - Transfer Start Flag

0: Non transfer

1: Start transfer

Bit 0 - Shift Clock

0: External Clock (500KHz Max.)

1: Internal Clock (8192Hz)

TAC: Timer Control $FF07

Timer control (R/W)

Bit 2 - Timer Stop

0: Stop Timer

1: Start Timer

Bits 1+0 - Input Clock Select

0: 4.096 KHz (~4.194 KHz SGB)

1: 262.144 KHz (~268.4 KHz SGB)

2: 65.536 KHz (~67.11 KHz SGB)

3: 16.384 KHz (~16.78 KHz SGB)

IF: Interrupts asserted $FF0F

Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Transition from	Serial I/O	Timer overflow	LCDC	V-Blank

high to low of Pin #P10-P13	transfer complete			

LCDC: LCD Control Register $FF40

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LCD Display Enable	Window Tile Map Select	Window Display Enable	Tile Data Select	BG Tile Map Select	Sprite Size	Sprite Enable	BG/Window Enable

Bit 7 : Enables screens - if set to 0, disables PPU completely

0: Stop completely (no picture on screen)

1: Operation

Bit 6: Uses $9800-$9BFF - if set to 1, sets Window to use background map at $9C00-$9FFF

0: $9800-$9BFF

1: $9C00-$9FFF

Bit 5: Enables display - if set to 0, hides window layer completely

0: off

1: on

Bit 4: Uses 8800 method for tile select - if set to 1, fetches Tile Data using 8000 method

0: $8800-$97FF

1: $8000-$8FFF <- Same area as OBJ

Bit 3: Uses $9800-$9BFF - if set to 1, Background will use background map at $9C00-$9FFF

0: $9800-$9BFF

1: $9C00-$9FFF

Bit 2: 1x1 Tile Sprite size - if set to 1, sprites displayed as 1x2 Tiles (8x16 pixel)

0: 8*8

1: 8*16 (width*height)

Bit 1: Sprites only drawn if set to 1

0: off

1: on

Bit 0: Sprites unaffected - if set to 0, neither Background nor Window tiles are drawn

0: off

1: on

STAT: LCD Status Register $FF41

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused (Always 1)	LYC=LY STAT Interrupt Enable	Mode 2 STAT Interrupt Enable	Mode 1 STAT Interrupt Enable	Mode 0 STAT Interrupt Enable	Coincidence Flag	PPU Mode	PPU Mode

Bit 7 : Unused

Bit 6: If set to 1, enable “LYC=LY condition” to trigger STAT interrupt

Bit 5-3: “Condition enablers” for STAT interrupt - trigger when PPU enters specific mode

Bit PPU Mode

STAT.5 Mode 2 (OAM Scan)

STAT.4 Mode 1 (VBlank)

STAT.3 Mode 0 (HBlank)

Bit 2: Set by PPU if value of LY register = LYC register

0: LYC ≠ LCDC LY

1: LYC = LCDC LY

Bit 1-0: Set by PPU depending on mode

0 : H-Blank

1 : V-Blank

2 : OAM Scan

3 : Drawing, Transferring Data to LCD Driver

SCX: Background horiz. scroll **$FF43**

8 bit value $00-$FF to scroll BG X screen position

SCY: Background vert. scroll **$FF42**

8 bit value $00-$FF to scroll BG Y screen position

BGP: Background palette **$FF47**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Data for Dot Data 11 (~ darkest color)		Data for Dot Data 10		Data for Dot data 01		Data for Dot Data 00 (~ lightest color)	

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

Results

Our main goal for this project was to create a custom functioning PPU to run the Tetris GameBoy ROM. We developed a PPU with accurate memory to pixel output. Our testbenches were inconsistent with proper operation of the game, causing the PPU to fail at switching background maps. However, we successfully loaded the title screen of a game. Additionally, with an open source PPU, we were able to run a fully functional game and used it to test the rest of our system.

We utilized an open source emulator to base our systems on [1]. We used a functional CPU in conjunction with our PPU, hardware/software interface, and peripherals (joypad). PPU was independently tested with testbenches and performed successfully in simulation. However, issues occurred with integration into the larger emulator. We confirmed issues were specific to our PPU by successfully running open source game play. With our custom PPU we were able to load the GameBoy boot screen, Figure 22, and the Dr. Mario start screen, Figure 23. We opted to use the Dr. Mario start screen for testing, as opposed to Tetris, because it had a more complex initial title screen.

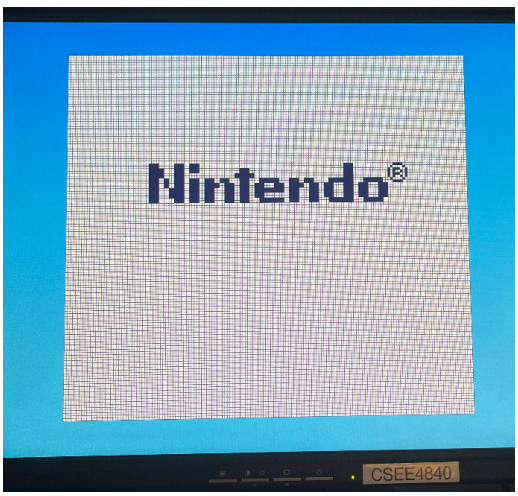

Figure 22 (left): GameBoy Boot Screen

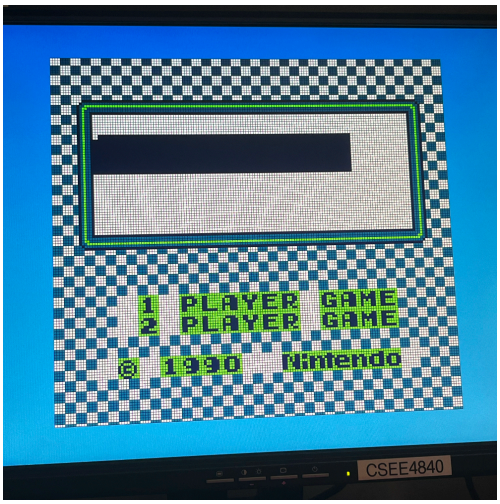

Figure 23 (right): Dr. Mario Start Screen

Contributions

Donovon Sproule	PPU (modes, OAM scan, scrolling, etc.), Testbenches
Nicolas Alarcon	System Integration, Cartridges, Joypad, Software, VGA
Claire Cizdziel	Hardware/Software Interface, VGA, Project Report, Presentation Slides

Through extensive research, we learned the intricacies of developing a hardware emulator, focusing on the pixel processing unit. Key lessons included the necessity of intensive testing and iterative testbench development to ensure proper PPU functionality and smooth interfacing with memory and CPU components. Towards the end of our project, remote work, necessitated by campus accessibility issues, highlighted the critical need for effective communication and collaboration tools. For future projects, a solid research base and comprehensive planning are essential, along with an expectation of roadblocks and a focus on essential project aspects to adapt to those challenges.

References

[1] Zheng and Hu, *CSEE4840 Embedded Systems GameBoy Project Report 2019*

https://www.cs.columbia.edu/~sedwards/classes/2019/4840-spring/reports/GameBoy.pdf

[2] Nintendo, *Video output diagram - Screen (p.57)*

https://archive.org/details/GameBoyProgManVer1.1/page/n53/mode/2up

[3] Terasic Inc., *Video output diagram - Banks*

https://www.terasic.com.tw/cgi-bin/page/archive.pl?

[4] *PPU: Gameboy Emulator Development Guide*

https://hacktix.github.io/GBEDG/ppu/#an-introduction

[5] *CPU Comparison with Z80*

https://gbdev.io/pandocs/CPU_Comparison_with_Z80.html

[6] *Z80 CPU User Manual*

https://www.zilog.com/docs/z80/um0080.pdf

[7] *GB- Z80 Instruction Set*

https://rgbds.gbdev.io/docs/v0.7.0/gbz80.7

[8] *Gameboy Hardware Test ROMs*

https://gbdev.gg8.se/files/roms/blargg-gb-tests/

[9] Nintendo, *GameBoy Technical Data*

https://www.nintendo.com/en-gb/Support/Game-Boy-Pocket-Color/Product-information

[10] Intel FPGA, *DE1-SoC Computer System On-Chip Memory*

https://ftp.intel.com/Public/Pub/fpgaup/pub/Intel_Material/18.1/Computer_Systems

[11] *VGA Timing*

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

[12] Avivace, *Compatible Cartridges*

https://github.com/gbdev/awesome-gbdev/blob/master/CartridgeList.csv

[13] Figure 1, *Gameboy and cartridge*

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

[14] Figure 17, *NES Controller*

https://www.amazon.com/Controller-suily-Joystick-RetroPie-Emulators

[15] Figures 2 & 18, *GameBoy US Patent*

https://patents.google.com/patent/US5184830A/en

Code

Makefile

```
Unset
	ifneq (${KERNELRELEASE},)


	# KERNELRELEASE defined: we are being compiled as part of the
Kernel
		obj-m := game_boy.o


	else


	# We are being compiled as a module: use the Kernel build system


		KERNEL_SOURCE := /usr/src/linux-headers-$(shell uname -r)
		  PWD := $(shell pwd)


	CFLAGS = -Wall
	CC = gcc
	OBJECTS = start.o controller.o


	default: module start


	start: $(OBJECTS)
		$(CC) $(CFLAGS) -o start $(OBJECTS) -lusb-1.0
```

```
	module:
		${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} modules

	clean:
		${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} clean
		${RM} start
		rm -rf *.o start

	TARFILES = Makefile README controller.c controller.h game_boy.h
game_boy.c start.c
	TARFILE = final-sw.tar.gz
	.PHONY : tar
	tar : $(TARFILE)

	$(TARFILE) : $(TARFILES)
		tar zcfC $(TARFILE) .. $(TARFILES:%=Project/%)

	endif
```

controller.c

```
Unset
      #include "controller.h"


      #include <stdio.h>

      #include <stdlib.h>

      #include <unistd.h>


      struct libusb_device_handle *opencontroller(void)

      {

            libusb_device_handle *controller;

          int error;


          // Initialize libusb

            if ((error = libusb_init(NULL)) < 0) {

              fprintf(stderr, "Error initializing libusb: %s\n",
libusb_error_name(error));

              return NULL;

          }


          // Open device with the given vendor ID and product ID

            if ((controller = libusb_open_device_with_vid_pid(NULL,
VENDOR_ID, PROD_ID)) == NULL) {

              fprintf(stderr, "Device not found.\n");

              return NULL;
```

```
        }

        // There's only one interface so we know we can use 0
        if (libusb_kernel_driver_active(controller, 0))
                libusb_detach_kernel_driver(controller, 0);

          libusb_set_auto_detach_kernel_driver(controller, 0);
        if ((error = libusb_claim_interface(controller, 0)) !=
LIBUSB_SUCCESS) {
            fprintf(stderr, "Error claiming interface: %s\n",
libusb_error_name(error));
            libusb_close(controller);
            return NULL;
        }

        return controller;
    }

    void read_inputs(void)
    {
          struct libusb_device_handle *controller;
          int transferred;
          unsigned char packet[8];
```

```
            unsigned char pressed, last_pressed;            //
DOWN_UP_LEFT_RIGHT_START_SELECT_B_A


            if ((controller = opencontroller()) == NULL) {
                fprintf(stderr, "Did not find a controller\n");
                exit(1);
              }


            pressed = 0;
            printf("Ready to read input\n");
            while (1) {
                          libusb_interrupt_transfer(controller,
ENDPT_ADDR_IN,
                 (unsigned char *)& packet, sizeof(packet),
                 &transferred, 500);



              if (transferred > 0 && !EMPTY_INTERR(packet)) {


                          /* --- Start/Select --- */
                          PROCESS(packet[6], SELECT, pressed);
                          PROCESS(packet[6], START, pressed);
```

```
                        /* --- A/B --- */
                        PROCESS(packet[5], A, pressed);
                        PROCESS(packet[5], B, pressed);

                        /* --- D_Pad --- */
                        PROCESS(packet[4], DOWN, pressed);
                        PROCESS(packet[3], RIGHT, pressed);
                        EMPTY_PROCESS(packet[4], UP, pressed);
                        EMPTY_PROCESS(packet[3], LEFT, pressed);

                // bytes are swapped, and not recognizing a header
file change
                pressed = ((pressed & 0x0F) << 4) | ((pressed & 0xF0)
>> 4);
                        /* Effectively debounces by introducing a delay
after input was recieved */
                        usleep(100 * 1000);
                } else if (transferred > 0){
                pressed = 0x00;
                }
            if (last_pressed != pressed) printf("Pressed: 0x%X\n",
pressed); //send to reg here
            last_pressed = pressed;
```

```
        }


        libusb_close(controller);
    }
```

controller.h

```
Unset
    #include <libusb-1.0/libusb.h>


    #define VENDOR_ID       0x0079

    #define PROD_ID              0x0011

    #define ENDPT_ADDR_IN   0x81


    #define EMPTY_INTERR(packet)    \
                      ((packet[3] == 127) & \
                    (packet[4] == 127) &  \
                    (packet[5] == 15)  &  \
                    (packet[6] == 0)   &  \
                      (packet[7] == 0))


    #define PROCESS(PACKET_BYTE, BIT, PRESSED_BYTE) \
```

```
                        if (PACKET_BYTE & BIT##_DET)  \
                              PRESSED_BYTE |= BIT;  \
                        else                    \
                              PRESSED_BYTE &= ~BIT


      #define EMPTY_PROCESS(PACKET_BYTE, BIT, PRESSED_BYTE) \
                        if (!(PACKET_BYTE & BIT##_DET))     \
                              PRESSED_BYTE |= BIT;  \
                        else                    \
                              PRESSED_BYTE &= ~BIT


      enum detect_bits { SELECT_DET=0x10,
                                 START_DET=0x20,
                                 A_DET=0x20,
                                 B_DET=0x40,
                                 DOWN_DET=0x80,
                                 RIGHT_DET=0x80,
                                 UP_DET=0xFF,
                                 LEFT_DET=0xFF };


      //    DOWN_UP_LEFT_RIGHT_START_SELECT_B_A
      enum press_bits {A=0x1, B=0x2, START=0x8, SELECT=0x4, LEFT=0x20,
RIGHT=0x10, UP=0x40, DOWN=0x80};
```

```
struct libusb_device_handle *opencontroller(void);

void read_inputs(void);
```

game_boy.c

Unset

```
/* * Device driver for the Game Boy joypad
 *
 * A Platform device implemented using the misc subsystem
 * Original By: Justin Hu
 * Modified By: Nicolas Alarcon
 *
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 *               drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 *
 * "make" to build
 * insmod game_boy.ko
 *
```

```
 */

#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "game_boy.h"

#define DRIVER_NAME "game_boy"

/* Joypad Register */
#define JOYPAD_REG(x)(x)

/*
```

```
       * Information about our device
       */
      struct game_boy_dev {
          struct resource res; /* Resource: our registers */
          void __iomem* virtbase; /* Where registers can be accessed in
memory */
          uint8_t joypad_status;  // current joypad status

      } dev;

      static void write_joypad_register(uint8_t * reg)
      {
          iowrite8(*reg, JOYPAD_REG(dev.virtbase));
          dev.joypad_status = *reg;
      }

      /*
       * Handle ioctl() calls from userspace:
       * Read or write the segments on single digits.
       * Note extensive error checking of arguments
       */
      static long game_boy_ioctl(struct file* f, unsigned int cmd,
unsigned long arg)
```

```
{
    uint8_t joypad_reg;

    switch (cmd) {
    case GAME_BOY_WRITE_JOYPAD:
        if (copy_from_user(&joypad_reg, (uint8_t*)arg,
            sizeof(uint8_t)))
            return -EACCES;
        write_joypad_register(&joypad_reg);
        break;
    default:
        return -EINVAL;
    }

    return 0;
}

/* The operations our device knows how to do */
static const struct file_operations game_boy_fops = {
    .owner = THIS_MODULE,
    .unlocked_ioctl = game_boy_ioctl,
};
```

```
      /* Information about our device for the "misc" framework -- like a
char dev */
      static struct miscdevice game_boy_misc_device = {
          .minor      = MISC_DYNAMIC_MINOR,
          .name       = DRIVER_NAME,
          .fops       = &game_boy_fops,
      };



      /*
       * Initialization code: get resources (registers) and display
       * a welcome message
       */
      static int __init game_boy_probe(struct platform_device* pdev)
      {
          uint8_t joypad_init = 0x00; // initialize joypad
          int ret;

          /* Register ourselves as a misc device: creates /dev/game_boy
*/
          ret = misc_register(&game_boy_misc_device);

          /* Get the address of our registers from the device tree */
```

```
        ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
        if (ret) {
            ret = -ENOENT;
            goto out_deregister;
        }

        /* Make sure we can use these registers */
        if (request_mem_region(dev.res.start, resource_size(&dev.res),
                               DRIVER_NAME) == NULL) {
            ret = -EBUSY;
            goto out_deregister;
        }

        /* Arrange access to our registers */
        dev.virtbase = of_iomap(pdev->dev.of_node, 0);
        if (dev.virtbase == NULL) {
            ret = -ENOMEM;
            goto out_release_mem_region;
        }

        /* Set an initial state */
        write_joypad_register(&joypad_init);
```

```
        return 0;


    out_release_mem_region:
        release_mem_region(dev.res.start, resource_size(&dev.res));
    out_deregister:
        misc_deregister(&game_boy_misc_device);
        return ret;
    }


    /* Clean-up code: release resources */
    static int game_boy_remove(struct platform_device* pdev)
    {
        iounmap(dev.virtbase);
        release_mem_region(dev.res.start, resource_size(&dev.res));
        misc_deregister(&game_boy_misc_device);
        return 0;
    }


    /* Which "compatible" string(s) to search for in the Device Tree
*/
    #ifdef CONFIG_OF
    static const struct of_device_id game_boy_of_match[] = {
          { .compatible = "csee4840,joypad-1.0" },
```

```
        {},
    };
    MODULE_DEVICE_TABLE(of, game_boy_of_match);
    #endif

    /* Information for registering ourselves as a "platform" driver */
    static struct platform_driver game_boy_driver = {
        .driver = {
            .name   = DRIVER_NAME,
            .owner  = THIS_MODULE,
            .of_match_table = of_match_ptr(game_boy_of_match),
        },
        .remove = __exit_p(game_boy_remove),
    };

    /* Called when the module is loaded: set things up */
    static int __init game_boy_init(void)
    {
        pr_info(DRIVER_NAME ": init\n");
        return platform_driver_probe(&game_boy_driver,
game_boy_probe);
    }
```

```
	/* Calball when the module is unloaded: release resources */
	static void __exit game_boy_exit(void)
	{
	    platform_driver_unregister(&game_boy_driver);
	    pr_info(DRIVER_NAME ": exit\n");
	}

	//
*************************************************************************
****

	module_init(game_boy_init);
	module_exit(game_boy_exit);

	MODULE_LICENSE("GPL");
	MODULE_AUTHOR("Nicolas Alarcon, Columbia University");
	MODULE_DESCRIPTION("Game Boy Joypad Driver");
```

game_boy.h

```
Unset
	#ifndef _GAME_BOY_H
```

```
      #define _GAME_BOY_H


      #include <linux/ioctl.h>


      #define GAME_BOY_MAGIC 'q'


      /* ioctls and their arguments */

      #define GAME_BOY_WRITE_JOYPAD      _IOW(GAME_BOY_MAGIC, 1, uint8_t
*)


      #endif
```

start.c

```
Unset
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ioctl.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>
```

```c
#include <stdbool.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/time.h>
#include <math.h>

#include "game_boy.h"
#include "controller.h"

#define CART_HEADER_ADDR 0x0100

// DE1-SoC H2F AXI bus address
#define H2F_AXI_BASE    0xC0000000
#define H2F_AXI_SPAN    0x04000000
#define SDRAM_OFFSET    0x02000000

// FUNCTION DECLARATIONS

uint8_t update_joypad_status(uint8_t key);
void send_joypad_status(uint8_t reg);
```

```
char parse_printable_key(int key, bool mod, bool caps);
void read_cart();
void read_cart_header(FILE* ptr);

// TYPEDEFs

// Cartridge Header Information
typedef struct {
    uint8_t begin[4];           // 0x0100-0x0103: cart begin code
    uint8_t N_logo[48];           // 0x0104-0x0133: scrolling Nintendo
graphic (MUST NOT MODIFY)
    uint8_t game_title[15];       // 0x0134-0x0142: title of the game in
upper case ASCII
    uint8_t color_gb;               // 0x0143: 0x80 if color GB; else
0x00
    uint8_t licensee_new[2];  // 0x0144-0x0145: (new) licensee code
(normally both 0x00 if address 0x014B != 0x33)
    uint8_t SGB_flag;               // 0x0146: GB/SGB indicator (0x00 =
GB; 0x03 = SGB)
    uint8_t type;                   // 0x0147: cartridge type
    uint8_t ROM_size;               // 0x0148: ROM size
    uint8_t RAM_size;               // 0x0149: RAM size
```

```
    uint8_t dest_code;            // 0x014A: destination code (0x00 =
Japanese; 0x01 = Non-Japanese)
    uint8_t licensee_old;           // 0x014B: (old) licensee code (0x33
= check addresses 0x0144-0x0145; 0x79 = Accolade; 0xA4 = Konami)
    uint8_t mask_ROM;               // 0x014C: mask ROM version number
(usually 0x00)
    uint8_t comp_check;           // 0x014D: complement check (PROGRAM
WON'T RUN IF INCORRECT)
    uint8_t checksum[2];      // 0x014E-0x014F: checksum (GB ignores
this value)
} cart_header;

// GLOBAL VARIABLES
int GB_fd;  // ioctl file descriptor

int mmap_fd;    // /dev/mem file id
void *h2f_virtual_base;  // H2F AXI bus virtual address
volatile uint8_t * sdram_ptr = NULL;

struct libusb_device_handle* controller;

// Cartridge information
cart_header cart_info;
```

```
uint8_t *cart_data; // pointer to cart data start address
uint8_t *save_data; // pointer to save data start address
int ROM_size;       // in bytes
int RAM_size;       // in bytes
uint16_t ROM_bank;  // number of ROM banks
uint8_t MBC_num;    // MBC number
uint8_t RAM_bank;   // number of RAM banks
char ROM_FILE[200];
char *ROM_name;

// MAIN PROGRAM
int main(int argc, char *argv[])
{
    // HANDLE COMMAND LINE ARGS
    int tempfilearg = 0;

    unsigned char double_speed = 0;
    for (int i = 1; i<argc; i++){
        if (strcmp(argv[i], "-d") == 0) {
            double_speed = 1;
        } else {tempfilearg = i;}
    }
```

```
if (tempfilearg == 0){
    printf("Usage: %s <ROM file> [-d] \n", argv[0]);
    exit(1);
}

strcpy(ROM_FILE, argv[tempfilearg]);
char tmp[200];
strcpy(tmp, ROM_FILE);
ROM_name = strtok(tmp, ".");

static const char filename[] = "/dev/game_boy";
if ((GB_fd = open(filename, O_RDWR)) == -1)
{
    fprintf(stderr, "could not open %s\n", filename);
    return -1;
}

// LOAD CARTRIDGE ROM AND SAV RAM TO DE1-SoC SDRAM

// Declare volatile pointers to I/O registers (volatile
// means that IO load and store instructions will be used
// to access these pointer locations,
```

```
    // === get FPGA addresses ==================
    // Open /dev/mem
    if((mmap_fd = open("/dev/mem", (O_RDWR | O_SYNC ))) == -1)
    {
        printf("ERROR: could not open \"/dev/mem\"...\n");
        return(1);
    }

    // =========================================
    // get virtual address for
    // AXI bus address
    h2f_virtual_base = mmap(NULL, H2F_AXI_SPAN, (PROT_READ |
PROT_WRITE), MAP_SHARED, mmap_fd, H2F_AXI_BASE);
    if(h2f_virtual_base == MAP_FAILED)
    {
        printf("ERROR: mmap3() failed...\n");
        close(mmap_fd);
        return(1);
    }

    printf("*************************************************\n");
    sdram_ptr = (uint8_t *) (h2f_virtual_base);
    read_cart();
```

```
printf("************************************************\n");


sdram_ptr = (uint8_t *) (h2f_virtual_base + SDRAM_OFFSET);


// MBC info
sdram_ptr = (uint8_t *) (h2f_virtual_base + SDRAM_OFFSET);
*(sdram_ptr - 5) = MBC_num;          // MBC number
*(sdram_ptr - 4) = RAM_bank;         // Number of banks
*(sdram_ptr - 3) = ROM_bank & 0xFF;  // Lower byte
*(sdram_ptr - 2) = ROM_bank >> 8;    // MSB
*(sdram_ptr - 1) = 1;                // load complete


/* Double Speed Control */
*(sdram_ptr - 6) = double_speed;
if (double_speed)
    printf("Double speed: ON \n");
else
    printf("Double speed: OFF \n");


// JOYPAD INPUT CONTROL
unsigned char packet[8];
int transferred;
unsigned char pressed;
```

```
    unsigned char last_pressed = 0;


    /* Open the controller */
    if ((controller = opencontroller()) == NULL) {
        fprintf(stderr, "Did not find a controller, check Vendor/Product
ID in controller.h\n");
        exit(1);
    }


    /* Handle NES Controller Inputs by sending only new values */
    printf("READY!");
    for (;;)
    {


        libusb_interrupt_transfer(controller, ENDPT_ADDR_IN,
            (unsigned char *)& packet, sizeof(packet),
            &transferred, 500);



        if (transferred > 0 && !EMPTY_INTERR(packet)) {
                  /* --- Start/Select --- */
                  PROCESS(packet[6], SELECT, pressed);
                  PROCESS(packet[6], START, pressed);
```

```
                /* --- A/B --- */
                PROCESS(packet[5], A, pressed);
                PROCESS(packet[5], B, pressed);

                /* --- D_Pad --- */
                PROCESS(packet[4], DOWN, pressed);
                PROCESS(packet[3], RIGHT, pressed);
                EMPTY_PROCESS(packet[4], UP, pressed);
                EMPTY_PROCESS(packet[3], LEFT, pressed);

            // bytes are swapped, and not recognizing a header file
change
            pressed = ((pressed & 0x0F) << 4) | ((pressed & 0xF0) >> 4);
                /* Effectively debounces by introducing a delay after
input was recieved */
                usleep(100 * 1000);
            } else if (transferred > 0){
            pressed = 0x00;
            }
        if (last_pressed != pressed) {
            send_joypad_status(pressed);
        }
```

```
        last_pressed = pressed;
    }

    libusb_close(controller);
    return 0;
}

// FUNCTION DEFINITIONS
void send_joypad_status(uint8_t reg)
{
    uint8_t byte = reg;
    printf("Joypad register is: %.2X \n", byte);
    if (ioctl(GB_fd, GAME_BOY_WRITE_JOYPAD, &byte))
    {
        perror("ioctl(GAME_BOY_WRITE_JOYPAD) failed");
        return;
    }
}

// read cartridge contents
void read_cart()
{
    FILE* cart_ptr;
```

```
    cart_ptr = fopen(ROM_FILE, "rb");


    if (cart_ptr == NULL)
    {
        printf("Unable to open the ROM file \"%s\"!\n", ROM_FILE);
        exit(1);
    }
    else
    {
        printf("ROM file \"%s\" opened successfully! \n\n", ROM_FILE);


        printf("Cartridge information: \n");
        read_cart_header(cart_ptr);


        cart_data = (uint8_t *)malloc(ROM_size);


        fseek(cart_ptr, 0, SEEK_SET);
        for (int i = 0; i < ROM_size; i++)
        {
            fread(cart_data + i, 1, 1, cart_ptr);
            //printf("Address %.4X: %.2X \n", i, *(cart_data + i));
            //printf("Address of cart_data[%d] is: %p \n", i, (void
*)(cart_data+i));
```

```
        }
        fclose(cart_ptr);

        printf("Loading %d bytes of ROM into SDRAM...", ROM_size);
        for (int i = 0; i < ROM_size; i++)
        {
            *(sdram_ptr + i) = *(cart_data + i);
            //printf("SDRAM %.4X: %.2X \n", i, *(sdram_ptr+i));
        }
        printf("complete! \n");
    }
}

// get cartridge information (e.g. MBC type, ROM size, RAM size, etc.)
void read_cart_header(FILE * ptr)
{
    fseek(ptr, CART_HEADER_ADDR, SEEK_SET);
    fread(&cart_info, 1, 0x14F - CART_HEADER_ADDR + 0x01, ptr);

    printf("- Game title: %s \n", cart_info.game_title);

    if (cart_info.color_gb == 0x80)
        printf("- Console: Game Boy Color \n");
```

```
else
    printf("- Console: Game Boy \n");


if (cart_info.SGB_flag == 0x03)
    printf("- Super Game Boy functions supported \n");


char cart_str[26];
switch (cart_info.type)
{
    case 0x00:
        strcpy(cart_str, "ROM ONLY");
        MBC_num = 0;
        break;
    case 0x01:
        strcpy(cart_str, "ROM+MBC1");
        MBC_num = 1;
        break;
    case 0x02:
        strcpy(cart_str, "ROM+MBC1+RAM");
        MBC_num = 1;
        break;
    case 0x03:
        strcpy(cart_str, "ROM+MBC1+RAM+BATT");
```

```
            MBC_num = 1;
            break;
        case 0x05:
            strcpy(cart_str, "ROM+MBC2");
            MBC_num = 2;
            break;
        case 0x06:
            strcpy(cart_str, "ROM+MBC2+BATTERY");
            MBC_num = 2;
            break;
        case 0x08:
            strcpy(cart_str, "ROM+RAM");
            MBC_num = 1;
            break;
        case 0x09:
            strcpy(cart_str, "ROM+RAM+BATTERY");
            MBC_num = 1;
            break;
        case 0x0B:
            strcpy(cart_str, "ROM+MMM01");
            break;
        case 0x0C:
            strcpy(cart_str, "ROM+MMM01+SRAM");
```

```
            break;
        case 0x0D:
            strcpy(cart_str, "ROM+MMM01+SRAM+BATT");
            break;
        case 0x0F:
            strcpy(cart_str, "ROM+MBC3+TIMER+BATT");
            MBC_num = 3;
            break;
        case 0x10:
            strcpy(cart_str, "ROM+MBC3+TIMER+RAM+BATT");
            MBC_num = 3;
            break;
        case 0x11:
            strcpy(cart_str, "ROM+MBC3");
            MBC_num = 3;
            break;
        case 0x12:
            strcpy(cart_str, "ROM+MBC3+RAM");
            MBC_num = 3;
            break;
        case 0x13:
            strcpy(cart_str, "ROM+MBC3+RAM+BATT");
            MBC_num = 3;
```

```
            break;
        case 0x19:
            strcpy(cart_str, "ROM+MBC5");
            MBC_num = 5;
            break;
        case 0x1A:
            strcpy(cart_str, "ROM+MBC5+RAM");
            MBC_num = 5;
            break;
        case 0x1B:
            strcpy(cart_str, "ROM+MBC5+RAM+BATT");
            MBC_num = 5;
            break;
        case 0x1C:
            strcpy(cart_str, "ROM+MBC5+RUMBLE");
            MBC_num = 5;
            break;
        case 0x1D:
            strcpy(cart_str, "ROM+MBC5+RUMBLE+SRAM");
            MBC_num = 5;
            break;
        case 0x1E:
            strcpy(cart_str, "ROM+MBC5+RUMBLE+SRAM+BATT");
```

```
            MBC_num = 5;
            break;
        case 0x1F:
            strcpy(cart_str, "Pocket Camera");
            break;
        case 0xFD:
            strcpy(cart_str, "Bandai TAMA5");
            break;
        case 0xFE:
            strcpy(cart_str, "Hudson HuC-3");
            break;
        case 0xFF:
            strcpy(cart_str, "Hudson HuC-1");
            break;
        default:
            strcpy(cart_str, "Invalid cartridge type");
            exit(1);
    }
    printf("- Cartridge type: %s \n", cart_str);

    switch (cart_info.ROM_size)
    {
        case 0x00:
```

```
            ROM_bank = 2;  // 32kB
            break;
        case 0x01:
            ROM_bank = 4;  // 64kB
            break;
        case 0x02:
            ROM_bank = 8;  // 128kB
            break;
        case 0x03:
            ROM_bank = 16;  // 256kB
            break;
        case 0x04:
            ROM_bank = 32;  // 512kB
            break;
        case 0x05:
            ROM_bank = 64;  // 1MB
            break;
        case 0x06:
            ROM_bank = 128;  // 2MB
            break;
        case 0x07:
            ROM_bank = 256;  // 4MB
            break;
```

```
        case 0x08:
            ROM_bank = 512;  // 8MB
            break;
        case 0x52:
            ROM_bank = 72;  // 1.1MB
            break;
        case 0x53:
            ROM_bank = 80;  // 1.2MB
            break;
        case 0x54:
            ROM_bank = 96;  // 1.5MB
            break;
        default:
            printf("Invalid ROM size \n");
            exit(1);
    }
    ROM_size = ROM_bank * 16 * 1024;
    printf("- ROM size: %d bytes (%d banks) \n", ROM_size, ROM_bank);

    switch (cart_info.RAM_size)
    {
        case 0x00:
            RAM_bank = 0;
```

```
            RAM_size = 0;
            break;
        case 0x01:
            RAM_bank = 1;
            RAM_size = 2 * 1024;        // 2kB
            break;
        case 0x02:
            RAM_bank = 1;
            RAM_size = 8 * 1024;        // 8kB
            break;
        case 0x03:
            RAM_bank = 4;
            RAM_size = 4 * 8 * 1024;    // 32kB
            break;
        case 0x04:
            RAM_bank = 16;
            RAM_size = 16 * 8 * 1024;   // 128kB
            break;
        default:
            printf("Invalid RAM size \n");
            exit(1);
    }
    printf("- RAM size: %d bytes (%d banks) \n", RAM_size, RAM_bank);
```

```
    // MBCs not supported in this version of hardware
    if (MBC_num != 0)
    {
        printf("Memory Bank Controllers not yet implemented!\n");
        exit(1);
    }

}
```

README

```
Unset
      # GameBoy-Hardware-Emulator
      A Hardware Emulator for the 1989 GameBoy for the Terasic DE1-SoC
Development Board, using an NES controller as a user input, and VGA to a
monitor for display

      ## Usage
      ### To Compile Hardware

      - Open a terminal. Change the directory to the RTL folder.

      - Add the following to joypad_hw.tcl if it isn't already there:
```

```
    set_module_assignment embeddedsw.dts.vendor "csee4840"
    set_module_assignment embeddedsw.dts.name "joypad"
    set_module_assignment embeddedsw.dts.group "game_boy"
```

- Run the following commands:

```
    chmod +x compile-scripts.sh
    ./compile-scripts.sh
    make dtb
```

which runs:
```
      make qsys-clean && make qsys && make rbf;
      embedded_command_shell.sh;
      make dtb;
```

- At this point, you may get several "ERROR (phandle_references): Reference to non-existent node or label Main_PLL". Add the following to the soc_system.dts file under the "clocks" node (e.g. line 43):

```
    Main_PLL: Main_PLL {
      compatible = "fixed-clock";
      #clock-cells = <0>;
      clock_frequency = <0>;  /* 0 Hz */
      clock-output-names = "Main_PLL-clk";
    };
```

- In reality, the Main_PLL is not used by Linux, so the clock frequency is not important. This just prevents "make dtb" from complaining.

```
    make dtb
```

- Copy the .rbf file (in output_files folder) and .dtb file onto the boot partition of the DE1-SoC board by. Reboot the board.

```
```

```
    mount /dev/mmcblk0p1 /mnt
    cp output_files/soc_system.rbf /mnt/
    cp soc_system.dtb /mnt/
    sync
```

If this does not work:

1. First verify the date of the .dtb and .rbf files on the SD Card.
2. Check "/proc/device-tree/sopc@0/bridge@0xc0000000/" for the devices listed under the soc_system.dts file

You may copy the files directly to the MicroSD card of the DE1 SoC from another device with an SD Card Reader.

From DE1 SoC, Software for loading/running ROM + interfacing with NES controller

- The controller Product and Vendor ID are configured in controller.h; verify they are correct with by running 'lsusb'

- Open a console. Change the directory to where software source files are. Run the following commands in order:

```
    make
    insmod game_boy.ko
```

- Before you run the program, press Key0 and Key1 on the DE1-SoC board simultaneously to reset the soc_system.

```
    ./start [-d] ROM.gb
```

[-d] option is to enable double speed

- After running the program, press Key2 and Key3 on the DE1-SoC board simultaneously to reset the Game Boy.

- To exit the game, press Ctrl-C on the console

- To clean up:

```
make clean
rmmod game_boy
clear
```

GameBoy_Joypad.sv

Unset

```
module GameBoy_Joypad
(
	input logic clk,
	input logic reset,
	/* Avalon Slave */
	input logic [7:0]  writedata_slv,
	input logic write_slv,
	input              chipselect_slv,
	/* Gameboy JoyPad Conduit */
	input logic P15,
	input logic P14,
	output  logic     P13,
	output  logic     P12,
	output  logic     P11,
	output  logic     P10
);
```

```
logic [7:0]       joypad;


always_ff @(posedge clk)
begin
      if (reset)
      begin
            joypad <= 8'h00;
      end
      else if (chipselect_slv && write_slv)
      begin
            joypad <= writedata_slv;
      end
end


always_comb
begin
       P10 = 1;
       P11 = 1;
       P12 = 1;
       P13 = 1;
    if (!P14)
       begin
```

```
            if (joypad[0])  // RIGHT
                P10 = 0;
            if (joypad[1])  // LEFT
                P11 = 0;
            if (joypad[2])  // UP
                P12 = 0;
            if (joypad[3])  // DOWN
                P13 = 0;
        end
    if (!P15)
        begin
            if (joypad[4])  // A
                P10 = 0;
            if (joypad[5])  // B
                P11 = 0;
            if (joypad[6])  // SELECT
                P12 = 0;
            if (joypad[7])  // START
                P13 = 0;
        end
end

endmodule
```

GameBoy_VGA.sv

```
Unset
      /*
       * Avalon memory-mapped peripheral that generates VGA
       *
       * Original By Stephen A. Edwards
       * Columbia University
       * Modified By Nicolas Alarcon
       * Columbia University
       * 1280 x 1024 @ 60 Hz
       */


      module GameBoy_VGA
      (
            input logic             clk, //only used for resetting
background 67.1 MHz
            input logic GameBoy_clk, // the 2^22 Hz GameBoy clk for
framebuffer, from Cartridge
            input logic             reset, //only used for resetting
background
            input logic GameBoy_reset, // Reset synced to GameBoy clk


            input logic clk_vga, // 108 MHz
```

```
      /* Avalon Slave from lab3*/
      input logic [7:0]   writedata,
      input logic     write,
    input                  chipselect,
    input logic [20:0]  address,

      /* VGA Conduit */
      output logic [7:0]      VGA_R, VGA_G, VGA_B,
      output logic            VGA_CLK, VGA_HS, VGA_VS,
                                    VGA_BLANK_n,
      output logic            VGA_SYNC_n,

      /* GameBoy Pixel Conduit */
      input logic [1:0] LD,
      input logic       PX_VALID);

// VGA signals
logic [15:0] LX;
logic [15:0] LY;

logic [7:0] bg_r, bg_g, bg_b;

logic [1:0] GB_PIXEL;
```

```
      // Instantiations
      vga_counters counters(.*);

      /* The Framebuffer for gameboy */
      logic [14:0] frame_buffer_cnt;
      logic frame_buffer_switch;

      always_ff @(posedge GameBoy_clk or posedge GameBoy_reset)
      begin
          if (GameBoy_reset) begin
              frame_buffer_cnt <= 0;
              frame_buffer_switch <= 0;
          end else if (PX_VALID) frame_buffer_cnt <= (frame_buffer_cnt
== 23039) ? 0: frame_buffer_cnt + 1;
      end

      //
      logic [7:0] GB_LX, GB_LY; //frame buffer addressing
      logic [2:0] GB_COL_CNT, GB_ROW_CNT;

      /* Output Scaling Machine */
      always_ff @(posedge clk_vga)
```

```
begin
      //in X window and read from same address
      if (LX < 160 || LX >= 1120) begin
            GB_LX <= 0;
            GB_COL_CNT <= 0;
      end else begin
            GB_COL_CNT <= GB_COL_CNT + 1;
      end

      // increment X framebuffer address
      if (GB_COL_CNT == 5)
      begin
            GB_COL_CNT <= 0;
            GB_LX <= GB_LX + 1;
      end

      //in Y window and read from same address
      if (LY <= 80 || LY >= 944) begin
            GB_LY <= 0;
            GB_ROW_CNT <= 0;
      end else if (LX == 1) begin
            GB_ROW_CNT <= GB_ROW_CNT + 1;
      end
```

```
            //increment framebuffer Y address
            if (GB_ROW_CNT == 6)
            begin
                  GB_ROW_CNT <= 0;
                  GB_LY <= GB_LY + 1;
            end
      end

      /*
      Runs at GameBoy clock Speed and read out at VGA clock speed

      2^22 Hz/(108MHz) = 606.815...

      But does not produce a noticeable jitter, 2^22 used to produce an
accurate in game clock

      GB_LX runs from 0-159
      GB_LY runs from 0-143

      Indexing finds pixel at spot
      */
      Quartus_dual_port_dual_clk_ram LCD_FRAME_BUFFER0(
```

```
            .write_clk(~GameBoy_clk),
            .read_clk(~clk_vga),
            .data(LD),
            .we(PX_VALID),
            .write_addr(frame_buffer_cnt),
            .read_addr({7'b0, GB_LX} + {2'b0, GB_LY, 5'b0} + {GB_LY,
7'b0}),
            .q(GB_PIXEL)
      );



      always_ff @(posedge clk)
      begin
            if (reset)
            begin //pantone 292!
                  bg_r <= 8'd105;
                  bg_g <= 8'd179;
                  bg_b <= 8'd231;
            end
            else if (chipselect && write)
            begin
                  bg_r <= 8'h80;
            end
```

```
      end


      // Kirokaze GB Color Palette, tried to match blue but was ugly
      // GBPixel -> RGB Decoder
      always_comb
      begin
         {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
         if (VGA_BLANK_n)
         begin
                  if (LX >= 160 && LX <= 1120 && LY >= 80 && LY <= 944)
//within screen window
                  begin
                        unique case (GB_PIXEL)
                              2'b11:
                              begin //dark blue
                                  VGA_R = 51;
                                  VGA_G = 44;
                                  VGA_B = 80;
                              end
                              2'b10:
                              begin //blue-green
                                  VGA_R = 70;
                                  VGA_G = 135;
```

```
                    VGA_B = 143;
                end
                2'b01:
                begin //neon green
                    VGA_R = 148;
                    VGA_G = 227;
                    VGA_B = 68;
                end
                2'b00:
                begin //lightest
                    VGA_R = 226;
                    VGA_G = 243;
                    VGA_B = 228;
                end
            endcase
            // Lines Between Pixels
            if (GB_ROW_CNT == 0 || GB_COL_CNT == 0)
            begin
                VGA_R = 51;
                VGA_G = 44;
                VGA_B = 80;
            end
        end
```

```
                else {VGA_R, VGA_G, VGA_B} = {bg_r, bg_g, bg_b};
            end
        end


        endmodule


        module vga_counters
        (
            input logic                 clk_vga, reset,
          output  logic [15:0]          LX,
            output      logic [15:0]          LY,
            output      logic                 VGA_CLK, VGA_HS, VGA_VS,
VGA_BLANK_n, VGA_SYNC_n
        );


        logic [15:0] hcount, vcount;
            /*
            * 1280 X 1024 VGA timing for a 108 MHz clock: one pixel
every cycle, from tinyvga.com
            *
            * HCOUNT 1687 0             1279       1687 0
            *             ---------------             --------
            * ___________|    Video      |____________|  Video
```

```
            *
            *
            * |SYNC| BP |<-- HACTIVE -->|FP|HACTIVESYNC| BP |<-- HACTIVE
            *       _______________________      _____________
            * |____|       VGA_HS          |____|
            */
            // Parameters for hcount
            parameter         HACTIVE      = 1280,
                                  HFRONT_PORCH = 48,
                                  HSYNC        = 112,
                                  HBACK_PORCH  = 248,
                                  HTOTAL       = HACTIVE +
HFRONT_PORCH + HSYNC + HBACK_PORCH; // 1688

            // Parameters for vcount
            parameter         VACTIVE      = 1024,
                                  VFRONT_PORCH = 1,
                                  VSYNC        = 3,
                                  VBACK_PORCH  = 38,
                                  VTOTAL       = VACTIVE +
VFRONT_PORCH + VSYNC + VBACK_PORCH; // 1066

            logic endOfLine;
```

```
            logic endOfField;

            always_ff @(posedge clk_vga or posedge reset)
                  if (reset)          hcount <= 0;
                  else if (endOfLine) hcount <= 0;
                  else           hcount <= hcount + 1;

            always_ff @(posedge clk_vga or posedge reset)
                  if (reset)          vcount <= 0;
                  else if (endOfLine)
                        if (endOfField)   vcount <= 0;
                        else              vcount <= vcount + 1;

            assign endOfField = vcount == VTOTAL - 1;
            assign endOfLine  = hcount == HTOTAL - 1;

            // Horizontal and Vertical Sync -> From Graph
            assign VGA_HS = !((hcount >= HACTIVE + HFRONT_PORCH) &&
(hcount < HACTIVE + HFRONT_PORCH + HSYNC));
            assign VGA_VS = !((vcount >= VACTIVE + VFRONT_PORCH) &&
(vcount < VACTIVE + VFRONT_PORCH + VSYNC));
```

```
            assign VGA_SYNC_n = 1'b0;     // For putting sync on the
green signal; unused

            // Horizontal Active and Vertical Active
            assign VGA_BLANK_n = (hcount < HACTIVE)  && (vcount <
VACTIVE);

            assign VGA_CLK = clk_vga;     // 108 MHz clock: rising edge
sensitive

            //assign module outputs
            assign LX = hcount;
            assign LY = vcount;

      endmodule
```

PPU3_D.sv

```
Unset
      // Authors: Nicolas Alarcon, Claire Cizdziel, Donovan Sproule

      `define MEM_CYCLE
```

```
	`define OAM_BASE_ADDR 16'hFE00
	`define OAM_END_ADDR 16'hFEA0

	`define BG_MAP_1_BASE_ADDR 16'h9800
	`define BG_MAP_2_BASE_ADDR 16'h9C00

	`define BG_MAP_1_END_ADDR 16'h9BFF

	`define MAX_LY 8'd155

	`define NO_BOOT 0

	`define PPU_ADDR_INC(x) `PPU_ADDR_SET(PPU_ADDR + x)
	`ifdef MEM_CYCLE
	`define PPU_ADDR_SET(x) PPU_ADDR <= x; mem_config <= MEM_REQ;
	`else
	`define PPU_ADDR_SET(x) PPU_ADDR <= x;
	`endif

	/* Macros that set STAT flags */
	`define PPU_MODE_SET(x) PPU_MODE <= x; FF41[1:0] <= x
	`define LY_UPDATE(x) LY <= x; FF41[2] <= (x == LYC); FF45 <=
{7'b0, (x == LYC)}
```

```
      `define WXC_UPDATE(x) if (LY >= WY && ((real_wx >= SCX) &&
(real_wx <= (168 + SCX)))) WXC <= x


      typedef enum bit [1:0] {PPU_H_BLANK, PPU_V_BLANK, PPU_SCAN,
PPU_DRAW} PPU_STATES_t;
      typedef enum bit [2:0] {BG_TILE_NO_STORE, BG_ROW_1_LOAD,
BG_ROW_2_LOAD, BG_READY, BG_PAUSE} BG_DRAW_STATES_t;
      typedef enum bit [2:0] {SP_SEARCH, SP_ROW_1_LOAD, SP_ROW_2_LOAD,
SP_READY, SP_RUN_BG, SP_TILE_LOAD} SP_DRAW_STATES_t;
      typedef enum bit [2:0] {MIX_LOAD, MIX_START, MIX_PUSH}
MIX_STATES_t;
      typedef enum bit [2:0] {MEM_EMPTY, MEM_REQ, MEM_LOAD, MEM_NO_REQ}
MEM_STATES_t;
      typedef enum bit [1:0] {W_NO_EDGE, W_BG_RUN, W_MASK_RUN}
W_STATES_t;


      /* Doesn't track clock cycles but instead measures progressions in
the form of 'dots'. Any instruction that processes
       * data and progresses the PPU iterates a dot, most of the
exceptions being stall clock cycles where we make a memory
       * request and must halt until we can use the data
       */
```

```
module PPU3
(
    input logic clk,
    input logic rst,

    /* System access for PPU internal registers */
    input logic [15:0] ADDR,
    input logic WR,
    input logic RD,
    input logic [7:0] MMIO_DATA_out,
    output logic [7:0] MMIO_DATA_in,

    /* Interrupts */
    output logic IRQ_PPU_V_BLANK,
    output logic IRQ_LCDC,

    output logic [1:0] PPU_MODE,

    /* VRAM access for PPU */
    output logic PPU_RD,
    output logic [15:0] PPU_ADDR,
    input logic [7:0] PPU_DATA_in,
```

```
        output logic [1:0] PX_OUT,
        output logic PX_valid
    );

    /* Extensions */
    logic [15:0] BIG_DATA_in, BIG_LY_SCY_MOD, BIG_X;
    logic signed [15:0] S_BIG_DATA_in;

    /* Scanline tracking */
    logic [7:0] LY, x_pos;              // x-pos in range [0, 159]
    logic [8:0] dots;
    logic [3:0] pixels_pushed;
    logic [15:0] x_tile_off;
    logic [15:0] y_tile_off;

    /* Sprite OAM Scan vars */
    logic [15:0] curr_off;
    logic [3:0] sp_loaded;
    logic sp_in_range;
    logic sp_found;
    logic tts;                                      // tall tile
selector
    logic [15:0] offset_jump;
```

```
      logic [7:0] sp_y_buff [9:0];
      logic [7:0] sp_x_buff [9:0];
      logic [7:0] sp_off_buff [9:0];            // stores start of
sprite entries in OAM (entry + 2 => for tile no.)

      /* BG fetching vars */
      logic bg_fifo_go;
      logic bg_fifo_load;
      logic [2:0] bg_fetch_mode;
      logic [7:0] bg_tile_row [1:0];
      logic [15:0] bg_map_sel;

      /* SP fetching vars */
      logic sp_fifo_go;
      logic sp_fifo_load;
      logic [15:0] alt_data_in_sel;
      logic [15:0] alt_tile_base;
      logic [7:0] curr_sp_flag;
      logic [7:0] sp_real_x;                    // used to account for
the 16 offset
      logic [3:0] sp_ind;
      logic [2:0] sp_fetch_mode;
```

```
logic [7:0] sp_tile_row [1:0];


/* Window */
logic [7:0] WXC;
logic [7:0] real_wx;
logic [15:0] x_w_off;
logic [15:0] y_w_off;
logic [1:0] w_check;
logic [7:0] w_mask;
logic [15:0] w_map_sel;


/* Pixel Masking */
logic unsigned [7:0] gen_mask;
logic unsigned [7:0] sp_mask;


/* Pixel mixing */
logic ready_load;
logic flush_buff;
logic [1:0] bg_out;
logic [1:0] sp_out;
logic [2:0] px_mix_mode;


/* Dealayed Data Input */
```

```
      `ifdef MEM_CYCLE
      logic [2:0] mem_config;
      `endif

      /* Frame Reset */
      logic frame_rst;

      /* Assigns */
      assign BIG_DATA_in = {8'b0, PPU_DATA_in};
      assign S_BIG_DATA_in = {8'b0, PPU_DATA_in};
      assign BIG_LY_SCY_MOD = {13'b0, LY[2:0] + SCY[2:0]};
      assign BIG_X = {8'b0,x_pos};

      assign sp_in_range = ((LY + SCY + 16 >= PPU_DATA_in) &&
                                              (LY + SCY + 16 <
PPU_DATA_in + (8 << LCDC[2])));
      assign curr_off = PPU_ADDR - `OAM_BASE_ADDR;
      assign x_tile_off = (((BIG_X >> 3) + ({8'b0, SCX} >> 3)) &
16'h3FFF);
      assign y_tile_off = (((({8'h0, LY} + {8'h0, SCY}) & 16'hFF) >> 3)
<< 5) & 16'h3FFF;

      assign x_w_off = ((({8'b0, x_pos - real_wx} >> 3)) & 16'h3FFF);
```

```
      assign y_w_off = ((({8'h0, WXC}) >> 3) << 5) & 16'h3FFF;

      assign sp_real_x = sp_x_buff[sp_ind] - 8;
      assign real_wx = WX - 7;

      assign w_map_sel = (LCDC[6]) ? `BG_MAP_2_BASE_ADDR :
`BG_MAP_1_BASE_ADDR;
      assign bg_map_sel = (LCDC[3]) ? `BG_MAP_2_BASE_ADDR :
`BG_MAP_1_BASE_ADDR;

      PPU_SHIFT_REG bg_fifo(.clk(clk), .rst(rst), .data(bg_tile_row),
.go(bg_fifo_go), .load(bg_fifo_load), .q(bg_out));
      PPU_SHIFT_REG sp_fifo(.clk(clk), .rst(rst), .data(sp_tile_row),
.go(sp_fifo_go), .load(sp_fifo_load), .q(sp_out));

      /* External registers */
      logic [7:0] LCDC, STAT, SCX, SCY, LYC, DMA, BGP, OBP0, OBP1, WX,
WY; // Register alias

      logic [7:0] FF40;
      assign LCDC = FF40;

      logic [7:0] FF41;
```

```
assign STAT = FF41;


logic [7:0] FF42;
assign SCY = FF42;


logic [7:0] FF43;
assign SCX = FF43;


logic [7:0] FF44;
assign FF44 = LY;


logic [7:0] FF45;
assign LYC = FF45;


logic [7:0] FF46;
assign DMA = FF46;


logic [7:0] FF47;
assign BGP = FF47;


logic [7:0] FF48;
assign OBP0 = {FF48[7:2], 2'b00}; // Last 2 bits are not used
```

```
	logic [7:0] FF49;
	assign OBP1 = {FF49[7:2], 2'b00};

	logic [7:0] FF4A;
	assign WY = FF4A;

	logic [7:0] FF4B;
	assign WX = FF4B;



	/* STAT Interrupts */
	logic IRQ_STAT, IRQ_STAT_NEXT; // The Internal IRQ signal, IRQ
LCDC Triggered on the rising edge of this

	always_comb begin
		IRQ_STAT_NEXT = (FF41[6] && LY == LYC) ||
				(FF41[3] && PPU_MODE == PPU_H_BLANK) ||
				(FF41[5] && PPU_MODE == PPU_SCAN) ||
				((FF41[4] || FF41[5]) && PPU_MODE ==
PPU_V_BLANK);
		IRQ_STAT_NEXT = IRQ_STAT_NEXT & LCDC[7];
	end
```

```
      assign IRQ_LCDC = IRQ_STAT_NEXT && !IRQ_STAT;

      /*
       *  Pixel Assigns
      */
      assign flush_buff = (PPU_MODE == PPU_DRAW) &&
                          !(ready_load & (pixels_pushed == 4'hA)) &&
                          (pixels_pushed > 0 & pixels_pushed <= 8);

      assign PX_OUT = (sp_out == 2'h0 || (bg_out != 2'h0 &&
curr_sp_flag[7])) ? bg_out : sp_out;
      assign PX_valid = (flush_buff && (x_pos <= 160 || (x_pos <= 168 &&
SCX != 0)));

      assign alt_tile_base = (LCDC[4]) ? 16'h8000 : 16'h9000;
      assign alt_data_in_sel = (LCDC[4]) ? BIG_DATA_in : S_BIG_DATA_in;

      /*
       * If we detect a memory request we return back the current
       * state of the register
       */
      always_latch
      begin
```

```
        if (RD) begin
            case (ADDR)
                16'hFF40: MMIO_DATA_in = FF40;
                16'hFF41: MMIO_DATA_in = {1'b1, FF41[6:0]};
                16'hFF42: MMIO_DATA_in = FF42;
                16'hFF43: MMIO_DATA_in = FF43;
                16'hFF44: MMIO_DATA_in = FF44;
                16'hFF45: MMIO_DATA_in = FF45;
                16'hFF46: MMIO_DATA_in = FF46;
                16'hFF47: MMIO_DATA_in = FF47;
                16'hFF48: MMIO_DATA_in = FF48;
                16'hFF49: MMIO_DATA_in = FF49;
                16'hFF4A: MMIO_DATA_in = FF4A;
                16'hFF4B: MMIO_DATA_in = FF4B;
                default : MMIO_DATA_in = 8'hFF;
            endcase
        end
    end

    always_ff @(posedge clk) begin

        /* Register Assignment
        *
```

```
            *     if a register memory address is being indexed it gets
updated here
            *  -- As a warning the following may introduce timing
oddities compared to other group
            */

            if (rst)
          begin
            IRQ_STAT <= 0;
              FF40 <= `NO_BOOT ? 8'h91 : 0;
              FF41 <= 0;
              FF42 <= 0;
              FF43 <= 0;
              FF45 <= 0;
              FF46 <= 0;
              FF47 <= `NO_BOOT ? 8'hFC : 0;
              FF48 <= `NO_BOOT ? 8'hFF : 0;
              FF49 <= `NO_BOOT ? 8'hFF : 0;
              FF4A <= 0;
              FF4B <= 0;
          end
          else
          begin
```

```
            IRQ_STAT <= IRQ_STAT_NEXT;
              FF40 <= (WR && (ADDR == 16'hFF40)) ? MMIO_DATA_out : FF40;
                FF41 <= (WR && (ADDR == 16'hFF41)) ?
{MMIO_DATA_out[7:3], FF41[2:0]} : {FF41[7:3], LYC == LY, PPU_MODE};
                FF42 <= (WR && (ADDR == 16'hFF42)) ? MMIO_DATA_out :
FF42;
                FF43 <= (WR && (ADDR == 16'hFF43)) ? MMIO_DATA_out :
FF43;
                FF45 <= (WR && (ADDR == 16'hFF45)) ? MMIO_DATA_out :
FF45;
                FF46 <= (WR && (ADDR == 16'hFF46)) ? MMIO_DATA_out :
FF46;
                FF47 <= (WR && (ADDR == 16'hFF47)) ? MMIO_DATA_out :
FF47;
                FF48 <= (WR && (ADDR == 16'hFF48)) ? MMIO_DATA_out :
FF48;
                FF49 <= (WR && (ADDR == 16'hFF49)) ? MMIO_DATA_out :
FF49;
                FF4A <= (WR && (ADDR == 16'hFF4A)) ? MMIO_DATA_out :
FF4A;
                FF4B <= (WR && (ADDR == 16'hFF4B)) ? MMIO_DATA_out :
FF4B;
          end
```

```
      /* -- Memory Loading machine -- */
`ifdef MEM_CYCLE
      if (mem_config != MEM_NO_REQ) mem_config <= mem_config + 1;
`endif

      frame_rst <= 0;
      /* -- State Switching machine -- */
    if (rst || frame_rst) begin
                  x_pos <= 0;
                  dots <= 0;
                  `LY_UPDATE(0);
                  WXC <= 0;
                  `PPU_ADDR_SET(`OAM_BASE_ADDR);
                  `PPU_MODE_SET(PPU_SCAN);
                  PPU_RD <= 1;
`ifdef  MEM_CYCLE
      end else if (LCDC[7] && mem_config == MEM_NO_REQ) begin
`else
      end else if (LCDC[7]) begin
`endif
      dots <= dots + 1;
```

```
                /* -- Following block happens on a per scanline basis
(456 dots per line) -- */
              case (PPU_MODE)
                  PPU_SCAN: begin
                        if (dots == 79) begin
                                    `PPU_MODE_SET(PPU_DRAW);
                                    ready_load <= 1;
                                    pixels_pushed <= 4'hA;
                                    x_pos <= 0;

                                    bg_fetch_mode <= BG_PAUSE;
                                    sp_fetch_mode <= SP_SEARCH;
                        end
                              if (LY >= 144) begin
                                    `PPU_MODE_SET(PPU_V_BLANK);
                                    PPU_RD <= 0;
                              end
                        end
                    PPU_DRAW: begin
                        if ((x_pos > 160 && SCX == 0) || x_pos > 168)
begin
                              `PPU_MODE_SET(PPU_H_BLANK);
                              PPU_RD <= 0;
```

```
                        end
                    end
                    PPU_H_BLANK: begin
                        if (dots >= 455) begin
// we reached the end of the scanline
                                `LY_UPDATE(LY + 1);
                                `WXC_UPDATE(WXC + 1);
                                for (int i = 0; i < 10; i++) begin
                                sp_x_buff[0] <= 8'd0;
                                sp_y_buff[0] <= 8'd0;
                                sp_off_buff[i] <= 8'd0;
                            end
                                x_pos <= 0;
                                sp_loaded <= 0;
                                dots <= 0;
                                `PPU_MODE_SET(PPU_SCAN);
                                PPU_RD <= 1;
                                `PPU_ADDR_SET(`OAM_BASE_ADDR);
                            end else begin
                                `PPU_ADDR_SET(0);
                            end
                        end
                    PPU_V_BLANK: begin
```

```
                              if (dots >= 455) begin        // we
reached the end of the scanline
                                    `LY_UPDATE(LY + 1);
                                    `WXC_UPDATE(WXC + 1);
                                    dots <= 0;
                                    if (LY >= `MAX_LY) begin// we are
ready to render the next frame
                                          frame_rst <= 1;
                                    end
                              end else begin
                                    `PPU_ADDR_SET(0);
                              end
                        end
              endcase
            end

            /* -- OAM Scan State Machine -- */
            if (rst || frame_rst || PPU_MODE == PPU_H_BLANK) begin
                  sp_loaded <= 0;
                  sp_found <= 0;
      `ifdef MEM_CYCLE
            end else if (mem_config == MEM_NO_REQ) begin
      `else
```

```
            end else begin
      `endif
                  if (PPU_MODE == PPU_SCAN) begin
                        if (!dots[0]) begin
                        // forces alternating clock dots
                              if (sp_in_range && sp_loaded < 10) begin
                                    sp_loaded <= sp_loaded + 1;
                                    sp_y_buff[sp_loaded] <= PPU_DATA_in;
                                    sp_off_buff[sp_loaded] <=
curr_off[7:0];
                                    sp_found <= 1;
                              end
                              `PPU_ADDR_INC(1);
                        end else begin
                              if (sp_found) begin
                                    sp_x_buff[sp_loaded - 1] <=
PPU_DATA_in;
                              end
                              `PPU_ADDR_INC(3);
            // jumps to next sprite in OAM
                              sp_found <= 0;
                        end
                  end
```

```
            end


      `ifdef MEM_CYCLE
            if (PPU_MODE == PPU_DRAW && px_mix_mode == MIX_LOAD)
                  if (!ready_load && pixels_pushed > 0) pixels_pushed <=
pixels_pushed - 1;
      `endif


            /* BG/Window Sprite Draw Machine
            *     State machine iterates through detected sprites for a
PPU_scanline,
            *     loads the rows into the sp_fifo and switches the bg
drawing on
            */
            if (rst || frame_rst) begin
                  pixels_pushed <= 4'hA;
                  sp_ind <= 0;
                  ready_load <= 1;
      `ifdef MEM_CYCLE
            end else if (mem_config == MEM_NO_REQ) begin
      `else
            end else begin
      `endif
```

```
                if (PPU_MODE == PPU_DRAW) begin
                      /* Background Fetch Machine */
                      case (bg_fetch_mode)
                            BG_TILE_NO_STORE: begin
                                  bg_fetch_mode <= BG_ROW_1_LOAD;

                                  `PPU_ADDR_SET(alt_tile_base +
(BIG_LY_SCY_MOD << 1) + (alt_data_in_sel << 4));      // tile_base + 2 *
(LY + SCY % 8) + (16 * tile_no)

                                  if (LCDC[5]) begin
                                        if (LY >= WY && (x_pos + SCX
>= real_wx)) begin

`PPU_ADDR_SET(alt_tile_base + ({13'h0, WXC[2:0]} << 1) +
(alt_data_in_sel << 4));      // tile_base + (16 * tile_no) + 2 * (LY +
SCY % 8)
                                        end
                                  end

                                  if (x_pos == 0) gen_mask <= 8'hFF >>
SCX[2:0];
```

```
                                    else if (x_pos == 160) gen_mask <=
~(8'hFF << SCX[2:0]);
                                    else gen_mask <= 8'hFF;

                                    if (w_check == W_BG_RUN) w_mask <=
8'hFF << (8 - (real_wx - x_pos - SCX));
                                    if (w_check == W_MASK_RUN) w_mask <=
8'hFF >> (real_wx - x_pos - SCX);
                              end
                              BG_ROW_1_LOAD: begin
                                    if (LCDC[0]) begin
                                          bg_tile_row[0] <= PPU_DATA_in
& gen_mask;
                                          if (w_check == W_BG_RUN)
bg_tile_row[0] <= PPU_DATA_in & w_mask;
                                          if (w_check == W_MASK_RUN)
bg_tile_row[0] <= bg_tile_row[0] | (PPU_DATA_in & w_mask);
                                    end else bg_tile_row[0] <= 8'h0;

                                    bg_fetch_mode <= BG_ROW_2_LOAD;
                                    `PPU_ADDR_INC(1);
                              end
                              BG_ROW_2_LOAD: begin
```

```
                                    if (LCDC[0]) begin
                                          bg_tile_row[1] <= PPU_DATA_in
& gen_mask;
                                          if (w_check == W_BG_RUN)
bg_tile_row[1] <= PPU_DATA_in & w_mask;
                                          if (w_check == W_MASK_RUN)
bg_tile_row[1] <= bg_tile_row[1] | (PPU_DATA_in & w_mask);
                                    end else bg_tile_row[1] <= 8'h0;

                                    if (w_check == W_BG_RUN) begin
                                          bg_fetch_mode <=
BG_TILE_NO_STORE;
                                          `PPU_ADDR_SET(w_map_sel +
{x_w_off[15:3], 3'h0} + y_w_off);
                                          w_check <= W_MASK_RUN;
                                    end else begin
                                          bg_fetch_mode <= BG_READY;
                                    end
                              end
                              default: begin
                              end
                        endcase
```

```
                        /* Sprite Fetch Machine */
                        case (sp_fetch_mode)
                              SP_SEARCH: begin
                                    if (sp_x_buff[sp_ind] >= 8 &&
                                                ((sp_real_x >= x_pos &&
sp_real_x < x_pos + 8) ||
                                                  sp_real_x + 8 > x_pos
&& sp_real_x + 8 <= x_pos + 8) &&
                                                      LCDC[1]) begin
            // end of sprite in tile

                                          `PPU_ADDR_SET(`OAM_BASE_ADDR +
{8'b0, sp_off_buff[sp_ind]} + 2);
                                          sp_fetch_mode <= SP_TILE_LOAD;

                                          if (x_pos == 0) gen_mask <=
8'hFF >> SCX[2:0];
                                          else if (x_pos == 160)
gen_mask <= ~(8'hFF << SCX[2:0]);
                                          else gen_mask <= 8'hFF;

                                          if (sp_real_x > x_pos &&
sp_real_x < x_pos + 8) sp_mask <= 8'hFF >> (x_pos + 6 - sp_real_x);
```

```
                                        else if (sp_real_x + 8 > x_pos
&& sp_real_x + 8 < x_pos + 8) sp_mask <= 8'hFF << (x_pos - sp_real_x);
                                        else sp_mask <= 8'hFF;

                                        tts <= 1;
                                        /* For tall sprites */
                                        if (LCDC[2]) begin
                                              if (LY + 16 <
sp_y_buff[sp_ind] + 16) tts <= 0;
                                              else tts <= 1;
                                        end

                                  end else sp_ind <= sp_ind + 1;

                                  if (sp_ind == 9) begin
                                        sp_fetch_mode <= SP_RUN_BG;
                                        sp_tile_row[0] <= 0;
                                        sp_tile_row[1] <= 0;
                                  end
                            end
                            SP_TILE_LOAD: begin
                                  sp_mask <= sp_mask & gen_mask;
```

```
                                    `PPU_ADDR_SET(16'h8000 +
(BIG_LY_SCY_MOD << 1) + ({BIG_DATA_in[15:1], tts} << 4));
                                    sp_fetch_mode <= SP_ROW_1_LOAD;
                              end
                              SP_ROW_1_LOAD: begin
                                    sp_tile_row[0] <= PPU_DATA_in &
sp_mask;
                                    sp_fetch_mode <= SP_ROW_2_LOAD;
                                    `PPU_ADDR_INC(1);
                              end
                              SP_ROW_2_LOAD: begin
                                    sp_tile_row[1] <= PPU_DATA_in &
sp_mask;
                                    `PPU_ADDR_SET(`OAM_BASE_ADDR +
{8'b0, sp_off_buff[sp_ind]} + 3);
                                    sp_fetch_mode <= SP_RUN_BG;
                              end
                              SP_RUN_BG: begin
                                    sp_ind <= 0;
                                    curr_sp_flag <= PPU_DATA_in;
                                    bg_fetch_mode <= BG_TILE_NO_STORE;
                                    if (LY >= WY && ((x_pos + SCX >=
real_wx) || (x_pos + SCX + 8 > real_wx)) && LCDC[5]) begin
```

```
                                        if ((x_pos + SCX + 8 >
real_wx) && (x_pos + SCX < real_wx)) begin
                                // In the middle of a tile
                                        `PPU_ADDR_SET(bg_map_sel
+ x_tile_off + y_tile_off);
                                        w_check <= W_BG_RUN;
                                    end else begin
                                        `PPU_ADDR_SET(w_map_sel
+ x_w_off + y_w_off);
                                        w_check <= W_NO_EDGE;
                                    end
                                end else begin
                                        `PPU_ADDR_SET(bg_map_sel +
x_tile_off + y_tile_off);
                                        w_check <= W_NO_EDGE;
                                end
                                sp_fetch_mode <= SP_READY;
                            end
                            SP_READY: begin
                                // Purposefully ignores multiple
sprites existing overlapping. This is because I don't want to overwrite
data
                            end
```

```
                              default: begin
                              end
                        endcase

                        /* Pixel Mixing & Output Machine */
                        case (px_mix_mode)
                              MIX_LOAD: begin
      `ifndef MEM_CYCLE
                                    if (!ready_load && pixels_pushed >
0) begin
                                          pixels_pushed <= pixels_pushed
- 1;
                                    end
      `endif

                                    if (pixels_pushed == 4'hA) begin
                                          if ((sp_fetch_mode ==
SP_READY) && (bg_fetch_mode == BG_READY)) begin
                                                // load both buffers
into fifos
                                                bg_fifo_load <= 1;
                                                sp_fifo_load <= 1;
```

```
                                        // make sure we stop
pushing for a sec
                                        bg_fifo_go <= 0;
                                        sp_fifo_go <= 0;

                                        px_mix_mode <=
MIX_START;

                                        x_pos <= x_pos + 8;
                                    end
                                end

                                if (pixels_pushed == 0) begin
                                    pixels_pushed <= 4'hA;
                                    ready_load <= 1;
                                end
                            end
                            MIX_START: begin
                                bg_fifo_load <= 0;
                                sp_fifo_load <= 0;

                                bg_fifo_go <= 1;
                                sp_fifo_go <= 1;
```

```
                                pixels_pushed <= 8;
                                ready_load <= 0;

                                bg_fetch_mode <= BG_PAUSE;
                                sp_fetch_mode <= SP_SEARCH;
                                px_mix_mode <= MIX_LOAD;
                            end
                            default: begin
                            end
                endcase
                end
        end

end

endmodule

module PPU_SHIFT_REG
(
    input clk,
    input rst,
```

```
    input logic [7:0] data [1:0],
    input logic go,
    input logic load,
    output logic [1:0] q
);

logic [7:0] shift_reg [0:1];

always_ff @(posedge clk)
begin
    if (rst)
    begin
        shift_reg[0] <= 0;
        shift_reg[1] <= 0;
    end
    else if (load)
    begin
        shift_reg[0] <= data[0];
        shift_reg[1] <= data[1];
    end
    else
    begin
        if (go)
```

```
            begin
                shift_reg[0][7:1] <= shift_reg[0][6:0];
                shift_reg[0][0] <= 0;
                shift_reg[1][7:1] <= shift_reg[1][6:0];
                shift_reg[1][0] <= 0;
            end
        end
    end

    assign q = {shift_reg[1][7], shift_reg[0][7]};

    endmodule
```

don2_tb.cpp

```
Unset
    #include <iostream>
    #include <fstream>
    #include <verilated.h>
    #include <verilated_vcd_c.h>
    #include "VPPU3.h"

    typedef enum {PPU_H_BLANK, PPU_V_BLANK, PPU_SCAN, PPU_DRAW} PPU_STATES_t;
```

```
	#define BG_MAP_1_BASE_ADDR 0x9800
	#define BG_MAP_1_END_ADDR 0x9BFF

	#define OAM_BASE_ADDR 0xFE00

	#define TILE_BASE 0x8000

	int main(int argc, const char ** argv, const char ** env)
	{
		int time, exit_code, last_clk, row_1_loaded, cycles, tile_toggle,
tile_row_cnt;
		char LCDC, tile_1[2], tile_2[2], tile_3, sprite_data[4];
		VPPU3 *dut;
		std::ofstream f("tb_gen.ppm");

		if (!f.is_open()) {
			std::cerr << "Error opening ppm" << std::endl;
			exit_code = -1;
			return exit_code;
		}
		f << "P2\n160 144\n4\n";

		tile_row_cnt = tile_toggle = cycles = last_clk = time = exit_code
= row_1_loaded = 0;
```

```
            tile_1[0] = 0xFF;   // 1111_1111
            tile_1[1] = 0x00;   // 0000_0000

            tile_2[0] = 0xAA;   // 1010_1010
            tile_2[1] = 0x55;   // 0101_0101

            tile_3 = 0x00;

            sprite_data[0] = 16 + (8 * 1);          // (y-value)        16 +
pos
            sprite_data[1] = 8 + (8 * 1);           // (x-value)         8 +
pos
            sprite_data[2] = 2;                             // (tile no.)
            sprite_data[3] = 0x00;                          // flags (prio and
other things)

            Verilated::commandArgs(argc, argv);

            dut = new VPPU3;    // Instantiate the ppu module

            Verilated::traceEverOn(true);
            VerilatedVcdC *tfp = new VerilatedVcdC;

            dut->trace(tfp, 99);
            tfp->open("ppu3.vcd");
```

```
        LCDC = 0xFB;
        dut->WR = 1;
        dut->ADDR = 0xFF40;
        dut->MMIO_DATA_out = LCDC;


        dut->PPU_DATA_in = 0x0;    // JUNK
        for (time = 0 ; time < 1368000 ; time += 10) {
               // Simulate a 50 MHz clock
               last_clk = dut->clk;
// used to detect negedge
        dut->clk = ((time % 20) >= 10) ? 1 : 0;
               if (dut->clk == 1)
               cycles++;


        // Triggers rst
        dut->rst = (time == 30) ? 1 : 0;
        if (dut->rst == 1)
               cycles = 0;


        if (dut->clk == 1) {
                     if (dut->PPU_ADDR == OAM_BASE_ADDR)
                            dut->PPU_DATA_in = sprite_data[0];
                     if (dut->PPU_ADDR == OAM_BASE_ADDR + 1)
                            dut->PPU_DATA_in = sprite_data[1];
                     if (dut->PPU_ADDR == OAM_BASE_ADDR + 2)
                            dut->PPU_DATA_in = sprite_data[2];
```

```
                        if (dut->PPU_ADDR == OAM_BASE_ADDR + 3)
                                dut->PPU_DATA_in = sprite_data[3];
                }

                if (dut->PPU_MODE == PPU_DRAW && dut->clk == 1) {
                        if (dut->PPU_ADDR >= BG_MAP_1_BASE_ADDR &&
dut->PPU_ADDR < BG_MAP_1_END_ADDR) {
                                dut->PPU_DATA_in = !(tile_row_cnt % 2 == 0);
                        // tells it to pull from file 1
                                row_1_loaded = 0;
                        }

                        else if ((TILE_BASE + (16*0) <= dut->PPU_ADDR) &&
(TILE_BASE + (16*1) > dut->PPU_ADDR)) {
                                        if (!row_1_loaded) {
                                                tile_row_cnt++;
                                                dut->PPU_DATA_in = tile_2[0];
                                                row_1_loaded = 1;
                                        }
                                        else
                                                dut->PPU_DATA_in = tile_2[1];
                        }

                        else if ((TILE_BASE + (16*1) <= dut->PPU_ADDR) &&
(TILE_BASE + (16*2) > dut->PPU_ADDR)) {
                                        if (!row_1_loaded) {
```

```
                                        tile_row_cnt++;
                                        dut->PPU_DATA_in = tile_1[0];
                                        row_1_loaded = 1;
                                }
                                else
                                        dut->PPU_DATA_in = tile_1[1];
                        }

                        else if ((TILE_BASE + (16*2) <= dut->PPU_ADDR) &&
(TILE_BASE + (16*3) > dut->PPU_ADDR)) {
                                dut->PPU_DATA_in = tile_3;
                        }
                }

                if (dut->PPU_MODE == PPU_H_BLANK) {
                        cycles = 0;
                        tile_row_cnt = 0;
                        dut->PPU_DATA_in = 0x0;
                }

        dut->eval();                    // Run the simulation for a cycle
        tfp->dump(time);                // Write the VCD file for this
cycle

        if (dut->clk != last_clk && dut->clk == 1) {   // on posedge of
clock
```

```
                /* Writes to ppm file */
                if ((int)dut->PX_valid == 1) {
                        f << (int) dut->PX_OUT << " ";
                }
            }
        }

            tfp->close(); // Stop dumping the VCD file
            delete tfp;

            dut->final(); // Stop the simulation
            delete dut;

            f.close();
            return exit_code;
    }
```

don3_tb.cpp

```
Unset
    #include <iostream>
    #include <fstream>
    #include <verilated.h>
    #include <verilated_vcd_c.h>
    #include "VPPU3.h"
```

```
	typedef enum {PPU_H_BLANK, PPU_V_BLANK, PPU_SCAN, PPU_DRAW} PPU_STATES_t;

	#define BG_MAP_1_BASE_ADDR 0x9800
	#define BG_MAP_2_BASE_ADDR 0x9C00

	#define BG_MAP_END_ADDR 0x9FFF

	#define OAM_BASE_ADDR 0xFE00

	#define TILE_BASE_ADDR 0x8000
	#define TILE_END_ADDR  0x97FF

	int main(int argc, const char ** argv, const char ** env)
	{
		int time, exit_code, last_clk, i;
		char tile_1[2], tile_2[2], tile_3, sprite_data[4], OAM_MEM[160],
BG_MAP[2048], TILE_MAP[6144], update_reg;
		VPPU3 *dut;
		std::ofstream f("tb_gen.ppm");

		if (!f.is_open()) {
			std::cerr << "Error opening ppm" << std::endl;
			exit_code = -1;
			return exit_code;
		}
```

```
            f << "P2\n160 144\n4\n";


          last_clk = time = exit_code == 0;


            for (i = 0; i < 1024; i++)
                  BG_MAP[i] = i % 2;


            BG_MAP[32] = 1;
            for (i = BG_MAP_2_BASE_ADDR - BG_MAP_1_BASE_ADDR; i < 2048; i++)
                  BG_MAP[i] = 3;


            for (i = 0; i < 16; i += 2) {
                  TILE_MAP[i] = 0xFF;                                  //
1111_1111
                  TILE_MAP[i+1] = 0x00;                                //
0000_0000


                  TILE_MAP[(16 * 1) + i] = 0xAA;                // 1010_1010
                  TILE_MAP[(16 * 1) + i + 1] = 0x55;            // 0101_0101


                  TILE_MAP[(16 * 17) + i] = 0xFF;               // "sprite"
                  TILE_MAP[(16 * 17) + i + 1] = 0xFF;


                  TILE_MAP[(16 * 16) + i] = 0xFF;               // "sprite"
                  TILE_MAP[(16 * 16) + i + 1] = 0xFF;
            }
```

```
            for (i = 0; i < 16; i += 4) {
                  TILE_MAP[(16 * 2) + i] = 0x96;                        //
1001_0110
                  TILE_MAP[(16 * 2) + i + 1] = 0x69;                    //
0110_1001

                  TILE_MAP[(16 * 2) + i + 2] = 0x69;                    //
1001_0110
                  TILE_MAP[(16 * 2) + i + 3] = 0x96;                    //
0110_1001

                  TILE_MAP[(16 * 3) + i] = 0xAA;                        //
1001_0110
                  TILE_MAP[(16 * 3) + i + 1] = 0x55;                    //
0110_1001

                  TILE_MAP[(16 * 3) + i + 2] = 0x55;                    //
1001_0110
                  TILE_MAP[(16 * 3) + i + 3] = 0xAA;                    //
0110_1001
            }

            for (i = 0; i < 8; i += 4) {
                  OAM_MEM[0] = 16 + (8 * 3);       // (y-value)        16 +
pos
```

```
                    OAM_MEM[1] = 8 + (8 * 0);        // (x-value)         8 +
pos

                    OAM_MEM[4] = 16 + (8 * 1);       // (y-value)        16 +
pos
                    OAM_MEM[5] = 8 + (8 * 2);        // (x-value)         8 +
pos

                    OAM_MEM[i+2] = 0x11;                          // (tile no.)
                    OAM_MEM[7] = 0xFF >> 1;                 // flags (prio and
other things)
             }

             Verilated::commandArgs(argc, argv);

             dut = new VPPU3;    // Instantiate the ppu module

             Verilated::traceEverOn(true);
             VerilatedVcdC *tfp = new VerilatedVcdC;

             dut->trace(tfp, 99);
             tfp->open("ppu3.vcd");

             dut->PPU_DATA_in = 0x0;    // JUNK

             for (time = 0 ; time < 2272110 ; time += 10) {
```

```
                    // Simulate a 50 MHz clock
                    last_clk = dut->clk;
      // used to detect posedge
            dut->clk = ((time % 20) >= 10) ? 1 : 0;


            // Triggers rst
            dut->rst = (time == 30) ? 1 : 0;


            // only on posedge
            if (dut->clk == 1) {
                  if (time > 40) { // after rst
                        if (!(update_reg & 0x1)) {                    //
LCDC
                                    dut->WR = 1;
                                    dut->ADDR = 0xFF40;
                                    dut->MMIO_DATA_out = 0xF3;
// 1111_0011
                                    // dut->MMIO_DATA_out = 0xFF;
                                    update_reg |= 0x1;
                        } else if (!(update_reg & 0x02)) {      // WY
                              dut->WR = 1;
                              dut->ADDR = 0xFF4A;
                              dut->MMIO_DATA_out = 8 * 8;
                              update_reg |= 0x2;
                        } else if (!(update_reg & 0x04)) {      // WX
                              dut->WR = 1;
```

```
                                dut->ADDR = 0xFF4B;
                                dut->MMIO_DATA_out = 7 + 16 - 2;
                                update_reg |= 0x4;
                        } else
                                dut->WR = 0;
                }



                        if (dut->PPU_ADDR >= OAM_BASE_ADDR && dut->PPU_ADDR <
OAM_BASE_ADDR + 160)
                                dut->PPU_DATA_in = OAM_MEM[dut->PPU_ADDR -
OAM_BASE_ADDR];
                        else if (dut->PPU_ADDR >= BG_MAP_1_BASE_ADDR &&
dut->PPU_ADDR <= BG_MAP_END_ADDR)
                                dut->PPU_DATA_in = BG_MAP[dut->PPU_ADDR -
BG_MAP_1_BASE_ADDR];
                        else if (dut->PPU_ADDR >= TILE_BASE_ADDR &&
dut->PPU_ADDR <= TILE_END_ADDR)
                                dut->PPU_DATA_in = TILE_MAP[dut->PPU_ADDR -
TILE_BASE_ADDR];
                }



                dut->eval();                    // Run the simulation for a cycle
                tfp->dump(time);                // Write the VCD file for this
cycle
```

```
            if (dut->clk != last_clk && dut->clk == 1) {   // on posedge of
clock
                  /* Writes to ppm file */
                  if ((int)dut->PX_valid == 1)
                        f << (int) dut->PX_OUT << " ";
            }
          }

            tfp->close(); // Stop dumping the VCD file
            delete tfp;

            dut->final(); // Stop the simulation
            delete dut;

            f.close();
            return exit_code;
      }
```

donovan_tb_ppu.cpp

```
Unset
      #include <iostream>
      #include <fstream>
      #include <verilated.h>
```

```
	#include <verilated_vcd_c.h>
	#include "VPPU3.h"

	typedef enum {H_BLANK, V_BLANK, SCAN, DRAW} PPU_STATES_t;

	#define BG_MAP_1_BASE_ADDR 0x9800
	#define BG_MAP_1_END_ADDR 0x9BFF

	#define TILE_BASE 0x8000

	int main(int argc, const char ** argv, const char ** env)
	{
		int time, offset, exit_code, row_code, cycles, tile_toggle,
last_clk, load;
		char LCDC, last_px_state, tile_1[2], tile_2[2];
		VPPU3 *dut;
		std::ofstream f("tb_gen.ppm");

		if (!f.is_open()) {
			std::cerr << "Error opening ppm" << std::endl;
			exit_code = -1;
			return exit_code;
		}
		f << "P2\n160 144\n4\n";
```

```
            load = last_clk = tile_toggle = cycles = row_code = offset =
exit_code = 0;
            last_px_state = 0;

            tile_1[0] = 0xFF;   // 1111_1111
            tile_1[1] = 0x00;   // 0000_0000

            tile_2[0] = 0xAA;   // 1010_1010
            tile_2[1] = 0x55;   // 0101_0101

            Verilated::commandArgs(argc, argv);

            dut = new VPPU3;    // Instantiate the ppu module

            Verilated::traceEverOn(true);
            VerilatedVcdC *tfp = new VerilatedVcdC;

            dut->trace(tfp, 99);
            tfp->open("ppu3.vcd");

            LCDC = 0xFF;
            dut->WR = 1;
            dut->ADDR = 0xFF40;
            dut->MMIO_DATA_out = LCDC;

            dut->PPU_DATA_in = 0xFF;   // JUNK
```

```
            for (time = 0 ; time < 1368000 ; time += 10) {
                  last_px_state = dut->PX_valid;
                  last_clk = dut->clk;
            dut->clk = ((time % 20) >= 10) ? 1 : 0;       // Simulate a 50
MHz clock
                  if (dut->PPU_MODE == H_BLANK)
                        cycles = 0;
            else if (dut->clk == 1)
                  cycles++;

            dut->rst = (time == 30) ? 1 : 0;              // pulses rst
            if (dut->rst == 1)
                  cycles = 0;

                  if (dut->PPU_MODE == DRAW && dut->clk == 1) {
                        if (dut->PPU_ADDR >= BG_MAP_1_BASE_ADDR &&
dut->PPU_ADDR < BG_MAP_1_END_ADDR) {
                              dut->PPU_DATA_in = tile_toggle;
                              load = 0;
                        }
                        else if (dut->PPU_ADDR >= TILE_BASE && cycles > 81)
                              if (load == 0) {
                                    dut->PPU_DATA_in = tile_2[0];
                                    load = 1;
                              } else if (load == 1)
                                    dut->PPU_DATA_in = tile_2[1];
```

```
                }
            dut->eval();                        // Run the simulation for a cycle
            tfp->dump(time);                    // Write the VCD file for this
cycle

            if (dut->clk != last_clk && dut->clk == 1) {   // on posedge of
clock
                if ((int)dut->PX_valid == 1) {
                    f << (int) dut->PX_OUT << " ";
                    // std::cout << (int) dut->PX_valid << " " << (int)
dut->PX_OUT << std::endl;
                }
            }
        }

            tfp->close(); // Stop dumping the VCD file
            delete tfp;

            dut->final(); // Stop the simulation
            delete dut;

            f.close();
            return exit_code;
    }
```

Testbench

Makefile

```
Unset
	SVFILE = ../GameBoy1/GameBoy_RTL_Qsys_submit/PPU3.sv ../PPU3.sv


	.PHONY default

	default: PPU3.vcd


	# Quickly check the System Verilog files for errors


	lint :

		for file in $(SVFILE); do \

		verilator --lint-only -Wall $$file; done



	#Create and Run Verilator Simulations


	PPU3.vcd : obj_dir/VPPU3

		obj_dir/VPPU3



	obj_dir/VPPU3 : PPU3.sv PPU_tb.cpp

		verilator -trace -Wall -cc PPU3.sv -exe PPU3.cpp -top-module PPU3

		cd obj_dir && make -j -f VPPU3.mk


	.PHONY clean
```

```
clean:
	rm -rf obj_dir *.vcd
```

PPU3.sv

```
Unset
`timescale 1ns / 1ns

//Authors: Nicolas Alarcon, Claire Cizdziel, Donovan Sproule

/* WARNING: DOES NOT HAVE INTERRUPTS OR ANY EXTERNAL OUTPUTS IMPLEMENTED
*/

module PPU3
(
    input logic clk,
    input logic rst,

    /* System access for PPU internal registers */
    input logic [15:0] ADDR,
    input logic WR,
    input logic RD,
    input logic [7:0] MMIO_DATA_out,
    output logic [7:0] MMIO_DATA_in,
```

```
    /* Interrupts */
    output logic IRQ_V_BLANK,
    output logic IRQ_LCDC,

    output logic [1:0] PPU_MODE,

    /* VRAM access for PPU */
    output logic PPU_RD,
    output logic [15:0] PPU_ADDR,
    input logic [7:0] PPU_DATA_in,

    output logic [1:0] PX_OUT,
    output logic PX_valid
);

`define OAM_BASE_ADDR 16'hFE00
`define OAM_END_ADDR 16'hFE9F
`define NO_BOOT 0

`define PPU_ADDR_INC(x) PPU_ADDR <= PPU_ADDR + x;

logic sprite_in_range;
logic [7:0] LY, x_pos;  // x-pos in range [0, 159]
logic [15:0] current_offset;
logic [3:0] sprites_loaded;
```

```
        logic sprite_found;

        logic [8:0] cycles;

        // logic [1:0] PPU_MODE;

        logic [7:0] sprite_y_buff [9:0];
        logic [7:0] sprite_x_buff [9:0];
        logic [7:0] sprite_offset_buff [9:0];          // stores start of sprite
entries in OAM (entry + 2 => for tile no.)

        logic sprite_shift_go;
        logic sprite_shift_load;
        logic [1:0] sprite_shift_output;
        logic [7:0] sprite_tile_row [1:0];

        PPU_SHIFT_REG sprite_fifo(.clk(clk), .rst(rst), .data(sprite_tile_row),
.go(sprite_shift_go), .load(sprite_shift_load), .q(sprite_shift_output));

        logic bg_shift_go;
        logic bg_shift_load;
        logic [1:0] bg_shift_output;
        logic [7:0] bg_tile_row [1:0];

        PPU_SHIFT_REG bg_fifo(.clk(clk), .rst(rst), .data(bg_tile_row),
.go(bg_shift_go), .load(bg_shift_load), .q(bg_shift_output));
```

```
        typedef enum bit [1:0] {H_BLANK, V_BLANK, SCAN, DRAW} PPU_STATES_t;

        /* External registers */
        logic [7:0] LCDC, STAT, SCX, SCY, LYC, DMA, BGP, OBP0, OBP1, WX, WY; //
Register alias

        logic [7:0] FF40;
        assign LCDC = FF40;

        logic [7:0] FF41;
        assign STAT = FF41;

        logic [7:0] FF42;
        assign SCY = FF42;

        logic [7:0] FF43;
        assign SCX = FF43;

        logic [7:0] FF44;

        logic [7:0] FF45;
        assign LYC = FF45;

        logic [7:0] FF46;
        assign DMA = FF46;
```

```
        logic [7:0] FF47;
        assign BGP = FF47;

        logic [7:0] FF48;
        assign OBP0 = {FF48[7:2], 2'b00}; // Last 2 bits are not used

        logic [7:0] FF49;
        assign OBP1 = {FF49[7:2], 2'b00};

        logic [7:0] FF4A;
        assign WY = FF4A;

        logic [7:0] FF4B;
        assign WX = FF4B;

        /* ----------------- End of declarations */

        /* Register Assignment
         *
         *     if a register memory address is being indexed it gets updated here
         *  -- As a warning the following may introduce timing oddities compared
to other group
         */
        always_ff @(posedge clk)
        begin
```

```
            if (rst)
            begin
                FF40 <= `NO_BOOT ? 8'h91 : 0;
                FF41 <= 0;
                FF42 <= 0;
                FF43 <= 0;
                FF45 <= 0;
                FF46 <= 0;
                FF47 <= `NO_BOOT ? 8'hFC : 0;
                FF48 <= `NO_BOOT ? 8'hFF : 0;
                FF49 <= `NO_BOOT ? 8'hFF : 0;
                FF4A <= 0;
                FF4B <= 0;
            end
            else
            begin
                FF40 <= (WR && (ADDR == 16'hFF40)) ? MMIO_DATA_out : FF40;
                        FF41 <= (WR && (ADDR == 16'hFF41)) ?
{MMIO_DATA_out[7:3], FF41[2:0]} : {FF41[7:3], LYC == LY, PPU_MODE};
                        FF42 <= (WR && (ADDR == 16'hFF42)) ? MMIO_DATA_out :
FF42;
                        FF43 <= (WR && (ADDR == 16'hFF43)) ? MMIO_DATA_out :
FF43;
                        FF45 <= (WR && (ADDR == 16'hFF45)) ? MMIO_DATA_out :
FF45;
```

```
                        FF46 <= (WR && (ADDR == 16'hFF46)) ? MMIO_DATA_out :
FF46;
                        FF47 <= (WR && (ADDR == 16'hFF47)) ? MMIO_DATA_out :
FF47;
                        FF48 <= (WR && (ADDR == 16'hFF48)) ? MMIO_DATA_out :
FF48;
                        FF49 <= (WR && (ADDR == 16'hFF49)) ? MMIO_DATA_out :
FF49;
                        FF4A <= (WR && (ADDR == 16'hFF4A)) ? MMIO_DATA_out :
FF4A;
                        FF4B <= (WR && (ADDR == 16'hFF4B)) ? MMIO_DATA_out :
FF4B;
           end
       end


       /*
        * If we detect a memory request we return back the current
        * state of the register
        */
       always_comb   // doesn't wait for a posedge clk
       begin
           case (ADDR)
               16'hFF40: MMIO_DATA_in = FF40;
               16'hFF41: MMIO_DATA_in = {1'b1, FF41[6:0]};
               16'hFF42: MMIO_DATA_in = FF42;
               16'hFF43: MMIO_DATA_in = FF43;
```

```
                16'hFF44: MMIO_DATA_in = FF44;
                16'hFF45: MMIO_DATA_in = FF45;
                16'hFF46: MMIO_DATA_in = FF46;
                16'hFF47: MMIO_DATA_in = FF47;
                16'hFF48: MMIO_DATA_in = FF48;
                16'hFF49: MMIO_DATA_in = FF49;
                16'hFF4A: MMIO_DATA_in = FF4A;
                16'hFF4B: MMIO_DATA_in = FF4B;
                default : MMIO_DATA_in = 8'hFF;
            endcase
        end

        /* -- State Switching machine -- */
        always_ff @(posedge clk) begin
            if (rst) begin
                            x_pos <= 0;
                            cycles <= 0;
                            LY <= 0;
                            PPU_ADDR <= `OAM_BASE_ADDR;
                            PPU_MODE <= SCAN;
            end else if (LCDC[7]) begin
               cycles <= cycles + 1;
                     /* -- Following block happens on a per scanline basis (456
cycles per line) -- */
                case (PPU_MODE)
                    SCAN: begin
```

```
                        if (PPU_ADDR > `OAM_END_ADDR)
                                    PPU_MODE <= DRAW;
                               if (LY >= 144)
                                    PPU_MODE <= V_BLANK;
                        end
                    DRAW:
                        LY <= LY;
                        // if x_pos is off-screen switch to H_blank
                    H_BLANK:
                        if (cycles >= 455) begin         // we reached the
end of the scanline
                                    LY <= LY + 1;
                                    x_pos <= 0;
                                    PPU_MODE <= SCAN;
                                    cycles <= 0;
                               end
                    V_BLANK:                                          // not
technically necessary but here for completeness
                               PPU_MODE <= H_BLANK;
                endcase
            end
        end


        assign sprite_in_range = ((LY + 16 >= PPU_DATA_in) &&
                                                    (LY + 16 < PPU_DATA_in +
(8 << LCDC[2])));
```

```
	assign current_offset = PPU_ADDR - `OAM_BASE_ADDR;
	/* -- OAM Scan State Machine -- */
	always_ff @(posedge clk) begin
		if (rst || PPU_MODE == H_BLANK) begin
			sprites_loaded <= 0;
			sprite_found <= 0;
		end else if (PPU_MODE == SCAN) begin
			if (!cycles[0])	begin			// forces
alternating clock cycles
				if (sprite_in_range && sprites_loaded < 10) begin
					sprites_loaded <= sprites_loaded + 1;
					sprite_y_buff[sprites_loaded] <= PPU_DATA_in;
					sprite_offset_buff[sprites_loaded] <=
current_offset[7:0];
					sprite_found <= 1;
					`PPU_ADDR_INC(1);
	// jumps to x-byte
				end else `PPU_ADDR_INC(4);
	// jumps to next sprite in OAM
			end else begin
				if (sprite_found) begin
					sprite_x_buff[sprites_loaded - 1] <=
PPU_DATA_in;
					`PPU_ADDR_INC(3);
	// jumps to next sprite in OAM
				end
```

```
                    sprite_found <= 0;
                end


        end
    end

    /* BG Draw Machine */
    always_ff @(posedge clk) begin
        // sprite-staging mode:
        //     1 - if sprite-x is in range grab it
        //     2 - pause bg-staging
        //     3 - fetches sprite tile from buffer
        //     4 - same as 2 & 3 in bg-stage mode
        //
        // bg-staging mode:
        //     1 - grabs the current tile (use x-pos as an offset
        //           from the base)
        //     2 - grabs the corresponding row of byte from the tile
        //           saved
        //     3 - grabs the next row so that our color can be
        //           encoded
        //     4 - decode and push the row into the FIFO
        //
        // drawing mode:
        //     1 - apply the bg fifo mask for SCX
        //     2 - grab pixel and do mixing
```

```
        //     3 - push to LCD
        //     4 - increment x-pos
        //     5 - check if we've hit the window
        //
        // if x-pos == 160 then move to h-blank
    end


    endmodule


    module PPU_SHIFT_REG
    (
        input clk,
        input rst,
        input logic [7:0] data [1:0],
        input logic go,
        input logic load,
        output logic [1:0] q
    );


    logic [7:0] shift_reg [0:1];


    always_ff @(posedge clk)
    begin
        if (rst)
        begin
            shift_reg[0] <= 0;
```

```
            shift_reg[1] <= 0;
        end
        else if (load)
        begin
            shift_reg[0] <= data[0];
            shift_reg[1] <= data[1];
        end
        else
        begin
            if (go)
            begin
                shift_reg[0][7:1] <= shift_reg[0][6:0];
                shift_reg[0][0] <= 0;
                shift_reg[1][7:1] <= shift_reg[1][6:0];
                shift_reg[1][0] <= 0;
            end
        end
    end

    assign q = {shift_reg[1][7], shift_reg[0][7]};

    endmodule
```

tb_ppu.cpp

```
Unset
	#include <iostream>

	#include "VPPU3.h"

	#include <verilated.h>

	#include <verilated_vcd_c.h>

	#include <cstdio>


	// input logic [15:0] ADDR,

	// input logic WR,

	// input logic [7:0] MMIO_DATA_out,

	// output logic [7:0] MMIO_DATA_in,


	int main(int argc, const char ** argv, const char ** env) {

		char Tile_Map[1024];


		for (int i = 0; i<8; i++) Tile_Map[240 + i] = 1;


		int i, time;

		OAM_Ent ent;

		int offset, exit_code;

		char LCDC;

		VPPU3 *dut;

		int tile_req;

		offset = exit_code = 0;

		char line_buf[160] = {0};
```

```
            char line_buf_cnt = 0;


            Verilated::commandArgs(argc, argv);


            dut = new VPPU3;    // Instantiate the ppu module


            Verilated::traceEverOn(true);
            VerilatedVcdC *tfp = new VerilatedVcdC;


            dut->trace(tfp, 99);
            tfp->open("ppu3.vcd");


            LCDC = 0xFF;
            0x9C00
            dut->WR = 1;
            dut->ADDR = 0xFF40;
            dut->MMIO_DATA_out = LCDC;


            for (time = 0 ; time < 10000 ; time += 10) {
            dut->clk = ((time % 20) >= 10) ? 1 : 0;        // Simulate a 50
MHz clock


            dut->rst = (time == 30) ? 1 : 0; // pulses rst
                   dut->PPU_DATA_in = 0;


                   //for map 2 and 8000 addressing method, due to LCDC = FF
```

```
                    tile_req = (dut->PPU_ADDR - 0x9C00)/16;
                    if (tile_req == 0) dut->PPU_DATA_in = 0;
                    if (tile_req == 1) dut->PPU_DATA_in = 1;

                    if (dut->PX_valid) line_buf[line_buf_cnt = (line_buf_cnt +
1)%160] = dut->PX_OUT;
                    if (line_buf_cnt == 159) {
                           for (int i = 0; i<160; i++) printf("%d",
line_buf[i]);
                           printf(" \n");
                    }
             dut->eval();                    // Run the simulation for a cycle
             tfp->dump(time);                // Write the VCD file for this
cycle
          }

             tfp->close(); // Stop dumping the VCD file
             delete tfp;

             dut->final(); // Stop the simulation
             delete dut;

             return exit_code;
       }
```

Sourced and/or Slightly Modified Code

GB_Z80_ALU.vh

```
Unset
      /* This are the ALU OPCODEs */

      `ifndef GB_Z80_ALU_H

        `define GB_Z80_ALU_H


      typedef enum

      {

          ALU_NOP,

          ALU_ADD,

          ALU_ADC,

          ALU_SUB,

          ALU_SBC,

          ALU_CP,

          ALU_AND,

          ALU_OR,

          ALU_XOR,

          ALU_INC,

          ALU_DEC,

          ALU_CPL,

          ALU_BIT,
```

```
        ALU_SET,
        ALU_RES,
        ALU_INC16, // 16 bit alu operation
        ALU_DEC16, // 16 bit alu operation
        ALU_DAA,

        /* Shifter Operations */
        SHIFTER_SWAP,
        SHIFTER_RLC,
        SHIFTER_RL,
        SHIFTER_RRC,
        SHIFTER_RR,
        SHIFTER_SLA,
        SHIFTER_SRA,
        SHIFTER_SRL

    } GB_Z80_ALU_OPCODE;

    `endif
```

GB_Z80_CPU.vh

```
Unset
	/* Internal Registers */
	`ifndef GB_Z80_CPU_H
	  `define GB_Z80_CPU_H

	typedef struct
	{
	    logic [7:0] A; logic [7:0] F;  // AF, F for Flag
	    logic [7:0] B; logic [7:0] C;  // BC, nn
	    logic [7:0] D; logic [7:0] E;  // DE, nn
	    logic [7:0] H; logic [7:0] L;  // HL, nn

	    logic [7:0] T; logic [7:0] X;  // Temp Result

	    logic [7:0] SPh, SPl;          // Stack Pointer
	    logic [15:0] PC;               // Program Counter
	} GB_Z80_REG;

	`define WR_nn(n1, n2) \
	    begin \
	        WR_NEXT = 1; \
	        ADDR_NEXT = {CPU_REG.``n1, CPU_REG.``n2}; \
	    end
```

```
`define WR_FFn(n) \
    begin \
        WR_NEXT = 1; \
        ADDR_NEXT = {8'hFF, CPU_REG.``n}; \
    end

`define RD_nn(n1, n2) \
    begin \
        RD_NEXT = 1; \
        ADDR_NEXT = {CPU_REG.``n1, CPU_REG.``n2}; \
    end

`define RD_FFn(n) \
    begin \
        RD_NEXT = 1; \
        ADDR_NEXT = {8'hFF, CPU_REG.``n}; \
    end

`define LD_n_n(n1, n2) \
    begin \
        CPU_REG_NEXT.``n1 = CPU_REG.``n2; \
    end
```

```
`define INC_n(n) \
    begin \
        ALU_OPCODE = ALU_INC; \
        ALU_OPD1_L = CPU_REG.``n; \
        CPU_REG_NEXT.``n = ALU_RESULT_L; \
        CPU_REG_NEXT.F = ALU_STATUS; \
    end


`define DEC_n(n) \
    begin \
        ALU_OPCODE = ALU_DEC; \
        ALU_OPD1_L = CPU_REG.``n; \
        CPU_REG_NEXT.``n = ALU_RESULT_L; \
        CPU_REG_NEXT.F = ALU_STATUS; \
    end


// {n1, n2}
`define INC_nn(n1, n2) \
    begin \
        ALU_OPCODE = ALU_INC16; \
        ALU_OPD1_L = CPU_REG.``n2; \
        ALU_OPD2_L = CPU_REG.``n1; \
        CPU_REG_NEXT.``n1 = ALU_RESULT_H; \
```

```
            CPU_REG_NEXT.``n2 = ALU_RESULT_L; \
        end
    `define DEC_nn(n1, n2) \
        begin \
            ALU_OPCODE = ALU_DEC16; \
            ALU_OPD1_L = CPU_REG.``n2; \
            ALU_OPD2_L = CPU_REG.``n1; \
            CPU_REG_NEXT.``n1 = ALU_RESULT_H; \
            CPU_REG_NEXT.``n2 = ALU_RESULT_L; \
        end


    `define ADDL_n(n) \
        begin \
            ALU_OPCODE = ALU_ADD; \
            ALU_OPD2_L = CPU_REG.L; \
            ALU_OPD1_L = CPU_REG.``n; \
            CPU_REG_NEXT.L = ALU_RESULT_L; \
            CPU_REG_NEXT.F = ALU_STATUS; \
        end


    `define ADCH_n(n) \
        begin \
            ALU_OPCODE = ALU_ADC; \
```

```
            ALU_OPD2_L = CPU_REG.H; \
            ALU_OPD1_L = CPU_REG.``n; \
            CPU_REG_NEXT.H = ALU_RESULT_L; \
            CPU_REG_NEXT.F = ALU_STATUS; \
        end


    `define ALU_A_op_n(op, n) \
        begin \
            ALU_OPCODE = ALU_``op; \
            ALU_OPD2_L = CPU_REG.A; \
            ALU_OPD1_L = CPU_REG.``n; \
            CPU_REG_NEXT.A = ALU_RESULT_L; \
            CPU_REG_NEXT.F = ALU_STATUS; \
        end


    `define ALU_A_op_Data_in(op) \
        begin \
            ALU_OPCODE = ALU_``op; \
            ALU_OPD2_L = CPU_REG.A; \
            ALU_OPD1_L = DATA_in; \
            CPU_REG_NEXT.A = ALU_RESULT_L; \
            CPU_REG_NEXT.F = ALU_STATUS; \
        end
```

```
`define ALU_op_n(op, n) \
    begin \
        ALU_OPCODE = ALU_``op; \
        ALU_OPD2_L = CPU_REG.A; \
        ALU_OPD1_L = CPU_REG.``n; \
        //CPU_REG_NEXT.A = ALU_RESULT_L; \
        CPU_REG_NEXT.F = ALU_STATUS; \
    end



`define ALU_BIT_b_n(b, n) \
    begin \
        ALU_OPCODE = ALU_BIT; \
        ALU_OPD2_L = ``b; \
        ALU_OPD1_L = CPU_REG.``n; \
        CPU_REG_NEXT.F = ALU_STATUS; \
    end


`define ALU_SETRST_op_b_n(op, b, n) \
    begin \
        ALU_OPCODE = ALU_``op; \
        ALU_OPD2_L = ``b; \
```

```
            ALU_OPD1_L = CPU_REG.``n; \
            CPU_REG_NEXT.``n = ALU_RESULT_L; \
        end


    `define ALU_op_Data_in(op) \
        begin \
            ALU_OPCODE = ALU_``op; \
            ALU_OPD2_L = CPU_REG.A; \
            ALU_OPD1_L = DATA_in; \
            //CPU_REG_NEXT.A = ALU_RESULT_L; \
            CPU_REG_NEXT.F = ALU_STATUS; \
        end


    `define SHIFTER_op_n(op, n) \
        begin \
            ALU_OPCODE = SHIFTER_``op; \
            ALU_OPD1_L = CPU_REG.``n; \
            CPU_REG_NEXT.``n = ALU_RESULT_L; \
            CPU_REG_NEXT.F = ALU_STATUS; \
        end


     `define DAA \
        begin \
            ALU_OPCODE = ALU_DAA; \
            ALU_OPD1_L = CPU_REG.A; \
            CPU_REG_NEXT.A = ALU_RESULT_L; \
```

```
                CPU_REG_NEXT.F = ALU_STATUS; \
            end
        `define DO_JPR8 {1'b0, CPU_REG.PC} + {3'b0, DATA_in[6:0]} - {1'b0,
DATA_in[7], 7'b000_0000}

        // H and C are based on Unsigned ! added to SPl
        `define ADD_SPT \
            begin \
                {CPU_REG_NEXT.SPh, CPU_REG_NEXT.SPl} = {1'b0, CPU_REG.SPh,
CPU_REG.SPl} + {3'b0, CPU_REG.T[6:0]} - {1'b0, CPU_REG.T[7], 7'b000_0000}; \
                CPU_REG_NEXT.F = \
                { \
                    2'b00, \
                    (({1'b0, CPU_REG.SPl[3:0]} + {1'b0, CPU_REG.T[3:0]}) >
5'h0F), \
                    (({1'b0, CPU_REG.SPl[7:0]} + {1'b0, CPU_REG.T[7:0]}) >
9'h0FF), \
                    CPU_REG.F[3:0] \
                }; \
            end

       `define LD_HL_SPR8 \
            begin \
                {CPU_REG_NEXT.H, CPU_REG_NEXT.L} = {1'b0, CPU_REG.SPh,
CPU_REG.SPl} + {3'b0, CPU_REG.T[6:0]} - {1'b0, CPU_REG.T[7], 7'b000_0000}; \
                CPU_REG_NEXT.F = \
```

```
                { \
                    2'b00, \
                     (({1'b0, CPU_REG.SPl[3:0]} + {1'b0, CPU_REG.T[3:0]}) >
5'h0F), \
                     (({1'b0, CPU_REG.SPl[7:0]} + {1'b0, CPU_REG.T[7:0]}) >
9'h0FF), \
                    CPU_REG.F[3:0] \
                }; \
            end


        `endif
```

GB_Z80_DECODER.vh

```
Unset
        `ifndef GB_Z80_DECODER_H
          `define GB_Z80_DECODER_H


          typedef enum
          {
            /* No Operation */
            NOP,


            HALT,
            STOP,
```

```
/* 8-bit register operations */

/* LD r1 <- r2 */

LD_AA,  // 7F , Same as original

LD_AB,  // 78 , Same as original

LD_AC,  // 7A , Same as original

LD_AD,

LD_AE,

LD_AH,

LD_AL,

LD_BB,

LD_BA,

LD_BC,

LD_BD,

LD_BE,

LD_BH,

LD_BL,

LD_CA,

LD_CB,

LD_CC,

LD_CD,

LD_CE,

LD_CH,

LD_CL,

LD_DA,

LD_DB,
```

```
        LD_DC,
        LD_DD,
        LD_DE,
        LD_DH,
        LD_DL,
        LD_EA,
        LD_EB,
        LD_EC,
        LD_ED,
        LD_EE,
        LD_EH,
        LD_EL,
        LD_HA,
        LD_HB,
        LD_HC,
        LD_HD,
        LD_HE,
        LD_HH,
        LD_HL,
        LD_LA,
        LD_LB,
        LD_LC,
        LD_LD,
        LD_LE,
        LD_LL,
        LD_LH,
```

```
        LD_SPlL, // low side of SP
        LD_SPhH, // high side of SP
        LD_PCHL,
        LD_SPHL,

        LD_HL_SPR8,



        /* LD r1 <- (nn) */
        LD_APC,
        LD_BPC,
        LD_CPC,
        LD_DPC,
        LD_EPC,
        LD_HPC,
        LD_LPC,
        LD_TPC,
        LD_XPC,
        LD_SPlPC,
        LD_SPhPC,
        LD_ABC,
        LD_ADE,
        LD_AHL,
        LD_BHL,
        LD_CHL,
        LD_DHL,
```

```
        LD_EHL,
        LD_HHL,
        LD_LHL,
        LD_THL,
        LD_BSP,
        LD_CSP,
        LD_DSP,
        LD_ESP,
        LD_HSP,
        LD_LSP,
        LD_ASP,
        LD_FSP,
        LD_PClSP,
        LD_PChSP,
        LD_AHT,
        LD_AHC,
        LD_ATX,


        /* LD (nn) <- r1 */
        LD_PCB,
        LD_PCC,
        LD_PCD,
        LD_PCE,
        LD_PCH,
        LD_PCL,
```

```
        LD_PCT,
        LD_PCSPl,
        LD_PCSPh,
        LD_BCA,
        LD_DEA,
        LD_HLA,
        LD_HLB,
        LD_HLC,
        LD_HLD,
        LD_HLE,
        LD_HLH,
        LD_HLL,
        LD_HLT,
        LD_SPA,
        LD_SPB,
        LD_SPC,
        LD_SPD,
        LD_SPE,
        LD_SPH,
        LD_SPL,
        LD_SPF,
        LD_SPPCh,
        LD_SPPCl,
        LD_HTA,
        LD_HCA,
        LD_TXA,
```

```
        LD_TXSPl,
        LD_TXSPh,

        /* Arithmetic Operations */

        ADD_AA,  // Write back to A
        ADD_AB,
        ADD_AC,
        ADD_AD,
        ADD_AE,
        ADD_AH,
        ADD_AL,
        ADD_AT,
        ADD_AHL,
        ADD_LC,
        ADD_LE,  // 16-bit
        ADD_LL,
        ADD_LSPl,

        ADD_SPT,

        ADC_AA,
        ADC_AB,
        ADC_AC,
        ADC_AD,
        ADC_AE,
```

```
        ADC_AH,
        ADC_AL,
        ADC_AT,
        ADC_AHL,
        ADC_HB,
        ADC_HD,
        ADC_HH,
        ADC_HSPh,

        SUB_AA,
        SUB_AB,
        SUB_AC,
        SUB_AD,
        SUB_AE,
        SUB_AH,
        SUB_AL,
        SUB_AT,
        SUB_AHL,

        SBC_AA,
        SBC_AB,
        SBC_AC,
        SBC_AD,
        SBC_AE,
        SBC_AH,
        SBC_AL,
```

```
        SBC_AT,
        SBC_AHL,

        AND_AA,
        AND_AB,
        AND_AC,
        AND_AD,
        AND_AE,
        AND_AH,
        AND_AL,
        AND_AT,
        AND_AHL,


        OR_AA,
        OR_AB,
        OR_AC,
        OR_AD,
        OR_AE,
        OR_AH,
        OR_AL,
        OR_AT,
        OR_AHL,

        XOR_AA,
        XOR_AB,
```

```
        XOR_AC,
        XOR_AD,
        XOR_AE,
        XOR_AH,
        XOR_AL,
        XOR_AT,
        XOR_AHL,

        CP_AA,
        CP_AB,
        CP_AC,
        CP_AD,
        CP_AE,
        CP_AH,
        CP_AL,
        CP_AT,
        CP_AHL,

        INC_A,
        INC_B,
        INC_C,
        INC_D,
        INC_E,
        INC_H,
        INC_L,
        INC_T,
```

```
    INC_BC, // 16-bit
    INC_DE,
    INC_HL,
    INC_SP,
    INC_TX,

    DEC_A,
    DEC_B,
    DEC_C,
    DEC_D,
    DEC_E,
    DEC_H,
    DEC_L,
    DEC_T,

    DEC_BC, // 16-bit
    DEC_DE,
    DEC_HL,
    DEC_SP,
    DEC_TX,

    RL_A,
    RL_B,
    RL_C,
```

```
        RL_D,
        RL_E,
        RL_H,
        RL_L,
        RL_T,


        RLC_A,
        RLC_B,
        RLC_C,
        RLC_D,
        RLC_E,
        RLC_H,
        RLC_L,
        RLC_T,

        RR_A,
        RR_B,
        RR_C,
        RR_D,
        RR_E,
        RR_H,
        RR_L,
        RR_T,

        RRC_A,
```

```
        RRC_B,
        RRC_C,
        RRC_D,
        RRC_E,
        RRC_H,
        RRC_L,
        RRC_T,

        SLA_A,
        SLA_B,
        SLA_C,
        SLA_D,
        SLA_E,
        SLA_H,
        SLA_L,
        SLA_T,

        SRA_A,
        SRA_B,
        SRA_C,
        SRA_D,
        SRA_E,
        SRA_H,
        SRA_L,
        SRA_T,
```

```
        SWAP_A,
        SWAP_B,
        SWAP_C,
        SWAP_D,
        SWAP_E,
        SWAP_H,
        SWAP_L,
        SWAP_T,

        SRL_A,
        SRL_B,
        SRL_C,
        SRL_D,
        SRL_E,
        SRL_H,
        SRL_L,
        SRL_T,

        DAA,
        CPL,
        SCF,
        CCF,

        JP_R8,
        JP_NZR8,
        JP_ZR8,
```

```
        JP_NCR8,
        JP_CR8,
        JP_TX,
        JP_Z_TX,
        JP_NZ_TX,
        JP_C_TX,
        JP_NC_TX,

        RST_00,
        RST_08,
        RST_10,
        RST_18,
        RST_20,
        RST_28,
        RST_30,
        RST_38,
        RST_40,
        RST_48,
        RST_50,
        RST_58,
        RST_60,

        BIT0_A,
        BIT1_A,
        BIT2_A,
        BIT3_A,
```

```
        BIT4_A,
        BIT5_A,
        BIT6_A,
        BIT7_A,

        BIT0_B,
        BIT1_B,
        BIT2_B,
        BIT3_B,
        BIT4_B,
        BIT5_B,
        BIT6_B,
        BIT7_B,

        BIT0_C,
        BIT1_C,
        BIT2_C,
        BIT3_C,
        BIT4_C,
        BIT5_C,
        BIT6_C,
        BIT7_C,

        BIT0_D,
        BIT1_D,
        BIT2_D,
```

```
        BIT3_D,
        BIT4_D,
        BIT5_D,
        BIT6_D,
        BIT7_D,

        BIT0_E,
        BIT1_E,
        BIT2_E,
        BIT3_E,
        BIT4_E,
        BIT5_E,
        BIT6_E,
        BIT7_E,

        BIT0_H,
        BIT1_H,
        BIT2_H,
        BIT3_H,
        BIT4_H,
        BIT5_H,
        BIT6_H,
        BIT7_H,

        BIT0_L,
        BIT1_L,
```

```
        BIT2_L,
        BIT3_L,
        BIT4_L,
        BIT5_L,
        BIT6_L,
        BIT7_L,

        BIT0_T,
        BIT1_T,
        BIT2_T,
        BIT3_T,
        BIT4_T,
        BIT5_T,
        BIT6_T,
        BIT7_T,

        RES0_A,
        RES1_A,
        RES2_A,
        RES3_A,
        RES4_A,
        RES5_A,
        RES6_A,
        RES7_A,

        RES0_B,
```

```
        RES1_B,
        RES2_B,
        RES3_B,
        RES4_B,
        RES5_B,
        RES6_B,
        RES7_B,

        RES0_C,
        RES1_C,
        RES2_C,
        RES3_C,
        RES4_C,
        RES5_C,
        RES6_C,
        RES7_C,

        RES0_D,
        RES1_D,
        RES2_D,
        RES3_D,
        RES4_D,
        RES5_D,
        RES6_D,
        RES7_D,
```

```
        RES0_E,
        RES1_E,
        RES2_E,
        RES3_E,
        RES4_E,
        RES5_E,
        RES6_E,
        RES7_E,

        RES0_H,
        RES1_H,
        RES2_H,
        RES3_H,
        RES4_H,
        RES5_H,
        RES6_H,
        RES7_H,

        RES0_L,
        RES1_L,
        RES2_L,
        RES3_L,
        RES4_L,
        RES5_L,
        RES6_L,
        RES7_L,
```

```
    RES0_T,
    RES1_T,
    RES2_T,
    RES3_T,
    RES4_T,
    RES5_T,
    RES6_T,
    RES7_T,

    SET0_A,
    SET1_A,
    SET2_A,
    SET3_A,
    SET4_A,
    SET5_A,
    SET6_A,
    SET7_A,

    SET0_B,
    SET1_B,
    SET2_B,
    SET3_B,
    SET4_B,
    SET5_B,
    SET6_B,
```

```
        SET7_B,


        SET0_C,
        SET1_C,
        SET2_C,
        SET3_C,
        SET4_C,
        SET5_C,
        SET6_C,
        SET7_C,

        SET0_D,
        SET1_D,
        SET2_D,
        SET3_D,
        SET4_D,
        SET5_D,
        SET6_D,
        SET7_D,

        SET0_E,
        SET1_E,
        SET2_E,
        SET3_E,
        SET4_E,
        SET5_E,
```

```
    SET6_E,
    SET7_E,

    SET0_H,
    SET1_H,
    SET2_H,
    SET3_H,
    SET4_H,
    SET5_H,
    SET6_H,
    SET7_H,

    SET0_L,
    SET1_L,
    SET2_L,
    SET3_L,
    SET4_L,
    SET5_L,
    SET6_L,
    SET7_L,

    SET0_T,
    SET1_T,
    SET2_T,
    SET3_T,
    SET4_T,
```

```
    SET5_T,
    SET6_T,
    SET7_T,

    EI,
    DI,
    LATCH_INTQ,
    RST_IF

  } GB_Z80_RISC_OPCODE;

`define DECODER_LDn_d8(n) \
begin \
    RISC_OPCODE[1] = LD_``n``PC; \
    NUM_Tcnt = 6'd8; \
end

`define DECODER_LDnn_d16(n1, n2) \
begin \
    RISC_OPCODE[1] = LD_``n2``PC; \
    RISC_OPCODE[3] = LD_``n1``PC; \
    NUM_Tcnt = 6'd12; \
end

`define DECODER_LDnn_A(nn) \
begin \
```

```
    RISC_OPCODE[1] = LD_``nn``A; \
    NUM_Tcnt = 6'd8; \
end

`define DECODER_LDA_nn(nn) \
begin \
    RISC_OPCODE[1] = LD_A``nn; \
    NUM_Tcnt = 6'd8; \
end

`define DECODER_ADDHL_nn(n1, n2) \
begin \
    RISC_OPCODE[1] = ADD_L``n2; \
    RISC_OPCODE[2] = ADC_H``n1; \
    NUM_Tcnt = 6'd8; \
end

`define DECODER_DEC_nn(nn) \
begin \
    RISC_OPCODE[1] = DEC_``nn; \
    NUM_Tcnt = 6'd8; \
end

`define DECODER_INC_nn(nn) \
begin \
    RISC_OPCODE[1] = INC_``nn; \
```

```
    NUM_Tcnt = 6'd8; \
end

`define DECODER_LD_HL_INC_A \
begin \
    RISC_OPCODE[1] = LD_HLA; \
    RISC_OPCODE[2] = INC_HL; \
    NUM_Tcnt = 6'd8; \
end
`define DECODER_LD_HL_DEC_A \
begin \
    RISC_OPCODE[1] = LD_HLA; \
    RISC_OPCODE[2] = DEC_HL; \
    NUM_Tcnt = 6'd8; \
end
`define DECODER_LD_A_HL_INC \
begin \
    RISC_OPCODE[1] = LD_AHL; \
    RISC_OPCODE[2] = INC_HL; \
    NUM_Tcnt = 6'd8; \
end
`define DECODER_LD_A_HL_DEC \
begin \
    RISC_OPCODE[1] = LD_AHL; \
    RISC_OPCODE[2] = DEC_HL; \
    NUM_Tcnt = 6'd8; \
```

```
end
`define DECODER_INC_MEM_HL \
begin \
    RISC_OPCODE[1] = LD_THL; \
    RISC_OPCODE[2] = INC_T; \
    RISC_OPCODE[3] = LD_HLT; \
    NUM_Tcnt = 6'd12; \
end
`define DECODER_DEC_MEM_HL \
begin \
    RISC_OPCODE[1] = LD_THL; \
    RISC_OPCODE[2] = DEC_T; \
    RISC_OPCODE[3] = LD_HLT; \
    NUM_Tcnt = 6'd12; \
end
`define DECODER_LD_MEM_HL_d8 \
begin \
    RISC_OPCODE[1] = LD_TPC; \
    RISC_OPCODE[3] = LD_HLT; \
    NUM_Tcnt = 6'd12; \
end
`define DECODER_LD_n_MEM_HL(n) \
begin \
    RISC_OPCODE[2] = LD_``n``HL; \
    NUM_Tcnt = 6'd8; \
end
```

```
`define DECODER_LD_MEM_HL_n(n) \
begin \
     RISC_OPCODE[2] = LD_HL``n; \
     NUM_Tcnt = 6'd8; \
end
`define DECODER_ALU_op_n(op, n) \
begin \
     RISC_OPCODE[0] = ``op``_A``n; \
end

`define DECODER_ALU_op_d8(op) \
begin \
    RISC_OPCODE[1] = LD_TPC; \
    RISC_OPCODE[2] = ``op``_AT; \
    NUM_Tcnt = 6'd8; \
end

`define DECODER_ALU_op_MEM_HL(op) \
begin \
     RISC_OPCODE[2] = ``op``_A``HL; \
     NUM_Tcnt = 6'd8; \
end

`define DECODER_RET \
begin \
    RISC_OPCODE[1] = LD_PClSP; \
```

```
    RISC_OPCODE[2] = INC_SP; \
    RISC_OPCODE[3] = LD_PChSP; \
    RISC_OPCODE[4] = INC_SP; \
    NUM_Tcnt = 6'd16; \
end

`define DECODER_RETI \
begin \
    RISC_OPCODE[1] = LD_PClSP; \
    RISC_OPCODE[2] = INC_SP; \
    RISC_OPCODE[3] = LD_PChSP; \
    RISC_OPCODE[4] = INC_SP; \
    RISC_OPCODE[5] = EI; \
    NUM_Tcnt = 6'd16; \
end

`define DECODER_RET_NZ \
begin \
    if (!FLAG[7]) \
        begin \
            RISC_OPCODE[3] = LD_PClSP; \
            RISC_OPCODE[4] = INC_SP; \
            RISC_OPCODE[5] = LD_PChSP; \
            RISC_OPCODE[6] = INC_SP; \
        end \
    NUM_Tcnt = FLAG[7] ? 6'd8 : 6'd20; \
```

```
end


`define DECODER_RET_Z \
begin \
    if (FLAG[7]) \
        begin \
            RISC_OPCODE[3] = LD_PClSP; \
            RISC_OPCODE[4] = INC_SP; \
            RISC_OPCODE[5] = LD_PChSP; \
            RISC_OPCODE[6] = INC_SP; \
        end \
    NUM_Tcnt = FLAG[7] ? 6'd20 : 6'd8; \
end


`define DECODER_RET_C \
begin \
    if (FLAG[4]) \
        begin \
            RISC_OPCODE[3] = LD_PClSP; \
            RISC_OPCODE[4] = INC_SP; \
            RISC_OPCODE[5] = LD_PChSP; \
            RISC_OPCODE[6] = INC_SP; \
        end \
    NUM_Tcnt = FLAG[4] ? 6'd20 : 6'd8; \
end
```

```
`define DECODER_RET_NC \
begin \
    if (!FLAG[4]) \
        begin \
            RISC_OPCODE[3] = LD_PClSP; \
            RISC_OPCODE[4] = INC_SP; \
            RISC_OPCODE[5] = LD_PChSP; \
            RISC_OPCODE[6] = INC_SP; \
        end \
    NUM_Tcnt = FLAG[4] ? 6'd8 : 6'd20; \
end

`define DECODER_PUSH_nn(n1, n2) \
begin \
    RISC_OPCODE[2] = DEC_SP; \
    RISC_OPCODE[3] = LD_SP``n1; \
    RISC_OPCODE[4] = DEC_SP; \
    RISC_OPCODE[5] = LD_SP``n2; \
    NUM_Tcnt = 6'd16; \
end

`define DECODER_POP_nn(n1, n2) \
begin \
    RISC_OPCODE[2] = LD_``n2``SP; \
    RISC_OPCODE[3] = INC_SP; \
    RISC_OPCODE[4] = LD_``n1``SP; \
```

```
    RISC_OPCODE[5] = INC_SP; \
    NUM_Tcnt = 6'd12; \
end

`define DECODER_JP_Z_a16 \
begin \
    RISC_OPCODE[1] = LD_XPC; \
    RISC_OPCODE[3] = LD_TPC; \
    RISC_OPCODE[6] = JP_Z_TX; \
    NUM_Tcnt = FLAG[7] ? 6'd16 : 6'd12; \
end

`define DECODER_JP_NZ_a16 \
begin \
    RISC_OPCODE[1] = LD_XPC; \
    RISC_OPCODE[3] = LD_TPC; \
    RISC_OPCODE[6] = JP_NZ_TX; \
    NUM_Tcnt = FLAG[7] ? 6'd12 : 6'd16; \
end

`define DECODER_JP_C_a16 \
begin \
    RISC_OPCODE[1] = LD_XPC; \
    RISC_OPCODE[3] = LD_TPC; \
    RISC_OPCODE[6] = JP_C_TX; \
    NUM_Tcnt = FLAG[4] ? 6'd16 : 6'd12; \
```

```
end


`define DECODER_JP_NC_a16 \
begin \
    RISC_OPCODE[1] = LD_XPC; \
    RISC_OPCODE[3] = LD_TPC; \
    RISC_OPCODE[6] = JP_NC_TX; \
    NUM_Tcnt = FLAG[4] ? 6'd12 : 6'd16; \
end



`define DECODER_JP_a16 \
begin \
    RISC_OPCODE[1] = LD_XPC; \
    RISC_OPCODE[3] = LD_TPC; \
    RISC_OPCODE[6] = JP_TX; \
    NUM_Tcnt = 6'd16; \
end


`define DECODER_CALL_a16 \
begin \
    RISC_OPCODE[2] = LD_XPC; \
    RISC_OPCODE[3] = LD_TPC; \
    RISC_OPCODE[5] = DEC_SP; \
    RISC_OPCODE[6] = LD_SPPCh; \
    RISC_OPCODE[7] = DEC_SP; \
```

```
        RISC_OPCODE[8] = LD_SPPCl; \
        RISC_OPCODE[9] = JP_TX; \
        NUM_Tcnt = 6'd24; \
    end

    `define DECODER_CALL_Z_a16 \
    begin \
        RISC_OPCODE[2] = LD_XPC; \
        RISC_OPCODE[3] = LD_TPC; \
        if (FLAG[7]) \
        begin \
            RISC_OPCODE[5] = DEC_SP; \
            RISC_OPCODE[6] = LD_SPPCh; \
            RISC_OPCODE[7] = DEC_SP; \
            RISC_OPCODE[8] = LD_SPPCl; \
            RISC_OPCODE[9] = JP_Z_TX; \
        end \
        NUM_Tcnt = FLAG[7] ? 6'd24 : 6'd12; \
    end

    `define DECODER_CALL_NZ_a16 \
    begin \
        RISC_OPCODE[2] = LD_XPC; \
        RISC_OPCODE[3] = LD_TPC; \
        if (!FLAG[7]) \
        begin \
```

```
            RISC_OPCODE[5] = DEC_SP; \
            RISC_OPCODE[6] = LD_SPPCh; \
            RISC_OPCODE[7] = DEC_SP; \
            RISC_OPCODE[8] = LD_SPPCl; \
            RISC_OPCODE[9] = JP_NZ_TX; \
        end \
        NUM_Tcnt = FLAG[7] ? 6'd12 : 6'd24; \
    end

    `define DECODER_CALL_C_a16 \
    begin \
        RISC_OPCODE[2] = LD_XPC; \
        RISC_OPCODE[3] = LD_TPC; \
        if (FLAG[4]) \
        begin \
            RISC_OPCODE[5] = DEC_SP; \
            RISC_OPCODE[6] = LD_SPPCh; \
            RISC_OPCODE[7] = DEC_SP; \
            RISC_OPCODE[8] = LD_SPPCl; \
            RISC_OPCODE[9] = JP_C_TX; \
        end \
        NUM_Tcnt = FLAG[4] ? 6'd24 : 6'd12; \
    end

    `define DECODER_CALL_NC_a16 \
    begin \
```

```
        RISC_OPCODE[2] = LD_XPC; \
        RISC_OPCODE[3] = LD_TPC; \
        if (!FLAG[4]) \
        begin \
            RISC_OPCODE[5] = DEC_SP; \
            RISC_OPCODE[6] = LD_SPPCh; \
            RISC_OPCODE[7] = DEC_SP; \
            RISC_OPCODE[8] = LD_SPPCl; \
            RISC_OPCODE[9] = JP_NC_TX; \
        end \
        NUM_Tcnt = FLAG[4] ? 6'd12 : 6'd24; \
    end

    `define DECODER_RST(addr) \
    begin \
        RISC_OPCODE[2] = DEC_SP; \
        RISC_OPCODE[3] = LD_SPPCh; \
        RISC_OPCODE[4] = DEC_SP; \
        RISC_OPCODE[5] = LD_SPPCl; \
        RISC_OPCODE[6] = RST_``addr; \
        NUM_Tcnt = 6'd16; \
    end

    // Read/Write timing is important for TIMER
    `define DECODER_LDH_a8_A \
    begin \
```

```
        RISC_OPCODE[1] = LD_TPC; \
        RISC_OPCODE[3] = LD_HTA; \
        NUM_Tcnt = 6'd12; \
    end

    `define DECODER_LDH_A_a8 \
    begin \
        RISC_OPCODE[1] = LD_TPC; \
        RISC_OPCODE[3] = LD_AHT; \
        NUM_Tcnt = 6'd12; \
    end

    `define DECODER_LDH_C_A \
    begin \
        RISC_OPCODE[2] = LD_HCA; \
        NUM_Tcnt = 6'd8; \
    end

    `define DECODER_LDH_A_C \
    begin \
        RISC_OPCODE[2] = LD_AHC; \
        NUM_Tcnt = 6'd8; \
    end

    `define DECODER_ADD_SP_R8 \
    begin \
```

```
     RISC_OPCODE[1] = LD_TPC; \
     RISC_OPCODE[3] = ADD_SPT; \
     NUM_Tcnt = 6'd16; \
end

`define DECODER_LD_HL_SPR8 \
begin \
    RISC_OPCODE[1] = LD_TPC; \
    RISC_OPCODE[3] = LD_HL_SPR8; \
    NUM_Tcnt = 6'd12; \
end

`define DECODER_LD_a16_SP \
begin \
    RISC_OPCODE[1] = LD_XPC; \
    RISC_OPCODE[3] = LD_TPC; \
    RISC_OPCODE[6] = LD_TXSPl; \
    RISC_OPCODE[7] = INC_TX; \
    RISC_OPCODE[8] = LD_TXSPh; \
    NUM_Tcnt = 6'd20; \
end

`define DECODER_LD_a16_A \
begin \
    RISC_OPCODE[1] = LD_XPC; \
    RISC_OPCODE[3] = LD_TPC; \
```

```
    RISC_OPCODE[5] = LD_TXA; \
    NUM_Tcnt = 6'd16; \
end

`define DECODER_LD_A_a16 \
begin \
    RISC_OPCODE[1] = LD_XPC; \
    RISC_OPCODE[3] = LD_TPC; \
    RISC_OPCODE[5] = LD_ATX; \
    NUM_Tcnt = 6'd16; \
end

`define DECODER_CB_ALU_op_MEM_HL(op) \
begin \
     RISC_OPCODE[2] = LD_THL; \
     RISC_OPCODE[3] = ``op``_T; \
     RISC_OPCODE[4] = LD_HLT; \
     NUM_Tcnt = 6'd12; \
end

`define DECODER_CB_BIT_op_b_n(op, b, n) \
begin \
    RISC_OPCODE[0] = ``op````b``_``n; \
end

 // Cycle count is wrong on the html
```

```
`define DECODER_CB_BIT_op_b_MEM_HL(op, b) \
begin \
    RISC_OPCODE[1] = LD_THL; \
    RISC_OPCODE[2] = ``op````b``_T; \
    NUM_Tcnt = 6'd8; \
end

`define DECODER_CB_RES_SET_op_b_MEM_HL(op, b) \
begin \
    RISC_OPCODE[1] = LD_THL; \
    RISC_OPCODE[2] = ``op````b``_T; \
    RISC_OPCODE[3] = LD_HLT; \
    NUM_Tcnt = 6'd12; \
end

`define DECODER_INTR(addr)\
begin \
    RISC_OPCODE[0] = DI; \
    RISC_OPCODE[1] = DEC_SP; \
    RISC_OPCODE[2] = LD_SPPCh; \
    RISC_OPCODE[3] = LATCH_INTQ; \
    RISC_OPCODE[4] = RST_IF; \
    RISC_OPCODE[5] = DEC_SP; \
    RISC_OPCODE[6] = LD_SPPCl; \
    RISC_OPCODE[7] = RST_``addr; \
    NUM_Tcnt = 6'd20; \
```

```
    end


    `endif
```

GB_Z80_SINGLE.sv

```
Unset
    `timescale 1ns / 1ns
    ////////////////////////////////////////////////////////////////////////////////
////////
    // This is the GB-Z80 CPU
    // All modules in a single file for simulation
    ////////////////////////////////////////////////////////////////////////////////
////////
    `include "GB_Z80_ALU.vh"
    `include "GB_Z80_CPU.vh"
    `include "GB_Z80_DECODER.vh"


    `define NO_BOOT 0


    module GB_Z80_SINGLE
    (
        input logic clk,
        input logic rst,
```

```
            output logic [15:0] ADDR, // Memory Address Bus
            input logic [7:0] DATA_in, // Input Data Bus
            output logic [7:0] DATA_out, // Output Data Bus
            output logic RD, // CPU wants to read data from Memory or IO, active
high
            output logic WR, // CPU holds valid data to be stored in Memory or
IO, active high
            output logic CPU_HALT, // CPU has executed a HALT instruction and is
awaiting an interrupt, active high
            input logic [4:0] INTQ, // Interrupt Request, Interrupt will be
honored at the end of the current instruction
            input logic [4:0] IE // Interrupt Enable
        );

        GB_Z80_REG CPU_REG, CPU_REG_NEXT;
        logic [15:0] ADDR_NEXT;

        /* Decoder */
        logic [7:0] INST, INST_NEXT; // Instruction Register
        logic [4:0] INTQ_INT, INTQ_INT_NEXT;
        GB_Z80_RISC_OPCODE RISC_OPCODE [0:10];
        logic [5:0] NUM_Tcnt;
        logic isCB, isCB_NEXT;
        logic isINT, isINT_NEXT;
        logic isPCMEM [0:10];
        logic [4:0] T_CNT, T_CNT_NEXT;
```

```
	logic [2:0] M_CNT, M_CNT_NEXT;
	byte cur_risc_num;

	GB_Z80_DECODER CPU_DECODER(.CPU_OPCODE(INST), .INTQ(INTQ_INT),
.isCB(isCB), .isINT(isINT), .RISC_OPCODE(RISC_OPCODE), .NUM_Tcnt(NUM_Tcnt),
.isPCMEM(isPCMEM),
                                .FLAG(CPU_REG.F));

	/* ALU */
	logic [7:0] ALU_OPD1_L, ALU_OPD2_L, ALU_STATUS, ALU_RESULT_L,
ALU_RESULT_H;
	GB_Z80_ALU_OPCODE ALU_OPCODE;
	GB_Z80_ALU CPU_ALU(.OPD1_L(ALU_OPD1_L), .OPD2_L(ALU_OPD2_L),
.OPCODE(ALU_OPCODE), .FLAG(CPU_REG.F), .STATUS(ALU_STATUS),
                        .RESULT_L(ALU_RESULT_L), .RESULT_H(ALU_RESULT_H));

	/* Main FSMD */
	// Main 4 Stages are IF -> DE -> EX -> (MEM)WB
	// Each takes 1 T cycle

	typedef enum {CPU_IF, CPU_DE, CPU_DE_CB, CPU_EX_RISC, CPU_WB_RISC}
CPU_STATE_t;
	CPU_STATE_t CPU_STATE, CPU_STATE_NEXT;

	logic RD_NEXT, WR_NEXT;
	logic EX_done;
```

```
logic IME, IME_NEXT; // Interrupt Master Enable


always_ff @(posedge clk)
begin
    /* Power On Reset */
    if (rst)
    begin
        CPU_STATE <= CPU_IF;
        CPU_REG.PC <= 0;
        CPU_REG.F <= 0;
        CPU_REG.T <= 0;
        ADDR <= 0;
        if (`NO_BOOT)
        begin
            CPU_REG.A <= 8'h01;
            CPU_REG.F <= 8'hB0;
            CPU_REG.B <= 8'h00;
            CPU_REG.C <= 8'h13;
            CPU_REG.D <= 8'h00;
            CPU_REG.E <= 8'hD8;
            CPU_REG.H <= 8'h01;
            CPU_REG.L <= 8'h4D;
            CPU_REG.SPh <= 8'hFF;
            CPU_REG.SPl <= 8'hFE;
```

```
                CPU_REG.PC <= 16'h0100;
                ADDR <= 16'h0100;
            end
            RD <= 1; WR <= 0;
            T_CNT <= 0; //M_CNT <= 0;
            isCB <= 0;
            isINT <= 0;
            IME <= 0;
            INTQ_INT <= 0;
        end
        else
        begin
            CPU_STATE <= CPU_STATE_NEXT;
            CPU_REG <= CPU_REG_NEXT;
            ADDR <= ADDR_NEXT;
            RD <= RD_NEXT; WR<= WR_NEXT;
            T_CNT <= T_CNT_NEXT; //M_CNT <= M_CNT_NEXT;
            isCB <= isCB_NEXT;
            isINT <= isINT_NEXT;
            INST <= INST_NEXT;
            IME <= IME_NEXT;
            INTQ_INT <= INTQ_INT_NEXT;
        end

    end
```

```
assign M_CNT = (T_CNT - 1) >> 2; // 1 M Cycle for every 4 T cycles
assign cur_risc_num = (T_CNT >> 1) - 1 - (isCB << 1) + isINT;


always_comb
begin
    CPU_STATE_NEXT = CPU_STATE;
    CPU_REG_NEXT = CPU_REG;
    ADDR_NEXT = CPU_REG.PC;
    isCB_NEXT = isCB;
    isINT_NEXT = isINT;
    IME_NEXT = IME;
    INTQ_INT_NEXT = INTQ_INT;
    RD_NEXT = 0; WR_NEXT = 0;
    DATA_out = 0;
    T_CNT_NEXT = T_CNT + 1;
    INST_NEXT = INST;
    ALU_OPD1_L = 0;
    ALU_OPD2_L = 0;
    ALU_OPCODE = ALU_NOP;
    CPU_HALT = 0;
    unique case (CPU_STATE)
        // Instruction Fetch From Memory at PC
        CPU_IF :
        begin
            RD_NEXT = 0;
            INST_NEXT = DATA_in;
```

```
            T_CNT_NEXT = T_CNT + 1;
            CPU_STATE_NEXT = CPU_DE;
            CPU_REG_NEXT.PC = CPU_REG.PC + 1;
        end
        CPU_DE :
        begin
            if ((INST) == 8'hCB && !isCB)
            begin
                CPU_STATE_NEXT = CPU_DE_CB;
                isCB_NEXT = 1;
                T_CNT_NEXT = T_CNT + 1;
            end
            else
            begin
                CPU_STATE_NEXT = CPU_EX_RISC;
                T_CNT_NEXT = T_CNT + 1;
            end
        end
        CPU_DE_CB :
        begin
            if (T_CNT != 3) // CB fetch delay
            begin
                T_CNT_NEXT = T_CNT + 1;
                CPU_STATE_NEXT = CPU_DE_CB;
            end
            else
```

```
                    begin
                        T_CNT_NEXT = T_CNT + 1;
                        CPU_STATE_NEXT = CPU_IF;
                        RD_NEXT = 1;
                    end
                end

                CPU_EX_RISC :
                begin
                    T_CNT_NEXT = T_CNT + 1;
                    CPU_STATE_NEXT = CPU_WB_RISC;

                    if (isPCMEM[cur_risc_num])
                            CPU_REG_NEXT.PC = CPU_REG.PC + 1;
                    //if (!IME && (RISC_OPCODE[cur_risc_num] == HALT)) // HALT
"skip" behaviour
                    //        CPU_REG_NEXT.PC = CPU_REG.PC + 1;

                    case (RISC_OPCODE[cur_risc_num])
                        NOP: ;  // no operations
                        LD_APC, LD_BPC, LD_CPC, LD_DPC, LD_EPC, LD_HPC, LD_LPC,
LD_TPC, LD_XPC, LD_SPlPC, LD_SPhPC, JP_R8, JP_NZR8, JP_ZR8, JP_NCR8, JP_CR8:
RD_NEXT = 1;
                        LD_ABC: `RD_nn(B, C)
                        LD_ADE: `RD_nn(D, E)
```

```
                        LD_AHL, LD_BHL, LD_CHL, LD_DHL, LD_EHL, LD_HHL, LD_LHL,
LD_THL, ADD_AHL, ADC_AHL, SUB_AHL, SBC_AHL, AND_AHL, XOR_AHL, OR_AHL, CP_AHL:
`RD_nn(H, L)
                        LD_ATX: `RD_nn(T, X)
                        LD_PClSP, LD_PChSP, LD_BSP, LD_CSP, LD_DSP, LD_ESP,
LD_HSP, LD_LSP, LD_ASP, LD_FSP: `RD_nn(SPh, SPl)
                        LD_AHT: `RD_FFn(T)
                        LD_AHC: `RD_FFn(C)

                        LD_PCSPl, LD_PCSPh: WR_NEXT = 1;
                        LD_BCA: `WR_nn(B, C)
                        LD_DEA: `WR_nn(D, E)
                        LD_HLA, LD_HLB, LD_HLC, LD_HLD, LD_HLE, LD_HLH, LD_HLL,
LD_HLT: `WR_nn(H, L)
                        LD_TXA, LD_TXSPh, LD_TXSPl: `WR_nn(T, X)
                        LD_SPA, LD_SPB, LD_SPC, LD_SPD, LD_SPE, LD_SPH, LD_SPL,
LD_SPF, LD_SPPCh, LD_SPPCl : `WR_nn(SPh, SPl)
                        LD_HTA: `WR_FFn(T)
                        LD_HCA: `WR_FFn(C)
                        DI: IME_NEXT = 0;
                        LATCH_INTQ: INTQ_INT_NEXT = INTQ;
                        RST_IF: begin ADDR_NEXT = 16'hFF0F; WR_NEXT = 1; RD_NEXT
= 1; end
                    default: ;
                endcase
                end
```

```
                CPU_WB_RISC :
                begin
                    if (T_CNT - (isCB << 2) == NUM_Tcnt - 1)
                    begin
                        T_CNT_NEXT = 0;
                        CPU_STATE_NEXT = CPU_IF;

                        isCB_NEXT = 0;
                        RD_NEXT = 1;
                        isINT_NEXT = 0;
                        INTQ_INT_NEXT = 0;
                        if (IME && (INTQ != 5'b00))
                        begin
                            CPU_STATE_NEXT = CPU_EX_RISC;
                            isINT_NEXT = 1;
                        end
                        else if ((INTQ == 5'b00) && (RISC_OPCODE[cur_risc_num] ==
HALT)) // Handle HALT
                        begin
                            CPU_STATE_NEXT = CPU_WB_RISC;
                            T_CNT_NEXT = T_CNT;
                            CPU_HALT = 1;
                        end
                    end
                    else
                    begin
```

```
            T_CNT_NEXT = T_CNT + 1;
            CPU_STATE_NEXT = CPU_EX_RISC;
        end

        case (RISC_OPCODE[cur_risc_num])
            NOP: ;
            LD_AA: `LD_n_n(A, A)
            LD_AB: `LD_n_n(A, B)
            LD_AC: `LD_n_n(A, C)
            LD_AD: `LD_n_n(A, D)
            LD_AE: `LD_n_n(A, E)
            LD_AH: `LD_n_n(A, H)
            LD_AL: `LD_n_n(A, L)

            LD_BA: `LD_n_n(B, A)
            LD_BB: `LD_n_n(B, B)
            LD_BC: `LD_n_n(B, C)
            LD_BD: `LD_n_n(B, D)
            LD_BE: `LD_n_n(B, E)
            LD_BH: `LD_n_n(B, H)
            LD_BL: `LD_n_n(B, L)

            LD_CA: `LD_n_n(C, A)
            LD_CB: `LD_n_n(C, B)
            LD_CC: `LD_n_n(C, C)
            LD_CD: `LD_n_n(C, D)
```

```
LD_CE: `LD_n_n(C, E)
LD_CH: `LD_n_n(C, H)
LD_CL: `LD_n_n(C, L)

LD_DA: `LD_n_n(D, A)
LD_DB: `LD_n_n(D, B)
LD_DC: `LD_n_n(D, C)
LD_DD: `LD_n_n(D, D)
LD_DE: `LD_n_n(D, E)
LD_DH: `LD_n_n(D, H)
LD_DL: `LD_n_n(D, L)

LD_EA: `LD_n_n(E, A)
LD_EB: `LD_n_n(E, B)
LD_EC: `LD_n_n(E, C)
LD_ED: `LD_n_n(E, D)
LD_EE: `LD_n_n(E, E)
LD_EH: `LD_n_n(E, H)
LD_EL: `LD_n_n(E, L)

LD_HA: `LD_n_n(H, A)
LD_HB: `LD_n_n(H, B)
LD_HC: `LD_n_n(H, C)
LD_HD: `LD_n_n(H, D)
LD_HE: `LD_n_n(H, E)
LD_HH: `LD_n_n(H, H)
```

```
                        LD_HL: `LD_n_n(H, L)


                        LD_LA: `LD_n_n(L, A)
                        LD_LB: `LD_n_n(L, B)
                        LD_LC: `LD_n_n(L, C)
                        LD_LD: `LD_n_n(L, D)
                        LD_LE: `LD_n_n(L, E)
                        LD_LH: `LD_n_n(L, H)
                        LD_LL: `LD_n_n(L, L)


                        LD_PCHL: CPU_REG_NEXT.PC = {CPU_REG.H, CPU_REG.L};


                        LD_SPHL: {CPU_REG_NEXT.SPh, CPU_REG_NEXT.SPl} =
{CPU_REG.H, CPU_REG.L};


                        LD_APC, LD_AHL, LD_ABC, LD_ADE, LD_ASP, LD_AHT, LD_AHC,
LD_ATX: CPU_REG_NEXT.A = DATA_in;
                        LD_BPC, LD_BHL, LD_BSP: CPU_REG_NEXT.B = DATA_in;
                        LD_CPC, LD_CHL, LD_CSP: CPU_REG_NEXT.C = DATA_in;
                        LD_DPC, LD_DHL, LD_DSP: CPU_REG_NEXT.D = DATA_in;
                        LD_EPC, LD_EHL, LD_ESP: CPU_REG_NEXT.E = DATA_in;
                        LD_HPC, LD_HHL, LD_HSP: CPU_REG_NEXT.H = DATA_in;
                        LD_LPC, LD_LHL, LD_LSP: CPU_REG_NEXT.L = DATA_in;
                        LD_FSP: CPU_REG_NEXT.F = DATA_in;
                        LD_TPC: CPU_REG_NEXT.T = DATA_in;
                        LD_XPC: CPU_REG_NEXT.X = DATA_in;
```

```
                        LD_SPlPC: CPU_REG_NEXT.SPl = DATA_in;

                        LD_SPhPC: CPU_REG_NEXT.SPh = DATA_in;

                        LD_THL: CPU_REG_NEXT.T = DATA_in;

                        LD_PClSP: CPU_REG_NEXT.PC = {CPU_REG.PC[15:8], DATA_in};

                        LD_PChSP: CPU_REG_NEXT.PC = {DATA_in, CPU_REG.PC[7:0]};




                        LD_BCA, LD_DEA, LD_HLA, LD_SPA, LD_HTA, LD_HCA, LD_TXA:
DATA_out = CPU_REG.A;
                        LD_HLB, LD_SPB: DATA_out = CPU_REG.B;

                        LD_HLC, LD_SPC: DATA_out = CPU_REG.C;

                        LD_HLD, LD_SPD: DATA_out = CPU_REG.D;

                        LD_HLE, LD_SPE: DATA_out = CPU_REG.E;

                        LD_HLH, LD_SPH: DATA_out = CPU_REG.H;

                        LD_HLL, LD_SPL: DATA_out = CPU_REG.L;

                        LD_SPF:  DATA_out = CPU_REG.F;

                        LD_HLT: DATA_out = CPU_REG.T;

                        LD_PCSPl, LD_TXSPl: DATA_out = CPU_REG.SPl;

                        LD_PCSPh, LD_TXSPh: DATA_out = CPU_REG.SPh;

                        LD_SPPCh: DATA_out = CPU_REG.PC[15:8];

                        LD_SPPCl: DATA_out = CPU_REG.PC[7:0];


                        LD_HL_SPR8: `LD_HL_SPR8


                        INC_BC : `INC_nn(B, C)
```

```
DEC_BC : `DEC_nn(B, C)

INC_DE : `INC_nn(D, E)

DEC_DE : `DEC_nn(D, E)

INC_HL : `INC_nn(H, L)

DEC_HL : `DEC_nn(H, L)

INC_TX : `INC_nn(T, X)

DEC_TX : `DEC_nn(T, X)

INC_SP : `INC_nn(SPh, SPl)

DEC_SP : `DEC_nn(SPh, SPl)

INC_A : `INC_n(A)

DEC_A : `DEC_n(A)

INC_B : `INC_n(B)

DEC_B : `DEC_n(B)

INC_C : `INC_n(C)

DEC_C : `DEC_n(C)

INC_D : `INC_n(D)

DEC_D : `DEC_n(D)

INC_E : `INC_n(E)

DEC_E : `DEC_n(E)

INC_H : `INC_n(H)

DEC_H : `DEC_n(H)

INC_L : `INC_n(L)

DEC_L : `DEC_n(L)

INC_T : `INC_n(T)

DEC_T : `DEC_n(T)
```

```
RLC_A : `SHIFTER_op_n(RLC, A)
RLC_B : `SHIFTER_op_n(RLC, B)
RLC_C : `SHIFTER_op_n(RLC, C)
RLC_D : `SHIFTER_op_n(RLC, D)
RLC_E : `SHIFTER_op_n(RLC, E)
RLC_H : `SHIFTER_op_n(RLC, H)
RLC_L : `SHIFTER_op_n(RLC, L)
RLC_T : `SHIFTER_op_n(RLC, T)

RRC_A : `SHIFTER_op_n(RRC, A)
RRC_B : `SHIFTER_op_n(RRC, B)
RRC_C : `SHIFTER_op_n(RRC, C)
RRC_D : `SHIFTER_op_n(RRC, D)
RRC_E : `SHIFTER_op_n(RRC, E)
RRC_H : `SHIFTER_op_n(RRC, H)
RRC_L : `SHIFTER_op_n(RRC, L)
RRC_T : `SHIFTER_op_n(RRC, T)

RR_A : `SHIFTER_op_n(RR, A)
RR_B : `SHIFTER_op_n(RR, B)
RR_C : `SHIFTER_op_n(RR, C)
RR_D : `SHIFTER_op_n(RR, D)
RR_E : `SHIFTER_op_n(RR, E)
RR_H : `SHIFTER_op_n(RR, H)
RR_L : `SHIFTER_op_n(RR, L)
RR_T : `SHIFTER_op_n(RR, T)
```

```
RL_A : `SHIFTER_op_n(RL, A)

RL_B : `SHIFTER_op_n(RL, B)

RL_C : `SHIFTER_op_n(RL, C)

RL_D : `SHIFTER_op_n(RL, D)

RL_E : `SHIFTER_op_n(RL, E)

RL_H : `SHIFTER_op_n(RL, H)

RL_L : `SHIFTER_op_n(RL, L)

RL_T : `SHIFTER_op_n(RL, T)


SRA_A : `SHIFTER_op_n(SRA, A)

SRA_B : `SHIFTER_op_n(SRA, B)

SRA_C : `SHIFTER_op_n(SRA, C)

SRA_D : `SHIFTER_op_n(SRA, D)

SRA_E : `SHIFTER_op_n(SRA, E)

SRA_H : `SHIFTER_op_n(SRA, H)

SRA_L : `SHIFTER_op_n(SRA, L)

SRA_T : `SHIFTER_op_n(SRA, T)


SLA_A : `SHIFTER_op_n(SLA, A)

SLA_B : `SHIFTER_op_n(SLA, B)

SLA_C : `SHIFTER_op_n(SLA, C)

SLA_D : `SHIFTER_op_n(SLA, D)

SLA_E : `SHIFTER_op_n(SLA, E)

SLA_H : `SHIFTER_op_n(SLA, H)

SLA_L : `SHIFTER_op_n(SLA, L)
```

```
SLA_T : `SHIFTER_op_n(SLA, T)


SWAP_A : `SHIFTER_op_n(SWAP, A)
SWAP_B : `SHIFTER_op_n(SWAP, B)
SWAP_C : `SHIFTER_op_n(SWAP, C)
SWAP_D : `SHIFTER_op_n(SWAP, D)
SWAP_E : `SHIFTER_op_n(SWAP, E)
SWAP_H : `SHIFTER_op_n(SWAP, H)
SWAP_L : `SHIFTER_op_n(SWAP, L)
SWAP_T : `SHIFTER_op_n(SWAP, T)

SRL_A : `SHIFTER_op_n(SRL, A)
SRL_B : `SHIFTER_op_n(SRL, B)
SRL_C : `SHIFTER_op_n(SRL, C)
SRL_D : `SHIFTER_op_n(SRL, D)
SRL_E : `SHIFTER_op_n(SRL, E)
SRL_H : `SHIFTER_op_n(SRL, H)
SRL_L : `SHIFTER_op_n(SRL, L)
SRL_T : `SHIFTER_op_n(SRL, T)




ADD_AA: `ALU_A_op_n(ADD, A)
ADD_AB: `ALU_A_op_n(ADD, B)
ADD_AC: `ALU_A_op_n(ADD, C)
ADD_AD: `ALU_A_op_n(ADD, D)
```

```
ADD_AE: `ALU_A_op_n(ADD, E)

ADD_AH: `ALU_A_op_n(ADD, H)

ADD_AL: `ALU_A_op_n(ADD, L)

ADD_AT: `ALU_A_op_n(ADD, T)

ADD_AHL: `ALU_A_op_Data_in(ADD)


ADD_SPT: `ADD_SPT


ADC_AA: `ALU_A_op_n(ADC, A)

ADC_AB: `ALU_A_op_n(ADC, B)

ADC_AC: `ALU_A_op_n(ADC, C)

ADC_AD: `ALU_A_op_n(ADC, D)

ADC_AE: `ALU_A_op_n(ADC, E)

ADC_AH: `ALU_A_op_n(ADC, H)

ADC_AL: `ALU_A_op_n(ADC, L)

ADC_AT: `ALU_A_op_n(ADC, T)

ADC_AHL: `ALU_A_op_Data_in(ADC)


SUB_AA: `ALU_A_op_n(SUB, A)

SUB_AB: `ALU_A_op_n(SUB, B)

SUB_AC: `ALU_A_op_n(SUB, C)

SUB_AD: `ALU_A_op_n(SUB, D)

SUB_AE: `ALU_A_op_n(SUB, E)

SUB_AH: `ALU_A_op_n(SUB, H)

SUB_AL: `ALU_A_op_n(SUB, L)

SUB_AT: `ALU_A_op_n(SUB, T)
```

```
SUB_AHL: `ALU_A_op_Data_in(SUB)


SBC_AA: `ALU_A_op_n(SBC, A)
SBC_AB: `ALU_A_op_n(SBC, B)
SBC_AC: `ALU_A_op_n(SBC, C)
SBC_AD: `ALU_A_op_n(SBC, D)
SBC_AE: `ALU_A_op_n(SBC, E)
SBC_AH: `ALU_A_op_n(SBC, H)
SBC_AL: `ALU_A_op_n(SBC, L)
SBC_AT: `ALU_A_op_n(SBC, T)
SBC_AHL: `ALU_A_op_Data_in(SBC)

AND_AA: `ALU_A_op_n(AND, A)
AND_AB: `ALU_A_op_n(AND, B)
AND_AC: `ALU_A_op_n(AND, C)
AND_AD: `ALU_A_op_n(AND, D)
AND_AE: `ALU_A_op_n(AND, E)
AND_AH: `ALU_A_op_n(AND, H)
AND_AL: `ALU_A_op_n(AND, L)
AND_AT: `ALU_A_op_n(AND, T)
AND_AHL: `ALU_A_op_Data_in(AND)

XOR_AA: `ALU_A_op_n(XOR, A)
XOR_AB: `ALU_A_op_n(XOR, B)
XOR_AC: `ALU_A_op_n(XOR, C)
XOR_AD: `ALU_A_op_n(XOR, D)
```

```
XOR_AE: `ALU_A_op_n(XOR, E)

XOR_AH: `ALU_A_op_n(XOR, H)

XOR_AL: `ALU_A_op_n(XOR, L)

XOR_AT: `ALU_A_op_n(XOR, T)

XOR_AHL: `ALU_A_op_Data_in(XOR)


OR_AA: `ALU_A_op_n(OR, A)

OR_AB: `ALU_A_op_n(OR, B)

OR_AC: `ALU_A_op_n(OR, C)

OR_AD: `ALU_A_op_n(OR, D)

OR_AE: `ALU_A_op_n(OR, E)

OR_AH: `ALU_A_op_n(OR, H)

OR_AL: `ALU_A_op_n(OR, L)

OR_AT: `ALU_A_op_n(OR, T)

OR_AHL: `ALU_A_op_Data_in(OR)


CP_AA: `ALU_op_n(CP, A)

CP_AB: `ALU_op_n(CP, B)

CP_AC: `ALU_op_n(CP, C)

CP_AD: `ALU_op_n(CP, D)

CP_AE: `ALU_op_n(CP, E)

CP_AH: `ALU_op_n(CP, H)

CP_AL: `ALU_op_n(CP, L)

CP_AT: `ALU_op_n(CP, T)

CP_AHL: `ALU_op_Data_in(CP)
```

```
                        ADD_LC: `ADDL_n(C)
                        ADD_LE: `ADDL_n(E)
                        ADD_LL: `ADDL_n(L)
                        ADD_LSPl: `ADDL_n(SPl)
                        ADC_HB: `ADCH_n(B)
                        ADC_HD: `ADCH_n(D)
                        ADC_HH: `ADCH_n(H)
                        ADC_HSPh: `ADCH_n(SPh)

                        DAA: `DAA
                        CPL: begin CPU_REG_NEXT.A = CPU_REG.A ^ 8'hFF;
CPU_REG_NEXT.F = CPU_REG.F | 8'b0110_0000; end// invert all bits in A
                        SCF: CPU_REG_NEXT.F = {CPU_REG.F[7], 3'b001,
CPU_REG.F[3:0]}; // set carry flag
                        CCF: CPU_REG_NEXT.F = {CPU_REG.F[7], 2'b00,
~CPU_REG.F[4], CPU_REG.F[3:0]}; // compliment carry flag

                        JP_R8: CPU_REG_NEXT.PC = `DO_JPR8;
                        JP_NZR8 : CPU_REG_NEXT.PC = CPU_REG.F[7] ? CPU_REG.PC :
`DO_JPR8;
                        JP_ZR8 : CPU_REG_NEXT.PC = CPU_REG.F[7] ? `DO_JPR8 :
CPU_REG.PC;
                        JP_NCR8 : CPU_REG_NEXT.PC = CPU_REG.F[4] ? CPU_REG.PC :
`DO_JPR8;
                        JP_CR8 : CPU_REG_NEXT.PC = CPU_REG.F[4] ? `DO_JPR8 :
CPU_REG.PC;
```

```
                        JP_TX : CPU_REG_NEXT.PC = {CPU_REG.T, CPU_REG.X};
                        JP_Z_TX : CPU_REG_NEXT.PC = CPU_REG.F[7] ? {CPU_REG.T,
CPU_REG.X} : CPU_REG.PC ;
                        JP_NZ_TX : CPU_REG_NEXT.PC = CPU_REG.F[7] ? CPU_REG.PC :
{CPU_REG.T, CPU_REG.X};
                        JP_C_TX : CPU_REG_NEXT.PC = CPU_REG.F[4] ? {CPU_REG.T,
CPU_REG.X} : CPU_REG.PC ;
                        JP_NC_TX : CPU_REG_NEXT.PC = CPU_REG.F[4] ? CPU_REG.PC :
{CPU_REG.T, CPU_REG.X};

                        RST_00 : CPU_REG_NEXT.PC = {8'h00, 8'h00};
                        RST_08 : CPU_REG_NEXT.PC = {8'h00, 8'h08};
                        RST_10 : CPU_REG_NEXT.PC = {8'h00, 8'h10};
                        RST_18 : CPU_REG_NEXT.PC = {8'h00, 8'h18};
                        RST_20 : CPU_REG_NEXT.PC = {8'h00, 8'h20};
                        RST_28 : CPU_REG_NEXT.PC = {8'h00, 8'h28};
                        RST_30 : CPU_REG_NEXT.PC = {8'h00, 8'h30};
                        RST_38 : CPU_REG_NEXT.PC = {8'h00, 8'h38};
                        RST_40 : CPU_REG_NEXT.PC = {8'h00, 8'h40};
                        RST_48 : CPU_REG_NEXT.PC = {8'h00, 8'h48};
                        RST_50 : CPU_REG_NEXT.PC = {8'h00, 8'h50};
                        RST_58 : CPU_REG_NEXT.PC = {8'h00, 8'h58};
                        RST_60 : CPU_REG_NEXT.PC = {8'h00, 8'h60};

                        BIT0_A: `ALU_BIT_b_n(0, A)
```

```
                BIT1_A: `ALU_BIT_b_n(1, A)
                BIT2_A: `ALU_BIT_b_n(2, A)
                BIT3_A: `ALU_BIT_b_n(3, A)
                BIT4_A: `ALU_BIT_b_n(4, A)
                BIT5_A: `ALU_BIT_b_n(5, A)
                BIT6_A: `ALU_BIT_b_n(6, A)
                BIT7_A: `ALU_BIT_b_n(7, A)

                BIT0_B: `ALU_BIT_b_n(0, B)
                BIT1_B: `ALU_BIT_b_n(1, B)
                BIT2_B: `ALU_BIT_b_n(2, B)
                BIT3_B: `ALU_BIT_b_n(3, B)
                BIT4_B: `ALU_BIT_b_n(4, B)
                BIT5_B: `ALU_BIT_b_n(5, B)
                BIT6_B: `ALU_BIT_b_n(6, B)
                BIT7_B: `ALU_BIT_b_n(7, B)

                BIT0_C: `ALU_BIT_b_n(0, C)
                BIT1_C: `ALU_BIT_b_n(1, C)
                BIT2_C: `ALU_BIT_b_n(2, C)
                BIT3_C: `ALU_BIT_b_n(3, C)
                BIT4_C: `ALU_BIT_b_n(4, C)
                BIT5_C: `ALU_BIT_b_n(5, C)
                BIT6_C: `ALU_BIT_b_n(6, C)
                BIT7_C: `ALU_BIT_b_n(7, C)
```

```
BIT0_D: `ALU_BIT_b_n(0, D)
BIT1_D: `ALU_BIT_b_n(1, D)
BIT2_D: `ALU_BIT_b_n(2, D)
BIT3_D: `ALU_BIT_b_n(3, D)
BIT4_D: `ALU_BIT_b_n(4, D)
BIT5_D: `ALU_BIT_b_n(5, D)
BIT6_D: `ALU_BIT_b_n(6, D)
BIT7_D: `ALU_BIT_b_n(7, D)

BIT0_E: `ALU_BIT_b_n(0, E)
BIT1_E: `ALU_BIT_b_n(1, E)
BIT2_E: `ALU_BIT_b_n(2, E)
BIT3_E: `ALU_BIT_b_n(3, E)
BIT4_E: `ALU_BIT_b_n(4, E)
BIT5_E: `ALU_BIT_b_n(5, E)
BIT6_E: `ALU_BIT_b_n(6, E)
BIT7_E: `ALU_BIT_b_n(7, E)

BIT0_H: `ALU_BIT_b_n(0, H)
BIT1_H: `ALU_BIT_b_n(1, H)
BIT2_H: `ALU_BIT_b_n(2, H)
BIT3_H: `ALU_BIT_b_n(3, H)
BIT4_H: `ALU_BIT_b_n(4, H)
BIT5_H: `ALU_BIT_b_n(5, H)
BIT6_H: `ALU_BIT_b_n(6, H)
BIT7_H: `ALU_BIT_b_n(7, H)
```

```
BIT0_L: `ALU_BIT_b_n(0, L)

BIT1_L: `ALU_BIT_b_n(1, L)

BIT2_L: `ALU_BIT_b_n(2, L)

BIT3_L: `ALU_BIT_b_n(3, L)

BIT4_L: `ALU_BIT_b_n(4, L)

BIT5_L: `ALU_BIT_b_n(5, L)

BIT6_L: `ALU_BIT_b_n(6, L)

BIT7_L: `ALU_BIT_b_n(7, L)


BIT0_T: `ALU_BIT_b_n(0, T)

BIT1_T: `ALU_BIT_b_n(1, T)

BIT2_T: `ALU_BIT_b_n(2, T)

BIT3_T: `ALU_BIT_b_n(3, T)

BIT4_T: `ALU_BIT_b_n(4, T)

BIT5_T: `ALU_BIT_b_n(5, T)

BIT6_T: `ALU_BIT_b_n(6, T)

BIT7_T: `ALU_BIT_b_n(7, T)


RES0_A: `ALU_SETRST_op_b_n(RES, 0, A)

RES1_A: `ALU_SETRST_op_b_n(RES, 1, A)

RES2_A: `ALU_SETRST_op_b_n(RES, 2, A)

RES3_A: `ALU_SETRST_op_b_n(RES, 3, A)

RES4_A: `ALU_SETRST_op_b_n(RES, 4, A)

RES5_A: `ALU_SETRST_op_b_n(RES, 5, A)

RES6_A: `ALU_SETRST_op_b_n(RES, 6, A)
```

```
RES7_A: `ALU_SETRST_op_b_n(RES, 7, A)


RES0_B: `ALU_SETRST_op_b_n(RES, 0, B)
RES1_B: `ALU_SETRST_op_b_n(RES, 1, B)
RES2_B: `ALU_SETRST_op_b_n(RES, 2, B)
RES3_B: `ALU_SETRST_op_b_n(RES, 3, B)
RES4_B: `ALU_SETRST_op_b_n(RES, 4, B)
RES5_B: `ALU_SETRST_op_b_n(RES, 5, B)
RES6_B: `ALU_SETRST_op_b_n(RES, 6, B)
RES7_B: `ALU_SETRST_op_b_n(RES, 7, B)

RES0_C: `ALU_SETRST_op_b_n(RES, 0, C)
RES1_C: `ALU_SETRST_op_b_n(RES, 1, C)
RES2_C: `ALU_SETRST_op_b_n(RES, 2, C)
RES3_C: `ALU_SETRST_op_b_n(RES, 3, C)
RES4_C: `ALU_SETRST_op_b_n(RES, 4, C)
RES5_C: `ALU_SETRST_op_b_n(RES, 5, C)
RES6_C: `ALU_SETRST_op_b_n(RES, 6, C)
RES7_C: `ALU_SETRST_op_b_n(RES, 7, C)

RES0_D: `ALU_SETRST_op_b_n(RES, 0, D)
RES1_D: `ALU_SETRST_op_b_n(RES, 1, D)
RES2_D: `ALU_SETRST_op_b_n(RES, 2, D)
RES3_D: `ALU_SETRST_op_b_n(RES, 3, D)
RES4_D: `ALU_SETRST_op_b_n(RES, 4, D)
RES5_D: `ALU_SETRST_op_b_n(RES, 5, D)
```

```
RES6_D: `ALU_SETRST_op_b_n(RES, 6, D)
RES7_D: `ALU_SETRST_op_b_n(RES, 7, D)

RES0_E: `ALU_SETRST_op_b_n(RES, 0, E)
RES1_E: `ALU_SETRST_op_b_n(RES, 1, E)
RES2_E: `ALU_SETRST_op_b_n(RES, 2, E)
RES3_E: `ALU_SETRST_op_b_n(RES, 3, E)
RES4_E: `ALU_SETRST_op_b_n(RES, 4, E)
RES5_E: `ALU_SETRST_op_b_n(RES, 5, E)
RES6_E: `ALU_SETRST_op_b_n(RES, 6, E)
RES7_E: `ALU_SETRST_op_b_n(RES, 7, E)

RES0_H: `ALU_SETRST_op_b_n(RES, 0, H)
RES1_H: `ALU_SETRST_op_b_n(RES, 1, H)
RES2_H: `ALU_SETRST_op_b_n(RES, 2, H)
RES3_H: `ALU_SETRST_op_b_n(RES, 3, H)
RES4_H: `ALU_SETRST_op_b_n(RES, 4, H)
RES5_H: `ALU_SETRST_op_b_n(RES, 5, H)
RES6_H: `ALU_SETRST_op_b_n(RES, 6, H)
RES7_H: `ALU_SETRST_op_b_n(RES, 7, H)

RES0_L: `ALU_SETRST_op_b_n(RES, 0, L)
RES1_L: `ALU_SETRST_op_b_n(RES, 1, L)
RES2_L: `ALU_SETRST_op_b_n(RES, 2, L)
RES3_L: `ALU_SETRST_op_b_n(RES, 3, L)
RES4_L: `ALU_SETRST_op_b_n(RES, 4, L)
```

```
RES5_L: `ALU_SETRST_op_b_n(RES, 5, L)

RES6_L: `ALU_SETRST_op_b_n(RES, 6, L)

RES7_L: `ALU_SETRST_op_b_n(RES, 7, L)


RES0_T: `ALU_SETRST_op_b_n(RES, 0, T)

RES1_T: `ALU_SETRST_op_b_n(RES, 1, T)

RES2_T: `ALU_SETRST_op_b_n(RES, 2, T)

RES3_T: `ALU_SETRST_op_b_n(RES, 3, T)

RES4_T: `ALU_SETRST_op_b_n(RES, 4, T)

RES5_T: `ALU_SETRST_op_b_n(RES, 5, T)

RES6_T: `ALU_SETRST_op_b_n(RES, 6, T)

RES7_T: `ALU_SETRST_op_b_n(RES, 7, T)


SET0_A: `ALU_SETRST_op_b_n(SET, 0, A)

SET1_A: `ALU_SETRST_op_b_n(SET, 1, A)

SET2_A: `ALU_SETRST_op_b_n(SET, 2, A)

SET3_A: `ALU_SETRST_op_b_n(SET, 3, A)

SET4_A: `ALU_SETRST_op_b_n(SET, 4, A)

SET5_A: `ALU_SETRST_op_b_n(SET, 5, A)

SET6_A: `ALU_SETRST_op_b_n(SET, 6, A)

SET7_A: `ALU_SETRST_op_b_n(SET, 7, A)


SET0_B: `ALU_SETRST_op_b_n(SET, 0, B)

SET1_B: `ALU_SETRST_op_b_n(SET, 1, B)

SET2_B: `ALU_SETRST_op_b_n(SET, 2, B)

SET3_B: `ALU_SETRST_op_b_n(SET, 3, B)
```

```
SET4_B: `ALU_SETRST_op_b_n(SET, 4, B)

SET5_B: `ALU_SETRST_op_b_n(SET, 5, B)

SET6_B: `ALU_SETRST_op_b_n(SET, 6, B)

SET7_B: `ALU_SETRST_op_b_n(SET, 7, B)


SET0_C: `ALU_SETRST_op_b_n(SET, 0, C)

SET1_C: `ALU_SETRST_op_b_n(SET, 1, C)

SET2_C: `ALU_SETRST_op_b_n(SET, 2, C)

SET3_C: `ALU_SETRST_op_b_n(SET, 3, C)

SET4_C: `ALU_SETRST_op_b_n(SET, 4, C)

SET5_C: `ALU_SETRST_op_b_n(SET, 5, C)

SET6_C: `ALU_SETRST_op_b_n(SET, 6, C)

SET7_C: `ALU_SETRST_op_b_n(SET, 7, C)


SET0_D: `ALU_SETRST_op_b_n(SET, 0, D)

SET1_D: `ALU_SETRST_op_b_n(SET, 1, D)

SET2_D: `ALU_SETRST_op_b_n(SET, 2, D)

SET3_D: `ALU_SETRST_op_b_n(SET, 3, D)

SET4_D: `ALU_SETRST_op_b_n(SET, 4, D)

SET5_D: `ALU_SETRST_op_b_n(SET, 5, D)

SET6_D: `ALU_SETRST_op_b_n(SET, 6, D)

SET7_D: `ALU_SETRST_op_b_n(SET, 7, D)


SET0_E: `ALU_SETRST_op_b_n(SET, 0, E)

SET1_E: `ALU_SETRST_op_b_n(SET, 1, E)

SET2_E: `ALU_SETRST_op_b_n(SET, 2, E)
```

```
SET3_E: `ALU_SETRST_op_b_n(SET, 3, E)

SET4_E: `ALU_SETRST_op_b_n(SET, 4, E)

SET5_E: `ALU_SETRST_op_b_n(SET, 5, E)

SET6_E: `ALU_SETRST_op_b_n(SET, 6, E)

SET7_E: `ALU_SETRST_op_b_n(SET, 7, E)


SET0_H: `ALU_SETRST_op_b_n(SET, 0, H)

SET1_H: `ALU_SETRST_op_b_n(SET, 1, H)

SET2_H: `ALU_SETRST_op_b_n(SET, 2, H)

SET3_H: `ALU_SETRST_op_b_n(SET, 3, H)

SET4_H: `ALU_SETRST_op_b_n(SET, 4, H)

SET5_H: `ALU_SETRST_op_b_n(SET, 5, H)

SET6_H: `ALU_SETRST_op_b_n(SET, 6, H)

SET7_H: `ALU_SETRST_op_b_n(SET, 7, H)


SET0_L: `ALU_SETRST_op_b_n(SET, 0, L)

SET1_L: `ALU_SETRST_op_b_n(SET, 1, L)

SET2_L: `ALU_SETRST_op_b_n(SET, 2, L)

SET3_L: `ALU_SETRST_op_b_n(SET, 3, L)

SET4_L: `ALU_SETRST_op_b_n(SET, 4, L)

SET5_L: `ALU_SETRST_op_b_n(SET, 5, L)

SET6_L: `ALU_SETRST_op_b_n(SET, 6, L)

SET7_L: `ALU_SETRST_op_b_n(SET, 7, L)


SET0_T: `ALU_SETRST_op_b_n(SET, 0, T)

SET1_T: `ALU_SETRST_op_b_n(SET, 1, T)
```

```
            SET2_T: `ALU_SETRST_op_b_n(SET, 2, T)
            SET3_T: `ALU_SETRST_op_b_n(SET, 3, T)
            SET4_T: `ALU_SETRST_op_b_n(SET, 4, T)
            SET5_T: `ALU_SETRST_op_b_n(SET, 5, T)
            SET6_T: `ALU_SETRST_op_b_n(SET, 6, T)
            SET7_T: `ALU_SETRST_op_b_n(SET, 7, T)

            EI: IME_NEXT = 1;

            RST_IF:
            begin
                DATA_out = DATA_in;
                for (int i = 0; i < 5; i++)
                begin
                    if (INTQ_INT[i])
                    begin
                        DATA_out[i] = 0;
                        break;
                    end
                end
            end

            default: ;

        endcase
        // Patch
```

```
                    if ((INST == 8'hC1 || INST == 8'hD1 || INST == 8'hE1 || INST
== 8'hF1) && !isCB && !isINT && cur_risc_num == 4)
                    `INC_nn(SPh, SPl)

                    if ((RISC_OPCODE[cur_risc_num] == RLC_A ||
RISC_OPCODE[cur_risc_num] == RL_A ||
                         RISC_OPCODE[cur_risc_num] == RRC_A ||
RISC_OPCODE[cur_risc_num] == RR_A)  && !isCB && !isINT)
                    begin
                        CPU_REG_NEXT.F = ALU_STATUS & 8'b0001_1111;
                    end

                    if ((INST == 8'h09 || INST == 8'h19 || INST == 8'h29 || INST
== 8'h39) && !isCB && !isINT)
                    begin
                        CPU_REG_NEXT.F = (ALU_STATUS & 8'b0111_1111) | (CPU_REG.F
& 8'b1000_0000);  // Dont change Zero Flag
                    end

                    CPU_REG_NEXT.F = CPU_REG_NEXT.F & 8'b1111_0000;

                    if (CPU_STATE_NEXT == CPU_IF) ADDR_NEXT = CPU_REG_NEXT.PC; //
When PC is update at the last cycle, ADDR won't change in time, fix this
                end

            endcase
```

```
end



endmodule



module GB_Z80_DECODER
(
    input logic [7:0] CPU_OPCODE,
    input logic [4:0] INTQ,
    input logic isCB,
    input logic isINT,
    input logic [7:0] FLAG,
    output GB_Z80_RISC_OPCODE RISC_OPCODE[0:10],
    output logic [5:0] NUM_Tcnt, // How many RISC opcodes in total (1-5)
    output logic isPCMEM [0:10]
);

always_comb
begin

for (int i = 0; i <= 10; i ++)
begin
    RISC_OPCODE[i] = NOP;
    isPCMEM[i] = 0;
end
```

```
NUM_Tcnt = 6'd4;


if (!isINT)
begin
unique case ( {isCB, CPU_OPCODE} )
    9'h000: RISC_OPCODE[0] = NOP;
    9'h001: `DECODER_LDnn_d16(B, C)
    9'h002: `DECODER_LDnn_A(BC)
    9'h003: `DECODER_INC_nn(BC)
    9'h004: RISC_OPCODE[0] = INC_B;
    9'h005: RISC_OPCODE[0] = DEC_B;
    9'h006: `DECODER_LDn_d8(B)
    9'h007: RISC_OPCODE[0] = RLC_A;
    9'h008: `DECODER_LD_a16_SP
    9'h009: `DECODER_ADDHL_nn(B, C)
    9'h00A: `DECODER_LDA_nn(BC)
    9'h00B: `DECODER_DEC_nn(BC)
    9'h00C: RISC_OPCODE[0] = INC_C;
    9'h00D: RISC_OPCODE[0] = DEC_C;
    9'h00E:  `DECODER_LDn_d8(C)
    9'h00F: RISC_OPCODE[0] = RRC_A;
    9'h010: // STOP 0
    begin
        RISC_OPCODE[0] = NOP; // STOP not implemented yet
    end
```

```
9'h011: `DECODER_LDnn_d16(D, E)
9'h012: `DECODER_LDnn_A(DE)
9'h013: `DECODER_INC_nn(DE)
9'h014: RISC_OPCODE[0] = INC_D;
9'h015: RISC_OPCODE[0] = DEC_D;
9'h016: `DECODER_LDn_d8(D)
9'h017: RISC_OPCODE[0]= RL_A;
9'h018: // JR r8
begin
    RISC_OPCODE[2] = JP_R8;
    NUM_Tcnt = 6'd12;
end
9'h019: `DECODER_ADDHL_nn(D, E)
9'h01A: `DECODER_LDA_nn(DE)
9'h01B: `DECODER_DEC_nn(DE)
9'h01C: RISC_OPCODE[0] = INC_E;
9'h01D: RISC_OPCODE[0] = DEC_E;
9'h01E: `DECODER_LDn_d8(E)
9'h01F: RISC_OPCODE[0] = RR_A;
9'h020: // JR NZ,r8
    begin
        RISC_OPCODE[2] = JP_NZR8;
        NUM_Tcnt = FLAG[7] ? 6'd8 : 6'd12;
    end
9'h021: `DECODER_LDnn_d16(H, L)
9'h022: `DECODER_LD_HL_INC_A
```

```
9'h023: `DECODER_INC_nn(HL)
9'h024: RISC_OPCODE[0] = INC_H;
9'h025: RISC_OPCODE[0] = DEC_H;
9'h026: `DECODER_LDn_d8(H)
9'h027: RISC_OPCODE[0] = DAA;
9'h028: // JR Z,r8
    begin
        RISC_OPCODE[2] = JP_ZR8;
        NUM_Tcnt = FLAG[7] ? 6'd12 : 6'd8;
    end
9'h029: `DECODER_ADDHL_nn(H, L)
9'h02A: `DECODER_LD_A_HL_INC
9'h02B: `DECODER_DEC_nn(HL)
9'h02C: RISC_OPCODE[0] = INC_L;
9'h02D: RISC_OPCODE[0] = DEC_L;
9'h02E: `DECODER_LDn_d8(L)
9'h02F: RISC_OPCODE[0] = CPL;
9'h030:
    begin
        RISC_OPCODE[2] = JP_NCR8;
        NUM_Tcnt = FLAG[4] ? 6'd8 : 6'd12;
    end
9'h031: `DECODER_LDnn_d16(SPh, SPl)
9'h032: `DECODER_LD_HL_DEC_A
9'h033: `DECODER_INC_nn(SP)
9'h034: `DECODER_INC_MEM_HL
```

```
9'h035: `DECODER_DEC_MEM_HL
9'h036: `DECODER_LD_MEM_HL_d8
9'h037: RISC_OPCODE[0] = SCF;
9'h038:
    begin
        RISC_OPCODE[2] = JP_CR8;
        NUM_Tcnt = FLAG[4] ? 6'd12 : 6'd8;
    end
9'h039: `DECODER_ADDHL_nn(SPh, SPl)
9'h03A: `DECODER_LD_A_HL_DEC
9'h03B: `DECODER_DEC_nn(SP)
9'h03C: RISC_OPCODE[0] = INC_A;
9'h03D: RISC_OPCODE[0] = DEC_A;
9'h03E: `DECODER_LDn_d8(A)
9'h03F: RISC_OPCODE[0] = CCF;
9'h040: RISC_OPCODE[0] = LD_BB;
9'h041: RISC_OPCODE[0] = LD_BC;
9'h042: RISC_OPCODE[0] = LD_BD;
9'h043: RISC_OPCODE[0] = LD_BE;
9'h044: RISC_OPCODE[0] = LD_BH;
9'h045: RISC_OPCODE[0] = LD_BL;
9'h046: `DECODER_LD_n_MEM_HL(B)
9'h047: RISC_OPCODE[0] = LD_BA;
9'h048: RISC_OPCODE[0] = LD_CB;
9'h049: RISC_OPCODE[0] = LD_CC;
9'h04A: RISC_OPCODE[0] = LD_CD;
```

```
9'h04B: RISC_OPCODE[0] = LD_CE;
9'h04C: RISC_OPCODE[0] = LD_CH;
9'h04D: RISC_OPCODE[0] = LD_CL;
9'h04E: `DECODER_LD_n_MEM_HL(C)
9'h04F: RISC_OPCODE[0] = LD_CA;
9'h050: RISC_OPCODE[0] = LD_DB;
9'h051: RISC_OPCODE[0] = LD_DC;
9'h052: RISC_OPCODE[0] = LD_DD;
9'h053: RISC_OPCODE[0] = LD_DE;
9'h054: RISC_OPCODE[0] = LD_DH;
9'h055: RISC_OPCODE[0] = LD_DL;
9'h056: `DECODER_LD_n_MEM_HL(D)
9'h057: RISC_OPCODE[0] = LD_DA;
9'h058: RISC_OPCODE[0] = LD_EB;
9'h059: RISC_OPCODE[0] = LD_EC;
9'h05A: RISC_OPCODE[0] = LD_ED;
9'h05B: RISC_OPCODE[0] = LD_EE;
9'h05C: RISC_OPCODE[0] = LD_EH;
9'h05D: RISC_OPCODE[0] = LD_EL;
9'h05E: `DECODER_LD_n_MEM_HL(E)
9'h05F: RISC_OPCODE[0] = LD_EA;
9'h060: RISC_OPCODE[0] = LD_HB;
9'h061: RISC_OPCODE[0] = LD_HC;
9'h062: RISC_OPCODE[0] = LD_HD;
9'h063: RISC_OPCODE[0] = LD_HE;
9'h064: RISC_OPCODE[0] = LD_HH;
```

```
9'h065: RISC_OPCODE[0] = LD_HL;
9'h066: `DECODER_LD_n_MEM_HL(H)
9'h067: RISC_OPCODE[0] = LD_HA;
9'h068: RISC_OPCODE[0] = LD_LB;
9'h069: RISC_OPCODE[0] = LD_LC;
9'h06A: RISC_OPCODE[0] = LD_LD;
9'h06B: RISC_OPCODE[0] = LD_LE;
9'h06C: RISC_OPCODE[0] = LD_LH;
9'h06D: RISC_OPCODE[0] = LD_LL;
9'h06E: `DECODER_LD_n_MEM_HL(L)
9'h06F: RISC_OPCODE[0] = LD_LA;
9'h070: `DECODER_LD_MEM_HL_n(B)
9'h071: `DECODER_LD_MEM_HL_n(C)
9'h072: `DECODER_LD_MEM_HL_n(D)
9'h073: `DECODER_LD_MEM_HL_n(E)
9'h074: `DECODER_LD_MEM_HL_n(H)
9'h075: `DECODER_LD_MEM_HL_n(L)
9'h076: RISC_OPCODE[0] = HALT;
9'h077: `DECODER_LD_MEM_HL_n(A)
9'h078: RISC_OPCODE[0] = LD_AB;
9'h079: RISC_OPCODE[0] = LD_AC;
9'h07A: RISC_OPCODE[0] = LD_AD;
9'h07B: RISC_OPCODE[0] = LD_AE;
9'h07C: RISC_OPCODE[0] = LD_AH;
9'h07D: RISC_OPCODE[0] = LD_AL;
9'h07E: `DECODER_LD_n_MEM_HL(A)
```

```
9'h07F: RISC_OPCODE[0] = LD_AA;
9'h080: `DECODER_ALU_op_n(ADD, B)
9'h081: `DECODER_ALU_op_n(ADD, C)
9'h082: `DECODER_ALU_op_n(ADD, D)
9'h083: `DECODER_ALU_op_n(ADD, E)
9'h084: `DECODER_ALU_op_n(ADD, H)
9'h085: `DECODER_ALU_op_n(ADD, L)
9'h086: `DECODER_ALU_op_MEM_HL(ADD)
9'h087: `DECODER_ALU_op_n(ADD, A)
9'h088: `DECODER_ALU_op_n(ADC, B)
9'h089: `DECODER_ALU_op_n(ADC, C)
9'h08A: `DECODER_ALU_op_n(ADC, D)
9'h08B: `DECODER_ALU_op_n(ADC, E)
9'h08C: `DECODER_ALU_op_n(ADC, H)
9'h08D: `DECODER_ALU_op_n(ADC, L)
9'h08E: `DECODER_ALU_op_MEM_HL(ADC)
9'h08F: `DECODER_ALU_op_n(ADC, A)
9'h090: `DECODER_ALU_op_n(SUB, B)
9'h091: `DECODER_ALU_op_n(SUB, C)
9'h092: `DECODER_ALU_op_n(SUB, D)
9'h093: `DECODER_ALU_op_n(SUB, E)
9'h094: `DECODER_ALU_op_n(SUB, H)
9'h095: `DECODER_ALU_op_n(SUB, L)
9'h096: `DECODER_ALU_op_MEM_HL(SUB)
9'h097: `DECODER_ALU_op_n(SUB, A)
9'h098: `DECODER_ALU_op_n(SBC, B)
```

```
9'h099: `DECODER_ALU_op_n(SBC, C)
9'h09A: `DECODER_ALU_op_n(SBC, D)
9'h09B: `DECODER_ALU_op_n(SBC, E)
9'h09C: `DECODER_ALU_op_n(SBC, H)
9'h09D: `DECODER_ALU_op_n(SBC, L)
9'h09E: `DECODER_ALU_op_MEM_HL(SBC)
9'h09F: `DECODER_ALU_op_n(SBC, A)
9'h0A0: `DECODER_ALU_op_n(AND, B)
9'h0A1: `DECODER_ALU_op_n(AND, C)
9'h0A2: `DECODER_ALU_op_n(AND, D)
9'h0A3: `DECODER_ALU_op_n(AND, E)
9'h0A4: `DECODER_ALU_op_n(AND, H)
9'h0A5: `DECODER_ALU_op_n(AND, L)
9'h0A6: `DECODER_ALU_op_MEM_HL(AND)
9'h0A7: `DECODER_ALU_op_n(AND, A)
9'h0A8: `DECODER_ALU_op_n(XOR, B)
9'h0A9: `DECODER_ALU_op_n(XOR, C)
9'h0AA: `DECODER_ALU_op_n(XOR, D)
9'h0AB: `DECODER_ALU_op_n(XOR, E)
9'h0AC: `DECODER_ALU_op_n(XOR, H)
9'h0AD: `DECODER_ALU_op_n(XOR, L)
9'h0AE: `DECODER_ALU_op_MEM_HL(XOR)
9'h0AF: `DECODER_ALU_op_n(XOR, A)
9'h0B0: `DECODER_ALU_op_n(OR, B)
9'h0B1: `DECODER_ALU_op_n(OR, C)
9'h0B2: `DECODER_ALU_op_n(OR, D)
```

```
9'h0B3: `DECODER_ALU_op_n(OR, E)

9'h0B4: `DECODER_ALU_op_n(OR, H)

9'h0B5: `DECODER_ALU_op_n(OR, L)

9'h0B6: `DECODER_ALU_op_MEM_HL(OR)

9'h0B7: `DECODER_ALU_op_n(OR, A)

9'h0B8: `DECODER_ALU_op_n(CP, B)

9'h0B9: `DECODER_ALU_op_n(CP, C)

9'h0BA: `DECODER_ALU_op_n(CP, D)

9'h0BB: `DECODER_ALU_op_n(CP, E)

9'h0BC: `DECODER_ALU_op_n(CP, H)

9'h0BD: `DECODER_ALU_op_n(CP, L)

9'h0BE: `DECODER_ALU_op_MEM_HL(CP)

9'h0BF: `DECODER_ALU_op_n(CP, A)

9'h0C0: `DECODER_RET_NZ

9'h0C1: `DECODER_POP_nn(B, C)

9'h0C2: `DECODER_JP_NZ_a16

9'h0C3: `DECODER_JP_a16

9'h0C4: `DECODER_CALL_NZ_a16

9'h0C5: `DECODER_PUSH_nn(B, C)

9'h0C6: `DECODER_ALU_op_d8(ADD)

9'h0C7: `DECODER_RST(00)

9'h0C8: `DECODER_RET_Z

9'h0C9: `DECODER_RET

9'h0CA: `DECODER_JP_Z_a16

9'h0CB: ; // CB Prefix

9'h0CC: `DECODER_CALL_Z_a16
```

```
9'h0CD: `DECODER_CALL_a16
9'h0CE: `DECODER_ALU_op_d8(ADC)
9'h0CF: `DECODER_RST(08)
9'h0D0: `DECODER_RET_NC
9'h0D1: `DECODER_POP_nn(D, E)
9'h0D2: `DECODER_JP_NC_a16
9'h0D3: ; // Undefined
9'h0D4: `DECODER_CALL_NC_a16
9'h0D5: `DECODER_PUSH_nn(D, E)
9'h0D6: `DECODER_ALU_op_d8(SUB)
9'h0D7: `DECODER_RST(10)
9'h0D8: `DECODER_RET_C
9'h0D9: `DECODER_RETI
9'h0DA: `DECODER_JP_C_a16
9'h0DB: ; // Undefined
9'h0DC: `DECODER_CALL_C_a16
9'h0DD: ; // Undefined
9'h0DE: `DECODER_ALU_op_d8(SBC)
9'h0DF: `DECODER_RST(18)
9'h0E0: `DECODER_LDH_a8_A
9'h0E1: `DECODER_POP_nn(H, L)
9'h0E2: `DECODER_LDH_C_A
9'h0E3: ; // Undefined
9'h0E4: ; // Undefined
9'h0E5: `DECODER_PUSH_nn(H, L)
9'h0E6: `DECODER_ALU_op_d8(AND)
```

```
9'h0E7: `DECODER_RST(20)
9'h0E8: `DECODER_ADD_SP_R8
9'h0E9: RISC_OPCODE[0] = LD_PCHL;
9'h0EA: `DECODER_LD_a16_A
9'h0EB: ; // Undefined
9'h0EC: ; // Undefined
9'h0ED: ; // Undefined
9'h0EE: `DECODER_ALU_op_d8(XOR)
9'h0EF: `DECODER_RST(28)
9'h0F0: `DECODER_LDH_A_a8
9'h0F1: `DECODER_POP_nn(A, F)
9'h0F2: `DECODER_LDH_A_C
9'h0F3: RISC_OPCODE[0] = DI;
9'h0F4: ; // Undefined
9'h0F5: `DECODER_PUSH_nn(A, F)
9'h0F6: `DECODER_ALU_op_d8(OR)
9'h0F7: `DECODER_RST(30)
9'h0F8: `DECODER_LD_HL_SPR8
9'h0F9: begin RISC_OPCODE[2] = LD_SPHL; NUM_Tcnt = 6'd8; end
9'h0FA: `DECODER_LD_A_a16
9'h0FB: RISC_OPCODE[0] = EI;
9'h0FC: ; // Undefined
9'h0FD: ; // Undefined
9'h0FE: `DECODER_ALU_op_d8(CP)
9'h0FF: `DECODER_RST(38)
/* CB Commands */
```

```
9'h100: RISC_OPCODE[0] = RLC_B;
9'h101: RISC_OPCODE[0] = RLC_C;
9'h102: RISC_OPCODE[0] = RLC_D;
9'h103: RISC_OPCODE[0] = RLC_E;
9'h104: RISC_OPCODE[0] = RLC_H;
9'h105: RISC_OPCODE[0] = RLC_L;
9'h106: `DECODER_CB_ALU_op_MEM_HL(RLC)
9'h107: RISC_OPCODE[0] = RLC_A;
9'h108: RISC_OPCODE[0] = RRC_B;
9'h109: RISC_OPCODE[0] = RRC_C;
9'h10A: RISC_OPCODE[0] = RRC_D;
9'h10B: RISC_OPCODE[0] = RRC_E;
9'h10C: RISC_OPCODE[0] = RRC_H;
9'h10D: RISC_OPCODE[0] = RRC_L;
9'h10E: `DECODER_CB_ALU_op_MEM_HL(RRC)
9'h10F: RISC_OPCODE[0] = RRC_A;
9'h110: RISC_OPCODE[0] = RL_B;
9'h111: RISC_OPCODE[0] = RL_C;
9'h112: RISC_OPCODE[0] = RL_D;
9'h113: RISC_OPCODE[0] = RL_E;
9'h114: RISC_OPCODE[0] = RL_H;
9'h115: RISC_OPCODE[0] = RL_L;
9'h116: `DECODER_CB_ALU_op_MEM_HL(RL)
9'h117: RISC_OPCODE[0] = RL_A;
9'h118: RISC_OPCODE[0] = RR_B;
9'h119: RISC_OPCODE[0] = RR_C;
```

```
9'h11A: RISC_OPCODE[0] = RR_D;
9'h11B: RISC_OPCODE[0] = RR_E;
9'h11C: RISC_OPCODE[0] = RR_H;
9'h11D: RISC_OPCODE[0] = RR_L;
9'h11E: `DECODER_CB_ALU_op_MEM_HL(RR)
9'h11F: RISC_OPCODE[0] = RR_A;
9'h120: RISC_OPCODE[0] = SLA_B;
9'h121: RISC_OPCODE[0] = SLA_C;
9'h122: RISC_OPCODE[0] = SLA_D;
9'h123: RISC_OPCODE[0] = SLA_E;
9'h124: RISC_OPCODE[0] = SLA_H;
9'h125: RISC_OPCODE[0] = SLA_L;
9'h126: `DECODER_CB_ALU_op_MEM_HL(SLA)
9'h127: RISC_OPCODE[0] = SLA_A;
9'h128: RISC_OPCODE[0] = SRA_B;
9'h129: RISC_OPCODE[0] = SRA_C;
9'h12A: RISC_OPCODE[0] = SRA_D;
9'h12B: RISC_OPCODE[0] = SRA_E;
9'h12C: RISC_OPCODE[0] = SRA_H;
9'h12D: RISC_OPCODE[0] = SRA_L;
9'h12E: `DECODER_CB_ALU_op_MEM_HL(SRA)
9'h12F: RISC_OPCODE[0] = SRA_A;
9'h130: RISC_OPCODE[0] = SWAP_B;
9'h131: RISC_OPCODE[0] = SWAP_C;
9'h132: RISC_OPCODE[0] = SWAP_D;
9'h133: RISC_OPCODE[0] = SWAP_E;
```

```
9'h134: RISC_OPCODE[0] = SWAP_H;
9'h135: RISC_OPCODE[0] = SWAP_L;
9'h136: `DECODER_CB_ALU_op_MEM_HL(SWAP)
9'h137: RISC_OPCODE[0] = SWAP_A;
9'h138: RISC_OPCODE[0] = SRL_B;
9'h139: RISC_OPCODE[0] = SRL_C;
9'h13A: RISC_OPCODE[0] = SRL_D;
9'h13B: RISC_OPCODE[0] = SRL_E;
9'h13C: RISC_OPCODE[0] = SRL_H;
9'h13D: RISC_OPCODE[0] = SRL_L;
9'h13E: `DECODER_CB_ALU_op_MEM_HL(SRL)
9'h13F: RISC_OPCODE[0] = SRL_A;
9'h140: `DECODER_CB_BIT_op_b_n(BIT, 0, B)
9'h141: `DECODER_CB_BIT_op_b_n(BIT, 0, C)
9'h142: `DECODER_CB_BIT_op_b_n(BIT, 0, D)
9'h143: `DECODER_CB_BIT_op_b_n(BIT, 0, E)
9'h144: `DECODER_CB_BIT_op_b_n(BIT, 0, H)
9'h145: `DECODER_CB_BIT_op_b_n(BIT, 0, L)
9'h146: `DECODER_CB_BIT_op_b_MEM_HL(BIT, 0)
9'h147: `DECODER_CB_BIT_op_b_n(BIT, 0, A)
9'h148: `DECODER_CB_BIT_op_b_n(BIT, 1, B)
9'h149: `DECODER_CB_BIT_op_b_n(BIT, 1, C)
9'h14A: `DECODER_CB_BIT_op_b_n(BIT, 1, D)
9'h14B: `DECODER_CB_BIT_op_b_n(BIT, 1, E)
9'h14C: `DECODER_CB_BIT_op_b_n(BIT, 1, H)
9'h14D: `DECODER_CB_BIT_op_b_n(BIT, 1, L)
```

```
9'h14E: `DECODER_CB_BIT_op_b_MEM_HL(BIT, 1)
9'h14F: `DECODER_CB_BIT_op_b_n(BIT, 1, A)
9'h150: `DECODER_CB_BIT_op_b_n(BIT, 2, B)
9'h151: `DECODER_CB_BIT_op_b_n(BIT, 2, C)
9'h152: `DECODER_CB_BIT_op_b_n(BIT, 2, D)
9'h153: `DECODER_CB_BIT_op_b_n(BIT, 2, E)
9'h154: `DECODER_CB_BIT_op_b_n(BIT, 2, H)
9'h155: `DECODER_CB_BIT_op_b_n(BIT, 2, L)
9'h156: `DECODER_CB_BIT_op_b_MEM_HL(BIT, 2)
9'h157: `DECODER_CB_BIT_op_b_n(BIT, 2, A)
9'h158: `DECODER_CB_BIT_op_b_n(BIT, 3, B)
9'h159: `DECODER_CB_BIT_op_b_n(BIT, 3, C)
9'h15A: `DECODER_CB_BIT_op_b_n(BIT, 3, D)
9'h15B: `DECODER_CB_BIT_op_b_n(BIT, 3, E)
9'h15C: `DECODER_CB_BIT_op_b_n(BIT, 3, H)
9'h15D: `DECODER_CB_BIT_op_b_n(BIT, 3, L)
9'h15E: `DECODER_CB_BIT_op_b_MEM_HL(BIT, 3)
9'h15F: `DECODER_CB_BIT_op_b_n(BIT, 3, A)
9'h160: `DECODER_CB_BIT_op_b_n(BIT, 4, B)
9'h161: `DECODER_CB_BIT_op_b_n(BIT, 4, C)
9'h162: `DECODER_CB_BIT_op_b_n(BIT, 4, D)
9'h163: `DECODER_CB_BIT_op_b_n(BIT, 4, E)
9'h164: `DECODER_CB_BIT_op_b_n(BIT, 4, H)
9'h165: `DECODER_CB_BIT_op_b_n(BIT, 4, L)
9'h166: `DECODER_CB_BIT_op_b_MEM_HL(BIT, 4)
9'h167: `DECODER_CB_BIT_op_b_n(BIT, 4, A)
```

```
9'h168: `DECODER_CB_BIT_op_b_n(BIT, 5, B)
9'h169: `DECODER_CB_BIT_op_b_n(BIT, 5, C)
9'h16A: `DECODER_CB_BIT_op_b_n(BIT, 5, D)
9'h16B: `DECODER_CB_BIT_op_b_n(BIT, 5, E)
9'h16C: `DECODER_CB_BIT_op_b_n(BIT, 5, H)
9'h16D: `DECODER_CB_BIT_op_b_n(BIT, 5, L)
9'h16E: `DECODER_CB_BIT_op_b_MEM_HL(BIT, 5)
9'h16F: `DECODER_CB_BIT_op_b_n(BIT, 5, A)
9'h170: `DECODER_CB_BIT_op_b_n(BIT, 6, B)
9'h171: `DECODER_CB_BIT_op_b_n(BIT, 6, C)
9'h172: `DECODER_CB_BIT_op_b_n(BIT, 6, D)
9'h173: `DECODER_CB_BIT_op_b_n(BIT, 6, E)
9'h174: `DECODER_CB_BIT_op_b_n(BIT, 6, H)
9'h175: `DECODER_CB_BIT_op_b_n(BIT, 6, L)
9'h176: `DECODER_CB_BIT_op_b_MEM_HL(BIT, 6)
9'h177: `DECODER_CB_BIT_op_b_n(BIT, 6, A)
9'h178: `DECODER_CB_BIT_op_b_n(BIT, 7, B)
9'h179: `DECODER_CB_BIT_op_b_n(BIT, 7, C)
9'h17A: `DECODER_CB_BIT_op_b_n(BIT, 7, D)
9'h17B: `DECODER_CB_BIT_op_b_n(BIT, 7, E)
9'h17C: `DECODER_CB_BIT_op_b_n(BIT, 7, H)
9'h17D: `DECODER_CB_BIT_op_b_n(BIT, 7, L)
9'h17E: `DECODER_CB_BIT_op_b_MEM_HL(BIT, 7)
9'h17F: `DECODER_CB_BIT_op_b_n(BIT, 7, A)
9'h180: `DECODER_CB_BIT_op_b_n(RES, 0, B)
9'h181: `DECODER_CB_BIT_op_b_n(RES, 0, C)
```

```
9'h182: `DECODER_CB_BIT_op_b_n(RES, 0, D)
9'h183: `DECODER_CB_BIT_op_b_n(RES, 0, E)
9'h184: `DECODER_CB_BIT_op_b_n(RES, 0, H)
9'h185: `DECODER_CB_BIT_op_b_n(RES, 0, L)
9'h186: `DECODER_CB_RES_SET_op_b_MEM_HL(RES, 0)
9'h187: `DECODER_CB_BIT_op_b_n(RES, 0, A)
9'h188: `DECODER_CB_BIT_op_b_n(RES, 1, B)
9'h189: `DECODER_CB_BIT_op_b_n(RES, 1, C)
9'h18A: `DECODER_CB_BIT_op_b_n(RES, 1, D)
9'h18B: `DECODER_CB_BIT_op_b_n(RES, 1, E)
9'h18C: `DECODER_CB_BIT_op_b_n(RES, 1, H)
9'h18D: `DECODER_CB_BIT_op_b_n(RES, 1, L)
9'h18E: `DECODER_CB_RES_SET_op_b_MEM_HL(RES, 1)
9'h18F: `DECODER_CB_BIT_op_b_n(RES, 1, A)
9'h190: `DECODER_CB_BIT_op_b_n(RES, 2, B)
9'h191: `DECODER_CB_BIT_op_b_n(RES, 2, C)
9'h192: `DECODER_CB_BIT_op_b_n(RES, 2, D)
9'h193: `DECODER_CB_BIT_op_b_n(RES, 2, E)
9'h194: `DECODER_CB_BIT_op_b_n(RES, 2, H)
9'h195: `DECODER_CB_BIT_op_b_n(RES, 2, L)
9'h196: `DECODER_CB_RES_SET_op_b_MEM_HL(RES, 2)
9'h197: `DECODER_CB_BIT_op_b_n(RES, 2, A)
9'h198: `DECODER_CB_BIT_op_b_n(RES, 3, B)
9'h199: `DECODER_CB_BIT_op_b_n(RES, 3, C)
9'h19A: `DECODER_CB_BIT_op_b_n(RES, 3, D)
9'h19B: `DECODER_CB_BIT_op_b_n(RES, 3, E)
```

```
9'h19C: `DECODER_CB_BIT_op_b_n(RES, 3, H)
9'h19D: `DECODER_CB_BIT_op_b_n(RES, 3, L)
9'h19E: `DECODER_CB_RES_SET_op_b_MEM_HL(RES, 3)
9'h19F: `DECODER_CB_BIT_op_b_n(RES, 3, A)
9'h1A0: `DECODER_CB_BIT_op_b_n(RES, 4, B)
9'h1A1: `DECODER_CB_BIT_op_b_n(RES, 4, C)
9'h1A2: `DECODER_CB_BIT_op_b_n(RES, 4, D)
9'h1A3: `DECODER_CB_BIT_op_b_n(RES, 4, E)
9'h1A4: `DECODER_CB_BIT_op_b_n(RES, 4, H)
9'h1A5: `DECODER_CB_BIT_op_b_n(RES, 4, L)
9'h1A6: `DECODER_CB_RES_SET_op_b_MEM_HL(RES, 4)
9'h1A7: `DECODER_CB_BIT_op_b_n(RES, 4, A)
9'h1A8: `DECODER_CB_BIT_op_b_n(RES, 5, B)
9'h1A9: `DECODER_CB_BIT_op_b_n(RES, 5, C)
9'h1AA: `DECODER_CB_BIT_op_b_n(RES, 5, D)
9'h1AB: `DECODER_CB_BIT_op_b_n(RES, 5, E)
9'h1AC: `DECODER_CB_BIT_op_b_n(RES, 5, H)
9'h1AD: `DECODER_CB_BIT_op_b_n(RES, 5, L)
9'h1AE: `DECODER_CB_RES_SET_op_b_MEM_HL(RES, 5)
9'h1AF: `DECODER_CB_BIT_op_b_n(RES, 5, A)
9'h1B0: `DECODER_CB_BIT_op_b_n(RES, 6, B)
9'h1B1: `DECODER_CB_BIT_op_b_n(RES, 6, C)
9'h1B2: `DECODER_CB_BIT_op_b_n(RES, 6, D)
9'h1B3: `DECODER_CB_BIT_op_b_n(RES, 6, E)
9'h1B4: `DECODER_CB_BIT_op_b_n(RES, 6, H)
9'h1B5: `DECODER_CB_BIT_op_b_n(RES, 6, L)
```

```
9'h1B6: `DECODER_CB_RES_SET_op_b_MEM_HL(RES, 6)
9'h1B7: `DECODER_CB_BIT_op_b_n(RES, 6, A)
9'h1B8: `DECODER_CB_BIT_op_b_n(RES, 7, B)
9'h1B9: `DECODER_CB_BIT_op_b_n(RES, 7, C)
9'h1BA: `DECODER_CB_BIT_op_b_n(RES, 7, D)
9'h1BB: `DECODER_CB_BIT_op_b_n(RES, 7, E)
9'h1BC: `DECODER_CB_BIT_op_b_n(RES, 7, H)
9'h1BD: `DECODER_CB_BIT_op_b_n(RES, 7, L)
9'h1BE: `DECODER_CB_RES_SET_op_b_MEM_HL(RES, 7)
9'h1BF: `DECODER_CB_BIT_op_b_n(RES, 7, A)
9'h1C0: `DECODER_CB_BIT_op_b_n(SET, 0, B)
9'h1C1: `DECODER_CB_BIT_op_b_n(SET, 0, C)
9'h1C2: `DECODER_CB_BIT_op_b_n(SET, 0, D)
9'h1C3: `DECODER_CB_BIT_op_b_n(SET, 0, E)
9'h1C4: `DECODER_CB_BIT_op_b_n(SET, 0, H)
9'h1C5: `DECODER_CB_BIT_op_b_n(SET, 0, L)
9'h1C6: `DECODER_CB_RES_SET_op_b_MEM_HL(SET, 0)
9'h1C7: `DECODER_CB_BIT_op_b_n(SET, 0, A)
9'h1C8: `DECODER_CB_BIT_op_b_n(SET, 1, B)
9'h1C9: `DECODER_CB_BIT_op_b_n(SET, 1, C)
9'h1CA: `DECODER_CB_BIT_op_b_n(SET, 1, D)
9'h1CB: `DECODER_CB_BIT_op_b_n(SET, 1, E)
9'h1CC: `DECODER_CB_BIT_op_b_n(SET, 1, H)
9'h1CD: `DECODER_CB_BIT_op_b_n(SET, 1, L)
9'h1CE: `DECODER_CB_RES_SET_op_b_MEM_HL(SET, 1)
9'h1CF: `DECODER_CB_BIT_op_b_n(SET, 1, A)
```

```
9'h1D0: `DECODER_CB_BIT_op_b_n(SET, 2, B)
9'h1D1: `DECODER_CB_BIT_op_b_n(SET, 2, C)
9'h1D2: `DECODER_CB_BIT_op_b_n(SET, 2, D)
9'h1D3: `DECODER_CB_BIT_op_b_n(SET, 2, E)
9'h1D4: `DECODER_CB_BIT_op_b_n(SET, 2, H)
9'h1D5: `DECODER_CB_BIT_op_b_n(SET, 2, L)
9'h1D6: `DECODER_CB_RES_SET_op_b_MEM_HL(SET, 2)
9'h1D7: `DECODER_CB_BIT_op_b_n(SET, 2, A)
9'h1D8: `DECODER_CB_BIT_op_b_n(SET, 3, B)
9'h1D9: `DECODER_CB_BIT_op_b_n(SET, 3, C)
9'h1DA: `DECODER_CB_BIT_op_b_n(SET, 3, D)
9'h1DB: `DECODER_CB_BIT_op_b_n(SET, 3, E)
9'h1DC: `DECODER_CB_BIT_op_b_n(SET, 3, H)
9'h1DD: `DECODER_CB_BIT_op_b_n(SET, 3, L)
9'h1DE: `DECODER_CB_RES_SET_op_b_MEM_HL(SET, 3)
9'h1DF: `DECODER_CB_BIT_op_b_n(SET, 3, A)
9'h1E0: `DECODER_CB_BIT_op_b_n(SET, 4, B)
9'h1E1: `DECODER_CB_BIT_op_b_n(SET, 4, C)
9'h1E2: `DECODER_CB_BIT_op_b_n(SET, 4, D)
9'h1E3: `DECODER_CB_BIT_op_b_n(SET, 4, E)
9'h1E4: `DECODER_CB_BIT_op_b_n(SET, 4, H)
9'h1E5: `DECODER_CB_BIT_op_b_n(SET, 4, L)
9'h1E6: `DECODER_CB_RES_SET_op_b_MEM_HL(SET, 4)
9'h1E7: `DECODER_CB_BIT_op_b_n(SET, 4, A)
9'h1E8: `DECODER_CB_BIT_op_b_n(SET, 5, B)
9'h1E9: `DECODER_CB_BIT_op_b_n(SET, 5, C)
```

```
        9'h1EA: `DECODER_CB_BIT_op_b_n(SET, 5, D)
        9'h1EB: `DECODER_CB_BIT_op_b_n(SET, 5, E)
        9'h1EC: `DECODER_CB_BIT_op_b_n(SET, 5, H)
        9'h1ED: `DECODER_CB_BIT_op_b_n(SET, 5, L)
        9'h1EE: `DECODER_CB_RES_SET_op_b_MEM_HL(SET, 5)
        9'h1EF: `DECODER_CB_BIT_op_b_n(SET, 5, A)
        9'h1F0: `DECODER_CB_BIT_op_b_n(SET, 6, B)
        9'h1F1: `DECODER_CB_BIT_op_b_n(SET, 6, C)
        9'h1F2: `DECODER_CB_BIT_op_b_n(SET, 6, D)
        9'h1F3: `DECODER_CB_BIT_op_b_n(SET, 6, E)
        9'h1F4: `DECODER_CB_BIT_op_b_n(SET, 6, H)
        9'h1F5: `DECODER_CB_BIT_op_b_n(SET, 6, L)
        9'h1F6: `DECODER_CB_RES_SET_op_b_MEM_HL(SET, 6)
        9'h1F7: `DECODER_CB_BIT_op_b_n(SET, 6, A)
        9'h1F8: `DECODER_CB_BIT_op_b_n(SET, 7, B)
        9'h1F9: `DECODER_CB_BIT_op_b_n(SET, 7, C)
        9'h1FA: `DECODER_CB_BIT_op_b_n(SET, 7, D)
        9'h1FB: `DECODER_CB_BIT_op_b_n(SET, 7, E)
        9'h1FC: `DECODER_CB_BIT_op_b_n(SET, 7, H)
        9'h1FD: `DECODER_CB_BIT_op_b_n(SET, 7, L)
        9'h1FE: `DECODER_CB_RES_SET_op_b_MEM_HL(SET, 7)
        9'h1FF: `DECODER_CB_BIT_op_b_n(SET, 7, A)
    endcase
    end
    else
    begin
```

```
        if (INTQ == 0) `DECODER_INTR(00)
        else
        begin
            for (int i = 0; i <= 4; i++)
            begin
                if(INTQ[i])
                begin
                    unique case (i)
                        0: `DECODER_INTR(40)
                        1: `DECODER_INTR(48)
                        2: `DECODER_INTR(50)
                        3: `DECODER_INTR(58)
                        4: `DECODER_INTR(60)
                    endcase
                    break;
                end
            end
        end
        NUM_Tcnt = 6'd20;
    end

    for (int i = 0; i <= 10; i++)
    begin
        if (RISC_OPCODE[i] == LD_BPC || RISC_OPCODE[i] == LD_CPC ||
            RISC_OPCODE[i] == LD_DPC || RISC_OPCODE[i] == LD_EPC ||
            RISC_OPCODE[i] == LD_HPC || RISC_OPCODE[i] == LD_LPC ||
```

```
            RISC_OPCODE[i] == LD_TPC || RISC_OPCODE[i] == LD_XPC ||
            RISC_OPCODE[i] == LD_APC ||
            RISC_OPCODE[i] == LD_PCB || RISC_OPCODE[i] == LD_PCC ||
            RISC_OPCODE[i] == LD_PCD || RISC_OPCODE[i] == LD_PCE ||
            RISC_OPCODE[i] == LD_PCH || RISC_OPCODE[i] == LD_PCL ||
            RISC_OPCODE[i] == LD_PCT ||
            RISC_OPCODE[i] == LD_PCSPl || RISC_OPCODE[i] == LD_PCSPh ||
            RISC_OPCODE[i] == LD_SPlPC || RISC_OPCODE[i] == LD_SPhPC ||
            RISC_OPCODE[i] == JP_R8 || RISC_OPCODE[i] == JP_NZR8 ||
            RISC_OPCODE[i] == JP_ZR8 || RISC_OPCODE[i] == JP_NCR8 ||
            RISC_OPCODE[i] == JP_CR8
            )
        begin
                isPCMEM[i] = 1;
        end
    end

    end


    endmodule



    module GB_Z80_ALU
    (
        input logic [7:0] OPD1_L,
        input logic [7:0] OPD2_L,
```

```
    input GB_Z80_ALU_OPCODE OPCODE,
    input logic [7:0] FLAG,    // the F register
    output logic [7:0] STATUS, // updated flag
    output logic [7:0] RESULT_L,
    output logic [7:0] RESULT_H // Not used for 8-bit ALU
);

// int is signed 32 bit 2 state integer
int opd1h_int;
int opd2h_int;
int opd16_int;
int result_int;
logic [7:0] status_int;

assign RESULT_L = result_int[7:0];
assign RESULT_H = result_int[15:8];
assign STATUS = status_int;
assign opd1h_int = {1'b0, OPD1_L};
assign opd2h_int = {1'b0, OPD2_L};

assign opd16_int = {OPD2_L, OPD1_L};

always_comb
begin

result_int = 0;
```

```
       status_int = FLAG;
       unique case (OPCODE)

           ALU_NOP : ;
           /* 8-bit Arithmetic */
           ALU_ADD, ALU_ADC :
           begin
               result_int = opd1h_int + opd2h_int + ((OPCODE == ALU_ADC) &
FLAG[4]);
               status_int[7] = RESULT_L == 0; // Zero Flag (Z)
               status_int[6] = 0; //Subtract Flag (N)
               status_int[5] = opd1h_int[3:0] + opd2h_int[3:0] + ((OPCODE ==
ALU_ADC) & FLAG[4])> 5'h0F; // Half Carry Flag (H)
               status_int[4] = result_int[8]; // Carry Flag (C)
           end
           ALU_SUB, ALU_SBC, ALU_CP : // SUB and CP are the same command to the
ALU
           begin
               result_int = opd2h_int - opd1h_int - ((OPCODE == ALU_SBC) &
FLAG[4]);
               status_int[7] = RESULT_L == 0;
               status_int[6] = 1;
               status_int[5] = {1'b0, opd2h_int[3:0]} < ({1'b0, opd1h_int[3:0]}
+ ((OPCODE == ALU_SBC) & FLAG[4]));
               status_int[4] = opd2h_int < (opd1h_int + ((OPCODE == ALU_SBC) &
FLAG[4]));
```

```
        end
        ALU_AND :
        begin
            result_int = opd1h_int & opd2h_int;
            status_int[7] = RESULT_L == 0;
            status_int[6] = 0;
            status_int[5] = 1;
            status_int[4] = 0;
        end
        ALU_OR :
        begin
            result_int = opd1h_int | opd2h_int;
            status_int[7] = RESULT_L == 0;
            status_int[6] = 0;
            status_int[5] = 0;
            status_int[4] = 0;
        end
        ALU_XOR :
        begin
            result_int = opd1h_int ^ opd2h_int;
            status_int[7] = RESULT_L == 0;
            status_int[6] = 0;
            status_int[5] = 0;
            status_int[4] = 0;
        end
        ALU_INC :
```

```
        begin
            result_int = opd1h_int + 1;
            status_int[7] = RESULT_L == 0;
            status_int[6] = 0;
            status_int[5] = opd1h_int[3:0] == 4'hF;
            status_int[4] = FLAG[4];
        end
        ALU_DEC :
        begin
            result_int = opd1h_int - 1;
            status_int[7] = RESULT_L == 0;
            status_int[6] = 1;
            status_int[5] = opd1h_int[3:0] == 4'h0;
            status_int[4] = FLAG[4];
        end
        ALU_CPL :
        begin
            for (int i = 0; i <= 7; i++)
                result_int[i] = ~opd1h_int[i];
            status_int[7] = FLAG[7];
            status_int[6] = 1;
            status_int[5] = 1;
            status_int[4] = FLAG[4];
        end
        ALU_BIT :
        begin
```

```
        status_int[7] = ~opd1h_int[opd2h_int];
        status_int[6] = 0;
        status_int[5] = 1;
        status_int[4] = FLAG[4];
    end
    ALU_SET :
    begin
        result_int = opd1h_int;
        result_int[opd2h_int] = 1;
    end
    ALU_RES :
    begin
        result_int = opd1h_int;
        result_int[opd2h_int] = 0;
    end
    ALU_INC16 :
    begin
        result_int = opd16_int + 1;
    end
    ALU_DEC16 :
    begin
        result_int = opd16_int - 1;
    end
    ALU_DAA :
    begin
        //https://ehaskins.com/2018-01-30%20Z80%20DAA/
```

```
                status_int[4] = 0;
                if (FLAG[5] || (!FLAG[6] && ((opd1h_int & 8'h0F) > 8'h09)))
result_int = result_int | 8'h06;
                if (FLAG[4] || (!FLAG[6] && (opd1h_int > 8'h99)))
                begin
                    result_int = result_int | 8'h60;
                    status_int[4] = 1;
                end
                result_int = FLAG[6] ? opd1h_int - result_int : opd1h_int +
result_int;
                status_int[7] = RESULT_L == 0;
                status_int[5] = 0;
            end
            SHIFTER_SWAP:
            begin
                result_int = {opd1h_int[3:0], opd1h_int[7:4]};
                status_int[7] = RESULT_L == 0;
                status_int[6] = 0;
                status_int[5] = 0;
                status_int[4] = 0;
            end
            SHIFTER_RLC :
            begin
                result_int = {opd1h_int[6:0], opd1h_int[7]};
                status_int[7] = RESULT_L == 0;
                status_int[6] = 0;
```

```
            status_int[5] = 0;
            status_int[4] = opd1h_int[7];
        end
        SHIFTER_RL :
        begin
            result_int = {opd1h_int[6:0], FLAG[4]};
            status_int[7] = RESULT_L == 0;
            status_int[6] = 0;
            status_int[5] = 0;
            status_int[4] = opd1h_int[7];
        end
        SHIFTER_RRC :
        begin
            result_int = {opd1h_int[0], opd1h_int[7:1]};
            status_int[7] = RESULT_L == 0;
            status_int[6] = 0;
            status_int[5] = 0;
            status_int[4] = opd1h_int[0];
        end
        SHIFTER_RR :
        begin
            result_int = {FLAG[4], opd1h_int[7:1]};
            status_int[7] = RESULT_L == 0;
            status_int[6] = 0;
            status_int[5] = 0;
            status_int[4] = opd1h_int[0];
```

```
        end
        SHIFTER_SLA :
        begin
            result_int = {opd1h_int[6:0], 1'b0};
            status_int[7] = RESULT_L == 0;
            status_int[6] = 0;
            status_int[5] = 0;
            status_int[4] = opd1h_int[7];
        end
        SHIFTER_SRA :
        begin
            result_int = {opd1h_int[7], opd1h_int[7:1]};
            status_int[7] = RESULT_L == 0;
            status_int[6] = 0;
            status_int[5] = 0;
            status_int[4] = opd1h_int[0];
        end
        SHIFTER_SRL :
        begin
            result_int = {1'b0, opd1h_int[7:1]};
            status_int[7] = RESULT_L == 0;
            status_int[6] = 0;
            status_int[5] = 0;
            status_int[4] = opd1h_int[0];
        end
    endcase
```

```
	end


	endmodule
```

GameBoy_Cartridge.sv

```
Unset
	/* This is the Cartridge Interface for SDRAM - MBC - GameBoy */
	`define SDRAM_RAM_BASE 26'h2000000
	module GameBoy_Cartridge
	(
		input logic clk,
		input logic reset,

		output logic SYNC_clk,
		output logic GB_rst,

		/* Avalon Master for On-Chip RAM Read/Write - 16bits of memory, 8
bits wide */
		output logic [25:0] address,
		output logic read,
		output logic write,
```

```
        input logic [7:0] readdata,
        output logic [7:0] writedata,
        input logic waitrequest,

        /* Avalon Slave for HPS */
        input logic [25:0] hps_address,
        input logic hps_read,
        input logic hps_write,
        output logic [7:0] hps_readdata,
        input logic [7:0] hps_writedata,
        output logic hps_waitrequest,

        /* GameBoy Cartrdige Conduit */
        input logic [15:0] CART_ADDR,
        output logic [7:0] CART_DATA_in,
        input logic [7:0] CART_DATA_out,
        input logic CART_RD,
        input logic CART_WR,

        output logic [9:0] LEDR
    );


    logic rom_load_done;
    assign LEDR = {10{rom_load_done}};
```

```
	logic double_speed, double_speed_req;
	always_ff @(posedge clk)
	begin
		if (reset)
		begin
			rom_load_done <= 0;
			double_speed_req <= 0;
		end
		else if (hps_address[25:0] == (`SDRAM_RAM_BASE - 1) && hps_write
&& !hps_waitrequest) rom_load_done <= hps_writedata[0];
		else if (hps_address[25:0] == (`SDRAM_RAM_BASE - 2) && hps_write
&& !hps_waitrequest);
		else if (hps_address[25:0] == (`SDRAM_RAM_BASE - 3) && hps_write
&& !hps_waitrequest);
		else if (hps_address[25:0] == (`SDRAM_RAM_BASE - 4) && hps_write
&& !hps_waitrequest);
		else if (hps_address[25:0] == (`SDRAM_RAM_BASE - 5) && hps_write
&& !hps_waitrequest);
		else if (hps_address[25:0] == (`SDRAM_RAM_BASE - 6) && hps_write
&& !hps_waitrequest) double_speed_req <= hps_writedata;
	end

	always_comb
	begin
		address = hps_address;
		read = hps_read;
```

```
        write = hps_write;
        hps_readdata = readdata;
        writedata = hps_writedata;
        hps_waitrequest = waitrequest;
        CART_DATA_in = 8'hFF;
        if (rom_load_done)
        begin
                address = {10'b0, CART_ADDR};
                read = CART_RD;
                write = 0;
                CART_DATA_in = readdata;
        end
    end

    logic [5:0] clk_div;
    always_ff @(posedge clk)
    begin
        if (reset || !rom_load_done)
        begin
                GB_rst <= 1;
            //clk_div <= 8'h00;
            double_speed <= 0;
        end
        else
        begin
```

```
                    if (clk_div[2] && !clk_div[3] && double_speed_req)
double_speed <= double_speed;
                    else if (!clk_div[2] && clk_div[3] && !double_speed_req)
double_speed <= double_speed;
                    else double_speed <= double_speed_req;

                    if (waitrequest && ((!double_speed && clk_div[3:0] ==
4'b0111) || (double_speed && clk_div[2:0] == 3'b011))) clk_div <= clk_div;
                    else clk_div <= clk_div + 1;
                    GB_rst <= 0;
             end
      end

      assign SYNC_clk = double_speed ? clk_div[2] : clk_div[3];

      endmodule
```

GameBoy_Cartridge_hw.tcl

```
Unset
      # TCL File Generated by Component Editor 18.1
      # Sun Jun 09 21:35:44 EDT 2019
      # DO NOT MODIFY
```

```
#
# GameBoy_Cartridge "GameBoy_Cartridge" v1.0
#  2019.06.09.21:35:44
# GameBoy Cartridge
#


#
# request TCL package from ACDS 16.1
#
package require -exact qsys 16.1



#
# module GameBoy_Cartridge
#
set_module_property DESCRIPTION "GameBoy Cartridge"
set_module_property NAME GameBoy_Cartridge
set_module_property VERSION 1.0
set_module_property INTERNAL false
set_module_property OPAQUE_ADDRESS_MAP true
set_module_property GROUP GameBoy
set_module_property AUTHOR ""
set_module_property DISPLAY_NAME GameBoy_Cartridge
set_module_property INSTANTIATE_IN_SYSTEM_MODULE true
set_module_property EDITABLE true
```

```
        set_module_property REPORT_TO_TALKBACK false
        set_module_property ALLOW_GREYBOX_GENERATION false
        set_module_property REPORT_HIERARCHY false



        #
        # file sets
        #
        add_fileset QUARTUS_SYNTH QUARTUS_SYNTH "" ""
        set_fileset_property QUARTUS_SYNTH TOP_LEVEL GameBoy_Cartridge
        set_fileset_property QUARTUS_SYNTH ENABLE_RELATIVE_INCLUDE_PATHS false
        set_fileset_property QUARTUS_SYNTH ENABLE_FILE_OVERWRITE_MODE false
        add_fileset_file GameBoy_Cartridge.sv SYSTEM_VERILOG PATH
GameBoy_Cartridge.sv TOP_LEVEL_FILE



        #
        # parameters
        #



        #
        # display items
        #
```

```
#
# connection point clock
#
add_interface clock clock end
set_interface_property clock clockRate 0
set_interface_property clock ENABLED true
set_interface_property clock EXPORT_OF ""
set_interface_property clock PORT_NAME_MAP ""
set_interface_property clock CMSIS_SVD_VARIABLES ""
set_interface_property clock SVD_ADDRESS_GROUP ""

add_interface_port clock clk clk Input 1



#
# connection point reset
#
add_interface reset reset end
set_interface_property reset associatedClock clock
set_interface_property reset synchronousEdges DEASSERT
set_interface_property reset ENABLED true
set_interface_property reset EXPORT_OF ""
set_interface_property reset PORT_NAME_MAP ""
set_interface_property reset CMSIS_SVD_VARIABLES ""
set_interface_property reset SVD_ADDRESS_GROUP ""
```

```
add_interface_port reset reset reset Input 1



#
# connection point GameBoy_Cartridge
#
add_interface GameBoy_Cartridge conduit end
set_interface_property GameBoy_Cartridge associatedClock ""
set_interface_property GameBoy_Cartridge associatedReset ""
set_interface_property GameBoy_Cartridge ENABLED true
set_interface_property GameBoy_Cartridge EXPORT_OF ""
set_interface_property GameBoy_Cartridge PORT_NAME_MAP ""
set_interface_property GameBoy_Cartridge CMSIS_SVD_VARIABLES ""
set_interface_property GameBoy_Cartridge SVD_ADDRESS_GROUP ""

add_interface_port GameBoy_Cartridge CART_ADDR cart_addr Input 16
add_interface_port GameBoy_Cartridge CART_RD cart_rd Input 1
add_interface_port GameBoy_Cartridge CART_WR cart_wr Input 1
add_interface_port GameBoy_Cartridge CART_DATA_in cart_data_in Output 8
add_interface_port GameBoy_Cartridge CART_DATA_out cart_data_out Input 8



#
# connection point avalon_master
#
add_interface avalon_master avalon start
```

```
set_interface_property avalon_master addressUnits SYMBOLS
set_interface_property avalon_master associatedClock clock
set_interface_property avalon_master associatedReset reset
set_interface_property avalon_master bitsPerSymbol 8
set_interface_property avalon_master burstOnBurstBoundariesOnly false
set_interface_property avalon_master burstcountUnits WORDS
set_interface_property avalon_master doStreamReads false
set_interface_property avalon_master doStreamWrites false
set_interface_property avalon_master holdTime 0
set_interface_property avalon_master linewrapBursts false
set_interface_property avalon_master maximumPendingReadTransactions 0
set_interface_property avalon_master maximumPendingWriteTransactions 0
set_interface_property avalon_master readLatency 0
set_interface_property avalon_master readWaitTime 1
set_interface_property avalon_master setupTime 0
set_interface_property avalon_master timingUnits Cycles
set_interface_property avalon_master writeWaitTime 0
set_interface_property avalon_master ENABLED true
set_interface_property avalon_master EXPORT_OF ""
set_interface_property avalon_master PORT_NAME_MAP ""
set_interface_property avalon_master CMSIS_SVD_VARIABLES ""
set_interface_property avalon_master SVD_ADDRESS_GROUP ""

add_interface_port avalon_master address address Output 26
add_interface_port avalon_master read read Output 1
add_interface_port avalon_master readdata readdata Input 8
```

```
add_interface_port avalon_master waitrequest waitrequest Input 1
add_interface_port avalon_master write write Output 1
add_interface_port avalon_master writedata writedata Output 8



#
# connection point reset_gameboy
#
add_interface reset_gameboy reset start
set_interface_property reset_gameboy associatedClock clock_gameboy
set_interface_property reset_gameboy associatedDirectReset ""
set_interface_property reset_gameboy associatedResetSinks ""
set_interface_property reset_gameboy synchronousEdges DEASSERT
set_interface_property reset_gameboy ENABLED true
set_interface_property reset_gameboy EXPORT_OF ""
set_interface_property reset_gameboy PORT_NAME_MAP ""
set_interface_property reset_gameboy CMSIS_SVD_VARIABLES ""
set_interface_property reset_gameboy SVD_ADDRESS_GROUP ""

add_interface_port reset_gameboy GB_rst reset Output 1



#
# connection point hps_slave
#
add_interface hps_slave avalon end
```

```
set_interface_property hps_slave addressUnits WORDS
set_interface_property hps_slave associatedClock clock
set_interface_property hps_slave associatedReset reset
set_interface_property hps_slave bitsPerSymbol 8
set_interface_property hps_slave burstOnBurstBoundariesOnly false
set_interface_property hps_slave burstcountUnits WORDS
set_interface_property hps_slave explicitAddressSpan 0
set_interface_property hps_slave holdTime 0
set_interface_property hps_slave linewrapBursts false
set_interface_property hps_slave maximumPendingReadTransactions 0
set_interface_property hps_slave maximumPendingWriteTransactions 0
set_interface_property hps_slave readLatency 0
set_interface_property hps_slave readWaitTime 1
set_interface_property hps_slave setupTime 0
set_interface_property hps_slave timingUnits Cycles
set_interface_property hps_slave writeWaitTime 0
set_interface_property hps_slave ENABLED true
set_interface_property hps_slave EXPORT_OF ""
set_interface_property hps_slave PORT_NAME_MAP ""
set_interface_property hps_slave CMSIS_SVD_VARIABLES ""
set_interface_property hps_slave SVD_ADDRESS_GROUP ""

add_interface_port hps_slave hps_address address Input 26
add_interface_port hps_slave hps_read read Input 1
add_interface_port hps_slave hps_write write Input 1
add_interface_port hps_slave hps_writedata writedata Input 8
```

```
       add_interface_port hps_slave hps_waitrequest waitrequest Output 1
       add_interface_port hps_slave hps_readdata readdata Output 8
       set_interface_assignment hps_slave embeddedsw.configuration.isFlash 0
       set_interface_assignment hps_slave
embeddedsw.configuration.isMemoryDevice 0
       set_interface_assignment hps_slave
embeddedsw.configuration.isNonVolatileStorage 0
       set_interface_assignment hps_slave
embeddedsw.configuration.isPrintableDevice 0



       #
       # connection point clock_gameboy
       #
       add_interface clock_gameboy clock start
       set_interface_property clock_gameboy associatedDirectClock ""
       set_interface_property clock_gameboy clockRate 0
       set_interface_property clock_gameboy clockRateKnown false
       set_interface_property clock_gameboy ENABLED true
       set_interface_property clock_gameboy EXPORT_OF ""
       set_interface_property clock_gameboy PORT_NAME_MAP ""
       set_interface_property clock_gameboy CMSIS_SVD_VARIABLES ""
       set_interface_property clock_gameboy SVD_ADDRESS_GROUP ""

       add_interface_port clock_gameboy SYNC_clk clk Output 1
```

```
#
# connection point LEDR
#
add_interface LEDR conduit end
set_interface_property LEDR associatedClock ""
set_interface_property LEDR associatedReset ""
set_interface_property LEDR ENABLED true
set_interface_property LEDR EXPORT_OF ""
set_interface_property LEDR PORT_NAME_MAP ""
set_interface_property LEDR CMSIS_SVD_VARIABLES ""
set_interface_property LEDR SVD_ADDRESS_GROUP ""

add_interface_port LEDR LEDR ledr Output 10
```

GameBoy_Top.sv

```
Unset
`timescale 1ns / 1ns
//////////////////////////////////////////////////////////////////////////////////
//////////
// The GameBoy Top Level
```

```
      ////////////////////////////////////////////////////////////////////
////////

      module GameBoy_Top
      (
          input logic clk,
          input logic rst,
          /* GameBoy Pixel Conduit */
          output logic PX_VALID,
          output logic [1:0] LD,
          /* GameBoy Joypad Conduit */
             input logic P10,
             input logic P11,
             input logic P12,
             input logic P13,
             output logic P14,
             output logic P15,
             /* GameBoy Cartridge Conduit */
             output logic [15:0] CART_ADDR,
             input logic [7:0] CART_DATA_in,
             output logic [7:0] CART_DATA_out,
             output logic CART_RD,
             output logic CART_WR

      );
```

```
/* Video SRAM */
logic [7:0] MD_in; // video sram data
logic [7:0] MD_out; // video sram data
logic [12:0] MA;
logic MWR; // high active
logic MCS; // high active
logic MOE; // high active
/* LCD */
logic CPG; // CONTROL
logic CP; // CLOCK
logic ST; // HORSYNC
logic CPL; // DATALCH
logic FR; // ALTSIGL
logic S; // VERTSYN

/* Work RAM/Cartridge */
logic CLK_GC; // Game Cartridge Clock
logic WR; // high active
logic RD; // high active
logic CS; // high active
logic [15:0] A;
logic [7:0] D_in; // data bus
logic [7:0] D_out; // data bus

/* The DMG-CPU */
```

```
      LR35902 DMG_CPU(.clk(clk), .rst(rst), .MD_in(MD_in), .MD_out(MD_out),
.MA(MA), .MWR(MWR), .MCS(MCS), .MOE(MOE), .LD(LD), .PX_VALID(PX_VALID),
                      .CPG(CPG), .CP(CP), .ST(ST), .CPL(CPL), .FR(FR), .S(S),
.P10(P10), .P11(P11), .P12(P12), .P13(P13), .P14(P14), .P15(P15),
                      .CLK_GC(CLK_GC), .WR(WR), .RD(RD), .CS(CS), .A(A),
.D_in(D_in),
                      .D_out(D_out));

      /* VRAM Connection */
      LH5264 VRAM(.D_out(MD_in), .D_in(MD_out), .CE1(MCS), .CE2(MWR), .A(MA),
.OE(MOE), .clk(~clk));

      /* WRAM Connection */
      logic [7:0] WRAM_Din, WRAM_Dout, WRAM_WR;
      LH5264 WRAM(.D_out(WRAM_Din), .D_in(WRAM_Dout), .CE1(WRAM_WR),
.CE2(A[14]), .A(A), .OE(A[14]), .clk(~clk));

      /* Cartridge */
      assign D_in = (A < 16'h8000 || (A >= 16'hA000 && A < 16'hC000)) ?
CART_DATA_in : WRAM_Din;
      assign CART_DATA_out = (A < 16'h8000 || (A >= 16'hA000 && A < 16'hC000))
? D_out : 0;
      assign CART_RD = RD && (A < 16'h8000 || (A >= 16'hA000 && A < 16'hC000));
      assign CART_WR = WR && (A < 16'h8000 || (A >= 16'hA000 && A < 16'hC000));
      assign CART_ADDR = (A < 16'h8000 || (A >= 16'hA000 && A < 16'hC000)) ? A
: 0;
```

```
	assign WRAM_Dout = !(A < 16'h8000 || (A >= 16'hA000 && A < 16'hC000)) ?
D_out : 8'hFF;
	assign WRAM_WR = WR && (A >= 16'hC000 && A < 16'hFE00);


	endmodule
```

GameBoy_VGA_hw.tcl

```
Unset
	# TCL File Generated by Component Editor 18.1
	# Tue Jun 18 23:56:27 EDT 2019
	# DO NOT MODIFY



	#
	# GameBoy_VGA "GameBoy_VGA" v1.0
	#  2019.06.18.23:56:27
	# The VGA output for GameBoy Display
	#


	#
	# request TCL package from ACDS 16.1
	#
	package require -exact qsys 16.1
```

```
#
# module GameBoy_VGA
#
set_module_property DESCRIPTION "The VGA output for GameBoy Display"
set_module_property NAME GameBoy_VGA
set_module_property VERSION 1.0
set_module_property INTERNAL false
set_module_property OPAQUE_ADDRESS_MAP true
set_module_property GROUP GameBoy
set_module_property AUTHOR ""
set_module_property DISPLAY_NAME GameBoy_VGA
set_module_property INSTANTIATE_IN_SYSTEM_MODULE true
set_module_property EDITABLE true
set_module_property REPORT_TO_TALKBACK false
set_module_property ALLOW_GREYBOX_GENERATION false
set_module_property REPORT_HIERARCHY false



#
# file sets
#
add_fileset QUARTUS_SYNTH QUARTUS_SYNTH "" ""
set_fileset_property QUARTUS_SYNTH TOP_LEVEL GameBoy_VGA
set_fileset_property QUARTUS_SYNTH ENABLE_RELATIVE_INCLUDE_PATHS false
```

```
	set_fileset_property QUARTUS_SYNTH ENABLE_FILE_OVERWRITE_MODE false
	add_fileset_file GameBoy_VGA.sv SYSTEM_VERILOG PATH GameBoy_VGA.sv
TOP_LEVEL_FILE



	#
	# parameters
	#



	#
	# display items
	#



	#
	# connection point clock
	#
	add_interface clock clock end
	set_interface_property clock clockRate 0
	set_interface_property clock ENABLED true
	set_interface_property clock EXPORT_OF ""
	set_interface_property clock PORT_NAME_MAP ""
	set_interface_property clock CMSIS_SVD_VARIABLES ""
	set_interface_property clock SVD_ADDRESS_GROUP ""
```

```
add_interface_port clock clk clk Input 1



#
# connection point reset
#
add_interface reset reset end
set_interface_property reset associatedClock clock
set_interface_property reset synchronousEdges DEASSERT
set_interface_property reset ENABLED true
set_interface_property reset EXPORT_OF ""
set_interface_property reset PORT_NAME_MAP ""
set_interface_property reset CMSIS_SVD_VARIABLES ""
set_interface_property reset SVD_ADDRESS_GROUP ""

add_interface_port reset reset reset Input 1



#
# connection point GameBoy_Pixel
#
add_interface GameBoy_Pixel conduit end
set_interface_property GameBoy_Pixel associatedClock GameBoy_clock
set_interface_property GameBoy_Pixel associatedReset ""
set_interface_property GameBoy_Pixel ENABLED true
set_interface_property GameBoy_Pixel EXPORT_OF ""
```

```
set_interface_property GameBoy_Pixel PORT_NAME_MAP ""
set_interface_property GameBoy_Pixel CMSIS_SVD_VARIABLES ""
set_interface_property GameBoy_Pixel SVD_ADDRESS_GROUP ""

add_interface_port GameBoy_Pixel LD ld Input 2
add_interface_port GameBoy_Pixel PX_VALID px_valid Input 1



#
# connection point VGA
#
add_interface VGA conduit end
set_interface_property VGA associatedClock clock
set_interface_property VGA associatedReset ""
set_interface_property VGA ENABLED true
set_interface_property VGA EXPORT_OF ""
set_interface_property VGA PORT_NAME_MAP ""
set_interface_property VGA CMSIS_SVD_VARIABLES ""
set_interface_property VGA SVD_ADDRESS_GROUP ""

add_interface_port VGA VGA_B vga_b Output 8
add_interface_port VGA VGA_BLANK_n vga_blank_n Output 1
add_interface_port VGA VGA_CLK vga_clk Output 1
add_interface_port VGA VGA_G vga_g Output 8
add_interface_port VGA VGA_HS vga_hs Output 1
add_interface_port VGA VGA_R vga_r Output 8
```

```
add_interface_port VGA VGA_SYNC_n vga_sync_n Output 1
add_interface_port VGA VGA_VS vga_vs Output 1



#
# connection point GameBoy_clock
#
add_interface GameBoy_clock clock end
set_interface_property GameBoy_clock clockRate 0
set_interface_property GameBoy_clock ENABLED true
set_interface_property GameBoy_clock EXPORT_OF ""
set_interface_property GameBoy_clock PORT_NAME_MAP ""
set_interface_property GameBoy_clock CMSIS_SVD_VARIABLES ""
set_interface_property GameBoy_clock SVD_ADDRESS_GROUP ""

add_interface_port GameBoy_clock GameBoy_clk clk Input 1



#
# connection point GameBoy_reset
#
add_interface GameBoy_reset reset end
set_interface_property GameBoy_reset associatedClock clock
set_interface_property GameBoy_reset synchronousEdges DEASSERT
set_interface_property GameBoy_reset ENABLED true
set_interface_property GameBoy_reset EXPORT_OF ""
```

```
set_interface_property GameBoy_reset PORT_NAME_MAP ""
set_interface_property GameBoy_reset CMSIS_SVD_VARIABLES ""
set_interface_property GameBoy_reset SVD_ADDRESS_GROUP ""

add_interface_port GameBoy_reset GameBoy_reset reset Input 1



#
# connection point avalon_slave
#
add_interface avalon_slave avalon end
set_interface_property avalon_slave addressUnits WORDS
set_interface_property avalon_slave associatedClock clock
set_interface_property avalon_slave associatedReset reset
set_interface_property avalon_slave bitsPerSymbol 8
set_interface_property avalon_slave burstOnBurstBoundariesOnly false
set_interface_property avalon_slave burstcountUnits WORDS
set_interface_property avalon_slave explicitAddressSpan 0
set_interface_property avalon_slave holdTime 0
set_interface_property avalon_slave linewrapBursts false
set_interface_property avalon_slave maximumPendingReadTransactions 0
set_interface_property avalon_slave maximumPendingWriteTransactions 0
set_interface_property avalon_slave readLatency 0
set_interface_property avalon_slave readWaitTime 1
set_interface_property avalon_slave setupTime 0
set_interface_property avalon_slave timingUnits Cycles
```

```
       set_interface_property avalon_slave writeWaitTime 0
       set_interface_property avalon_slave ENABLED true
       set_interface_property avalon_slave EXPORT_OF ""
       set_interface_property avalon_slave PORT_NAME_MAP ""
       set_interface_property avalon_slave CMSIS_SVD_VARIABLES ""
       set_interface_property avalon_slave SVD_ADDRESS_GROUP ""

       add_interface_port avalon_slave writedata writedata Input 8
       add_interface_port avalon_slave write write Input 1
       add_interface_port avalon_slave chipselect chipselect Input 1
       add_interface_port avalon_slave address address Input 21
       set_interface_assignment avalon_slave embeddedsw.configuration.isFlash 0
       set_interface_assignment avalon_slave
embeddedsw.configuration.isMemoryDevice 0
       set_interface_assignment avalon_slave
embeddedsw.configuration.isNonVolatileStorage 0
       set_interface_assignment avalon_slave
embeddedsw.configuration.isPrintableDevice 0



       #
       # connection point clock_vga
       #
       add_interface clock_vga clock end
       set_interface_property clock_vga clockRate 0
       set_interface_property clock_vga ENABLED true
```

```
set_interface_property clock_vga EXPORT_OF ""
set_interface_property clock_vga PORT_NAME_MAP ""
set_interface_property clock_vga CMSIS_SVD_VARIABLES ""
set_interface_property clock_vga SVD_ADDRESS_GROUP ""

add_interface_port clock_vga clk_vga clk Input 1
```

GameBoy_hw.tcl

```
Unset
# TCL File Generated by Component Editor 18.1
# Wed Jun 26 17:18:14 EDT 2019
# DO NOT MODIFY


#
# GameBoy "GameBoy" v1.0
#  2019.06.26.17:18:14
# The GameBoy
#


#
# request TCL package from ACDS 16.1
```

```
#
package require -exact qsys 16.1



#
# module GameBoy
#
set_module_property DESCRIPTION "The GameBoy"
set_module_property NAME GameBoy
set_module_property VERSION 1.0
set_module_property INTERNAL false
set_module_property OPAQUE_ADDRESS_MAP true
set_module_property GROUP GameBoy
set_module_property AUTHOR ""
set_module_property DISPLAY_NAME GameBoy
set_module_property INSTANTIATE_IN_SYSTEM_MODULE true
set_module_property EDITABLE true
set_module_property REPORT_TO_TALKBACK false
set_module_property ALLOW_GREYBOX_GENERATION false
set_module_property REPORT_HIERARCHY false



#
# file sets
#
add_fileset QUARTUS_SYNTH QUARTUS_SYNTH "" ""
```

```
       set_fileset_property QUARTUS_SYNTH TOP_LEVEL GameBoy_Top
       set_fileset_property QUARTUS_SYNTH ENABLE_RELATIVE_INCLUDE_PATHS false
       set_fileset_property QUARTUS_SYNTH ENABLE_FILE_OVERWRITE_MODE false
       add_fileset_file GameBoy_Top.sv SYSTEM_VERILOG PATH GameBoy_Top.sv
TOP_LEVEL_FILE


       #
       # parameters
       #



       #
       # display items
       #



       #
       # connection point clock
       #
       add_interface clock clock end
       set_interface_property clock clockRate 0
       set_interface_property clock ENABLED true
       set_interface_property clock EXPORT_OF ""
       set_interface_property clock PORT_NAME_MAP ""
       set_interface_property clock CMSIS_SVD_VARIABLES ""
```

```
set_interface_property clock SVD_ADDRESS_GROUP ""


add_interface_port clock clk clk Input 1



#
# connection point GameBoy_ROM
#
add_interface GameBoy_ROM conduit end
set_interface_property GameBoy_ROM associatedClock ""
set_interface_property GameBoy_ROM associatedReset ""
set_interface_property GameBoy_ROM ENABLED true
set_interface_property GameBoy_ROM EXPORT_OF ""
set_interface_property GameBoy_ROM PORT_NAME_MAP ""
set_interface_property GameBoy_ROM CMSIS_SVD_VARIABLES ""
set_interface_property GameBoy_ROM SVD_ADDRESS_GROUP ""

add_interface_port GameBoy_ROM CART_ADDR cart_addr Output 16
add_interface_port GameBoy_ROM CART_RD cart_rd Output 1
add_interface_port GameBoy_ROM CART_WR cart_wr Output 1
add_interface_port GameBoy_ROM CART_DATA_in cart_data_in Input 8
add_interface_port GameBoy_ROM CART_DATA_out cart_data_out Output 8



#
# connection point GameBoy_Pixel
```

```
#
add_interface GameBoy_Pixel conduit end
set_interface_property GameBoy_Pixel associatedClock clock
set_interface_property GameBoy_Pixel associatedReset ""
set_interface_property GameBoy_Pixel ENABLED true
set_interface_property GameBoy_Pixel EXPORT_OF ""
set_interface_property GameBoy_Pixel PORT_NAME_MAP ""
set_interface_property GameBoy_Pixel CMSIS_SVD_VARIABLES ""
set_interface_property GameBoy_Pixel SVD_ADDRESS_GROUP ""

add_interface_port GameBoy_Pixel LD ld Output 2
add_interface_port GameBoy_Pixel PX_VALID px_valid Output 1



#
# connection point GameBoy_Joypad
#
add_interface GameBoy_Joypad conduit end
set_interface_property GameBoy_Joypad associatedClock ""
set_interface_property GameBoy_Joypad associatedReset ""
set_interface_property GameBoy_Joypad ENABLED true
set_interface_property GameBoy_Joypad EXPORT_OF ""
set_interface_property GameBoy_Joypad PORT_NAME_MAP ""
set_interface_property GameBoy_Joypad CMSIS_SVD_VARIABLES ""
set_interface_property GameBoy_Joypad SVD_ADDRESS_GROUP ""
```

```
add_interface_port GameBoy_Joypad P10 p10 Input 1
add_interface_port GameBoy_Joypad P11 p11 Input 1
add_interface_port GameBoy_Joypad P12 p12 Input 1
add_interface_port GameBoy_Joypad P13 p13 Input 1
add_interface_port GameBoy_Joypad P14 p14 Output 1
add_interface_port GameBoy_Joypad P15 p15 Output 1



#
# connection point reset
#
add_interface reset reset end
set_interface_property reset associatedClock clock
set_interface_property reset synchronousEdges DEASSERT
set_interface_property reset ENABLED true
set_interface_property reset EXPORT_OF ""
set_interface_property reset PORT_NAME_MAP ""
set_interface_property reset CMSIS_SVD_VARIABLES ""
set_interface_property reset SVD_ADDRESS_GROUP ""

add_interface_port reset rst reset Input 1



#
# connection point GameBoy_Audio
#
```

```
add_interface GameBoy_Audio conduit end
set_interface_property GameBoy_Audio associatedClock ""
set_interface_property GameBoy_Audio associatedReset ""
set_interface_property GameBoy_Audio ENABLED true
set_interface_property GameBoy_Audio EXPORT_OF ""
set_interface_property GameBoy_Audio PORT_NAME_MAP ""
set_interface_property GameBoy_Audio CMSIS_SVD_VARIABLES ""
set_interface_property GameBoy_Audio SVD_ADDRESS_GROUP ""

add_interface_port GameBoy_Audio LOUT lout Output 16
add_interface_port GameBoy_Audio ROUT rout Output 16
```

LH5264.sv

```
Unset
`timescale 1ns / 1ns
////////////////////////////////////////////////////////////////////////////////
/*
    This is the functional block of LH5264 SRAM
    Size 8192 x 1 bytes (8bits)
*/
////////////////////////////////////////////////////////////////////////////////
```

```
module LH5264
(
    input logic [7:0] D_in,
    input logic [12:0] A,
    input logic CE1,
    input logic CE2,
    input logic clk,
    input logic OE,
    output logic [7:0] D_out
);

logic we;
assign we = CE1 && CE2;

logic [7:0] q;
assign D_out = OE ? q : 8'hFF;
Quartus_single_port_ram  #(.DATA_WIDTH(8), .ADDR_WIDTH(13),.DEPTH(2**13))
RAM_8K(.data(D_in), .addr(A), .clk(clk), .we(we), .q(q));

endmodule
```

LR35902.sv

```
Unset
       `timescale 1ns / 1ns
       ////////////////////////////////////////////////////////////////////////
////////
       /*
           This is the functional block of Sharp LR35902 AKA DMG-CPU
           Clock Frequency: 4194304(2^22) Hz
           Machine Cycle:   1048576(2^20) Hz
           Port naming based on Gameboy1-cpuboard.gif
       */
       ////////////////////////////////////////////////////////////////////////
////////
       `define NO_BOOT 0

       // All tristate signals are redesigned to be separate in/out
       module LR35902
       (
           input logic clk,   // XTAL
           input logic rst, // Power On Reset
           /* Video SRAM */
           input logic [7:0] MD_in,   // video sram data
           output logic [7:0] MD_out, // video sram data
           output logic [12:0] MA,
           output logic MWR, // high active
           output logic MCS, // high active
           output logic MOE, // high active
           /* LCD */
```

```
        output logic [1:0] LD, // PPU DATA 1-0
        output logic PX_VALID,
        output logic CPG, // CONTROL
        output logic CP, // CLOCK
        output logic ST, // HORSYNC
        output logic CPL, // DATALCH
        output logic FR, // ALTSIGL
        output logic S, // VERTSYN
        /* Joy Pads */
        input logic P10,
        input logic P11,
        input logic P12,
        input logic P13,
        output logic P14,
        output logic P15,
        /* Work RAM/Cartridge */
        output logic CLK_GC, // Game Cartridge Clock
        output logic WR, // high active
        output logic RD, // high active
        output logic CS, // high active
        output logic [15:0] A,
        input logic [7:0] D_in, // work ram/cartridge data bus
        output logic [7:0] D_out // work ram/cartridge data bus
    );

    /* GB-Z80 CPU */
```

```
	logic [7:0] GB_Z80_D_in;
	logic [7:0] GB_Z80_D_out;
	logic [15:0] GB_Z80_ADDR;
	logic GB_Z80_RD, GB_Z80_WR;
	logic GB_Z80_HALT;
	logic [4:0] GB_Z80_INTQ;

	GB_Z80_SINGLE GB_Z80_CPU(.clk(clk), .rst(rst), .ADDR(GB_Z80_ADDR),
.DATA_in(GB_Z80_D_in), .DATA_out(GB_Z80_D_out),
                                .RD(GB_Z80_RD), .WR(GB_Z80_WR),
.CPU_HALT(GB_Z80_HALT), .INTQ(GB_Z80_INTQ));

	/* Begin Peripherals for GB-Z80 */

	/* ROM Region $0x0000 to 0x7FFF*/

	// The Boot Rom is mapped from $0x0000 to $0x00FF if $0xFF50 is not
written before
	logic brom_en, brom_en_next;
	logic [7:0] DATA_BROM;
	brom boot_rom(.addr(GB_Z80_ADDR[7:0]), .data(DATA_BROM), .clk(~clk));

	/* Video RAM Region $0x8000 to $0x9FFF */
```

```
	/* Cartridge RAM Region $0xA000 to $0xBFFF */



	/* Work RAM Region $0xC000 to $0xDFFF */ /* Echo RAM Region $0xE000 to
$0xFDFF */



	/* OAM Region $0xFE00 to $0xFE9F */ /* Reserved Unusable Region $0xFEA0
to $0xFEFF */
	logic OAM_WR;
	logic [7:0] DATA_OAM_in;
	logic [7:0] DATA_OAM_out;
	logic [7:0] OAM_ADDR;
	Quartus_single_port_ram #(.DATA_WIDTH(8), .ADDR_WIDTH(8),.DEPTH(160))
OAM(.q(DATA_OAM_in), .addr(OAM_ADDR), .clk(~clk), .we(OAM_WR),
.data(DATA_OAM_out));



	/* Hardware I0 Register Region $0xFF00 to $0xFF4B */
	logic [7:0] FF00, FF00_NEXT;
	assign P15 = FF00[5];
	assign P14 = FF00[4];
	logic [7:0] FF0F, FF0F_NEXT; // Interrupt Flag



	// Timer
```

```
	logic MMIO_TIMER_WR, MMIO_TIMER_RD;
	logic [7:0] MMIO_TIMER_DATA_in, MMIO_TIMER_DATA_out;
	logic IRQ_TIMER;
	TIMER GB_TIMER (.clk(clk), .rst(rst), .ADDR(GB_Z80_ADDR),
.WR(MMIO_TIMER_WR), .RD(MMIO_TIMER_RD), .MMIO_DATA_out(MMIO_TIMER_DATA_out),
                    .MMIO_DATA_in(MMIO_TIMER_DATA_in),
.IRQ_TIMER(IRQ_TIMER));

	// DMA Controller
	logic [7:0] FF46;
	logic [7:0] DMA_ADDR, DMA_ADDR_NEXT;
	logic [7:0] DMA_SETUP_ADDR, DMA_SETUP_ADDR_NEXT;
	logic [2:0] DMA_SETUP_CNT, DMA_SETUP_CNT_NEXT;
	logic DMA_SETUP, DMA_SETUP_NEXT;
	typedef enum {DMA_IDLE, DMA_GO} DMA_STATE_t;
	DMA_STATE_t DMA_STATE, DMA_STATE_NEXT;
	logic [9:0] DMA_CNT, DMA_CNT_NEXT;

	/* Reserved Unusable Region $0xFF4C to $0xFF7F */



	/* High RAM Region $0xFF80 to $0xFFFE */

	/* Interrupt Enable Register $0xFFFF */
	logic [7:0] FFFF, FFFF_NEXT;
```

```
	assign GB_Z80_INTQ = (DMA_STATE == DMA_GO) ? 0 : FFFF_NEXT[4:0] &
FF0F_NEXT[4:0];

	logic HRAM_WR;
	logic [7:0] DATA_HRAM_in;
	logic [7:0] DATA_HRAM_out;
	Quartus_single_port_ram #(.DATA_WIDTH(8), .ADDR_WIDTH(7), .DEPTH(128))
HRAM(.q(DATA_HRAM_in), .addr(GB_Z80_ADDR[6:0]), .clk(~clk), .we(HRAM_WR),
.data(DATA_HRAM_out));

	/* PPU */
	logic MMIO_PPU_WR, MMIO_PPU_RD;
	logic [7:0] MMIO_PPU_DATA_in, MMIO_PPU_DATA_out;
	logic IRQ_V_BLANK, IRQ_LCDC;
	logic [1:0] PPU_MODE;
	logic PPU_RD;
	logic [7:0] PPU_DATA_in;
	logic [15:0] PPU_ADDR;
	PPU3 GB_PPU(.clk(clk), .rst(rst), .ADDR(GB_Z80_ADDR), .WR(MMIO_PPU_WR),
.RD(MMIO_PPU_RD), .MMIO_DATA_out(MMIO_PPU_DATA_out),
		.MMIO_DATA_in(MMIO_PPU_DATA_in),
.IRQ_PPU_V_BLANK(IRQ_V_BLANK), .IRQ_LCDC(IRQ_LCDC), .PPU_MODE(PPU_MODE),
		.PPU_ADDR(PPU_ADDR), .PPU_RD(PPU_RD),
.PPU_DATA_in(PPU_DATA_in), .PX_OUT(LD), .PX_valid(PX_VALID));
```

```
/* Memory Management Unit */
// Map the CPU Memory Address to correct Peripheral Address
always_ff @(posedge clk)
begin
    if (rst)
    begin
        brom_en <= `NO_BOOT ? 0 : 1;
        FF00 <= 8'hCF;
        FF0F <= 8'hE0;
        FFFF <= 8'h00;

        DMA_ADDR <= 0;
        DMA_STATE <= DMA_IDLE;
        DMA_CNT <= 0;
        DMA_SETUP_CNT <= 0;
        DMA_SETUP_ADDR <= 0;
        DMA_SETUP <= 0;
    end
    else
    begin
        brom_en <= brom_en_next;
        FF00 <= FF00_NEXT;
        FF0F <= FF0F_NEXT;
        FFFF <= FFFF_NEXT;

        DMA_STATE <= DMA_STATE_NEXT;
```

```
            DMA_CNT <= DMA_CNT_NEXT;
            DMA_ADDR <= DMA_ADDR_NEXT;
            DMA_SETUP_CNT <= DMA_SETUP_CNT_NEXT;
            DMA_SETUP_ADDR <= DMA_SETUP_ADDR_NEXT;
            DMA_SETUP <= DMA_SETUP_NEXT;
        end
    end



    always_comb
    begin
        GB_Z80_D_in = 8'hFF;
        MWR = 0; MOE = 0; MCS = 0;
        MD_out = 0;
        MA = 0;
        A = 0;
        D_out = 0;
        WR = 0; RD = 0; CS = 0;
        HRAM_WR = 0; OAM_WR = 0;
        DATA_HRAM_out = 8'hFF;
        DATA_OAM_out = 8'hFF;
        brom_en_next = brom_en;
        OAM_ADDR = GB_Z80_ADDR[7:0];
        MMIO_PPU_WR = 0; MMIO_PPU_RD = 0; MMIO_PPU_DATA_out = 8'hFF;
        PPU_DATA_in = 8'hFF;
        MMIO_TIMER_WR = 0; MMIO_TIMER_RD = 0; MMIO_TIMER_DATA_out = 8'hFF;
```

```
            /* Interrupt Register */
            FF00_NEXT = FF00;
            FF0F_NEXT = FF0F;
            if (IRQ_V_BLANK) FF0F_NEXT[0] = 1;
            if (IRQ_LCDC) FF0F_NEXT[1] = 1;
            if (IRQ_TIMER) FF0F_NEXT[2] = 1;
            FFFF_NEXT = FFFF;

            /* Memory Access Handlers */
            if (GB_Z80_ADDR == 16'hFF50 && GB_Z80_WR) brom_en_next = 0; //
Capture Write to FF50 which disables Boot Rom

            /* DMA */
            DMA_STATE_NEXT = DMA_STATE;
            DMA_CNT_NEXT = DMA_CNT;
            DMA_ADDR_NEXT = DMA_ADDR;
            DMA_SETUP_CNT_NEXT = DMA_SETUP_CNT;
            DMA_SETUP_ADDR_NEXT = DMA_SETUP_ADDR;
            DMA_SETUP_NEXT = DMA_SETUP;

            if (GB_Z80_ADDR == 16'hFF46 && GB_Z80_WR) // Capture DMA write
            begin
                DMA_SETUP_NEXT = 1;
                DMA_SETUP_CNT_NEXT = 1;
                DMA_SETUP_ADDR_NEXT = GB_Z80_D_out;
```

```
            end

            unique case (DMA_STATE)
                DMA_IDLE: DMA_CNT_NEXT = 0;
                DMA_GO:
                begin
                    DMA_CNT_NEXT = DMA_CNT + 1;
                    OAM_WR = 1;
                    OAM_ADDR = DMA_CNT >> 2;
                    if (({DMA_ADDR, 8'h00} + (DMA_CNT >> 2)) >= 16'h8000 &&
({DMA_ADDR, 8'h00} + (DMA_CNT >> 2)) <= 16'h9FFF) // Copy from VRAM
                    begin
                        MA = {DMA_ADDR, 8'h00} + (DMA_CNT >> 2);
                        MCS = 1; MOE = 1;
                        DATA_OAM_out = MD_in;

                        if (GB_Z80_ADDR <= 16'h7FFF || (GB_Z80_ADDR >= 16'hA000
&& GB_Z80_ADDR < 16'hFE00)) // Allow CPU to access WRAM/CART Bus at this time
                        begin
                            A = GB_Z80_ADDR;
                            GB_Z80_D_in = D_in;
                            D_out = GB_Z80_D_out;
                            CS = 1; RD = GB_Z80_RD; WR = GB_Z80_WR;
                        end
                    end
                    else // Copy from ROM or Work RAM
```

```
                    begin
                        A = {DMA_ADDR, 8'h00} + (DMA_CNT >> 2);
                        CS = 1; RD = 1;
                        DATA_OAM_out = D_in;

                        if (PPU_MODE == 2'b11 && PPU_ADDR >= 16'h8000 && PPU_ADDR
<= 16'h9FFF) // Allow GPU to Access VRAM at this time
                        begin
                            MA = PPU_ADDR;
                            PPU_DATA_in = MD_in;
                            MCS = 1; MOE = PPU_RD; MWR = 0;
                        end
                    end
                    if (DMA_CNT == ((160 << 2) - 1))
                    begin
                        DMA_CNT_NEXT = 0;
                        DMA_STATE_NEXT = DMA_IDLE;
                    end
                end
            endcase

            if (DMA_SETUP)
            begin
                DMA_SETUP_CNT_NEXT = DMA_SETUP_CNT + 1;
                if (DMA_SETUP_CNT == 3'b100)
                begin
```

```
                DMA_SETUP_NEXT = 0;
                DMA_ADDR_NEXT = DMA_SETUP_ADDR;
                DMA_CNT_NEXT = 0;
                DMA_STATE_NEXT = DMA_GO;
                DMA_SETUP_CNT_NEXT = 0;
            end
        end

        /* ADDR MUX */

        if (DMA_STATE == DMA_GO)
        begin
            if (GB_Z80_ADDR >= 16'hFF80 && GB_Z80_ADDR < 16'hFFFF) // only
high ram acess is allowed
            begin
                GB_Z80_D_in = DATA_HRAM_in;
                DATA_HRAM_out = GB_Z80_D_out;
                HRAM_WR = GB_Z80_WR;
            end
        end

        if (DMA_STATE != DMA_GO) // DMA has higher priority than any of other
memory access
        begin
            if (GB_Z80_ADDR >= 16'h0000 && GB_Z80_ADDR <= 16'h00FF)
            begin
```

```
                A = brom_en ? 0 : GB_Z80_ADDR;
                GB_Z80_D_in = brom_en ? DATA_BROM : D_in;
                D_out = brom_en ? 0 : GB_Z80_D_out;
                CS = brom_en ? 0 : 1;
                RD = brom_en ? 0 : GB_Z80_RD;
                WR = brom_en ? 0 : GB_Z80_WR;
            end
            else if (GB_Z80_ADDR >= 16'h0100 && GB_Z80_ADDR <= 16'h7FFF)
            begin
                A = GB_Z80_ADDR;
                GB_Z80_D_in = D_in;
                D_out = GB_Z80_D_out;
                CS = 1; RD = GB_Z80_RD; WR = GB_Z80_WR;
            end
            else if (GB_Z80_ADDR >= 16'h8000 && GB_Z80_ADDR <= 16'h9FFF) //
VRAM
            begin
                if (PPU_MODE != 2'b11)
                begin
                    MA = GB_Z80_ADDR;
                    GB_Z80_D_in = MD_in;
                    MD_out = GB_Z80_D_out;
                    MCS = 1; MOE = GB_Z80_RD; MWR = GB_Z80_WR;
                end
                else GB_Z80_D_in = 16'hFF;
            end
```

```
                else if (GB_Z80_ADDR >= 16'hA000 &&  GB_Z80_ADDR <= 16'hBFFF) //
RAM for MBC
                begin
                    A = GB_Z80_ADDR;
                    GB_Z80_D_in = D_in;
                    D_out = GB_Z80_D_out;
                    CS = 1; RD = GB_Z80_RD; WR = GB_Z80_WR;
                end

                else if (GB_Z80_ADDR >= 16'hC000 && GB_Z80_ADDR <= 16'hFDFF)  //
WRAM with its echo
                begin
                   A = GB_Z80_ADDR;
                   GB_Z80_D_in = D_in;
                   D_out = GB_Z80_D_out;
                   CS = 1; RD = GB_Z80_RD; WR = GB_Z80_WR;
                end
                else if (GB_Z80_ADDR >= 16'hFE00 && GB_Z80_ADDR<= 16'hFEFF)// OAM
                begin
                    if (!PPU_MODE[1])
                    begin
                        GB_Z80_D_in = GB_Z80_ADDR < 16'hFEA0 ? DATA_OAM_in :
8'hFF;
                        DATA_OAM_out = GB_Z80_ADDR < 16'hFEA0 ? GB_Z80_D_out :
8'hFF;
                        OAM_WR = GB_Z80_ADDR < 16'hFEA0 ? GB_Z80_WR : 0;
```

```
                    end
                    else GB_Z80_D_in = 16'hFF;
                end
                else if (GB_Z80_ADDR == 16'hFF00) // JoyPad
                begin
                    GB_Z80_D_in = {2'b11, FF00[5:4], P13, P12, P11, P10};
                    if (GB_Z80_WR) FF00_NEXT = GB_Z80_D_out & 8'h30;
                end
                else if (GB_Z80_ADDR == 16'hFF01 || GB_Z80_ADDR == 16'hFF02); //
Serial
                else if (GB_Z80_ADDR == 16'hFF03) GB_Z80_D_in = 8'hFF; //
Undocumented
                else if (GB_Z80_ADDR >= 16'hFF04 && GB_Z80_ADDR <= 16'hFF07) //
Timer
                    begin
                    MMIO_TIMER_WR = GB_Z80_WR;
                    MMIO_TIMER_RD = GB_Z80_RD;
                    GB_Z80_D_in = MMIO_TIMER_DATA_in;
                    MMIO_TIMER_DATA_out = GB_Z80_D_out;
                end
                else if (GB_Z80_ADDR >= 16'hFF08 && GB_Z80_ADDR <= 16'hFF0E)
GB_Z80_D_in = 8'hFF; // Undocumented
                else if (GB_Z80_ADDR == 16'hFF0F) //Interrupt Flag
                begin
                    if (GB_Z80_RD) GB_Z80_D_in = {3'b111, FF0F[4:0]};
                    if (GB_Z80_WR) FF0F_NEXT = GB_Z80_D_out;
```

```
                end
                else if (GB_Z80_ADDR >= 16'hFF10 && GB_Z80_ADDR <= 16'hFF3F); //
Sound
                else if (GB_Z80_ADDR >= 16'hFF40 && GB_Z80_ADDR <= 16'hFF4B)
//PPU
                begin
                    MMIO_PPU_WR = GB_Z80_WR;
                    MMIO_PPU_RD = GB_Z80_RD;
                    GB_Z80_D_in = MMIO_PPU_DATA_in;
                    MMIO_PPU_DATA_out = GB_Z80_D_out;
                end
               else if (GB_Z80_ADDR >= 16'hFF4C && GB_Z80_ADDR <= 16'hFF7F) begin
                      GB_Z80_D_in = 8'hFF; // Unusable
               end
                else if (GB_Z80_ADDR >= 16'hFF80 && GB_Z80_ADDR < 16'hFFFF) //
High Ram
                begin
                    GB_Z80_D_in = DATA_HRAM_in;
                    DATA_HRAM_out = GB_Z80_D_out;
                    HRAM_WR = GB_Z80_WR;
                end
                else if (GB_Z80_ADDR == 16'hFFFF)
                begin
                    if (GB_Z80_RD) GB_Z80_D_in = FFFF;
                    if (GB_Z80_WR) FFFF_NEXT = GB_Z80_D_out;
                end
```

```
                else GB_Z80_D_in = 8'hFF ;

                if (PPU_MODE == 2'b11 && PPU_ADDR >= 16'h8000 && PPU_ADDR <=
16'h9FFF)
                begin
                    MA = PPU_ADDR;
                    PPU_DATA_in = MD_in;
                    MCS = 1; MOE = PPU_RD; MWR = 0;
                end

                if (PPU_MODE[1] && PPU_ADDR >= 16'hFE00 && PPU_ADDR < 16'hFEA0)
                begin
                    OAM_WR = 0; OAM_ADDR = PPU_ADDR;
                    PPU_DATA_in = DATA_OAM_in;
                end
            end

            if (GB_Z80_ADDR == 16'hFF46 && GB_Z80_RD) // DMA Register can be read
anytime
            begin
                    GB_Z80_D_in = DMA_SETUP_ADDR;
            end

        end

        endmodule
```

Makefile

```
Unset
	SYSTEM = soc_system


	TCL = $(SYSTEM).tcl

	QSYS = $(SYSTEM).qsys

	SOPCINFO = $(SYSTEM).sopcinfo

	QIP = $(SYSTEM)/synthesis/$(SYSTEM).qip

	HPS_PIN_TCL =
$(SYSTEM)/synthesis/submodules/hps_sdram_p0_pin_assignments.tcl

	HPS_PIN_MAP = hps_sdram_p0_all_pins.txt

	QPF = $(SYSTEM).qpf

	QSF = $(SYSTEM).qsf

	SDC = $(SYSTEM).sdc


	BOARD_INFO = $(SYSTEM)_board_info.xml

	DTS = $(SYSTEM).dts

	DTB = $(SYSTEM).dtb


	SOF = output_files/$(SYSTEM).sof

	RBF = output_files/$(SYSTEM).rbf
```

```
SOFTWARE_DIR = software


BSP_DIR = $(SOFTWARE_DIR)/spl_bsp

BSP_SETTINGS = $(BSP_DIR)/settings.bsp

PRELOADER_SETTINGS_DIR = hps_isw_handoff/soc_system_hps_0

PRELOADER_MAKEFILE = $(BSP_DIR)/Makefile

PRELOADER_MKPIMAGE = $(BSP_DIR)/preloader-mkpimage.bin


UBOOT_IMAGE = $(BSP_DIR)/uboot-socfpga/u-boot.img


KERNEL_REPO = https://github.com/altera-opensource/linux-socfpga.git

KERNEL_BRANCH = socfpga-4.19

DEFAULT_CONFIG = socfpga_defconfig

CROSS = env CROSS_COMPILE=arm-altera-eabi- ARCH=arm


KERNEL_DIR = $(SOFTWARE_DIR)/linux-socfpga

KERNEL_CONFIG = $(KERNEL_DIR)/.config

ZIMAGE = $(KERNEL_DIR)/arch/arm/boot/zImage


TARFILES = Makefile \

       $(TCL) \

       $(QSYS) \

       $(SYSTEM)_top.sv \

       $(BOARD_INFO) \

       ip/intr_capturer/intr_capturer.v \

       ip/intr_capturer/intr_capturer_hw.tcl \
```

```
		game_boy.sv


	TARFILE = lab3-hw.tar.gz


	# project
	#
	# Run the topmost tcl script to generate the initial project files


	.PHONY : project
	project : $(QPF) $(QSF) $(SDC)


	$(QPF) $(QSF) $(SDC) : $(TCL)
		quartus_sh -t $(TCL)


	# qsys
	#
	# From the .qsys file, generate the .sopcinfo, .qip, and directory
	# (named according to the system) with all the Verilog files, etc.


	.PHONY : qsys
	qsys : $(SOPCINFO)


	$(SOPCINFO) $(QIP) $(HPS_PIN_TCL) $(SYSTEM)/ $(PRELOADER_SETTINGS_DIR) :
$(QSYS)
		rm -rf $(SOPCINFO) $(SYSTEM)/
		qsys-generate $(QSYS) --synthesis=VERILOG
```

```
# quartus
#
# Run Quartus on the Qsys-generated files
#
#    Build netlist
# quartus_map soc_system
#
#    Use netlist information to determine HPS stuff
# quartus_sta -t hps_sdram_p0_pin_assignments.tcl soc_system
#
#    Do the rest
#   FIXME: this is wasteful.  Really want not to repeat the "map" step
# quartus_sh --flow compile
#
# quartus_fit
# quartus_asm
# quartus_sta

.PHONY : quartus
quartus : $(SOF)

$(SOF) $(HPS_PIN_MAP) : $(QIP) $(QPF) $(QSF) $(HPS_PIN_TCL)
	quartus_map $(SYSTEM)
	quartus_sta -t $(HPS_PIN_TCL) $(SYSTEM)
	quartus_fit $(SYSTEM)
```

```
	quartus_asm $(SYSTEM)
#	quartus_sh --flow compile $(QPF)

# rbf
#
# Convert the .sof file (for programming through the USB blaster)
# to an .rbf file to be placed on an SD card and written by u-boot
.PHONY : rbf
rbf : $(RBF)

$(RBF) : $(SOF)
	quartus_cpf -c $(SOF) $(RBF)

# dtb
#
# Use the .sopcinfo file to generate a device tree blob file
# with information about the memory map of the peripherals
.PHONY : dtb
dtb : $(DTB)

$(DTB) : $(DTS)
	dtc -I dts -O dtb -o $(DTB) $(DTS)

$(DTS) : $(SOPCINFO) $(BOARD_INFO)
	sopc2dts --input $(SOPCINFO) \
		--output $(DTS) \
```

```
                --type dts \
                --board $(BOARD_INFO) \
                --clocks


        # preloader
        #
        # Builds the SPL's preloader-mkpimage.bin image file, which should
        # be written to the "magic" 3rd parition on the SD card
        # in software/spl_bsp
        #
        # Requires the embedded_command_shell.sh script to have run so the
compiler,
        # etc is available
        #
        .PHONY : preloader
        preloader : $(PRELOADER_MKPIMAGE)


        $(PRELOADER_MKPIMAGE) : $(PRELOADER_MAKEFILE) $(BSP_SETTINGS)
              $(MAKE) -C $(BSP_DIR)


        $(BSP_SETTINGS) $(PRELOADER_MAKEFILE) : $(PRELOADER_SETTINGS_DIR)
              mkdir -p $(BSP_DIR)
              bsp-create-settings \
                --type spl \
                --bsp-dir $(BSP_DIR) \
                --settings $(BSP_SETTINGS) \
```

```
        --preloader-settings-dir $(PRELOADER_SETTINGS_DIR) \
        --set spl.boot.FAT_SUPPORT 1


# uboot
#
# Build the bootloader

.PHONY : uboot
uboot : $(UBOOT_IMAGE)

$(UBOOT_IMAGE) : $(PRELOADER_MAKEFILE) $(BSP_SETTINGS)
	$(MAKE) -C $(BSP_DIR) uboot

# kernel-download
#
# Clone the Linux kernel repository
#
# kernel-config
#
#   Set up the kernel configuration
#
# zimage
#
#   Compile the kernel

.PHONY : download-kernel config-kernel zimage
```

```
kernel-download : $(KERNEL_DIR)

kernel-config : $(KERNEL_CONFIG)

zimage : $(ZIMAGE)


$(KERNEL_DIR) :

	mkdir -p $(KERNEL_DIR)

	git clone --branch $(KERNEL_BRANCH) $(KERNEL_REPO) $(KERNEL_DIR)


# Configure the kernel.  Start from a provided default,

#

# Turn off version checking (makes it easier to compile kernel

# modules and not have them complain about version)

#

# Turn on large file (+2TB) support, which the ext4 filesystem

# requires by default (it will not be able to mount the root

# filesystem read/write otherwise)

$(KERNEL_CONFIG) : $(KERNEL_DIR)

	$(CROSS) $(MAKE) -C $(KERNEL_DIR) $(DEFAULT_CONFIG)

	$(KERNEL_DIR)/scripts/config --file $(KERNEL_CONFIG) \

	  --disable CONFIG_LOCALVERSION_AUTO \

	  --enable CONFIG_LBDAF \

	  --disable CONFIG_XFS_FS \

	  --disable CONFIG_GFS2_FS \

	  --disable CONFIG_TEST_KMOD


# Compile the kernel
```

```
	$(ZIMAGE) : $(KERNEL_CONFIG)
		$(CROSS) $(MAKE) -C $(KERNEL_DIR) LOCALVERSION= zImage

	# tar
	#
	# Build soc_system.tar.gz

	.phony : tar
	tar : $(TARFILE)

	$(TARFILE) : $(TARFILES)
		tar zcfC $(TARFILE) .. $(TARFILES:%=lab3-hw/%)

	# clean
	#
	# Remove all generated files

	.PHONY : clean quartus-clean qsys-clean project-clean
	clean : quartus-clean qsys-clean project-clean dtb-clean preloader-clean
\
		uboot-clean

	project-clean :
		rm -rf $(QPF) $(QSF) $(SDC)
```

```
	qsys-clean :
		rm -rf $(SOPCINFO) $(QIP) $(SYSTEM)/ .qsys_edit \
		hps_isw_handoff/ hps_sdram_p0_summary.csv


	quartus-clean :
		rm -rf  $(SOF) output_files db incremental_db $(SYSTEM).qdf \
		c5_pin_model_dump.txt $(HPS_PIN_MAP)


	dtb-clean :
		rm -rf $(DTS) $(DTB)


	preloader-clean :
		rm -rf $(BSP_DIR)


	uboot-clean :
		rm -rf $(BSP_DIR)/uboot-socfpga


	kernel-clean :
		rm -rf $(KERNEL_DIR)


	config-clean :
		rm -rf $(KERNEL_CONFIG)
```

PPU3.sv

```
Unset
	`timescale 1ns / 1ns
```

```
//`include "PPU.vh"
`define NO_BOOT 0

module PPU3
(
    input logic clk,
    input logic rst,

    input logic [15:0] ADDR,
    input logic WR,
    input logic RD,
    input logic [7:0] MMIO_DATA_out,
    output logic [7:0] MMIO_DATA_in,

    output logic IRQ_PPU_V_BLANK,
    output logic IRQ_LCDC,

    output logic [1:0] PPU_MODE,

    output logic PPU_RD,
    output logic [15:0] PPU_ADDR,
    input logic [7:0] PPU_DATA_in,

    output logic [1:0] PX_OUT,
    output logic PX_valid
```

```
        );

        logic [7:0] LCDC, STAT, SCX, SCY, LYC, DMA, BGP, OBP0, OBP1, WX, WY; //
Register alias

        logic [7:0] FF40, FF40_NEXT;
        assign LCDC = FF40;

        logic [7:0] FF41, FF41_NEXT;
        assign STAT = FF41;

        logic [7:0] FF42, FF42_NEXT;
        assign SCY = FF42;

        logic [7:0] FF43, FF43_NEXT;
        assign SCX = FF43;

        logic [7:0] FF44;

        logic [7:0] FF45, FF45_NEXT;
        assign LYC = FF45;

        logic [7:0] FF46, FF46_NEXT;
        assign DMA = FF46;

        logic [7:0] FF47, FF47_NEXT;
```

```
        assign BGP = FF47;


        logic [7:0] FF48, FF48_NEXT;
        assign OBP0 = {FF48[7:2], 2'b00}; // Last 2 bits are not used


        logic [7:0] FF49, FF49_NEXT;
        assign OBP1 = {FF49[7:2], 2'b00};


        logic [7:0] FF4A, FF4A_NEXT;
        assign WY = FF4A;


        logic [7:0] FF4B, FF4B_NEXT;
        assign WX = FF4B;


        typedef enum {OAM_SEARCH, RENDER, H_BLANK, V_BLANK} PPU_STATE_t;


        PPU_STATE_t PPU_STATE, PPU_STATE_NEXT;


        // Current Coordinates
        logic [7:0] LX, LX_NEXT; // LX starts from 0, LCD starts from LX + SCX &
7
        logic [7:0] LY, LY_NEXT;
        assign FF44 = LY;



        // OAM Machine
```

```
        logic OAM_SEARCH_GO;
        logic [15:0] OAM_SEARCH_PPU_ADDR;

        // BGWD Machine
        logic BGWD_RENDER_GO;
        logic SHIFT_REG_GO;

        // Current Rendering Tile Map Pattern Number
        logic [7:0] BG_MAP;
        logic [7:0] WD_MAP;
        logic [7:0] SP_MAP;

        // PPU Running Counter for every 60Hz refresh
        shortint unsigned PPU_CNT, PPU_CNT_NEXT;
        logic [2:0] SCX_CNT, SCX_CNT_NEXT;

        //assign IRQ_PPU_V_BLANK = (LY == 144 && PPU_CNT == 0);

        // Sprite Logic
        logic isSpriteOnLine;
       assign isSpriteOnLine = (((PPU_DATA_in + (LCDC[2] << 3)) > (LY + 8)) &&
(PPU_DATA_in <= (LY + 16)));
       logic [3:0] sp_table_cnt; //sp_table_cnt_next;
       logic [5:0] sp_name_table [0:9]; //logic [5:0] sp_name_table_next [0:9];
       logic [7:0] sp_name_table_x [0:9];
```

```
genvar sp_n_gi;
generate
for (sp_n_gi = 0; sp_n_gi < 10; sp_n_gi++)
begin : sp_n_gen
    assign sp_name_table_x[sp_n_gi] = {sp_name_table[sp_n_gi], 2'b00};
end
endgenerate

logic [7:0] sp_y_table [0:9];    //logic [7:0] sp_y_table_next [0:9];
logic [7:0] sp_x_table [0:9];    //logic [7:0] sp_x_table_next [0:9];
logic sp_found; //sp_found_next; // Search Result
logic isHitSP; // is there a sprite to fetch on current X?
logic [3:0] sp_to_fetch;
logic [9:0] sp_not_used, sp_not_used_next; // which sprite has been used
logic SP_RENDER_GO;
//logic [15:0] SPRITE_PPU_ADDR;
logic [9:0] SP_SHIFT_REG_LOAD;
logic [8:0] SP_TILE_DATA0, SP_TILE_DATA1;
logic [1:0] SP_PX_MAP [9:0];
logic [3:0] SP_NEXT_SLOT, SP_NEXT_SLOT_NEXT;
logic [2:0] SP_CNT;
logic [7:0] SP_FLAG;
logic [1:0] SP_PRIPN [0:9];
logic [1:0] SP_PRIPN_NEXT [0:9];
```

```
      PPU_SHIFT_REG SP_SHIFT_REG9(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[9]),
.q(SP_PX_MAP[9]));
      PPU_SHIFT_REG SP_SHIFT_REG8(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[8]),
.q(SP_PX_MAP[8]));
      PPU_SHIFT_REG SP_SHIFT_REG7(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[7]),
.q(SP_PX_MAP[7]));
      PPU_SHIFT_REG SP_SHIFT_REG6(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[6]),
.q(SP_PX_MAP[6]));
      PPU_SHIFT_REG SP_SHIFT_REG5(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[5]),
.q(SP_PX_MAP[5]));
      PPU_SHIFT_REG SP_SHIFT_REG4(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[4]),
.q(SP_PX_MAP[4]));
      PPU_SHIFT_REG SP_SHIFT_REG3(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[3]),
.q(SP_PX_MAP[3]));
      PPU_SHIFT_REG SP_SHIFT_REG2(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[2]),
.q(SP_PX_MAP[2]));
```

```
	PPU_SHIFT_REG SP_SHIFT_REG1(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[1]),
.q(SP_PX_MAP[1]));
	PPU_SHIFT_REG SP_SHIFT_REG0(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[0]),
.q(SP_PX_MAP[0]));

	// Fetch Logic
	localparam OAM_BASE = 16'hFE00;
	logic [15:0] VRAM_DATA_BASE;
	assign VRAM_DATA_BASE = LCDC[4] ? 16'h8000 : 16'h9000;

	// LY + SCY is the effictive Y for Background, LX - 8 is the effective X
for Background
	// LY - WY is the effective Y for Window, LX - WX - 1 is the effective X
for Window
	`define GET_BG_TILE_ON_LINE_AT_x(x) (16'h9800 | {LCDC[3], 10'b0}) | {((LY
+ SCY) & 8'hF8), 2'b00} | ((``x + SCX) & 8'hF8) >> 3)
	`define GET_xth_BG_TILE_DATA0(x) LCDC[4] ? VRAM_DATA_BASE + {``x, 4'b0} |
{(LY + SCY) & 7, 1'b0} : VRAM_DATA_BASE -{``x[7], 11'b0} + {``x[6:0], 4'b0} |
{(LY + SCY) & 8'h07, 1'b0}
	`define GET_xth_BG_TILE_DATA1(x) LCDC[4] ? VRAM_DATA_BASE + {``x, 4'b0} |
{(LY + SCY) & 7, 1'b1} : VRAM_DATA_BASE -{``x[7], 11'b0} + {``x[6:0], 4'b0} |
{(LY + SCY) & 8'h07, 1'b1}
	`define GET_WD_TILE_ON_LINE_AT_x(x) (16'h9800 | {LCDC[6], 10'b0}) | {((LY
- WY) & 8'hF8), 2'b00}  | ((``x - WX - 1) & 8'hF8) >> 3)
```

```
	`define GET_xth_WD_TILE_DATA0(x) LCDC[4] ? VRAM_DATA_BASE + {``x, 4'b0} |
{(LY - WY) & 7, 1'b0} : VRAM_DATA_BASE -{``x[7], 11'b0} + {``x[6:0], 4'b0} |
{(LY - WY) & 8'h07, 1'b0}
	`define GET_xth_WD_TILE_DATA1(x) LCDC[4] ? VRAM_DATA_BASE + {``x, 4'b0} |
{(LY - WY) & 7, 1'b1} : VRAM_DATA_BASE -{``x[7], 11'b0} + {``x[6:0], 4'b0} |
{(LY - WY) & 8'h07, 1'b1}
	`define GET_xth_SP_TILE_DATA0(x) SP_FLAG[6] ? 16'h8000 + ({``x, 4'b0} |
(((8 + (LCDC[2] << 3) + sp_y_table[sp_to_fetch] - LY - 16 - 1) & 15) << 1)) :
16'h8000 + ({``x, 4'b0} | (((LY + 16 - sp_y_table[sp_to_fetch]) & 15) << 1))
	`define GET_xth_SP_TILE_DATA1(x) SP_FLAG[6] ? 16'h8000 + ({``x, 4'b0} |
(((8 + (LCDC[2] << 3) + sp_y_table[sp_to_fetch] - LY - 16 - 1) & 15) << 1)) + 1
: 16'h8000 + ({``x, 4'b0} | (((LY + 16 - sp_y_table[sp_to_fetch]) & 15) << 1))
+ 1


	logic isHitWD;


	// Fetched Data
	logic [15:0] BGWD_PPU_ADDR;
	//logic bgwd_to_fetch;
	logic [2:0] BGWD_CNT;
	logic [7:0] BGWD_MAP;
	logic [7:0] BGWD_TILE_DATA0, BGWD_TILE_DATA1;
	logic isFetchWD, isFetchWD_NEXT;
	logic FIRST_FETCH_WD_DONE, FIRST_FETCH_WD_DONE_NEXT;
	logic [1:0] BGWD_PX_MAP_A, BGWD_PX_MAP_B;
```

```
       logic BGWD_SHIFT_REG_SEL, BGWD_SHIFT_REG_SEL_NEXT; // 0 selects A, 1
selects B, selected shift register will run, unselected one will load
       logic [1:0] BGWD_SHIFT_REG_LOAD;

       PPU_SHIFT_REG BGWD_SHIFT_REG_A (.clk(clk), .rst(rst),
.data('{BGWD_TILE_DATA1, BGWD_TILE_DATA0}), .go(SHIFT_REG_GO &&
!BGWD_SHIFT_REG_SEL), .load(BGWD_SHIFT_REG_LOAD[0]), .q(BGWD_PX_MAP_A));
       PPU_SHIFT_REG BGWD_SHIFT_REG_B (.clk(clk), .rst(rst),
.data('{BGWD_TILE_DATA1, BGWD_TILE_DATA0}), .go(SHIFT_REG_GO &&
BGWD_SHIFT_REG_SEL), .load(BGWD_SHIFT_REG_LOAD[1]), .q(BGWD_PX_MAP_B));

       // Display Logic

       logic [1:0] BGWD_PX_MAP;
       assign BGWD_PX_MAP = BGWD_SHIFT_REG_SEL ? BGWD_PX_MAP_B : BGWD_PX_MAP_A;
       logic [1:0] BGWD_PX_DATA;

       assign BGWD_PX_DATA = {BGP[{BGWD_PX_MAP, 1'b1}],BGP[{BGWD_PX_MAP,
1'b0}]};

       always_comb
       begin
           PX_OUT = BGWD_PX_DATA;
           if (LCDC[1]) // Sprite Display?
           begin
               for (int i = 9 ; i > -1 ; i --)
```

```
                begin
                    if (SP_PRIPN[i][1] && (SP_PX_MAP[i] != 2'b00)) // SP below
BGWD
                    begin
                        PX_OUT = SP_PRIPN[i][0] ? {OBP1[{SP_PX_MAP[i], 1'b1}],
OBP1[{SP_PX_MAP[i], 1'b0}]} : {OBP0[{SP_PX_MAP[i], 1'b1}], OBP0[{SP_PX_MAP[i],
1'b0}]};
                    end
                end
            end
            if (LCDC[0]) // BG Display?
            begin
                PX_OUT = (BGWD_PX_MAP == 2'b00) ? PX_OUT : BGWD_PX_DATA;
            end
            if (LCDC[1]) // Sprite Display?
            begin
                for (int i = 9 ; i > -1 ; i --)
                begin
                    if (!SP_PRIPN[i][1] && (SP_PX_MAP[i] != 2'b00)) // SP above
BGWD
                    begin
                        PX_OUT = SP_PRIPN[i][0] ? {OBP1[{SP_PX_MAP[i], 1'b1}],
OBP1[{SP_PX_MAP[i], 1'b0}]} : {OBP0[{SP_PX_MAP[i], 1'b1}], OBP0[{SP_PX_MAP[i],
1'b0}]};
                    end
                end
```

```
            end
        end



        logic BGWD_SHIFT_REG_A_VALID, BGWD_SHIFT_REG_A_VALID_NEXT;
        logic BGWD_SHIFT_REG_B_VALID, BGWD_SHIFT_REG_B_VALID_NEXT;

        logic [2:0] RENDER_CNT, RENDER_CNT_NEXT;

        /* STAT Interrupts */
        logic IRQ_STAT, IRQ_STAT_NEXT; // The Internal IRQ signal, IRQ LCDC
Triggered on the rising edge of this

        always_ff @(posedge clk)
        begin
            if (rst) IRQ_STAT <= 0;
            else IRQ_STAT <= IRQ_STAT_NEXT;
        end

        always_comb
        begin
        IRQ_STAT_NEXT = (FF41_NEXT[6] && LY == LYC) ||
                        (FF41_NEXT[3] && PPU_STATE == H_BLANK) ||
                        (FF41_NEXT[5] && PPU_STATE == OAM_SEARCH) ||
                        ((FF41_NEXT[4] || FF41_NEXT[5]) && PPU_STATE == V_BLANK);
        IRQ_STAT_NEXT = IRQ_STAT_NEXT & LCDC[7];
```

```
end


assign IRQ_LCDC = IRQ_STAT_NEXT && !IRQ_STAT;


/* Register State Machine */
always_ff @(posedge clk)
begin
    if (rst)
    begin
        FF40 <= `NO_BOOT ? 8'h91 : 0;
        FF41 <= 0;
        FF42 <= 0;
        FF43 <= 0;
        FF45 <= 0;
        FF46 <= 0;
        FF47 <= `NO_BOOT ? 8'hFC : 0;
        FF48 <= `NO_BOOT ? 8'hFF : 0;
        FF49 <= `NO_BOOT ? 8'hFF : 0;
        FF4A <= 0;
        FF4B <= 0;
    end
    else
    begin
        FF40 <= FF40_NEXT;
        FF41 <= FF41_NEXT;
        FF42 <= FF42_NEXT;
```

```
                FF43 <= FF43_NEXT;
                FF45 <= FF45_NEXT;
                FF46 <= FF46_NEXT;
                FF47 <= FF47_NEXT;
                FF48 <= FF48_NEXT;
                FF49 <= FF49_NEXT;
                FF4A <= FF4A_NEXT;
                FF4B <= FF4B_NEXT;
            end
        end

        always_comb
        begin
            FF40_NEXT = (WR && (ADDR == 16'hFF40)) ? MMIO_DATA_out : FF40;
            FF41_NEXT = (WR && (ADDR == 16'hFF41)) ? {MMIO_DATA_out[7:3],
FF41[2:0]} : {FF41[7:3], LYC == LY, PPU_MODE};
            FF42_NEXT = (WR && (ADDR == 16'hFF42)) ? MMIO_DATA_out : FF42;
            FF43_NEXT = (WR && (ADDR == 16'hFF43)) ? MMIO_DATA_out : FF43;
            FF45_NEXT = (WR && (ADDR == 16'hFF45)) ? MMIO_DATA_out : FF45;
            FF46_NEXT = (WR && (ADDR == 16'hFF46)) ? MMIO_DATA_out : FF46;
            FF47_NEXT = (WR && (ADDR == 16'hFF47)) ? MMIO_DATA_out : FF47;
            FF48_NEXT = (WR && (ADDR == 16'hFF48)) ? MMIO_DATA_out : FF48;
            FF49_NEXT = (WR && (ADDR == 16'hFF49)) ? MMIO_DATA_out : FF49;
            FF4A_NEXT = (WR && (ADDR == 16'hFF4A)) ? MMIO_DATA_out : FF4A;
            FF4B_NEXT = (WR && (ADDR == 16'hFF4B)) ? MMIO_DATA_out : FF4B;
            case (ADDR)
```

```
            16'hFF40: MMIO_DATA_in = FF40;
            16'hFF41: MMIO_DATA_in = {1'b1, FF41[6:0]};
            16'hFF42: MMIO_DATA_in = FF42;
            16'hFF43: MMIO_DATA_in = FF43;
            16'hFF44: MMIO_DATA_in = FF44;
            16'hFF45: MMIO_DATA_in = FF45;
            16'hFF46: MMIO_DATA_in = FF46;
            16'hFF47: MMIO_DATA_in = FF47;
            16'hFF48: MMIO_DATA_in = FF48;
            16'hFF49: MMIO_DATA_in = FF49;
            16'hFF4A: MMIO_DATA_in = FF4A;
            16'hFF4B: MMIO_DATA_in = FF4B;
            default : MMIO_DATA_in = 8'hFF;
        endcase
    end


     /* PPU State Machine */
    always_ff @(posedge clk)
    begin
        if (rst)
        begin
            PPU_STATE <= V_BLANK;
            LX <= 0;
            LY <= 8'h91;
            PPU_CNT <= 0;
```

```
        sp_not_used <= 10'b11_1111_1111;
        SCX_CNT <= 0;
        isFetchWD <= 0;
        FIRST_FETCH_WD_DONE <= 0;

        BGWD_SHIFT_REG_SEL <= 0;
        BGWD_SHIFT_REG_A_VALID <= 0;
        BGWD_SHIFT_REG_B_VALID <= 0;

        RENDER_CNT <= 0;

        SP_NEXT_SLOT <= 0;

        for (int i = 0; i < 10; i++) SP_PRIPN[i] <= 0;
    end

    else
    begin
        PPU_STATE <= PPU_STATE_NEXT;
        LX <= LX_NEXT;
        LY <= LY_NEXT;
        PPU_CNT <= PPU_CNT_NEXT;

        sp_not_used <= sp_not_used_next;
        SCX_CNT <= SCX_CNT_NEXT;
```

```
            isFetchWD <= isFetchWD_NEXT;
            FIRST_FETCH_WD_DONE <= FIRST_FETCH_WD_DONE_NEXT;

            BGWD_SHIFT_REG_SEL <= BGWD_SHIFT_REG_SEL_NEXT;
            BGWD_SHIFT_REG_A_VALID <= BGWD_SHIFT_REG_A_VALID_NEXT;
            BGWD_SHIFT_REG_B_VALID <= BGWD_SHIFT_REG_B_VALID_NEXT;

            RENDER_CNT <= RENDER_CNT_NEXT;

            SP_NEXT_SLOT <= SP_NEXT_SLOT_NEXT;

            for (int i = 0; i < 10; i++) SP_PRIPN[i] <= SP_PRIPN_NEXT[i];

        end
    end

    always_comb
    begin
        // Registers Defualts
        PPU_STATE_NEXT = PPU_STATE;
        LX_NEXT = LX;
        LY_NEXT = LY;
        PPU_CNT_NEXT = PPU_CNT;


        SCX_CNT_NEXT = SCX_CNT;
```

```
sp_not_used_next = sp_not_used;


isFetchWD_NEXT = isFetchWD;
FIRST_FETCH_WD_DONE_NEXT = FIRST_FETCH_WD_DONE;


BGWD_SHIFT_REG_SEL_NEXT = BGWD_SHIFT_REG_SEL;
BGWD_SHIFT_REG_A_VALID_NEXT = BGWD_SHIFT_REG_A_VALID;
BGWD_SHIFT_REG_B_VALID_NEXT = BGWD_SHIFT_REG_B_VALID;


RENDER_CNT_NEXT = RENDER_CNT;


SP_NEXT_SLOT_NEXT = SP_NEXT_SLOT;


for (int i = 0; i < 10; i++) SP_PRIPN_NEXT[i] = SP_PRIPN[i];


// Combinational Defaults
PPU_ADDR = 0;
PPU_RD = 0;
PPU_MODE = 2'b01; // VBLANK


OAM_SEARCH_GO = 0;
BGWD_RENDER_GO = 0;


isHitWD = (WY <= LY) && (LX == WX + 1) && LCDC[5];
```

```
            SP_RENDER_GO = 0;
            SP_SHIFT_REG_LOAD = 0;

            SHIFT_REG_GO = 0;
            BGWD_SHIFT_REG_LOAD = 2'b00;

            PX_valid = 0;

            IRQ_PPU_V_BLANK = 0;

            if (LCDC[7]) // LCD Enable
            begin
                PPU_CNT_NEXT = PPU_CNT + 1;
                unique case (PPU_STATE)
                    OAM_SEARCH:
                    begin
                        PPU_MODE = 2'b10;
                        PPU_RD = 1;
                        OAM_SEARCH_GO = 1;
                        PPU_ADDR = PPU_CNT[0] ? OAM_BASE + (PPU_CNT << 1) - 1 :
OAM_BASE + (PPU_CNT << 1);
                        sp_not_used_next = 10'b11_1111_1111;
                        if (PPU_CNT == 79) PPU_STATE_NEXT = RENDER;
                    end

                    RENDER:
```

```
                    begin
                        PPU_MODE = 2'b11;
                        PPU_RD = 1;
                        if (isHitWD && !isFetchWD)
                        begin
                            RENDER_CNT_NEXT = 0;
                            BGWD_SHIFT_REG_A_VALID_NEXT = 0;
                            BGWD_SHIFT_REG_B_VALID_NEXT = 0;
                            isFetchWD_NEXT = 1;
                        end
                        else if ((!BGWD_SHIFT_REG_A_VALID ||
!BGWD_SHIFT_REG_B_VALID) && RENDER_CNT <= 6)
                        begin
                            BGWD_RENDER_GO = 1;
                            if (!isFetchWD)
                            begin
                                unique case (BGWD_CNT)
                                    0: PPU_ADDR = `GET_BG_TILE_ON_LINE_AT_x(LX);
                                    1: PPU_ADDR =
`GET_xth_BG_TILE_DATA0(BGWD_MAP);
                                    2: PPU_ADDR =
`GET_xth_BG_TILE_DATA1(BGWD_MAP);
                                    3,4,5:;
                                endcase
                            end
                            else
```

```
                            begin
                                unique case (BGWD_CNT)
                                    0: PPU_ADDR = `GET_WD_TILE_ON_LINE_AT_x(LX +
{FIRST_FETCH_WD_DONE, 3'b00});
                                    1: PPU_ADDR =
`GET_xth_WD_TILE_DATA0(BGWD_MAP);
                                    2: PPU_ADDR =
`GET_xth_WD_TILE_DATA1(BGWD_MAP);
                                    3,4,5:;
                                endcase
                            end
                            if (BGWD_CNT == (5 & {2'b11, !isHitSP})) // Why
sprite will only stall 5 - LX & 7 ?
                            begin
                                if (BGWD_SHIFT_REG_SEL)
                                begin
                                    BGWD_SHIFT_REG_A_VALID_NEXT = 1;
                                    BGWD_SHIFT_REG_LOAD[0] = 1;
                                end
                                else
                                begin
                                    BGWD_SHIFT_REG_B_VALID_NEXT = 1;
                                    BGWD_SHIFT_REG_LOAD[1] = 1;
                                end
                                if (!BGWD_SHIFT_REG_A_VALID &&
!BGWD_SHIFT_REG_B_VALID) BGWD_SHIFT_REG_SEL_NEXT = !BGWD_SHIFT_REG_SEL;
```

```
                            if (isFetchWD) FIRST_FETCH_WD_DONE_NEXT = 1;
                        end
                    end
                    else if (isHitSP)
                    begin
                        SP_RENDER_GO = 1;
                        unique case (SP_CNT)
                            0: PPU_ADDR = OAM_BASE +
sp_name_table_x[sp_to_fetch] + 2; // Get Pattern Number
                            1: PPU_ADDR = OAM_BASE +
sp_name_table_x[sp_to_fetch] + 3; // Get Attributes
                            2,3: PPU_ADDR = `GET_xth_SP_TILE_DATA0(LCDC[2] ?
{SP_MAP[7:1], 1'b0} : SP_MAP);
                            4,5: PPU_ADDR = `GET_xth_SP_TILE_DATA1(LCDC[2] ?
{SP_MAP[7:1], 1'b0} : SP_MAP);
                        endcase
                        if (SP_CNT == 5)
                        begin
                            sp_not_used_next[sp_to_fetch] = 0;
                            SP_SHIFT_REG_LOAD[SP_NEXT_SLOT] = 1;
                            SP_PRIPN_NEXT[SP_NEXT_SLOT] = {SP_FLAG[7],
SP_FLAG[4]};
                            SP_NEXT_SLOT_NEXT = SP_NEXT_SLOT + 1;
                        end
                    end
```

```
                        if ((BGWD_SHIFT_REG_A_VALID || BGWD_SHIFT_REG_B_VALID) &&
!isHitSP && !(isHitWD && !isFetchWD))
                        begin
                            RENDER_CNT_NEXT = RENDER_CNT + 1;
                            SHIFT_REG_GO = 1;

                            if (SCX_CNT != (SCX & 7)) SCX_CNT_NEXT = SCX_CNT + 1;
                            else
                            begin
                                LX_NEXT = LX + 1;
                                if (LX >= 8)
                                    PX_valid = 1; // On screen
                            end

                            if (RENDER_CNT == 7)
                            begin
                                BGWD_SHIFT_REG_SEL_NEXT = !BGWD_SHIFT_REG_SEL;
                                if (BGWD_SHIFT_REG_SEL == 0)
BGWD_SHIFT_REG_A_VALID_NEXT = 0;
                                else BGWD_SHIFT_REG_B_VALID_NEXT = 0;
                            end
                        end

                        if(LX_NEXT == 160 + 8) // Start of Horizontal Blank
                        begin
                            PPU_STATE_NEXT = H_BLANK;
```

```
            isFetchWD_NEXT = 0;
            FIRST_FETCH_WD_DONE_NEXT = 0;
            BGWD_SHIFT_REG_A_VALID_NEXT = 0;
            BGWD_SHIFT_REG_B_VALID_NEXT = 0;
            RENDER_CNT_NEXT = 0;
            sp_not_used_next = 10'b11_1111_1111;
            SP_NEXT_SLOT_NEXT = 0;
            SCX_CNT_NEXT = 0;
        end
    end

    H_BLANK:
    begin
        PPU_MODE = 2'b00;
        if (PPU_CNT == 455) // end of line
        begin
            LY_NEXT = LY + 1;
            LX_NEXT = 0;
            PPU_CNT_NEXT = 0;
            PPU_STATE_NEXT = OAM_SEARCH;
            if (LY_NEXT == 144)
            begin
                PPU_STATE_NEXT = V_BLANK;
                IRQ_PPU_V_BLANK = 1;
            end
        end
```

```
                    end


                    V_BLANK:
                    begin
                        PPU_MODE = 2'b01;
                        /*
                        " Line 153 takes only a few clocks to complete (the exact
timings are below). The rest of
                        the clocks of line 153 are spent in line 0 in mode 1! "
                        */
                        if (LY == 153)
                        begin
                            LY_NEXT = 0;
                            LX_NEXT = 0;
                        end
                        if (PPU_CNT == 455 && LY != 0) // end of line
                        begin
                            LY_NEXT = LY + 1;
                            PPU_CNT_NEXT = 0;
                        end
                        if (PPU_CNT == 455 && LY == 0)
                                    begin
                                           PPU_STATE_NEXT = OAM_SEARCH; // end of
Vertical Blank
                                           PPU_CNT_NEXT = 0;
                                    end
```

```
                end
            endcase
        end
        else // LCD is off
        begin
            PPU_MODE = 2'b00;
            LY_NEXT = 0;
            LX_NEXT = 0;
            PPU_CNT_NEXT = 0;
            PPU_STATE_NEXT = OAM_SEARCH;
            PPU_CNT_NEXT = 0;
        end
    end

    /* OAM Serach Machine */
    always_ff @(posedge clk)
    begin
        if (rst || PPU_STATE == H_BLANK) // reset at the end of the scanline
        begin
            sp_table_cnt <= 0;
            sp_found <= 0;
            for (int i = 0; i < 10; i ++)
            begin
                sp_y_table[i] <= 8'hFF;
                sp_x_table[i] <= 8'hFF;
            end
```

```
        end
        else if (OAM_SEARCH_GO)
        begin
            if (!PPU_CNT[0]) // even cycles
            begin
                if (isSpriteOnLine && (sp_table_cnt < 10))
                begin
                    sp_table_cnt <= (sp_table_cnt + 1);
                    sp_name_table[sp_table_cnt] <= (PPU_CNT >> 1);
                    sp_y_table[sp_table_cnt] <= PPU_DATA_in;
                    sp_found <= 1;
                end
            end
            else // odd cycles
            begin
                if (sp_found)
                begin
                    sp_x_table[sp_table_cnt - 1] <= PPU_DATA_in;
                end
                sp_found <= 0;
            end
        end
    end

    /* BGWD Machine */
    always_ff @(posedge clk)
```

```
begin
    if (rst || !BGWD_RENDER_GO)
    begin
        BGWD_CNT <= 0;
        BGWD_TILE_DATA0 <= 0;
        BGWD_TILE_DATA1 <= 0;
        BGWD_MAP <= 0;
    end
    else
    begin
        BGWD_CNT <= BGWD_CNT == 5 ? 0 : BGWD_CNT + 1;
        unique case (BGWD_CNT)
            0: BGWD_MAP <= PPU_DATA_in;
            1: BGWD_TILE_DATA0 <= PPU_DATA_in;
            2: BGWD_TILE_DATA1 <= PPU_DATA_in;
            3,4,5:;
        endcase
    end
end

/* Sprite Machine */
always_ff @(posedge clk)
begin
    if (rst || PPU_STATE == H_BLANK) // reset at the end of the scanline
    begin
        SP_CNT <= 0;
```

```
                SP_TILE_DATA0 <= 0;
                SP_TILE_DATA1 <= 0;
                SP_MAP <= 0;
                SP_FLAG <= 0;
            end
            else if (SP_RENDER_GO)
            begin
                SP_CNT <= (SP_CNT == 5) ? 0 : SP_CNT + 1;
                unique case (SP_CNT)
                    0: SP_MAP <= PPU_DATA_in;
                    1: SP_FLAG <= PPU_DATA_in;
                    //2,3: if (!SP_FLAG[5]) SP_TILE_DATA0 <= PPU_DATA_in; else
SP_TILE_DATA0 <= {<<{PPU_DATA_in}};
                    //4,5: if (!SP_FLAG[5]) SP_TILE_DATA1 <= PPU_DATA_in; else
SP_TILE_DATA1 <= {<<{PPU_DATA_in}};
                    2: if (!SP_FLAG[5]) SP_TILE_DATA0 <= PPU_DATA_in; else
SP_TILE_DATA0 <= {PPU_DATA_in[0], PPU_DATA_in[1], PPU_DATA_in[2],
PPU_DATA_in[3], PPU_DATA_in[4], PPU_DATA_in[5], PPU_DATA_in[6],
PPU_DATA_in[7]};
                    4: if (!SP_FLAG[5]) SP_TILE_DATA1 <= PPU_DATA_in; else
SP_TILE_DATA1 <= {PPU_DATA_in[0], PPU_DATA_in[1], PPU_DATA_in[2],
PPU_DATA_in[3], PPU_DATA_in[4], PPU_DATA_in[5], PPU_DATA_in[6],
PPU_DATA_in[7]};
                    3,5:;
                endcase
            end
```

```
end


always_comb
begin
    isHitSP = 0;
    sp_to_fetch = 0;
    if (LCDC[1])
    begin
        for (int i = 9; i >= 0; i--)
        begin
            if (sp_x_table[i] == LX && sp_not_used[i])
            begin
                isHitSP = 1;
                sp_to_fetch = i;
            end
        end
    end
end


endmodule


module PPU_SHIFT_REG
(
    input clk,
    input rst,
    input logic [7:0] data [1:0],
```

```
    input logic go,
    input logic load,
    output logic [1:0] q
);

logic [7:0] shift_reg [0:1];

always_ff @(posedge clk)
begin
    if (rst)
    begin
        shift_reg[0] <= 0;
        shift_reg[1] <= 0;
    end
    else if (load)
    begin
        shift_reg[0] <= data[0];
        shift_reg[1] <= data[1];
    end
    else
    begin
        if (go)
        begin
            shift_reg[0][7:1] <= shift_reg[0][6:0];
            shift_reg[0][0] <= 0;
            shift_reg[1][7:1] <= shift_reg[1][6:0];
```

```
                shift_reg[1][0] <= 0;
            end
        end
    end

    assign q = {shift_reg[1][7], shift_reg[0][7]};

    endmodule
```

PPU3_OAM_SEARCH.cpp

```
Unset
    #include <iostream>
    #include "VPPU3.h"
    #include <verilated.h>
    #include <verilated_vcd_c.h>

    typedef struct {
        char y;
        char x;
        char tile_no;
        char flags;

    } OAM_Ent;
```

```
int main(int argc, const char ** argv, const char ** env) {
	char OAM_BUFF[160];
	int i, time;
	OAM_Ent ent;
	int offset, exit_code;
	char LCDC;
	VPPU3 *dut;

	offset = exit_code = 0;

	for (i = 0; i < 160; i++)
		OAM_BUFF[i] = 0;

	ent.y = 16;
	ent.x = 30;
	ent.tile_no = 5;
	ent.flags = 5;

	for (i = 0; i < 5; i++) {
		OAM_BUFF[offset] = ent.y;
		OAM_BUFF[offset+1] = ent.x;
		OAM_BUFF[offset+2] = ent.tile_no;
		OAM_BUFF[offset+3] = ent.flags;

		offset += 4;
	}
```

```
            Verilated::commandArgs(argc, argv);

            dut = new VPPU3;    // Instantiate the ppu module

            Verilated::traceEverOn(true);
            VerilatedVcdC *tfp = new VerilatedVcdC;

            dut->trace(tfp, 99);
            tfp->open("ppu3.vcd");

            LCDC = 0xFF;

            dut->WR = 1;
            dut->ADDR = 0xFF40;
            dut->MMIO_DATA_out = LCDC;

            for (time = 0 ; time < 10000 ; time += 10) {
            dut->clk = ((time % 20) >= 10) ? 1 : 0;        // Simulate a 50
MHz clock

            dut->rst = (time == 30) ? 1 : 0;               // pulses rst

            dut->PPU_DATA_in = OAM_BUFF[dut->PPU_ADDR - 0xFE00];
            dut->eval();                                                  // Run
the simulation for a cycle
```

```
            tfp->dump(time);                                                //
Write the VCD file for this cycle
          }

            tfp->close(); // Stop dumping the VCD file
            delete tfp;

            dut->final(); // Stop the simulation
            delete dut;

            return exit_code;
      }
```

PPU3_W.sv

```
Unset
      `timescale 1ns / 1ns

      //`include "PPU.vh"
      `define NO_BOOT 0

      module PPU3
      (
```

```
            input logic clk,
            input logic rst,

            input logic [15:0] ADDR,
            input logic WR,
            input logic RD,
            input logic [7:0] MMIO_DATA_out,
            output logic [7:0] MMIO_DATA_in,

            output logic IRQ_PPU_V_BLANK,
            output logic IRQ_LCDC,

            output logic [1:0] PPU_MODE,

            output logic PPU_RD,
            output logic [15:0] PPU_ADDR,
            input logic [7:0] PPU_DATA_in,

            output logic [1:0] PX_OUT,
            output logic PX_valid
        );

        logic [7:0] LCDC, STAT, SCX, SCY, LYC, DMA, BGP, OBP0, OBP1, WX, WY; //
Register alias

        logic [7:0] FF40, FF40_NEXT;
```

```
assign LCDC = FF40;


logic [7:0] FF41, FF41_NEXT;
assign STAT = FF41;


logic [7:0] FF42, FF42_NEXT;
assign SCY = FF42;


logic [7:0] FF43, FF43_NEXT;
assign SCX = FF43;


logic [7:0] FF44;


logic [7:0] FF45, FF45_NEXT;
assign LYC = FF45;


logic [7:0] FF46, FF46_NEXT;
assign DMA = FF46;


logic [7:0] FF47, FF47_NEXT;
assign BGP = FF47;


logic [7:0] FF48, FF48_NEXT;
assign OBP0 = {FF48[7:2], 2'b00}; // Last 2 bits are not used


logic [7:0] FF49, FF49_NEXT;
```

```
        assign OBP1 = {FF49[7:2], 2'b00};


        logic [7:0] FF4A, FF4A_NEXT;
        assign WY = FF4A;


        logic [7:0] FF4B, FF4B_NEXT;
        assign WX = FF4B;


        typedef enum {OAM_SEARCH, RENDER, H_BLANK, V_BLANK} PPU_STATE_t;


        PPU_STATE_t PPU_STATE, PPU_STATE_NEXT;


        // Current Coordinates
        logic [7:0] LX, LX_NEXT; // LX starts from 0, LCD starts from LX + SCX &
7
        logic [7:0] LY, LY_NEXT;
        assign FF44 = LY;



        // OAM Machine
        logic OAM_SEARCH_GO;
        logic [15:0] OAM_SEARCH_PPU_ADDR;


        // BGWD Machine
        logic BGWD_RENDER_GO;
        logic SHIFT_REG_GO;
```

```
        // Current Rendering Tile Map Pattern Number
        logic [7:0] BG_MAP;
        logic [7:0] WD_MAP;
        logic [7:0] SP_MAP;

        // PPU Running Counter for every 60Hz refresh
        shortint unsigned PPU_CNT, PPU_CNT_NEXT;
        logic [2:0] SCX_CNT, SCX_CNT_NEXT;

        //assign IRQ_PPU_V_BLANK = (LY == 144 && PPU_CNT == 0);

        // Sprite Logic
        logic isSpriteOnLine;
        assign isSpriteOnLine = (((PPU_DATA_in + (LCDC[2] << 3)) > (LY + 8)) &&
(PPU_DATA_in <= (LY + 16)));
        logic [3:0] sp_table_cnt; //sp_table_cnt_next;
        logic [5:0] sp_name_table [0:9]; //logic [5:0] sp_name_table_next [0:9];
        logic [7:0] sp_name_table_x [0:9];

        genvar sp_n_gi;
        generate
        for (sp_n_gi = 0; sp_n_gi < 10; sp_n_gi++)
        begin : sp_n_gen
            assign sp_name_table_x[sp_n_gi] = {sp_name_table[sp_n_gi], 2'b00};
        end
```

```
        endgenerate


        logic [7:0] sp_y_table [0:9];    //logic [7:0] sp_y_table_next [0:9];
        logic [7:0] sp_x_table [0:9];    //logic [7:0] sp_x_table_next [0:9];
        logic sp_found; //sp_found_next; // Search Result
        logic isHitSP; // is there a sprite to fetch on current X?
        logic [3:0] sp_to_fetch;
        logic [9:0] sp_not_used, sp_not_used_next; // which sprite has been used
        logic SP_RENDER_GO;
        //logic [15:0] SPRITE_PPU_ADDR;
        logic [9:0] SP_SHIFT_REG_LOAD;
        logic [8:0] SP_TILE_DATA0, SP_TILE_DATA1;
        logic [1:0] SP_PX_MAP [9:0];
        logic [3:0] SP_NEXT_SLOT, SP_NEXT_SLOT_NEXT;
        logic [2:0] SP_CNT;
        logic [7:0] SP_FLAG;
        logic [1:0] SP_PRIPN [0:9];
        logic [1:0] SP_PRIPN_NEXT [0:9];


        PPU_SHIFT_REG SP_SHIFT_REG9(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[9]),
.q(SP_PX_MAP[9]));
        PPU_SHIFT_REG SP_SHIFT_REG8(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[8]),
.q(SP_PX_MAP[8]));
```

```
      PPU_SHIFT_REG SP_SHIFT_REG7(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[7]),
.q(SP_PX_MAP[7]));
      PPU_SHIFT_REG SP_SHIFT_REG6(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[6]),
.q(SP_PX_MAP[6]));
      PPU_SHIFT_REG SP_SHIFT_REG5(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[5]),
.q(SP_PX_MAP[5]));
      PPU_SHIFT_REG SP_SHIFT_REG4(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[4]),
.q(SP_PX_MAP[4]));
      PPU_SHIFT_REG SP_SHIFT_REG3(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[3]),
.q(SP_PX_MAP[3]));
      PPU_SHIFT_REG SP_SHIFT_REG2(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[2]),
.q(SP_PX_MAP[2]));
      PPU_SHIFT_REG SP_SHIFT_REG1(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[1]),
.q(SP_PX_MAP[1]));
      PPU_SHIFT_REG SP_SHIFT_REG0(.clk(clk), .rst(rst), .data('{SP_TILE_DATA1,
SP_TILE_DATA0}), .go(SHIFT_REG_GO), .load(SP_SHIFT_REG_LOAD[0]),
.q(SP_PX_MAP[0]));

      // Fetch Logic
```

```
        localparam OAM_BASE = 16'hFE00;
        logic [15:0] VRAM_DATA_BASE;
        assign VRAM_DATA_BASE = LCDC[4] ? 16'h8000 : 16'h9000;

        // LY + SCY is the effictive Y for Background, LX - 8 is the effective X
for Background
        // LY - WY is the effective Y for Window, LX - WX - 1 is the effective X
for Window
        `define GET_BG_TILE_ON_LINE_AT_x(x) (16'h9800 | {LCDC[3], 10'b0}) | {((LY
+ SCY) & 8'hF8), 2'b00} | ((((``x + SCX) & 8'hF8) >> 3)
        `define GET_xth_BG_TILE_DATA0(x) LCDC[4] ? VRAM_DATA_BASE + {``x, 4'b0} |
{(LY + SCY) & 7, 1'b0} : VRAM_DATA_BASE -{``x[7], 11'b0} + {``x[6:0], 4'b0} |
{(LY + SCY) & 8'h07, 1'b0}
        `define GET_xth_BG_TILE_DATA1(x) LCDC[4] ? VRAM_DATA_BASE + {``x, 4'b0} |
{(LY + SCY) & 7, 1'b1} : VRAM_DATA_BASE -{``x[7], 11'b0} + {``x[6:0], 4'b0} |
{(LY + SCY) & 8'h07, 1'b1}
        `define GET_WD_TILE_ON_LINE_AT_x(x) (16'h9800 | {LCDC[6], 10'b0}) | {((LY
- WY) & 8'hF8), 2'b00}  | ((((``x - WX - 1) & 8'hF8) >> 3)
        `define GET_xth_WD_TILE_DATA0(x) LCDC[4] ? VRAM_DATA_BASE + {``x, 4'b0} |
{(LY - WY) & 7, 1'b0} : VRAM_DATA_BASE -{``x[7], 11'b0} + {``x[6:0], 4'b0} |
{(LY - WY) & 8'h07, 1'b0}
        `define GET_xth_WD_TILE_DATA1(x) LCDC[4] ? VRAM_DATA_BASE + {``x, 4'b0} |
{(LY - WY) & 7, 1'b1} : VRAM_DATA_BASE -{``x[7], 11'b0} + {``x[6:0], 4'b0} |
{(LY - WY) & 8'h07, 1'b1}
```

```
	`define GET_xth_SP_TILE_DATA0(x) SP_FLAG[6] ? 16'h8000 + ({``x, 4'b0} |
(((8 + (LCDC[2] << 3) + sp_y_table[sp_to_fetch] - LY - 16 - 1) & 15) << 1)) :
16'h8000 + ({``x, 4'b0} | (((LY + 16 - sp_y_table[sp_to_fetch]) & 15) << 1))
	`define GET_xth_SP_TILE_DATA1(x) SP_FLAG[6] ? 16'h8000 + ({``x, 4'b0} |
(((8 + (LCDC[2] << 3) + sp_y_table[sp_to_fetch] - LY - 16 - 1) & 15) << 1)) + 1
: 16'h8000 + ({``x, 4'b0} | (((LY + 16 - sp_y_table[sp_to_fetch]) & 15) << 1))
+ 1


	logic isHitWD;


	// Fetched Data
	logic [15:0] BGWD_PPU_ADDR;
	//logic bgwd_to_fetch;
	logic [2:0] BGWD_CNT;
	logic [7:0] BGWD_MAP;
	logic [7:0] BGWD_TILE_DATA0, BGWD_TILE_DATA1;
	logic isFetchWD, isFetchWD_NEXT;
	logic FIRST_FETCH_WD_DONE, FIRST_FETCH_WD_DONE_NEXT;
	logic [1:0] BGWD_PX_MAP_A, BGWD_PX_MAP_B;
	logic BGWD_SHIFT_REG_SEL, BGWD_SHIFT_REG_SEL_NEXT; // 0 selects A, 1
selects B, selected shift register will run, unselected one will load
	logic [1:0] BGWD_SHIFT_REG_LOAD;


	PPU_SHIFT_REG BGWD_SHIFT_REG_A (.clk(clk), .rst(rst),
.data('{BGWD_TILE_DATA1, BGWD_TILE_DATA0}), .go(SHIFT_REG_GO &&
!BGWD_SHIFT_REG_SEL), .load(BGWD_SHIFT_REG_LOAD[0]), .q(BGWD_PX_MAP_A));
```

```
       PPU_SHIFT_REG BGWD_SHIFT_REG_B (.clk(clk), .rst(rst),
.data('{BGWD_TILE_DATA1, BGWD_TILE_DATA0}), .go(SHIFT_REG_GO &&
BGWD_SHIFT_REG_SEL), .load(BGWD_SHIFT_REG_LOAD[1]), .q(BGWD_PX_MAP_B));

       // Display Logic

       logic [1:0] BGWD_PX_MAP;
       assign BGWD_PX_MAP = BGWD_SHIFT_REG_SEL ? BGWD_PX_MAP_B : BGWD_PX_MAP_A;
       logic [1:0] BGWD_PX_DATA;

       assign BGWD_PX_DATA = {BGP[{BGWD_PX_MAP, 1'b1}],BGP[{BGWD_PX_MAP,
1'b0}]};

       always_comb
       begin
           PX_OUT = BGWD_PX_DATA;
           if (LCDC[1]) // Sprite Display?
           begin
               for (int i = 9 ; i > -1 ; i --)
               begin
                   if (SP_PRIPN[i][1] && (SP_PX_MAP[i] != 2'b00)) // SP below
BGWD
                   begin
                       PX_OUT = SP_PRIPN[i][0] ? {OBP1[{SP_PX_MAP[i], 1'b1}],
OBP1[{SP_PX_MAP[i], 1'b0}]} : {OBP0[{SP_PX_MAP[i], 1'b1}], OBP0[{SP_PX_MAP[i],
1'b0}]};
```

```
                end
            end
        end
        if (LCDC[0]) // BG Display?
        begin
            PX_OUT = (BGWD_PX_MAP == 2'b00) ? PX_OUT : BGWD_PX_DATA;
        end
        if (LCDC[1]) // Sprite Display?
        begin
            for (int i = 9 ; i > -1 ; i --)
            begin
                if (!SP_PRIPN[i][1] && (SP_PX_MAP[i] != 2'b00)) // SP above
BGWD
                begin
                    PX_OUT = SP_PRIPN[i][0] ? {OBP1[{SP_PX_MAP[i], 1'b1}],
OBP1[{SP_PX_MAP[i], 1'b0}]} : {OBP0[{SP_PX_MAP[i], 1'b1}], OBP0[{SP_PX_MAP[i],
1'b0}]};
                end
            end
        end
    end



    logic BGWD_SHIFT_REG_A_VALID, BGWD_SHIFT_REG_A_VALID_NEXT;
    logic BGWD_SHIFT_REG_B_VALID, BGWD_SHIFT_REG_B_VALID_NEXT;
```

```
        logic [2:0] RENDER_CNT, RENDER_CNT_NEXT;


        /* STAT Interrupts */
        logic IRQ_STAT, IRQ_STAT_NEXT; // The Internal IRQ signal, IRQ LCDC
Triggered on the rising edge of this


        always_ff @(posedge clk)
        begin
            if (rst) IRQ_STAT <= 0;
            else IRQ_STAT <= IRQ_STAT_NEXT;
        end


        always_comb
        begin
        IRQ_STAT_NEXT = (FF41_NEXT[6] && LY == LYC) ||
                        (FF41_NEXT[3] && PPU_STATE == H_BLANK) ||
                        (FF41_NEXT[5] && PPU_STATE == OAM_SEARCH) ||
                        ((FF41_NEXT[4] || FF41_NEXT[5]) && PPU_STATE == V_BLANK);
        IRQ_STAT_NEXT = IRQ_STAT_NEXT & LCDC[7];
        end


        assign IRQ_LCDC = IRQ_STAT_NEXT && !IRQ_STAT;


        /* Register State Machine */
        always_ff @(posedge clk)
        begin
```

```
        if (rst)
        begin
            FF40 <= `NO_BOOT ? 8'h91 : 0;
            FF41 <= 0;
            FF42 <= 0;
            FF43 <= 0;
            FF45 <= 0;
            FF46 <= 0;
            FF47 <= `NO_BOOT ? 8'hFC : 0;
            FF48 <= `NO_BOOT ? 8'hFF : 0;
            FF49 <= `NO_BOOT ? 8'hFF : 0;
            FF4A <= 0;
            FF4B <= 0;
        end
        else
        begin
            FF40 <= FF40_NEXT;
            FF41 <= FF41_NEXT;
            FF42 <= FF42_NEXT;
            FF43 <= FF43_NEXT;
            FF45 <= FF45_NEXT;
            FF46 <= FF46_NEXT;
            FF47 <= FF47_NEXT;
            FF48 <= FF48_NEXT;
            FF49 <= FF49_NEXT;
            FF4A <= FF4A_NEXT;
```

```
                FF4B <= FF4B_NEXT;
            end
        end

        always_comb
        begin
            FF40_NEXT = (WR && (ADDR == 16'hFF40)) ? MMIO_DATA_out : FF40;
            FF41_NEXT = (WR && (ADDR == 16'hFF41)) ? {MMIO_DATA_out[7:3],
FF41[2:0]} : {FF41[7:3], LYC == LY, PPU_MODE};
            FF42_NEXT = (WR && (ADDR == 16'hFF42)) ? MMIO_DATA_out : FF42;
            FF43_NEXT = (WR && (ADDR == 16'hFF43)) ? MMIO_DATA_out : FF43;
            FF45_NEXT = (WR && (ADDR == 16'hFF45)) ? MMIO_DATA_out : FF45;
            FF46_NEXT = (WR && (ADDR == 16'hFF46)) ? MMIO_DATA_out : FF46;
            FF47_NEXT = (WR && (ADDR == 16'hFF47)) ? MMIO_DATA_out : FF47;
            FF48_NEXT = (WR && (ADDR == 16'hFF48)) ? MMIO_DATA_out : FF48;
            FF49_NEXT = (WR && (ADDR == 16'hFF49)) ? MMIO_DATA_out : FF49;
            FF4A_NEXT = (WR && (ADDR == 16'hFF4A)) ? MMIO_DATA_out : FF4A;
            FF4B_NEXT = (WR && (ADDR == 16'hFF4B)) ? MMIO_DATA_out : FF4B;
            case (ADDR)
                16'hFF40: MMIO_DATA_in = FF40;
                16'hFF41: MMIO_DATA_in = {1'b1, FF41[6:0]};
                16'hFF42: MMIO_DATA_in = FF42;
                16'hFF43: MMIO_DATA_in = FF43;
                16'hFF44: MMIO_DATA_in = FF44;
                16'hFF45: MMIO_DATA_in = FF45;
                16'hFF46: MMIO_DATA_in = FF46;
```

```
            16'hFF47: MMIO_DATA_in = FF47;
            16'hFF48: MMIO_DATA_in = FF48;
            16'hFF49: MMIO_DATA_in = FF49;
            16'hFF4A: MMIO_DATA_in = FF4A;
            16'hFF4B: MMIO_DATA_in = FF4B;
            default : MMIO_DATA_in = 8'hFF;
        endcase
    end

     /* PPU State Machine */
    always_ff @(posedge clk)
    begin
        if (rst)
        begin
            PPU_STATE <= V_BLANK;
            LX <= 0;
            LY <= 8'h91;
            PPU_CNT <= 0;

            sp_not_used <= 10'b11_1111_1111;
            SCX_CNT <= 0;
            isFetchWD <= 0;
            FIRST_FETCH_WD_DONE <= 0;

            BGWD_SHIFT_REG_SEL <= 0;
            BGWD_SHIFT_REG_A_VALID <= 0;
```

```
        BGWD_SHIFT_REG_B_VALID <= 0;


        RENDER_CNT <= 0;


        SP_NEXT_SLOT <= 0;


        for (int i = 0; i < 10; i++) SP_PRIPN[i] <= 0;
    end


    else
    begin
        PPU_STATE <= PPU_STATE_NEXT;
        LX <= LX_NEXT;
        LY <= LY_NEXT;
        PPU_CNT <= PPU_CNT_NEXT;


        sp_not_used <= sp_not_used_next;
        SCX_CNT <= SCX_CNT_NEXT;


        isFetchWD <= isFetchWD_NEXT;
        FIRST_FETCH_WD_DONE <= FIRST_FETCH_WD_DONE_NEXT;


        BGWD_SHIFT_REG_SEL <= BGWD_SHIFT_REG_SEL_NEXT;
        BGWD_SHIFT_REG_A_VALID <= BGWD_SHIFT_REG_A_VALID_NEXT;
        BGWD_SHIFT_REG_B_VALID <= BGWD_SHIFT_REG_B_VALID_NEXT;
```

```
            RENDER_CNT <= RENDER_CNT_NEXT;


            SP_NEXT_SLOT <= SP_NEXT_SLOT_NEXT;


            for (int i = 0; i < 10; i++) SP_PRIPN[i] <= SP_PRIPN_NEXT[i];


        end
    end


    always_comb
    begin
        // Registers Defualts
        PPU_STATE_NEXT = PPU_STATE;
        LX_NEXT = LX;
        LY_NEXT = LY;
        PPU_CNT_NEXT = PPU_CNT;



        SCX_CNT_NEXT = SCX_CNT;


        sp_not_used_next = sp_not_used;


        isFetchWD_NEXT = isFetchWD;
        FIRST_FETCH_WD_DONE_NEXT = FIRST_FETCH_WD_DONE;


        BGWD_SHIFT_REG_SEL_NEXT = BGWD_SHIFT_REG_SEL;
```

```
        BGWD_SHIFT_REG_A_VALID_NEXT = BGWD_SHIFT_REG_A_VALID;
        BGWD_SHIFT_REG_B_VALID_NEXT = BGWD_SHIFT_REG_B_VALID;

        RENDER_CNT_NEXT = RENDER_CNT;

        SP_NEXT_SLOT_NEXT = SP_NEXT_SLOT;

        for (int i = 0; i < 10; i++) SP_PRIPN_NEXT[i] = SP_PRIPN[i];

        // Combinational Defaults
        PPU_ADDR = 0;
        PPU_RD = 0;
        PPU_MODE = 2'b01; // VBLANK

        OAM_SEARCH_GO = 0;
        BGWD_RENDER_GO = 0;

        isHitWD = (WY <= LY) && (LX == WX + 1) && LCDC[5];

        SP_RENDER_GO = 0;
        SP_SHIFT_REG_LOAD = 0;

        SHIFT_REG_GO = 0;
        BGWD_SHIFT_REG_LOAD = 2'b00;

        PX_valid = 0;
```

```
            IRQ_PPU_V_BLANK = 0;


            if (LCDC[7]) // LCD Enable
            begin
                PPU_CNT_NEXT = PPU_CNT + 1;
                unique case (PPU_STATE)
                    OAM_SEARCH:
                    begin
                        PPU_MODE = 2'b10;
                        PPU_RD = 1;
                        OAM_SEARCH_GO = 1;
                        PPU_ADDR = PPU_CNT[0] ? OAM_BASE + (PPU_CNT << 1) - 1 :
OAM_BASE + (PPU_CNT << 1);
                        sp_not_used_next = 10'b11_1111_1111;
                        if (PPU_CNT == 79) PPU_STATE_NEXT = RENDER;
                    end


                    RENDER:
                    begin
                        PPU_MODE = 2'b11;
                        PPU_RD = 1;
                        if (isHitWD && !isFetchWD)
                        begin
                            RENDER_CNT_NEXT = 0;
                            BGWD_SHIFT_REG_A_VALID_NEXT = 0;
```

```
                            BGWD_SHIFT_REG_B_VALID_NEXT = 0;
                            isFetchWD_NEXT = 1;
                        end
                        else if ((!BGWD_SHIFT_REG_A_VALID ||
!BGWD_SHIFT_REG_B_VALID) && RENDER_CNT <= 6)
                        begin
                            BGWD_RENDER_GO = 1;
                            if (!isFetchWD)
                            begin
                                unique case (BGWD_CNT)
                                    0: PPU_ADDR = `GET_BG_TILE_ON_LINE_AT_x(LX);
                                    1: PPU_ADDR =
`GET_xth_BG_TILE_DATA0(BGWD_MAP);
                                    2: PPU_ADDR =
`GET_xth_BG_TILE_DATA1(BGWD_MAP);
                                    3,4,5:;
                                endcase
                            end
                            else
                            begin
                                unique case (BGWD_CNT)
                                    0: PPU_ADDR = `GET_WD_TILE_ON_LINE_AT_x(LX +
{FIRST_FETCH_WD_DONE, 3'b00});
                                    1: PPU_ADDR =
`GET_xth_WD_TILE_DATA0(BGWD_MAP);
```

```
                                    2: PPU_ADDR =
`GET_xth_WD_TILE_DATA1(BGWD_MAP);
                                    3,4,5:;
                                endcase
                            end
                            if (BGWD_CNT == (5 & {2'b11, !isHitSP})) // Why
sprite will only stall 5 - LX & 7 ?
                            begin
                                if (BGWD_SHIFT_REG_SEL)
                                begin
                                    BGWD_SHIFT_REG_A_VALID_NEXT = 1;
                                    BGWD_SHIFT_REG_LOAD[0] = 1;
                                end
                                else
                                begin
                                    BGWD_SHIFT_REG_B_VALID_NEXT = 1;
                                    BGWD_SHIFT_REG_LOAD[1] = 1;
                                end
                                if (!BGWD_SHIFT_REG_A_VALID &&
!BGWD_SHIFT_REG_B_VALID) BGWD_SHIFT_REG_SEL_NEXT = !BGWD_SHIFT_REG_SEL;
                                if (isFetchWD) FIRST_FETCH_WD_DONE_NEXT = 1;
                            end
                        end
                        else if (isHitSP)
                        begin
                            SP_RENDER_GO = 1;
```

```
                            unique case (SP_CNT)
                                0: PPU_ADDR = OAM_BASE +
sp_name_table_x[sp_to_fetch] + 2; // Get Pattern Number
                                1: PPU_ADDR = OAM_BASE +
sp_name_table_x[sp_to_fetch] + 3; // Get Attributes
                                2,3: PPU_ADDR = `GET_xth_SP_TILE_DATA0(LCDC[2] ?
{SP_MAP[7:1], 1'b0} : SP_MAP);
                                4,5: PPU_ADDR = `GET_xth_SP_TILE_DATA1(LCDC[2] ?
{SP_MAP[7:1], 1'b0} : SP_MAP);
                            endcase
                            if (SP_CNT == 5)
                            begin
                                sp_not_used_next[sp_to_fetch] = 0;
                                SP_SHIFT_REG_LOAD[SP_NEXT_SLOT] = 1;
                                SP_PRIPN_NEXT[SP_NEXT_SLOT] = {SP_FLAG[7],
SP_FLAG[4]};
                                SP_NEXT_SLOT_NEXT = SP_NEXT_SLOT + 1;
                            end
                        end

                        if ((BGWD_SHIFT_REG_A_VALID || BGWD_SHIFT_REG_B_VALID) &&
!isHitSP && !(isHitWD && !isFetchWD))
                        begin
                            RENDER_CNT_NEXT = RENDER_CNT + 1;
                            SHIFT_REG_GO = 1;
```

```
                            if (SCX_CNT != (SCX & 7)) SCX_CNT_NEXT = SCX_CNT + 1;
                            else
                            begin
                                LX_NEXT = LX + 1;
                                if (LX >= 8)
                                    PX_valid = 1; // On screen
                            end

                            if (RENDER_CNT == 7)
                            begin
                                BGWD_SHIFT_REG_SEL_NEXT = !BGWD_SHIFT_REG_SEL;
                                if (BGWD_SHIFT_REG_SEL == 0)
BGWD_SHIFT_REG_A_VALID_NEXT = 0;
                                else BGWD_SHIFT_REG_B_VALID_NEXT = 0;
                            end
                        end

                        if(LX_NEXT == 160 + 8) // Start of Horizontal Blank
                        begin
                            PPU_STATE_NEXT = H_BLANK;
                            isFetchWD_NEXT = 0;
                            FIRST_FETCH_WD_DONE_NEXT = 0;
                            BGWD_SHIFT_REG_A_VALID_NEXT = 0;
                            BGWD_SHIFT_REG_B_VALID_NEXT = 0;
                            RENDER_CNT_NEXT = 0;
                            sp_not_used_next = 10'b11_1111_1111;
```

```
            SP_NEXT_SLOT_NEXT = 0;
            SCX_CNT_NEXT = 0;
        end
    end

    H_BLANK:
    begin
        PPU_MODE = 2'b00;
        if (PPU_CNT == 455) // end of line
        begin
            LY_NEXT = LY + 1;
            LX_NEXT = 0;
            PPU_CNT_NEXT = 0;
            PPU_STATE_NEXT = OAM_SEARCH;
            if (LY_NEXT == 144)
            begin
                PPU_STATE_NEXT = V_BLANK;
                IRQ_PPU_V_BLANK = 1;
            end
        end
    end

    V_BLANK:
    begin
        PPU_MODE = 2'b01;
        /*
```

```
                        " Line 153 takes only a few clocks to complete (the exact
timings are below). The rest of
                        the clocks of line 153 are spent in line 0 in mode 1! "
                        */
                        if (LY == 153)
                        begin
                            LY_NEXT = 0;
                            LX_NEXT = 0;
                        end
                        if (PPU_CNT == 455 && LY != 0) // end of line
                        begin
                            LY_NEXT = LY + 1;
                            PPU_CNT_NEXT = 0;
                        end
                        if (PPU_CNT == 455 && LY == 0)
                                    begin
                                           PPU_STATE_NEXT = OAM_SEARCH; // end of
Vertical Blank
                                           PPU_CNT_NEXT = 0;
                                    end
                    end
                endcase
            end
            else // LCD is off
            begin
                PPU_MODE = 2'b00;
```

```
            LY_NEXT = 0;
            LX_NEXT = 0;
            PPU_CNT_NEXT = 0;
            PPU_STATE_NEXT = OAM_SEARCH;
            PPU_CNT_NEXT = 0;
        end
    end

    /* OAM Serach Machine */
    always_ff @(posedge clk)
    begin
        if (rst || PPU_STATE == H_BLANK) // reset at the end of the scanline
        begin
            sp_table_cnt <= 0;
            sp_found <= 0;
            for (int i = 0; i < 10; i ++)
            begin
                sp_y_table[i] <= 8'hFF;
                sp_x_table[i] <= 8'hFF;
            end
        end
        else if (OAM_SEARCH_GO)
        begin
            if (!PPU_CNT[0]) // even cycles
            begin
                if (isSpriteOnLine && (sp_table_cnt < 10))
```

```
                begin
                    sp_table_cnt <= (sp_table_cnt + 1);
                    sp_name_table[sp_table_cnt] <= (PPU_CNT >> 1);
                    sp_y_table[sp_table_cnt] <= PPU_DATA_in;
                    sp_found <= 1;
                end
            end
            else // odd cycles
            begin
                if (sp_found)
                begin
                    sp_x_table[sp_table_cnt - 1] <= PPU_DATA_in;
                end
                sp_found <= 0;
            end
        end
    end

    /* BGWD Machine */
    always_ff @(posedge clk)
    begin
        if (rst || !BGWD_RENDER_GO)
        begin
            BGWD_CNT <= 0;
            BGWD_TILE_DATA0 <= 0;
            BGWD_TILE_DATA1 <= 0;
```

```
            BGWD_MAP <= 0;
        end
        else
        begin
            BGWD_CNT <= BGWD_CNT == 5 ? 0 : BGWD_CNT + 1;
            unique case (BGWD_CNT)
                0: BGWD_MAP <= PPU_DATA_in;
                1: BGWD_TILE_DATA0 <= PPU_DATA_in;
                2: BGWD_TILE_DATA1 <= PPU_DATA_in;
                3,4,5:;
            endcase
        end
    end

    /* Sprite Machine */
    always_ff @(posedge clk)
    begin
        if (rst || PPU_STATE == H_BLANK) // reset at the end of the scanline
        begin
            SP_CNT <= 0;
            SP_TILE_DATA0 <= 0;
            SP_TILE_DATA1 <= 0;
            SP_MAP <= 0;
            SP_FLAG <= 0;
        end
        else if (SP_RENDER_GO)
```

```
            begin
                SP_CNT <= (SP_CNT == 5) ? 0 : SP_CNT + 1;
                unique case (SP_CNT)
                    0: SP_MAP <= PPU_DATA_in;
                    1: SP_FLAG <= PPU_DATA_in;
                    //2,3: if (!SP_FLAG[5]) SP_TILE_DATA0 <= PPU_DATA_in; else
SP_TILE_DATA0 <= {<<{PPU_DATA_in}};
                    //4,5: if (!SP_FLAG[5]) SP_TILE_DATA1 <= PPU_DATA_in; else
SP_TILE_DATA1 <= {<<{PPU_DATA_in}};
                    2: if (!SP_FLAG[5]) SP_TILE_DATA0 <= PPU_DATA_in; else
SP_TILE_DATA0 <= {PPU_DATA_in[0], PPU_DATA_in[1], PPU_DATA_in[2],
PPU_DATA_in[3], PPU_DATA_in[4], PPU_DATA_in[5], PPU_DATA_in[6],
PPU_DATA_in[7]};
                    4: if (!SP_FLAG[5]) SP_TILE_DATA1 <= PPU_DATA_in; else
SP_TILE_DATA1 <= {PPU_DATA_in[0], PPU_DATA_in[1], PPU_DATA_in[2],
PPU_DATA_in[3], PPU_DATA_in[4], PPU_DATA_in[5], PPU_DATA_in[6],
PPU_DATA_in[7]};
                    3,5:;
                endcase
            end
        end


        always_comb
        begin
            isHitSP = 0;
            sp_to_fetch = 0;
```

```
        if (LCDC[1])
        begin
            for (int i = 9; i >= 0; i--)
            begin
                if (sp_x_table[i] == LX && sp_not_used[i])
                begin
                    isHitSP = 1;
                    sp_to_fetch = i;
                end
            end
        end
    end

    endmodule

    module PPU_SHIFT_REG
    (
        input clk,
        input rst,
        input logic [7:0] data [1:0],
        input logic go,
        input logic load,
        output logic [1:0] q
    );

    logic [7:0] shift_reg [0:1];
```

```
always_ff @(posedge clk)
begin
    if (rst)
    begin
        shift_reg[0] <= 0;
        shift_reg[1] <= 0;
    end
    else if (load)
    begin
        shift_reg[0] <= data[0];
        shift_reg[1] <= data[1];
    end
    else
    begin
        if (go)
        begin
            shift_reg[0][7:1] <= shift_reg[0][6:0];
            shift_reg[0][0] <= 0;
            shift_reg[1][7:1] <= shift_reg[1][6:0];
            shift_reg[1][0] <= 0;
        end
    end
end

assign q = {shift_reg[1][7], shift_reg[0][7]};
```

```
endmodule
```

Quartus_dual_port_dual_clk_ram.sv

```
Unset
module Quartus_dual_port_dual_clk_ram
#(parameter DATA_WIDTH=2, parameter ADDR_WIDTH=15, parameter DEPTH =
23040)
(
	input [(DATA_WIDTH-1):0] data,
	input [(ADDR_WIDTH-1):0] read_addr, write_addr,
	input we, read_clk, write_clk,
	output reg [(DATA_WIDTH-1):0] q
);

	// Declare the RAM variable
	reg [DATA_WIDTH-1:0] ram[DEPTH-1:0];

	always @ (posedge write_clk)
	begin
		// Write
		if (we)
```

```
                        ram[write_addr] <= data;
        end

        always @ (posedge read_clk)
        begin
                // Read
                q <= ram[read_addr];
        end

endmodule
```

Quartus_single_port_ram.sv

```
Unset
        // Quartus Prime Verilog Template
        // Single port RAM with single read/write address

        module Quartus_single_port_ram
        #(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=8, DEPTH = 160)
        (
                input [(DATA_WIDTH-1):0] data,
                input [(ADDR_WIDTH-1):0] addr,
                input we, clk,
                output [(DATA_WIDTH-1):0] q
        );
```

```
        // Declare the RAM variable
        reg [DATA_WIDTH-1:0] ram[DEPTH-1:0];

        // Variable to hold the registered read address
        reg [ADDR_WIDTH-1:0] addr_reg;

        always @ (posedge clk)
        begin
               // Write
               if (we)
                      ram[addr] <= data;

               addr_reg <= addr;
        end

        // Continuous assignment implies read returns NEW data.
        // This is the natural behavior of the TriMatrix memory
        // blocks in Single Port mode.
        assign q = ram[addr_reg];

endmodule
```

TIMER.sv

```
Unset
	`timescale 1ns / 1ps
	////////////////////////////////////////////////////////////////////////////////
////////
	// Timier for the Gameboy
//
	// Based On http://gbdev.gg8.se/wiki/articles/Timer_Obscure_Behaviour
//
	////////////////////////////////////////////////////////////////////////////////
////////



	module TIMER
	(
	    input logic clk,
	    input logic rst,

	    input logic [15:0] ADDR,
	    input logic WR,
	    input logic RD,
	    input logic [7:0] MMIO_DATA_out,
	    output logic [7:0] MMIO_DATA_in,

	    output logic IRQ_TIMER
	);

	logic [7:0] DIV, TIMA, TMA, TAC;
```

```
logic [15:0] BIG_COUNTER, BIG_COUNTER_NEXT;
logic FALL_EDGE_TIMER_CLK;
logic TIMER_CLK_PREV, TIMER_CLK_PREV_NEXT, TIMER_CLK_NOW;
logic TIMER_OVERFLOW, TIMER_OVERFLOW_NEXT;
logic [1:0] TIMER_OVERFLOW_CNT, TIMER_OVERFLOW_CNT_NEXT;

logic [7:0] FF04;
assign FF04 = DIV;
assign DIV = BIG_COUNTER[15:8];

logic [7:0] FF05, FF05_NEXT;
assign TIMA = FF05;

logic [7:0] FF06, FF06_NEXT;
assign TMA = FF06;

logic [7:0] FF07, FF07_NEXT;
logic [7:0] TAC_PREV, TAC_PREV_NEXT, TAC_NEXT;
assign TAC = FF07;
assign TAC_NEXT = FF07_NEXT;

/* Main State Machine */
always_ff @(posedge clk)
begin
    if (rst)
    begin
```

```
            BIG_COUNTER <= 0;
            FF05 <= 0;
            FF06 <= 0;
            FF07 <= 8'hF8;
            TIMER_CLK_PREV <= 0;
            TIMER_OVERFLOW <= 0;
            TIMER_OVERFLOW_CNT <= 0;
            TAC_PREV <= 0;
        end
        else
        begin
            BIG_COUNTER <= BIG_COUNTER_NEXT;
            FF05 <= FF05_NEXT;
            FF06 <= FF06_NEXT;
            FF07 <= FF07_NEXT;
            TIMER_CLK_PREV <= TIMER_CLK_PREV_NEXT;
            TIMER_OVERFLOW <= TIMER_OVERFLOW_NEXT;
            TIMER_OVERFLOW_CNT <= TIMER_OVERFLOW_CNT_NEXT;
            TAC_PREV <= TAC_PREV_NEXT;
        end
    end

    always_comb
    begin
        FF05_NEXT = FF05;
        FF06_NEXT = FF06;
```

```
        FF07_NEXT = FF07;
        if (WR && (ADDR == 16'hFF07)) FF07_NEXT =  MMIO_DATA_out;
        TIMER_CLK_NOW = 0;
        FALL_EDGE_TIMER_CLK = 0;
        unique case (TAC[1:0])
            2'd0:
            begin
                TIMER_CLK_PREV_NEXT = BIG_COUNTER[9];
                TIMER_CLK_NOW = BIG_COUNTER[9];
            end
            2'd3:
            begin
                TIMER_CLK_PREV_NEXT = BIG_COUNTER[7];
                TIMER_CLK_NOW = BIG_COUNTER[7];
            end
            2'd2:
            begin
                TIMER_CLK_PREV_NEXT = BIG_COUNTER[5];
                TIMER_CLK_NOW = BIG_COUNTER[5];
            end
            2'd1:
            begin
                TIMER_CLK_PREV_NEXT = BIG_COUNTER[3];
                TIMER_CLK_NOW = BIG_COUNTER[3];
            end
        endcase
```

```
            TAC_PREV_NEXT = TAC;
            FALL_EDGE_TIMER_CLK = (TIMER_CLK_PREV && !TIMER_CLK_NOW && TAC[2]) ||
(TIMER_CLK_PREV && !TAC_NEXT[2] && TAC[2]);
            //FALL_EDGE_TIMER_CLK = (TIMER_CLK_PREV && !TIMER_CLK_NOW && TAC[2])
|| (TIMER_CLK_PREV && !TAC[2] && TAC_PREV[2]);
            //FALL_EDGE_TIMER_CLK = (!TIMER_CLK_PREV && TIMER_CLK_NOW && TAC[2])
|| (!TIMER_CLK_PREV && !TAC[2] && TAC_PREV[2]);

            TIMER_OVERFLOW_NEXT = TIMER_OVERFLOW;
            TIMER_OVERFLOW_CNT_NEXT = TIMER_OVERFLOW_CNT;
            IRQ_TIMER = 0;
            if (FALL_EDGE_TIMER_CLK)
            begin
                FF05_NEXT = FF05 + 1; // increase TIMA when there is a falling
edge of Timer clock
                if (FF05 == 8'hFF)
                begin
                    TIMER_OVERFLOW_NEXT = 1;
                end
            end
            if (TIMER_OVERFLOW) TIMER_OVERFLOW_CNT_NEXT = TIMER_OVERFLOW_CNT + 1;
            if (TIMER_OVERFLOW_CNT == 2'b11)
                TIMER_OVERFLOW_NEXT = 0;
```

```
            BIG_COUNTER_NEXT = (WR && (ADDR == 16'hFF04)) ? 1 : BIG_COUNTER + 1;
// Reset big counter if write into FF04
            if (WR && (ADDR == 16'hFF05)) FF05_NEXT =  (TIMER_OVERFLOW_CNT ==
2'b11) ? FF05 : MMIO_DATA_out;  // Latch behavior
            if (WR && (ADDR == 16'hFF06))
            begin
                FF06_NEXT =  MMIO_DATA_out;
                if (TIMER_OVERFLOW_CNT == 2'b11) // Latch behavior
                begin
                    FF05_NEXT = MMIO_DATA_out;
                end
            end

            case (ADDR)
                16'hFF04: MMIO_DATA_in = FF04;
                16'hFF05: MMIO_DATA_in = FALL_EDGE_TIMER_CLK ? FF05_NEXT : FF05;
// Since the original Timer is Latch based, increase happens at the same clock
cycle
                16'hFF06: MMIO_DATA_in = FF06;
                16'hFF07: MMIO_DATA_in = {5'b11111, FF07[2:0]};
                default : MMIO_DATA_in = 8'hFF;
            endcase

            if (FALL_EDGE_TIMER_CLK) // When TIMA is about to overflow but
writting something to it
            begin
```

```
                if (FF05 == 8'hFF && FF05_NEXT != 8'h00) TIMER_OVERFLOW_NEXT = 0;
            end
            if (TIMER_OVERFLOW_CNT == 2'b10) FF05_NEXT = FF06_NEXT; // count 3T
after overflow
            if (TIMER_OVERFLOW && TIMER_OVERFLOW_CNT == 2'b00) IRQ_TIMER = 1; //
INTQ to CPU is delayed by 2T from overflow (Anywhere from 1T-4T is acceptable?)
        end


        endmodule
```

TestBench.sv

```
Unset
        `timescale 1ns / 1ns



        module TestBench();


        logic clk, clk2;
        logic rst;


        logic [15:0] MEM_ADDR;
        logic [7:0] DATA_in [0:31];
        logic [7:0] DATA_out;
```

```
        logic RD; logic WR; logic HALT; logic INTQ;
        logic [7:0] DATA_brom;
        logic [14:0] LCD_ADDR;
        logic [1:0] LCD_PIXEL;
        logic [7:0] IN_BCD, OUT_BCD;
        logic [7:0] joypad;

        logic [1:0] AAAA [1:0];
        logic B1, B2, B3, B4;
        assign B1 = AAAA[0][0];
        assign B2 = AAAA[1][0];
        assign B3 = AAAA[0][1];
        assign B4 = AAAA[1][1];

       //GB_Z80_SINGLE GB_Z80_CPU(.clk(clk), .rst(rst), .ADDR(MEM_ADDR),
.DATA_in(DATA_brom), .DATA_out(DATA_out), .*);

        initial begin

            AAAA[0] <= 2'b11;
            AAAA[1] <= 2'b00;
            //$readmemh("dmg_boot.mem", boot_rom, 0, 255);
        //    for (int i = 0; i < 32; i++)
        //        DATA_in [i] <= 8'h00;
        //    DATA_in[1] <= 8'h01;
        //    DATA_in[2] <= 8'h12;
```

```
//    DATA_in[3] <= 8'h34;
//    DATA_in[4] <= 8'h03;
//    DATA_in[5] <= 8'h04;
//    DATA_in[6] <= 8'h05;
//    DATA_in[7] <= 8'h06;
//    DATA_in[8] <= 8'hAB;
//    DATA_in[9] <= 8'h07;
//    DATA_in[10] <= 8'h08;
//    DATA_in[13] <= 8'h09;
//    DATA_in[14] <= 8'h02;
//    DATA_in[15] <= 8'h01;
//    DATA_in[16] <= 8'h01;
//    DATA_in[17] <= 8'h00;
//    DATA_in[18] <= 8'h0A;
//    DATA_in[19] <= 8'h0B;
//    DATA_in[20] <= 8'h0C;
//    DATA_in[21] <= 8'h0D;
//    DATA_in[22] <= 8'h0E;
    IN_BCD <= 8'hFF;
//    DATA_in[23] <= 8'h33;
//    DATA_in[24] <= 8'h0F;
//    DATA_in[25] <= 8'h18;
//    DATA_in[26] <= 8'hFE;
    clk <= 0;
    clk2 <= 0;
    rst <= 1;
```

```
        joypad <= 0;
        #20 rst <= 0;
        //#2665000 $finish; // wait for vertical blank boot rom $0x0064
        //#2711100 $finish;//First Fetch M0
        // #5478400 ADDR 0070 LD C0x13
        //# 5479080 $finish;// 0x0088 SUB
       // LD FE50 to disable rom
        //#235061200 $finish; // 03 test op hl jp C000
       // #238391550 $finish; // 03 test op hl $CB21 POP HL
       //#338391550 $finish;
       //#251578800 $finish; // $C67C LD a16 SP
       //#255246000 $finish; //$DEFB JP Z C67D
         //#259340000 $finish; // RST 0
//      #1138469109 joypad[4] <= 1;
//      #100000000 joypad[4] <= 0;
//      #100000000 joypad[4] <= 1;
//      #100000000 joypad[4] <= 0;
       #1048641959 $finish;
      //#100000000 joypad[4] <= 1;
      //#100000000 joypad[4] <= 0;
      //#2138469109 $finish; // RST 0
end

logic DE1_VGA_CLK;
assign DE1_VGA_CLK = clk;
//brom boot_rom(.addr(MEM_ADDR[7:0]), .clk(~clk), .data(DATA_brom));
```

```
	logic [15:0] CART_ADDR;
	logic [25:0] MBC1_ADDR;
	logic [7:0] CART_DATA;
	logic [7:0] MBC_CART_DATA_in, MBC1_DATA_out;
	logic CART_RD;
	logic CART_WR;
	logic CART_CS;
	logic MBC1_RD;
	logic MBC1_WR;

	logic [7:0] CART_DATA_int;
	logic [7:0] CART_DATA_out;
	logic [7:0] CART_RAM_DATA;
	logic [7:0] CART_ROM_DATA;

	logic [14:0] CART_ADDR_int;
	assign CART_DATA = CART_RD ? CART_DATA_int : 8'hFF;
	//assign CART_ADDR_int = CART_RD ? CART_ADDR : 0;
	MBC1 GB_MBC1( .clk(clk2), .reset(rst), .CART_ADDR(CART_ADDR),
.CART_DATA_in(CART_DATA), .CART_DATA_out(CART_DATA_out),
                    .CART_RD(CART_RD), .CART_WR(CART_WR),
.MBC1_ADDR(MBC1_ADDR), .MBC1_RD(MBC1_RD), .MBC1_WR(MBC1_WR),
                    .MBC1_DATA_in(CART_DATA_int),
.MBC1_DATA_out(MBC1_DATA_out), .NUM_ROM_BANK(8'd32), .NUM_RAM_BANK(8'd4));
```

```
	Tetris_ROM CART(.addr(MBC1_ADDR), .clk(~clk), .data(CART_ROM_DATA));


	ram_128 CART_RAM(.data(MBC1_DATA_out), .we(MBC1_WR), .clk(clk2),
.q(CART_RAM_DATA), .addr(MBC1_ADDR[6:0]));
	assign CART_DATA_int = MBC1_ADDR >= 26'h2000000 ? CART_RAM_DATA :
CART_ROM_DATA;



	Top GameBoy(.*);
	//logic [7:0] VGA_R, VGA_G, VGA_B;
	//vga_ball vga_ball(.clk(clk), .reset(rst), .VGA_R(VGA_R), .VGA_G(VGA_G),
.VGA_B(VGA_B));

	always @(*)
	begin
	    #5 clk <= !clk;
	    #1 clk2 <= !clk2;
	end


	endmodule


	module ram_128
	#(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=7)
	(
		input [(DATA_WIDTH-1):0] data,
		input [(ADDR_WIDTH-1):0] addr,
```

```
        input we, clk,
        output [(DATA_WIDTH-1):0] q
    );

        // Declare the RAM variable
        reg [DATA_WIDTH-1:0] ram[127:0];

        // Variable to hold the registered read address
        reg [ADDR_WIDTH-1:0] addr_reg;

        always @ (posedge clk)
        begin
            // Write
            if (we)
                ram[addr] <= data;

            addr_reg <= addr;
        end

        // Continuous assignment implies read returns NEW data.
        // This is the natural behavior of the TriMatrix memory
        // blocks in Single Port mode.
        assign q = ram[addr_reg];

    endmodule
```

alt_tests.sh

```
Unset
	cp PPU3.sv debug_tests/

	cp tb_ppu.cpp debug_tests/

	cd debug_tests/

	verilator --cc PPU3.sv

	verilator -trace -cc PPU3.sv -exe tb_ppu.cpp -top-module PPU3

	cd obj_dir && make -j -f VPPU3.mk

	./VPPU3
```

brom.sv

```
Unset
	`timescale 1ns / 1ns

	////////////////////////////////////////////////////////////////////////////////
////////

	//This is the internal boot rom for GameBoy

	//http://gbdev.gg8.se/files/roms/bootroms/

	//Convert to mem file by

	//hexdump -e '8/1 "%02x " "\n"' input.bin > output.mem

	////////////////////////////////////////////////////////////////////////////////
////////


	// Quartus Prime Verilog Template

	// Single Port ROM
```

```
module brom
(
    input [7:0] addr,
    input clk,
    output reg [7:0] data
);

    reg [7:0] boot_rom [0:255];

    initial begin
        $readmemh("dmg_boot.mem", boot_rom, 0, 255);
    end

    always @ (posedge clk)
    begin
        data <= boot_rom[addr];
    end

endmodule
```

compile-scripts.sh

```
Unset
#!/usr/bin/bash
```

```
make qsys-clean && make qsys && make rbf;

embedded_command_shell.sh;

make dtb;
```

dmg_boot.mem

```
Unset
31 fe ff af 21 ff 9f 32

cb 7c 20 fb 21 26 ff 0e

11 3e 80 32 e2 0c 3e f3

e2 32 3e 77 77 3e fc e0

47 11 04 01 21 10 80 1a

cd 95 00 cd 96 00 13 7b

fe 34 20 f3 11 d8 00 06

08 1a 13 22 23 05 20 f9

3e 19 ea 10 99 21 2f 99

0e 0c 3d 28 08 32 0d 20

f9 2e 0f 18 f3 67 3e 64

57 e0 42 3e 91 e0 40 04

1e 02 0e 0c f0 44 fe 90

20 fa 0d 20 f7 1d 20 f2

0e 13 24 7c 1e 83 fe 62

28 06 1e c1 fe 64 20 06

7b e2 0c 3e 87 e2 f0 42

90 e0 42 15 20 d2 05 20
```

```
4f 16 20 18 cb 4f 06 04

c5 cb 11 17 c1 cb 11 17

05 20 f5 22 23 22 23 c9

ce ed 66 66 cc 0d 00 0b

03 73 00 83 00 0c 00 0d

00 08 11 1f 88 89 00 0e

dc cc 6e e6 dd dd d9 99

bb bb 67 63 6e 0e ec cc

dd dc 99 9f bb b9 33 3e

3c 42 b9 a5 b9 a5 42 3c

21 04 01 11 a8 00 1a 13

be 20 fe 23 7d fe 34 20

f5 06 19 78 86 23 05 20

fb 86 20 fe 3e 01 e0 50
```

joypad_hw.tcl

```
Unset
# TCL File Generated by Component Editor 18.1

# Tue Jun 04 15:02:56 EDT 2019

# DO NOT MODIFY



#

# GameBoy_Joypad "GameBoy_Joypad" v1.0
```

```
#  2019.06.04.15:02:56
# Gameboy Joypad
#


#
# request TCL package from ACDS 16.1
#
package require -exact qsys 16.1



#
# module GameBoy_Joypad
#
set_module_property DESCRIPTION "Gameboy Joypad"
set_module_property NAME GameBoy_Joypad
set_module_property VERSION 1.0
set_module_property INTERNAL false
set_module_property OPAQUE_ADDRESS_MAP true
set_module_property GROUP GameBoy
set_module_property AUTHOR ""
set_module_property DISPLAY_NAME GameBoy_Joypad
set_module_property INSTANTIATE_IN_SYSTEM_MODULE true
set_module_property EDITABLE true
set_module_property REPORT_TO_TALKBACK false
set_module_property ALLOW_GREYBOX_GENERATION false
set_module_property REPORT_HIERARCHY false
```

```
       #
       # file sets
       #
       add_fileset QUARTUS_SYNTH QUARTUS_SYNTH "" ""
       set_fileset_property QUARTUS_SYNTH TOP_LEVEL GameBoy_Joypad
       set_fileset_property QUARTUS_SYNTH ENABLE_RELATIVE_INCLUDE_PATHS false
       set_fileset_property QUARTUS_SYNTH ENABLE_FILE_OVERWRITE_MODE false
       add_fileset_file GameBoy_Joypad.sv SYSTEM_VERILOG PATH GameBoy_Joypad.sv
TOP_LEVEL_FILE


       #
       # parameters
       #



       #
       # display items
       #



       #
       # connection point clock
       #
```

```
add_interface clock clock end
set_interface_property clock clockRate 0
set_interface_property clock ENABLED true
set_interface_property clock EXPORT_OF ""
set_interface_property clock PORT_NAME_MAP ""
set_interface_property clock CMSIS_SVD_VARIABLES ""
set_interface_property clock SVD_ADDRESS_GROUP ""

add_interface_port clock clk clk Input 1



#
# connection point reset
#
add_interface reset reset end
set_interface_property reset associatedClock clock
set_interface_property reset synchronousEdges DEASSERT
set_interface_property reset ENABLED true
set_interface_property reset EXPORT_OF ""
set_interface_property reset PORT_NAME_MAP ""
set_interface_property reset CMSIS_SVD_VARIABLES ""
set_interface_property reset SVD_ADDRESS_GROUP ""

add_interface_port reset reset reset Input 1
```

```
#
# connection point GameBoy_JoyPad
#
add_interface GameBoy_JoyPad conduit end
set_interface_property GameBoy_JoyPad associatedClock ""
set_interface_property GameBoy_JoyPad associatedReset ""
set_interface_property GameBoy_JoyPad ENABLED true
set_interface_property GameBoy_JoyPad EXPORT_OF ""
set_interface_property GameBoy_JoyPad PORT_NAME_MAP ""
set_interface_property GameBoy_JoyPad CMSIS_SVD_VARIABLES ""
set_interface_property GameBoy_JoyPad SVD_ADDRESS_GROUP ""

add_interface_port GameBoy_JoyPad P10 p10 Output 1
add_interface_port GameBoy_JoyPad P11 p11 Output 1
add_interface_port GameBoy_JoyPad P12 p12 Output 1
add_interface_port GameBoy_JoyPad P13 p13 Output 1
add_interface_port GameBoy_JoyPad P14 p14 Input 1
add_interface_port GameBoy_JoyPad P15 p15 Input 1



#
# connection point avalon_slave
#
add_interface avalon_slave avalon end
set_interface_property avalon_slave addressUnits WORDS
set_interface_property avalon_slave associatedClock clock
```

```
        set_interface_property avalon_slave associatedReset reset
        set_interface_property avalon_slave bitsPerSymbol 8
        set_interface_property avalon_slave burstOnBurstBoundariesOnly false
        set_interface_property avalon_slave burstcountUnits WORDS
        set_interface_property avalon_slave explicitAddressSpan 0
        set_interface_property avalon_slave holdTime 0
        set_interface_property avalon_slave linewrapBursts false
        set_interface_property avalon_slave maximumPendingReadTransactions 0
        set_interface_property avalon_slave maximumPendingWriteTransactions 0
        set_interface_property avalon_slave readLatency 0
        set_interface_property avalon_slave readWaitTime 1
        set_interface_property avalon_slave setupTime 0
        set_interface_property avalon_slave timingUnits Cycles
        set_interface_property avalon_slave writeWaitTime 0
        set_interface_property avalon_slave ENABLED true
        set_interface_property avalon_slave EXPORT_OF ""
        set_interface_property avalon_slave PORT_NAME_MAP ""
        set_interface_property avalon_slave CMSIS_SVD_VARIABLES ""
        set_interface_property avalon_slave SVD_ADDRESS_GROUP ""

        add_interface_port avalon_slave chipselect_slv chipselect Input 1
        add_interface_port avalon_slave write_slv write Input 1
        add_interface_port avalon_slave writedata_slv writedata Input 8
        set_interface_assignment avalon_slave embeddedsw.configuration.isFlash 0
        set_interface_assignment avalon_slave
embeddedsw.configuration.isMemoryDevice 0
```

```
       set_interface_assignment avalon_slave
embeddedsw.configuration.isNonVolatileStorage 0
       set_interface_assignment avalon_slave
embeddedsw.configuration.isPrintableDevice 0
       set_module_assignment embeddedsw.dts.vendor "csee4840"
       set_module_assignment embeddedsw.dts.name "joypad"
       set_module_assignment embeddedsw.dts.group "game_boy"
```

oam_search_test.sh

```
Unset
       cp PPU3.sv debug_tests/
       cp PPU3_OAM_SEARCH.cpp debug_tests/
       cd debug_tests/
       verilator --cc PPU3.sv
       verilator -trace -cc PPU3.sv -exe PPU3_OAM_SEARCH.cpp -top-module PPU3
       cd obj_dir && make -j -f VPPU3.mk
       ./VPPU3
       gtkwave ppu3.vcd
```

soc_system.qsys

```
Unset
       <?xml version="1.0" encoding="UTF-8"?>
       <system name="$${FILENAME}">
```

```
 <component
   name="$${FILENAME}"
   displayName="$${FILENAME}"
   version="1.0"
   description=""
   tags=""
   categories="System" />
 <parameter name="bonusData"><![CDATA[bonusData
{
   element AV_Config
   {
      datum _sortIndex
      {
         value = "8";
         type = "int";
      }
      datum sopceditor_expanded
      {
         value = "1";
         type = "boolean";
      }
   }
   element AV_Config.avalon_av_config_slave
   {
      datum baseAddress
      {
```

```
            value = "69218304";
            type = "String";
         }
      }
      element GameBoy
      {
         datum _sortIndex
         {
            value = "3";
            type = "int";
         }
      }
      element GameBoy_Cartridge
      {
         datum _sortIndex
         {
            value = "5";
            type = "int";
         }
         datum sopceditor_expanded
         {
            value = "1";
            type = "boolean";
         }
      }
      element GameBoy_Cartridge.hps_slave
```

```
        {
           datum _lockedAddress
           {
              value = "1";
              type = "boolean";
           }
        }
        element GameBoy_Joypad
        {
           datum _sortIndex
           {
              value = "2";
              type = "int";
           }
           datum sopceditor_expanded
           {
              value = "1";
              type = "boolean";
           }
        }
        element GameBoy_Reset
        {
           datum _sortIndex
           {
              value = "4";
              type = "int";
```

```
      }
   }
   element GameBoy_VGA
   {
      datum _sortIndex
      {
         value = "1";
         type = "int";
      }
      datum sopceditor_expanded
      {
         value = "1";
         type = "boolean";
      }
   }
   element GameBoy_VGA.avalon_slave
   {
      datum _lockedAddress
      {
         value = "1";
         type = "boolean";
      }
      datum baseAddress
      {
         value = "67108864";
         type = "String";
```

```
      }
   }
   element Main_PLL
   {
      datum _sortIndex
      {
         value = "6";
         type = "int";
      }
   }
   element SOC_System_RST_Bridge
   {
      datum _sortIndex
      {
         value = "7";
         type = "int";
      }
      datum sopceditor_expanded
      {
         value = "1";
         type = "boolean";
      }
   }
   element hps_0
   {
      datum _sortIndex
```

```
      {
         value = "0";
         type = "int";
      }
   }
   element onchip_memory2_0
   {
      datum _sortIndex
      {
         value = "9";
         type = "int";
      }
   }
   element soc_system
   {
      datum _originalDeviceFamily
      {
         value = "Cyclone V";
         type = "String";
      }
   }
   element soc_system
   {
      datum _originalDeviceFamily
      {
         value = "Cyclone V";
```

```
      type = "String";
   }
}
element soc_system
{
   datum _originalDeviceFamily
   {
      value = "Cyclone V";
      type = "String";
   }
}
element soc_system
{
   datum _originalDeviceFamily
   {
      value = "Cyclone V";
      type = "String";
   }
}
element soc_system
{
   datum _originalDeviceFamily
   {
      value = "Cyclone V";
      type = "String";
   }
```

```
        }
        element soc_system
        {
           datum _originalDeviceFamily
           {
              value = "Cyclone V";
              type = "String";
           }
        }
        element soc_system
        {
           datum _originalDeviceFamily
           {
              value = "Cyclone V";
              type = "String";
           }
        }
        element soc_system
        {
           datum _originalDeviceFamily
           {
              value = "Cyclone V";
              type = "String";
           }
        }
        element soc_system
```

```
        {
           datum _originalDeviceFamily
           {
              value = "Cyclone V";
              type = "String";
           }
        }
        element soc_system
        {
           datum _originalDeviceFamily
           {
              value = "Cyclone V";
              type = "String";
           }
        }
        element soc_system
        {
           datum _originalDeviceFamily
           {
              value = "Cyclone V";
              type = "String";
           }
        }
        element soc_system
        {
           datum _originalDeviceFamily
```

```
      {
         value = "Cyclone V";
         type = "String";
      }
   }
   element soc_system
   {
      datum _originalDeviceFamily
      {
         value = "Cyclone V";
         type = "String";
      }
   }
   element soc_system
   {
      datum _originalDeviceFamily
      {
         value = "Cyclone V";
         type = "String";
      }
   }
   element soc_system
   {
      datum _originalDeviceFamily
      {
         value = "Cyclone V";
```

```
      type = "String";
   }
}
element soc_system
{
   datum _originalDeviceFamily
   {
      value = "Cyclone V";
      type = "String";
   }
}
element soc_system
{
   datum _originalDeviceFamily
   {
      value = "Cyclone V";
      type = "String";
   }
}
element soc_system
{
   datum _originalDeviceFamily
   {
      value = "Cyclone V";
      type = "String";
   }
```

```
        }
        element soc_system
        {
           datum _originalDeviceFamily
           {
              value = "Cyclone V";
              type = "String";
           }
        }
        element soc_system
        {
           datum _originalDeviceFamily
           {
              value = "Cyclone V";
              type = "String";
           }
        }
        element soc_system
        {
           datum _originalDeviceFamily
           {
              value = "Cyclone V";
              type = "String";
           }
        }
        element soc_system
```

```
      {
         datum _originalDeviceFamily
         {
            value = "Cyclone V";
            type = "String";
         }
      }
      element soc_system
      {
         datum _originalDeviceFamily
         {
            value = "Cyclone V";
            type = "String";
         }
      }
   }
   ]]></parameter>
    <parameter name="clockCrossingAdapter" value="HANDSHAKE" />
    <parameter name="device" value="5CSEMA5F31C6" />
    <parameter name="deviceFamily" value="Cyclone V" />
    <parameter name="deviceSpeedGrade" value="6" />
    <parameter name="fabricMode" value="QSYS" />
    <parameter name="generateLegacySim" value="false" />
    <parameter name="generationId" value="0" />
    <parameter name="globalResetBus" value="false" />
    <parameter name="hdlLanguage" value="VERILOG" />
```

```
        <parameter name="hideFromIPCatalog" value="false" />
        <parameter name="lockedInterfaceDefinition" value="" />
        <parameter name="maxAdditionalLatency" value="1" />
        <parameter name="projectName" value="soc_system.qpf" />
        <parameter name="sopcBorderPoints" value="false" />
        <parameter name="systemHash" value="0" />
        <parameter name="testBenchDutName" value="" />
        <parameter name="timeStamp" value="0" />
        <parameter name="useTestBenchNamingPattern" value="false" />
        <instanceScript></instanceScript>
        <interface
          name="av_config"
          internal="AV_Config.external_interface"
          type="conduit"
          dir="end" />
        <interface name="clk" internal="Main_PLL.refclk" type="clock" dir="end"
/>
        <interface name="gameboy_reset" internal="GameBoy.reset" type="reset"
dir="end" />
        <interface name="hps" internal="hps_0.hps_io" type="conduit" dir="end"
/>
        <interface name="hps_ddr3" internal="hps_0.memory" type="conduit"
dir="end" />
        <interface
          name="ledr"
          internal="GameBoy_Cartridge.LEDR"
```

```
          type="conduit"
          dir="end" />
        <interface
          name="reset"
          internal="SOC_System_RST_Bridge.in_reset"
          type="reset"
          dir="end" />
        <interface name="sdram_clk" internal="Main_PLL.outclk0" type="clock"
dir="start" />
        <interface name="vga" internal="GameBoy_VGA.VGA" type="conduit"
dir="end" />
        <module
          name="AV_Config"
          kind="altera_up_avalon_audio_and_video_config"
          version="18.0"
          enabled="1">
         <parameter name="AUTO_CLK_CLOCK_RATE" value="108406625" />
         <parameter name="AUTO_DEVICE_FAMILY" value="Cyclone V" />
         <parameter name="audio_in" value="Line In to ADC" />
         <parameter name="bit_length" value="16" />
         <parameter name="board" value="DE1-SoC" />
         <parameter name="d5m_resolution" value="2592 x 1944" />
         <parameter name="dac_enable" value="true" />
         <parameter name="data_format" value="Left Justified" />
         <parameter name="device">On-Board Peripherals</parameter>
         <parameter name="eai" value="true" />
```

```
          <parameter name="exposure" value="false" />
          <parameter name="line_in_bypass" value="false" />
          <parameter name="mic_attenuation" value="-6dB" />
          <parameter name="mic_bypass" value="false" />
          <parameter name="sampling_rate" value="48 kHz" />
          <parameter name="video_format" value="NTSC" />
        </module>
        <module name="GameBoy" kind="GameBoy" version="1.0" enabled="1" />
        <module
          name="GameBoy_Cartridge"
          kind="GameBoy_Cartridge"
          version="1.0"
          enabled="1" />
        <module name="GameBoy_Joypad" kind="GameBoy_Joypad" version="1.0"
enabled="1" />
        <module
          name="GameBoy_Reset"
          kind="altera_reset_bridge"
          version="21.1"
          enabled="1">
          <parameter name="ACTIVE_LOW_RESET" value="0" />
          <parameter name="AUTO_CLK_CLOCK_RATE" value="67108863" />
          <parameter name="NUM_RESET_OUTPUTS" value="1" />
          <parameter name="SYNCHRONOUS_EDGES" value="both" />
          <parameter name="USE_RESET_REQUEST" value="0" />
        </module>
```

```
        <module name="GameBoy_VGA" kind="GameBoy_VGA" version="1.0" enabled="1"
/>
        <module name="Main_PLL" kind="altera_pll" version="21.1" enabled="1">
         <parameter name="debug_print_output" value="false" />
         <parameter name="debug_use_rbc_taf_method" value="false" />
         <parameter name="device" value="5CSEMA5F31C6" />
         <parameter name="device_family" value="Cyclone V" />
         <parameter name="gui_active_clk" value="false" />
         <parameter name="gui_actual_output_clock_frequency0" value="0 MHz" />
         <parameter name="gui_actual_output_clock_frequency1" value="0 MHz" />
         <parameter name="gui_actual_output_clock_frequency10" value="0 MHz" />
         <parameter name="gui_actual_output_clock_frequency11" value="0 MHz" />
         <parameter name="gui_actual_output_clock_frequency12" value="0 MHz" />
         <parameter name="gui_actual_output_clock_frequency13" value="0 MHz" />
         <parameter name="gui_actual_output_clock_frequency14" value="0 MHz" />
         <parameter name="gui_actual_output_clock_frequency15" value="0 MHz" />
         <parameter name="gui_actual_output_clock_frequency16" value="0 MHz" />
         <parameter name="gui_actual_output_clock_frequency17" value="0 MHz" />
         <parameter name="gui_actual_output_clock_frequency2" value="25.303342
MHz" />
         <parameter name="gui_actual_output_clock_frequency3" value="0 MHz" />
         <parameter name="gui_actual_output_clock_frequency4" value="0 MHz" />
         <parameter name="gui_actual_output_clock_frequency5" value="0 MHz" />
         <parameter name="gui_actual_output_clock_frequency6" value="0 MHz" />
         <parameter name="gui_actual_output_clock_frequency7" value="0 MHz" />
         <parameter name="gui_actual_output_clock_frequency8" value="0 MHz" />
```

```
<parameter name="gui_actual_output_clock_frequency9" value="0 MHz" />
<parameter name="gui_actual_phase_shift0" value="0" />
<parameter name="gui_actual_phase_shift1" value="0" />
<parameter name="gui_actual_phase_shift10" value="0" />
<parameter name="gui_actual_phase_shift11" value="0" />
<parameter name="gui_actual_phase_shift12" value="0" />
<parameter name="gui_actual_phase_shift13" value="0" />
<parameter name="gui_actual_phase_shift14" value="0" />
<parameter name="gui_actual_phase_shift15" value="0" />
<parameter name="gui_actual_phase_shift16" value="0" />
<parameter name="gui_actual_phase_shift17" value="0" />
<parameter name="gui_actual_phase_shift2" value="0" />
<parameter name="gui_actual_phase_shift3" value="0" />
<parameter name="gui_actual_phase_shift4" value="0" />
<parameter name="gui_actual_phase_shift5" value="0" />
<parameter name="gui_actual_phase_shift6" value="0" />
<parameter name="gui_actual_phase_shift7" value="0" />
<parameter name="gui_actual_phase_shift8" value="0" />
<parameter name="gui_actual_phase_shift9" value="0" />
<parameter name="gui_cascade_counter0" value="false" />
<parameter name="gui_cascade_counter1" value="false" />
<parameter name="gui_cascade_counter10" value="false" />
<parameter name="gui_cascade_counter11" value="false" />
<parameter name="gui_cascade_counter12" value="false" />
<parameter name="gui_cascade_counter13" value="false" />
<parameter name="gui_cascade_counter14" value="false" />
```

```
<parameter name="gui_cascade_counter15" value="false" />
<parameter name="gui_cascade_counter16" value="false" />
<parameter name="gui_cascade_counter17" value="false" />
<parameter name="gui_cascade_counter2" value="false" />
<parameter name="gui_cascade_counter3" value="false" />
<parameter name="gui_cascade_counter4" value="false" />
<parameter name="gui_cascade_counter5" value="false" />
<parameter name="gui_cascade_counter6" value="false" />
<parameter name="gui_cascade_counter7" value="false" />
<parameter name="gui_cascade_counter8" value="false" />
<parameter name="gui_cascade_counter9" value="false" />
<parameter name="gui_cascade_outclk_index" value="0" />
<parameter name="gui_channel_spacing" value="0.0" />
<parameter name="gui_clk_bad" value="false" />
<parameter name="gui_device_speed_grade" value="2" />
<parameter name="gui_divide_factor_c0" value="1" />
<parameter name="gui_divide_factor_c1" value="1" />
<parameter name="gui_divide_factor_c10" value="1" />
<parameter name="gui_divide_factor_c11" value="1" />
<parameter name="gui_divide_factor_c12" value="1" />
<parameter name="gui_divide_factor_c13" value="1" />
<parameter name="gui_divide_factor_c14" value="1" />
<parameter name="gui_divide_factor_c15" value="1" />
<parameter name="gui_divide_factor_c16" value="1" />
<parameter name="gui_divide_factor_c17" value="1" />
<parameter name="gui_divide_factor_c2" value="1" />
```

```
<parameter name="gui_divide_factor_c3" value="1" />
<parameter name="gui_divide_factor_c4" value="1" />
<parameter name="gui_divide_factor_c5" value="1" />
<parameter name="gui_divide_factor_c6" value="1" />
<parameter name="gui_divide_factor_c7" value="1" />
<parameter name="gui_divide_factor_c8" value="1" />
<parameter name="gui_divide_factor_c9" value="1" />
<parameter name="gui_divide_factor_n" value="1" />
<parameter name="gui_dps_cntr" value="C0" />
<parameter name="gui_dps_dir" value="Positive" />
<parameter name="gui_dps_num" value="1" />
<parameter name="gui_dsm_out_sel" value="1st_order" />
<parameter name="gui_duty_cycle0" value="50" />
<parameter name="gui_duty_cycle1" value="50" />
<parameter name="gui_duty_cycle10" value="50" />
<parameter name="gui_duty_cycle11" value="50" />
<parameter name="gui_duty_cycle12" value="50" />
<parameter name="gui_duty_cycle13" value="50" />
<parameter name="gui_duty_cycle14" value="50" />
<parameter name="gui_duty_cycle15" value="50" />
<parameter name="gui_duty_cycle16" value="50" />
<parameter name="gui_duty_cycle17" value="50" />
<parameter name="gui_duty_cycle2" value="50" />
<parameter name="gui_duty_cycle3" value="50" />
<parameter name="gui_duty_cycle4" value="50" />
<parameter name="gui_duty_cycle5" value="50" />
```

```
<parameter name="gui_duty_cycle6" value="50" />
<parameter name="gui_duty_cycle7" value="50" />
<parameter name="gui_duty_cycle8" value="50" />
<parameter name="gui_duty_cycle9" value="50" />
<parameter name="gui_en_adv_params" value="false" />
<parameter name="gui_en_dps_ports" value="false" />
<parameter name="gui_en_phout_ports" value="false" />
<parameter name="gui_en_reconf" value="false" />
<parameter name="gui_enable_cascade_in" value="false" />
<parameter name="gui_enable_cascade_out" value="false" />
<parameter name="gui_enable_mif_dps" value="false" />
<parameter name="gui_feedback_clock" value="Global Clock" />
<parameter name="gui_frac_multiply_factor" value="1" />
<parameter name="gui_fractional_cout" value="32" />
<parameter name="gui_mif_generate" value="false" />
<parameter name="gui_multiply_factor" value="1" />
<parameter name="gui_number_of_clocks" value="5" />
<parameter name="gui_operation_mode" value="direct" />
<parameter name="gui_output_clock_frequency0" value="67.108864" />
<parameter name="gui_output_clock_frequency1" value="67.108864" />
<parameter name="gui_output_clock_frequency10" value="100.0" />
<parameter name="gui_output_clock_frequency11" value="100.0" />
<parameter name="gui_output_clock_frequency12" value="100.0" />
<parameter name="gui_output_clock_frequency13" value="100.0" />
<parameter name="gui_output_clock_frequency14" value="100.0" />
<parameter name="gui_output_clock_frequency15" value="100.0" />
```

```
<parameter name="gui_output_clock_frequency16" value="100.0" />
<parameter name="gui_output_clock_frequency17" value="100.0" />
<parameter name="gui_output_clock_frequency2" value="108.0" />
<parameter name="gui_output_clock_frequency3" value="12.288" />
<parameter name="gui_output_clock_frequency4" value="12.288" />
<parameter name="gui_output_clock_frequency5" value="100.0" />
<parameter name="gui_output_clock_frequency6" value="100.0" />
<parameter name="gui_output_clock_frequency7" value="100.0" />
<parameter name="gui_output_clock_frequency8" value="100.0" />
<parameter name="gui_output_clock_frequency9" value="100.0" />
<parameter name="gui_phase_shift0" value="-3000" />
<parameter name="gui_phase_shift1" value="0" />
<parameter name="gui_phase_shift10" value="0" />
<parameter name="gui_phase_shift11" value="0" />
<parameter name="gui_phase_shift12" value="0" />
<parameter name="gui_phase_shift13" value="0" />
<parameter name="gui_phase_shift14" value="0" />
<parameter name="gui_phase_shift15" value="0" />
<parameter name="gui_phase_shift16" value="0" />
<parameter name="gui_phase_shift17" value="0" />
<parameter name="gui_phase_shift2" value="0" />
<parameter name="gui_phase_shift3" value="0" />
<parameter name="gui_phase_shift4" value="0" />
<parameter name="gui_phase_shift5" value="0" />
<parameter name="gui_phase_shift6" value="0" />
<parameter name="gui_phase_shift7" value="0" />
```

```
        <parameter name="gui_phase_shift8" value="0" />
        <parameter name="gui_phase_shift9" value="0" />
        <parameter name="gui_phase_shift_deg0" value="0.0" />
        <parameter name="gui_phase_shift_deg1" value="0.0" />
        <parameter name="gui_phase_shift_deg10" value="0.0" />
        <parameter name="gui_phase_shift_deg11" value="0.0" />
        <parameter name="gui_phase_shift_deg12" value="0.0" />
        <parameter name="gui_phase_shift_deg13" value="0.0" />
        <parameter name="gui_phase_shift_deg14" value="0.0" />
        <parameter name="gui_phase_shift_deg15" value="0.0" />
        <parameter name="gui_phase_shift_deg16" value="0.0" />
        <parameter name="gui_phase_shift_deg17" value="0.0" />
        <parameter name="gui_phase_shift_deg2" value="0.0" />
        <parameter name="gui_phase_shift_deg3" value="0.0" />
        <parameter name="gui_phase_shift_deg4" value="0.0" />
        <parameter name="gui_phase_shift_deg5" value="0.0" />
        <parameter name="gui_phase_shift_deg6" value="0.0" />
        <parameter name="gui_phase_shift_deg7" value="0.0" />
        <parameter name="gui_phase_shift_deg8" value="0.0" />
        <parameter name="gui_phase_shift_deg9" value="0.0" />
        <parameter name="gui_phout_division" value="1" />
        <parameter name="gui_pll_auto_reset" value="Off" />
        <parameter name="gui_pll_bandwidth_preset" value="Auto" />
        <parameter name="gui_pll_cascading_mode">Create an adjpllin signal to
connect with an upstream PLL</parameter>
        <parameter name="gui_pll_mode" value="Fractional-N PLL" />
```

```
 <parameter name="gui_ps_units0" value="ps" />
 <parameter name="gui_ps_units1" value="ps" />
 <parameter name="gui_ps_units10" value="ps" />
 <parameter name="gui_ps_units11" value="ps" />
 <parameter name="gui_ps_units12" value="ps" />
 <parameter name="gui_ps_units13" value="ps" />
 <parameter name="gui_ps_units14" value="ps" />
 <parameter name="gui_ps_units15" value="ps" />
 <parameter name="gui_ps_units16" value="ps" />
 <parameter name="gui_ps_units17" value="ps" />
 <parameter name="gui_ps_units2" value="ps" />
 <parameter name="gui_ps_units3" value="ps" />
 <parameter name="gui_ps_units4" value="ps" />
 <parameter name="gui_ps_units5" value="ps" />
 <parameter name="gui_ps_units6" value="ps" />
 <parameter name="gui_ps_units7" value="ps" />
 <parameter name="gui_ps_units8" value="ps" />
 <parameter name="gui_ps_units9" value="ps" />
 <parameter name="gui_refclk1_frequency" value="100.0" />
 <parameter name="gui_refclk_switch" value="false" />
 <parameter name="gui_reference_clock_frequency" value="50.0" />
 <parameter name="gui_switchover_delay" value="0" />
 <parameter name="gui_switchover_mode">Automatic Switchover</parameter>
 <parameter name="gui_use_locked" value="false" />
</module>
<module
```

```
    name="SOC_System_RST_Bridge"
    kind="altera_reset_bridge"
    version="21.1"
    enabled="1">
   <parameter name="ACTIVE_LOW_RESET" value="0" />
   <parameter name="AUTO_CLK_CLOCK_RATE" value="67108863" />
   <parameter name="NUM_RESET_OUTPUTS" value="1" />
   <parameter name="SYNCHRONOUS_EDGES" value="both" />
   <parameter name="USE_RESET_REQUEST" value="0" />
  </module>
  <module name="hps_0" kind="altera_hps" version="21.1" enabled="1">
   <parameter name="ABSTRACT_REAL_COMPARE_TEST" value="false" />
   <parameter name="ABS_RAM_MEM_INIT_FILENAME" value="meminit" />
   <parameter name="ACV_PHY_CLK_ADD_FR_PHASE" value="0.0" />
   <parameter name="AC_PACKAGE_DESKEW" value="false" />
   <parameter name="AC_ROM_USER_ADD_0" value="0_0000_0000_0000" />
   <parameter name="AC_ROM_USER_ADD_1" value="0_0000_0000_1000" />
   <parameter name="ADDR_ORDER" value="0" />
   <parameter name="ADD_EFFICIENCY_MONITOR" value="false" />
   <parameter name="ADD_EXTERNAL_SEQ_DEBUG_NIOS" value="false" />
   <parameter name="ADVANCED_CK_PHASES" value="false" />
   <parameter name="ADVERTIZE_SEQUENCER_SW_BUILD_FILES" value="false" />
   <parameter name="AFI_DEBUG_INFO_WIDTH" value="32" />
   <parameter name="ALTMEMPHY_COMPATIBLE_MODE" value="false" />
   <parameter name="AP_MODE" value="false" />
   <parameter name="AP_MODE_EN" value="0" />
```

```
        <parameter name="AUTO_DEVICE_SPEEDGRADE" value="6" />
        <parameter name="AUTO_PD_CYCLES" value="0" />
        <parameter name="AUTO_POWERDN_EN" value="false" />
        <parameter name="AVL_DATA_WIDTH_PORT" value="32,32,32,32,32,32" />
        <parameter name="AVL_MAX_SIZE" value="4" />
        <parameter name="BONDING_OUT_ENABLED" value="false" />
        <parameter name="BOOTFROMFPGA_Enable" value="false" />
        <parameter name="BSEL" value="1" />
        <parameter name="BSEL_EN" value="false" />
        <parameter name="BYTE_ENABLE" value="true" />
        <parameter name="C2P_WRITE_CLOCK_ADD_PHASE" value="0.0" />
        <parameter name="CALIBRATION_MODE" value="Skip" />
        <parameter name="CALIB_REG_WIDTH" value="8" />
        <parameter name="CAN0_Mode" value="N/A" />
        <parameter name="CAN0_PinMuxing" value="Unused" />
        <parameter name="CAN1_Mode" value="N/A" />
        <parameter name="CAN1_PinMuxing" value="Unused" />
        <parameter name="CFG_DATA_REORDERING_TYPE" value="INTER_BANK" />
        <parameter name="CFG_REORDER_DATA" value="true" />
        <parameter name="CFG_TCCD_NS" value="2.5" />
        <parameter name="COMMAND_PHASE" value="0.0" />
        <parameter name="CONTROLLER_LATENCY" value="5" />
        <parameter name="CORE_DEBUG_CONNECTION" value="EXPORT" />
        <parameter
name="CPORT_TYPE_PORT">Bidirectional,Bidirectional,Bidirectional,Bidirectional,
Bidirectional,Bidirectional</parameter>
```

```
<parameter name="CSEL" value="0" />
<parameter name="CSEL_EN" value="false" />
<parameter name="CTI_Enable" value="false" />
<parameter name="CTL_AUTOPCH_EN" value="false" />
<parameter name="CTL_CMD_QUEUE_DEPTH" value="8" />
<parameter name="CTL_CSR_CONNECTION" value="INTERNAL_JTAG" />
<parameter name="CTL_CSR_ENABLED" value="false" />
<parameter name="CTL_CSR_READ_ONLY" value="1" />
<parameter name="CTL_DEEP_POWERDN_EN" value="false" />
<parameter name="CTL_DYNAMIC_BANK_ALLOCATION" value="false" />
<parameter name="CTL_DYNAMIC_BANK_NUM" value="4" />
<parameter name="CTL_ECC_AUTO_CORRECTION_ENABLED" value="false" />
<parameter name="CTL_ECC_ENABLED" value="false" />
<parameter name="CTL_ENABLE_BURST_INTERRUPT" value="false" />
<parameter name="CTL_ENABLE_BURST_TERMINATE" value="false" />
<parameter name="CTL_HRB_ENABLED" value="false" />
<parameter name="CTL_LOOK_AHEAD_DEPTH" value="4" />
<parameter name="CTL_SELF_REFRESH_EN" value="false" />
<parameter name="CTL_USR_REFRESH_EN" value="false" />
<parameter name="CTL_ZQCAL_EN" value="false" />
<parameter name="CUT_NEW_FAMILY_TIMING" value="true" />
<parameter name="DAT_DATA_WIDTH" value="32" />
<parameter name="DEBUGAPB_Enable" value="false" />
<parameter name="DEBUG_MODE" value="false" />
<parameter name="DEVICE_DEPTH" value="1" />
<parameter name="DEVICE_FAMILY_PARAM" value="" />
```

```
<parameter name="DISABLE_CHILD_MESSAGING" value="false" />
<parameter name="DISCRETE_FLY_BY" value="true" />
<parameter name="DLL_SHARING_MODE" value="None" />
<parameter name="DMA_Enable">No,No,No,No,No,No,No,No</parameter>
<parameter name="DQS_DQSN_MODE" value="DIFFERENTIAL" />
<parameter name="DQ_INPUT_REG_USE_CLKN" value="false" />
<parameter name="DUPLICATE_AC" value="false" />
<parameter name="ED_EXPORT_SEQ_DEBUG" value="false" />
<parameter name="EMAC0_Mode" value="N/A" />
<parameter name="EMAC0_PTP" value="false" />
<parameter name="EMAC0_PinMuxing" value="Unused" />
<parameter name="EMAC1_Mode" value="RGMII" />
<parameter name="EMAC1_PTP" value="false" />
<parameter name="EMAC1_PinMuxing" value="HPS I/O Set 0" />
<parameter name="ENABLE_ABS_RAM_MEM_INIT" value="false" />
<parameter name="ENABLE_BONDING" value="false" />
<parameter name="ENABLE_BURST_MERGE" value="false" />
<parameter name="ENABLE_CTRL_AVALON_INTERFACE" value="true" />
<parameter name="ENABLE_DELAY_CHAIN_WRITE" value="false" />
<parameter name="ENABLE_EMIT_BFM_MASTER" value="false" />
<parameter name="ENABLE_EXPORT_SEQ_DEBUG_BRIDGE" value="false" />
<parameter name="ENABLE_EXTRA_REPORTING" value="false" />
<parameter name="ENABLE_ISS_PROBES" value="false" />
<parameter name="ENABLE_NON_DESTRUCTIVE_CALIB" value="false" />
<parameter name="ENABLE_NON_DES_CAL" value="false" />
<parameter name="ENABLE_NON_DES_CAL_TEST" value="false" />
```

```
         <parameter name="ENABLE_SEQUENCER_MARGINING_ON_BY_DEFAULT"
value="false" />
         <parameter name="ENABLE_USER_ECC" value="false" />
         <parameter name="EXPORT_AFI_HALF_CLK" value="false" />
         <parameter name="EXTRA_SETTINGS" value="" />
         <parameter name="F2H_AXI_CLOCK_FREQ" value="67108863" />
         <parameter name="F2H_SDRAM0_CLOCK_FREQ" value="100" />
         <parameter name="F2H_SDRAM1_CLOCK_FREQ" value="100" />
         <parameter name="F2H_SDRAM2_CLOCK_FREQ" value="100" />
         <parameter name="F2H_SDRAM3_CLOCK_FREQ" value="100" />
         <parameter name="F2H_SDRAM4_CLOCK_FREQ" value="100" />
         <parameter name="F2H_SDRAM5_CLOCK_FREQ" value="100" />
         <parameter name="F2SCLK_COLDRST_Enable" value="false" />
         <parameter name="F2SCLK_DBGRST_Enable" value="false" />
         <parameter name="F2SCLK_PERIPHCLK_Enable" value="false" />
         <parameter name="F2SCLK_PERIPHCLK_FREQ" value="0" />
         <parameter name="F2SCLK_SDRAMCLK_Enable" value="false" />
         <parameter name="F2SCLK_SDRAMCLK_FREQ" value="0" />
         <parameter name="F2SCLK_WARMRST_Enable" value="false" />
         <parameter name="F2SDRAM_Type" value="" />
         <parameter name="F2SDRAM_Width" value="" />
         <parameter name="F2SINTERRUPT_Enable" value="true" />
         <parameter name="F2S_Width" value="2" />
         <parameter name="FIX_READ_LATENCY" value="8" />
         <parameter name="FORCED_NON_LDC_ADDR_CMD_MEM_CK_INVERT" value="false"
/>
```

```
         <parameter name="FORCED_NUM_WRITE_FR_CYCLE_SHIFTS" value="0" />
         <parameter name="FORCE_DQS_TRACKING" value="AUTO" />
         <parameter name="FORCE_MAX_LATENCY_COUNT_WIDTH" value="0" />
         <parameter name="FORCE_SEQUENCER_TCL_DEBUG_MODE" value="false" />
         <parameter name="FORCE_SHADOW_REGS" value="AUTO" />
         <parameter name="FORCE_SYNTHESIS_LANGUAGE" value="" />
         <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC0_RX_CLK_IN"
value="100" />
         <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC0_TX_CLK_IN"
value="100" />
         <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC1_RX_CLK_IN"
value="100" />
         <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC1_TX_CLK_IN"
value="100" />
         <parameter
            name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC_PTP_REF_CLOCK"
            value="100" />
         <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C0_SCL_IN"
value="100" />
         <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C1_SCL_IN"
value="100" />
         <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C2_SCL_IN"
value="100" />
         <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C3_SCL_IN"
value="100" />
```

```
         <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_SPIS0_SCLK_IN"
value="100" />
         <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_SPIS1_SCLK_IN"
value="100" />
         <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_USB0_CLK_IN"
value="100" />
         <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_USB1_CLK_IN"
value="100" />
         <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC0_GTX_CLK"
value="125" />
         <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC0_MD_CLK"
value="2.5" />
         <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC1_GTX_CLK"
value="125" />
         <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC1_MD_CLK"
value="2.5" />
         <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C0_CLK"
value="100" />
         <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C1_CLK"
value="100" />
         <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C2_CLK"
value="100" />
         <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C3_CLK"
value="100" />
         <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_QSPI_SCLK_OUT"
value="100" />
```

```
         <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_SDIO_CCLK"
value="100" />
         <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_SPIM0_SCLK_OUT"
value="100" />
         <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_SPIM1_SCLK_OUT"
value="100" />
         <parameter
name="GPIO_Enable">No,No,No,No,No,No,No,No,No,Yes,No,No,No,No,No,No,No,No,No,No
,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,Yes,No,No,No,No,Yes,No,No,No,No,N
o,No,No,Yes,No,No,No,No,Yes,Yes,No,No,No,No,No,No,Yes,No,No,No,No,No,No,No,No,N
o,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No
,No,No,No</parameter>
         <parameter name="GP_Enable" value="false" />
         <parameter name="H2F_AXI_CLOCK_FREQ" value="67108863" />
         <parameter name="H2F_CTI_CLOCK_FREQ" value="100" />
         <parameter name="H2F_DEBUG_APB_CLOCK_FREQ" value="100" />
         <parameter name="H2F_LW_AXI_CLOCK_FREQ" value="12254661" />
         <parameter name="H2F_TPIU_CLOCK_IN_FREQ" value="100" />
         <parameter name="HARD_EMIF" value="true" />
         <parameter name="HCX_COMPAT_MODE" value="false" />
         <parameter name="HHP_HPS" value="true" />
         <parameter name="HHP_HPS_SIMULATION" value="false" />
         <parameter name="HHP_HPS_VERIFICATION" value="false" />
         <parameter name="HLGPI_Enable" value="false" />
         <parameter name="HPS_PROTOCOL" value="DDR3" />
         <parameter name="I2C0_Mode" value="I2C" />
```

```
        <parameter name="I2C0_PinMuxing" value="HPS I/O Set 0" />
        <parameter name="I2C1_Mode" value="I2C" />
        <parameter name="I2C1_PinMuxing" value="HPS I/O Set 0" />
        <parameter name="I2C2_Mode" value="N/A" />
        <parameter name="I2C2_PinMuxing" value="Unused" />
        <parameter name="I2C3_Mode" value="N/A" />
        <parameter name="I2C3_PinMuxing" value="Unused" />
        <parameter name="INCLUDE_BOARD_DELAY_MODEL" value="false" />
        <parameter name="INCLUDE_MULTIRANK_BOARD_DELAY_MODEL" value="false" />
        <parameter name="IS_ES_DEVICE" value="false" />
        <parameter
name="LOANIO_Enable">No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,N
o,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No
,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,
No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,N
o,No</parameter>
        <parameter name="LOCAL_ID_WIDTH" value="8" />
        <parameter
name="LRDIMM_EXTENDED_CONFIG">0x000000000000000000</parameter>
        <parameter name="LWH2F_Enable" value="true" />
        <parameter name="MARGIN_VARIATION_TEST" value="false" />
        <parameter name="MAX_PENDING_RD_CMD" value="32" />
        <parameter name="MAX_PENDING_WR_CMD" value="16" />
        <parameter name="MEM_ASR" value="Manual" />
        <parameter name="MEM_ATCL" value="Disabled" />
        <parameter name="MEM_AUTO_LEVELING_MODE" value="true" />
```

```
<parameter name="MEM_BANKADDR_WIDTH" value="3" />
<parameter name="MEM_BL" value="OTF" />
<parameter name="MEM_BT" value="Sequential" />
<parameter name="MEM_CK_PHASE" value="0.0" />
<parameter name="MEM_CK_WIDTH" value="1" />
<parameter name="MEM_CLK_EN_WIDTH" value="1" />
<parameter name="MEM_CLK_FREQ" value="400.0" />
<parameter name="MEM_CLK_FREQ_MAX" value="800.0" />
<parameter name="MEM_COL_ADDR_WIDTH" value="10" />
<parameter name="MEM_CS_WIDTH" value="1" />
<parameter name="MEM_DEVICE" value="MISSING_MODEL" />
<parameter name="MEM_DLL_EN" value="true" />
<parameter name="MEM_DQ_PER_DQS" value="8" />
<parameter name="MEM_DQ_WIDTH" value="32" />
<parameter name="MEM_DRV_STR" value="RZQ/7" />
<parameter name="MEM_FORMAT" value="DISCRETE" />
<parameter name="MEM_GUARANTEED_WRITE_INIT" value="false" />
<parameter name="MEM_IF_BOARD_BASE_DELAY" value="10" />
<parameter name="MEM_IF_DM_PINS_EN" value="true" />
<parameter name="MEM_IF_DQSN_EN" value="true" />
<parameter name="MEM_IF_SIM_VALID_WINDOW" value="0" />
<parameter name="MEM_INIT_EN" value="false" />
<parameter name="MEM_INIT_FILE" value="" />
<parameter name="MEM_MIRROR_ADDRESSING" value="0" />
<parameter name="MEM_NUMBER_OF_DIMMS" value="1" />
<parameter name="MEM_NUMBER_OF_RANKS_PER_DEVICE" value="1" />
```

```
<parameter name="MEM_NUMBER_OF_RANKS_PER_DIMM" value="1" />
<parameter name="MEM_PD" value="DLL off" />
<parameter name="MEM_RANK_MULTIPLICATION_FACTOR" value="1" />
<parameter name="MEM_ROW_ADDR_WIDTH" value="15" />
<parameter name="MEM_RTT_NOM" value="RZQ/4" />
<parameter name="MEM_RTT_WR" value="RZQ/4" />
<parameter name="MEM_SRT" value="Normal" />
<parameter name="MEM_TCL" value="11" />
<parameter name="MEM_TFAW_NS" value="30.0" />
<parameter name="MEM_TINIT_US" value="500" />
<parameter name="MEM_TMRD_CK" value="4" />
<parameter name="MEM_TRAS_NS" value="35.0" />
<parameter name="MEM_TRCD_NS" value="13.75" />
<parameter name="MEM_TREFI_US" value="7.8" />
<parameter name="MEM_TRFC_NS" value="260.0" />
<parameter name="MEM_TRP_NS" value="13.75" />
<parameter name="MEM_TRRD_NS" value="7.7" />
<parameter name="MEM_TRTP_NS" value="7.5" />
<parameter name="MEM_TWR_NS" value="15.0" />
<parameter name="MEM_TWTR" value="4" />
<parameter name="MEM_USER_LEVELING_MODE" value="Leveling" />
<parameter name="MEM_VENDOR" value="JEDEC" />
<parameter name="MEM_VERBOSE" value="true" />
<parameter name="MEM_VOLTAGE" value="1.5V DDR3" />
<parameter name="MEM_WTCL" value="8" />
<parameter name="MPU_EVENTS_Enable" value="false" />
```

```
         <parameter name="MRS_MIRROR_PING_PONG_ATSO" value="false" />
         <parameter name="MULTICAST_EN" value="false" />
         <parameter name="NAND_Mode" value="N/A" />
         <parameter name="NAND_PinMuxing" value="Unused" />
         <parameter name="NEXTGEN" value="true" />
         <parameter name="NIOS_ROM_DATA_WIDTH" value="32" />
         <parameter name="NUM_DLL_SHARING_INTERFACES" value="1" />
         <parameter name="NUM_EXTRA_REPORT_PATH" value="10" />
         <parameter name="NUM_OCT_SHARING_INTERFACES" value="1" />
         <parameter name="NUM_OF_PORTS" value="1" />
         <parameter name="NUM_PLL_SHARING_INTERFACES" value="1" />
         <parameter name="OCT_SHARING_MODE" value="None" />
         <parameter name="P2C_READ_CLOCK_ADD_PHASE" value="0.0" />
         <parameter name="PACKAGE_DESKEW" value="false" />
         <parameter name="PARSE_FRIENDLY_DEVICE_FAMILY_PARAM" value="" />
         <parameter name="PARSE_FRIENDLY_DEVICE_FAMILY_PARAM_VALID"
value="false" />
         <parameter name="PHY_CSR_CONNECTION" value="INTERNAL_JTAG" />
         <parameter name="PHY_CSR_ENABLED" value="false" />
         <parameter name="PHY_ONLY" value="false" />
         <parameter name="PINGPONGPHY_EN" value="false" />
         <parameter name="PLL_ADDR_CMD_CLK_DIV_PARAM" value="0" />
         <parameter name="PLL_ADDR_CMD_CLK_FREQ_PARAM" value="0.0" />
         <parameter name="PLL_ADDR_CMD_CLK_FREQ_SIM_STR_PARAM" value="" />
         <parameter name="PLL_ADDR_CMD_CLK_MULT_PARAM" value="0" />
         <parameter name="PLL_ADDR_CMD_CLK_PHASE_PS_PARAM" value="0" />
```

```
<parameter name="PLL_ADDR_CMD_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_AFI_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_AFI_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_AFI_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_AFI_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_HALF_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_AFI_HALF_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_AFI_HALF_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_HALF_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_AFI_HALF_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_AFI_HALF_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_PHY_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_AFI_PHY_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_AFI_PHY_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_PHY_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_AFI_PHY_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_AFI_PHY_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_C2P_WRITE_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_C2P_WRITE_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_C2P_WRITE_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_C2P_WRITE_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_C2P_WRITE_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_C2P_WRITE_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_CLK_PARAM_VALID" value="false" />
```

```
<parameter name="PLL_CONFIG_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_CONFIG_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_CONFIG_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_CONFIG_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_CONFIG_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_CONFIG_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_DR_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_DR_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_DR_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_DR_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_DR_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_DR_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_HR_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_HR_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_HR_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_HR_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_HR_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_HR_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_LOCATION" value="Top_Bottom" />
<parameter name="PLL_MEM_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_MEM_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_MEM_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_MEM_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_MEM_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_MEM_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_NIOS_CLK_DIV_PARAM" value="0" />
```

```
<parameter name="PLL_NIOS_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_NIOS_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_NIOS_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_NIOS_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_NIOS_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_P2C_READ_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_P2C_READ_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_P2C_READ_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_P2C_READ_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_P2C_READ_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_P2C_READ_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_SHARING_MODE" value="None" />
<parameter name="PLL_WRITE_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_WRITE_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_WRITE_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_WRITE_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_WRITE_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_WRITE_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="POWER_OF_TWO_BUS" value="false" />
<parameter name="PRIORITY_PORT" value="1,1,1,1,1,1" />
<parameter name="QSPI_Mode" value="N/A" />
<parameter name="QSPI_PinMuxing" value="Unused" />
<parameter name="RATE" value="Full" />
<parameter name="RDIMM_CONFIG" value="0000000000000000" />
<parameter name="READ_DQ_DQS_CLOCK_SOURCE" value="INVERTED_DQS_BUS" />
<parameter name="READ_FIFO_SIZE" value="8" />
```

```
<parameter name="REFRESH_BURST_VALIDATION" value="false" />
<parameter name="REFRESH_INTERVAL" value="15000" />
<parameter name="REF_CLK_FREQ" value="25.0" />
<parameter name="REF_CLK_FREQ_MAX_PARAM" value="0.0" />
<parameter name="REF_CLK_FREQ_MIN_PARAM" value="0.0" />
<parameter name="REF_CLK_FREQ_PARAM_VALID" value="false" />
<parameter name="S2FCLK_COLDRST_Enable" value="false" />
<parameter name="S2FCLK_PENDINGRST_Enable" value="false" />
<parameter name="S2FCLK_USER0CLK_Enable" value="false" />
<parameter name="S2FCLK_USER1CLK_Enable" value="true" />
<parameter name="S2FCLK_USER1CLK_FREQ" value="100.0" />
<parameter name="S2FCLK_USER2CLK" value="5" />
<parameter name="S2FCLK_USER2CLK_Enable" value="false" />
<parameter name="S2FCLK_USER2CLK_FREQ" value="100.0" />
<parameter name="S2FINTERRUPT_CAN_Enable" value="false" />
<parameter name="S2FINTERRUPT_CLOCKPERIPHERAL_Enable" value="false" />
<parameter name="S2FINTERRUPT_CTI_Enable" value="false" />
<parameter name="S2FINTERRUPT_DMA_Enable" value="false" />
<parameter name="S2FINTERRUPT_EMAC_Enable" value="false" />
<parameter name="S2FINTERRUPT_FPGAMANAGER_Enable" value="false" />
<parameter name="S2FINTERRUPT_GPIO_Enable" value="false" />
<parameter name="S2FINTERRUPT_I2CEMAC_Enable" value="false" />
<parameter name="S2FINTERRUPT_I2CPERIPHERAL_Enable" value="false" />
<parameter name="S2FINTERRUPT_L4TIMER_Enable" value="false" />
<parameter name="S2FINTERRUPT_NAND_Enable" value="false" />
<parameter name="S2FINTERRUPT_OSCTIMER_Enable" value="false" />
```

```
<parameter name="S2FINTERRUPT_QSPI_Enable" value="false" />
<parameter name="S2FINTERRUPT_SDMMC_Enable" value="false" />
<parameter name="S2FINTERRUPT_SPIMASTER_Enable" value="false" />
<parameter name="S2FINTERRUPT_SPISLAVE_Enable" value="false" />
<parameter name="S2FINTERRUPT_UART_Enable" value="false" />
<parameter name="S2FINTERRUPT_USB_Enable" value="false" />
<parameter name="S2FINTERRUPT_WATCHDOG_Enable" value="false" />
<parameter name="S2F_Width" value="2" />
<parameter name="SDIO_Mode" value="4-bit Data" />
<parameter name="SDIO_PinMuxing" value="HPS I/O Set 0" />
<parameter name="SEQUENCER_TYPE" value="NIOS" />
<parameter name="SEQ_MODE" value="0" />
<parameter name="SKIP_MEM_INIT" value="true" />
<parameter name="SOPC_COMPAT_RESET" value="false" />
<parameter name="SPEED_GRADE" value="7" />
<parameter name="SPIM0_Mode" value="N/A" />
<parameter name="SPIM0_PinMuxing" value="Unused" />
<parameter name="SPIM1_Mode" value="Single Slave Select" />
<parameter name="SPIM1_PinMuxing" value="HPS I/O Set 0" />
<parameter name="SPIS0_Mode" value="N/A" />
<parameter name="SPIS0_PinMuxing" value="Unused" />
<parameter name="SPIS1_Mode" value="N/A" />
<parameter name="SPIS1_PinMuxing" value="Unused" />
<parameter name="STARVE_LIMIT" value="10" />
<parameter name="STM_Enable" value="false" />
<parameter name="SYS_INFO_DEVICE_FAMILY" value="Cyclone V" />
```

```
        <parameter name="TEST_Enable" value="false" />
        <parameter name="TIMING_BOARD_AC_EYE_REDUCTION_H" value="0.0" />
        <parameter name="TIMING_BOARD_AC_EYE_REDUCTION_SU" value="0.0" />
        <parameter name="TIMING_BOARD_AC_SKEW" value="0.03" />
        <parameter name="TIMING_BOARD_AC_SLEW_RATE" value="1.0" />
        <parameter name="TIMING_BOARD_AC_TO_CK_SKEW" value="0.0" />
        <parameter name="TIMING_BOARD_CK_CKN_SLEW_RATE" value="2.0" />
        <parameter name="TIMING_BOARD_DELTA_DQS_ARRIVAL_TIME" value="0.0" />
        <parameter name="TIMING_BOARD_DELTA_READ_DQS_ARRIVAL_TIME" value="0.0"
/>
        <parameter name="TIMING_BOARD_DERATE_METHOD" value="AUTO" />
        <parameter name="TIMING_BOARD_DQS_DQSN_SLEW_RATE" value="2.0" />
        <parameter name="TIMING_BOARD_DQ_EYE_REDUCTION" value="0.0" />
        <parameter name="TIMING_BOARD_DQ_SLEW_RATE" value="1.0" />
        <parameter name="TIMING_BOARD_DQ_TO_DQS_SKEW" value="0.0" />
        <parameter name="TIMING_BOARD_ISI_METHOD" value="AUTO" />
        <parameter name="TIMING_BOARD_MAX_CK_DELAY" value="0.03" />
        <parameter name="TIMING_BOARD_MAX_DQS_DELAY" value="0.02" />
        <parameter name="TIMING_BOARD_READ_DQ_EYE_REDUCTION" value="0.0" />
        <parameter name="TIMING_BOARD_SKEW_BETWEEN_DIMMS" value="0.05" />
        <parameter name="TIMING_BOARD_SKEW_BETWEEN_DQS" value="0.08" />
        <parameter name="TIMING_BOARD_SKEW_CKDQS_DIMM_MAX" value="0.16" />
        <parameter name="TIMING_BOARD_SKEW_CKDQS_DIMM_MIN" value="0.09" />
        <parameter name="TIMING_BOARD_SKEW_WITHIN_DQS" value="0.01" />
        <parameter name="TIMING_BOARD_TDH" value="0.0" />
        <parameter name="TIMING_BOARD_TDS" value="0.0" />
```

```
<parameter name="TIMING_BOARD_TIH" value="0.0" />
<parameter name="TIMING_BOARD_TIS" value="0.0" />
<parameter name="TIMING_TDH" value="65" />
<parameter name="TIMING_TDQSCK" value="255" />
<parameter name="TIMING_TDQSCKDL" value="1200" />
<parameter name="TIMING_TDQSCKDM" value="900" />
<parameter name="TIMING_TDQSCKDS" value="450" />
<parameter name="TIMING_TDQSH" value="0.35" />
<parameter name="TIMING_TDQSQ" value="125" />
<parameter name="TIMING_TDQSS" value="0.25" />
<parameter name="TIMING_TDS" value="30" />
<parameter name="TIMING_TDSH" value="0.2" />
<parameter name="TIMING_TDSS" value="0.2" />
<parameter name="TIMING_TIH" value="140" />
<parameter name="TIMING_TIS" value="180" />
<parameter name="TIMING_TQH" value="0.38" />
<parameter name="TIMING_TQHS" value="300" />
<parameter name="TIMING_TQSH" value="0.4" />
<parameter name="TPIUFPGA_Enable" value="false" />
<parameter name="TPIUFPGA_alt" value="false" />
<parameter name="TRACE_Mode" value="N/A" />
<parameter name="TRACE_PinMuxing" value="Unused" />
<parameter name="TRACKING_ERROR_TEST" value="false" />
<parameter name="TRACKING_WATCH_TEST" value="false" />
<parameter name="TREFI" value="35100" />
<parameter name="TRFC" value="350" />
```

```
<parameter name="UART0_Mode" value="No Flow Control" />
<parameter name="UART0_PinMuxing" value="HPS I/O Set 0" />
<parameter name="UART1_Mode" value="N/A" />
<parameter name="UART1_PinMuxing" value="Unused" />
<parameter name="USB0_Mode" value="N/A" />
<parameter name="USB0_PinMuxing" value="Unused" />
<parameter name="USB1_Mode" value="SDR" />
<parameter name="USB1_PinMuxing" value="HPS I/O Set 0" />
<parameter name="USER_DEBUG_LEVEL" value="1" />
<parameter name="USE_AXI_ADAPTOR" value="false" />
<parameter name="USE_FAKE_PHY" value="false" />
<parameter name="USE_MEM_CLK_FREQ" value="false" />
<parameter name="USE_MM_ADAPTOR" value="true" />
<parameter name="USE_SEQUENCER_BFM" value="false" />
<parameter name="WEIGHT_PORT" value="0,0,0,0,0,0" />
<parameter name="WRBUFFER_ADDR_WIDTH" value="6" />
<parameter name="can0_clk_div" value="1" />
<parameter name="can1_clk_div" value="1" />
<parameter name="configure_advanced_parameters" value="false" />
<parameter name="customize_device_pll_info" value="false" />
<parameter name="dbctrl_stayosc1" value="true" />
<parameter name="dbg_at_clk_div" value="0" />
<parameter name="dbg_clk_div" value="1" />
<parameter name="dbg_trace_clk_div" value="0" />
<parameter name="desired_can0_clk_mhz" value="100.0" />
<parameter name="desired_can1_clk_mhz" value="100.0" />
```

```
        <parameter name="desired_cfg_clk_mhz" value="100.0" />
        <parameter name="desired_emac0_clk_mhz" value="250.0" />
        <parameter name="desired_emac1_clk_mhz" value="250.0" />
        <parameter name="desired_gpio_db_clk_hz" value="32000" />
        <parameter name="desired_l4_mp_clk_mhz" value="100.0" />
        <parameter name="desired_l4_sp_clk_mhz" value="100.0" />
        <parameter name="desired_mpu_clk_mhz" value="800.0" />
        <parameter name="desired_nand_clk_mhz" value="12.5" />
        <parameter name="desired_qspi_clk_mhz" value="400.0" />
        <parameter name="desired_sdmmc_clk_mhz" value="200.0" />
        <parameter name="desired_spi_m_clk_mhz" value="200.0" />
        <parameter name="desired_usb_mp_clk_mhz" value="200.0" />
        <parameter name="device_name" value="5CSEMA5F31C6" />
        <parameter name="device_pll_info_manual">{320000000 1600000000}
{320000000 1000000000} {800000000 400000000 400000000}</parameter>
        <parameter name="eosc1_clk_mhz" value="25.0" />
        <parameter name="eosc2_clk_mhz" value="25.0" />
        <parameter name="gpio_db_clk_div" value="6249" />
        <parameter name="l3_mp_clk_div" value="1" />
        <parameter name="l3_sp_clk_div" value="1" />
        <parameter name="l4_mp_clk_div" value="1" />
        <parameter name="l4_mp_clk_source" value="1" />
        <parameter name="l4_sp_clk_div" value="1" />
        <parameter name="l4_sp_clk_source" value="1" />
        <parameter name="main_pll_c3" value="3" />
        <parameter name="main_pll_c4" value="3" />
```

```
<parameter name="main_pll_c5" value="15" />
<parameter name="main_pll_m" value="63" />
<parameter name="main_pll_n" value="0" />
<parameter name="nand_clk_source" value="2" />
<parameter name="periph_pll_c0" value="3" />
<parameter name="periph_pll_c1" value="3" />
<parameter name="periph_pll_c2" value="1" />
<parameter name="periph_pll_c3" value="19" />
<parameter name="periph_pll_c4" value="4" />
<parameter name="periph_pll_c5" value="9" />
<parameter name="periph_pll_m" value="79" />
<parameter name="periph_pll_n" value="1" />
<parameter name="periph_pll_source" value="0" />
<parameter name="qspi_clk_source" value="1" />
<parameter name="quartus_ini_hps_emif_pll" value="false" />
<parameter
   name="quartus_ini_hps_ip_enable_all_peripheral_fpga_interfaces"
   value="false" />
<parameter name="quartus_ini_hps_ip_enable_bsel_csel" value="false" />
<parameter
   name="quartus_ini_hps_ip_enable_emac0_peripheral_fpga_interface"
   value="false" />
<parameter
   name="quartus_ini_hps_ip_enable_low_speed_serial_fpga_interfaces"
   value="false" />
```

```
         <parameter name="quartus_ini_hps_ip_enable_test_interface"
value="false" />
         <parameter name="quartus_ini_hps_ip_f2sdram_bonding_out" value="false"
/>
         <parameter name="quartus_ini_hps_ip_fast_f2sdram_sim_model"
value="false" />
         <parameter name="quartus_ini_hps_ip_suppress_sdram_synth" value="false"
/>
         <parameter name="sdmmc_clk_source" value="2" />
         <parameter name="show_advanced_parameters" value="false" />
         <parameter name="show_debug_info_as_warning_msg" value="false" />
         <parameter name="show_warning_as_error_msg" value="false" />
         <parameter name="spi_m_clk_div" value="0" />
         <parameter name="usb_mp_clk_div" value="0" />
         <parameter name="use_default_mpu_clk" value="true" />
        </module>
        <module
          name="onchip_memory2_0"
          kind="altera_avalon_onchip_memory2"
          version="21.1"
          enabled="1">
         <parameter name="allowInSystemMemoryContentEditor" value="false" />
         <parameter
name="autoInitializationFileName">$${FILENAME}_onchip_memory2_0</parameter>
         <parameter name="blockType" value="AUTO" />
         <parameter name="copyInitFile" value="false" />
```

```
         <parameter name="dataWidth" value="8" />
         <parameter name="dataWidth2" value="32" />
         <parameter name="deviceFamily" value="Cyclone V" />
         <parameter name="deviceFeatures">COMPILER_SUPPORT 1
CELL_LEVEL_BACK_ANNOTATION_DISABLED 0 ANY_QFP 0 ADDRESS_STALL 1 ADVANCED_INFO 0
ALLOWS_COMPILING_OTHER_FAMILY_IP 1
GENERATE_DC_ON_CURRENT_WARNING_FOR_INTERNAL_CLAMPING_DIODE 1 DSP 0
DSP_SHIFTER_BLOCK 0 DUMP_ASM_LAB_BITS_FOR_POWER 0 EMUL 1
ENABLE_ADVANCED_IO_ANALYSIS_GUI_FEATURES 1 ENABLE_PIN_PLANNER 0
ENGINEERING_SAMPLE 0 EPCS 1 ESB 0 FAKE1 0 FAKE2 0 FAKE3 0
FAMILY_LEVEL_INSTALLATION_ONLY 0 FASTEST 0 FINAL_TIMING_MODEL 0
FITTER_USE_FALLING_EDGE_DELAY 1 FPP_COMPLETELY_PLACES_AND_ROUTES_PERIPHERY 0
HARDCOPY 0 HAS_MICROPROCESSOR 0 HAS_MIF_SMART_COMPILE_SUPPORT 1
HAS_MINMAX_TIMING_MODELING_SUPPORT 1 HAS_MIN_TIMING_ANALYSIS_SUPPORT 1
HAS_MUX_RESTRUCTURE_SUPPORT 1 HAS_NADDER_STYLE_CLOCKING 0 HAS_NADDER_STYLE_FF 0
HAS_NADDER_STYLE_LCELL_COMB 0 HAS_NEW_CDB_NAME_FOR_M20K_SCLR 0
HAS_NEW_HC_FLOW_SUPPORT 0 HAS_NEW_SERDES_MAX_RESOURCE_COUNT_REPORTING_SUPPORT 0
HAS_NEW_VPR_SUPPORT 1 HAS_NONSOCKET_TECHNOLOGY_MIGRATION_SUPPORT 0
HAS_NO_HARDBLOCK_PARTITION_SUPPORT 0 HAS_NO_JTAG_USERCODE_SUPPORT 0
HAS_OPERATING_SETTINGS_AND_CONDITIONS_REPORTING_SUPPORT 1 HAS_ACE_SUPPORT 1
HAS_ACTIVE_PARALLEL_FLASH_SUPPORT 0 HAS_ADJUSTABLE_OUTPUT_IO_TIMING_MEAS_POINT
1 HAS_ADVANCED_IO_INVERTED_CORNER 1 HAS_ADVANCED_IO_POWER_SUPPORT 1
HAS_ADVANCED_IO_TIMING_SUPPORT 1 HAS_ALM_SUPPORT 1
HAS_ATOM_AND_ROUTING_POWER_MODELED_TOGETHER 0
HAS_AUTO_DERIVE_CLOCK_UNCERTAINTY_SUPPORT 1 HAS_AUTO_FIT_SUPPORT 1
HAS_BALANCED_OPT_TECHNIQUE_SUPPORT 1 HAS_BENEFICIAL_SKEW_SUPPORT 0
```

```
HAS_BITLEVEL_DRIVE_STRENGTH_CONTROL 1 HAS_BSDL_FILE_GENERATION 1

HAS_CDB_RE_NETWORK_PRESERVATION_SUPPORT 0 HAS_CGA_SUPPORT 1

HAS_CHECK_NETLIST_SUPPORT 1 HAS_CLOCK_REGION_CHECKER_ENABLED 1

HAS_CORE_JUNCTION_TEMP_DERATING 0 HAS_CROSSTALK_SUPPORT 0

HAS_CUSTOM_REGION_SUPPORT 1 HAS_DAP_JTAG_FROM_HPS 0

HAS_DATA_DRIVEN_ACVQ_HSSI_SUPPORT 1 HAS_DDB_FDI_SUPPORT 1

HAS_DESIGN_ANALYZER_SUPPORT 1 HAS_DETAILED_IO_RAIL_POWER_MODEL 1

HAS_DETAILED_LEIM_STATIC_POWER_MODEL 0 HAS_DETAILED_LE_POWER_MODEL 1

HAS_DETAILED_ROUTING_MUX_STATIC_POWER_MODEL 0

HAS_DETAILED_THERMAL_CIRCUIT_PARAMETER_SUPPORT 1 HAS_DEVICE_MIGRATION_SUPPORT 1

HAS_DIAGONAL_MIGRATION_SUPPORT 0 HAS_EMIF_TOOLKIT_SUPPORT 1

HAS_ERROR_DETECTION_SUPPORT 1 HAS_FAMILY_VARIANT_MIGRATION_SUPPORT 0

HAS_FANOUT_FREE_NODE_SUPPORT 1 HAS_FAST_FIT_SUPPORT 1

HAS_FIT_NETLIST_OPT_RETIME_SUPPORT 1 HAS_FIT_NETLIST_OPT_SUPPORT 1

HAS_FITTER_ECO_SUPPORT 1 HAS_FORMAL_VERIFICATION_SUPPORT 0

HAS_FPGA_XCHANGE_SUPPORT 1 HAS_FSAC_LUTRAM_REGISTER_PACKING_SUPPORT 1

HAS_FULL_DAT_MIN_TIMING_SUPPORT 1 HAS_FULL_INCREMENTAL_DESIGN_SUPPORT 1

HAS_FUNCTIONAL_SIMULATION_SUPPORT 0 HAS_FUNCTIONAL_VERILOG_SIMULATION_SUPPORT 1

HAS_FUNCTIONAL_VHDL_SIMULATION_SUPPORT 1 HAS_GLITCH_FILTERING_SUPPORT 1

HAS_HARDCOPYII_SUPPORT 0 HAS_HC_READY_SUPPORT 0

HAS_HIGH_SPEED_LOW_POWER_TILE_SUPPORT 0

HAS_HOLD_TIME_AVOIDANCE_ACROSS_CLOCK_SPINE_SUPPORT 1 HAS_HSSI_POWER_CALCULATOR

1 HAS_HSPICE_WRITER_SUPPORT 1 HAS_IBISO_WRITER_SUPPORT 0 HAS_ICD_DATA_IP 0

HAS_IDB_SUPPORT 1 HAS_INCREMENTAL_DAT_SUPPORT 1

HAS_INCREMENTAL_SYNTHESIS_SUPPORT 1 HAS_IO_ASSIGNMENT_ANALYSIS_SUPPORT 1

HAS_IO_DECODER 1 HAS_IO_PLACEMENT_OPTIMIZATION_SUPPORT 1
```

```
HAS_IO_PLACEMENT_USING_GEOMETRY_RULE 0 HAS_IO_PLACEMENT_USING_PHYSIC_RULE 0
HAS_IO_SMART_RECOMPILE_SUPPORT 0 HAS_JITTER_SUPPORT 1 HAS_JTAG_SLD_HUB_SUPPORT
1 HAS_LOGIC_LOCK_SUPPORT 1 HAS_PAD_LOCATION_ASSIGNMENT_SUPPORT 0
HAS_PASSIVE_PARALLEL_SUPPORT 0 HAS_PARTIAL_RECONFIG_SUPPORT 1
HAS_PDN_MODEL_STATUS 0 HAS_PHYSICAL_NETLIST_OUTPUT 0
HAS_PHYSICAL_DESIGN_PLANNER_SUPPORT 0 HAS_PHYSICAL_ROUTING_SUPPORT 1
HAS_PIN_SPECIFIC_VOLTAGE_SUPPORT 1 HAS_PLDM_REF_SUPPORT 0
HAS_POWER_BINNING_LIMITS_DATA 1 HAS_POWER_ESTIMATION_SUPPORT 1
HAS_PRELIMINARY_CLOCK_UNCERTAINTY_NUMBERS 0 HAS_PRE_FITTER_FPP_SUPPORT 1
HAS_PRE_FITTER_LUTRAM_NETLIST_CHECKER_ENABLED 1 HAS_PVA_SUPPORT 1
HAS_QUARTUS_HIERARCHICAL_DESIGN_SUPPORT 0 HAS_RAPID_RECOMPILE_SUPPORT 1
HAS_RCF_SUPPORT 1 HAS_RCF_SUPPORT_FOR_DEBUGGING 0
HAS_RED_BLACK_SEPARATION_SUPPORT 0 HAS_RE_LEVEL_TIMING_GRAPH_SUPPORT 1
HAS_RISEFALL_DELAY_SUPPORT 1 HAS_SIGNAL_PROBE_SUPPORT 1 HAS_SIGNAL_TAP_SUPPORT
1 HAS_SIMULATOR_SUPPORT 0 HAS_SPLIT_IO_SUPPORT 1 HAS_SPLIT_LC_SUPPORT 1
HAS_STRICT_PRESERVATION_SUPPORT 1 HAS_SYNTHESIS_ON_ATOMS 1
HAS_SYNTH_NETLIST_OPT_RETIME_SUPPORT 0 HAS_SYNTH_NETLIST_OPT_SUPPORT 1
HAS_SYNTH_FSYN_NETLIST_OPT_SUPPORT 1 HAS_TCL_FITTER_SUPPORT 0
HAS_TECHNOLOGY_MIGRATION_SUPPORT 0 HAS_TEMPLATED_REGISTER_PACKING_SUPPORT 1
HAS_TIME_BORROWING_SUPPORT 0 HAS_TIMING_DRIVEN_SYNTHESIS_SUPPORT 1
HAS_TIMING_INFO_SUPPORT 1 HAS_TIMING_OPERATING_CONDITIONS 1
HAS_TIMING_SIMULATION_SUPPORT 0 HAS_TITAN_BASED_MAC_REGISTER_PACKER_SUPPORT 1
HAS_U2B2_SUPPORT 0 HAS_USE_FITTER_INFO_SUPPORT 0
HAS_USER_HIGH_SPEED_LOW_POWER_TILE_SUPPORT 0 HAS_VCCPD_POWER_RAIL 1
HAS_VERTICAL_MIGRATION_SUPPORT 1 HAS_VIEWDRAW_SYMBOL_SUPPORT 0 HAS_VIO_SUPPORT
1 HAS_VIRTUAL_DEVICES 0 HAS_WYSIWYG_DFFEAS_SUPPORT 1 HAS_XIBISO_WRITER_SUPPORT
```

```
1 HAS_XIBISO2_WRITER_SUPPORT 0 HAS_18_BIT_MULTS 1
INCREMENTAL_DESIGN_SUPPORTS_COMPATIBLE_CONSTRAINTS 0 INSTALLED 0
INTERNAL_POF_SUPPORT_ENABLED 0 INTERNAL_USE_ONLY 0 IFP_USE_LEGACY_IO_CHECKER 1
ISSUE_MILITARY_TEMPERATURE_WARNING 0 IS_CONFIG_ROM 0 IS_BARE_DIE 0
IS_DEFAULT_FAMILY 0 IS_FOR_INTERNAL_TESTING_ONLY 0 IS_HARDCOPY_FAMILY 0
IS_HBGA_PACKAGE 0 IS_HIGH_CURRENT_PART 0 IS_JW_NEW_BINNING_PLAN 0
IS_JZ_NEW_BINNING_PLAN 0 IS_LOW_POWER_PART 0 IS_SMI_PART 0 IS_SDM_ONLY_PACKAGE
0 IS_REVE_SILICON 0 LOAD_BLK_TYPE_DATA_FROM_ATOM_WYS_INFO 0 LVDS_IO 1
M144K_MEMORY 0 M10K_MEMORY 1 M20K_MEMORY 0 M4K_MEMORY 0 M512_MEMORY 0
M9K_MEMORY 0 MLAB_MEMORY 1 MRAM_MEMORY 0 NOT_MIGRATABLE 0 NOT_LISTED 0
NO_FITTER_DELAY_CACHE_GENERATED 0
NO_SUPPORT_FOR_LOGICLOCK_CONTENT_BACK_ANNOTATION 1
NO_SUPPORT_FOR_STA_CLOCK_UNCERTAINTY_CHECK 0 NO_POF 0 NO_PIN_OUT 0
NO_RPE_SUPPORT 0 NO_TDC_SUPPORT 0 SHOW_HIDDEN_FAMILY_IN_PROGRAMMER 0
STRICT_TIMING_DB_CHECKS 0 SUPPORT_HIGH_SPEED_HPS 0 SUPPORTS_1P0V_IOSTD 0
SUPPORTS_CRC 1 SUPPORTS_ADDITIONAL_OPTIONS_FOR_UNUSED_IO 1
SUPPORTS_GENERATION_OF_EARLY_POWER_ESTIMATOR_FILE 1
SUPPORTS_GLOBAL_SIGNAL_BACK_ANNOTATION 1
SUPPORTS_DIFFERENTIAL_AIOT_BOARD_TRACE_MODEL 1
SUPPORTS_DSP_BALANCING_BACK_ANNOTATION 0 SUPPORTS_HIPI_RETIMING 0
SUPPORTS_LICENSE_FREE_PARTIAL_RECONFIG 0 SUPPORTS_MAC_CHAIN_OUT_ADDER 1
SUPPORTS_NEW_BINNING_PLAN 0 SUPPORTS_SIGNALPROBE_REGISTER_PIPELINING 1
SUPPORTS_SINGLE_ENDED_AIOT_BOARD_TRACE_MODEL 1
SUPPORTS_RAM_PACKING_BACK_ANNOTATION 0 SUPPORTS_REG_PACKING_BACK_ANNOTATION 0
SUPPORTS_USER_MANUAL_LOGIC_DUPLICATION 1 SUPPORTS_VID 0
POSTMAP_BAK_DATABASE_EXPORT_ENABLED 1 POSTFIT_BAK_DATABASE_EXPORT_ENABLED 1
```

```
PROGRAMMER_ONLY 0 PROGRAMMER_SUPPORT 1 PVA_SUPPORTS_ONLY_SUBSET_OF_ATOMS 0
QMAP_IN_DEVELOPMENT 0 QFIT_IN_DEVELOPMENT 0
RAM_LOGICAL_NAME_CHECKING_IN_CUT_ENABLED 1 REPORTS_METASTABILITY_MTBF 1
REQUIRE_QUARTUS_HIERARCHICAL_DESIGN 0
REQUIRE_SPECIAL_HANDLING_FOR_LOCAL_LABLINE 0 REQUIRES_INSTALLATION_PATCH 0
REQUIRES_LIST_OF_TEMPERATURE_AND_VOLTAGE_OPERATING_CONDITIONS 1
RESERVES_SIGNAL_PROBE_PINS 0 RESOLVE_MAX_FANOUT_EARLY 1 RESOLVE_MAX_FANOUT_LATE
0 RESPECTS_FIXED_SIZED_LOCKED_LOCATION_LOGICLOCK 1 RESTRICTED_USER_SELECTION 0
RESTRICT_PARTIAL_RECONFIG 0 RISEFALL_SUPPORT_IS_HIDDEN 0
WYSIWYG_BUS_WIDTH_CHECKING_IN_CUT_ENABLED 1 TMV_RUN_CUSTOMIZABLE_VIEWER 1
TMV_RUN_INTERNAL_DETAILS 1 TMV_RUN_INTERNAL_DETAILS_ON_IO 0
TMV_RUN_INTERNAL_DETAILS_ON_IOBUF 1 TMV_RUN_INTERNAL_DETAILS_ON_LCELL 0
TMV_RUN_INTERNAL_DETAILS_ON_LRAM 0 TRANSCEIVER_3G_BLOCK 1 TRANSCEIVER_6G_BLOCK
1 USES_ACV_FOR_FLED 1 USES_ADB_FOR_BACK_ANNOTATION 1 USES_ALTERA_LNSIM 0
USES_ASIC_ROUTING_POWER_CALCULATOR 0 USES_DATA_DRIVEN_PLL_COMPUTATION_UTIL 1
USES_DEV 1 USES_ICP_FOR_ECO_FITTER 0 USES_LIBERTY_TIMING 0
USES_NETWORK_ROUTING_POWER_CALCULATOR 0
USES_PART_INFO_FOR_DISPLAYING_CORE_VOLTAGE_VALUE 0 USES_POWER_SIGNAL_ACTIVITIES
1 USES_PVAFAM2 0 USES_SECOND_GENERATION_PART_INFO 0
USES_SECOND_GENERATION_POWER_ANALYZER 0 USES_THIRD_GENERATION_TIMING_MODELS_TIS
1 USES_U2B2_TIMING_MODELS 0 USES_XML_FORMAT_FOR_EMIF_PIN_MAP_FILE 0
USE_OCT_AUTO_CALIBRATION 1 USE_ADVANCED_IO_POWER_BY_DEFAULT 1
USE_ADVANCED_IO_TIMING_BY_DEFAULT 1 USE_BASE_FAMILY_DDB_PATH 0
USE_RELAX_IO_ASSIGNMENT_RULES 0 USE_RISEFALL_ONLY 1
USE_SEPARATE_LIST_FOR_TECH_MIGRATION 0
USE_SINGLE_COMPILER_PASS_PLL_MIF_FILE_WRITER 1
```

```
USE_TITAN_IO_BASED_IO_REGISTER_PACKER_UTIL 1
USING_28NM_OR_OLDER_TIMING_METHODOLOGY 1</parameter>
         <parameter name="dualPort" value="false" />
         <parameter name="ecc_enabled" value="false" />
         <parameter name="enPRInitMode" value="false" />
         <parameter name="enableDiffWidth" value="false" />
         <parameter name="initMemContent" value="true" />
         <parameter name="initializationFileName" value="onchip_mem.hex" />
         <parameter name="instanceID" value="NONE" />
         <parameter name="memorySize" value="65536" />
         <parameter name="readDuringWriteMode" value="DONT_CARE" />
         <parameter name="resetrequest_enabled" value="true" />
         <parameter name="simAllowMRAMContentsFile" value="false" />
         <parameter name="simMemInitOnlyFilename" value="0" />
         <parameter name="singleClockOperation" value="false" />
         <parameter name="slave1Latency" value="1" />
         <parameter name="slave2Latency" value="1" />
         <parameter name="useNonDefaultInitFile" value="false" />
         <parameter name="useShallowMemBlocks" value="false" />
         <parameter name="writable" value="true" />
        </module>
        <connection
          kind="avalon"
          version="21.1"
          start="GameBoy_Cartridge.avalon_master"
          end="onchip_memory2_0.s1">
```

```
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0000" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="21.1"
  start="hps_0.h2f_axi_master"
  end="AV_Config.avalon_av_config_slave">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x04203000" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="21.1"
  start="hps_0.h2f_axi_master"
  end="GameBoy_VGA.avalon_slave">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x04000000" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="21.1"
  start="hps_0.h2f_axi_master"
```

```
    end="GameBoy_Joypad.avalon_slave">
   <parameter name="arbitrationPriority" value="1" />
   <parameter name="baseAddress" value="0x04200000" />
   <parameter name="defaultConnection" value="false" />
  </connection>
  <connection
    kind="avalon"
    version="21.1"
    start="hps_0.h2f_axi_master"
    end="GameBoy_Cartridge.hps_slave">
   <parameter name="arbitrationPriority" value="1" />
   <parameter name="baseAddress" value="0x0000" />
   <parameter name="defaultConnection" value="false" />
  </connection>
  <connection
    kind="clock"
    version="21.1"
    start="GameBoy_Cartridge.clock_gameboy"
    end="GameBoy_VGA.GameBoy_clock" />
  <connection
    kind="clock"
    version="21.1"
    start="GameBoy_Cartridge.clock_gameboy"
    end="GameBoy.clock" />
  <connection
    kind="clock"
```

```
      version="21.1"
      start="Main_PLL.outclk1"
      end="GameBoy_Reset.clk" />
    <connection
      kind="clock"
      version="21.1"
      start="Main_PLL.outclk1"
      end="SOC_System_RST_Bridge.clk" />
    <connection
      kind="clock"
      version="21.1"
      start="Main_PLL.outclk1"
      end="onchip_memory2_0.clk1" />
    <connection
      kind="clock"
      version="21.1"
      start="Main_PLL.outclk1"
      end="GameBoy_VGA.clock" />
    <connection
      kind="clock"
      version="21.1"
      start="Main_PLL.outclk1"
      end="GameBoy_Joypad.clock" />
    <connection
      kind="clock"
      version="21.1"
```

```
      start="Main_PLL.outclk1"
      end="GameBoy_Cartridge.clock" />
    <connection
      kind="clock"
      version="21.1"
      start="Main_PLL.outclk1"
      end="hps_0.f2h_axi_clock" />
    <connection
      kind="clock"
      version="21.1"
      start="Main_PLL.outclk1"
      end="hps_0.h2f_axi_clock" />
    <connection
      kind="clock"
      version="21.1"
      start="Main_PLL.outclk2"
      end="AV_Config.clk" />
    <connection
      kind="clock"
      version="21.1"
      start="Main_PLL.outclk2"
      end="GameBoy_VGA.clock_vga" />
    <connection
      kind="clock"
      version="21.1"
      start="Main_PLL.outclk4"
```

```
    end="hps_0.h2f_lw_axi_clock" />
  <connection
    kind="conduit"
    version="21.1"
    start="GameBoy.GameBoy_Joypad"
    end="GameBoy_Joypad.GameBoy_JoyPad">
   <parameter name="endPort" value="" />
   <parameter name="endPortLSB" value="0" />
   <parameter name="startPort" value="" />
   <parameter name="startPortLSB" value="0" />
   <parameter name="width" value="0" />
  </connection>
  <connection
    kind="conduit"
    version="21.1"
    start="GameBoy.GameBoy_Pixel"
    end="GameBoy_VGA.GameBoy_Pixel">
   <parameter name="endPort" value="" />
   <parameter name="endPortLSB" value="0" />
   <parameter name="startPort" value="" />
   <parameter name="startPortLSB" value="0" />
   <parameter name="width" value="0" />
  </connection>
  <connection
    kind="conduit"
    version="21.1"
```

```
    start="GameBoy.GameBoy_ROM"
    end="GameBoy_Cartridge.GameBoy_Cartridge">
   <parameter name="endPort" value="" />
   <parameter name="endPortLSB" value="0" />
   <parameter name="startPort" value="" />
   <parameter name="startPortLSB" value="0" />
   <parameter name="width" value="0" />
  </connection>
  <connection
    kind="reset"
    version="21.1"
    start="GameBoy_Reset.out_reset"
    end="GameBoy_VGA.GameBoy_reset" />
  <connection
    kind="reset"
    version="21.1"
    start="SOC_System_RST_Bridge.out_reset"
    end="GameBoy_Cartridge.reset" />
  <connection
    kind="reset"
    version="21.1"
    start="SOC_System_RST_Bridge.out_reset"
    end="GameBoy_Joypad.reset" />
  <connection
    kind="reset"
    version="21.1"
```

```
      start="SOC_System_RST_Bridge.out_reset"
      end="GameBoy_VGA.reset" />
    <connection
      kind="reset"
      version="21.1"
      start="SOC_System_RST_Bridge.out_reset"
      end="AV_Config.reset" />
    <connection
      kind="reset"
      version="21.1"
      start="SOC_System_RST_Bridge.out_reset"
      end="Main_PLL.reset" />
    <connection
      kind="reset"
      version="21.1"
      start="SOC_System_RST_Bridge.out_reset"
      end="onchip_memory2_0.reset1" />
    <connection
      kind="reset"
      version="21.1"
      start="GameBoy_Cartridge.reset_gameboy"
      end="GameBoy_VGA.GameBoy_reset" />
    <connection
      kind="reset"
      version="21.1"
      start="GameBoy_Cartridge.reset_gameboy"
```

```
          end="GameBoy_Reset.in_reset" />
        <interconnectRequirement for="$system"
name="qsys_mm.clockCrossingAdapter" value="HANDSHAKE" />
        <interconnectRequirement for="$system"
name="qsys_mm.enableEccProtection" value="FALSE" />
        <interconnectRequirement for="$system" name="qsys_mm.insertDefaultSlave"
value="FALSE" />
        <interconnectRequirement for="$system"
name="qsys_mm.maxAdditionalLatency" value="1" />
       </system>
```

soc_system.tcl

Unset

```
       # Generate Quartus project files for the DE1-SoC board
       #
       # Stephen A. Edwards, Columbia University


       # Invoke as
       #
       # quartus_sh -t soc_system.tcl


       set project "soc_system"
```

```
        # Top-level SystemVerilog file should be <project>_top.sv, with Verilog
module
        # <project>_top in it

        set systemVerilogSource "${project}_top.sv"
        set qip "${project}/synthesis/${project}.qip"

        project_new $project -overwrite

        foreach {name value} {
            FAMILY "Cyclone V"
            DEVICE 5CSEMA5F31C6

            PROJECT_OUTPUT_DIRECTORY output_files

            CYCLONEII_RESERVE_NCEO_AFTER_CONFIGURATION "USE AS REGULAR IO"

            NUM_PARALLEL_PROCESSORS 4

        } { set_global_assignment -name $name $value }

        set_global_assignment -name TOP_LEVEL_ENTITY "${project}_top"

        foreach filename $systemVerilogSource {
            set_global_assignment -name SYSTEMVERILOG_FILE $filename
        }
```

```
foreach filename $qip {
    set_global_assignment -name QIP_FILE $filename
}

# FPGA pin assignments

foreach {pin port} {
    PIN_AJ4 ADC_CS_N
    PIN_AK4 ADC_DIN
    PIN_AK3 ADC_DOUT
    PIN_AK2 ADC_SCLK

    PIN_K7 AUD_ADCDAT
    PIN_K8 AUD_ADCLRCK
    PIN_H7 AUD_BCLK
    PIN_J7 AUD_DACDAT
    PIN_H8 AUD_DACLRCK
    PIN_G7 AUD_XCK

    PIN_AA16 CLOCK2_50
    PIN_Y26 CLOCK3_50
    PIN_K14 CLOCK4_50
    PIN_AF14 CLOCK_50

    PIN_AK14 DRAM_ADDR[0]
```

```
PIN_AH14 DRAM_ADDR[1]

PIN_AG15 DRAM_ADDR[2]

PIN_AE14 DRAM_ADDR[3]

PIN_AB15 DRAM_ADDR[4]

PIN_AC14 DRAM_ADDR[5]

PIN_AD14 DRAM_ADDR[6]

PIN_AF15 DRAM_ADDR[7]

PIN_AH15 DRAM_ADDR[8]

PIN_AG13 DRAM_ADDR[9]

PIN_AG12 DRAM_ADDR[10]

PIN_AH13 DRAM_ADDR[11]

PIN_AJ14 DRAM_ADDR[12]

PIN_AF13 DRAM_BA[0]

PIN_AJ12 DRAM_BA[1]

PIN_AF11 DRAM_CAS_N

PIN_AK13 DRAM_CKE

PIN_AH12 DRAM_CLK

PIN_AG11 DRAM_CS_N

PIN_AK6 DRAM_DQ[0]

PIN_AJ7 DRAM_DQ[1]

PIN_AK7 DRAM_DQ[2]

PIN_AK8 DRAM_DQ[3]

PIN_AK9 DRAM_DQ[4]

PIN_AG10 DRAM_DQ[5]

PIN_AK11 DRAM_DQ[6]

PIN_AJ11 DRAM_DQ[7]
```

```
PIN_AH10 DRAM_DQ[8]
PIN_AJ10 DRAM_DQ[9]
PIN_AJ9 DRAM_DQ[10]
PIN_AH9 DRAM_DQ[11]
PIN_AH8 DRAM_DQ[12]
PIN_AH7 DRAM_DQ[13]
PIN_AJ6 DRAM_DQ[14]
PIN_AJ5 DRAM_DQ[15]
PIN_AB13 DRAM_LDQM
PIN_AE13 DRAM_RAS_N
PIN_AK12 DRAM_UDQM
PIN_AA13 DRAM_WE_N

PIN_AA12 FAN_CTRL

PIN_J12 FPGA_I2C_SCLK
PIN_K12 FPGA_I2C_SDAT

PIN_AC18 GPIO_0[0]
PIN_Y17  GPIO_0[1]
PIN_AD17 GPIO_0[2]
PIN_Y18  GPIO_0[3]
PIN_AK16 GPIO_0[4]
PIN_AK18 GPIO_0[5]
PIN_AK19 GPIO_0[6]
PIN_AJ19 GPIO_0[7]
```

```
PIN_AJ17 GPIO_0[8]

PIN_AJ16 GPIO_0[9]

PIN_AH18 GPIO_0[10]

PIN_AH17 GPIO_0[11]

PIN_AG16 GPIO_0[12]

PIN_AE16 GPIO_0[13]

PIN_AF16 GPIO_0[14]

PIN_AG17 GPIO_0[15]

PIN_AA18 GPIO_0[16]

PIN_AA19 GPIO_0[17]

PIN_AE17 GPIO_0[18]

PIN_AC20 GPIO_0[19]

PIN_AH19 GPIO_0[20]

PIN_AJ20 GPIO_0[21]

PIN_AH20 GPIO_0[22]

PIN_AK21 GPIO_0[23]

PIN_AD19 GPIO_0[24]

PIN_AD20 GPIO_0[25]

PIN_AE18 GPIO_0[26]

PIN_AE19 GPIO_0[27]

PIN_AF20 GPIO_0[28]

PIN_AF21 GPIO_0[29]

PIN_AF19 GPIO_0[30]

PIN_AG21 GPIO_0[31]

PIN_AF18 GPIO_0[32]

PIN_AG20 GPIO_0[33]
```

```
PIN_AG18 GPIO_0[34]
PIN_AJ21 GPIO_0[35]

PIN_AB17 GPIO_1[0]
PIN_AA21 GPIO_1[1]
PIN_AB21 GPIO_1[2]
PIN_AC23 GPIO_1[3]
PIN_AD24 GPIO_1[4]
PIN_AE23 GPIO_1[5]
PIN_AE24 GPIO_1[6]
PIN_AF25 GPIO_1[7]
PIN_AF26 GPIO_1[8]
PIN_AG25 GPIO_1[9]
PIN_AG26 GPIO_1[10]
PIN_AH24 GPIO_1[11]
PIN_AH27 GPIO_1[12]
PIN_AJ27 GPIO_1[13]
PIN_AK29 GPIO_1[14]
PIN_AK28 GPIO_1[15]
PIN_AK27 GPIO_1[16]
PIN_AJ26 GPIO_1[17]
PIN_AK26 GPIO_1[18]
PIN_AH25 GPIO_1[19]
PIN_AJ25 GPIO_1[20]
PIN_AJ24 GPIO_1[21]
PIN_AK24 GPIO_1[22]
```

```
PIN_AG23 GPIO_1[23]

PIN_AK23 GPIO_1[24]

PIN_AH23 GPIO_1[25]

PIN_AK22 GPIO_1[26]

PIN_AJ22 GPIO_1[27]

PIN_AH22 GPIO_1[28]

PIN_AG22 GPIO_1[29]

PIN_AF24 GPIO_1[30]

PIN_AF23 GPIO_1[31]

PIN_AE22 GPIO_1[32]

PIN_AD21 GPIO_1[33]

PIN_AA20 GPIO_1[34]

PIN_AC22 GPIO_1[35]


PIN_AE26 HEX0[0]

PIN_AE27 HEX0[1]

PIN_AE28 HEX0[2]

PIN_AG27 HEX0[3]

PIN_AF28 HEX0[4]

PIN_AG28 HEX0[5]

PIN_AH28 HEX0[6]


PIN_AJ29 HEX1[0]

PIN_AH29 HEX1[1]

PIN_AH30 HEX1[2]

PIN_AG30 HEX1[3]
```

```
PIN_AF29 HEX1[4]
PIN_AF30 HEX1[5]
PIN_AD27 HEX1[6]

PIN_AB23 HEX2[0]
PIN_AE29 HEX2[1]
PIN_AD29 HEX2[2]
PIN_AC28 HEX2[3]
PIN_AD30 HEX2[4]
PIN_AC29 HEX2[5]
PIN_AC30 HEX2[6]

PIN_AD26 HEX3[0]
PIN_AC27 HEX3[1]
PIN_AD25 HEX3[2]
PIN_AC25 HEX3[3]
PIN_AB28 HEX3[4]
PIN_AB25 HEX3[5]
PIN_AB22 HEX3[6]

PIN_AA24 HEX4[0]
PIN_Y23  HEX4[1]
PIN_Y24  HEX4[2]
PIN_W22  HEX4[3]
PIN_W24  HEX4[4]
PIN_V23  HEX4[5]
```

```
PIN_W25  HEX4[6]


PIN_V25  HEX5[0]
PIN_AA28 HEX5[1]
PIN_Y27  HEX5[2]
PIN_AB27 HEX5[3]
PIN_AB26 HEX5[4]
PIN_AA26 HEX5[5]
PIN_AA25 HEX5[6]


PIN_AA30 IRDA_RXD
PIN_AB30 IRDA_TXD


PIN_AA14 KEY[0]
PIN_AA15 KEY[1]
PIN_W15  KEY[2]
PIN_Y16  KEY[3]


PIN_V16 LEDR[0]
PIN_W16 LEDR[1]
PIN_V17 LEDR[2]
PIN_V18 LEDR[3]
PIN_W17 LEDR[4]
PIN_W19 LEDR[5]
PIN_Y19 LEDR[6]
PIN_W20 LEDR[7]
```

```
PIN_W21 LEDR[8]
PIN_Y21 LEDR[9]

PIN_AD7 PS2_CLK
PIN_AD9 PS2_CLK2
PIN_AE7 PS2_DAT
PIN_AE9 PS2_DAT2

PIN_AB12 SW[0]
PIN_AC12 SW[1]
PIN_AF9  SW[2]
PIN_AF10 SW[3]
PIN_AD11 SW[4]
PIN_AD12 SW[5]
PIN_AE11 SW[6]
PIN_AC9  SW[7]
PIN_AD10 SW[8]
PIN_AE12 SW[9]

PIN_H15 TD_CLK27
PIN_D2  TD_DATA[0]
PIN_B1  TD_DATA[1]
PIN_E2  TD_DATA[2]
PIN_B2  TD_DATA[3]
PIN_D1  TD_DATA[4]
PIN_E1  TD_DATA[5]
```

```
PIN_C2  TD_DATA[6]
PIN_B3  TD_DATA[7]
PIN_A5  TD_HS
PIN_F6  TD_RESET_N
PIN_A3  TD_VS

PIN_A13 VGA_R[0]
PIN_C13 VGA_R[1]
PIN_E13 VGA_R[2]
PIN_B12 VGA_R[3]
PIN_C12 VGA_R[4]
PIN_D12 VGA_R[5]
PIN_E12 VGA_R[6]
PIN_F13 VGA_R[7]

PIN_J9  VGA_G[0]
PIN_J10 VGA_G[1]
PIN_H12 VGA_G[2]
PIN_G10 VGA_G[3]
PIN_G11 VGA_G[4]
PIN_G12 VGA_G[5]
PIN_F11 VGA_G[6]
PIN_E11 VGA_G[7]

PIN_B13 VGA_B[0]
PIN_G13 VGA_B[1]
```

```
        PIN_H13 VGA_B[2]
        PIN_F14 VGA_B[3]
        PIN_H14 VGA_B[4]
        PIN_F15 VGA_B[5]
        PIN_G15 VGA_B[6]
        PIN_J14 VGA_B[7]

        PIN_A11 VGA_CLK
        PIN_B11 VGA_HS
        PIN_D11 VGA_VS
        PIN_F10 VGA_BLANK_N
        PIN_C10 VGA_SYNC_N
    } {
        set_location_assignment $pin -to $port
        set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to $port
    }

    # HPS assignments

    # 3.3-V LVTTL pins
    foreach port {
        HPS_CONV_USB_N
        HPS_ENET_GTX_CLK
        HPS_ENET_INT_N
        HPS_ENET_MDC
        HPS_ENET_MDIO
```

```
HPS_ENET_RX_CLK
HPS_ENET_RX_DATA[0]
HPS_ENET_RX_DATA[1]
HPS_ENET_RX_DATA[2]
HPS_ENET_RX_DATA[3]
HPS_ENET_RX_DV
HPS_ENET_TX_DATA[0]
HPS_ENET_TX_DATA[1]
HPS_ENET_TX_DATA[2]
HPS_ENET_TX_DATA[3]
HPS_ENET_TX_EN
HPS_GSENSOR_INT
HPS_I2C1_SCLK
HPS_I2C1_SDAT
HPS_I2C2_SCLK
HPS_I2C2_SDAT
HPS_I2C_CONTROL
HPS_KEY
HPS_LED
HPS_LTC_GPIO
HPS_SD_CLK
HPS_SD_CMD
HPS_SD_DATA[0]
HPS_SD_DATA[1]
HPS_SD_DATA[2]
HPS_SD_DATA[3]
```

```
    HPS_SPIM_CLK
    HPS_SPIM_MISO
    HPS_SPIM_MOSI
    HPS_SPIM_SS
    HPS_UART_RX
    HPS_UART_TX
    HPS_USB_CLKOUT
    HPS_USB_DATA[0]
    HPS_USB_DATA[1]
    HPS_USB_DATA[2]
    HPS_USB_DATA[3]
    HPS_USB_DATA[4]
    HPS_USB_DATA[5]
    HPS_USB_DATA[6]
    HPS_USB_DATA[7]
    HPS_USB_DIR
    HPS_USB_NXT
    HPS_USB_STP
} {
    set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to $port
}

# There are a lot of settings for the HPS_DDR3 interface not listed here.
# Instead, the
#
# soc_system/synthesis/submodules/hps_sdram_p0_pin_assignments.tcl
```

```
#
# script generated by qsys adds that information.  However, quartus_map
# must be run before this .tcl script may run because the script
# relies on being able to look at the (HPS) netlist to determine which
# pins to constrain

set sdcFilename "${project}.sdc"

set_global_assignment -name SDC_FILE $sdcFilename

set sdcf [open $sdcFilename "w"]
puts $sdcf {
    foreach {clock port} {
       clock_50_1 CLOCK_50
       clock_50_2 CLOCK2_50
       clock_50_3 CLOCK3_50
       clock_50_4 CLOCK4_50
    } {
       create_clock -name $clock -period 20ns [get_ports $port]
    }

    create_clock -name clock_27_1 -period 37 [get_ports TD_CLK27]

    derive_pll_clocks -create_base_clocks
    derive_clock_uncertainty
}
```

```
close $sdcf


project_close
```

soc_system_board_info.xml

```
Unset
	<BoardInfo pov="hps_0_arm_a9_0">
	<!--
	This file is intended to be used when building device trees
	for the Altera Cyclone5 SOC Development Kits.
	This board info file and hps_clock_info.xml are required input
	to sopc2dts to create a device tree suitable for the 3.9 version
	of the Linux kernel.  One typically executes sopc2dts as follows:

		sopc2dts -i soc_system.sopcinfo -b soc_system_board_info.xml
			-b hps_clock_info.xml -o soc_system.dts


	-->

	<DTAppend name="model" type="string" parentlabel="" val="Altera SOCFPGA
Cyclone V"/>
	<DTAppend name="compatible" parentlabel="" >
		<val type="string">altr,socfpga-cyclone5</val>
```

```
              <val type="string">altr,socfpga</val>
       </DTAppend>
       <IRQMasterIgnore className="intr_capturer"/>
       <IRQMasterIgnore className="d_irq"/>

       <DTAppend name="next-level-cache" type="phandle"
parentlabel="hps_0_arm_a9_0" val="hps_0_L2"/>
       <DTAppend name="next-level-cache" type="phandle"
parentlabel="hps_0_arm_a9_1" val="hps_0_L2"/>

       <DTAppend name="cache-unified" type="bool" parentlabel="hps_0_L2"
val="true"/>
       <DTAppend name="arm,tag-latency" parentlabel="hps_0_L2">
              <val type="number">1</val>
              <val type="number">1</val>
              <val type="number">1</val>
       </DTAppend>
       <DTAppend name="arm,data-latency" parentlabel="hps_0_L2">
              <val type="number">2</val>
              <val type="number">1</val>
              <val type="number">1</val>
       </DTAppend>

       <DTAppend name="interrupt-controller" parentlabel="hps_0_gpio0"/>
       <DTAppend name="#interrupt-cells" type="number" parentlabel="hps_0_gpio0"
val="2"/>
```

```
       <DTAppend name="interrupt-controller" parentlabel="hps_0_gpio1"/>
       <DTAppend name="#interrupt-cells" type="number" parentlabel="hps_0_gpio1"
val="2"/>

       <DTAppend name="interrupt-controller" parentlabel="hps_0_gpio2"/>
       <DTAppend name="#interrupt-cells" type="number" parentlabel="hps_0_gpio2"
val="2"/>

       <DTAppend name="clock-frequency" type="number" parentlabel="hps_0_uart0"
val="100000000"/>
       <DTAppend name="clock-frequency" type="number" parentlabel="hps_0_uart1"
val="100000000"/>
       <DTAppend name="clock-frequency" type="number" parentlabel="hps_0_timer0"
val="100000000"/>
       <DTAppend name="clock-frequency" type="number" parentlabel="hps_0_timer1"
val="100000000"/>
       <DTAppend name="clock-frequency" type="number" parentlabel="hps_0_timer2"
val="25000000"/>
       <DTAppend name="clock-frequency" type="number" parentlabel="hps_0_timer3"
val="25000000"/>

       <DTAppend name="cpu1-start-addr" type="hex" parentlabel="hps_0_sysmgr"
val="0xffd080c4"/>
       <DTAppend name="compatible" type="string" parentlabel="hps_0_sysmgr"
val="syscon" action="add"/>
```

```
      <DTAppend name="compatible" type="string" parentlabel="hps_0_rstmgr"
val="syscon" action="add"/>

      <DTAppend name="speed-mode" type="number" parentlabel="hps_0_i2c0"
val="0"/>
      <DTAppend name="status" type="string" parentlabel="hps_0_i2c1"
val="disabled"/>
      <DTAppend name="status" type="string" parentlabel="hps_0_i2c2"
val="disabled"/>
      <DTAppend name="status" type="string" parentlabel="hps_0_i2c3"
val="disabled"/>

      <DTAppend name="phy-mode" type="string" parentlabel="hps_0_gmac1"
val="rgmii"/>
      <DTAppend name="clock-names" type="string" parentlabel="hps_0_gmac1"
val="stmmaceth"/>
      <DTAppend name="clocks" type="phandle" parentlabel="hps_0_gmac1"
val="emac1_clk"/>
      <DTAppend name="phy-addr" type="hex" parentlabel="hps_0_gmac1"
val="0xffffffff"/>

      <DTAppend name="micrel-ksz9021rlrn-clk-skew" type="hex"
parentlabel="hps_0_gmac1" val="0xa0e0"/>
      <DTAppend name="micrel-ksz9021rlrn-rx-skew" type="hex"
parentlabel="hps_0_gmac1" val="0x0"/>
```

```
       <!-- Added by sedwards -->
       <DTAppend name="altr,sysmgr-syscon" parentlabel="hps_0_gmac1">
         <val type="phandle">hps_0_sysmgr</val>
         <val type="hex">0x60</val>
         <val type="hex">0x2</val>
       </DTAppend>
       <DTAppend name="reset-names" type="string" parentlabel="hps_0_gmac1"
val="stmmaceth"/>
       <DTAppend name="resets" parentlabel="hps_0_gmac1">
         <val type="hex">0x21</val>
         <val type="hex">0x21</val>
       </DTAppend>
       <DTAppend name="txd0-skew-ps"  parentlabel="hps_0_gmac1" type="hex"
val="0x0"/>
       <DTAppend name="txd1-skew-ps"  parentlabel="hps_0_gmac1" type="hex"
val="0x0"/>
       <DTAppend name="txd2-skew-ps"  parentlabel="hps_0_gmac1" type="hex"
val="0x0"/>
       <DTAppend name="txd3-skew-ps"  parentlabel="hps_0_gmac1" type="hex"
val="0x0"/>
       <DTAppend name="rxd0-skew-ps"  parentlabel="hps_0_gmac1" type="hex"
val="0x1a4"/>
       <DTAppend name="rxd1-skew-ps"  parentlabel="hps_0_gmac1" type="hex"
val="0x1a4"/>
```

```
      <DTAppend name="rxd2-skew-ps"  parentlabel="hps_0_gmac1" type="hex"
val="0x1a4"/>
      <DTAppend name="rxd3-skew-ps"  parentlabel="hps_0_gmac1" type="hex"
val="0x1a4"/>
      <DTAppend name="txen-skew-ps"  parentlabel="hps_0_gmac1" type="hex"
val="0x0"/>
      <DTAppend name="txc-skew-ps"  parentlabel="hps_0_gmac1" type="hex"
val="0x744"/>
      <DTAppend name="rxdv-skew-ps"  parentlabel="hps_0_gmac1" type="hex"
val="0x1a4"/>
      <DTAppend name="rxc-skew-ps"  parentlabel="hps_0_gmac1" type="hex"
val="0x690"/>




      <DTAppend name="status" type="string" parentlabel="hps_0_gmac0"
val="disabled"/>
      <DTAppend name="status" type="string" parentlabel="hps_0_uart1"
val="disabled"/>
      <DTAppend name="status" type="string" parentlabel="hps_0_usb0"
val="disabled"/>
      <DTAppend name="status" type="string" parentlabel="hps_0_nand0"
val="disabled"/>
      <DTAppend name="clocks" type="phandle" parentlabel="hps_0_nand0"
val="nand_clk"/>
```

```
       <I2CBus master="hps_0_i2c0">
             <I2CChip addr="0x28" label="lcd"
name="newhaven,nhd-0216k3z-nsw-bbw"></I2CChip>
             <I2CChip addr="0x51" label="eeprom" name="atmel,24c32"></I2CChip>
       </I2CBus>
       <DTAppend name="height" type="number" parentlabel="lcd" val="2"/>
       <DTAppend name="width" type="number" parentlabel="lcd" val="16"/>
       <DTAppend name="brightness" type="number" parentlabel="lcd" val="8"/>
       <DTAppend name="pagesize" type="number" parentlabel="eeprom" val="32"/>
       <DTAppend name="clocks"  parentlabel="hps_0_sdmmc" >
       <val type="phandle">l4_mp_clk</val>
       <val type="phandle">sdmmc_clk</val>
       </DTAppend>
       <DTAppend name="clock-names"  parentlabel="hps_0_sdmmc" >
       <val type="string">biu</val>
       <val type="string">ciu</val>
       </DTAppend>
       <DTAppend name="#address-cells" type="number" parentlabel="hps_0_sdmmc"
val="1"/>
       <DTAppend name="#size-cells" type="number" parentlabel="hps_0_sdmmc"
val="0"/>
       <DTAppend name="supports-highspeed"  parentlabel="hps_0_sdmmc" />
       <DTAppend name="broken-cd" type="bool" parentlabel="hps_0_sdmmc"
val="true"/>
```

```
       <DTAppend name="compatible" type="string" parentlabel="hps_0_sdmmc"
val="altr,socfpga-dw-mshc" action="replace"/>
       <DTAppend name="altr,dw-mshc-ciu-div" type="number"
parentlabel="hps_0_sdmmc" val="3"/>
       <DTAppend name="altr,dw-mshc-sdr-timing"  parentlabel="hps_0_sdmmc" >
       <val type="number">0</val>
       <val type="number">3</val>
       </DTAppend>
       <DTAppend name="slot@0" type="node" parentlabel="hps_0_sdmmc"
newlabel="slot_0"/>
       <DTAppend name="reg" type="number" parentlabel="slot_0" val="0"/>
       <DTAppend name="bus-width" type="number" parentlabel="slot_0" val="4"/>

       <DTAppend name="master-ref-clk" type="number" parentlabel="hps_0_qspi"
val="400000000"/>
       <DTAppend name="ext-decoder" type="number" parentlabel="hps_0_qspi"
val="0"/>
       <DTAppend name="n25q00@0" type="node" parentlabel="hps_0_qspi"
newlabel="flash0"/>
       <DTAppend name="#address-cells" type="number" parentlabel="hps_0_qspi"
val="1"/>
       <DTAppend name="#size-cells" type="number" parentlabel="hps_0_qspi"
val="0"/>
       <DTAppend name="#address-cells" type="number" parentlabel="flash0"
val="1"/>
       <DTAppend name="#size-cells" type="number" parentlabel="flash0" val="1"/>
```

```
       <DTAppend name="compatible" type="string" parentlabel="flash0"
val="n25q00"/>
       <DTAppend name="reg" type="number" parentlabel="flash0" val="0"/>
       <DTAppend name="spi-max-frequency" type="number" parentlabel="flash0"
val="100000000"/>
       <DTAppend name="page-size" type="number" parentlabel="flash0" val="256"/>
       <DTAppend name="block-size" type="number" parentlabel="flash0" val="16"/>
       <DTAppend name="m25p,fast-read" type="bool" parentlabel="flash0"
val="true"/>
       <DTAppend name="read-delay" type="number" parentlabel="flash0" val="4"/>
       <DTAppend name="tshsl-ns" type="number" parentlabel="flash0" val="50"/>
       <DTAppend name="tsd2d-ns" type="number" parentlabel="flash0" val="50"/>
       <DTAppend name="tchsh-ns" type="number" parentlabel="flash0" val="4"/>
       <DTAppend name="tslch-ns" type="number" parentlabel="flash0" val="4"/>
       <DTAppend name="partition@0" type="node" parentlabel="flash0"
newlabel="part0"/>
       <DTAppend name="label" type="string" parentlabel="part0" val="Flash 0 Raw
Data"/>
       <DTAppend name="reg" parentlabel="part0" >
       <val type="hex">0x0</val>
       <val type="hex">0x800000</val>
       </DTAppend>
       <DTAppend name="partition@800000" type="node" parentlabel="flash0"
newlabel="part1"/>
       <DTAppend name="label" type="string" parentlabel="part1" val="Flash 1
jffs2 Filesystem"/>
```

```
       <DTAppend name="reg" parentlabel="part1">
       <val type="hex">0x800000</val>
       <val type="hex">0x800000</val>
       </DTAppend>

       <DTAppend name="pmu0" type="node" parentlabel="sopc0" newlabel="pmu"/>
       <DTAppend name="#address-cells" type="number" parentlabel="pmu" val="1"/>
       <DTAppend name="#size-cells" type="number" parentlabel="pmu" val="1"/>
       <DTAppend name="compatible" type="string" parentlabel="pmu"
val="arm,cortex-a9-pmu"/>
       <DTAppend name="interrupt-parent" type="phandle" parentlabel="pmu"
val="hps_0_arm_gic_0"/>
       <DTAppend name="interrupts" parentlabel="pmu">
       <val type="number">0</val>
       <val type="number">176</val>
       <val type="number">4</val>
       <val type="number">0</val>
       <val type="number">177</val>
       <val type="number">4</val>
       </DTAppend>
       <DTAppend name="ranges" type="bool" parentlabel="pmu" val="true"/>

       <DTAppend name="cti0@ff118000" type="node" parentlabel="pmu"
newlabel="cti0"/>
       <DTAppend name="compatible" type="string" parentlabel="cti0"
val="arm,coresight-cti"/>
```

```
       <DTAppend name="reg" parentlabel="cti0">
       <val type="hex">0xff118000</val>
       <val type="hex">0x100</val>
       </DTAppend>

       <DTAppend name="cti0@ff119000" type="node" parentlabel="pmu"
newlabel="cti1"/>
       <DTAppend name="compatible" type="string" parentlabel="cti1"
val="arm,coresight-cti"/>
       <DTAppend name="reg" parentlabel="cti1">
       <val type="hex">0xff119000</val>
       <val type="hex">0x100</val>
       </DTAppend>

       <DTAppend name="fpgabridge@0" type="node" parentlabel="sopc0"
newlabel="fpgabridge0"/>
       <DTAppend name="compatible" type="string" parentlabel="fpgabridge0"
val="altr,socfpga-hps2fpga-bridge"/>
       <DTAppend name="label" type="string" parentlabel="fpgabridge0"
val="hps2fpga"/>
       <DTAppend name="clocks" type="phandle" parentlabel="fpgabridge0"
val="l4_main_clk"/>

       <DTAppend name="fpgabridge@1" type="node" parentlabel="sopc0"
newlabel="fpgabridge1"/>
```

```
       <DTAppend name="compatible" type="string" parentlabel="fpgabridge1"
val="altr,socfpga-lwhps2fpga-bridge"/>
       <DTAppend name="label" type="string" parentlabel="fpgabridge1"
val="lwhps2fpga"/>
       <DTAppend name="clocks" type="phandle" parentlabel="fpgabridge1"
val="l4_main_clk"/>

       <DTAppend name="fpgabridge@2" type="node" parentlabel="sopc0"
newlabel="fpgabridge2"/>
       <DTAppend name="compatible" type="string" parentlabel="fpgabridge2"
val="altr,socfpga-fpga2hps-bridge"/>
       <DTAppend name="label" type="string" parentlabel="fpgabridge2"
val="fpga2hps"/>
       <DTAppend name="clocks" type="phandle" parentlabel="fpgabridge2"
val="l4_main_clk"/>

       <DTAppend name="l3regs@0xff800000" type="node" parentlabel="sopc0"
newlabel="l3regs"/>
       <DTAppend name="compatible"  parentlabel="l3regs" >
       <val type="string">altr,l3regs</val>
       <val type="string">syscon</val>
       </DTAppend>
       <DTAppend name="reg"  parentlabel="l3regs" >
       <val type="hex">0xff800000</val>
       <val type="hex">0x1000</val>
       </DTAppend>
```

```
       <DTAppend name="sdrctl@0xffc25000" type="node" parentlabel="sopc0"
newlabel="sdctrl"/>
       <DTAppend name="compatible"  parentlabel="sdctrl" >
       <val type="string">altr,sdr-ctl</val>
       <val type="string">syscon</val>
       </DTAppend>
       <DTAppend name="reg"  parentlabel="sdctrl" >
       <val type="hex">0xffc25000</val>
       <val type="hex">0x1000</val>
       </DTAppend>



       <Chosen>
              <Bootargs val="console=ttyS0,115200"></Bootargs>
       </Chosen>
       </BoardInfo>
```

soc_system_nativelink_simulation.rpt

```
Unset
       Info: Start Nativelink Simulation process
```

```
       ========== EDA Simulation Settings ======================


       Sim Mode              :  Gate

       Family                :  cyclonev

       Quartus root          :  c:/intelfpga_lite/18.1/quartus/bin64/

       Quartus sim root      :  c:/intelfpga_lite/18.1/quartus/eda/sim_lib

       Simulation Tool       :  modelsim-altera

       Simulation Language   :  systemverilog

       Simulation Mode       :  GUI

       Sim Output File       :  soc_system.svo

       Sim SDF file          :  soc_system__verilog.sdo

       Sim dir               :  simulation\modelsim


       ============================================================


       Info: Starting NativeLink simulation with ModelSim-Altera software

       Sourced NativeLink script
c:/intelfpga_lite/18.1/quartus/common/tcl/internal/nativelink/modelsim.tcl

       Probing transcript

       ModelSim-Altera Info: # Reading
C:/intelFPGA/18.1/modelsim_ase/tcl/vsim/pref.tcl

       ModelSim-Altera Info: # do soc_system_run_msim_gate_systemverilog.do

       ModelSim-Altera Info: # if {[file exists gate_work]} {

       ModelSim-Altera Info: #    vdel -lib gate_work -all

       ModelSim-Altera Info: # }
```

```
       ModelSim-Altera Info: # vlib gate_work
       ModelSim-Altera Info: # vmap work gate_work
       ModelSim-Altera Info: # Model Technology ModelSim - Intel FPGA Edition
vmap 10.5b Lib Mapping Utility 2016.10 Oct  5 2016
       ModelSim-Altera Info: # vmap work gate_work
       ModelSim-Altera Info: # Copying
C:/intelFPGA/18.1/modelsim_ase/win32aloem/../modelsim.ini to modelsim.ini
       ModelSim-Altera Info: # Modifying modelsim.ini
       ModelSim-Altera Info: #
       ModelSim-Altera Info: # vlog -sv -work work +incdir+. {soc_system.svo}
       ModelSim-Altera Info: # Model Technology ModelSim - Intel FPGA Edition
vlog 10.5b Compiler 2016.10 Oct  5 2016
       ModelSim-Altera Info: # Start time: 20:40:34 on Jun 03,2019
       ModelSim-Altera Info: # vlog -reportprogress 300 -sv -work work
"+incdir+." soc_system.svo
       ModelSim-Altera Info: # -- Compiling module soc_system_top
       ModelSim-Altera Info: #
       ModelSim-Altera Info: # Top level modules:
       ModelSim-Altera Info: #    soc_system_top
       ModelSim-Altera Info: # End time: 20:40:36 on Jun 03,2019, Elapsed time:
0:00:02
       ModelSim-Altera Info: # Errors: 0, Warnings: 0
       ModelSim-Altera Info: #
       ModelSim-Altera Info: # vlog -sv -work work
+incdir+C:/Users/nanyu/Dropbox/columbia/2019S/CSEE\ 4840\ Embeded\
```

```
System/Nanyu/GB/GameBoy_RTL_Qsys {C:/Users/nanyu/Dropbox/columbia/2019S/CSEE
4840 Embeded System/Nanyu/GB/GameBoy_RTL_Qsys/TestBench.sv}
       ModelSim-Altera Info: # Model Technology ModelSim - Intel FPGA Edition
vlog 10.5b Compiler 2016.10 Oct  5 2016
       ModelSim-Altera Info: # Start time: 20:40:36 on Jun 03,2019
       ModelSim-Altera Info: # vlog -reportprogress 300 -sv -work work
"+incdir+C:/Users/nanyu/Dropbox/columbia/2019S/CSEE 4840 Embeded
System/Nanyu/GB/GameBoy_RTL_Qsys" C:/Users/nanyu/Dropbox/columbia/2019S/CSEE
4840 Embeded System/Nanyu/GB/GameBoy_RTL_Qsys/TestBench.sv
       ModelSim-Altera Info: # -- Compiling module TestBench
       ModelSim-Altera Info: #
       ModelSim-Altera Info: # Top level modules:
       ModelSim-Altera Info: #    TestBench
       ModelSim-Altera Info: # End time: 20:40:36 on Jun 03,2019, Elapsed time:
0:00:00
       ModelSim-Altera Info: # Errors: 0, Warnings: 0
       ModelSim-Altera Info: #
       ModelSim-Altera Info: # vsim -t 1ps +transport_int_delays
+transport_path_delays -L altera_ver -L altera_lnsim_ver -L cyclonev_ver -L
lpm_ver -L sgate_ver -L cyclonev_hssi_ver -L altera_mf_ver -L
cyclonev_pcie_hip_ver -L gate_work -L work -voptargs="+acc"  TB
       ModelSim-Altera Info: # vsim -t 1ps "+transport_int_delays"
"+transport_path_delays" -L altera_ver -L altera_lnsim_ver -L cyclonev_ver -L
lpm_ver -L sgate_ver -L cyclonev_hssi_ver -L altera_mf_ver -L
cyclonev_pcie_hip_ver -L gate_work -L work -voptargs=""+acc"" TB
       ModelSim-Altera Info: # Start time: 20:40:37 on Jun 03,2019
```

```
       ModelSim-Altera Error: # ** Error: (vsim-3170) Could not find 'TB'.
       ModelSim-Altera Info: #         Searched libraries:
       ModelSim-Altera Info: #
C:/intelFPGA/18.1/modelsim_ase/altera/verilog/altera
       ModelSim-Altera Info: #
C:/intelFPGA/18.1/modelsim_ase/altera/verilog/altera_lnsim
       ModelSim-Altera Info: #
C:/intelFPGA/18.1/modelsim_ase/altera/verilog/cyclonev
       ModelSim-Altera Info: #
C:/intelFPGA/18.1/modelsim_ase/altera/verilog/220model
       ModelSim-Altera Info: #
C:/intelFPGA/18.1/modelsim_ase/altera/verilog/sgate
       ModelSim-Altera Info: #
C:/intelFPGA/18.1/modelsim_ase/altera/verilog/cyclonev_hssi
       ModelSim-Altera Info: #
C:/intelFPGA/18.1/modelsim_ase/altera/verilog/altera_mf
       ModelSim-Altera Info: #
C:/intelFPGA/18.1/modelsim_ase/altera/verilog/cyclonev_pcie_hip
       ModelSim-Altera Info: #
C:/Users/nanyu/Dropbox/columbia/2019S/CSEE 4840 Embeded
System/Nanyu/GB/GameBoy_RTL_Qsys/simulation/modelsim/gate_work
       ModelSim-Altera Info: #
C:/Users/nanyu/Dropbox/columbia/2019S/CSEE 4840 Embeded
System/Nanyu/GB/GameBoy_RTL_Qsys/simulation/modelsim/gate_work
       ModelSim-Altera Info: # Error loading design
       ModelSim-Altera Info: # Error: Error loading design
```

```
       ModelSim-Altera Info: #        Pausing macro execution
       ModelSim-Altera Info: # MACRO ./soc_system_run_msim_gate_systemverilog.do
PAUSED at line 12
       ModelSim-Altera Info: # End time: 16:01:22 on Jun 04,2019, Elapsed time:
19:20:45
       ModelSim-Altera Info: # Errors: 1, Warnings: 0
       Error: Errors encountered while running modelsim do file
       Error: NativeLink simulation flow was NOT successful




       ================The following additional information is provided to help
identify the cause of error while running nativelink scripts=================
       Nativelink TCL script failed with errorCode:  1
       Nativelink TCL script failed with errorInfo:  1
           (procedure "launch_sim" line 1)
           invoked from within
       "launch_sim launch_args_hash"
           ("eval" body line 1)
           invoked from within
       "eval launch_sim launch_args_hash"
           invoked from within
       "if [ info exists ::errorCode ] {
                       set savedCode $::errorCode
                       set savedInfo $::errorInfo
                       error $result $..."
```

```
    invoked from within
"if [catch {eval launch_sim launch_args_hash} result ] {
            set status 1
            if [ info exists ::errorCode ] {
                set save..."
    (procedure "run_sim" line 74)
    invoked from within
"run_sim run_sim_args_hash"
    invoked from within
"if [ info exists ::errorCode ] {
            set savedCode $::errorCode
            set savedInfo $::errorInfo
            error "$result" $savedInfo ..."
    (procedure "run_eda_simulation_tool" line 334)
    invoked from within
"run_eda_simulation_tool eda_opts_hash"
```

soc_system_top.sv

```
Unset
// ====================================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
// ====================================================================
//
```

```
// Modified 2019 by Stephen A. Edwards
//
// Permission:
//
//   Terasic grants permission to use and modify this code for use in
//   synthesis for all Terasic Development Boards and Altera
//   Development Kits made by Terasic.  Other use of this code,
//   including the selling ,duplication, or modification of any
//   portion is strictly prohibited.
//
// Disclaimer:
//
//   This VHDL/Verilog or C/C++ source code is intended as a design
//   reference which illustrates how these types of functions can be
//   implemented.  It is the user's responsibility to verify their
//   design for consistency and functionality through the use of
//   formal verification methods.  Terasic provides no warranty
//   regarding the use or functionality of this code.
//
// ============================================================
//
//  Terasic Technologies Inc

//  9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070.
Taiwan
//
```

```
//
//                      web: http://www.terasic.com/
//                      email: support@terasic.com
module soc_system_top(

 ///////// ADC /////////
 inout              ADC_CS_N,
 output        ADC_DIN,
 input              ADC_DOUT,
 output        ADC_SCLK,

 ///////// CLOCK2 /////////
 input              CLOCK2_50,

 ///////// CLOCK3 /////////
 input              CLOCK3_50,

 ///////// CLOCK4 /////////
 input              CLOCK4_50,

 ///////// CLOCK /////////
 input              CLOCK_50,

 ///////// DRAM /////////
 output [12:0] DRAM_ADDR,
 output [1:0]  DRAM_BA,
```

```
output        DRAM_CAS_N,
output        DRAM_CKE,
output        DRAM_CLK,
output        DRAM_CS_N,
inout [15:0]  DRAM_DQ,
output        DRAM_LDQM,
output        DRAM_RAS_N,
output        DRAM_UDQM,
output        DRAM_WE_N,

//////////// FAN //////////
output        FAN_CTRL,

//////////// FPGA //////////
output        FPGA_I2C_SCLK,
inout              FPGA_I2C_SDAT,

//////////// GPIO //////////
inout [35:0]  GPIO_0,
inout [35:0]  GPIO_1,

//////////// HEX0 //////////
output [6:0]  HEX0,

//////////// HEX1 //////////
output [6:0]  HEX1,
```

```
//////// HEX2 ////////
output [6:0]  HEX2,

//////// HEX3 ////////
output [6:0]  HEX3,

//////// HEX4 ////////
output [6:0]  HEX4,

//////// HEX5 ////////
output [6:0]  HEX5,

//////// HPS ////////
inout              HPS_CONV_USB_N,
output [14:0] HPS_DDR3_ADDR,
output [2:0]  HPS_DDR3_BA,
output        HPS_DDR3_CAS_N,
output        HPS_DDR3_CKE,
output        HPS_DDR3_CK_N,
output        HPS_DDR3_CK_P,
output        HPS_DDR3_CS_N,
output [3:0]  HPS_DDR3_DM,
inout [31:0]  HPS_DDR3_DQ,
inout [3:0]   HPS_DDR3_DQS_N,
inout [3:0]   HPS_DDR3_DQS_P,
```

```
        output        HPS_DDR3_ODT,
        output        HPS_DDR3_RAS_N,
        output        HPS_DDR3_RESET_N,
        input              HPS_DDR3_RZQ,
        output        HPS_DDR3_WE_N,
        output        HPS_ENET_GTX_CLK,
        inout              HPS_ENET_INT_N,
        output        HPS_ENET_MDC,
        inout              HPS_ENET_MDIO,
        input              HPS_ENET_RX_CLK,
        input [3:0]   HPS_ENET_RX_DATA,
        input              HPS_ENET_RX_DV,
        output [3:0]  HPS_ENET_TX_DATA,
        output        HPS_ENET_TX_EN,
        inout              HPS_GSENSOR_INT,
        inout              HPS_I2C1_SCLK,
        inout              HPS_I2C1_SDAT,
        inout              HPS_I2C2_SCLK,
        inout              HPS_I2C2_SDAT,
        inout              HPS_I2C_CONTROL,
        inout              HPS_KEY,
        inout              HPS_LED,
        inout              HPS_LTC_GPIO,
        output        HPS_SD_CLK,
        inout              HPS_SD_CMD,
        inout [3:0]   HPS_SD_DATA,
```

```
output        HPS_SPIM_CLK,
input              HPS_SPIM_MISO,
output        HPS_SPIM_MOSI,
inout              HPS_SPIM_SS,
input              HPS_UART_RX,
output        HPS_UART_TX,
input              HPS_USB_CLKOUT,
inout [7:0]   HPS_USB_DATA,
input              HPS_USB_DIR,
input              HPS_USB_NXT,
output        HPS_USB_STP,

//////////// IRDA ////////////
input              IRDA_RXD,
output        IRDA_TXD,

//////////// KEY ////////////
input [3:0]   KEY,

//////////// LEDR ////////////
output [9:0]  LEDR,

//////////// PS2 ////////////
inout              PS2_CLK,
inout              PS2_CLK2,
inout              PS2_DAT,
```

```
 inout              PS2_DAT2,


 ///////// SW /////////
 input [9:0]   SW,


 ///////// TD /////////
 input              TD_CLK27,
 input [7:0]   TD_DATA,
 input              TD_HS,
 output        TD_RESET_N,
 input              TD_VS,



 ///////// VGA /////////
 output [7:0]  VGA_B,
 output        VGA_BLANK_N,
 output        VGA_CLK,
 output [7:0]  VGA_G,
 output        VGA_HS,
 output [7:0]  VGA_R,
 output        VGA_SYNC_N,
 output        VGA_VS
);


   soc_system soc_system0(
     .clk_clk                       ( CLOCK_50 ),
```

```
        .reset_reset                    ( (~KEY[0] && ~KEY[1]) ),

        .hps_ddr3_mem_a                 ( HPS_DDR3_ADDR ),
        .hps_ddr3_mem_ba                ( HPS_DDR3_BA ),
        .hps_ddr3_mem_ck                ( HPS_DDR3_CK_P ),
        .hps_ddr3_mem_ck_n              ( HPS_DDR3_CK_N ),
        .hps_ddr3_mem_cke               ( HPS_DDR3_CKE ),
        .hps_ddr3_mem_cs_n              ( HPS_DDR3_CS_N ),
        .hps_ddr3_mem_ras_n             ( HPS_DDR3_RAS_N ),
        .hps_ddr3_mem_cas_n             ( HPS_DDR3_CAS_N ),
        .hps_ddr3_mem_we_n              ( HPS_DDR3_WE_N ),
        .hps_ddr3_mem_reset_n           ( HPS_DDR3_RESET_N ),
        .hps_ddr3_mem_dq                ( HPS_DDR3_DQ ),
        .hps_ddr3_mem_dqs               ( HPS_DDR3_DQS_P ),
        .hps_ddr3_mem_dqs_n             ( HPS_DDR3_DQS_N ),
        .hps_ddr3_mem_odt               ( HPS_DDR3_ODT ),
        .hps_ddr3_mem_dm                ( HPS_DDR3_DM ),
        .hps_ddr3_oct_rzqin             ( HPS_DDR3_RZQ ),

        .hps_hps_io_emac1_inst_TX_CLK ( HPS_ENET_GTX_CLK ),
        .hps_hps_io_emac1_inst_TXD0   ( HPS_ENET_TX_DATA[0] ),
        .hps_hps_io_emac1_inst_TXD1   ( HPS_ENET_TX_DATA[1] ),
        .hps_hps_io_emac1_inst_TXD2   ( HPS_ENET_TX_DATA[2] ),
        .hps_hps_io_emac1_inst_TXD3   ( HPS_ENET_TX_DATA[3] ),
        .hps_hps_io_emac1_inst_RXD0   ( HPS_ENET_RX_DATA[0] ),
        .hps_hps_io_emac1_inst_MDIO   ( HPS_ENET_MDIO  ),
```

```
.hps_hps_io_emac1_inst_MDC    ( HPS_ENET_MDC   ),
.hps_hps_io_emac1_inst_RX_CTL ( HPS_ENET_RX_DV ),
.hps_hps_io_emac1_inst_TX_CTL ( HPS_ENET_TX_EN ),
.hps_hps_io_emac1_inst_RX_CLK ( HPS_ENET_RX_CLK ),
.hps_hps_io_emac1_inst_RXD1   ( HPS_ENET_RX_DATA[1]  ),
.hps_hps_io_emac1_inst_RXD2   ( HPS_ENET_RX_DATA[2]  ),
.hps_hps_io_emac1_inst_RXD3   ( HPS_ENET_RX_DATA[3]  ),

.hps_hps_io_sdio_inst_CMD     ( HPS_SD_CMD          ),
.hps_hps_io_sdio_inst_D0      ( HPS_SD_DATA[0]      ),
.hps_hps_io_sdio_inst_D1      ( HPS_SD_DATA[1]      ),
.hps_hps_io_sdio_inst_CLK     ( HPS_SD_CLK          ),
.hps_hps_io_sdio_inst_D2      ( HPS_SD_DATA[2]      ),
.hps_hps_io_sdio_inst_D3      ( HPS_SD_DATA[3]      ),

.hps_hps_io_usb1_inst_D0      ( HPS_USB_DATA[0]     ),
.hps_hps_io_usb1_inst_D1      ( HPS_USB_DATA[1]     ),
.hps_hps_io_usb1_inst_D2      ( HPS_USB_DATA[2]     ),
.hps_hps_io_usb1_inst_D3      ( HPS_USB_DATA[3]     ),
.hps_hps_io_usb1_inst_D4      ( HPS_USB_DATA[4]     ),
.hps_hps_io_usb1_inst_D5      ( HPS_USB_DATA[5]     ),
.hps_hps_io_usb1_inst_D6      ( HPS_USB_DATA[6]     ),
.hps_hps_io_usb1_inst_D7      ( HPS_USB_DATA[7]     ),
.hps_hps_io_usb1_inst_CLK     ( HPS_USB_CLKOUT      ),
.hps_hps_io_usb1_inst_STP     ( HPS_USB_STP         ),
.hps_hps_io_usb1_inst_DIR     ( HPS_USB_DIR         ),
```

```
.hps_hps_io_usb1_inst_NXT     ( HPS_USB_NXT         ),

.hps_hps_io_spim1_inst_CLK    ( HPS_SPIM_CLK  ),
.hps_hps_io_spim1_inst_MOSI   ( HPS_SPIM_MOSI ),
.hps_hps_io_spim1_inst_MISO   ( HPS_SPIM_MISO ),
.hps_hps_io_spim1_inst_SS0    ( HPS_SPIM_SS   ),

.hps_hps_io_uart0_inst_RX     ( HPS_UART_RX     ),
.hps_hps_io_uart0_inst_TX     ( HPS_UART_TX     ),

.hps_hps_io_i2c0_inst_SDA     ( HPS_I2C1_SDAT     ),
.hps_hps_io_i2c0_inst_SCL     ( HPS_I2C1_SCLK     ),

.hps_hps_io_i2c1_inst_SDA     ( HPS_I2C2_SDAT     ),
.hps_hps_io_i2c1_inst_SCL     ( HPS_I2C2_SCLK     ),

.hps_hps_io_gpio_inst_GPIO09  ( HPS_CONV_USB_N ),
.hps_hps_io_gpio_inst_GPIO35  ( HPS_ENET_INT_N ),
.hps_hps_io_gpio_inst_GPIO40  ( HPS_LTC_GPIO ),

.hps_hps_io_gpio_inst_GPIO48  ( HPS_I2C_CONTROL ),
.hps_hps_io_gpio_inst_GPIO53  ( HPS_LED ),
.hps_hps_io_gpio_inst_GPIO54  ( HPS_KEY ),
.hps_hps_io_gpio_inst_GPIO61  ( HPS_GSENSOR_INT ),

 /* VGA Conduit */
```

```
                .vga_vga_r                                              (VGA_R),
                .vga_vga_g                                              (VGA_G),
                .vga_vga_b                                              (VGA_B),
                .vga_vga_clk
(VGA_CLK),
                .vga_vga_hs
(VGA_HS),
                .vga_vga_vs
(VGA_VS),
                .vga_vga_blank_n                                (VGA_BLANK_N),
                .vga_vga_sync_n
(VGA_SYNC_N),

                /* SDRAM Conduit */ /*
                .sdram_addr(DRAM_ADDR),
                .sdram_ba(DRAM_BA),
                .sdram_cas_n(DRAM_CAS_N),
                .sdram_cke(DRAM_CKE),
                .sdram_cs_n(DRAM_CS_N),
                .sdram_dq(DRAM_DQ),
                .sdram_dqm({DRAM_UDQM, DRAM_LDQM}),
                .sdram_ras_n(DRAM_RAS_N),
                .sdram_we_n(DRAM_WE_N),
                .sdram_clk_clk(DRAM_CLK), */

                .gameboy_reset_reset(~KEY[2] && ~KEY[3]),
```

```
        /* red led */
        .ledr_ledr(LEDR)

        /* HEX display */
        //.hex_display_hex0(HEX0),
        //.hex_display_hex1(HEX1),
        //.hex_display_hex2(HEX2),
        //.hex_display_hex3(HEX3),
        //.hex_display_hex4(HEX4),
        //.hex_display_hex5(HEX5),

        /* Button */
        //.button_key(KEY)
        /* audio */
        /*.audio_clk_clk(AUD_XCK),
        .audio_BCLK(AUD_BCLK),
        .audio_DACDAT(AUD_DACDAT),
        .audio_DACLRCK(AUD_DACLRCK),
        .av_config_SDAT(FPGA_I2C_SDAT),
        .av_config_SCLK(FPGA_I2C_SCLK),
        .reset_audio_reset(~KEY[0] && ~KEY[1]) */

    );

     // The following quiet the "no driver" warnings for output
```

```
 // pins and should be removed if you use any of these peripherals


 assign ADC_CS_N = SW[1] ? SW[0] : 1'bZ;
 assign ADC_DIN = SW[0];
 assign ADC_SCLK = SW[0];


// assign AUD_ADCLRCK = SW[1] ? SW[0] : 1'bZ;
// assign AUD_BCLK = SW[1] ? SW[0] : 1'bZ;
// assign AUD_DACDAT = SW[0];
// assign AUD_DACLRCK = SW[1] ? SW[0] : 1'bZ;
 //assign AUD_XCK = SW[0];


 assign FAN_CTRL = SW[0];


 //assign FPGA_I2C_SCLK = SW[0];
 //assign FPGA_I2C_SDAT = SW[1] ? SW[0] : 1'bZ;


 assign GPIO_0 = SW[1] ? { 36{ SW[0] } } : {36{1'bz}};
 assign GPIO_1 = SW[1] ? { 36{ SW[0] } } : {36{1'bz}};


 //assign HEX0 = { 7{ SW[1] } };
 //assign HEX1 = { 7{ SW[2] } };
 //assign HEX2 = { 7{ SW[3] } };
 //assign HEX3 = { 7{ SW[4] } };
 //assign HEX4 = { 7{ SW[5] } };
 //assign HEX5 = { 7{ SW[6] } };
```

```
    assign IRDA_TXD = SW[0];

    //assign LEDR = { 10{SW[7]} };

    assign PS2_CLK = SW[1] ? SW[0] : 1'bZ;
    assign PS2_CLK2 = SW[1] ? SW[0] : 1'bZ;
    assign PS2_DAT = SW[1] ? SW[0] : 1'bZ;
    assign PS2_DAT2 = SW[1] ? SW[0] : 1'bZ;

    assign TD_RESET_N = SW[0];

//   assign {VGA_R, VGA_G, VGA_B} = { 24{ SW[0] } };
//   assign {VGA_BLANK_N, VGA_CLK,
//        VGA_HS, VGA_SYNC_N, VGA_VS} = { 5{ SW[0] } };


endmodule
```

tb_ppu.cpp

```
Unset
#include <iostream>
#include <fstream>
```

```
	#include <verilated.h>
	#include <verilated_vcd_c.h>
	#include "VPPU3.h"

	typedef enum {PPU_H_BLANK, PPU_V_BLANK, PPU_SCAN, PPU_DRAW} PPU_STATES_t;

	#define BG_MAP_1_BASE_ADDR 0x9800
	#define BG_MAP_2_BASE_ADDR 0x9C00

	#define BG_MAP_END_ADDR 0x9FFF

	#define OAM_BASE_ADDR 0xFE00

	#define TILE_BASE_ADDR 0x8000
	#define TILE_END_ADDR  0x97FF

	int main(int argc, const char ** argv, const char ** env)
	{
		int time, exit_code, last_clk, i;
		char tile_1[2], tile_2[2], tile_3, sprite_data[4], OAM_MEM[160],
BG_MAP[2048], TILE_MAP[6144], update_reg;
		VPPU3 *dut;
		std::ofstream f("tb_gen.ppm");

		if (!f.is_open()) {
			std::cerr << "Error opening ppm" << std::endl;
```

```
                exit_code = -1;
                return exit_code;
            }
            f << "P2\n160 144\n4\n";

        last_clk = time = exit_code == 0;

            for (i = 0; i < 1024; i++)
                BG_MAP[i] = i % 2;

            BG_MAP[32] = 1;
            for (i = BG_MAP_2_BASE_ADDR - BG_MAP_1_BASE_ADDR; i < 2048; i++)
                BG_MAP[i] = 3;

            for (i = 0; i < 16; i += 2) {
                TILE_MAP[i] = 0xFF;                                 //
1111_1111
                TILE_MAP[i+1] = 0x00;                               //
0000_0000

                TILE_MAP[(16 * 1) + i] = 0xAA;              // 1010_1010
                TILE_MAP[(16 * 1) + i + 1] = 0x55;          // 0101_0101

                TILE_MAP[(16 * 17) + i] = 0xFF;             // "sprite"
                TILE_MAP[(16 * 17) + i + 1] = 0xFF;
```

```
                TILE_MAP[(16 * 16) + i] = 0xFF;                // "sprite"
                TILE_MAP[(16 * 16) + i + 1] = 0xFF;
            }

            for (i = 0; i < 16; i += 4) {
                TILE_MAP[(16 * 2) + i] = 0x96;                         //
1001_0110
                TILE_MAP[(16 * 2) + i + 1] = 0x69;                     //
0110_1001

                TILE_MAP[(16 * 2) + i + 2] = 0x69;                     //
1001_0110
                TILE_MAP[(16 * 2) + i + 3] = 0x96;                     //
0110_1001

                TILE_MAP[(16 * 3) + i] = 0xAA;                         //
1001_0110
                TILE_MAP[(16 * 3) + i + 1] = 0x55;                     //
0110_1001

                TILE_MAP[(16 * 3) + i + 2] = 0x55;                     //
1001_0110
                TILE_MAP[(16 * 3) + i + 3] = 0xAA;                     //
0110_1001
            }
```

```
              for (i = 0; i < 8; i += 4) {
                     OAM_MEM[0] = 16 + (8 * 3);       // (y-value)        16 +
pos
                     OAM_MEM[1] = 8 + (8 * 0);        // (x-value)         8 +
pos


                     OAM_MEM[4] = 16 + (8 * 1);       // (y-value)        16 +
pos
                     OAM_MEM[5] = 8 + (8 * 2);        // (x-value)         8 +
pos


                     OAM_MEM[i+2] = 0x11;                          // (tile no.)
                     OAM_MEM[7] = 0xFF >> 1;                 // flags (prio and
other things)
              }


              Verilated::commandArgs(argc, argv);


              dut = new VPPU3;    // Instantiate the ppu module


              Verilated::traceEverOn(true);
              VerilatedVcdC *tfp = new VerilatedVcdC;


              dut->trace(tfp, 99);
              tfp->open("ppu3.vcd");
```

```
              dut->PPU_DATA_in = 0x0;    // JUNK


              for (time = 0 ; time < 2272110 ; time += 10) {
                     // Simulate a 50 MHz clock
                     last_clk = dut->clk;
       // used to detect posedge
                     dut->clk = ((time % 20) >= 10) ? 1 : 0;


                     // Triggers rst
                     dut->rst = (time == 30) ? 1 : 0;


                     // only on posedge
                     if (dut->clk == 1) {
                            if (time > 40) { // after rst
                                   if (!(update_reg & 0x1)) {                     //
LCDC
                                                 dut->WR = 1;
                                                 dut->ADDR = 0xFF40;
                                                 dut->MMIO_DATA_out = 0xF3;
// 1111_0011
                                                 // dut->MMIO_DATA_out = 0xFF;
                                                 update_reg |= 0x1;
                                   } else if (!(update_reg & 0x02)) {      // WY
                                          dut->WR = 1;
                                          dut->ADDR = 0xFF4A;
                                          dut->MMIO_DATA_out = 8 * 8;
```

```
                                update_reg |= 0x2;
                        } else if (!(update_reg & 0x04)) {      // WX
                                dut->WR = 1;
                                dut->ADDR = 0xFF4B;
                                dut->MMIO_DATA_out = 7 + 16;
                                update_reg |= 0x4;
                        } else
                                dut->WR = 0;
                }



                        if (dut->PPU_ADDR >= OAM_BASE_ADDR && dut->PPU_ADDR <
OAM_BASE_ADDR + 160)
                                dut->PPU_DATA_in = OAM_MEM[dut->PPU_ADDR -
OAM_BASE_ADDR];
                        else if (dut->PPU_ADDR >= BG_MAP_1_BASE_ADDR &&
dut->PPU_ADDR <= BG_MAP_END_ADDR)
                                dut->PPU_DATA_in = BG_MAP[dut->PPU_ADDR -
BG_MAP_1_BASE_ADDR];
                        else if (dut->PPU_ADDR >= TILE_BASE_ADDR &&
dut->PPU_ADDR <= TILE_END_ADDR)
                                dut->PPU_DATA_in = TILE_MAP[dut->PPU_ADDR -
TILE_BASE_ADDR];
                }
```

```
            dut->eval();                        // Run the simulation for a cycle
            tfp->dump(time);                    // Write the VCD file for this
cycle

            if (dut->clk != last_clk && dut->clk == 1) {   // on posedge of
clock
                  /* Writes to ppm file */
                  if ((int)dut->PX_valid == 1)
                        f << (int) dut->PX_OUT << " ";
            }
          }

            tfp->close(); // Stop dumping the VCD file
            delete tfp;

            dut->final(); // Stop the simulation
            delete dut;

            f.close();
            return exit_code;
      }
```

tile_map.h

```
Unset
	#include <stdint.h>


	#define GB_TEST_FRAME_COUNT 4

	#define GB_TEST_FRAME_WIDTH 8

	#define GB_TEST_FRAME_HEIGHT 8


	/* Piskel data for "GB_TEST" */


	//tiles

	static const uint8_t tile_map[360] = {

	2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,

	2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,3,3,2,2,

	2,2,2,3,3,2,2,2,2,2,2,2,2,2,2,3,3,3,3,2,

	2,2,3,3,3,3,2,2,2,2,2,2,2,2,2,3,3,3,2,

	2,2,2,3,3,2,2,2,2,2,2,2,2,2,2,2,3,3,2,

	2,2,2,2,2,2,2,2,2,2,2,3,3,3,2,2,2,2,2,2,

	2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,2,2,2,2,2,

	2,2,2,3,2,2,2,2,2,2,2,3,3,3,2,2,2,2,2,2,

	2,2,3,3,3,3,2,2,2,2,2,2,2,2,2,2,2,2,2,2,

	2,2,3,3,3,3,2,2,2,2,2,2,2,2,2,2,2,2,2,2,

	2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,

	2,2,2,2,2,2,2,2,2,2,2,0,2,2,2,2,2,2,2,2,

	2,2,2,2,2,2,2,2,2,2,0,1,0,2,2,2,2,2,2,2,

	0,0,0,0,0,2,2,0,0,0,1,1,1,0,0,0,0,0,0,0,

	1,1,1,1,1,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,

	1,1,1,1,1,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,
```

```
1,1,1,1,1,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1};

static const uint8_t tile_data[4][64] = {
{
2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2,
1, 2, 0, 2, 2, 1, 2, 1,
0, 0, 1, 2, 2, 0, 2, 0,
0, 1, 0, 2, 2, 0, 2, 0,
0, 0, 0, 0, 2, 0, 0, 1,
0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0
},
{
0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 1,
0, 0, 0, 1, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0,
0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0
},
{
2, 2, 2, 2, 2, 2, 2, 2,
```

```
2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2
},
{
3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3
}
};
```