Columbia University

Embedded Systems CSEE 4840

Final Report

Chip8 Emulator

Authors:
Ashley Kling
Levi Oliver
Gabrielle Taylor
David Watkins

Supervisor:
Stephen A. Edwards

May 12, 2016

Contents

Chapter 1

Introduction

Chip-8 is an interpreted programming language from the 1970s. It ran on the COSMAC VIP, and supported many programs such as Pac-Man, Pong, Space Invaders, and Tetris. We aim to create a processor using SystemVerilog and the FPGA on the SoCKit board that runs these programs. During the boot process of the processor chip8 ROM files will be transferred onto the main memory of the processor. The processor will also allow for save states and restoring of states. The processor will handle keyboard inputs and output graphics and sound.

Figure 1.1: A popular Chip8 game: Pong

Chapter 2

System Overview

2.1 Memory Overview

The Chip-8 specification requires the use of sixteen 8-bit registers (V0-VF), a 16-bit index register, a 64-byte stack with 8-bit stack pointer, an 8-bit delay timer, an 8-bit sound timer, a 64x32 bit frame buffer, and a 16-bit program counter. The Chip8 specification also supported 4096 bytes of addressable memory. All of the supported programs will start at memory location 0x200.

- The sound and delay timers sequentially decrease at a rate of 1 per tick of a 60Hz clock. When the sound timer is above 0, the sound will play as a single monotone beep.
- The framebuffer is an (x, y) addressable memory array that designates whether a pixel is currently on or off. This will be implemented with a write address, an (x, y) position, a offset in the x direction, and an 8-bit group of pixels to be drawn to the screen.
- The return address stack stores previous program counters when jumping into a new routine.
- The VF register is frequently used for storing carry values from a subtraction or addition action, and also specifies whether a particular pixel is to be drawn on the screen.

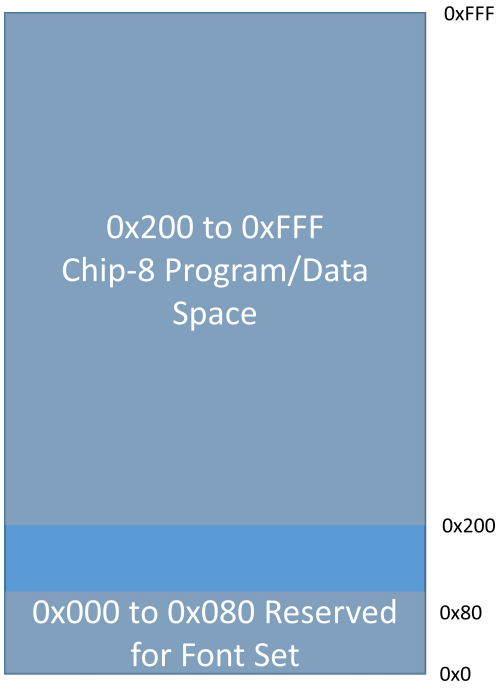


Figure 2.1: Chip8 4K memory layout

2.2 Graphics

Important to the specification is the 64x32 pixel display that is associated with the Chip8. Each pixel only contains the information as to whether it is on or off. All setting of pixels of this display are done through the use of sprites that are always $8 \times N$ where N is the pixel height of the sprite.
Chip8 comes with a font set (sprites) that allows character 0-9 and A-F to be printed directly to the screen. Each one of these characters fit within a 8x5 grid. [1]

2.2.1 Hardware Software Interface

To start running a program, the function loadROM and loadfontset (located in chip8.c) are called. The latter writes the fontset to the Chip8 memory,

[1] http://devernay.free.fr/hacks/chip8/C8TECH10.HTM

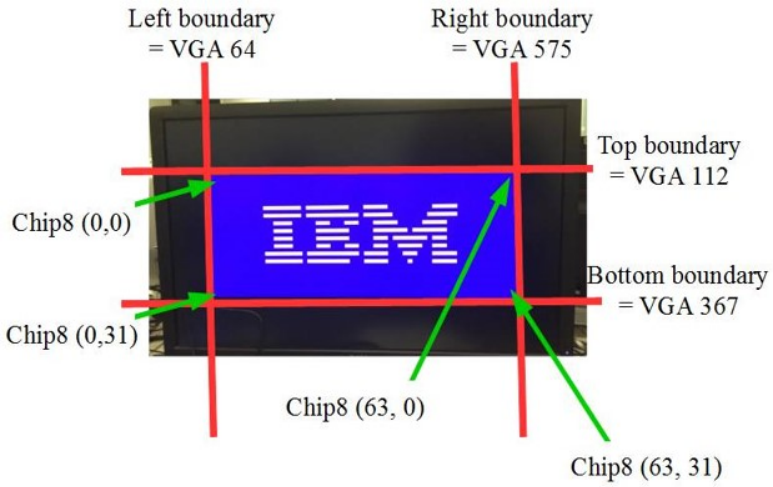


Figure 2.2: The screen dimensions for the VGA output

while the former writes a chip8 program (like Pong!) to the chip8 memory.

From the software side, loadfontset copies the pre-defined font set from linux memory to Chip8 memory by writing byte by byte to processor memory, starting from chip8 memory address 0. loadROM copies a chip8 program from the linux memory to the Chip8 memory byte by byte starting at chip8 memory address 0x200, which is the standard initial address for Chip8 programs.

A 51-bit channel exists that writes from the linux side to the hardware. This channel is broken into two parts: an 18-bit channel that is used to tell the chip what type memory to modify or read (stack, memory, registers, etc.), a 32 bit data channel that includes whatever data is needed (writedata, write enable, and sometimes addressing), and a one-bit channel signifying if the call is a read or a write..

From the hardware side, if chipselect goes high, the processor reads data from the bus. It takes in the 18-bit channel (called address), and the 32-bit channel (called writedata). The stage is frozen while input is processed. The address channel tells what needs to be modified. For example, if address==0x19, the processor begins to modify memory. If the write channel is high, it starts to write the data signified by bits writedata[7:0] to memory addressed by writedata[18:9]. (See Chip8_Top line 348.)

As soon as chipselect goes low again, we resume normal operation of the

"8"	Binary	Hex
****	11110000	0xF0
* *	10010000	0x90
****	11110000	0xF0
* *	10010000	0x90
****	11110000	0xF0

"9"	Binary	Hex
****	11110000	0xF0
* *	10010000	0x90
****	11110000	0xF0
*	00010000	0x10
****	11110000	0xF0

"0"	Binary	Hex
****	11110000	0xF0
* *	10010000	0x90
* *	10010000	0x90
* *	10010000	0x90
****	11110000	0xF0

"1"	Binary	Hex
*	00100000	0x20
**	01100000	0x60
*	00100000	0x20
*	00100000	0x20
***	01110000	0x70

"A"	Binary	Hex
****	11110000	0xF0
* *	10010000	0x90
****	11110000	0xF0
* *	10010000	0x90
* *	10010000	0x90

"B"	Binary	Hex
***	11100000	0xE0
* *	10010000	0x90
***	11100000	0xE0
* *	10010000	0x90
***	11100000	0xE0

"2"	Binary	Hex
****	11110000	0xF0
*	00010000	0x10
****	11110000	0xF0
*	10000000	0x80
****	11110000	0xF0

"3"	Binary	Hex
****	11110000	0xF0
*	00010000	0x10
****	11110000	0xF0
*	00010000	0x10
****	11110000	0xF0

"C"	Binary	Hex
****	11110000	0xF0
*	10000000	0x80
*	10000000	0x80
*	10000000	0x80
****	11110000	0xF0

"D"	Binary	Hex
***	11100000	0xE0
* *	10010000	0x90
* *	10010000	0x90
* *	10010000	0x90
***	11100000	0xE0

"4"	Binary	Hex
* *	10010000	0x90
* *	10010000	0x90
****	11110000	0xF0
*	00010000	0x10
*	00010000	0x10

"5"	Binary	Hex
****	11110000	0xF0
*	10000000	0x80
****	11110000	0xF0
*	00010000	0x10
****	11110000	0xF0

"E"	Binary	Hex
****	11110000	0xF0
*	10000000	0x80
****	11110000	0xF0
*	10000000	0x80
****	11110000	0xF0

"F"	Binary	Hex
****	11110000	0xF0
*	10000000	0x80
****	11110000	0xF0
*	10000000	0x80
*	10000000	0x80

"6"	Binary	Hex
****	11110000	0xF0
*	10000000	0x80
****	11110000	0xF0
* *	10010000	0x90
****	11110000	0xF0

"7"	Binary	Hex
****	11110000	0xF0
*	00010000	0x10
*	00100000	0x20
*	01000000	0x40
*	01000000	0x40

Figure 2.3: Chip8 character sprite specification

processor, restoring all possibly changed values to what they were previously.

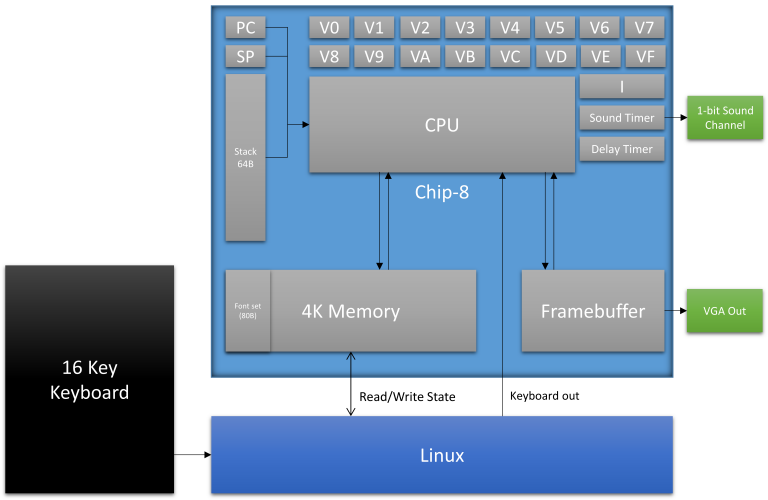


Figure 2.4: Overview of the design of the emulator

2.3 Op Codes

The Chip8 interpreter works by parsing 16 bit opcodes and operating on the data. All supported op codes in the original Chip8 specification are included.

2.3.1 0nnn - SYS addr

Jump to a machine code routine at nnn. This instruction is only used on the old computers on which Chip-8 was originally implemented. It is ignored by modern interpreters. This will not be implemented.

2.3.2 00E0 - CLS

Clear the display.

2.3.3 00EE - RET

Return from a subroutine.The interpreter sets the program counter to the address at the top of the stack, then subtracts 1 from the stack pointer.

2.3.4 1nnn - JP addr

Jump to location nnn. The interpreter sets the program counter to nnn.

2.3.5 2nnn - CALL addr

Call subroutine at nnn. The interpreter increments the stack pointer, then puts the current PC on the top of the stack. The PC is then set to nnn.

2.3.6 3xkk - SE Vx, byte

Skip next instruction if Vx = kk. The interpreter compares register Vx to kk, and if they are equal, increments the program counter by 2.

2.3.7 4xkk - SNE Vx, byte

Skip next instruction if Vx != kk. The interpreter compares register Vx to kk, and if they are not equal, increments the program counter by 2.

2.3.8 5xy0 - SE Vx, Vy

Skip next instruction if Vx = Vy. The interpreter compares register Vx to register Vy, and if they are equal, increments the program counter by 2.

2.3.9 6xkk - LD Vx, byte

Set Vx = kk. The interpreter puts the value kk into register Vx.

2.3.10 7xkk - ADD Vx, byte

Set Vx = Vx + kk. Adds the value kk to the value of register Vx, then stores the result in Vx.

2.3.11 8xy0 - LD Vx, Vy

Set Vx = Vy. Stores the value of register Vy in register Vx.

2.3.12 8xy1 - OR Vx, Vy

Set Vx = Vx OR Vy. Performs a bitwise OR on the values of Vx and Vy, then stores the result in Vx. A bitwise OR compares the corresponding bits from two values, and if either bit is 1, then the same bit in the result is also 1. Otherwise, it is 0.

2.3.13 8xy2 - AND Vx, Vy

Set Vx = Vx AND Vy. Performs a bitwise AND on the values of Vx and Vy, then stores the result in Vx. A bitwise AND compares the corresponding bits from two values, and if both bits are 1, then the same bit in the result is also 1. Otherwise, it is 0.

2.3.14 8xy3 - XOR Vx, Vy

Set Vx = Vx XOR Vy. Performs a bitwise exclusive OR on the values of Vx and Vy, then stores the result in Vx. An exclusive OR compares the corresponding bits from two values, and if the bits are not both the same, then the corresponding bit in the result is set to 1. Otherwise, it is 0.

2.3.15 8xy4 - ADD Vx, Vy

Set Vx = Vx + Vy, set VF = carry. The values of Vx and Vy are added together. If the result is greater than 8 bits (i.e., ¿ 255,) VF is set to 1, otherwise 0. Only the lowest 8 bits of the result are kept, and stored in Vx.

2.3.16 8xy5 - SUB Vx, Vy

Set Vx = Vx - Vy, set VF = NOT borrow. If Vx ¿ Vy, then VF is set to 1, otherwise 0. Then Vy is subtracted from Vx, and the results stored in Vx.

2.3.17 8xy6 - SHR Vx {, Vy}

Set Vx = Vx SHR 1. If the least-significant bit of Vx is 1, then VF is set to 1, otherwise 0. Then Vx is divided by 2.

2.3.18 8xy7 - SUBN Vx, Vy

Set Vx = Vy - Vx, set VF = NOT borrow. If Vy ¿ Vx, then VF is set to 1, otherwise 0. Then Vx is subtracted from Vy, and the results stored in Vx.

2.3.19 8xyE - SHL Vx {, Vy}

Set Vx = Vx SHL 1. If the most-significant bit of Vx is 1, then VF is set to 1, otherwise to 0. Then Vx is multiplied by 2.

2.3.20 9xy0 - SNE Vx, Vy

Skip next instruction if Vx != Vy. The values of Vx and Vy are compared, and if they are not equal, the program counter is increased by 2.

2.3.21 Annn - LD I, addr

Set I = nnn. The value of register I is set to nnn.

2.3.22 Bnnn - JP V0, addr

Jump to location nnn + V0. The program counter is set to nnn plus the value of V0.

2.3.23 Cxkk - RND Vx, byte

Set Vx = random byte AND kk. The interpreter generates a random number from 0 to 255, which is then ANDed with the value kk. The results are stored in Vx. See instruction 8xy2 for more information on AND.

2.3.24 Dxyn - DRW Vx, Vy, nibble

Display n-byte sprite starting at memory location I at (Vx, Vy), set VF = collision. The interpreter reads n bytes from memory, starting at the address stored in I. These bytes are then displayed as sprites on screen at coordinates (Vx, Vy). Sprites are XOR'd onto the existing screen. If this causes any pixels to be erased, VF is set to 1, otherwise it is set to 0. If the sprite is positioned so part of it is outside the coordinates of the display, it wraps around to the opposite side of the screen.

2.3.25 Ex9E - SKP Vx

Skip next instruction if key with the value of Vx is pressed. Checks the keyboard, and if the key corresponding to the value of Vx is currently in the down position, PC is increased by 2.

2.3.26 ExA1 - SKNP Vx

Skip next instruction if key with the value of Vx is not pressed. Checks the keyboard, and if the key corresponding to the value of Vx is currently in the up position, PC is increased by 2.

2.3.27 Fx07 - LD Vx, DT

Set Vx = delay timer value. The value of DT is placed into Vx.

2.3.28 Fx0A - LD Vx, K

Wait for a key press, store the value of the key in Vx. All execution stops until a key is pressed, then the value of that key is stored in Vx.

2.3.29 Fx15 - LD DT, Vx

Set delay timer = Vx. Delay Timer is set equal to the value of Vx.

2.3.30 Fx18 - LD ST, Vx

Set sound timer = Vx. Sound Timer is set equal to the value of Vx.

2.3.31 Fx1E - ADD I, Vx

Set I = I + Vx. The values of I and Vx are added, and the results are stored in I.

2.3.32 Fx29 - LD F, Vx

Set I = location of sprite for digit Vx. The value of I is set to the location for the hexadecimal sprite corresponding to the value of Vx. See section 2.4,

Display, for more information on the Chip-8 hexadecimal font. To obtain this value, multiply VX by 5 (all font data stored in first 80 bytes of memory).

2.3.33 Fx33 - LD B, Vx

Store BCD representation of Vx in memory locations I, I+1, and I+2. The interpreter takes the decimal value of Vx, and places the hundreds digit in memory at location in I, the tens digit at location I+1, and the ones digit at location I+2.

2.3.34 Fx55 - LD [I], Vx

Stores V0 to VX in memory starting at address I. I is then set to I + x + 1.

2.3.35 Fx65 - LD Vx, [I]

Fills V0 to VX with values from memory starting at address I. I is then set to I + x + 1.

2.4 Keyboard Input

The keyboard input was a 16-key keyboard with keys $0-9, A-F$. There are a series of op codes (listed in the previous section) that use these key presses. In the design associated with this emulator, the keyboard input will be read in from the ARM processor running Linux and streamed to the emulator.

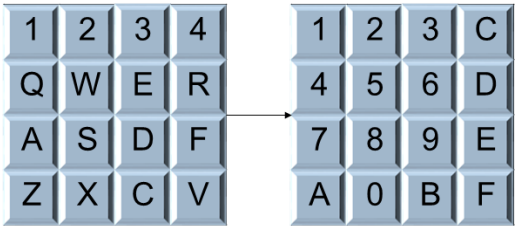


Figure 2.5: Chip8 16-key keyboard specification

For this project we also added keyboard input for pausing, starting, and resetting the device. The "P" key is mapped to pause, the "O" key is mapped to reset, and the "Enter" key is mapped to run.

2.5 Sound

Chip-8 provides 2 timers, a delay timer and a sound timer. The delay timer is active whenever the delay timer register (DT) is non-zero. This timer does nothing more than subtract 1 from the value of DT at a rate of 60Hz. When DT reaches 0, it deactivates.
The sound timer is active whenever the sound timer register (ST) is non-zero. This timer also decrements at a rate of 60Hz, however, as long as ST's value is greater than zero, the Chip-8 buzzer will sound. When ST reaches zero, the sound timer deactivates.
The output of the sound generator has one tone. In the following implementation it will have a soft tone so as to not aggravate the user. [2]

2.6 Screenshots

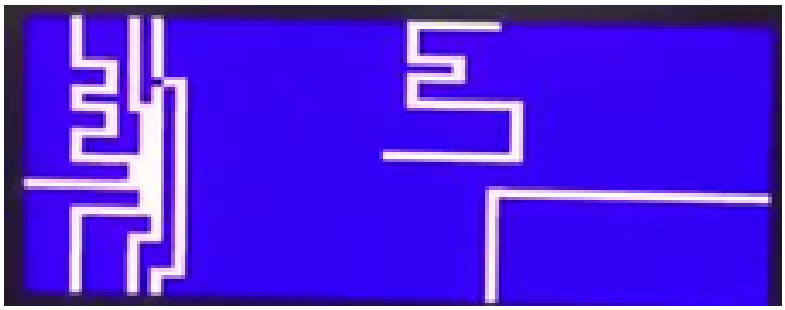


Figure 2.6: Tapeworm game running on the device

[2]http://devernay.free.fr/hacks/chip8/C8TECH10.HTM

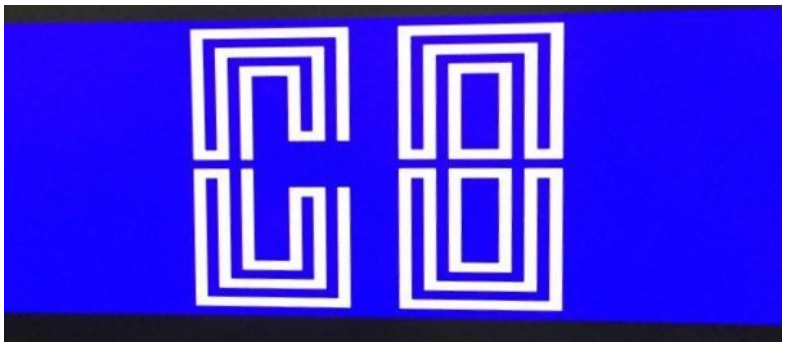


Figure 2.7: Chip8 logo ROM running on the device

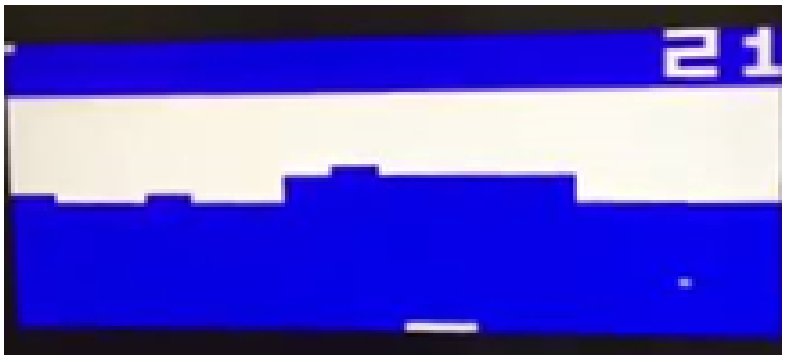


Figure 2.8: Brick game running on the device

Chapter 3

Hardware Design

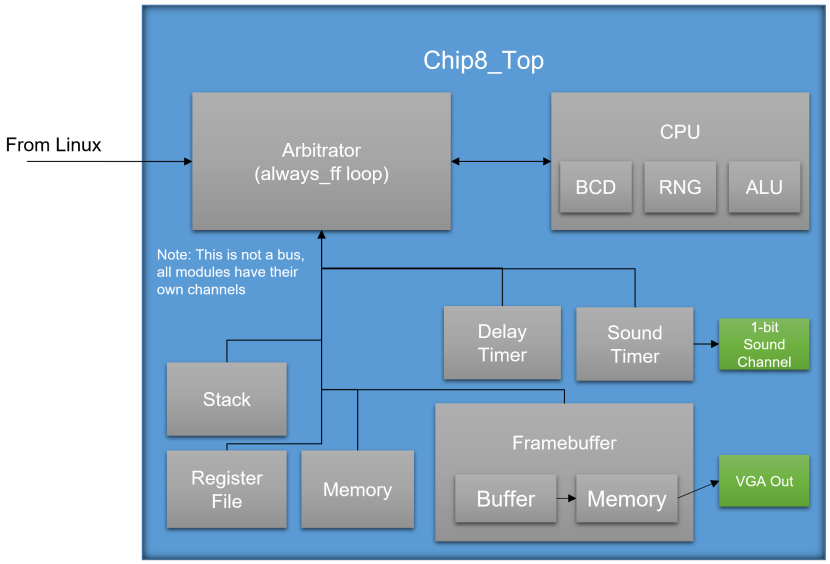


Figure 3.1: Overview of the hardware design

3.1 CPU, Top and Module Access Control

Chip8_Top.sv handles input from the linux side and its effects on the hardware. The linux side reads from and writes to the hardware on startup and requires access to the register file, as well as the memory. The CPU also

requires access to memory and the register file. To deal with the multiple pieces requiring access to multiple modules, Chip8_Top.sv handles arbitration, deciding what gets access to what and when. It controls access to the framebuffer, memory, register file, sound controller, delay timers, and the stack. Access to all of these are required by both the C code, as well as the CPU, so all requests are put through Chip8_Top.

Chip8_Top is mostly a large arbitration unit, but it also has some functionality. It contains the I-register (see section Memory and Register File), which is important in branching, jumping, and draw-sprite instructions. It also manages the program counter and instruction loading. It is important to note that instructions are 16 bits each, so each time the program counter increments by a single instruction, the value of the program counter increments at 2. Since Chip8 programs start at address 0x200, the program counter starts at this value.

The CPU (Chip8_CPU.sv) is completely combinational. It has no latches and is contained in a very large always_comb block. However, it does not execute any instructions in a single 50MHz clock cycle. We cut the effective cycle time of the chip down from 50MHz down to 1kHz, as is appropriate for real Chip8 systems. We made a variable in Chip8_Top.sv called stage. stage increments to a value of 50,000 before it resets to zero.

When stage is 0 or 1, the processor is loading the next instruction to execute from memory. When stage is greater 1, the processor is executing the instruction. The instruction that is using the most stages is instruction 0x00E0–clear screen, which keeps working until stage is 8189 (which is approximately 2048 ¡¡ 2). While this large value of stage is not necessary, it uses a minimal amount of logic for this instruction. During most values of stage, nothing is actually happening other than waiting.

As stated previously, the CPU is completely combinational. Many instructions are broken apart into a few steps. For example, 0x7XKK first has a 4-cycle period of waiting for data to come in, and then a one-cycle period of writing the correct output. When an instruction is loaded and being executed, Chip8_CPU.sv sets all pins to the correct values and Chip8_Top.sv connects those pins to the appropriate destinations.

For example, say we are in stage 40,000. No instructions run during this period. Say the program counter is equal to 0x2C2. The next instruction is in memory at address 0x2C4, and that instruction is equal to 0x63E1. At stage 50,000, the program counter is set equal to 0x2C4. Then, stage is reset to 0. During stages 0 and 1, the instruction is set equal to the values in memory

at 0x2C4 and 0x2C5. In this case, that instruction is 0x63E1. This is the 0x6XKK instruction, which sets Vx = kk. See the section on opcodes to read more about what this instruction does. During stages 2 and 3, the CPU will set reg_addr1 to 0x3, and reg_writedata1 to 0xE1. It will also set reg_WE1 high to enable writing. These values will be picked up by Chip8_Top and directed to the register file to assign the value 0xE1 to register V3.

Chip8_CPU has direct access to the random number generator, the binary-coded-decimal converter, and the ALU, all of which take in and output values combinationally.

3.2 ALU

In order to reduce the number of operations that the CPU used, we implemented an ALU that has functions for addition, subtraction, AND, OR, XOR, left shift by 1, and right shift by 1. The ALU takes in two 16 bit inputs, and results are truncated when needed by the CPU.

3.3 Random Number Generator

The random number generator is a 16 bit random number output that starts with an initial value and does a naive xor loop over the 16 bit number on each clock cycle. This is critical for the CXKK instruction which requires a random number anded with the KK byte.

3.4 BCD

The binary to BCD conversion was was implemented combinatorially using a sequence of bit shifts to extract the ones, tens and hundreds place from an 8 bit binary number. The algorithm works by shifting the binary encoding of the number to the left and then determining as each bit is shifted if the value is too large for a BCD digit (between 0 and 9). This occurs because with each successive left shift, the original binary value is doubled. This is mitigated by checking the value of each BCD digit before shifting. If the value of a BCD digit is greater than 4, then 3 is added to carry the value over into the next digit.

3.5 Graphics and Framebuffer

The screen of the original Chip8 system was 64x32 (which is 2048 pixels). It uses sprite-based drawing, but in a slightly strange way–it draws images by XORing the value of what is to be drawn with the value of the existing display. This is interesting and useful–if a sprite is to be erased, one would simply draw the sprite again in the exact location. However, this rapid drawing and erasing leads to rapidly blinking displays.

To actually display the screen, we adapted Professor Edwards's VGA_LED_Emulator code. The screen displayed is centered in the VGA monitor, with each bit of framebuffer memory representing an eight-by-eight square of pixels (see figure below). Our adaptation of the VGA_LED_Emulator uses simple arithmetic and bit shifting to make sure that each VGA coordinate is properly translated to the correct location in framebuffer memory.

Figure 3.2: Double buffering diagram for the framebuffer

To reduce the amount of blinking, we implemented a double-buffered framebuffer. We used the pre-built Quartus MegaFunction Wizard to create a 2048-bit dual-ported memory file with bit-level granularity, which we used twice. Using the Wizard greatly reduced compile time. The first of these was written to by the CPU whenever the draw command was called. There was only a single opcode that ever wrote to the framebuffer: opcode 0xDXYN (see section on opcodes for more details). The other memory file was connected to the display, which was constantly requesting data to draw. The buffer attached to the VGA Emulator would only have data copied into it 200,000 cycles (4 milliseconds at our 50 MHz clock frequency) after a draw command was given. However, this could give rise to more problems. Oftentimes, programs have looping draw commands. If draw commands happen more than once every 200,000 cycles, data would never be copied over. To combat this, we added an override: if the time since the last draw ever exceeded 500,000 cycles (10ms), we would force the data to copy over. The draw sprite instruction (0xDXYN) specifies the coordinates that the sprites are supposed to be drawn at in (x-coord, y-coord) format. Because this was the only type of addressing the CPU would do, the framebuffer takes in an x and

y coordinate, and writes to the corresponding location in memory. In the actual memory, bits 0 to 63 represent the first row, 64 to 127 represent the second row, 128 to 193 represent the third row, and so on.

The memory file generated by the Wizard does not have combinational reads–all data coming from a read command comes at the rising edge of the next clock cycle. Because of this, when copying data from the CPU-side memory to the screen-side memory, every address requested by the screen-side memory had to be offset by one. We requested data from address 0 to 2047, but every address that data was requested from was always one more than the address being written to. This completely copies the entire memory, including all edges and corners.

Relevant code: `Chip8_VGA_Emulator.sv, Chip8_Framebuffer.sv, Framebuffer.v, enums.svh`

3.6 Memory and Registers

The original Chip8 system used 4096 (0xfff) bytes of memory. Addresses 0 to 511 (0x1ff) were reserved for the interpreter. The first 80 bytes of memory were generally reserved for the fontset (one 5-byte character for each hexadecimal character 0-F) for the display. Address 512 (0x200) was the start of most Chip8 programs. From there, all the way up to the end of memory, the program could use as much space as needed.

The register file was composed of 16 1-byte registers. They are called V0 through VF. V0 through VE are used for general purpose computing, while VF is generally reserved for flags. For example, VF is set to 1 if a certain add instruction (opcode 0x8XY4) overflows past a single byte, and 0 if not. VF is also used to tell if the draw instruction (0xDXYN) has erased any bits.

The Chip8 memory file is very large. We decided to use the MegaFunction Wizard to create our memory file. While it is not the most adaptable, it was much faster than using inferred memory. It is dual-ported. While no instructions actually involve reading from or writing to more than one memory address, we still decided to use a dual-ported memory. This left us with a channel open to send data to the linux side. Since instructions are 16 bits, and each entry is 8 bits, this also allowed us to request both bytes of an instruction in a single cycle.

Because many instructions require reading from two registers at the same time, we used a dual-ported memory for the register file. This allowed us to

get both values simultaneously, as well as write back to the register file (for example, instructions like 0x8XY4: Vx = Vx + Vy).

One very important fact to note regarding these memory modules is that they do not have combinational reads. When an address is set, the output only reflects this change an entire cycle afterwards.

Chip8 also uses a special 16-bit register called the I-register. This register was simply declared as a logic[15:0] variable in Chip8_Top.sv.

3.7 Sound

Sound was implemented writing an audio codec that implements the Inter-Integrated Circuit (I^2C) Protocol. The I^2C controller and I^2C configuration modules were implemented in order to correctly interface with the audio hardware on the FPGA. An audio codec contained the necessary clocks used to drive the output, in particular a phase locked loop (PLL) generated using the Quartus Megawizard for the Master clock, which had to operate at a frequency of 11.2896 MHz and was not easily or accurately approximated using a clock divider. The samples for the 440 Hz sine wave were stored in memory on the chip. Since the only sound required is a single beep, this was sufficient. The top level module assigned the remaining signals.

3.8 Return Address Stack and Program Counter

The program counter is kept as a register (a register–not a reg datatype) in Chip8_Top.sv. We calculate what the next program counter should be in stage 12. (See NEXT_PC_WRITE_STAGE in enums.svh.) This is a somewhat arbitrary value. The stage needed to be late enough so that the processor would have enough time to calculate what the next instruction should be. Whenever the stage reaches 12, we calculate what the next PC should be. By default, next_pc is set to pc+2. Since the memory granularity is a single byte, and instructions are two bytes each, pointing the program counter to the next instruction involves incrementing the program counter by 2. However, most programs involve branches, jumps, and subroutine calls. For that, we have an enum declared in enums.svh. Depending on the instruction, the CPU sets a flag, pc_src, to different values. Chip8_Top interprets them as follows: –if pc_src == PC_SRC_ALU, next_pc is set to the value that the

CPU is writing to it –if pc_src == PRC_SRC_SKIP, next_pc is set to pc+4. This effectively skips the next instruction –if pc_src == PC_SRC_NEXT, next_pc is set to pc+2. This is the default, which increments the PC by one instruction.

There is also a special case for setting next_pc. The CPU sets flags regarding the return address stack (RAS). This flag, stk_op, acts as follows: –if the CPU sets stk_op to STACK_HOLD, the PC is unaffected by the RAS –if the CPU sets stk_op to STACK_PUSH, pc+2 is pushed to the top of the stack –if the CPU sets stk_op to STACK_POP, next_pc is set to the value on top of the RAS in in stage 12, and the top of the RAS is popped off. The stack pointer decreases by 1.

The RAS has 16 entries of 16 bits each. The stack does not push every cycle that stk_op == STACK_PUSH, nor does it pop every time it equals STACK_POP. These operations only happen when stk_op changes from STACK_HOLD to either of these operations

3.9 Timers

There are two timers used in Chip8. These are the delay timer and sound timer. Both timers function the same way, that is, they are set to a particular value and count down at a rate of about 60 Hz per second. In order to implement this, a simple clock divider was implemented as a separate unit, which was then fed into the delay timer. The clock divider simply counts the relevant number of 50 MHz CPU cycles to equal one 60 Hz cycle, then is high for a single 50 MHz cycle and the counter is reset.

Chapter 4

Software Overview

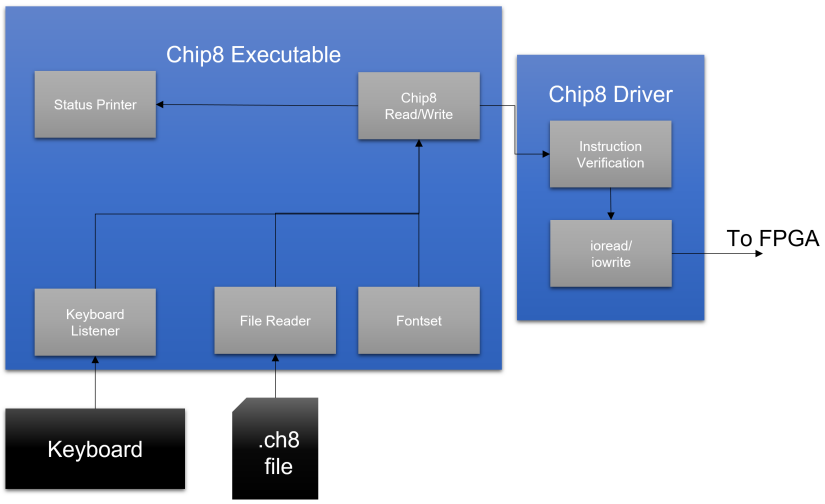


Figure 4.1: Overview of the software design

4.1 Chip8.c

The chip8.c file is an executable meant to load ROM files and listen for keyboard input. It is capable of resetting the device and contains the entire font set as a c array. It has a series of functions which operate on and manipulate the device. On each key press event it writes either that the key

is currently pressed and the value (mapped using the mapping mentioned earlier) or writes that no key is currently pressed. Upon each reset, the entire ROM file is rewritten to the device and the fontset is sent over as well. Something that is not supported is changing the runtime speed of the device, but this is something that could be trivially added.

4.2 Chip8driver.c

This filters and ensures that command sent to the device of the correct format. It is loaded using standard linux kernel module loading standards. It is loaded using sudo insmod chip8driver.ko after running make in the directory. For a list of appropriate opcodes that can be sent to the device, see the next section.

4.3 Data Transfer ISA

- **V0_ADDR** - To write data to a particular register, use iowrite with NNNNNNXX Where NNNNNN is ignored XX is the 8 bits to be written To read from a register use ioread with the address
- **V1_ADDR** - To write data to a particular register, use iowrite with NNNNNNXX Where NNNNNN is ignored XX is the 8 bits to be written To read from a register use ioread with the address
- **V2_ADDR** - To write data to a particular register, use iowrite with NNNNNNXX Where NNNNNN is ignored XX is the 8 bits to be written To read from a register use ioread with the address
- **V3_ADDR** - To write data to a particular register, use iowrite with NNNNNNXX Where NNNNNN is ignored XX is the 8 bits to be written To read from a register use ioread with the address
- **V4_ADDR** - To write data to a particular register, use iowrite with NNNNNNXX Where NNNNNN is ignored XX is the 8 bits to be written To read from a register use ioread with the address
- **V5_ADDR** - To write data to a particular register, use iowrite with NNNNNNXX Where NNNNNN is ignored XX is the 8 bits to be written To read from a register use ioread with the address

- **V6_ADDR** - To write data to a particular register, use iowrite with NNNNNNXX Where NNNNNN is ignored XX is the 8 bits to be written To read from a register use ioread with the address
- **V7_ADDR** - To write data to a particular register, use iowrite with NNNNNNXX Where NNNNNN is ignored XX is the 8 bits to be written To read from a register use ioread with the address
- **V8_ADDR** - To write data to a particular register, use iowrite with NNNNNNXX Where NNNNNN is ignored XX is the 8 bits to be written To read from a register use ioread with the address
- **V9_ADDR** - To write data to a particular register, use iowrite with NNNNNNXX Where NNNNNN is ignored XX is the 8 bits to be written To read from a register use ioread with the address
- **VA_ADDR** - To write data to a particular register, use iowrite with NNNNNNXX Where NNNNNN is ignored XX is the 8 bits to be written To read from a register use ioread with the address
- **VB_ADDR** - To write data to a particular register, use iowrite with NNNNNNXX Where NNNNNN is ignored XX is the 8 bits to be written To read from a register use ioread with the address
- **VC_ADDR** - To write data to a particular register, use iowrite with NNNNNNXX Where NNNNNN is ignored XX is the 8 bits to be written To read from a register use ioread with the address
- **VD_ADDR** - To write data to a particular register, use iowrite with NNNNNNXX Where NNNNNN is ignored XX is the 8 bits to be written To read from a register use ioread with the address
- **VE_ADDR** - To write data to a particular register, use iowrite with NNNNNNXX Where NNNNNN is ignored XX is the 8 bits to be written To read from a register use ioread with the address
- **VF_ADDR** - To write data to a particular register, use iowrite with NNNNNNXX Where NNNNNN is ignored XX is the 8 bits to be written To read from a register use ioread with the address

- **I_ADDR** - To write to the I index register NNNNDDDD DDDD is the 16 bits to write Use ioread to read from the I register
- **SOUND_TIMER_ADDR** - To write to the sound timer NNNNNNDD Where DD is the number to write to the sound timer Use ioread to read from the sound timer
- **DELAY_TIMER_ADDR** - To write to the delay timer NNNNNNDD Where DD is the number to write to the delay timer Use ioread to read from the delay timer
- **STACK_POINTER_ADDR** - To write to the stack pointer NNNNNNDD Where DD is the number to write to the stack pointer Only the last six bits are considered

  Use ioread to read from the stack pointer
- **STACK_ADDR** - To reset the stack, iowrite
- **PROGRAM_COUNTER_ADDR** - To write to the program counter 0000DDDD Where DDDD is the number to write to the program counter

  Use ioread to read from the program counter
- **KEY_PRESS_ADDR** - To write a keypress to the Chip8 control unit NNNNNNPD Where D is the number corresponding to a keypress 0-F Where P is whether a key is currently pressed or not (0x1, 0x0)
- **STATE_ADDR** - To change the state of the Chip8 000000DD Where DD is an 8-bit number corresponding to varying states
  - 0x00 - Running
  - 0x02 - Paused

  The state is initially set to loading font set Use ioread to read the state of the Chip8
- **MEMORY_ADDR** - To write to a location in memory 0000_0000_0001_AAAA_AAAA_AAA Where DD is the 8-bit data that is to be written Where AAA is the 12-bit address to write the data Where W is a 1-bit value corresponding to a read or a write To read data from memory, use iowrite with

0000_0000_0000_AAAA_AAAA_AAAA_NNNN_NNNN Where AAA is the 12-bit address to read the data from

- **FRAMEBUFFER_ADDR** - In order to write data to the instruction 0000_0000_0000_0000_IIII_IIII_IIII_IIII Where I corresponds to the 16 bits in the instruction The state must currently be in Chip8_RUN_INSTRUCTION In order to read data from the framebuffer 0000_0000_0000_0000_0000_NXXX_XXXY_YYYY Where XX is the x position (6 bits) Where YY is the y position (5 bits) Where NN is ignored

Chapter 5

Project Plan

5.1 Lessons Learned

- **Ashley** - The lecture part of this course is very good for learning the theoretical aspects of embedded systems, but the most informative part is the homeworks and final project, which give you a feeling for how hardware design really works, especially the pitfalls. It is important to always start early and work often on the homeworks and project (especially the project), and to communicate frequently with the group members so that everyone is aware of exactly what each person is working on and when they think it will be done. Furthermore, I learned that it is imperative to think critically about everything you are doing, even the tools you are using. We had several issues with Quartus and Qsys, and even the lab computers. We also had a significant bug caused by assumptions we made about memory access times. We learned to make sure that we really understood the tools we used, and to test EVERYTHING. And last but not least, as David said, when the long nights take their toll (as they will), the difficulty is abated by the camaraderie of a good group.

- **Levi** - So far, we have learned and been taught how to program and how to fix broken code. But programming is sometimes only half of the job of a programmer. The other half is using tools that are supposedly used to make life easier. We encountered many problems using ModelSim, Qsys, and Quartus, and learned to a good degree how to make them all do what we wanted them to. What I've learned is that it is

just as important to learn and understand the tools being used as it is to program. I've also learned that unless a component's functionality is explicitly stated, it should probably be tested–at least basically. Our biggest and longest-lasting problem came from misunderstanding how the timing of the MegaFunction Wizard-generated memory files worked. I'd also like to repeat what David said–after too many late and frustrating nights in the lab, the only thing that kept us (at least partially) sane was each other. Stupid jokes really help.

- **Gabrielle** - Hardware is particularly tricky to work with because of how little of the internal functioning you can see. Although this is something you know at an intellectual level, and something mentioned by anyone with experience in hardware, it's still different from actually encountering this in practice. One of the more difficult challenges was figuring out why what seemed like reasonable code (in software, perhaps) does not translate well into hardware. Indeed, there was a point during the semester where I had to find a basic logic design textbook and look over some more complicated things than what we studied in Fundamentals. Overall it was an interesting class and project, particularly the fact that this was designing an entire interpreter from scratch and reminded me of the process one would go through to design their own computer, their own processor, etc. 10/10 would take again.

- **David** - This course requires a good comprehensive understanding and appreciation for the design and creation of hardware based programs. I have certainly learned some techniques for going into making a project such as this, but certainly it would be advisable to have as much hardware design knowledge going into this project as possible. I also recommend choosing good teammates which help make the project so much easier. People you can joke around with during the most frustrating parts of the project is an important quality in a good team.

5.2 Timeline

During the course of the semester we worked diligently to get as much done as we could on time. Unfortunately some of our predicted goals were too ambitious and it required a big crunch at the end to get a fully working emulator.

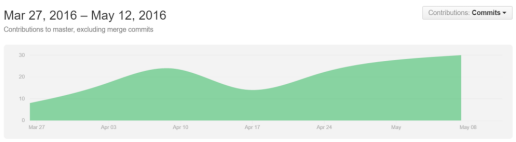


Figure 5.1: Git history over the semester

Chapter 6

Debugging

As stated earlier on in the semester, about a quarter of the time of the entire project was dedicated to debugging. Because this was our first time using these tools and designing in this way, it may have been even more than a quarter for us. Our project was particularly difficult to test since it was all in hardware–it was a black box situation. We worked to incorporate a versatile set of functions and instructions to read from and write to the board, and this helped us greatly–especially in the initial stages. With this, we were able to find out errors like when the PC was not being incremented properly, or when the return address stack was being misused.

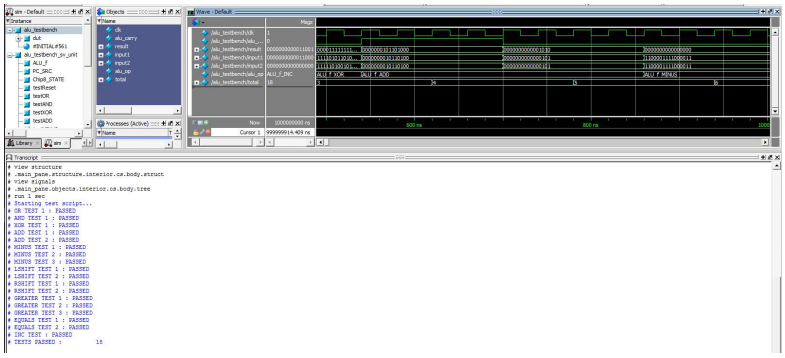


Figure 6.1: Our testbench running during the testing process

It was also difficult to test individual errors, since most of them were in SystemVerilog. If we wanted to test a single section, the code would be changed and a recompile would need to happen. This could be time

consuming and problematic.

We got to a stage in the debugging process when we couldn't think of what could possibly be wrong, and our limited output wasn't telling us anything other than something was being stored incorrectly. We stepped through from the ground up, making tracing the executable's assembly code, re-reading Chip8 documentation, and aggressively testing all of our modules and checking our base assumptions. It was only when we got to the MegaFunction Wizard-generated memory module and questioned our base assumptions on the timing of reading that we were able to diagnose and fix the problem.

Below are two pictures. Both of them are the output resulting from running the same executable.

Figure 6.2: An early stage of IBM during the debugging process

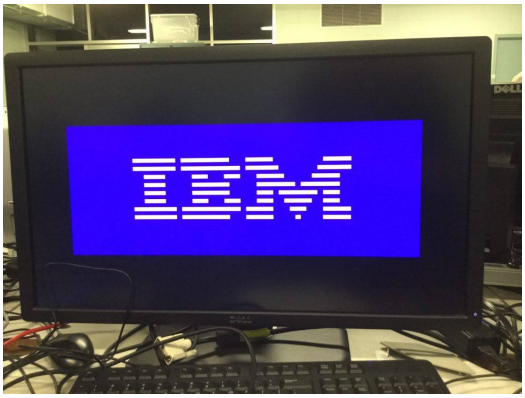


Figure 6.3: IBM after several rounds of debugging

Chapter 7

Code Listing

7.1 SystemVerilog Code

7.1.1 bcd.sv

```
module bcd(
    input logic [7:0] num,
    output logic [3:0] hundreds,
    output logic [3:0] tens,
    output logic [3:0] ones);

    logic [19:0] shift;

    always_comb begin
        shift[19:8] = 11'd0;
        shift[7:0] = num;
        repeat (8) begin
            if(shift[11:8] >= 3'd5)
                shift[11:8] = shift[11:8] + 3'd3;
            if(shift[15:12] >= 3'd5)
                shift[15:12] = shift[15:12] + 3'd3;
            if(shift[19:16] >= 3'd5)
                shift[19:16] = shift[19:16] + 3'd3;
            // Shift entire register left once
            shift = shift << 1;
```

```
        end
        hundreds = shift[19:16];
        tens     = shift[15:12];
        ones     = shift[11:8];
    end

endmodule
```

7.1.2 Chip8_CPU.sv

```
/*********************************************************************************
 * CHIP8_CPU.sv
 *
 * Contains the code for interpreting and running instructions as per the Chip8
 * ISA defined at: http://devernay.free.fr/hacks/chip8/C8TECH10.HTM
 *
 * The main idea behind the instructions is that initially the CPU will request
 * data from registers and memory at stage 0, and then operate on the data
 * returned by the request at stage i, where i > 0. This way the CPU can handle
 * instructions that operate over multiple cycles and return values that are
 * a function of the number of cycles that have occured.
 *
 * AUTHORS: David Watkins, Levi Oliver
 * Dependencies:
 *     - Chip8_CPU/Chip8_ALU.sv
 *    - Chip8_CPU/Chip8_rand_num_generator.sv
 *    - Chip8_CPU/bcd.sv
 *     - enums.svh
 *     - utils.svh

 *********************************************************************************
```

```
21
22 `include "../enums.svh"
23 `include "../utils.svh"
24
25 module Chip8_CPU(
26     input logic cpu_clk,
27     input logic[15:0] instruction,
28     input logic[7:0] reg_readdata1, reg_readdata2,
29                      mem_readdata1, mem_readdata2,
30     input logic[15:0] reg_I_readdata,
31     input logic[7:0] delay_timer_readdata,
32
33     input logic key_pressed,
34     input logic[3:0] key_press,
35
36     input logic[11:0] PC_readdata,
37
38     input logic[31:0] stage,
39
40     input logic fb_readdata,
41
42     input Chip8_STATE top_level_state,
43
44     output logic delay_timer_WE, sound_timer_WE,
45     output logic[7:0] delay_timer_writedata,
       ↪  sound_timer_writedata,
46
47     output PC_SRC pc_src,
48     output logic[11:0] PC_writedata,
49
50     output logic reg_WE1, reg_WE2,
51     output logic[3:0] reg_addr1, reg_addr2,
52     output logic[7:0] reg_writedata1, reg_writedata2,
53
54     output logic mem_WE1, mem_WE2,
55     output logic[11:0] mem_addr1, mem_addr2,
56     output logic[ 7:0] mem_writedata1, mem_writedata2,
57     output logic       mem_request,
```

```
58
59      output logic reg_I_WE,
60      output logic[15:0] reg_I_writedata,
61
62      output logic stk_reset,
63      output STACK_OP stk_op,
64      output logic[15:0] stk_writedata,
65
66      output logic [4:0]    fb_addr_y,//max val = 31
67      output logic [5:0]    fb_addr_x,//max val = 63
68      output logic        fb_writedata, //data to write to
        ↪ addresse.
69      output logic        fb_WE, //enable writing to address
70      output logic        fbreset,
71
72      output logic        bit_overwritten, //VF overwritten
73      output logic         isDrawing,          //Draw
        ↪ instruction where VF could be overwritten
74
75      output logic halt_for_keypress
76  );
77
78      logic[15:0] alu_in1, alu_in2, alu_out;
79      ALU_f alu_cmd;
80      logic alu_carry;
81
82      wire[15:0] rand_num;
83      logic[7:0] to_bcd;
84      wire[3:0] bcd_hundreds, bcd_tens, bcd_ones;
85
86      wire[31:0] stage_shifted_by4_minus1 = (stage >> 32'h4) -
        ↪ 32'h1;
87      logic[31:0] num_rows_written; //used for sprite writing
88
89      logic [31:0] stageminus16;
90
91      Chip8_rand_num_generator rand_num_generator(cpu_clk,
        ↪ rand_num);
```

```
92      bcd binary_to_dec(to_bcd, bcd_hundreds, bcd_tens,
        ↪  bcd_ones);
93      Chip8_ALU alu(alu_in1, alu_in2, alu_cmd, alu_out,
        ↪  alu_carry);
94
95      always_comb begin
96          /*DEFAULT WIRE VALUES BEGIN*/
97          delay_timer_WE             = 1'b0;
98          sound_timer_WE             = 1'b0;
99          delay_timer_writedata    = 8'b0;
100         sound_timer_writedata    = 8'b0;
101         pc_src                     = PC_SRC_NEXT;
102         PC_writedata             = 12'b0;
103         reg_WE1                  = 1'b0;
104         reg_WE2                  = 1'b0;
105         reg_addr1                  = 4'b0;
106         reg_addr2                  = 4'b0;
107         reg_writedata1             = 8'b0;
108         reg_writedata2             = 8'b0;
109         mem_WE1                  = 1'b0;
110         mem_WE2                  = 1'b0;
111         mem_addr1                  = 12'h0;
112         mem_addr2                  = 12'h0;
113         mem_request              = 1'b0;
114         mem_writedata1             = 8'h0;
115         mem_writedata2             = 8'h0;
116         reg_I_WE                 = 1'b0;
117         reg_I_writedata            = 16'h0;
118         fb_addr_y                = 5'h0;
119         fb_addr_x                = 6'h0;
120         fb_writedata             = 1'b0;
121         fb_WE                    = 1'b0;
122         fbreset                  = 1'b0;
123         num_rows_written         = 4'h0;
124         bit_overwritten          = 1'b0;
125         halt_for_keypress          = 1'b0;
126         alu_in1                  = 16'h0;
127         alu_in2                  = 16'h0;
```

```
128         alu_cmd                 = ALU_f_NOP;
129         to_bcd                   = 8'h0;
130         stk_op                  = STACK_HOLD;
131         stk_reset               = 1'b0;
132         stk_writedata           = 16'b0;
133         isDrawing               = 1'b0;
134         stageminus16             = stage - 32'd16;
135         /*END DEFAULT VALUES*/
136
137
138         /*BEGIN INSTRUCTION DECODE*/
139         if(top_level_state == Chip8_RUNNING && stage != 32'h0)
            ↪ begin
140         casex (instruction)
141             // 16'h???: begin
142             //This instruction is only used on the old
                ↪ computers on which Chip-8
143             //was originally implemented. It is ignored by
                ↪ modern interpreters.
144             // end
145
146             16'h00E0: begin //00E0 - CLS
147                 //Clear the screen
148                 if(stage == 32'h2) begin
149                     fbreset = 1'b1;
150                 end else if (stage > 32'h2 & stage < 32'd8189)
                    ↪ begin
151                     fb_addr_x = stage[7:2];
152                     fb_addr_y = stage[12:8];
153                     fb_WE = 1'b1;
154                     fb_writedata = 1'b0;
155                     //CPU DONE
156                 end
157             end
158
159             //memory module fix May 10
160             16'h00EE: begin //00EE - RET
161                 //Return from a subroutine.
```

```
162                 //The interpreter sets the program counter to
                     ↪  the address at the
163                 //top of the stack, then subtracts 1 from the
                     ↪  stack pointer.
164                 if(stage >= 32'h3 & stage <=
                     ↪  NEXT_PC_WRITE_STAGE) begin //two stages
                     ↪  b/c stack takes two cycles
165                     stk_op = STACK_POP;
166                     pc_src = PC_SRC_STACK;
167                 end else begin
168                     //CPU DONE
169                 end
170             end
171
172             //memory module fix May 10
173             16'h1xxx: begin //1nnn - JP addr
174                 //Jump to location nnn.
175                 //The interpreter sets the program counter to
                     ↪  nnn.
176                 if(stage >= 32'h3 & stage <=
                     ↪  NEXT_PC_WRITE_STAGE) begin
177                     pc_src = PC_SRC_ALU;
178                     PC_writedata = instruction[11:0];
179                 end else begin
180                     //CPU DONE
181                 end
182             end
183
184             //memory module fix May 10
185             16'h2xxx: begin //2nnn - CALL addr
186                 //Call subroutine at nnn.
187                 //The interpreter increments the stack
                     ↪  pointer, then puts the
188                 //current PC on the top of the stack. The PC
                     ↪  is then set to nnn.
189                 if(stage >= 32'h3 & stage <=
                     ↪  NEXT_PC_WRITE_STAGE) begin
190                     stk_op = STACK_PUSH;
```

```
191                     stk_writedata = PC_readdata + 12'h2;
192                     pc_src = PC_SRC_ALU;
193                     PC_writedata = instruction[11:0];
194                 end else begin
195                     //CPU DONE
196                 end
197             end
198
199             //memory module fix May 10
200             16'h3xxx: begin //3xkk - SE Vx, byte
201                 //Skip next instruction if Vx = kk.
202                 //The interpreter compares register Vx to kk,
                    ↪ and if they are
203                 //equal, increments the program counter by
                    ↪ 2.
204
205                 if(stage >= 32'h3 & stage <=
                    ↪ NEXT_PC_WRITE_STAGE) begin
206                     reg_addr1 = instruction[11:8];
207                     if(reg_readdata1 == instruction[7:0])
                        ↪ pc_src = PC_SRC_SKIP;
208                     else pc_src = PC_SRC_NEXT;
209                 end
210
211             end
212
213             //memory module fix May 10
214             16'h4xxx: begin //4xkk - SNE Vx, byte
215                 //Skip next instruction if Vx != kk.
216                 //The interpreter compares register Vx to kk,
                    ↪ and if they are
217                 //not equal, increments the program counter
                    ↪ by 2.
218                 if(stage >= 32'h3 & stage <=
                    ↪ NEXT_PC_WRITE_STAGE) begin
219                     reg_addr1 = instruction[11:8];
220                     if(reg_readdata1 != instruction[7:0])
                        ↪ pc_src = PC_SRC_SKIP;
```

```
221                 else pc_src = PC_SRC_NEXT;
222             end
223         end
224
225         //memory module fix May 10
226         16'h5xx0: begin //5xy0 - SE Vx, Vy
227             //Skip next instruction if Vx = Vy.
228             //The interpreter compares register Vx to
                ↪ register Vy, and if
229             //they are equal, increments the program
                ↪ counter by 2.
230             if(stage >= 32'h3 & stage <=
                ↪ NEXT_PC_WRITE_STAGE) begin
231                 reg_addr1 = instruction[11:8];
232                 reg_addr2 = instruction[ 7:4];
233                 if(reg_readdata1 == reg_readdata2) pc_src
                    ↪ = PC_SRC_SKIP;
234                 else pc_src = PC_SRC_NEXT;
235             end
236         end
237
238         //memory module fix May 10
239         16'h6xxx: begin //6xkk - LD Vx, byte
240             //Set Vx = kk.
241             //The interpreter puts the value kk into
                ↪ register Vx.
242
243             if(stage == 32'h2 || stage == 32'h3) begin
244                 reg_addr1 = instruction[11:8];
245                 reg_writedata1 = instruction[7:0];
246                 reg_WE1 = 1'b1;
247             end else begin
248                 //CPU DONE
249             end
250         end
251
252         //memory module fix May 10
253         16'h7xxx: begin //7xkk - ADD Vx, byte
```

```
254                 //Set Vx = Vx + kk.
255                 //Adds the value kk to the value of register
                     ↪  Vx, then stores the
256                 //result in Vx.
257                 if(stage >= 32'h2 & stage <= 32'h6) begin
258                     reg_addr1 = instruction[11:8];
259                 end else if(stage == 32'h7) begin
260                     reg_addr1 = instruction[11:8];
261                     reg_writedata1 = alu_out[7:0];
262                     reg_WE1 = 1'b1;
263 
264                     alu_in1 = reg_readdata1;
265                     alu_in2 = instruction[7:0];
266                     alu_cmd = ALU_f_ADD;
267                 end else begin
268                     //CPU DONE
269                 end
270             end
271 
272             //memory module fix May 10
273             //Arithmetic operators
274             16'h8xxx: begin //8xyk
275                 if(stage >= 32'h2 && stage <= 32'h7) begin
276                     reg_addr1 = instruction[11:8];
277                     reg_addr2 = instruction[ 7:4];
278                 end else if(stage >= 32'h2 && stage <= 32'h8)
                     ↪  begin
279                     case (instruction[3:0])
280                         4'h0: begin //8xy0 - LD Vx, Vy
281                             //Set Vx = Vy.
282                             //Stores the value of register Vy
                              ↪  in register Vx.
283                             reg_addr1 = instruction[11:8];
284                             reg_addr2 = instruction[ 7:4];
285                             reg_writedata1 = reg_readdata2;
286                             reg_WE1 = 1'b1;
287                         end
288 
```

```
289                     4'h1: begin //8xy1 - OR Vx, Vy
290                         //Set Vx = Vx OR Vy.
291                         //Performs a bitwise OR on the
                             ↪  values of Vx and Vy,
292                         //then stores the result in Vx. A
                             ↪  bitwise OR
293                         //compares the corrseponding bits
                             ↪  from two values,
294                         //and if either bit is 1, then
                             ↪  the same bit in the
295                         //result is also 1. Otherwise, it
                             ↪  is 0.
296
297                         alu_cmd = ALU_f_OR;
298                         alu_in1 = reg_readdata1;
299                         alu_in2 = reg_readdata2;
300
301                         reg_addr1 = instruction[11:8];
302                         reg_addr2 = instruction[ 7:4];
303                         reg_WE1 = 1'b1;
304                         reg_writedata1 = alu_out[7:0];
305                     end
306
307                     4'h2: begin //8xy2 - AND Vx, Vy
308                         //Set Vx = Vx AND Vy.
309                         //Performs a bitwise AND on the
                             ↪  values of Vx and Vy,
310                         //then stores the result in Vx. A
                             ↪  bitwise AND
311                         //compares the corrseponding bits
                             ↪  from two values,
312                         //and if both bits are 1, then
                             ↪  the same bit in the
313                         //result is also 1. Otherwise, it
                             ↪  is 0.
314
315                         alu_cmd = ALU_f_AND;
316                         alu_in1 = reg_readdata1;
```

```
317                     alu_in2 = reg_readdata2;
318
319                     reg_addr1 = instruction[11:8];
320                     reg_addr2 = instruction[ 7:4];
321                     reg_WE1 = 1'b1;
322                     reg_writedata1 = alu_out[7:0];
323                 end
324
325                 4'h3: begin //8xy3 - XOR Vx, Vy
326                     //Set Vx = Vx XOR Vy.
327                     //Performs a bitwise exclusive OR
                        ↪ on the values of
328                     //Vx and Vy, then stores the
                        ↪ result in Vx. An
329                     //exclusive OR compares the
                        ↪ corrseponding bits from
330                     //two values, and if the bits are
                        ↪ not both the same,
331                     //then the corresponding bit in
                        ↪ the result is set to
332                     //1. Otherwise, it is 0.
333
334                     alu_cmd = ALU_f_XOR;
335                     alu_in1 = reg_readdata1;
336                     alu_in2 = reg_readdata2;
337
338                     reg_addr1 = instruction[11:8];
339                     reg_addr2 = instruction[ 7:4];
340                     reg_WE1 = 1'b1;
341                     reg_writedata1 = alu_out[7:0];
342                 end
343
344                 4'h4: begin //8xy4 - ADD Vx, Vy
345                     //Set Vx = Vx + Vy, set VF =
                        ↪ carry.
346                     //The values of Vx and Vy are
                        ↪ added together. If the
```

```
347                     //result is greater than 8 bits
                        ↪  (i.e., > 255,) VF is
348                     //set to 1, otherwise 0. Only the
                        ↪  lowest 8 bits of
349                     //the result are kept, and stored
                        ↪  in Vx.
350
351                     alu_cmd = ALU_f_ADD;
352                     alu_in1 = reg_readdata1;
353                     alu_in2 = reg_readdata2;
354
355                     reg_addr1 = instruction[11:8];
356                     reg_WE1 = 1'b1;
357                     reg_writedata1 = alu_out[7:0];
358
359                     reg_addr2 = 4'hF;
360                     reg_WE2 = 1'b1;
361                     reg_writedata2 = alu_carry;
362                 end
363
364                 4'h5: begin //8xy5 - SUB Vx, Vy
365                     //Set Vx = Vx - Vy, set VF = NOT
                        ↪  borrow.
366                     //If Vx > Vy, then VF is set to
                        ↪  1, otherwise 0. Then
367                     //Vy is subtracted from Vx, and
                        ↪  the results stored
368                     //in Vx.
369
370                     alu_cmd = ALU_f_MINUS;
371                     alu_in1 = reg_readdata1;
372                     alu_in2 = reg_readdata2;
373
374                     reg_addr1 = instruction[11:8];
375                     reg_WE1 = 1'b1;
376                     reg_writedata1 = alu_out[7:0];
377
378                     reg_addr2 = 4'hF;
```

```
379                     reg_WE2 = 1'b1;
380                     reg_writedata2 = alu_carry;
381                 end
382
383                 4'h6: begin //8xy6 - SHR Vx {, Vy}
384                     //Set Vx = Vx SHR 1.
385                     //If the least-significant bit of
                        ↪  Vx is 1, then VF
386                     //is set to 1, otherwise 0. Then
                        ↪  Vx is divided by 2.
387
388                     reg_addr1 = instruction[11:8];
389                     reg_WE2 = 1'b1;
390                     reg_writedata2 = {7'h0,
                        ↪  reg_readdata1[0]};
391
392                     alu_cmd = ALU_f_RSHIFT;
393                     alu_in1 = reg_readdata1;
394                     alu_in2 = 1;
395
396                     reg_addr2 = 4'hF;
397                     reg_WE1 = 1'b1;
398                     reg_writedata1 = alu_out[7:0];
399                 end
400
401                 4'h7: begin //8xy7 - SUBN Vx, Vy
402                     //Set Vx = Vy - Vx, set VF = NOT
                        ↪  borrow.
403                     //If Vy > Vx, then VF is set to
                        ↪  1, otherwise 0. Then
404                     //Vx is subtracted from Vy, and
                        ↪  the results stored
405                     //in Vx.
406
407                     alu_cmd = ALU_f_MINUS;
408                     alu_in1 = reg_readdata2;
409                     alu_in2 = reg_readdata1;
410
```

```
411                         reg_addr1 = instruction[11:8];
412                         reg_WE1 = 1'b1;
413                         reg_writedata1 = alu_out[7:0];
414
415                         reg_addr2 = 4'hF;
416                         reg_WE2 = 1'b1;
417                         reg_writedata2 = alu_carry;
418                     end
419
420                     4'hE: begin //8xyE - SHL Vx {, Vy}
421                         //Set Vx = Vx SHL 1.
422                         //If the most-significant bit of
                            ↪  Vx is 1, then VF is
423                         //set to 1, otherwise to 0. Then
                            ↪  Vx is multiplied
424                         //by 2.
425
426                         reg_addr2 = 4'hF;
427                         reg_WE2 = 1'b1;
428                         reg_writedata2 = {7'h0,
                            ↪  reg_readdata1[7]};
429
430                         alu_cmd = ALU_f_LSHIFT;
431                         alu_in1 = reg_readdata1;
432                         alu_in2 = 1;
433
434                         reg_addr1 = instruction[11:8];
435                         reg_WE1 = 1'b1;
436                         reg_writedata1 = alu_out[7:0];
437                     end
438
439                     default : /* default */;
440                 endcase
441             end else begin
442                 //CPU DONE
443             end
444         end
445
```

```
446             //memory module fix May 10
447             16'h9xx0: begin //9xy0 - SNE Vx, Vy
448                 //Skip next instruction if Vx != Vy.
449                 //The values of Vx and Vy are compared, and
                    ↪ if they are not
450                 //equal, the program counter is increased by
                    ↪ 2.
451
452                 if(stage >= 32'h3 & stage <=
                    ↪ NEXT_PC_WRITE_STAGE) begin
453                     reg_addr1 = instruction[11:8];
454                     reg_addr2 = instruction[ 7:4];
455                     if(reg_readdata1 != reg_readdata2) pc_src
                        ↪ = PC_SRC_SKIP;
456                     else pc_src = PC_SRC_NEXT;
457                 end
458             end
459
460             //memory module fix May 10
461             16'hAxxx: begin //Annn - LD I, addr
462                 //Set I = nnn.
463                 //The value of register I is set to nnn.
464                 if(stage == 32'h2 || stage == 32'h3) begin
465                     reg_I_WE = 1'b1;
466                     reg_I_writedata = {4'h0,
                        ↪ instruction[11:0]};
467                 end else begin
468                     //CPU DONE
469                 end
470
471             end
472
473             //memory module fix May 10
474             16'hBxxx: begin //Bnnn - JP V0, addr
475                 //Jump to location nnn + V0.
476                 //The program counter is set to nnn plus the
                    ↪ value of V0.
```

```
477                 if(stage >= 32'h2 & stage <=
                     ↪  NEXT_PC_WRITE_STAGE) begin
478                     reg_addr1 = 4'h0;
479                     PC_writedata = instruction[11:0] + {4'h0,
                         ↪  reg_readdata1};
480                     pc_src = PC_SRC_ALU;
481                 end
482             end
483
484             //memory module fix May 10
485             16'hCxxx: begin //Cxkk - RND Vx, byte
486                 //Set Vx = random byte AND kk.
487                 //The interpreter generates a random number
                     ↪  from 0 to 255, which
488                 //is then ANDed with the value kk. The
                     ↪  results are stored in Vx.
489                 //See instruction 8xy2 for more information
                     ↪  on AND.
490
491                 if(stage >= 32'h3 & stage <=
                     ↪  NEXT_PC_WRITE_STAGE) begin
492                     alu_cmd = ALU_f_AND;
493                     alu_in1 = rand_num[7:0];
494                     alu_in2 = instruction[7:0];
495
496                     reg_addr1 = instruction[11:8];
497                     reg_WE1 = 1'b1;
498                     reg_writedata1 = alu_out[7:0];
499                 end else begin
500                     //CPU DONE
501                 end
502
503             end
504
505             //memory module fix May 10
506             16'hDxxx: begin //Dxyn - DRW Vx, Vy, nibble
507                 //Display n-byte sprite starting at memory
                     ↪  location I at
```

```
508                 //(Vx, Vy), set VF = collision.
509
510                 //The interpreter reads n bytes from memory,
                    ↪  starting at the
511                 //address stored in I. These bytes are then
                    ↪  displayed as sprites
512                 //on screen at coordinates (Vx, Vy). Sprites
                    ↪  are XORed onto the
513                 //existing screen. If this causes any pixels
                    ↪  to be erased, VF is
514                 //set to 1, otherwise it is set to 0. If the
                    ↪  sprite is
515                 //positioned so part of it is outside the
                    ↪  coordinates of the
516                 //display, it wraps around to the opposite
                    ↪  side of the screen.
517                 //See instruction 8xy3 for more information
                    ↪  on XOR, and section
518                 //2.4, Display, for more information on the
                    ↪  Chip-8 screen and
519                 //sprites.
520
521                 if(stage > 4'b1111) begin
522                     reg_addr1 = instruction[11:8];
523                     reg_addr2 = instruction[ 7:4];
524                     num_rows_written =
                        ↪  {7'b0,stageminus16[31:7]};
525                     mem_addr1 = num_rows_written +
                        ↪  reg_I_readdata;
526                     mem_request = 1'b1;
527                     fb_addr_x = reg_readdata1 + ({5'b0,
                        ↪  stageminus16[6:4]});
528                     fb_addr_y = reg_readdata2 + ({4'b0,
                        ↪  num_rows_written[3:0]});
529
530                     fb_writedata = mem_readdata1[3'h7 -
                        ↪  stageminus16[6:4]] ^ fb_readdata;
531
```

```
532                     fb_WE = (num_rows_written < {28'h0,
                        ↪  instruction[3:0]}) & (&(stage[3:0]));
533                     bit_overwritten = (mem_readdata1[3'h7 -
                        ↪  stageminus16[6:4]]) & (fb_readdata) &
                        ↪  fb_WE;
534                     isDrawing = 1'b1;
535                 end
536                 /*
537                  if(stage >= 32'h2) begin
538                      reg_addr1 = instruction[11:8];
539                      reg_addr2 = instruction[ 7:4];
540
541                      if(stage <= 32'h15) num_rows_written =
    ↪  4'b0;
542                      else num_rows_written =
    ↪  stage_shifted_by4_minus1[3:0];//((stage >> 32'h4) -
    ↪  32'h1);
543
544                      mem_addr1 = reg_I_readdata[11:0] +
    ↪  {8'b0,num_rows_written};
545                      mem_request = (stage >= 32'd16) &
    ↪  (stage_shifted_by4_minus1 < instruction[3:0]) &
    ↪  !(stage[0]);
546                      fb_WE = (stage >= 32'd16) &
    ↪  (stage_shifted_by4_minus1 < instruction[3:0]) &
    ↪  (stage[0]);
547                      fb_addr_x = reg_readdata1 + ({5'b0,
    ↪  stage[3:1]});
548                      fb_addr_y = reg_readdata2 + ({4'b0,
    ↪  num_rows_written});
549                      fb_writedata = mem_readdata1[stage[3:1]]
    ↪  ^ fb_readdata;
550                      bit_overwritten =
    ↪  (mem_readdata1[stage[3:1]]) & (fb_readdata) & fb_WE;
551                          //bit_overwritten goes high whenever
    ↪  a pixel is set from 1 to 0
552                      isDrawing = 1'b1;
553                  end
```

```
554                 */
555             end
556
557             //memory module fix May 10
558             16'hEx9E: begin //Ex9E - SKP Vx
559                 //Skip next instruction if key with the value
                    ↪ of Vx is pressed.
560                 //Checks the keyboard, and if the key
                    ↪ corresponding to the value
561                 //of Vx is currently in the down position, PC
                    ↪ is increased by 2.
562
563                 if(stage >= 32'h2 & stage <=
                    ↪ NEXT_PC_WRITE_STAGE) begin
564                     reg_addr1 = instruction[11:8];
565                     if(key_pressed && key_press ==
                        ↪ reg_readdata1) begin
566                         pc_src = PC_SRC_SKIP;
567                     end
568                 end else begin
569                     //CPU DONE
570                 end
571             end
572
573             //memory module fix May 10
574             16'hExA1: begin //ExA1 - SKNP Vx
575                 //Skip next instruction if key with the value
                    ↪ of Vx is not
576                 //pressed.
577                 //Checks the keyboard, and if the key
                    ↪ corresponding to the value
578                 //of Vx is currently in the up position, PC
                    ↪ is increased by 2.
579
580                 if(stage >= 32'h2 & stage <=
                    ↪ NEXT_PC_WRITE_STAGE) begin
581                     reg_addr1 = instruction[11:8];
```

```
582                     if(~key_pressed || key_press !=
                        ↪  reg_readdata1) begin
583                         pc_src = PC_SRC_SKIP;
584                     end
585                 end else begin
586                     //CPU DONE
587                 end
588             end
589
590             //memory module fix May 10
591             //F Instructions
592             16'hFx07: begin //Fx07 - LD Vx, DT
593                 //Set Vx = delay timer value.
594                 //The value of DT is placed into Vx.
595
596                 if(stage >= 32'h2 & stage <= 32'h6) begin
597                     reg_addr1 = instruction[11:8];
598                     reg_writedata1 = delay_timer_readdata;
599                     reg_WE1 = 1'b1;
600                 end else begin
601                     //CPU DONE
602                 end
603             end
604
605             16'hFx0A: begin //Fx0A - LD Vx, K
606                 //Wait for a key press, store the value of
                    ↪  the key in Vx.
607                 //All execution stops until a key is pressed,
                    ↪  then the value of
608                 //that key is stored in Vx.
609
610                 if((stage >= 32'h2) && (NEXT_PC_WRITE_STAGE >=
                    ↪  stage) & !halt_for_keypress) begin
611                     halt_for_keypress = 1'b1;
612                 end else if(key_pressed) begin
613                     halt_for_keypress = 1'b0;
614                     reg_addr1 = instruction[11:8];
615                     reg_writedata1 = key_press;
```

```
616                     reg_WE1 = 1'b1;
617                 end else begin
618                     //CPU DONE
619                 end
620             end
621
622             //memory module fix May 10
623             16'hFx15: begin //Fx15 - LD DT, Vx
624                 //Set delay timer = Vx.
625                 //DT is set equal to the value of Vx.
626
627                 if(stage >= 32'h2 & stage <= 32'h6) begin
628                     reg_addr1 = instruction[11:8];
629                 end else if(stage <= NEXT_PC_WRITE_STAGE)
                     ↪ begin
630                     delay_timer_writedata = reg_readdata1;
631                     delay_timer_WE = 1'b1;
632                 end else begin
633                     //CPU DONE
634                 end
635             end
636
637             //memory module fix May 10
638             16'hFx18: begin //Fx18 - LD ST, Vx
639                 //Set sound timer = Vx.
640                 //ST is set equal to the value of Vx.
641
642                 if(stage >= 32'h2 & stage <= 32'h6) begin
643                     reg_addr1 = instruction[11:8];
644                 end else if(stage == 32'h7) begin
645                     sound_timer_writedata = reg_readdata1;
646                     sound_timer_WE = 1'b1;
647                 end else begin
648                     //CPU DONE
649                 end
650             end
651
652             //memory module fix May 10
```

```
653                 16'hFx1E: begin //Fx1E - ADD I, Vx
654                     //Set I = I + Vx.
655                     //The values of I and Vx are added, and the
                         ↪  results are stored
656                     //in I.
657 
658                     if(stage >= 32'h2 & stage <= 32'h5) begin
659                         reg_addr1 = instruction[11:8];
660                     end else if(stage == 32'h6) begin
661                         alu_cmd = ALU_f_ADD;
662                         alu_in1 = reg_I_readdata;
663                         alu_in2 = reg_readdata1;
664 
665                         reg_I_writedata = alu_out;
666                         reg_I_WE = 1'b1;
667                     end else begin
668                         //CPU DONE
669                     end
670                 end
671 
672                 //memory module fix May 10
673                 16'hFx29: begin //Fx29 - LD F, Vx
674                     //Set I = location of sprite for digit Vx.
675                     //The value of I is set to the location for
                         ↪  the hexadecimal
676                     //sprite corresponding to the value of Vx.
                         ↪  See section 2.4,
677                     //Display, for more information on the Chip-8
                         ↪  hexadecimal font.
678 
679                     //The chip8 fontset has each character
                         ↪  starting from 0 to 80,
680                     //where each character takes 5 bytes each.
681 
682                     if(stage >= 32'h2 & stage <= 32'h5) begin
683                         reg_addr1 = instruction[11:8];
684                     end else if(stage == 32'h6) begin
```

```
685                     reg_I_writedata = {10'h0,
                        ↪  reg_readdata1[3:0], 2'h0} +
                        ↪  reg_readdata1[3:0];
686                     reg_I_WE = 1'b1;
687                 end else begin
688                     //CPU DONE
689                 end
690             end
691
692             //memory module fix May 10
693             16'hFx33: begin //Fx33 - LD B, Vx
694                 //Store BCD representation of Vx in memory
                    ↪  locations I, I+1, and
695                 //I+2.
696                 //The interpreter takes the decimal value of
                    ↪  Vx, and places the
697                 //hundreds digit in memory at location in I,
                    ↪  the tens digit at
698                 //location I+1, and the ones digit at
                    ↪  location I+2.
699
700                 if(stage >= 32'h2 & stage <= 32'h5) begin
701                     reg_addr1 = instruction[11:8];
702                 end else if(stage >= 32'h6 & stage <= 32'h9)
                    ↪  begin
703                     to_bcd = reg_readdata1;
704                     reg_addr1 = instruction[11:8];
705
706                     mem_addr1 = reg_I_readdata[11:0];
707                     mem_request = 1'b1;
708                     mem_writedata1 = bcd_hundreds;
709                     mem_WE1 = 1'b1;
710                 end else if(stage >= 32'hA & stage <= 32'hD)
                    ↪  begin
711                     to_bcd = reg_readdata1;
712                     reg_addr1 = instruction[11:8];
713
714                     mem_addr1 = reg_I_readdata + 12'h1;
```

```
715                     mem_request = 1'b1;
716                     mem_writedata1 = bcd_tens;
717                     mem_WE1 = 1'b1;
718                 end else if(stage >= 32'hE & stage <= 32'hF1)
                     ↪  begin
719                     to_bcd = reg_readdata1;
720                     reg_addr1 = instruction[11:8];
721
722                     mem_addr1 = reg_I_readdata + 12'h2;
723                     mem_request = 1'b1;
724                     mem_writedata1 = bcd_ones;
725                     mem_WE1 = 1'b1;
726                 end else begin
727                     //CPU DONE
728                 end
729             end
730
731             //memory module fix May 10
732             16'hFx55: begin //Fx55 - LD [I], Vx
733                 //Store registers V0 through Vx in memory
                     ↪  starting at location I
734                 //The interpreter copies the values of
                     ↪  registers V0 through Vx
735                 //into memory, starting at the address in I.
736                 if(stage >= 32'h7 & (stage[18:3] <=
                     ↪  instruction[11:8])) begin
737                     reg_addr1 = stage[6:3];
738
739                     alu_cmd = ALU_f_ADD;
740                     alu_in1 = reg_I_readdata;
741                     alu_in2 = stage[18:3];
742
743                     mem_addr1 = alu_out[11:0];
744                     mem_request = 1'b1;
745                     mem_writedata1 = reg_readdata1;
746                     mem_WE1 = &(stage[2:0]);
747
748                 end
```

```
749
750             end
751
752             //memory module fix May 10
753             16'hFx65: begin //Fx65 - LD Vx, [I]
754                 //Read registers V0 through Vx from memory
                    ↪ starting at location
755                 //I.
756                 //The interpreter reads values from memory
                    ↪ starting at location
757                 //I into registers V0 through Vx.
758                 if(stage >= 32'h7 & (stage[18:3] <=
                    ↪ instruction[11:8])) begin
759                     reg_addr1 = stage[6:3];
760
761                     alu_cmd = ALU_f_ADD;
762                     alu_in1 = reg_I_readdata;
763                     alu_in2 = stage[18:3];
764
765                     mem_addr1 = alu_out[11:0];
766                     mem_request = 1'b1;
767
768                     reg_writedata1 = mem_readdata1[7:0];
769                     reg_WE1 = &(stage[2:0]);
770
771                 end
772             end
773
774             default : /* default */;
775         endcase
776         end
777         /*END INSTRUCTION DECODE*/
778
779     end
780
781 endmodule
```

7.1.3 Chip8_rand_num_generator.sv

```
/*
 *
 * Semi-naive pseudo-random number generator
 *
 * Implemented by Levi
 *
 */
module Chip8_rand_num_generator(input logic cpu_clk, output
 ↪  logic[15:0] out);

    logic [15:0] rand_num;

    initial begin
        rand_num <= 16'b1111010111010010;
    end

    always_ff @(posedge cpu_clk) begin
        if(~|(rand_num[15:0])) begin
            rand_num[15:0] <= 16'b1111010111010010;
        end else begin
            rand_num[0] <= rand_num[15] ^ rand_num[14];
            rand_num[1] <= rand_num[14] ^ rand_num[13];
            rand_num[2] <= rand_num[13] ^ rand_num[12];
            rand_num[3] <= rand_num[12] ^ rand_num[11];
            rand_num[4] <= rand_num[11] ^ rand_num[10];
            rand_num[5] <= rand_num[10] ^ rand_num[9];
            rand_num[6] <= rand_num[9] ^ rand_num[8];
            rand_num[7] <= rand_num[8] ^ rand_num[7];
            rand_num[8] <= rand_num[7] ^ rand_num[6];
            rand_num[9] <= rand_num[6] ^ rand_num[5];
            rand_num[10] <= rand_num[5] ^ rand_num[4];
            rand_num[11] <= rand_num[4] ^ rand_num[3];
            rand_num[12] <= rand_num[3] ^ rand_num[2];
            rand_num[13] <= rand_num[2] ^ rand_num[1];
            rand_num[14] <= rand_num[1] ^ rand_num[0];
            rand_num[15] <= rand_num[0] ^ rand_num[15];
```

```
        end

        out <= rand_num;
    end
endmodule
```

7.1.4 Chip8_Stack.sv

```
`include "../enums.svh"

module Chip8_Stack(
    input logic         cpu_clk,    //clock
    input logic         reset,        //reset
    input STACK_OP         op,            //See enums.svh for
    ↪  ops
    input logic [15:0]    writedata,    //input PC
    output logic [15:0]    outdata        //data output
);

    logic [3:0]  address = 4'd0;
    logic [15:0] data;
    logic           wren;
    logic [15:0] q;

    stack_ram stack(address, cpu_clk, data, wren, q);

    logic[3:0]     stackptr = 4'h0;
    logic         secondcycle = 1'h0;
    logic         hold = 1'h0;

    always_ff @(posedge cpu_clk) begin
        if(reset) begin
            address <= 4'd0;
            hold <= 1'd0;
            wren <= 1'b0;
            secondcycle <= 1'h0;
            stackptr <= 4'd0;
```

```
29          end else begin
30              case (op)
31                  STACK_PUSH: begin
32                      if(~hold) begin
33                          address <= stackptr;
34                          data <= writedata;
35                          if(secondcycle == 1'h0) begin
36                              wren <= 1'h1;
37                              secondcycle <= 1'h1;
38                          end else begin
39                              wren <= 1'h0;
40                              stackptr <= stackptr + 4'b0001;
41                              secondcycle <= 1'h0;
42                              hold <= 1'b1;
43                          end
44                      end
45                  end
46                  STACK_POP: begin
47                      if(~hold) begin
48                          address <= stackptr - 4'b0001;
49                          wren <= 1'h0;
50                          outdata <= q;
51                          if(secondcycle == 1'h0) begin
52                              secondcycle <= 1'h1;
53                              stackptr <= stackptr - 4'b0001;
54                          end else begin
55                              secondcycle <= 1'h0;
56                              hold <= 1'b1;
57                          end
58                      end
59                  end
60                  STACK_HOLD: hold = 1'b0;
61                  default : /* default */;
62              endcase
63          end
64      end
65  endmodule
```

7.1.5 Chip8_ALU.sv

```
/**********************************************************************
 * Chip8_ALU.sv
 *
 * Simple ALU supporting instructions:
 *      - OR          - bitwise OR
 *      - AND          - bitwise AND
 *      - XOR         - bitwise XOR
 *      - ADD         - Addition
 *      - MINUS         - Subtract
 *      - LSHIFT     - Shift left
 *      - RSHIFT     - Shift right
 *      - EQUALS      - Equals compare
 *      - GREATER     - Greater than compare
 *      - INC          - Increment
 *
 * This module is solely used by the Chip8_CPU module, and
 ↪  relies on the ALU_f
 * enum defined in enums.svh
 *
 * AUTHORS: David Watkins, Ashley Kling
 * Dependencies:
 *      - enums.svh

 ↪  **********************************************************************

`include "../enums.svh"

module Chip8_ALU(
       input logic[15:0] input1, input2,
       input ALU_f sel,

       output logic[15:0] out,
       output logic alu_carry);

       logic[15:0] intermediate;

```

```
35          always_comb begin
36              case    (sel)
37
38                  ALU_f_OR : begin
39                      alu_carry = 0;
40                      out = input1 | input2;
41                  end
42
43                  ALU_f_AND : begin
44                      alu_carry = 0;
45                      out = input1 & input2;
46                  end
47
48                  ALU_f_XOR : begin
49                      alu_carry = 0;
50                      out = input1 ^ input2;
51                  end
52
53                  ALU_f_ADD : begin
54                      out = input1 + input2;
55                      alu_carry = |(out[15:8]);
56                  end
57
58                  ALU_f_MINUS : begin
59                      alu_carry = input1 > input2;
60                      out = input1 - input2;
61                  end
62
63                  ALU_f_LSHIFT : begin
64                      alu_carry = 0;
65                      out = input1 << input2;
66                  end
67
68                  ALU_f_RSHIFT : begin
69                      alu_carry = 0;
70                      out = input1 >> input2;
71                  end
72
```

```
                ALU_f_EQUALS : begin
                    alu_carry = 0;
                    out = (input1 == input2);
                end

                ALU_f_GREATER : begin
                    alu_carry = 0;
                    out = (input1 > input2);
                end

                ALU_f_INC : begin
                    alu_carry = 0;
                    out = input1 + 1'h1;
                end

                default: begin
                    alu_carry = 0;
                    out = 0;
                end
            endcase
        end
endmodule
```

7.1.6 Chip8_framebuffer.sv

```
/*************************************************************************
 * Chip8_framebuffer.sv
 *
 * Top level framebuffer module that contains the memory for
 ↪  the main view
 * and has wires into the VGA emulator to ouput video
 *
 * Built off of Stephen Edwards's VGA_LED code
 *
 * AUTHORS: David Watkins, Levi Oliver, Ashley Kling,
 ↪  Gabrielle Taylor
 * Dependencies:
```

```
 *  - Chip8_VGA_Emulator.sv
 *  - Framebuffer.v

 ↪  ********************************************************************************
module Chip8_framebuffer(
    input logic             clk,
    input logic             reset,

    input logic [4:0]    fb_addr_y,//max val = 31
    input logic [5:0]   fb_addr_x,//max val = 63
    input logic             fb_writedata, //data to write to
     ↪  addresse.
    input logic             fb_WE, //enable writing to address
    input logic         is_paused,

    output logic        fb_readdata, //data to write to
     ↪  addresse.

    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
);


    //The framebuffer memory has two ports. One is used
    //constantly and combinationaly by the VGA module.
    //The other one is general purpose. They are named
    //as such ("_addr" and "_general")
    wire[10:0] fb_addr_general = (fb_addr_y << 6) +
     ↪  (fb_addr_x);
    wire[10:0] fb_addr_vga;
    wire fb_writedata_general = fb_writedata;
    wire fb_writedata_vga;
    wire fb_WE_general = fb_WE;
    wire fb_WE_vga = 1'b0;//vga emulator will never write to
     ↪  FB mem
    wire fb_readdata_general;
    wire fb_readdata_vga;
```

```
44      assign fb_readdata = fb_readdata_general;
45
46      logic[10:0] counter;
47
48      initial begin
49          counter = 11'b0;
50      end
51
52      wire[10:0] copy_from_addr;
53      wire[10:0] copy_to_addr;
54      wire copy_data;
55      wire copyWE;
56      wire deadwire;
57
58      logic [31:0] fb_stage;
59      logic [31:0] time_since_last_copy;
60
61      initial begin
62          time_since_last_copy <= 32'h0;
63          fb_stage <= 32'h0;
64      end
65
66      always_ff @(posedge clk) begin
67          if(fb_WE) begin
68              fb_stage <= 32'h0;
69          end else if(fb_stage < FRAMEBUFFER_REFRESH_HOLD) begin
70              fb_stage <= fb_stage;
71          end else begin
72              fb_stage <= fb_stage + 32'h1;
73          end
74      end
75
76      always_ff @(posedge clk) begin
77          if(fb_stage >= FRAMEBUFFER_REFRESH_HOLD ||
            ↪ time_since_last_copy > COPY_THRESHOLD) begin
78              counter <= counter + 1;
79
80              copyWE <= 1'b1;
```

```
81
82              copy_from_addr <= counter + 11'h1;
83              copy_to_addr <= counter;
84
85              if(counter == 11'b111_1111_1111)
                ↪  time_since_last_copy <= 32'h0;
86          end else begin
87              copyWE <= 1'b0;
88              time_since_last_copy <= time_since_last_copy +
                ↪  32'h1;
89              counter <= 11'h0;
90          end
91      end
92
93      Chip8_VGA_Emulator led_emulator(
94              .clk50(clk),
95              .reset(reset),
96              .fb_pixel_data(fb_readdata_vga),
97              .fb_request_addr(fb_addr_vga),
98              .is_paused(is_paused),
99              .VGA_R(VGA_R),
100             .VGA_G(VGA_G),
101             .VGA_B(VGA_B),
102             .VGA_CLK(VGA_CLK),
103             .VGA_HS(VGA_HS),
104             .VGA_VS(VGA_VS),
105             .VGA_BLANK_n(VGA_BLANK_n),
106             .VGA_SYNC_n(VGA_SYNC_n)
107     );
108
109     Framebuffer from_cpu (
110             .clock(clk),
111             .address_a(fb_addr_general),
112             .address_b(copy_from_addr),
113             .data_a(fb_writedata_general),
114             .data_b(fb_writedata_vga),
115             .wren_a(fb_WE_general),
116             .wren_b(fb_WE_vga),
```

```
117             .q_a(fb_readdata_general),
118             .q_b(copy_data)
119     );
120
121     Framebuffer toscreen (
122             .clock(clk),
123             .address_a(fb_addr_vga),
124             .address_b(copy_to_addr),
125             .data_a(fb_writedata_vga),
126             .data_b(copy_data),
127             .wren_a(fb_WE_vga),
128             .wren_b(copyWE),
129             .q_a(fb_readdata_vga),
130             .q_b(deadwire)
131     );
132
133
134 endmodule
```

7.1.7 Chip8_VGA_Emulator.sv

```
1  /*
2   * Chip8-Framebuffer to VGA module.
3   * Adjusts the dimensions of the screen so that it appears 8
    ↪  times larger.
4   *
5   * Developed by Levi and Ash
6   *
7   * Built off of Stephen Edwards's code
8   * Columbia University
9   */
10
11  module Chip8_VGA_Emulator(
12      input logic        clk50, reset,
13      //input logic [2047:0] framebuffer,
14     input logic fb_pixel_data,
15     input logic is_paused,
```

```
16      output logic[10:0] fb_request_addr,
17       output logic [7:0] VGA_R, VGA_G, VGA_B,
18      output logic       VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n,
         ↪  VGA_SYNC_n);
19
20      /*
21      * 640 X 480 VGA timing for a 50 MHz clock: one pixel
    ↪   every other cycle
22      *
23      * HCOUNT 1599 0             1279       1599 0
24      *             _______________              ________
25      * ___________|    Video      |____________|  Video
26      *
27      *
28      * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
29      *       _______________________      _____________
30      * |____|       VGA_HS          |____|
31      */
32      // Parameters for hcount
33      parameter HACTIVE      = 11'd 1280,
34      HFRONT_PORCH = 11'd 32,
35      HSYNC        = 11'd 192,
36      HBACK_PORCH  = 11'd 96,
37      HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
         ↪  HBACK_PORCH; // 1600
38
39      // Parameters for vcount
40      parameter VACTIVE      = 10'd 480,
41      VFRONT_PORCH = 10'd 10,
42      VSYNC        = 10'd 2,
43      VBACK_PORCH  = 10'd 33,
44      VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC +
         ↪  VBACK_PORCH; // 525
45
46      logic [10:0] hcount; // Horizontal counter //
         ↪  Hcount[10:1] indicates pixel column (0-639)
47      logic endOfLine;
48
```

```
49      always_ff @(posedge clk50 or posedge reset)
50      if (reset)          hcount <= 0;
51      else if (endOfLine) hcount <= 0;
52      else                hcount <= hcount + 11'd 1;
53
54      assign endOfLine = hcount == HTOTAL - 1;
55
56      // Vertical counter
57      logic [9:0]                 vcount;
58      logic                   endOfField;
59
60      always_ff @(posedge clk50 or posedge reset)
61      if (reset)          vcount <= 0;
62      else if (endOfLine)
63      if (endOfField)   vcount <= 0;
64      else              vcount <= vcount + 10'd 1;
65
66      assign endOfField = vcount == VTOTAL - 1;
67
68      // Horizontal sync: from 0x520 to 0x5DF (0x57F)
69      // 101 0010 0000 to 101 1101 1111
70      assign VGA_HS = !( (hcount[10:8] == 3'b101) &
        ↪  !(hcount[7:5] == 3'b111));
71      assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH)
        ↪  / 2);
72
73      assign VGA_SYNC_n = 1; // For adding sync to video
        ↪  signals; not used for VGA
74
75      // Horizontal active: 0 to 1279     Vertical active: 0 to
        ↪  479
76      // 101 0000 0000  1280           01 1110 0000  480
77      // 110 0011 1111  1599           10 0000 1100  524
78      assign VGA_BLANK_n = !( hcount[10] & (hcount[9] |
        ↪  hcount[8]) ) &
79      !( vcount[9] | (vcount[8:5] == 4'b1111) );
80
81      /* VGA_CLK is 25 MHz
```

```
82      *               __    __    __
83      * clk50    __|  |__|  |__|
84      *
85      *              _____       __
86      * hcount[0]__|     |_____|
87      */
88      assign VGA_CLK = hcount[0]; // 25 MHz clock: pixel
        ↪  latched on rising edge
89
90      parameter chip_hend = 7'd 64;
91      parameter chip_vend = 6'd 32;
92
93      parameter left_bound = 7'd64;
94      parameter right_bound = 10'd576;
95      parameter top_bound = 7'd112;
96      parameter bottom_bound = 9'd368;
97
98      logic[11:0] fb_pos;
99      assign fb_request_addr = ((((vcount[8:0] - top_bound) &
        ↪  (8'b1111_1000)) << (4'd3)) + ((hcount[10:1] -
        ↪  left_bound) >> (4'd3)));
100                             ////((vcount[8:0] - top_bound)
                                ↪  >> (4'd3))*(7'd64)) +
                                ↪  ((hcount[10:1] -
                                ↪  left_bound) >> (4'd3));
101                             //this commented out line is
                                ↪  the old code (which was
                                ↪  believed to work). New
                                ↪  code is more efficient
102                             //in area and time. Hopefully
                                ↪  this helps if we have a
                                ↪  too long critical path
103
104
105     logic inChip;
106        //120 <= Y-dim < 360
107        //64 <= X-dim < 576
108
```

```
109     /*
110     *    +-------------------------------+
111     *    | VGA Screen (640x480)          |
112     *    |   64                      576 |
113     *    |    +---------------------+112 |
114     *    |    |      Chip8 Screen   |    |
115     *    |    |      (64*8x32*8)    |    |
116     *    |    +---------------------+368 |
117     *    |                               |
118     *    +-------------------------------+
119        */
120
121
122 //       assign inChip = (((hcount[10:1]) >= (chip_hend *
    ↪  (5'd8) + 7'd64)) & (((hcount[10:1]) < (chip_hend * (5'd8)
    ↪  + 10'd576)) &
123 //                ((vcount[8:0]) >= (chip_vend * (5'd8) +
    ↪  7'd112))  & ((vcount[8:0]) < (chip_vend * (5'd8) +
    ↪  10'd368));
124
125
126         assign inChip = (((hcount[10:1]) > (left_bound))     &
127                         ((hcount[10:1]) <
                            ↪  (right_bound))    &
128                         ((vcount[8:0])  > (top_bound))     &
129                         ((vcount[8:0])) <  (bottom_bound));
130
131
132         /**
133          * 16 columns, 8 rows
134          * +---------------------------------------+
135          * |      px24                             |
136          * |   px48    === === = = === === ===  px432   |
137          * |        = = = = = = =   =   = ==       |
138          * |        === === = = === === =  =       |
139          * |        =   = = = =   = =   = ==       |
140          * |        =   = = === === === ===  px88  |
141          *
```

```
142          */
143
144         parameter paused_left = 10'd64;
145         parameter paused_right = 10'd576;
146         parameter paused_top = 10'd24;
147         parameter paused_bottom = 10'd88;
148
149         logic [9:0] hcount_offseted, vcount_offseted;
150         assign hcount_offseted = (hcount[10:1] - paused_left)
            ↪  >> 4;
151         assign vcount_offseted = (vcount[8:0] - paused_top) >>
            ↪  3;
152
153          reg  [31:0] romdata [7:0];
154
155         initial begin
156             romdata[7] =
                ↪  32'b0000_1110_1110_1010_1110_1110_1110_0000;
157             romdata[6] =
                ↪  32'b0000_1010_1010_1010_1000_1000_1011_0000;
158             romdata[5] =
                ↪  32'b0000_1110_1110_1010_1110_1110_1001_0000;
159             romdata[4] =
                ↪  32'b0000_1000_1010_1010_0010_1000_1011_0000;
160             romdata[3] =
                ↪  32'b0000_1000_1010_1110_1110_1110_1110_0000;
161             romdata[2] = 32'b0;
162             romdata[1] = 32'b0;
163             romdata[0] = 32'b0;
164         end
165         assign inPaused = is_paused & (
166                 ((hcount[10:1]) >= (paused_left)) &
167                 ((hcount[10:1]) < (paused_right)) &
168                 ((vcount[8:0]) >= (paused_top)) &
169                 ((vcount[8:0]) < (paused_bottom)) &
170                 romdata[vcount_offseted][hcount_offseted]
171             );
172
```

```
173     always_comb begin
174             {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0}; //
                ↪ Black
175             if (inChip & fb_pixel_data)
                ↪ begin//framebuffer[fb_pos]) begin
176                 //White to show on-pixel
177               {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
178           end else if(inChip) begin
179               //purple to show general area
180               {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'hFF};
181           end else if(inPaused) begin
182               {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
183           end
184       end
185
186 endmodule // VGA_LED_Emulator
```

7.1.8 audio_codec.sv

```
1  // Original audio codec code taken from
2  //Howard Mao's FPGA blog
3  //http://zhehaomao.com/blog/fpga/2014/01/15/sockit-8.html
4  //MOdified as needed
5
6  /*
7  audio_codec.sv
8  Sends samples to the audio codec ssm 2603 at audio clock
   ↪ rate.
9  */
10
11
12 //Audio codec interface
13 module audio_codec (
14     input  clk,
15     input  reset,
16     output [1:0]  sample_end,
17     output [1:0]  sample_req,
```

```
18      input  [15:0] audio_output,
19      output [15:0] audio_input,
20      // 1 - left, 0 - right
21      input  [1:0] channel_sel,
22
23      output AUD_ADCLRCK,
24      input AUD_ADCDAT,
25      output AUD_DACLRCK,
26      output AUD_DACDAT,
27      output AUD_BCLK
28  );
29
30  reg [7:0] lrck_divider;
31  reg [1:0] bclk_divider;
32
33  reg [15:0] shift_out;
34  reg [15:0] shift_temp;
35  reg [15:0] shift_in;
36
37  wire lrck = !lrck_divider[7];
38
39  assign AUD_ADCLRCK = lrck;
40  assign AUD_DACLRCK = lrck;
41  assign AUD_BCLK = bclk_divider[1];
42  assign AUD_DACDAT = shift_out[15];
43
44  always @(posedge clk) begin
45      if (reset) begin
46          lrck_divider <= 8'hff;
47          bclk_divider <= 2'b11;
48      end else begin
49          lrck_divider <= lrck_divider + 1'b1;
50          bclk_divider <= bclk_divider + 1'b1;
51      end
52  end
53
54  assign sample_end[1] = (lrck_divider == 8'h40);
55  assign sample_end[0] = (lrck_divider == 8'hc0);
```

```
assign audio_input = shift_in;
assign sample_req[1] = (lrck_divider == 8'hfe);
assign sample_req[0] = (lrck_divider == 8'h7e);

wire clr_lrck = (lrck_divider == 8'h7f);
wire set_lrck = (lrck_divider == 8'hff);
// high right after bclk is set
wire set_bclk = (bclk_divider == 2'b10 && !lrck_divider[6]);
// high right before bclk is cleared
wire clr_bclk = (bclk_divider == 2'b11 && !lrck_divider[6]);

always @(posedge clk) begin
    if (reset) begin
        shift_out <= 16'h0;
        shift_in <= 16'h0;
        shift_in <= 16'h0;
    end else if (set_lrck || clr_lrck) begin
        // check if current channel is selected
        if (channel_sel[set_lrck]) begin
            shift_out <= audio_output;
            shift_temp <= audio_output;
            shift_in <= 16'h0;
        // repeat the sample from the other channel if not
        end else shift_out <= shift_temp;
    end else if (set_bclk == 1) begin
        // only read in if channel is selected
        if (channel_sel[lrck])
            shift_in <= {shift_in[14:0], AUD_ADCDAT};
    end else if (clr_bclk == 1) begin
        shift_out <= {shift_out[14:0], 1'b0};
    end
end

endmodule
```

7.1.9 audio_effects.sv

```
//Original audio codec code taken from
//Howard Mao's FPGA blog
//http://zhehaomao.com/blog/fpga/2014/01/15/sockit-8.html
//MOdified as needed

/* audio_effects.sv
    Sends hardcoded beep samples to audio codec interface
*/

module audio_effects (
    input  audio_clk,
    input  main_clk,
    input  reset,

    input  sample_end,
    input  sample_req,

    output [15:0] audio_output,
    input  [15:0] audio_input,

    input    control
);

reg  [15:0] romdata [0:99];
reg  [6:0]  index = 7'd0;
reg  [15:0] last_sample;
reg  [15:0] dat;
wire [15:0] filter_output;
wire filter_finish;

assign audio_output = dat;

parameter SINE     = 0;
parameter FEEDBACK = 1;
parameter FILTER   = 2;

```

```
37 parameter SINE_LAST = 7'd99;
38
39 initial begin
40     romdata[0] = 16'h0000;
41     romdata[1] = 16'h0805;
42     romdata[2] = 16'h1002;
43     romdata[3] = 16'h17ee;
44     romdata[4] = 16'h1fc3;
45     romdata[5] = 16'h2777;
46     romdata[6] = 16'h2f04;
47     romdata[7] = 16'h3662;
48     romdata[8] = 16'h3d89;
49     romdata[9] = 16'h4472;
50     romdata[10] = 16'h4b16;
51     romdata[11] = 16'h516f;
52     romdata[12] = 16'h5776;
53     romdata[13] = 16'h5d25;
54     romdata[14] = 16'h6276;
55     romdata[15] = 16'h6764;
56     romdata[16] = 16'h6bea;
57     romdata[17] = 16'h7004;
58     romdata[18] = 16'h73ad;
59     romdata[19] = 16'h76e1;
60     romdata[20] = 16'h799e;
61     romdata[21] = 16'h7be1;
62     romdata[22] = 16'h7da7;
63     romdata[23] = 16'h7eef;
64     romdata[24] = 16'h7fb7;
65     romdata[25] = 16'h7fff;
66     romdata[26] = 16'h7fc6;
67     romdata[27] = 16'h7f0c;
68     romdata[28] = 16'h7dd3;
69     romdata[29] = 16'h7c1b;
70     romdata[30] = 16'h79e6;
71     romdata[31] = 16'h7737;
72     romdata[32] = 16'h7410;
73     romdata[33] = 16'h7074;
74     romdata[34] = 16'h6c67;
```

```
75      romdata[35] = 16'h67ed;
76      romdata[36] = 16'h630a;
77      romdata[37] = 16'h5dc4;
78      romdata[38] = 16'h5820;
79      romdata[39] = 16'h5222;
80      romdata[40] = 16'h4bd3;
81      romdata[41] = 16'h4537;
82      romdata[42] = 16'h3e55;
83      romdata[43] = 16'h3735;
84      romdata[44] = 16'h2fdd;
85      romdata[45] = 16'h2855;
86      romdata[46] = 16'h20a5;
87      romdata[47] = 16'h18d3;
88      romdata[48] = 16'h10e9;
89      romdata[49] = 16'h08ee;
90      romdata[50] = 16'h00e9;
91      romdata[51] = 16'hf8e4;
92      romdata[52] = 16'hf0e6;
93      romdata[53] = 16'he8f7;
94      romdata[54] = 16'he120;
95      romdata[55] = 16'hd967;
96      romdata[56] = 16'hd1d5;
97      romdata[57] = 16'hca72;
98      romdata[58] = 16'hc344;
99      romdata[59] = 16'hbc54;
100     romdata[60] = 16'hb5a7;
101     romdata[61] = 16'haf46;
102     romdata[62] = 16'ha935;
103     romdata[63] = 16'ha37c;
104     romdata[64] = 16'h9e20;
105     romdata[65] = 16'h9926;
106     romdata[66] = 16'h9494;
107     romdata[67] = 16'h906e;
108     romdata[68] = 16'h8cb8;
109     romdata[69] = 16'h8976;
110     romdata[70] = 16'h86ab;
111     romdata[71] = 16'h845a;
112     romdata[72] = 16'h8286;
```

```
113     romdata[73] = 16'h8130;
114     romdata[74] = 16'h8059;
115     romdata[75] = 16'h8003;
116     romdata[76] = 16'h802d;
117     romdata[77] = 16'h80d8;
118     romdata[78] = 16'h8203;
119     romdata[79] = 16'h83ad;
120     romdata[80] = 16'h85d3;
121     romdata[81] = 16'h8875;
122     romdata[82] = 16'h8b8f;
123     romdata[83] = 16'h8f1d;
124     romdata[84] = 16'h931e;
125     romdata[85] = 16'h978c;
126     romdata[86] = 16'h9c63;
127     romdata[87] = 16'ha19e;
128     romdata[88] = 16'ha738;
129     romdata[89] = 16'had2b;
130     romdata[90] = 16'hb372;
131     romdata[91] = 16'hba05;
132     romdata[92] = 16'hc0df;
133     romdata[93] = 16'hc7f9;
134     romdata[94] = 16'hcf4b;
135     romdata[95] = 16'hd6ce;
136     romdata[96] = 16'hde7a;
137     romdata[97] = 16'he648;
138     romdata[98] = 16'hee30;
139     romdata[99] = 16'hf629;
140 end
141 
142 always @(*) begin
143 
144     if (control)
145         dat <= romdata[index];
146     else
147         dat <= 16'd0;
148 end
149 
150 always @(posedge audio_clk) begin
```

```
151     if (sample_req) begin
152         if (index == SINE_LAST)
153             index <= 7'd00;
154         else
155             index <= index + 1'b1;
156     end
157 end
158 
159 
160 endmodule
```

7.1.10 Chip8_SoundController.sv

```
1  /*****************************************************************************
2   * Chip8_SoundController.sv
3   *
4   * Top level controller for audio output on the Chip8
5   * Designed to interact directly with a Altera SoCKit
    ↪  Cyclone V board
6   * Source originally borrowed from:
7   *  Howard Mao's FPGA blog
8   *  http://zhehaomao.com/blog/fpga/2014/01/15/sockit-8.html
9   *
10  * AUTHORS: David Watkins, Gabrielle Taylor
11  * Dependencies:
12  *  - Chip8_Sound/clock_pll.v
13  *  - Chip8_Sound/audio_effects.sv
14  *  - Chip8_Sound/i2c_av_config.sv
15  *  - Chip8_Sound/audio_codec.sv
16  
    ↪  *****************************************************************************
17  
18  module Chip8_SoundController (
19      input  OSC_50_B8A,   //reference clock
20      inout  AUD_ADCLRCK, //Channel clock for ADC
21      input  AUD_ADCDAT,
22      inout  AUD_DACLRCK, //Channel clock for DAC
```

```
23      output AUD_DACDAT,  //DAC data
24      output AUD_XCK,
25      inout  AUD_BCLK, // Bit clock
26      output AUD_I2C_SCLK, //I2C clock
27      inout  AUD_I2C_SDAT, //I2C data
28      output AUD_MUTE,   //Audio mute
29
30      input logic clk,
31      input logic is_on, //Turn on the output
32      input logic reset
33  );
34
35  wire main_clk;
36  wire audio_clk;
37
38  wire [1:0] sample_end;
39  wire [1:0] sample_req;
40  wire [15:0] audio_output;
41  wire [15:0] audio_input;
42
43  wire [3:0] status;
44
45  clock_pll pll (
46      .refclk (OSC_50_B8A),
47      .rst (reset),
48      .outclk_0 (audio_clk),
49      .outclk_1 (main_clk)
50  );
51
52  i2c_av_config av_config (
53      .clk (main_clk),
54      .reset (reset),
55      .i2c_sclk (AUD_I2C_SCLK),
56      .i2c_sdat (AUD_I2C_SDAT),
57      .status (status)
58  );
59
60  assign AUD_XCK = audio_clk;
```

```
61 assign AUD_MUTE = is_on;
62
63 audio_codec ac (
64     .clk (audio_clk),
65     .reset (reset),
66     .sample_end (sample_end),
67     .sample_req (sample_req),
68     .audio_output (audio_output),
69     .audio_input (audio_input),
70     .channel_sel (2'b10),
71
72     .AUD_ADCLRCK (AUD_ADCLRCK),
73     .AUD_ADCDAT (AUD_ADCDAT),
74     .AUD_DACLRCK (AUD_DACLRCK),
75     .AUD_DACDAT (AUD_DACDAT),
76     .AUD_BCLK (AUD_BCLK)
77 );
78
79 audio_effects ae (
80     .audio_clk (audio_clk),
81     .main_clk (main_clk),
82     .sample_end (sample_end[1]),
83     .sample_req (sample_req[1]),
84     .audio_output (audio_output),
85     .audio_input  (audio_input),
86     .control (1'b1)
87 );
88
89 endmodule
```

7.1.11 i2c_av_config.sv

```
1 // Original audio codec code taken from
2 //Howard Mao's FPGA blog
3 //http://zhehaomao.com/blog/fpga/2014/01/15/sockit-8.html
4 //MOdified as needed
5
```

```
6  //configure Audio codec using the I2C protocol
7  module i2c_av_config (
8      input clk,
9      input reset,
10
11     output i2c_sclk,
12     inout  i2c_sdat,
13
14     output [3:0] status
15 );
16
17 reg [23:0] i2c_data;
18 reg [15:0] lut_data;
19 reg [3:0]  lut_index = 4'd0;
20
21 parameter LAST_INDEX = 4'ha;
22
23 reg  i2c_start = 1'b0;
24 wire i2c_done;
25 wire i2c_ack;
26
27 i2c_controller control (
28     .clk (clk),
29     .i2c_sclk (i2c_sclk),
30     .i2c_sdat (i2c_sdat),
31     .i2c_data (i2c_data),
32     .start (i2c_start),
33     .done (i2c_done),
34     .ack (i2c_ack)
35 );
36
37 always @(*) begin
38     case (lut_index)
39         4'h0: lut_data <= 16'h0c10; // power on everything
           ↪  except out
40         4'h1: lut_data <= 16'h0017; // left input
41         4'h2: lut_data <= 16'h0217; // right input
42         4'h3: lut_data <= 16'h0479; // left output
```

```
43          4'h4: lut_data <= 16'h0679; // right output
44          4'h5: lut_data <= 16'h08d4; // analog path
45          4'h6: lut_data <= 16'h0a04; // digital path
46          4'h7: lut_data <= 16'h0e01; // digital IF
47          4'h8: lut_data <= 16'h1020; // sampling rate
48          4'h9: lut_data <= 16'h0c00; // power on everything
49          4'ha: lut_data <= 16'h1201; // activate
50          default: lut_data <= 16'h0000;
51      endcase
52  end
53
54  reg [1:0] control_state = 2'b00;
55
56  assign status = lut_index;
57
58  always @(posedge clk) begin
59      if (reset) begin
60          lut_index <= 4'd0;
61          i2c_start <= 1'b0;
62          control_state <= 2'b00;
63      end else begin
64          case (control_state)
65              2'b00: begin
66                  i2c_start <= 1'b1;
67                  i2c_data <= {8'h34, lut_data};
68                  control_state <= 2'b01;
69              end
70              2'b01: begin
71                  i2c_start <= 1'b0;
72                  control_state <= 2'b10;
73              end
74              2'b10: if (i2c_done) begin
75                  if (i2c_ack) begin
76                      if (lut_index == LAST_INDEX)
77                          control_state <= 2'b11;
78                      else begin
79                          lut_index <= lut_index + 1'b1;
80                          control_state <= 2'b00;
```

```
                    end
                end else
                    control_state <= 2'b00;
            end
        endcase
    end
end

endmodule
```

7.1.12 i2c_controller.sv

```
// Original audio codec code taken from
//Howard Mao's FPGA blog
//http://zhehaomao.com/blog/fpga/2014/01/15/sockit-8.html
//MOdified as needed

//implement the I2C protocol to configure registers in ssm 2603 audio codec
module i2c_controller (
    input  clk,

    output i2c_sclk,    //i2c clock
    inout  i2c_sdat,    //i2c data  out

    input  start,
    output done,
    output ack,

    input [23:0] i2c_data
);

reg [23:0] data;

reg [4:0] stage;
reg [6:0] sclk_divider;
reg clock_en = 1'b0;
```

```
25
26  // don't toggle the clock unless we're sending data
27  // clock will also be kept high when sending START and STOP
    ↪  symbols
28  assign i2c_sclk = (!clock_en) || sclk_divider[6];
29  wire midlow = (sclk_divider == 7'h1f);
30
31  reg sdat = 1'b1;
32  // rely on pull-up resistor to set SDAT high
33  assign i2c_sdat = (sdat) ? 1'bz : 1'b0;
34
35  reg [2:0] acks;
36
37  parameter LAST_STAGE = 5'd29;
38
39  assign ack = (acks == 3'b000);
40  assign done = (stage == LAST_STAGE);
41
42
43  //implementing I2C protocol
44  always @(posedge clk) begin
45      if (start) begin
46          sclk_divider <= 7'd0;
47          stage <= 5'd0;
48          clock_en = 1'b0;
49          sdat <= 1'b1;
50          acks <= 3'b111;
51          data <= i2c_data;
52      end else begin
53          if (sclk_divider == 7'd127) begin
54              sclk_divider <= 7'd0;
55
56              if (stage != LAST_STAGE)
57                  stage <= stage + 1'b1;
58
59              case (stage)
60                  // after start
61                  5'd0:  clock_en <= 1'b1;
```

```
62                  // receive acks
63                  5'd9:  acks[0] <= i2c_sdat;
64                  5'd18: acks[1] <= i2c_sdat;
65                  5'd27: acks[2] <= i2c_sdat;
66                  // before stop
67                  5'd28: clock_en <= 1'b0;
68              endcase
69          end else
70              sclk_divider <= sclk_divider + 1'b1;
71
72          if (midlow) begin
73              case (stage)
74                  // start
75                  5'd0:  sdat <= 1'b0;
76                  // byte 1
77                  5'd1:  sdat <= data[23];
78                  5'd2:  sdat <= data[22];
79                  5'd3:  sdat <= data[21];
80                  5'd4:  sdat <= data[20];
81                  5'd5:  sdat <= data[19];
82                  5'd6:  sdat <= data[18];
83                  5'd7:  sdat <= data[17];
84                  5'd8:  sdat <= data[16];
85                  // ack 1
86                  5'd9:  sdat <= 1'b1;
87                  // byte 2
88                  5'd10: sdat <= data[15];
89                  5'd11: sdat <= data[14];
90                  5'd12: sdat <= data[13];
91                  5'd13: sdat <= data[12];
92                  5'd14: sdat <= data[11];
93                  5'd15: sdat <= data[10];
94                  5'd16: sdat <= data[9];
95                  5'd17: sdat <= data[8];
96                  // ack 2
97                  5'd18: sdat <= 1'b1;
98                  // byte 3
99                  5'd19: sdat <= data[7];
```

```
                5'd20: sdat <= data[6];
                5'd21: sdat <= data[5];
                5'd22: sdat <= data[4];
                5'd23: sdat <= data[3];
                5'd24: sdat <= data[2];
                5'd25: sdat <= data[1];
                5'd26: sdat <= data[0];
                // ack 3
                5'd27: sdat <= 1'b1;
                // stop
                5'd28: sdat <= 1'b0;
                5'd29: sdat <= 1'b1;
            endcase
        end
    end
end

endmodule
```

7.1.13 clk_div.sv

```
/*************************************************************************
 * clk_div.sv
 *
 * A clock divider module for outputting a 60 Hz clock from
 ↪  a 50 MHz clock
 *
 * AUTHORS: Gabrielle Taylor
 * Dependencies:

 ↪  *************************************************************************

module clk_div (
    input logic clk_in,
    input logic reset,    //resets on high
    output logic clk_out);

```

```
    //Input: 50 MHz clock
    //Output: 60 Hz clock
    //NOTE - Actual output clock frequency: 60.000024 Hz
    //Calculation:
    //50 MHz = 50,000,000 hz
    //50,000,000 hz / 60 hz = 833,333.33
    //Scaling factor rounded to 833,333

    logic [19:0] count = 20'd0;           //counts up to
     ↪  833333
    logic [19:0] stop = 20'd833333;       //833333 in hex

    always @(posedge clk_in)
    begin
        if(~reset) begin
            clk_out <= 1'b0;
            if (count==stop) begin
                count <= 20'd0;
                clk_out <= 1'b1; //set clock high for one 60
                 ↪  MHz cycle
            end else begin
                count <= count + 20'd1;
            end
        end else begin
            count <= 20'd0;
            clk_out <= 1'b0;
        end
    end
endmodule
```

7.1.14 timer.sv

```
/*************************************************************************
 * timer.sv
 *
 * Module for outputting a count down value based on a 60 Hz
 ↪  clock
```

```
 * Used for both the sound_timer and delay_timer in the Chip8_Top module
 *
 * AUTHORS: David Watkins, Gabrielle Taylor
 * Dependencies:

 ************************************************************************

module timer (
        input logic write_enable,             //Write enable
        input logic clk,        //50 MHz clock
        input logic clk_60,         //60 Hz clock
        input logic [7:0] data,
        output logic out,
        output logic [7:0] output_data);

        logic [7:0] delay_reg = 8'b0000_0000;

        always @(posedge clk)
        begin
            if(write_enable) begin
                delay_reg <= data;
            end else if (clk_60 & (|delay_reg)) begin
                delay_reg <= delay_reg - 8'd1;
            end
            out <= |delay_reg;
            output_data <= delay_reg;
        end

endmodule
```

7.1.15 Chip8_Top.sv

```

 /************************************************************************
 * Chip8_Top.sv
 *
```

```
4  * Top level Chip8 module that controls all other modules
5  *
6  * AUTHORS: David Watkins, Levi Oliver, Ashley Kling,
   ↪  Gabrielle Taylor
7  * Dependencies:
8  *  - Chip8_SoundController.sv
9  *  - Chip8_framebuffer.sv
10 *  - timer.sv
11 *  - clk_div.sv
12 *  - memory.v
13 *  - reg_file.v
14 *  - enums.svh
15 *  - utils.svh
16 *  - Chip8_CPU.sv
17
   ↪  ************************************************************************
18
19 `include "enums.svh"
20 `include "utils.svh"
21
22 module Chip8_Top(
23    input logic         clk,
24    input logic         reset,
25    input logic [31:0]  writedata,
26    input logic         write,
27    input               chipselect,
28    input logic [17:0]  address,
29
30    output logic [31:0] data_out,
31
32    //VGA Output
33    output logic [7:0]  VGA_R, VGA_G, VGA_B,
34    output logic        VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n,
35    output logic        VGA_SYNC_n,
36
37    //Audio Output
38    input  OSC_50_B8A,      //reference clock
39    inout  AUD_ADCLRCK,     //Channel clock for ADC
```

```
40      input  AUD_ADCDAT,
41      inout  AUD_DACLRCK,      //Channel clock for DAC
42      output AUD_DACDAT,       //DAC data
43      output AUD_XCK,
44      inout  AUD_BCLK,         //Bit clock
45      output AUD_I2C_SCLK,     //I2C clock
46      inout  AUD_I2C_SDAT,     //I2C data
47      output AUD_MUTE          //Audio mute
48      );
49
50      //Index register
51      logic [15:0] I;
52
53      //Program counter
54      logic [7:0]  pc_state;
55      logic [11:0] pc = 12'h200;
56      logic [11:0] next_pc;
57      logic [31:0] stage;
58      logic        halt_for_keypress;
59      logic [31:0] last_stage;
60      //Framebuffer values
61      logic        fbreset;
62      logic [4:0]    fb_addr_y;//max val = 31
63      logic [5:0] fb_addr_x;//max val = 63
64      logic        fb_writedata; //data to write to addresse.
65      logic        fb_WE; //enable writing to address
66      logic        fb_readdata; //data to write to addresse.
67      logic       fb_paused;
68
69      //Keyboard
70      logic       ispressed;
71      logic [3:0] key;
72
73      //Memory
74      logic [7:0]  memwritedata1, memwritedata2;
75      logic        memWE1, memWE2;
76      logic [11:0] memaddr1, memaddr2;
77      logic [7:0]  memreaddata1, memreaddata2;
```

```
78
79      //Reg file
80      logic [7:0] reg_writedata1, reg_writedata2;
81      logic       regWE1, regWE2;
82      logic [3:0] reg_addr1, reg_addr2;
83      logic [7:0] reg_readdata1, reg_readdata2;
84
85      //CPU
86      logic [15:0] cpu_instruction;
87      logic        cpu_delay_timer_WE, cpu_sound_timer_WE;
88      logic [7:0]  cpu_delay_timer_writedata,
        ↪  cpu_sound_timer_writedata;
89      PC_SRC       cpu_pc_src;
90      logic [11:0] cpu_PC_writedata;
91      logic        cpu_reg_WE1, cpu_reg_WE2;
92      logic [3:0]  cpu_reg_addr1, cpu_reg_addr2;
93      logic [7:0]  cpu_reg_writedata1, cpu_reg_writedata2;
94      logic        cpu_mem_WE1, cpu_mem_WE2;
95      logic [11:0] cpu_mem_addr1, cpu_mem_addr2;
96      logic        cpu_mem_request;
97      logic [ 7:0] cpu_mem_writedata1, cpu_mem_writedata2;
98      logic        cpu_reg_I_WE;
99      logic [15:0] cpu_reg_I_writedata;
100     logic        cpu_fbreset;
101     logic [4:0]     cpu_fb_addr_y;
102     logic [5:0]     cpu_fb_addr_x;
103     logic         cpu_fb_writedata;
104     logic         cpu_fb_readdata;
105     logic         cpu_fb_WE;
106     logic        cpu_halt_for_keypress;
107     logic        cpu_stk_reset;
108     STACK_OP     cpu_stk_op;
109     logic[15:0]  cpu_stk_writedata;
110     logic        cpu_bit_overwritten;
111     logic        cpu_is_drawing;
112
113     //Sound
114     logic sound_on;
```

```
115     logic sound_reset;
116
117     //Timers
118     logic       clk_div_reset, clk_div_clk_out;
119     logic       delay_timer_write_enable, delay_timer_out;
120     logic [7:0] delay_timer_data, delay_timer_output_data;
121     logic       sound_timer_write_enable, sound_timer_out;
122     logic [7:0] sound_timer_data, sound_timer_output_data;
123
124     //Stack
125     logic           stack_reset;
126     STACK_OP        stack_op;
127     logic [15:0]    stack_writedata;
128     logic [15:0]    stack_outdata;
129
130     //State
131     Chip8_STATE state = Chip8_PAUSED;
132     logic       bit_ovewritten;
133     logic       is_drawing;
134
135     //ARM Registers
136     logic [5:0]  fbvx_prev;
137     logic [4:0]  fbvy_prev;
138     logic [11:0] mem_addr_prev;
139     logic        chipselect_happened;
140
141     //Chipselect temporary values
142     logic [3:0] chip_reg_addr1_prev;
143     logic       chip_regWE1_prev;
144     logic [15:0]chip_reg_writedata1_prev;
145     logic [5:0] chip_fb_addr_x_prev;
146     logic [4:0] chip_fb_addr_y_prev;
147     logic       chip_fb_writedata_prev;
148     logic       chip_fb_WE_prev;
149     logic [11:0]chip_memaddr1_prev;
150     logic       chip_memWE1_prev;
151     logic [7:0] chip_memwritedata1_prev;
152     logic       chip_sound_timer_write_enable_prev;
```

```
153     logic       chip_delay_timer_write_enable_prev;
154
155     initial begin
156         pc <= 12'h200;
157
158         last_stage <= 32'h0;
159         memWE1 <= 1'b0;
160         memWE2 <= 1'b0;
161         cpu_instruction <= 16'h0;
162         delay_timer_write_enable <= 1'b0;
163         sound_timer_write_enable <= 1'b0;
164         I <= 16'h0;
165         regWE1 <= 1'b0;
166         regWE2 <= 1'b0;
167
168         fbreset <= 1'b0;
169         fb_addr_y <= 5'b0;
170         fb_addr_x <= 6'b0;
171         fb_writedata <= 1'b0;
172         fb_WE <= 1'b0;
173
174         stack_op <= STACK_HOLD;
175
176         state <= Chip8_PAUSED;
177         cpu_instruction <= 16'h0;
178         stage <= 32'h0;
179
180         stack_reset <= 1'b1;
181
182         fbvx_prev <= 6'h0;
183         fbvy_prev <= 5'h0;
184         mem_addr_prev <= 12'h0;
185
186         sound_on <= 1'b0;
187         chipselect_happened <= 1'b0;
188
189         fb_paused <= 1'b1;
190
```

```
191         halt_for_keypress <= 1'b0;
192     end
193 
194     always_ff @(posedge clk) begin
195         if(reset) begin
196             //Add initial values for code
197             pc <= 12'h200;
198 
199             memWE1 <= 1'b0;
200             memWE2 <= 1'b0;
201             cpu_instruction <= 16'h0;
202             delay_timer_write_enable <= 1'b0;
203             sound_timer_write_enable <= 1'b0;
204             I <= 16'h0;
205             regWE1 <= 1'b0;
206             regWE2 <= 1'b0;
207 
208             fbreset <= 1'b0;
209             fb_addr_y <= 5'b0;
210             fb_addr_x <= 6'b0;
211             fb_writedata <= 1'b0;
212             fb_WE <= 1'b0;
213 
214             stack_op <= STACK_HOLD;
215 
216             state <= Chip8_PAUSED;
217             cpu_instruction <= 16'h0;
218             stage <= 32'h0;
219 
220             stack_reset <= 1'b1;
221 
222             fbvx_prev <= 6'h0;
223             fbvy_prev <= 5'h0;
224             mem_addr_prev <= 12'h0;
225 
226             sound_on <= 1'b0;
227             chipselect_happened <= 1'b0;
228 
```

```
229             fb_paused <= 1'b1;
230 
231             halt_for_keypress <= 1'b0;
232 
233         //Handle input from the ARM processor
234     end else if(chipselect) begin
235 
236         chipselect_happened <= 1'b1;
237 
238         if(~chipselect_happened) begin
239             chip_sound_timer_write_enable_prev <=
                ↪  sound_timer_write_enable;
240             chip_delay_timer_write_enable_prev <=
                ↪  delay_timer_write_enable;
241             chip_reg_addr1_prev <= reg_addr1;
242             chip_regWE1_prev <= regWE1;
243             chip_reg_writedata1_prev <= reg_writedata1;
244             chip_fb_addr_x_prev <= fb_addr_x;
245             chip_fb_addr_y_prev <= fb_addr_y;
246             chip_fb_writedata_prev <= fb_writedata;
247             chip_fb_WE_prev <= fb_WE;
248             chip_memaddr1_prev <= memaddr1;
249             chip_memWE1_prev <= memWE1;
250             chip_memwritedata1_prev <= memwritedata1;
251         end
252 
253         casex (address)
254 
255                 //Read/write from register
256                 18'b00_0000_0000_0000_xxxx: begin
257                     reg_addr1 <= address[3:0];
258                     data_out <= {24'h0, reg_readdata1};
259                     if(write) begin
260                         regWE1 <= 1'b1;
261                         reg_writedata1 <= writedata[7:0];
262                     end
263                 end
264 
```

```
265                 18'h10 : begin
266                     if(write)
267                         I <= writedata[15:0];
268                     data_out <= {16'b0, I};
269                 end
270 
271                 //Read/write to sound_timer
272                 18'h11 : begin
273                     data_out <= {24'h0,
                     ↪  sound_timer_output_data};
274                     if(write) begin
275                         sound_timer_write_enable <= 1'b1;
276                         sound_timer_data <= writedata[7:0];
277                     end
278                 end
279 
280                 //Read/write to delay_timer
281                 18'h12 : begin
282                     data_out <= {24'h0,
                     ↪  delay_timer_output_data};
283                     if(write) begin
284                      delay_timer_write_enable <= 1'b1;
285                      delay_timer_data <= writedata[7:0];
286                  end
287             end
288 
289                 //Reset stack
290                 18'h13 : begin
291                     if(write) stack_reset <= 1'b1;
292                     data_out <= 32'h13;
293                 end
294 
295                 //Read/write to program counter
296                 18'h14 : begin
297                     data_out <= {4'h0, cpu_instruction, pc};
298                     if(write) pc <= writedata[11:0]; //0-4095
299                 end
300 
```

```
301                 //Read/write key presses
302                 18'h15 : begin
303                     data_out <= {27'b0, ispressed, key};
304                     if(write) begin
305                         ispressed <= writedata[4];
306                         key <= writedata[3:0];
307                     end
308                 end
309
310                 //Read/write the state of the emulator
311                 18'h16 : begin
312                     data_out <= state;
313                     if(write) begin
314                         case (writedata[1:0])
315                             2'h0: state <= Chip8_RUNNING;
316                             2'h1: state <=
                                ↪  Chip8_RUN_INSTRUCTION;
317                             2'h2: state <= Chip8_PAUSED;
318                             default : state <= Chip8_PAUSED;
319                         endcase
320                     end
321                 end
322
323                 //Modify framebuffer
324                 18'h17 : begin
325                     if(write) begin
326                         fbvx_prev <= writedata[10:5];
327                         fbvy_prev <= writedata[4:0];
328
329                         fb_addr_x <= writedata[10:5];
330                         fb_addr_y <= writedata[4:0];
331                         fb_writedata <= writedata[11];
332                         fb_WE <= writedata[12];
333                     end else begin
334                         data_out <= {31'h0, fb_readdata};
335                         fb_addr_x <= fbvx_prev;
336                         fb_addr_y <= fbvy_prev;
337                         fb_WE <= 1'b0;
```

```
338                     end
339                 end
340
341                 //Read/write stack
342                 18'h18 : begin
343                     $display("READ/WRITE STACK NOT
                        ↪  IMPLEMENTED");
344                     data_out <= 32'h18;
345                 end
346
347                 //MODIFY MEMORY
348                 18'h19 : begin
349                     if(write) begin
350                         memaddr1 <= writedata[19:8];
351                         memWE1 <= writedata[20] & write;
352                         memwritedata1 <= writedata[7:0];
353
354                         mem_addr_prev <= writedata[19:8];
355                     end else begin
356                         data_out <= {12'h0, mem_addr_prev,
                            ↪  memreaddata1};
357                         memWE1 <= 1'b0;
358                         memaddr1 <= mem_addr_prev;
359                     end
360                 end
361
362                 //Load single instruction
363                 18'h1A : begin
364                     if(write)
365                         cpu_instruction <= writedata[15:0];
366                     data_out <= {stage[15:0],
                        ↪  cpu_instruction};
367                     stage <= 32'h0;
368                 end
369
370                 18'h1B : begin
371                                 //Add initial values for code
372                     pc <= 12'h200;
```

```
373
374                     memWE1 <= 1'b0;
375                     memWE2 <= 1'b0;
376                     cpu_instruction <= 16'h0;
377                     delay_timer_write_enable <= 1'b0;
378                     sound_timer_write_enable <= 1'b0;
379                     I <= 16'h0;
380                     regWE1 <= 1'b0;
381                     regWE2 <= 1'b0;
382
383                     fbreset <= 1'b0;
384                     fb_addr_y <= 5'b0;
385                     fb_addr_x <= 6'b0;
386                     fb_writedata <= 1'b0;
387                     fb_WE <= 1'b0;
388
389                     stack_op <= STACK_HOLD;
390
391                     state <= Chip8_PAUSED;
392                     cpu_instruction <= 16'h0;
393                     stage <= 32'h0;
394
395                     stack_reset <= 1'b1;
396
397                     fbvx_prev <= 6'h0;
398                     fbvy_prev <= 5'h0;
399                     mem_addr_prev <= 12'h0;
400
401                     sound_on <= 1'b0;
402                     chipselect_happened <= 1'b0;
403
404                     fb_paused <= 1'b1;
405
406                     halt_for_keypress <= 1'b0;
407                 end
408
409                 default: begin
410                    data_out <= 32'd101;
```

```
411                 end
412
413             endcase
414
415         end else if(chipselect_happened) begin
416             // sound_timer_write_enable <=
                ↪  chip_sound_timer_write_enable_prev;
417             // delay_timer_write_enable <=
                ↪  chip_delay_timer_write_enable_prev;
418     //          reg_addr1 <= chip_reg_addr1_prev;
419     //          regWE1 <= chip_regWE1_prev;
420     //          reg_writedata1 <= chip_reg_writedata1_prev;
421     //          fb_addr_x <= chip_fb_addr_x_prev;
422     //          fb_addr_y <= chip_fb_addr_y_prev;
423     //          fb_writedata <= chip_fb_writedata_prev;
424     //          fb_WE <= chip_fb_WE_prev;
425     //          memaddr1 <= chip_memaddr1_prev;
426     //          memWE1 <= chip_memWE1_prev;
427     //          memwritedata1 <= chip_memwritedata1_prev;
428
429             chipselect_happened <= 1'b0;
430             stack_reset <= 1'b0;
431         end else begin
432             fb_paused <= state == Chip8_PAUSED;
433
434             case (state)
435                 Chip8_RUNNING: begin
436                     sound_on <= sound_timer_out;
437
438                     if(halt_for_keypress) begin
439                         if(ispressed) begin
440                             halt_for_keypress <= 1'b0;
441                         end
442                     end else if(stage == 32'h0) begin
443                         memaddr1 <= pc;
444                         memaddr2 <= pc + 12'h1;
445                         cpu_instruction <= 16'h0;
446
```

```
447                     bit_ovewritten <= 1'b0;
448                     is_drawing <= 1'b0;
449
450                     delay_timer_write_enable <= 1'b0;
451                     sound_timer_write_enable <= 1'b0;
452                     regWE1 <= 1'b0;
453                     regWE2 <= 1'b0;
454                     memWE1 <= 1'b0;
455                     memWE2 <= 1'b0;
456                     stack_op <= STACK_HOLD;
457                 end else if (stage == 32'h1) begin
458                     memaddr1 <= pc;
459                     memaddr2 <= pc + 12'h1;
460                     cpu_instruction <= {memreaddata1,
                        ↪ memreaddata2};
461                     last_stage <= stage;
462                 end else if (stage >= 32'h2) begin
463                     last_stage <= stage;
464                     if(cpu_delay_timer_WE) begin
465                         delay_timer_write_enable <= 1'b1;
466                         delay_timer_data <=
                            ↪ cpu_delay_timer_writedata;
467                     end else begin
468                         delay_timer_write_enable <= 1'b0;
469                     end
470
471                     if(cpu_sound_timer_WE) begin
472                         sound_timer_write_enable <= 1'b1;
473                         sound_timer_data <=
                            ↪ cpu_sound_timer_writedata;
474                     end else begin
475                         sound_timer_write_enable <= 1'b0;
476                     end
477
478                     if(cpu_reg_WE1) begin
479                         regWE1 <= 1'b1;
480                         reg_writedata1 <=
                            ↪ cpu_reg_writedata1;
```

```
481                     end else begin
482                         regWE1 <= 1'b0;
483                     end
484
485                     if(cpu_is_drawing) begin
486                         is_drawing <= 1'b1;
487                     end
488
489                     if(cpu_bit_overwritten) begin
490                         bit_ovewritten <= 1'b1;
491                     end
492
493                     if(stage == 32'd30000 && is_drawing)
                    ↪   begin
494                         regWE2 <= 1'b1;
495                         reg_writedata2 <= {7'h0,
                        ↪   bit_ovewritten};
496                         reg_addr2 <= 4'hF; //Setting VF
                        ↪   register to write
497                     end else if(cpu_reg_WE2) begin
498                         regWE2 <= 1'b1;
499                         reg_writedata2 <=
                        ↪   cpu_reg_writedata2;
500                         reg_addr2 <= cpu_reg_addr2;
501                     end else begin
502                         regWE2 <= 1'b0;
503                         reg_addr2 <= cpu_reg_addr2;
504                     end
505
506                     if(cpu_reg_I_WE) begin
507                         I <= cpu_reg_I_writedata;
508                     end
509
510                     if(cpu_stk_op == STACK_PUSH) begin
511                         stack_op <= STACK_PUSH;
512                         stack_writedata <=
                        ↪   cpu_stk_writedata;
513                     end
```

```
514
515                         //next_pc final modification on
                            ↪ NEXT_PC_WRITE_STAGE
516                         if((stage >= NEXT_PC_WRITE_STAGE -
                            ↪ 32'h3) & (stage <=
                            ↪ NEXT_PC_WRITE_STAGE)) begin
517                             if(cpu_stk_op == STACK_POP) begin
518                                 stack_op <= STACK_POP;
519                                 next_pc <=
                                    ↪ stack_outdata[11:0];
520                             end else begin
521                                 case (cpu_pc_src)
522                                     PC_SRC_ALU  : next_pc <=
                                        ↪ cpu_PC_writedata;
523                                     PC_SRC_SKIP : next_pc <=
                                        ↪ pc + 12'd4;
524                                     PC_SRC_NEXT : next_pc <=
                                        ↪ pc + 12'd2;
525                                     default : next_pc <= pc
                                        ↪ /*default next_pc <=
                                        ↪ pc + 12'd2*/;
526                                 endcase
527                             end
528                         end
529
530                         if(cpu_fb_WE) begin
531                             fb_writedata <= cpu_fb_writedata;
532                             fb_WE <= cpu_fb_WE;
533                         end else begin
534                             fb_WE <= 1'b0;
535                         end
536
537                         if(cpu_halt_for_keypress & !ispressed)
                            ↪ begin
538                             halt_for_keypress <= 1'b1;
539                         end
540
541                         if(cpu_mem_request) begin
```

```
542                     memaddr1 <= cpu_mem_addr1;
543                     memaddr2 <= cpu_mem_addr2;
544                     memwritedata1 <=
                         ↪  cpu_mem_writedata1;
545                     memwritedata2 <=
                         ↪  cpu_mem_writedata2;
546                     memWE1 <= cpu_mem_WE1;
547                     memWE2 <= cpu_mem_WE2;
548                 end
549
550                 //Always
551                 reg_addr1 <= cpu_reg_addr1;
552                 fb_addr_x <= cpu_fb_addr_x;
553                 fb_addr_y <= cpu_fb_addr_y;
554                 // memaddr1 <= cpu_mem_addr1;
555                 // memaddr2 <= cpu_mem_addr2;
556             end
557
558             //Cap of 50000, since 1000
                 ↪  instructions/sec is reasonable
559             if(!halt_for_keypress) begin
560                 if(stage >= CPU_CYCLE_LENGTH) begin
561                     stage <= 32'h0;
562                     pc <= next_pc;
563                 end
564                 else if (stage == 32'h1) begin
565                     if(stage == last_stage) stage <=
                         ↪  32'h2;
566                     else stage <= 32'h1;
567                 end else if(stage == 32'h2) begin
568                     if(stage == last_stage) stage <=
                         ↪  32'h3;
569                     else stage <= 32'h2;
570                 end
571
572                 else begin
573                     stage <= stage + 32'h1;
574                 end
```

```
575                     end
576                 end
577                 Chip8_RUN_INSTRUCTION: begin
578                     // sound_on <= 1'b1;
579                 end
580                 Chip8_PAUSED: begin
581                     // sound_on <= 1'b1;
582                 end
583                 default : /* default */;
584             endcase
585         end
586     end
587 
588     Chip8_framebuffer framebuffer(
589         .clk(clk),
590         .reset(fbreset),
591         .fb_addr_y(fb_addr_y),
592         .fb_addr_x(fb_addr_x),
593         .fb_writedata(fb_writedata),
594         .fb_WE(fb_WE),
595         .fb_readdata(fb_readdata),
596         .is_paused   (fb_paused),
597         .VGA_R(VGA_R),
598         .VGA_G(VGA_G),
599         .VGA_B(VGA_B),
600         .VGA_CLK(VGA_CLK),
601         .VGA_HS(VGA_HS),
602         .VGA_VS(VGA_VS),
603         .VGA_BLANK_n(VGA_BLANK_n),
604         .VGA_SYNC_n(VGA_SYNC_n)
605         );
606 
607     memory memory(
608         .address_a(memaddr1),
609         .address_b(memaddr2),
610         .clock(clk),
611         .data_a(memwritedata1),
612         .data_b(memwritedata2),
```

```
613         .wren_a(memWE1),
614         .wren_b(memWE2),
615         .q_a(memreaddata1),
616         .q_b(memreaddata2)
617         );
618 
619     reg_file reg_file(
620         .clock(clk),
621         .address_a(reg_addr1),
622         .address_b(reg_addr2),
623         .data_a(reg_writedata1),
624         .data_b(reg_writedata2),
625         .wren_a(regWE1),
626         .wren_b(regWE2),
627         .q_a(reg_readdata1),
628         .q_b(reg_readdata2)
629         );
630 
631     Chip8_CPU cpu(
632         .cpu_clk(clk),
633         .instruction(cpu_instruction),
634         .reg_readdata1(reg_readdata1),
635         .reg_readdata2(reg_readdata2),
636         .mem_readdata1(memreaddata1),
637         .mem_readdata2(memreaddata2),
638         .reg_I_readdata(I),
639         .delay_timer_readdata(delay_timer_output_data),
640         .key_pressed(ispressed),
641         .key_press(key),
642         .PC_readdata(pc),
643         .stage(stage),
644         .top_level_state(state),
645         .delay_timer_WE(cpu_delay_timer_WE),
646         .sound_timer_WE(cpu_sound_timer_WE),
647         .delay_timer_writedata(cpu_delay_timer_writedata),
648         .sound_timer_writedata(cpu_sound_timer_writedata),
649         .pc_src(cpu_pc_src),
650         .PC_writedata(cpu_PC_writedata),
```

```
651         .reg_WE1(cpu_reg_WE1),
652         .reg_WE2(cpu_reg_WE2),
653         .reg_addr1(cpu_reg_addr1),
654         .reg_addr2(cpu_reg_addr2),
655         .reg_writedata1(cpu_reg_writedata1),
656         .reg_writedata2(cpu_reg_writedata2),
657         .mem_WE1(cpu_mem_WE1),
658         .mem_WE2(cpu_mem_WE2),
659         .mem_addr1(cpu_mem_addr1),
660         .mem_addr2(cpu_mem_addr2),
661         .mem_request      (cpu_mem_request),
662         .mem_writedata1(cpu_mem_writedata1),
663         .mem_writedata2(cpu_mem_writedata2),
664         .reg_I_WE(cpu_reg_I_WE),
665         .reg_I_writedata(cpu_reg_I_writedata),
666         .fbreset(cpu_fbreset),
667 
668         .fb_addr_y(cpu_fb_addr_y),
669         .fb_addr_x(cpu_fb_addr_x),
670         .fb_writedata(cpu_fb_writedata),
671         .fb_WE(cpu_fb_WE),
672         .fb_readdata(fb_readdata),
673 
674         .bit_overwritten(cpu_bit_overwritten),
675         .isDrawing(cpu_is_drawing),
676 
677         .stk_reset(cpu_stk_reset),
678         .stk_op(cpu_stk_op),
679         .stk_writedata(cpu_stk_writedata),
680 
681         .halt_for_keypress(cpu_halt_for_keypress)
682         );
683 
684     Chip8_SoundController sound(
685         .OSC_50_B8A(OSC_50_B8A),
686         .AUD_ADCLRCK(AUD_ADCLRCK),
687         .AUD_ADCDAT(AUD_ADCDAT),
688         .AUD_DACLRCK(AUD_DACLRCK),
```

```
689         .AUD_DACDAT(AUD_DACDAT),
690         .AUD_XCK(AUD_XCK),
691         .AUD_BCLK(AUD_BCLK),
692         .AUD_I2C_SCLK(AUD_I2C_SCLK),
693         .AUD_I2C_SDAT(AUD_I2C_SDAT),
694         .AUD_MUTE(AUD_MUTE),
695 
696         .clk(clk),
697         .is_on(sound_on),
698         .reset(sound_reset)
699         );
700 
701     clk_div clk_div(
702         .clk_in(clk),
703         .reset(clk_div_reset),
704         .clk_out(clk_div_clk_out)
705         );
706 
707     timer delay_timer(
708         .clk(clk),
709         .clk_60(clk_div_clk_out),
710         .write_enable(delay_timer_write_enable),
711         .data(delay_timer_data),
712         .out(delay_timer_out),
713         .output_data (delay_timer_output_data)
714         );
715 
716     timer sound_timer(
717         .clk(clk),
718         .clk_60(clk_div_clk_out),
719         .write_enable(sound_timer_write_enable),
720         .data(sound_timer_data),
721         .out(sound_timer_out),
722         .output_data (sound_timer_output_data)
723         );
724 
725     Chip8_Stack stack (
726         .reset(stack_reset),
```

```
727         .cpu_clk(clk),
728         .op(stack_op),
729         .writedata(stack_writedata),
730         .outdata(stack_outdata)
731         );
732
733
734 endmodule
```

7.1.16 SoCKit_top.sv

```
1  //
    ↪  ====================================================================
2  // Copyright (c) 2013 by Terasic Technologies Inc.
3  //
    ↪  ====================================================================
4  //
5  // Permission:
6  //
7  //   Terasic grants permission to use and modify this code
    ↪  for use
8  //   in synthesis for all Terasic Development Boards and
    ↪  Altera Development
9  //   Kits made by Terasic.  Other use of this code,
    ↪  including the selling
10 //   ,duplication, or modification of any portion is
    ↪  strictly prohibited.
11 //
12 // Disclaimer:
13 //
14 //   This VHDL/Verilog or C/C++ source code is intended as a
    ↪  design reference
15 //   which illustrates how these types of functions can be
    ↪  implemented.
16 //   It is the user's responsibility to verify their design
    ↪  for
17 //   consistency and functionality through the use of formal
```

```
18 //   verification methods.  Terasic provides no warranty
   ↪  regarding the use
19 //   or functionality of this code.
20 //
21 //
   ↪  =============================================================================
22 //
23 //  Terasic Technologies Inc
24 //  9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu
   ↪  City, 30070. Taiwan
25 //
26 //
27 //                     web: http://www.terasic.com/
28 //                     email: support@terasic.com
29 //
30 //
   ↪  =============================================================================
31 //
   ↪  =============================================================================
32 //
33 // Major Functions:    SoCKit_Default
34 //
35 //
   ↪  =============================================================================
36 // Revision History :
37 //
   ↪  =============================================================================
38 //   Ver  :| Author              :| Mod. Date :| Changes
   ↪  Made:
39 //   V1.0 :| xinxian            :| 04/02/13  :| Initial
   ↪  Revision
40 //
   ↪  =============================================================================
41
42 //`define ENABLE_DDR3
43 //`define ENABLE_HPS
44 //`define ENABLE_HSMC_XCVR
45
```

```
46 module SoCKit_top(
47
48           //////////AUD//////////
49           AUD_ADCDAT,
50           AUD_ADCLRCK,
51           AUD_BCLK,
52           AUD_DACDAT,
53           AUD_DACLRCK,
54           AUD_I2C_SCLK,
55           AUD_I2C_SDAT,
56           AUD_MUTE,
57           AUD_XCK,
58
59 `ifdef ENABLE_DDR3
60           ////////DDR3////////
61           DDR3_A,
62           DDR3_BA,
63           DDR3_CAS_n,
64           DDR3_CKE,
65           DDR3_CK_n,
66           DDR3_CK_p,
67           DDR3_CS_n,
68           DDR3_DM,
69           DDR3_DQ,
70           DDR3_DQS_n,
71           DDR3_DQS_p,
72           DDR3_ODT,
73           DDR3_RAS_n,
74           DDR3_RESET_n,
75           DDR3_RZQ,
76           DDR3_WE_n,
77 `endif /*ENABLE_DDR3*/
78
79           ////////FAN////////
80           FAN_CTRL,
81
82 `ifdef ENABLE_HPS
83           ////////HPS////////
```

```
84              HPS_CLOCK_25,
85              HPS_CLOCK_50,
86              HPS_CONV_USB_n,
87              HPS_DDR3_A,
88              HPS_DDR3_BA,
89              HPS_DDR3_CAS_n,
90              HPS_DDR3_CKE,
91              HPS_DDR3_CK_n,
92              HPS_DDR3_CK_p,
93              HPS_DDR3_CS_n,
94              HPS_DDR3_DM,
95              HPS_DDR3_DQ,
96              HPS_DDR3_DQS_n,
97              HPS_DDR3_DQS_p,
98              HPS_DDR3_ODT,
99              HPS_DDR3_RAS_n,
100             HPS_DDR3_RESET_n,
101             HPS_DDR3_RZQ,
102             HPS_DDR3_WE_n,
103             HPS_ENET_GTX_CLK,
104             HPS_ENET_INT_n,
105             HPS_ENET_MDC,
106             HPS_ENET_MDIO,
107             HPS_ENET_RESET_n,
108             HPS_ENET_RX_CLK,
109             HPS_ENET_RX_DATA,
110             HPS_ENET_RX_DV,
111             HPS_ENET_TX_DATA,
112             HPS_ENET_TX_EN,
113             HPS_FLASH_DATA,
114             HPS_FLASH_DCLK,
115             HPS_FLASH_NCSO,
116             HPS_GSENSOR_INT,
117             HPS_I2C_CLK,
118             HPS_I2C_SDA,
119             HPS_KEY,
120             HPS_LCM_D_C,
121             HPS_LCM_RST_N,
```

```
122             HPS_LCM_SPIM_CLK,
123             HPS_LCM_SPIM_MISO,
124             HPS_LCM_SPIM_MOSI,
125             HPS_LCM_SPIM_SS,
126             HPS_LED,
127             HPS_LTC_GPIO,
128             HPS_RESET_n,
129             HPS_SD_CLK,
130             HPS_SD_CMD,
131             HPS_SD_DATA,
132             HPS_SPIM_CLK,
133             HPS_SPIM_MISO,
134             HPS_SPIM_MOSI,
135             HPS_SPIM_SS,
136             HPS_SW,
137             HPS_UART_RX,
138             HPS_UART_TX,
139             HPS_USB_CLKOUT,
140             HPS_USB_DATA,
141             HPS_USB_DIR,
142             HPS_USB_NXT,
143             HPS_USB_RESET_PHY,
144             HPS_USB_STP,
145             HPS_WARM_RST_n,
146 `endif /*ENABLE_HPS*/
147
148             /////////HSMC/////////
149             HSMC_CLKIN_n,
150             HSMC_CLKIN_p,
151             HSMC_CLKOUT_n,
152             HSMC_CLKOUT_p,
153             HSMC_CLK_IN0,
154             HSMC_CLK_OUT0,
155             HSMC_D,
156
157 `ifdef ENABLE_HSMC_XCVR
158
159             HSMC_GXB_RX_p,
```

```
160             HSMC_GXB_TX_p,
161             HSMC_REF_CLK_p,
162 `endif
163             HSMC_RX_n,
164             HSMC_RX_p,
165             HSMC_SCL,
166             HSMC_SDA,
167             HSMC_TX_n,
168             HSMC_TX_p,
169
170             /////////IRDA/////////
171             IRDA_RXD,
172
173             /////////KEY/////////
174             KEY,
175
176             /////////LED/////////
177             LED,
178
179             /////////OSC/////////
180             OSC_50_B3B,
181             OSC_50_B4A,
182             OSC_50_B5B,
183             OSC_50_B8A,
184
185             /////////PCIE/////////
186             PCIE_PERST_n,
187             PCIE_WAKE_n,
188
189             /////////RESET/////////
190             RESET_n,
191
192             /////////SI5338/////////
193             SI5338_SCL,
194             SI5338_SDA,
195
196             /////////SW/////////
197             SW,
```

```
198
199             //////////TEMP//////////
200             TEMP_CS_n,
201             TEMP_DIN,
202             TEMP_DOUT,
203             TEMP_SCLK,
204
205             //////////USB//////////
206             USB_B2_CLK,
207             USB_B2_DATA,
208             USB_EMPTY,
209             USB_FULL,
210             USB_OE_n,
211             USB_RD_n,
212             USB_RESET_n,
213             USB_SCL,
214             USB_SDA,
215             USB_WR_n,
216
217             //////////VGA//////////
218             VGA_B,
219             VGA_BLANK_n,
220             VGA_CLK,
221             VGA_G,
222             VGA_HS,
223             VGA_R,
224             VGA_SYNC_n,
225             VGA_VS,
226             ///////////hps//////////
227             memory_mem_a,
228             memory_mem_ba,
229             memory_mem_ck,
230             memory_mem_ck_n,
231             memory_mem_cke,
232             memory_mem_cs_n,
233             memory_mem_ras_n,
234             memory_mem_cas_n,
235             memory_mem_we_n,
```

```
236             memory_mem_reset_n,
237             memory_mem_dq,
238             memory_mem_dqs,
239             memory_mem_dqs_n,
240             memory_mem_odt,
241             memory_mem_dm,
242             memory_oct_rzqin,
243             hps_io_hps_io_emac1_inst_TX_CLK,
244             hps_io_hps_io_emac1_inst_TXD0,
245             hps_io_hps_io_emac1_inst_TXD1,
246             hps_io_hps_io_emac1_inst_TXD2,
247             hps_io_hps_io_emac1_inst_TXD3,
248             hps_io_hps_io_emac1_inst_RXD0,
249             hps_io_hps_io_emac1_inst_MDIO,
250             hps_io_hps_io_emac1_inst_MDC,
251             hps_io_hps_io_emac1_inst_RX_CTL,
252             hps_io_hps_io_emac1_inst_TX_CTL,
253             hps_io_hps_io_emac1_inst_RX_CLK,
254             hps_io_hps_io_emac1_inst_RXD1,
255             hps_io_hps_io_emac1_inst_RXD2,
256             hps_io_hps_io_emac1_inst_RXD3,
257             hps_io_hps_io_qspi_inst_IO0,
258             hps_io_hps_io_qspi_inst_IO1,
259             hps_io_hps_io_qspi_inst_IO2,
260             hps_io_hps_io_qspi_inst_IO3,
261             hps_io_hps_io_qspi_inst_SS0,
262             hps_io_hps_io_qspi_inst_CLK,
263             hps_io_hps_io_sdio_inst_CMD,
264             hps_io_hps_io_sdio_inst_D0,
265             hps_io_hps_io_sdio_inst_D1,
266             hps_io_hps_io_sdio_inst_CLK,
267             hps_io_hps_io_sdio_inst_D2,
268             hps_io_hps_io_sdio_inst_D3,
269             hps_io_hps_io_usb1_inst_D0,
270             hps_io_hps_io_usb1_inst_D1,
271             hps_io_hps_io_usb1_inst_D2,
272             hps_io_hps_io_usb1_inst_D3,
273             hps_io_hps_io_usb1_inst_D4,
```

```
274             hps_io_hps_io_usb1_inst_D5,
275             hps_io_hps_io_usb1_inst_D6,
276             hps_io_hps_io_usb1_inst_D7,
277             hps_io_hps_io_usb1_inst_CLK,
278             hps_io_hps_io_usb1_inst_STP,
279             hps_io_hps_io_usb1_inst_DIR,
280             hps_io_hps_io_usb1_inst_NXT,
281             hps_io_hps_io_spim0_inst_CLK,
282             hps_io_hps_io_spim0_inst_MOSI,
283             hps_io_hps_io_spim0_inst_MISO,
284             hps_io_hps_io_spim0_inst_SS0,
285             hps_io_hps_io_spim1_inst_CLK,
286             hps_io_hps_io_spim1_inst_MOSI,
287             hps_io_hps_io_spim1_inst_MISO,
288             hps_io_hps_io_spim1_inst_SS0,
289             hps_io_hps_io_uart0_inst_RX,
290             hps_io_hps_io_uart0_inst_TX,
291             hps_io_hps_io_i2c1_inst_SDA,
292             hps_io_hps_io_i2c1_inst_SCL,
293             hps_io_hps_io_gpio_inst_GPIO00
294             );
295
296     //=======================================================
297     //  PORT declarations
298     //=======================================================
299
300     ///////// AUD /////////
301     input
        ↪  AUD_ADCDAT;
302     inout
        ↪  AUD_ADCLRCK;
303     inout
        ↪  AUD_BCLK;
304     output
        ↪  AUD_DACDAT;
305     inout
        ↪  AUD_DACLRCK;
```

```
306     output
        ↪  AUD_I2C_SCLK;
307     inout
        ↪  AUD_I2C_SDAT;
308     output
        ↪  AUD_MUTE;
309     output                                                  AUD_XCK;
310
311 `ifdef ENABLE_DDR3
312     ///////// DDR3 /////////
313     output [14:0]                          DDR3_A;
314     output [2:0]                          DDR3_BA;
315     output
        ↪  DDR3_CAS_n;
316     output
        ↪  DDR3_CKE;
317     output
        ↪  DDR3_CK_n;
318     output
        ↪  DDR3_CK_p;
319     output
        ↪  DDR3_CS_n;
320     output [3:0]                          DDR3_DM;
321     inout [31:0]                          DDR3_DQ;
322     inout [3:0]                              DDR3_DQS_n;
323     inout [3:0]                              DDR3_DQS_p;
324     output
        ↪  DDR3_ODT;
325     output
        ↪  DDR3_RAS_n;
326     output
        ↪  DDR3_RESET_n;
327     input
        ↪  DDR3_RZQ;
328     output
        ↪  DDR3_WE_n;
329 `endif /*ENABLE_DDR3*/
330
```

```
331     ///////// FAN /////////
332     output
        ↪  FAN_CTRL;
333
334 `ifdef ENABLE_HPS
335     ///////// HPS /////////
336     input
        ↪  HPS_CLOCK_25;
337     input
        ↪  HPS_CLOCK_50;
338     input
        ↪  HPS_CONV_USB_n;
339     output [14:0]                    HPS_DDR3_A;
340     output [2:0]                     HPS_DDR3_BA;
341     output
        ↪  HPS_DDR3_CAS_n;
342     output
        ↪  HPS_DDR3_CKE;
343     output
        ↪  HPS_DDR3_CK_n;
344     output
        ↪  HPS_DDR3_CK_p;
345     output
        ↪  HPS_DDR3_CS_n;
346     output [3:0]                     HPS_DDR3_DM;
347     inout [31:0]                     HPS_DDR3_DQ;
348     inout [3:0]                         HPS_DDR3_DQS_n;
349     inout [3:0]                         HPS_DDR3_DQS_p;
350     output
        ↪  HPS_DDR3_ODT;
351     output
        ↪  HPS_DDR3_RAS_n;
352     output
        ↪  HPS_DDR3_RESET_n;
353     input
        ↪  HPS_DDR3_RZQ;
354     output
        ↪  HPS_DDR3_WE_n;
```

```
355     input
        ↪  HPS_ENET_GTX_CLK;
356     input
        ↪  HPS_ENET_INT_n;
357     output
        ↪  HPS_ENET_MDC;
358     inout
        ↪  HPS_ENET_MDIO;
359     output
        ↪  HPS_ENET_RESET_n;
360     input
        ↪  HPS_ENET_RX_CLK;
361     input [3:0]                    HPS_ENET_RX_DATA;
362     input
        ↪  HPS_ENET_RX_DV;
363     output [3:0]                   HPS_ENET_TX_DATA;
364     output
        ↪  HPS_ENET_TX_EN;
365     inout [3:0]                    HPS_FLASH_DATA;
366     output
        ↪  HPS_FLASH_DCLK;
367     output
        ↪  HPS_FLASH_NCSO;
368     input
        ↪  HPS_GSENSOR_INT;
369     inout
        ↪  HPS_I2C_CLK;
370     inout
        ↪  HPS_I2C_SDA;
371     inout [3:0]                    HPS_KEY;
372     output
        ↪  HPS_LCM_D_C;
373     output
        ↪  HPS_LCM_RST_N;
374     input
        ↪  HPS_LCM_SPIM_CLK;
375     inout
        ↪  HPS_LCM_SPIM_MISO;
```

```
376     output
        ↪  HPS_LCM_SPIM_MOSI;
377     output
        ↪  HPS_LCM_SPIM_SS;
378     output [3:0]                        HPS_LED;
379     inout
        ↪  HPS_LTC_GPIO;
380     input
        ↪  HPS_RESET_n;
381     output
        ↪  HPS_SD_CLK;
382     inout
        ↪  HPS_SD_CMD;
383     inout [3:0]                           HPS_SD_DATA;
384     output
        ↪  HPS_SPIM_CLK;
385     input
        ↪  HPS_SPIM_MISO;
386     output
        ↪  HPS_SPIM_MOSI;
387     output
        ↪  HPS_SPIM_SS;
388     input [3:0]                           HPS_SW;
389     input
        ↪  HPS_UART_RX;
390     output
        ↪  HPS_UART_TX;
391     input
        ↪  HPS_USB_CLKOUT;
392     inout [7:0]                           HPS_USB_DATA;
393     input
        ↪  HPS_USB_DIR;
394     input
        ↪  HPS_USB_NXT;
395     output
        ↪  HPS_USB_RESET_PHY;
396     output
        ↪  HPS_USB_STP;
```

```
397     input
        ↪  HPS_WARM_RST_n;
398 `endif /*ENABLE_HPS*/
399
400     ///////// HSMC /////////
401     input [2:1]                         HSMC_CLKIN_n;
402     input [2:1]                         HSMC_CLKIN_p;
403     output [2:1]                        HSMC_CLKOUT_n;
404     output [2:1]                        HSMC_CLKOUT_p;
405     input
        ↪  HSMC_CLK_IN0;
406     output
        ↪  HSMC_CLK_OUT0;
407     inout [3:0]                         HSMC_D;
408 `ifdef ENABLE_HSMC_XCVR
409     input [7:0]                         HSMC_GXB_RX_p;
410     output [7:0]                        HSMC_GXB_TX_p;
411     input
        ↪  HSMC_REF_CLK_p;
412 `endif
413     inout [16:0]                        HSMC_RX_n;
414     inout [16:0]                        HSMC_RX_p;
415     output
        ↪  HSMC_SCL;
416     inout
        ↪  HSMC_SDA;
417     inout [16:0]                        HSMC_TX_n;
418     inout [16:0]                        HSMC_TX_p;
419
420     ///////// IRDA /////////
421     input
        ↪  IRDA_RXD;
422
423     ///////// KEY /////////
424     input [3:0]                         KEY;
425
426     ///////// LED /////////
427     output [3:0]                        LED;
```

```
428
429     ///////// OSC /////////
430     input
        ↪  OSC_50_B3B;
431     input
        ↪  OSC_50_B4A;
432     input
        ↪  OSC_50_B5B;
433     input
        ↪  OSC_50_B8A;
434
435     ///////// PCIE /////////
436     input
        ↪  PCIE_PERST_n;
437     input
        ↪  PCIE_WAKE_n;
438
439     ///////// RESET /////////
440     input                                                   RESET_n;
441
442     ///////// SI5338 /////////
443     inout
        ↪  SI5338_SCL;
444     inout
        ↪  SI5338_SDA;
445
446     ///////// SW /////////
447     input [3:0]                             SW;
448
449     ///////// TEMP /////////
450     output
        ↪  TEMP_CS_n;
451     output
        ↪  TEMP_DIN;
452     input
        ↪  TEMP_DOUT;
453     output
        ↪  TEMP_SCLK;
```

```
454
455     ///////// USB /////////
456     input
        ↪ USB_B2_CLK;
457     inout [7:0]                     USB_B2_DATA;
458     output
        ↪ USB_EMPTY;
459     output
        ↪ USB_FULL;
460     input
        ↪ USB_OE_n;
461     input
        ↪ USB_RD_n;
462     input
        ↪ USB_RESET_n;
463     inout                                       USB_SCL;
464     inout                                       USB_SDA;
465     input
        ↪ USB_WR_n;
466
467     ///////// VGA /////////
468     output [7:0]                VGA_B;
469     output
        ↪ VGA_BLANK_n;
470     output                                      VGA_CLK;
471     output [7:0]                VGA_G;
472     output                                      VGA_HS;
473     output [7:0]                VGA_R;
474     output
        ↪ VGA_SYNC_n;
475     output                                      VGA_VS;
476
477     ////////hps pin///////
478     output wire [14:0]              memory_mem_a;
479     output wire [2:0]               memory_mem_ba;
480     output wire                     memory_mem_ck;
481     output wire                     memory_mem_ck_n;
482     output wire                     memory_mem_cke;
```

```
483     output wire                                 memory_mem_cs_n;
484     output wire                                 memory_mem_ras_n;
485     output wire                                 memory_mem_cas_n;
486     output wire                                 memory_mem_we_n;
487     output wire                                 memory_mem_reset_n;
488     inout  wire [31:0]                             memory_mem_dq;
489     inout  wire [3:0]                           memory_mem_dqs;
490     inout  wire [3:0]                           memory_mem_dqs_n;
491     output wire                                 memory_mem_odt;
492     output wire [3:0]                           memory_mem_dm;
493     input  wire                                 memory_oct_rzqin;
494     output wire
        ↪ hps_io_hps_io_emac1_inst_TX_CLK;
495     output wire
        ↪ hps_io_hps_io_emac1_inst_TXD0;
496     output wire
        ↪ hps_io_hps_io_emac1_inst_TXD1;
497     output wire
        ↪ hps_io_hps_io_emac1_inst_TXD2;
498     output wire
        ↪ hps_io_hps_io_emac1_inst_TXD3;
499     input  wire
        ↪ hps_io_hps_io_emac1_inst_RXD0;
500     inout  wire
        ↪ hps_io_hps_io_emac1_inst_MDIO;
501     output wire
        ↪ hps_io_hps_io_emac1_inst_MDC;
502     input  wire
        ↪ hps_io_hps_io_emac1_inst_RX_CTL;
503     output wire
        ↪ hps_io_hps_io_emac1_inst_TX_CTL;
504     input  wire
        ↪ hps_io_hps_io_emac1_inst_RX_CLK;
505     input  wire
        ↪ hps_io_hps_io_emac1_inst_RXD1;
506     input  wire
        ↪ hps_io_hps_io_emac1_inst_RXD2;
```

```
507     input  wire
        ↪  hps_io_hps_io_emac1_inst_RXD3;
508     inout  wire
        ↪  hps_io_hps_io_qspi_inst_IO0;
509     inout  wire
        ↪  hps_io_hps_io_qspi_inst_IO1;
510     inout  wire
        ↪  hps_io_hps_io_qspi_inst_IO2;
511     inout  wire
        ↪  hps_io_hps_io_qspi_inst_IO3;
512     output wire
        ↪  hps_io_hps_io_qspi_inst_SS0;
513     output wire
        ↪  hps_io_hps_io_qspi_inst_CLK;
514     inout  wire
        ↪  hps_io_hps_io_sdio_inst_CMD;
515     inout  wire
        ↪  hps_io_hps_io_sdio_inst_D0;
516     inout  wire
        ↪  hps_io_hps_io_sdio_inst_D1;
517     output wire
        ↪  hps_io_hps_io_sdio_inst_CLK;
518     inout  wire
        ↪  hps_io_hps_io_sdio_inst_D2;
519     inout  wire
        ↪  hps_io_hps_io_sdio_inst_D3;
520     inout  wire
        ↪  hps_io_hps_io_usb1_inst_D0;
521     inout  wire
        ↪  hps_io_hps_io_usb1_inst_D1;
522     inout  wire
        ↪  hps_io_hps_io_usb1_inst_D2;
523     inout  wire
        ↪  hps_io_hps_io_usb1_inst_D3;
524     inout  wire
        ↪  hps_io_hps_io_usb1_inst_D4;
525     inout  wire
        ↪  hps_io_hps_io_usb1_inst_D5;
```

```
526     inout  wire
        ↪  hps_io_hps_io_usb1_inst_D6;
527     inout  wire
        ↪  hps_io_hps_io_usb1_inst_D7;
528     input  wire
        ↪  hps_io_hps_io_usb1_inst_CLK;
529     output wire
        ↪  hps_io_hps_io_usb1_inst_STP;
530     input  wire
        ↪  hps_io_hps_io_usb1_inst_DIR;
531     input  wire
        ↪  hps_io_hps_io_usb1_inst_NXT;
532     output wire
        ↪  hps_io_hps_io_spim0_inst_CLK;
533     output wire
        ↪  hps_io_hps_io_spim0_inst_MOSI;
534     input  wire
        ↪  hps_io_hps_io_spim0_inst_MISO;
535     output wire
        ↪  hps_io_hps_io_spim0_inst_SS0;
536     output wire
        ↪  hps_io_hps_io_spim1_inst_CLK;
537     output wire
        ↪  hps_io_hps_io_spim1_inst_MOSI;
538     input  wire
        ↪  hps_io_hps_io_spim1_inst_MISO;
539     output wire
        ↪  hps_io_hps_io_spim1_inst_SS0;
540     input  wire
        ↪  hps_io_hps_io_uart0_inst_RX;
541     output wire
        ↪  hps_io_hps_io_uart0_inst_TX;
542     inout  wire
        ↪  hps_io_hps_io_i2c1_inst_SDA;
543     inout  wire
        ↪  hps_io_hps_io_i2c1_inst_SCL;
544     inout  wire
        ↪  hps_io_hps_io_gpio_inst_GPIO00;
```

```
545     //=======================================================
546     //  REG/WIRE declarations
547     //=======================================================
548
549     //    For Audio CODEC
550     wire                            AUD_CTRL_CLK;    //    For
        ↪  Audio Controller
551
552     reg [31:0]                      Cont;
553     wire                            VGA_CTRL_CLK;
554     wire [9:0]                      mVGA_R;
555     wire [9:0]                      mVGA_G;
556     wire [9:0]                      mVGA_B;
557     wire [19:0]                   mVGA_ADDR;
558     wire                            DLY_RST;
559
560     //    For VGA Controller
561     wire                            mVGA_CLK;
562     wire [9:0]                      mRed;
563     wire [9:0]                      mGreen;
564     wire [9:0]                      mBlue;
565     wire                            VGA_Read;    //    VGA data
        ↪  request
566
567     wire [9:0]                      recon_VGA_R;
568     wire [9:0]                      recon_VGA_G;
569     wire [9:0]                      recon_VGA_B;
570
571     //    For Down Sample
572     wire [3:0]                      Remain;
573     wire [9:0]                      Quotient;
574
575     wire                            AUD_MUTE;
576
577     // Drive the LEDs with the switches
578     assign LED = SW;
579
580     // Make the FPGA reset cause an HPS reset
```

```
581     reg [19:0]                              hps_reset_counter =
        ↪  20'h0;
582     reg                                   hps_fpga_reset_n = 0;
583
584     always @(posedge OSC_50_B4A) begin
585        if (hps_reset_counter == 20'h fffffff) hps_fpga_reset_n
           ↪  <= 1;
586        hps_reset_counter <= hps_reset_counter + 1;
587     end
588
589     Chip8 u0 (
590               .clk_clk
                  ↪  (OSC_50_B4A),                        //
                  ↪  clk.clk
591               .reset_reset_n
                  ↪  (hps_fpga_reset_n),                   //
                  ↪  reset.reset_n
592               .memory_mem_a
                  ↪  (memory_mem_a),                      //
                  ↪  memory.mem_a
593               .memory_mem_ba
                  ↪  (memory_mem_ba),                     //
                  ↪  .mem_ba
594               .memory_mem_ck
                  ↪  (memory_mem_ck),                     //
                  ↪  .mem_ck
595               .memory_mem_ck_n
                  ↪  (memory_mem_ck_n),                   //
                  ↪  .mem_ck_n
596               .memory_mem_cke
                  ↪  (memory_mem_cke),                    //
                  ↪  .mem_cke
597               .memory_mem_cs_n
                  ↪  (memory_mem_cs_n),                   //
                  ↪  .mem_cs_n
598               .memory_mem_ras_n
                  ↪  (memory_mem_ras_n),                  //
                  ↪  .mem_ras_n
```

```
599                 .memory_mem_cas_n
                    ↪  (memory_mem_cas_n),                  //
                    ↪  .mem_cas_n
600                 .memory_mem_we_n
                    ↪  (memory_mem_we_n),                   //
                    ↪  .mem_we_n
601                 .memory_mem_reset_n
                    ↪  (memory_mem_reset_n),                //
                    ↪  .mem_reset_n
602                 .memory_mem_dq
                    ↪  (memory_mem_dq),                     //
                    ↪  .mem_dq
603                 .memory_mem_dqs
                    ↪  (memory_mem_dqs),                    //
                    ↪  .mem_dqs
604                 .memory_mem_dqs_n
                    ↪  (memory_mem_dqs_n),                  //
                    ↪  .mem_dqs_n
605                 .memory_mem_odt
                    ↪  (memory_mem_odt),                    //
                    ↪  .mem_odt
606                 .memory_mem_dm
                    ↪  (memory_mem_dm),                     //
                    ↪  .mem_dm
607                 .memory_oct_rzqin
                    ↪  (memory_oct_rzqin),                  //
                    ↪  .oct_rzqin
608                 .hps_io_hps_io_emac1_inst_TX_CLK
                    ↪  (hps_io_hps_io_emac1_inst_TX_CLK), //
                    ↪
                    ↪  .hps_0_hps_io.hps_io_emac1_inst_TX_CLK
609                 .hps_io_hps_io_emac1_inst_TXD0
                    ↪  (hps_io_hps_io_emac1_inst_TXD0),     //
                    ↪  .hps_io_emac1_inst_TXD0
610                 .hps_io_hps_io_emac1_inst_TXD1
                    ↪  (hps_io_hps_io_emac1_inst_TXD1),     //
                    ↪  .hps_io_emac1_inst_TXD1
```

```
611             .hps_io_hps_io_emac1_inst_TXD2
                ↪  (hps_io_hps_io_emac1_inst_TXD2),   //
                ↪  .hps_io_emac1_inst_TXD2
612             .hps_io_hps_io_emac1_inst_TXD3
                ↪  (hps_io_hps_io_emac1_inst_TXD3),   //
                ↪  .hps_io_emac1_inst_TXD3
613             .hps_io_hps_io_emac1_inst_RXD0
                ↪  (hps_io_hps_io_emac1_inst_RXD0),   //
                ↪  .hps_io_emac1_inst_RXD0
614             .hps_io_hps_io_emac1_inst_MDIO
                ↪  (hps_io_hps_io_emac1_inst_MDIO),   //
                ↪  .hps_io_emac1_inst_MDIO
615             .hps_io_hps_io_emac1_inst_MDC
                ↪  (hps_io_hps_io_emac1_inst_MDC),    //
                ↪  .hps_io_emac1_inst_MDC
616             .hps_io_hps_io_emac1_inst_RX_CTL
                ↪  (hps_io_hps_io_emac1_inst_RX_CTL), //
                ↪  .hps_io_emac1_inst_RX_CTL
617             .hps_io_hps_io_emac1_inst_TX_CTL
                ↪  (hps_io_hps_io_emac1_inst_TX_CTL), //
                ↪  .hps_io_emac1_inst_TX_CTL
618             .hps_io_hps_io_emac1_inst_RX_CLK
                ↪  (hps_io_hps_io_emac1_inst_RX_CLK), //
                ↪  .hps_io_emac1_inst_RX_CLK
619             .hps_io_hps_io_emac1_inst_RXD1
                ↪  (hps_io_hps_io_emac1_inst_RXD1),   //
                ↪  .hps_io_emac1_inst_RXD1
620             .hps_io_hps_io_emac1_inst_RXD2
                ↪  (hps_io_hps_io_emac1_inst_RXD2),   //
                ↪  .hps_io_emac1_inst_RXD2
621             .hps_io_hps_io_emac1_inst_RXD3
                ↪  (hps_io_hps_io_emac1_inst_RXD3),   //
                ↪  .hps_io_emac1_inst_RXD3
622             .hps_io_hps_io_qspi_inst_IO0
                ↪  (hps_io_hps_io_qspi_inst_IO0),     //
                ↪  .hps_io_qspi_inst_IO0
```

```
623             .hps_io_hps_io_qspi_inst_IO1
                ↪  (hps_io_hps_io_qspi_inst_IO1),     //
                ↪  .hps_io_qspi_inst_IO1
624             .hps_io_hps_io_qspi_inst_IO2
                ↪  (hps_io_hps_io_qspi_inst_IO2),     //
                ↪  .hps_io_qspi_inst_IO2
625             .hps_io_hps_io_qspi_inst_IO3
                ↪  (hps_io_hps_io_qspi_inst_IO3),     //
                ↪  .hps_io_qspi_inst_IO3
626             .hps_io_hps_io_qspi_inst_SS0
                ↪  (hps_io_hps_io_qspi_inst_SS0),     //
                ↪  .hps_io_qspi_inst_SS0
627             .hps_io_hps_io_qspi_inst_CLK
                ↪  (hps_io_hps_io_qspi_inst_CLK),     //
                ↪  .hps_io_qspi_inst_CLK
628             .hps_io_hps_io_sdio_inst_CMD
                ↪  (hps_io_hps_io_sdio_inst_CMD),     //
                ↪  .hps_io_sdio_inst_CMD
629             .hps_io_hps_io_sdio_inst_D0
                ↪  (hps_io_hps_io_sdio_inst_D0),      //
                ↪  .hps_io_sdio_inst_D0
630             .hps_io_hps_io_sdio_inst_D1
                ↪  (hps_io_hps_io_sdio_inst_D1),      //
                ↪  .hps_io_sdio_inst_D1
631             .hps_io_hps_io_sdio_inst_CLK
                ↪  (hps_io_hps_io_sdio_inst_CLK),     //
                ↪  .hps_io_sdio_inst_CLK
632             .hps_io_hps_io_sdio_inst_D2
                ↪  (hps_io_hps_io_sdio_inst_D2),      //
                ↪  .hps_io_sdio_inst_D2
633             .hps_io_hps_io_sdio_inst_D3
                ↪  (hps_io_hps_io_sdio_inst_D3),      //
                ↪  .hps_io_sdio_inst_D3
634             .hps_io_hps_io_usb1_inst_D0
                ↪  (hps_io_hps_io_usb1_inst_D0),      //
                ↪  .hps_io_usb1_inst_D0
```

```
635                 .hps_io_hps_io_usb1_inst_D1
                    ↪  (hps_io_hps_io_usb1_inst_D1),      //
                    ↪  .hps_io_usb1_inst_D1
636                 .hps_io_hps_io_usb1_inst_D2
                    ↪  (hps_io_hps_io_usb1_inst_D2),      //
                    ↪  .hps_io_usb1_inst_D2
637                 .hps_io_hps_io_usb1_inst_D3
                    ↪  (hps_io_hps_io_usb1_inst_D3),      //
                    ↪  .hps_io_usb1_inst_D3
638                 .hps_io_hps_io_usb1_inst_D4
                    ↪  (hps_io_hps_io_usb1_inst_D4),      //
                    ↪  .hps_io_usb1_inst_D4
639                 .hps_io_hps_io_usb1_inst_D5
                    ↪  (hps_io_hps_io_usb1_inst_D5),      //
                    ↪  .hps_io_usb1_inst_D5
640                 .hps_io_hps_io_usb1_inst_D6
                    ↪  (hps_io_hps_io_usb1_inst_D6),      //
                    ↪  .hps_io_usb1_inst_D6
641                 .hps_io_hps_io_usb1_inst_D7
                    ↪  (hps_io_hps_io_usb1_inst_D7),      //
                    ↪  .hps_io_usb1_inst_D7
642                 .hps_io_hps_io_usb1_inst_CLK
                    ↪  (hps_io_hps_io_usb1_inst_CLK),     //
                    ↪  .hps_io_usb1_inst_CLK
643                 .hps_io_hps_io_usb1_inst_STP
                    ↪  (hps_io_hps_io_usb1_inst_STP),     //
                    ↪  .hps_io_usb1_inst_STP
644                 .hps_io_hps_io_usb1_inst_DIR
                    ↪  (hps_io_hps_io_usb1_inst_DIR),     //
                    ↪  .hps_io_usb1_inst_DIR
645                 .hps_io_hps_io_usb1_inst_NXT
                    ↪  (hps_io_hps_io_usb1_inst_NXT),     //
                    ↪  .hps_io_usb1_inst_NXT
646                 .hps_io_hps_io_spim0_inst_CLK
                    ↪  (hps_io_hps_io_spim0_inst_CLK),    //
                    ↪  .hps_io_spim0_inst_CLK
```

```
647                 .hps_io_hps_io_spim0_inst_MOSI
                    ↪  (hps_io_hps_io_spim0_inst_MOSI),   //
                    ↪  .hps_io_spim0_inst_MOSI
648                 .hps_io_hps_io_spim0_inst_MISO
                    ↪  (hps_io_hps_io_spim0_inst_MISO),   //
                    ↪  .hps_io_spim0_inst_MISO
649                 .hps_io_hps_io_spim0_inst_SS0
                    ↪  (hps_io_hps_io_spim0_inst_SS0),    //
                    ↪  .hps_io_spim0_inst_SS0
650                 .hps_io_hps_io_spim1_inst_CLK
                    ↪  (hps_io_hps_io_spim1_inst_CLK),    //
                    ↪  .hps_io_spim1_inst_CLK
651                 .hps_io_hps_io_spim1_inst_MOSI
                    ↪  (hps_io_hps_io_spim1_inst_MOSI),   //
                    ↪  .hps_io_spim1_inst_MOSI
652                 .hps_io_hps_io_spim1_inst_MISO
                    ↪  (hps_io_hps_io_spim1_inst_MISO),   //
                    ↪  .hps_io_spim1_inst_MISO
653                 .hps_io_hps_io_spim1_inst_SS0
                    ↪  (hps_io_hps_io_spim1_inst_SS0),    //
                    ↪  .hps_io_spim1_inst_SS0
654                 .hps_io_hps_io_uart0_inst_RX
                    ↪  (hps_io_hps_io_uart0_inst_RX),     //
                    ↪  .hps_io_uart0_inst_RX
655                 .hps_io_hps_io_uart0_inst_TX
                    ↪  (hps_io_hps_io_uart0_inst_TX),     //
                    ↪  .hps_io_uart0_inst_TX
656                 .hps_io_hps_io_i2c1_inst_SDA
                    ↪  (hps_io_hps_io_i2c1_inst_SDA),     //
                    ↪  .hps_io_i2c1_inst_SDA
657                 .hps_io_hps_io_i2c1_inst_SCL
                    ↪  (hps_io_hps_io_i2c1_inst_SCL),      //
                    ↪  .hps_io_i2c1_inst_SCL
658                     .chip8_device_VGA_R
                        ↪  (VGA_R),
659                     .chip8_device_VGA_G
                        ↪  (VGA_G),
```

```
660                     .chip8_device_VGA_B
                        ↪  (VGA_B),
661                     .chip8_device_VGA_CLK
                        ↪  (VGA_CLK),
662                     .chip8_device_VGA_HS
                        ↪  (VGA_HS),
663                     .chip8_device_VGA_VS
                        ↪  (VGA_VS),
664                     .chip8_device_VGA_BLANK_n
                        ↪  (VGA_BLANK_n),
665                     .chip8_device_OSC_50_B8A
                        ↪  (OSC_50_B8A),
666                     .chip8_device_AUD_ADCDAT
                        ↪  (AUD_ADCLRCK),
667                     .chip8_device_AUD_DACLRCK
                        ↪  (AUD_ADCDAT),
668                     .chip8_device_AUD_ADCLRCK
                        ↪  (AUD_DACLRCK),
669                     .chip8_device_AUD_DACDAT
                        ↪  (AUD_DACDAT),
670                     .chip8_device_AUD_XCK
                        ↪  (AUD_XCK),
671                     .chip8_device_AUD_BCLK
                        ↪  (AUD_BCLK),
672                     .chip8_device_AUD_I2C_SCLK
                        ↪  (AUD_I2C_SCLK),
673                     .chip8_device_AUD_I2C_SDAT
                        ↪  (AUD_I2C_SDAT),
674                     .chip8_device_AUD_MUTE
                        ↪  (AUD_MUTE),
675                     .chip8_device_VGA_SYNC_n
                        ↪  (VGA_SYNC_n)
676             );
677
678     endmodule
```

7.1.17 utils.svh

```
`ifndef CHIP8_UTILS_SVH
`define CHIP8_UTILS_SVH

function reg inbetween(input [17:0] low, value, high);
begin
  inbetween = value >= low && value <= high;
end
endfunction

`endif
```

7.1.18 enums.svh

```
/*************************************************************************
 * enums.svh
 *
 * Defines the enums used by Chip8_Top, Chip8_ALU, Chip8_CPU
 *
 * AUTHORS: David Watkins
 * Updated: Gabrielle Taylor 5/3/2016
 * Dependencies:

 ↪  *************************************************************************

`ifndef CHIP8_ENUMS
`define CHIP8_ENUMS

/**
 * ALU_f is an input into the ALU to specify which operation
 ↪  to execute
 *
 *     - ALU_f_OR          : bitwise OR
 *     - ALU_f_AND      : bitwise AND
 *     - ALU_f_XOR         : bitwise XOR
 *     - ALU_f_ADD         : Addition
 *     - ALU_f_MINUS         : Subtract
```

```
22  *      - ALU_f_LSHIFT    : Shift left
23  *      - ALU_f_RSHIFT    : Shift right
24  *      - ALU_f_EQUALS     : Equals compare
25  *      - ALU_f_GREATER    : Greater than compare
26  *      - ALU_f_INC     : Increment
27  */
28 typedef enum {
29     ALU_f_OR,
30     ALU_f_AND,
31     ALU_f_XOR,
32     ALU_f_ADD,
33     ALU_f_MINUS,
34     ALU_f_LSHIFT,
35     ALU_f_RSHIFT,
36     ALU_f_EQUALS,
37     ALU_f_GREATER,
38     ALU_f_INC,
39     ALU_f_NOP
40 } ALU_f ;
41
42 /**
43  * PC_SRC defines the behavior of the program counter from
    ↪  output from the CPU
44  *
45  * PC_SRC_STACK : Read from the current stack pointer
46  * PC_SRC_ALU   : Read the output from the processor
47  * PC_SRC_DEVICE: Read from linux input
48  * PC_SRC_SKIP  : Assign PC = PC + 4
49  * PC_SRC_HOLD  : Assign PC = PC
50  * PC_SRC_NEXT  : Assign PC = PC + 2
51  */
52 typedef enum {
53     PC_SRC_STACK,
54     PC_SRC_ALU,
55     PC_SRC_DEVICE,
56     PC_SRC_SKIP,
57     PC_SRC_HOLD,
58     PC_SRC_NEXT
```

```
59  } PC_SRC;
60
61  /**
62   * Chip8_STATE defines the current state of the emulator
63   *
64   * Chip8_RUNNING          : The emulator is loading and
    ↪  executing instructions
65   * Chip8_RUN_INSTRUCTION: The emulator will run only one
    ↪  instruction
66   * Chip8_PAUSED          : The emulator is paused and will
    ↪  only respond to linux
67   */
68  typedef enum {
69      Chip8_RUNNING,
70      Chip8_RUN_INSTRUCTION,
71     Chip8_PAUSED
72  } Chip8_STATE;
73
74  /**
75   * STACK_OP defines the behavior of the stack
76   *
77   * STACK_POP      : Pop the stack and write out value
78   * STACK_PUSH     : Push the input onto the stack
79   * STACK_HOLD     : Do nothing
80   */
81  typedef enum {
82      STACK_POP,
83      STACK_PUSH,
84      STACK_HOLD
85  } STACK_OP;
86
87  parameter NEXT_PC_WRITE_STAGE = 32'd12;
88  parameter CPU_CYCLE_LENGTH = 32'd50000;
89  parameter FRAMEBUFFER_REFRESH_HOLD = 32'd200000;
90  parameter COPY_THRESHOLD = 32'd500000;
91
92  `endif
```

7.2 Testbenches

7.2.1 Chip8_CPU_6xkk_7xkk.sv

```
1  `timescale 1ns/100ps
2
3  `include "../enums.svh"
4
5  /**
6   * Test to make sure that after an instruction happens,
7   * all output values get reset to their defaults
8   */
9  task automatic test_resets(ref logic[31:0] stage, ref int
   ↪  total, ref int failed,
10     ref logic delay_timer_WE, sound_timer_WE,
11     ref logic[7:0] delay_timer_writedata,
       ↪  sound_timer_writedata,
12     //ref PC_SRC pc_src,
13     ref logic[11:0] PC_writedata,
14     ref logic reg_WE1, reg_WE2,
15     ref logic[3:0] reg_addr1, reg_addr2,
16     ref logic[7:0] reg_writedata1, reg_writedata2,
17     ref logic mem_WE1, mem_WE2,
18     ref logic[11:0] mem_addr1, mem_addr2,
19     ref logic[ 7:0] mem_writedata1, mem_writedata2,
20     ref logic reg_I_WE,
21     ref logic[15:0] reg_I_writedata,
22     ref logic sp_push, sp_pop,
23     ref logic [4:0]    fb_addr_y,//max val = 31
24     ref logic [5:0]    fb_addr_x,//max val = 63
25     ref logic fb_writedata,fb_WE, fbreset,
26     ref logic halt_for_keypress);
27     #1ns;
28     assert(
29             delay_timer_WE             == 1'b0 &
30             sound_timer_WE             == 1'b0 &
31             delay_timer_writedata    == 8'b0 &
```

```
32              sound_timer_writedata    == 8'b0 &
33  //            pc_src                     == PC_SRC_NEXT &
34              PC_writedata              == 12'b0 &
35              reg_WE1                  == 1'b0 &
36              reg_WE2                  == 1'b0 &
37              reg_addr1                  == 4'b0 &
38              reg_addr2                  == 4'b0 &
39              reg_writedata1              == 8'b0 &
40              reg_writedata2             == 8'b0 &
41              mem_WE1                  == 1'b0 &
42              mem_WE2                  == 1'b0 &
43              mem_addr1                  == 12'h0 &
44              mem_addr2                  == 12'h0 &
45              mem_writedata1             == 8'h0 &
46              mem_writedata2             == 8'h0 &
47              reg_I_WE                  == 1'b0 &
48              reg_I_writedata             == 16'h0 &
49              sp_push                  == 1'b0 &
50              sp_pop                     == 1'b0 &
51              fb_addr_y    ==    5'h0 &
52              fb_addr_x    ==    6'h0 &
53              fb_writedata    ==    1'b0 &
54              fb_WE        ==    1'b0 &
55              fbreset                  == 1'b0 &
56              halt_for_keypress         == 1'b0) begin
57          $display("All outputs reset to their defaults");
58          total = total + 1;
59      end else begin
60          $display("Outputs were NOT reset to their defaults.
            ↪ Current stage: %d",stage);
61          failed = failed + 1;
62      end
63  endtask
64
65
66  /**
67   * Tests to make sure CPU outputs proper
68   * request signafor 7xkk by sending 7E54;
```

```
 */
task automatic test7xkk(ref logic cpu_clk,
                    ref logic[15:0] instruction,
                    ref logic[31:0] stage,
                    ref int total,
                    ref int failed,
                    ref logic reg_WE1, reg_WE2,
                    ref logic[3:0] reg_addr1, reg_addr2,
                    ref logic[7:0] reg_writedata1,
                     ↪  reg_writedata2,
                                        reg_readdata1);

    repeat (2) @(posedge cpu_clk);
    stage = 32'h0;
    instruction = 16'h7EF0;
    reg_readdata1 = 8'h01;

    wait(stage == 32'h2); #1ns;
    assert(reg_addr1 == instruction[11:8])
    else    $display("Instruction 7xkk failed with instruction
     ↪  %h in stage %d", instruction, stage);

    wait(stage == 32'h3); #1ns;
    assert(reg_addr1 == instruction[11:8] &
                (reg_writedata1 == (reg_readdata1 +
                 ↪  instruction[7:0])) &
                reg_WE1 == 1'b1
                //cannot check alu_in1, alu_in2, alu_cmd
                 ↪  without internal access
                ) begin
        total = total + 1;
        $display("Instruction 7xkk passed with instruction
         ↪  %h", instruction);
    end else begin
        failed = failed + 1;
        $display("Instruction 7xkk failed with instruction %h
         ↪  in stage %d", instruction, stage);
    end
```

```
101
102     wait(stage == 32'h4);
103
104
105 endtask
106
107 /**
108  * Tests to make sure instruction 6xkk works
109  * by testing 61F0.
110  */
111 task automatic test6xkk(ref logic cpu_clk,
112                         ref logic[15:0] instruction,
113                         ref logic[31:0] stage,
114                         ref int total,
115                         ref int failed,
116                         ref logic reg_WE1, reg_WE2,
117                         ref logic[3:0] reg_addr1, reg_addr2,
118                         ref logic[7:0] reg_writedata1,
                             ↪  reg_writedata2);
119     repeat (2) @(posedge cpu_clk);
120     stage = 32'h0;
121     repeat (1) @(posedge cpu_clk);
122
123     instruction = 16'h61F0;
124
125     wait(instruction && 16'h61F0 && stage == 32'h2);
126     #1ns;
127     assert((reg_addr1 == 4'h1) && (reg_writedata1 == 8'hF0) &&
         ↪  (reg_WE1 == 1'h1))begin
128         $display("6xkk passed with instr %h", instruction);
129         total = total + 1;
130     end else begin
131         $display("6xkk FAILED with instr %h", instruction);
132         failed = failed + 1;
133     end
134     wait(stage == 32'h3);
135
136 endtask
```

```
137
138 module Chip8_CPU_6xkk_7xkk( ) ;
139
140     logic cpu_clk;
141     logic[15:0] instruction;
142     logic[7:0]    reg_readdata1, reg_readdata2,
143                     mem_readdata1, mem_readdata2;
144     logic[15:0] reg_I_readdata;
145     logic[7:0] delay_timer_readdata;
146
147     logic key_pressed;
148     logic[3:0] key_press;
149
150     logic[11:0] PC_readdata;
151
152     logic[31:0] stage;
153
154     logic fb_readdata;
155
156     Chip8_STATE top_level_state;
157
158     logic delay_timer_WE, sound_timer_WE;
159     logic[7:0] delay_timer_writedata, sound_timer_writedata;
160
161     PC_SRC pc_src;
162     logic[11:0] PC_writedata;
163
164     logic reg_WE1, reg_WE2;
165     logic[3:0] reg_addr1, reg_addr2;
166     logic[7:0] reg_writedata1, reg_writedata2;
167
168     logic mem_WE1, mem_WE2;
169     logic[11:0] mem_addr1, mem_addr2;
170     logic[ 7:0] mem_writedata1, mem_writedata2;
171
172     logic reg_I_WE;
173     logic[15:0] reg_I_writedata;
174     logic sp_push, sp_pop;
```

```
175
176     logic [4:0]    fb_addr_y;//max val = 31
177     logic [5:0]    fb_addr_x;//max val = 63
178     logic        fb_writedata, //data to write to addresse.
179                             fb_WE, //enable writing to
                                ↪  address
180                             fbreset;
181
182     logic halt_for_keypress;
183     int total = 0;
184     int failed = 0;
185
186
187     Chip8_CPU dut(.*);
188
189
190
191     initial begin
192         cpu_clk = 0;
193         stage = 32'b0;
194         forever begin
195             #20ns cpu_clk = 1;
196             stage = stage + 1;
197             #20ns cpu_clk = 0;
198         end
199     end
200
201     initial begin
202         $display("Starting test tasks.");
203         test6xkk(cpu_clk, instruction,
                ↪  stage,total,failed,reg_WE1, reg_WE2,
204                         reg_addr1, reg_addr2,reg_writedata1,
                            ↪  reg_writedata2);
205
206         test_resets(stage, total,failed,delay_timer_WE,
                ↪  sound_timer_WE,
207             delay_timer_writedata, sound_timer_writedata,
                    ↪  /*PC_SRC pc_src,*/
```

```
208             PC_writedata,reg_WE1, reg_WE2,reg_addr1,
                ↪  reg_addr2,reg_writedata1, reg_writedata2,
209             mem_WE1, mem_WE2, mem_addr1,
                ↪  mem_addr2,mem_writedata1, mem_writedata2,
210             reg_I_WE,reg_I_writedata,sp_push,
                ↪  sp_pop,fb_addr_y,fb_addr_x,
211             fb_writedata,fb_WE, fbreset,halt_for_keypress);
212
213         test7xkk(cpu_clk, instruction,
                ↪  stage,total,failed,reg_WE1, reg_WE2,
214                         reg_addr1, reg_addr2, reg_writedata1,
                            ↪  reg_writedata2, reg_readdata1);
215
216         test_resets(stage, total,failed,delay_timer_WE,
                ↪  sound_timer_WE,
217             delay_timer_writedata, sound_timer_writedata,
                ↪  /*PC_SRC pc_src,*/
218             PC_writedata,reg_WE1, reg_WE2,reg_addr1,
                ↪  reg_addr2,reg_writedata1, reg_writedata2,
219             mem_WE1, mem_WE2, mem_addr1,
                ↪  mem_addr2,mem_writedata1, mem_writedata2,
220             reg_I_WE,reg_I_writedata,sp_push,
                ↪  sp_pop,fb_addr_y,fb_addr_x,
221             fb_writedata,fb_WE, fbreset,halt_for_keypress);
222
223         $display("Total number of tests passed: %d", total);
224         $display("Total number of tests failed: %d", failed);
225     end
226
227
228 endmodule
```

7.2.2 Chip8_CPU_big_testbench.sv

```
1 /**
2  * Author: Levi Oliver
3  * This code tests instructions
```

```
4   * 00e0 -- clear screen
5   * 6xkk -- load kk into Vx
6   * 7xkk -- sets Vx = Vx + kk
7   * Dxyn -- draws sprite! see instruction description in
    ↪  Chip8_CPU.sv
8   * Fx29 -- sets I to memory address of sprite representing
    ↪  value in Vx
9   * Fx33 -- stores BCD value for Vx in memory at I/I+1/I+2 ::
    ↪  h/t/o
10  */
11
12
13
14  `timescale 1ns/100ps
15
16  `include "../enums.svh"
17
18  /**
19   * Test to make sure that after an instruction happens,
20   * all output values get reset to their defaults
21   */
22  task automatic test_resets(ref logic cpu_clk, ref logic[31:0]
    ↪  stage, ref int total, ref int failed,
23      ref logic delay_timer_WE, sound_timer_WE,
24      ref logic[7:0] delay_timer_writedata,
        ↪  sound_timer_writedata,
25      //ref PC_SRC pc_src,
26      ref logic[11:0] PC_writedata,
27      ref logic reg_WE1, reg_WE2,
28      ref logic[3:0] reg_addr1, reg_addr2,
29      ref logic[7:0] reg_writedata1, reg_writedata2,
30      ref logic mem_WE1, mem_WE2,
31      ref logic[11:0] mem_addr1, mem_addr2,
32      ref logic[ 7:0] mem_writedata1, mem_writedata2,
33      ref logic reg_I_WE,
34      ref logic[15:0] reg_I_writedata,
35      ref logic sp_push, sp_pop,
36      ref logic [4:0]    fb_addr_y,//max val = 31
```

```
37      ref logic [5:0]    fb_addr_x,//max val = 63
38      ref logic fb_writedata,fb_WE, fbreset,
39      ref logic halt_for_keypress);
40      #3ns;
41      wait(cpu_clk == 1'b0);
42      assert(
43              delay_timer_WE             == 1'b0 &
44              sound_timer_WE             == 1'b0 &
45              delay_timer_writedata    == 8'b0 &
46              sound_timer_writedata    == 8'b0 &
47  //            pc_src                       == PC_SRC_NEXT &
48              PC_writedata             == 12'b0 &
49              reg_WE1                  == 1'b0 &
50              reg_WE2                  == 1'b0 &
51              reg_addr1                  == 4'b0 &
52              reg_addr2                  == 4'b0 &
53              reg_writedata1             == 8'b0 &
54              reg_writedata2             == 8'b0 &
55              mem_WE1                  == 1'b0 &
56              mem_WE2                  == 1'b0 &
57              mem_addr1                  == 12'h0 &
58              mem_addr2                  == 12'h0 &
59              mem_writedata1             == 8'h0 &
60              mem_writedata2             == 8'h0 &
61              reg_I_WE                 == 1'b0 &
62              reg_I_writedata            == 16'h0 &
63              fb_addr_y    ==    5'h0 &
64              fb_addr_x    ==    6'h0 &
65              fb_writedata    ==    1'b0 &
66              fb_WE        ==    1'b0 &
67              fbreset                  == 1'b0 &
68              halt_for_keypress        == 1'b0) begin
69          $display("All outputs reset to their defaults");
70          total = total + 1;
71      end else begin
72          $display("Outputs were NOT reset to their defaults.
            ↪ Current stage: %d",stage);
73          failed = failed + 1;
```

```
74      end
75      repeat (2) @(posedge cpu_clk);
76
77  endtask
78
79  task automatic testFx33(ref logic cpu_clk,
80                          ref logic[15:0] instruction,
81                          ref logic[31:0] stage,
82                          ref int              total, failed,
83                          ref logic[ 3:0] reg_addr1,
84                          ref logic[ 7:0] reg_readdata1,
85                          ref logic[11:0] mem_addr1,
86                          ref logic[ 7:0] mem_writedata1,
87                          ref logic          mem_WE1,
88                          ref logic[15:0] reg_I_readdata);
89
90      instruction = 16'hFe33;
91      stage = 32'b0;
92
93      wait(stage == 32'h2);#1ns;
94      assert(reg_addr1==instruction[11:8]);
95
96      reg_readdata1 = 8'd195;//1100 0011
97      reg_I_readdata = 16'h03F2;
98
99      wait(stage == 32'h3);#1ns;
100     assert(reg_addr1==instruction[11:8]  &
        ↪  mem_addr1==reg_I_readdata[11:0] &
101                 mem_addr1==reg_I_readdata[11:0] &
                    ↪  mem_writedata1==8'd1& mem_WE1)
102     else begin
103         $display("Improper BCD conversion. \n\tGiven value:
            ↪  %d\n\tCalculated hundreds place:
            ↪  %d",reg_readdata1, mem_writedata1);
104     end
105
106
107     wait(stage == 32'h4);#1ns;
```

```
108     assert(reg_addr1==instruction[11:8]  &
        ↪  mem_addr1==(1+reg_I_readdata[11:0]) &
        ↪  mem_writedata1==8'd9 & mem_WE1)
109     else begin
110         $display("Improper BCD conversion. \n\tGiven value:
            ↪  %d\n\tCalculated tens place: %d",reg_readdata1,
            ↪  mem_writedata1);
111     end
112
113     wait(stage == 32'h5);#1ns;
114     assert(mem_addr1==(2+reg_I_readdata[11:0]) &
        ↪  mem_writedata1==8'd5 & mem_WE1) begin
115         total = total + 1;
116         $display("Instruction Fx33 (store Vx as BCD in I/+1/+2
            ↪  in mem) is a success.");
117     end else begin
118         $display("Improper BCD conversion. \n\tGiven value:
            ↪  %d\n\tCalculated ones place: %d",reg_readdata1,
            ↪  mem_writedata1);
119     end
120
121     wait(stage == 32'h6); #1ns;
122 endtask
123
124 task automatic testFx29(ref logic cpu_clk,
125                         ref logic[15:0] instruction,
126                         ref logic[31:0] stage,
127                         ref int              total, failed,
128                         ref logic[ 3:0] reg_addr1,
129                         ref logic[ 7:0] reg_readdata1,
130                         ref logic[15:0] reg_I_writedata,
131                         ref logic         reg_I_WE);
132
133
134     stage = 32'b0;
135     instruction = 16'hFC29;
136
137     wait(stage == 32'h2);#1ns;
```

```
138     assert(reg_addr1==instruction[11:8])
139     else begin
140         failed = failed + 1;
141         $display("FX29 FAILED IN STAGE 2.");
142     end
143 
144     reg_readdata1 = 8'h0E;
145 
146     wait(stage == 32'h3);#1ns;
147     assert(reg_I_writedata == 16'd70  &  reg_I_WE == 1'b1)
        ↪ begin
148         $display("Fx29 (set I to Font by Vx) works!");
149         total = total + 1;
150     end else begin
151         $display("Fx29 failed in stage 3");
152     end
153 
154     wait(stage == 32'h4); #1ns;
155 
156 
157 endtask
158 
159 
160 task automatic testDxyn(ref logic cpu_clk,
161                         ref logic[15:0] instruction,
162                         ref logic[31:0] stage,
163                         ref int total,
164                         ref int failed,
165                         ref logic[3:0] reg_addr1, reg_addr2,
166                         ref logic reg_WE1, reg_WE2,
167                         ref logic[7:0] reg_readdata1,
                            ↪ reg_readdata2,
168                         ref logic[7:0] mem_readdata1,
                            ↪ mem_writedata1,
169                         ref logic mem_WE1,
170                         ref logic[11:0] mem_addr1,
171                         ref logic[5:0] fb_addr_x,
172                         ref logic[4:0] fb_addr_y,
```

```
173                     ref logic fb_writedata, fb_readdata,
                         ↪  fb_WE,
174                     ref logic[15:0] reg_I_readdata,
175                     ref logic bit_overwritten, isDrawing);
176     instruction = 16'hd392;
177     stage = 32'h0;
178     reg_I_readdata = 16'h0F0F;
179
180     if((stage == 32'h0) || (stage == 32'h1)) begin
181         assert(reg_WE1==1'b0  &  reg_WE2==1'b0  &
             ↪  mem_WE1==1'b0  &  fb_WE==1'b0)
182         else begin
183             $display("INSTR DXYN HAS INVALID WRITE_ENEABLE
                 ↪  VALS BEFORE STAGE 2");
184             failed = failed + 1;
185         end
186     end
187
188
189     wait(stage == 32'h2); #1ns;
190     assert(reg_addr1==instruction[11:8] &
         ↪  reg_addr2==instruction[7:4] &
191             mem_addr1==reg_I_readdata & reg_WE1==1'b0  &
                 ↪  reg_WE2==1'b0  &
192             mem_WE1==1'b0  &  fb_WE==1'b0 & isDrawing)
193     else begin
194         $display("INSTR DXYN HAS FAILED TO SET INITIAL OUTPUT
             ↪  VALS IN STAGE 3");
195         failed = failed + 1;
196     end
197
198     fb_readdata = 1'b0;
199     reg_readdata1 = 8'd3; //write to x = 3
200     reg_readdata2 = 8'd9; //write to y = 9
201                                 //that is position x + 64y =
                                     ↪  576
202     mem_readdata1 = 8'hAF;
203
```

```
204     wait(stage == 32'h3); #1ns;
205     assert(reg_addr1==instruction[11:8] &
        ↪  reg_addr2==instruction[7:4] &
206             mem_addr1==reg_I_readdata & reg_WE1==1'b0  &
                ↪  reg_WE2==1'b0  &
207             mem_WE1==1'b0  &  fb_WE==1'b0 & isDrawing)
208     else begin
209         $display("INSTR DXYN HAS FAILED TO HOLD VALS FROM
            ↪  STATE 2 IN STATE 3\n\tEx--mem_addr1=
            ↪  %h",mem_addr1);
210         failed = failed + 1;
211     end
212
213     wait(stage == 32'd15); #1ns;
214     assert(reg_addr1==instruction[11:8] &
        ↪  reg_addr2==instruction[7:4] &
215             mem_addr1==reg_I_readdata & reg_WE1==1'b0  &
                ↪  reg_WE2==1'b0  &
216             mem_WE1==1'b0  &  fb_WE==1'b0 & isDrawing)
217     else begin
218         $display("INSTR DXYN HAS FAILED TO HOLD VALS FROM
            ↪  STATE 2 IN STATE 15\n\tEx--mem_addr1=
            ↪  %h",mem_addr1);
219         failed = failed + 1;
220     end
221
222
223
224     wait(stage == 32'd17); #1ns;
225     assert(reg_addr1==instruction[11:8] &
        ↪  reg_addr2==instruction[7:4] &
226             mem_addr1==reg_I_readdata & fb_WE==1'b1  &
227             fb_addr_x==reg_readdata1 &
                ↪  fb_addr_y==reg_readdata2 &
228             fb_writedata==1'b1 & bit_overwritten==1'b0 &
                ↪  isDrawing)
229     else begin
```

```
230             $display("INSTR DXYN FAILED IN ITS FIRST WRITE
                ↪  STAGE");
231             failed = failed + 1;
232         end
233         wait(stage == 32'd18); #1ns;
234         assert(reg_addr1==instruction[11:8] &
                ↪  reg_addr2==instruction[7:4] &
235                 mem_addr1==reg_I_readdata & fb_WE==1'b0  &
236                 fb_addr_x==(reg_readdata1+1) &
                    ↪  fb_addr_y==reg_readdata2 &
237                 bit_overwritten==1'b0 & isDrawing)
238         else begin
239             $display("INSTR DXYN FAILED IN STAGE 18");
240             failed = failed + 1;
241         end
242         fb_readdata = 1'b1;
243         wait(stage == 32'd31); #1ns;
244         assert(reg_addr1==instruction[11:8] &
                ↪  reg_addr2==instruction[7:4] &
245                 mem_addr1==reg_I_readdata & fb_WE==1'b1  &
246                 fb_addr_x==(reg_readdata1+7) &
                    ↪  fb_addr_y==reg_readdata2 &
247                 bit_overwritten==1'b1 & isDrawing)
248         else begin
249             $display("INSTR DXYN FAILED IN STAGE 31");
250             failed = failed + 1;
251         end
252         fb_readdata = 1'b0;
253
254
255         wait(stage == 32'd32); #3ns;
256         assert(reg_addr1==instruction[11:8] &
                ↪  reg_addr2==instruction[7:4] &
257                 mem_addr1==(reg_I_readdata+1) & fb_WE==1'b0  &
258                 fb_addr_x==(reg_readdata1) &
                    ↪  fb_addr_y==(reg_readdata2+1) &
259                 bit_overwritten==1'b0 & isDrawing)
260         else begin
```

```
261             $display("INSTR DXYN FAILED IN STAGE
                ↪  32\n\treg_readdata2=%h\n\tfb_addr_y=%h",reg_readdata2,fb_addr_y);
262             $display("\tmem_addr1=%h\n\treg_I_readdata=%h",mem_addr1,reg_I_readdata);
263             failed = failed + 1;
264         end
265
266         mem_readdata1 = 8'b1101000;
267
268         wait(stage == 32'd33); #1ns;
269         assert(reg_addr1==instruction[11:8] &
                ↪  reg_addr2==instruction[7:4] &
270                 mem_addr1==(reg_I_readdata+1) & fb_WE==1'b1  &
271                 fb_addr_x==(reg_readdata1) &
                    ↪  fb_addr_y==(reg_readdata2+1) &
272                 bit_overwritten==1'b0 & isDrawing)
273         else begin
274             $display("INSTR DXYN FAILED IN STAGE
                ↪  33\n\treg_readdata2=%h\n\tfb_addr_y=%h",reg_readdata2,fb_addr_y);
275             $display("\tmem_addr1=%h\n\treg_I_readdata=%h",mem_addr1,reg_I_readdata);
276             failed = failed + 1;
277         end
278
279         wait(stage == 32'd39); #1ns;
280         assert(reg_addr1==instruction[11:8] &
                ↪  reg_addr2==instruction[7:4] &
281                 mem_addr1==(reg_I_readdata+1) & fb_WE==1'b1  &
282                 fb_addr_x==(reg_readdata1+8'h3) &
                    ↪  fb_addr_y==(reg_readdata2+1) &
283                 fb_writedata==1'b1 & bit_overwritten==1'b0 &
                    ↪  isDrawing)
284         else begin
285             $display("INSTR DXYN FAILED IN STAGE
                ↪  39\n\treg_readdata2=%h\n\tfb_addr_y=%h",reg_readdata2,fb_addr_y);
286             $display("\tmem_addr1=%h\n\treg_I_readdata=%h",mem_addr1,reg_I_readdata);
287             failed = failed + 1;
288         end
289
290         wait(stage == 32'd47); #1ns;
```

```
291     assert(reg_addr1==instruction[11:8] &
        ↪  reg_addr2==instruction[7:4] &
292             mem_addr1==(reg_I_readdata+1) & fb_WE==1'b1  &
293             fb_addr_x==(reg_readdata1+8'h7) &
                ↪  fb_addr_y==(reg_readdata2+1) &
294             fb_writedata==1'b0 & bit_overwritten==1'b0 &
                ↪  isDrawing)
295     else begin
296         $display("INSTR DXYN FAILED IN STAGE
            ↪  47\n\treg_readdata2=%h\n\tfb_addr_y=%h",reg_readdata2,fb_addr_y);
297         $display("\tmem_addr1=%h\n\treg_I_readdata=%h",mem_addr1,reg_I_readdata);
298         failed = failed + 1;
299     end
300 
301     wait(stage == 32'd48); #1ns;
302     assert(fb_WE==1'b0  &  bit_overwritten==1'b0 & isDrawing)
303     else begin
304         failed = failed + 1;
305         $display("INSTR DXYN FAILED IN STAGE: %d", stage);
306     end
307 
308 
309     wait(stage == 32'd61); #1ns;
310     assert(fb_WE==1'b0  &  bit_overwritten==1'b0 & isDrawing)
        ↪  begin
311         $display("Dxyn draw sprite works!! :D");
312         total = total + 1;
313     end else begin
314         $display("INSTR DXYN FAILED IN STAGE: %d", stage);
315         failed = failed + 1;
316     end
317 
318     $display("WOOOOOOBL");
319     wait(stage == 32'd100);#1ns;
320     stage = 32'b0;
321 endtask
322 
323 /**
```

```
 * Tests to make sure CPU outputs proper
 * request signafor 7xkk by sending 7E54;
 */
task automatic test7xkk(ref logic cpu_clk,
                    ref logic[15:0] instruction,
                    ref logic[31:0] stage,
                    ref int total,
                    ref int failed,
                    ref logic reg_WE1, reg_WE2,
                    ref logic[3:0] reg_addr1, reg_addr2,
                    ref logic[7:0] reg_writedata1,
                     ↪ reg_writedata2,
                                        reg_readdata1);

    repeat (2) @(posedge cpu_clk);
    stage = 32'h0;
    instruction = 16'h7EF0;
    reg_readdata1 = 8'h01;

    wait(stage == 32'h2); #1ns;
    assert(reg_addr1 == instruction[11:8])
    else    $display("Instruction 7xkk failed with instruction
     ↪ %h in stage %d", instruction, stage);

    wait(stage == 32'h3); #1ns;
    assert(reg_addr1 == instruction[11:8] &
                (reg_writedata1 == (reg_readdata1 +
                 ↪ instruction[7:0])) &
                reg_WE1 == 1'b1
                //cannot check alu_in1, alu_in2, alu_cmd
                 ↪ without internal access
                ) begin
        total = total + 1;
        $display("Instruction 7xkk passed with instruction
         ↪ %h", instruction);
    end else begin
        failed = failed + 1;
```

```
356         $display("Instruction 7xkk failed with instruction %h
             ↪  in stage %d", instruction, stage);
357     end
358 
359     wait(stage == 32'h4);
360 
361 
362 endtask
363 
364 /**
365  * Tests to make sure instruction 6xkk works
366  * by testing 61F0.
367  */
368 task automatic test6xkk(ref logic cpu_clk,
369                         ref logic[15:0] instruction,
370                         ref logic[31:0] stage,
371                         ref int total,
372                         ref int failed,
373                         ref logic reg_WE1, reg_WE2,
374                         ref logic[3:0] reg_addr1, reg_addr2,
375                         ref logic[7:0] reg_writedata1,
                             ↪  reg_writedata2);
376     repeat (2) @(posedge cpu_clk);
377     stage = 32'h0;
378     repeat (1) @(posedge cpu_clk);
379 
380     instruction = 16'h61F0;
381 
382     wait(instruction && 16'h61F0 && stage == 32'h2);
383     #1ns;
384     assert((reg_addr1 == 4'h1) && (reg_writedata1 == 8'hF0) &&
         ↪  (reg_WE1 == 1'h1))begin
385         $display("6xkk passed with instr %h", instruction);
386         total = total + 1;
387     end else begin
388         $display("6xkk FAILED with instr %h", instruction);
389         failed = failed + 1;
390     end
```

```
391     wait(stage == 32'h3);
392
393 endtask
394
395
396 task automatic test00E0(ref logic cpu_clk,
397                         ref logic[15:0] instruction,
398                         ref logic[31:0] stage,
399                         ref int total, failed,
400                         ref logic fb_WE, fb_writedata, fbreset,
401                         ref logic[5:0] fb_addr_x,
402                         ref logic[4:0] fb_addr_y);
403
404     stage = 32'h0;
405     instruction = 16'h00e0;
406
407     wait(stage == 32'h2); #1ns;
408     assert(fbreset == 1'b1)
409     else begin
410         failed = failed + 1;
411         $display("INSTR 00E0: fbreset NEVER SET HIGH.");
412     end
413
414     wait(stage == 32'h3); #1ns;
415     assert(fb_WE==1'b1  &  fb_writedata==1'b0  &  fb_addr_x==0
      ↪  &  fb_addr_y==0  &  fbreset==1'b0)
416     else begin
417             failed = failed + 1;
418             $display("INSTR 00E0: Did not start clearing at
              ↪  (x,y)=(0, 0). stage: %h", stage);
419     end
420     if(stage > 32'h2 & stage < 32'd28189  &  fbreset==1'b0)
      ↪  begin
421         assert(fb_WE==1'b1  &  fb_writedata==1'b0)
422         else begin
423             failed = failed + 1;
424             $display("INSTR 00E0: failed in stage: %h",
              ↪  stage);
```

```
425         end
426
427         wait(stage==32'd8188);#1ns;
428         assert(fb_WE==1'b1  &  fb_writedata==1'b0 &
            ↪  (&fb_addr_x) & (&fb_addr_y)  &  fbreset==1'b0)
            ↪  begin
429             $display("00e0 clear screen success!");
430             total = total + 1;
431         end else begin
432             failed = failed + 1;
433             $display("INSTR 00E0: x and y addresses of clear
                ↪  never reach
                ↪  63x31\n\tx=%h\n\ty=%h\stage=%d=%b",fb_addr_x,fb_addr_y,stage,stag
434         end
435
436     end
437
438     wait(stage == 32'd8189); #1ns;
439
440 endtask
441
442 module Chip8_CPU_big_testbench( ) ;
443
444     logic cpu_clk;
445     logic[15:0] instruction;
446     logic[7:0]    reg_readdata1, reg_readdata2,
447                     mem_readdata1, mem_readdata2;
448     logic[15:0] reg_I_readdata;
449     logic[7:0] delay_timer_readdata;
450
451     logic key_pressed;
452     logic[3:0] key_press;
453
454     logic[11:0] PC_readdata;
455
456     logic[31:0] stage;
457
458     logic fb_readdata;
```

```
459
460     Chip8_STATE top_level_state;
461
462     logic delay_timer_WE, sound_timer_WE;
463     logic[7:0] delay_timer_writedata, sound_timer_writedata;
464
465     PC_SRC pc_src;
466     logic[11:0] PC_writedata;
467
468     logic reg_WE1, reg_WE2;
469     logic[3:0] reg_addr1, reg_addr2;
470     logic[7:0] reg_writedata1, reg_writedata2;
471
472     logic mem_WE1, mem_WE2;
473     logic[11:0] mem_addr1, mem_addr2;
474     logic[ 7:0] mem_writedata1, mem_writedata2;
475
476     logic reg_I_WE;
477     logic[15:0] reg_I_writedata;
478     logic sp_push, sp_pop;
479
480     logic [4:0]    fb_addr_y;//max val = 31
481     logic [5:0]    fb_addr_x;//max val = 63
482     logic        fb_writedata, //data to write to addresse.
483                               fb_WE, //enable writing to
                                  ↪  address
484                               fbreset, isDrawing;
485
486     logic halt_for_keypress;
487
488     logic stk_reset;
489     STACK_OP stk_op;
490     logic[15:0] stk_writedata;
491     logic bit_overwritten;
492     logic mem_request;
493
494     int total = 0;
495     int failed = 0;
```

```
496
497
498     Chip8_CPU dut(.*);
499
500
501
502     initial begin
503         cpu_clk = 0;
504         stage = 32'b0;
505         forever begin
506             #20ns cpu_clk = 1;
507             stage = stage + 1;
508             #20ns cpu_clk = 0;
509         end
510     end
511
512     initial begin
513         $display("Starting test tasks.");
514         test6xkk(cpu_clk, instruction,
            ↪ stage,total,failed,reg_WE1, reg_WE2,
515                     reg_addr1, reg_addr2,reg_writedata1,
                        ↪ reg_writedata2);
516
517         test_resets(cpu_clk, stage,
            ↪ total,failed,delay_timer_WE, sound_timer_WE,
518             delay_timer_writedata, sound_timer_writedata,
                ↪ /*PC_SRC pc_src,*/
519             PC_writedata,reg_WE1, reg_WE2,reg_addr1,
                ↪ reg_addr2,reg_writedata1, reg_writedata2,
520             mem_WE1, mem_WE2, mem_addr1,
                ↪ mem_addr2,mem_writedata1, mem_writedata2,
521             reg_I_WE,reg_I_writedata,sp_push,
                ↪ sp_pop,fb_addr_y,fb_addr_x,
522             fb_writedata,fb_WE, fbreset,halt_for_keypress);
523
524         test7xkk(cpu_clk, instruction,
            ↪ stage,total,failed,reg_WE1, reg_WE2,
```

```
525                 reg_addr1, reg_addr2, reg_writedata1,
                     ↪ reg_writedata2, reg_readdata1);
526
527         test_resets(cpu_clk, stage,
             ↪ total,failed,delay_timer_WE, sound_timer_WE,
528             delay_timer_writedata, sound_timer_writedata,
                 ↪ /*PC_SRC pc_src,*/
529             PC_writedata,reg_WE1, reg_WE2,reg_addr1,
                 ↪ reg_addr2,reg_writedata1, reg_writedata2,
530             mem_WE1, mem_WE2, mem_addr1,
                 ↪ mem_addr2,mem_writedata1, mem_writedata2,
531             reg_I_WE,reg_I_writedata,sp_push,
                 ↪ sp_pop,fb_addr_y,fb_addr_x,
532             fb_writedata,fb_WE, fbreset,halt_for_keypress);
533
534         testDxyn(cpu_clk, instruction, stage,total,failed,
535                 reg_addr1, reg_addr2, reg_WE1, reg_WE2,
536                 reg_readdata1,
                     ↪ reg_readdata2,mem_readdata1,
537                 mem_writedata1,mem_WE1, mem_addr1,
                     ↪ fb_addr_x,fb_addr_y,
538                 fb_writedata, fb_readdata,
                     ↪ fb_WE,reg_I_readdata,
                     ↪ bit_overwritten,isDrawing);
539
540         test_resets(cpu_clk, stage,
             ↪ total,failed,delay_timer_WE, sound_timer_WE,
541             delay_timer_writedata, sound_timer_writedata,
                 ↪ /*PC_SRC pc_src,*/
542             PC_writedata,reg_WE1, reg_WE2,reg_addr1,
                 ↪ reg_addr2,reg_writedata1, reg_writedata2,
543             mem_WE1, mem_WE2, mem_addr1,
                 ↪ mem_addr2,mem_writedata1, mem_writedata2,
544             reg_I_WE,reg_I_writedata,sp_push,
                 ↪ sp_pop,fb_addr_y,fb_addr_x,
545             fb_writedata,fb_WE, fbreset,halt_for_keypress);
546
```

```
547         test00E0(cpu_clk,
            ↪ instruction,stage,total,failed,fb_WE,
            ↪ fb_writedata,fbreset,fb_addr_x,fb_addr_y);
548
549         test_resets(cpu_clk, stage,
            ↪ total,failed,delay_timer_WE, sound_timer_WE,
550             delay_timer_writedata, sound_timer_writedata,
                ↪ /*PC_SRC pc_src,*/
551             PC_writedata,reg_WE1, reg_WE2,reg_addr1,
                ↪ reg_addr2,reg_writedata1, reg_writedata2,
552             mem_WE1, mem_WE2, mem_addr1,
                ↪ mem_addr2,mem_writedata1, mem_writedata2,
553             reg_I_WE,reg_I_writedata,sp_push,
                ↪ sp_pop,fb_addr_y,fb_addr_x,
554             fb_writedata,fb_WE, fbreset,halt_for_keypress);
555
556         testFx29(
            ↪ cpu_clk,instruction,stage,total,failed,reg_addr1,reg_readdata1,reg_I_
557
558         test_resets(cpu_clk, stage,
            ↪ total,failed,delay_timer_WE, sound_timer_WE,
559             delay_timer_writedata, sound_timer_writedata,
                ↪ /*PC_SRC pc_src,*/
560             PC_writedata,reg_WE1, reg_WE2,reg_addr1,
                ↪ reg_addr2,reg_writedata1, reg_writedata2,
561             mem_WE1, mem_WE2, mem_addr1,
                ↪ mem_addr2,mem_writedata1, mem_writedata2,
562             reg_I_WE,reg_I_writedata,sp_push,
                ↪ sp_pop,fb_addr_y,fb_addr_x,
563             fb_writedata,fb_WE, fbreset,halt_for_keypress);
564
565         testFx33(cpu_clk,instruction,stage,total,
            ↪ failed,reg_addr1,reg_readdata1,mem_addr1,mem_writedata1,mem_WE1,
            ↪ reg_I_readdata);
566
567         test_resets(cpu_clk, stage,
            ↪ total,failed,delay_timer_WE, sound_timer_WE,
```

```
568             delay_timer_writedata, sound_timer_writedata,
                ↪ /*PC_SRC pc_src,*/
569             PC_writedata,reg_WE1, reg_WE2,reg_addr1,
                ↪ reg_addr2,reg_writedata1, reg_writedata2,
570             mem_WE1, mem_WE2, mem_addr1,
                ↪ mem_addr2,mem_writedata1, mem_writedata2,
571             reg_I_WE,reg_I_writedata,sp_push,
                ↪ sp_pop,fb_addr_y,fb_addr_x,
572             fb_writedata,fb_WE, fbreset,halt_for_keypress);
573
574         $display("Total number of tests passed: %d", total);
575         $display("Total number of tests failed: %d", failed);
576     end
577
578
579 endmodule
```

7.2.3 Chip8_CPU_testbench.sv

```
1  module Chip8_CPU_testbench ( ) ;
2      logic clk;
3      logic[15:0] instruction;
4      logic[3:0] testIn1, testIn2;
5      wire[7:0] testOut1, testOut2;
6
7      //Initialize module here
8      Chip8_CPU cpu (.cpu_clk(clk), .*);
9
10     initial begin
11         clk = 0;
12         forever
13             #20ns clk = ~clk;
14     end
15
16     initial begin
17         instruction = 16'h6122;
18         repeat (2)
```

```
            @(posedge clk);
        instruction = 16'h6020;
        repeat (2)
            @(posedge clk);
        instruction = 16'h8014;
        repeat (2)
            @(posedge clk);
        instruction = 16'h8014;
        repeat (2)
            @(posedge clk);
        instruction = 16'h8013;
        repeat (2)
            @(posedge clk);
        instruction = 16'h8015;
    end
endmodule
```

7.2.4 Chip8_Top_test.sv

```
module Chip8_Top_test ( ) ;
    //Garbage variables here
    logic             clk;
    logic             reset;
    logic [31:0]      writedata;
    logic             write;
    logic               chipselect;
    logic [17:0]     address;

    logic [31:0] data_out;
    logic [7:0]  VGA_R, VGA_G, VGA_B;
    logic        VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n;
    logic        VGA_SYNC_n;

    //Initialize module here
    Chip8_Top(.*);

    initial begin
```

```
        clk = 0;
        forever
            #20ns clk = ~clk;
    end

    initial begin // Initially set the PC to 200
        //Reset
        reset = 0;
        repeat (2)
            @(posedge clk);
        reset = 1;
        repeat (2)
            @(posedge clk);
        reset = 0;
    end
endmodule
```

7.2.5 delay_timer_testbench.sv

```
/*********************
* Timer Test Bench
* GT
*********************/

module delay_timer_testbench();
    logic clk;
    logic clk_60;
    logic reset;
    logic [7:0] data;
    logic write_enable;
    wire out;

    delay_timer dut(.write_enable, .clk, .clk_60, .data,
    ↪  .out);

    initial begin
        clk = 0;
```

```
18          forever
19              #20ns clk = ~clk;
20      end
21
22      initial begin
23          clk_60 = 1'b0;
24          reset = 0;
25          data = 8'b0000_0000;
26          write_enable = 0;
27
28          repeat (8) begin
29              @(posedge clk);
30              clk_60 = ~clk_60;
31          end
32
33          clk_60 = 1'b1;
34          data = 8'b0000_1000;
35          write_enable = 1;
36
37          repeat (2)
38              @(posedge clk);
39          write_enable = 0;
40
41          repeat (64) begin
42              @(posedge clk);
43              clk_60 = ~clk_60;
44          end
45
46      end
47
48  endmodule
49
```

7.2.6 Triple_port_reg_file_test.sv

```
1   `timescale 10ns/10ns
2
```

```
3  module Triple_port_reg_file_test ( ) ;
4      logic            cpu_clk; //system clock that controls
       ↪  writing data
5      logic[7:0]     writedata1, writedata2, VFwritedata; //data
       ↪  to be written to corresponding addresses
6      logic         WE1, WE2, WEVF; //enable writing on
       ↪  addressed registers
7      logic[3:0]     addr1, addr2; //addresses to write to and
       ↪  read from
8      logic[7:0]     readdata1, readdata2, VFreaddata; //data
       ↪  output from addressed registers
9
10
11     //Initialize module here
12     Chip8_register_file tprf (.cpu_clk(cpu_clk), .*);
13
14     initial begin
15         cpu_clk = 0;
16         forever
17             #20ns cpu_clk = ~cpu_clk;
18     end
19
20     initial begin
21         WE1 = 1;
22         writedata1 = 8'hEE;
23         addr1 = 4'h0;
24         repeat (2)
25             @(posedge cpu_clk);
26         WE2 = 1;
27         writedata2 = 8'h44;
28         addr2 = 4'h1;
29         repeat (2)
30             @(posedge cpu_clk);
31         WEVF = 1;
32         VFwritedata = 8'hFF;
33         repeat (2)
34             @(posedge cpu_clk);
35         WE1 = 0;
```

```
        WE2 = 0;
        WEVF = 0;
        addr1 = 4'h0;
        addr2 = 4'h0;
        repeat (2)
            @(posedge cpu_clk);
        WE1 = 1;
        WE2 = 1;
        WEVF = 1;
        addr1 = 4'h5;
        writedata1 = 8'h5;
        addr2 = 4'h6;
        writedata2 = 8'h6;
        VFwritedata = 8'hFE;
        repeat (2)
            @(posedge cpu_clk);
        WE1 = 0;
        WE2 = 0;
        WEVF = 0;
        addr1 = 4'h0;
        addr2 = 4'h0;
        repeat (2)
            @(posedge cpu_clk);
        addr1 = 4'h5;
        addr2 = 4'h6;
        repeat (2)
            @(posedge cpu_clk);
        addr1 = 4'h0;
        addr2 = 4'h1;
    end
endmodule
```

7.2.7 fb_testbench.sv

```
/*****************************************************************************
 * Stack Test Bench
 *
```

```
4   * Author: Gabrielle Taylor, Ashley Kling
5
    ↪  ********************************************************************************
6
7   //task automatic testReset(logic clk,
8   //                                  logic[4:0] fb_addr_y,
9   //                                  logic[5:0] fb_addr_x,
10  //                                  logic fb_writedata,
11  //                                  logic fb_WE,
12  //                                  logic reset);
13  //    fb_addr_y = 6'h0000;
14  //    fb_addr_x = 5'h0000;
15  //    fb_writedata = 1'h0;
16  //    fb_WE = 1'h0;
17  //
18  //    reset = 1'h1;
19  //    repeat(2) @(posedge clk);
20  //    reset = 1'h0;
21  //    repeat(2) @(posedge clk);
22  //endtask
23
24  task automatic testWriteOne(ref logic clk,
25                                  ref logic[4:0] fb_addr_y,
26                                  ref logic[5:0] fb_addr_x,
27                                  ref logic fb_writedata,
28                                  ref logic fb_WE,
29                                  ref logic fb_readdata,
30                                  ref logic reset,
31                                  ref int total);
32      fb_addr_y = 5'b00001;
33      fb_addr_x = 6'b000001;
34      fb_writedata = 1;
35      fb_WE = 1;
36      repeat(2) @(posedge clk);
37
38      fb_addr_y = 5'b00000;
39      fb_addr_x = 6'b000000;
40      fb_writedata = 0;
```

```
41      fb_WE = 0;
42      repeat(2) @(posedge clk);
43
44      fb_addr_y = 5'b00001;
45      fb_addr_x = 6'b000001;
46      fb_writedata = 0;
47      fb_WE = 0;
48      repeat(2) @(posedge clk);
49
50      repeat(2)
51          @(posedge clk);
52      assert (fb_readdata == 1'h1) begin
53          $display ("WriteOne TEST 1 : PASSED");
54          total = total + 1;
55      end
56      else $error("OR TEST 1 : FAILED (Got %h, Expected 1)",
         ↪  fb_readdata);
57
58      fb_addr_y = 6'h0000;
59      fb_addr_x = 5'h0000;
60      fb_writedata = 1'h0;
61      fb_WE = 1'h0;
62
63      reset = 1'h1;
64      repeat(2) @(posedge clk);
65      reset = 1'h0;
66      repeat(2) @(posedge clk);
67      //testReset(clk, fb_addr_y, fb_addr_x, fb_writedata,
         ↪  fb_WE, reset);
68
69  endtask
70
71  task automatic testWriteOneReadElse(ref logic clk,
72                                      ref logic[4:0] fb_addr_y,
73                                      ref logic[5:0] fb_addr_x,
74                                      ref logic fb_writedata,
75                                      ref logic fb_WE,
```

```
                                    ref logic fb_readdata,
                                    ref logic reset,
                                    ref int total);
    fb_addr_y = 5'b00001;
    fb_addr_x = 6'b000001;
    fb_writedata = 1;
    fb_WE = 1;
    repeat(2) @(posedge clk);

    fb_addr_y = 5'b00000;
    fb_addr_x = 6'b000000;
    fb_writedata = 0;
    fb_WE = 0;
    repeat(2) @(posedge clk);

    repeat(2)
        @(posedge clk);
    assert (fb_readdata == 1'h0) begin
        $display ("WriteOneRead Else TEST 2 : PASSED");
        total = total + 1;
    end
    else $error("OR TEST 2 : FAILED (Got %h, Expected 0)",
     ↪  fb_readdata);

    fb_addr_y = 6'h0000;
    fb_addr_x = 5'h0000;
    fb_writedata = 1'h0;
    fb_WE = 1'h0;

    reset = 1'h1;
    repeat(2) @(posedge clk);
    reset = 1'h0;
    repeat(2) @(posedge clk);
    //testReset(clk, fb_addr_y, fb_addr_x, fb_writedata,
     ↪  fb_WE, reset);

endtask
```

```
111
112
113 task automatic testWriteManyReadMany(ref logic clk,
114                                       ref logic[4:0] fb_addr_y,
115                                       ref logic[5:0] fb_addr_x,
116                                       ref logic fb_writedata,
117                                       ref logic fb_WE,
118                                       ref logic fb_readdata,
119                                       ref logic reset,
120                                       ref int total);
121     repeat(4) @(posedge clk);
122
123         fb_addr_y = 5'b00010;
124         fb_addr_x = 6'b000010;
125         fb_writedata = 1;
126         fb_WE = 1;
127         repeat(2) @(posedge clk);
128
129         fb_addr_y = 5'b00000;
130         fb_addr_x = 6'b000000;
131         fb_writedata = 0;
132         fb_WE = 0;
133         repeat(2) @(posedge clk);
134
135         fb_addr_y = 5'b00100;
136         fb_addr_x = 6'b000100;
137         fb_writedata = 1;
138         fb_WE = 1;
139         repeat(2) @(posedge clk);
140
141         fb_addr_y = 5'b01000;
142         fb_addr_x = 6'b001000;
143         fb_writedata = 1;
144         repeat(2) @(posedge clk);
145
146         fb_addr_y = 5'b10000;
147         fb_addr_x = 6'b010000;
148         fb_writedata = 1;
```

```
149         repeat(2) @(posedge clk);
150
151         //fb_WE = 0;
152         fb_addr_y = 5'b00000;
153         fb_addr_x = 6'b000000;
154         fb_writedata = 0;
155         fb_WE = 0;
156         repeat(2) @(posedge clk);
157
158         fb_addr_y = 5'b00010;
159         fb_addr_x = 6'b000010;
160         fb_writedata = 0;
161         fb_WE = 0;
162         repeat(2) @(posedge clk);
163
164         repeat(2)
165         @(posedge clk);
166         assert (fb_readdata == 1'h1) begin
167             $display ("WriteManyReadMany TEST 3 part 1 :
                ↪  PASSED");
168         end
169         else $error("WriteManyReadMany TEST 3 part 1 : FAILED
            ↪  (Got %h, Expected 1)", fb_readdata);
170
171         fb_addr_y = 5'b00000;
172         fb_addr_x = 6'b000000;
173         fb_writedata = 0;
174         fb_WE = 0;
175         repeat(2) @(posedge clk);
176
177         fb_addr_y = 5'b00100;
178         fb_addr_x = 6'b000100;
179         fb_writedata = 0;
180         fb_WE = 0;
181         repeat(2) @(posedge clk);
182
183         repeat(2)
```

```
184         @(posedge clk);
185         assert (fb_readdata == 1'h1) begin
186             $display ("WriteManyReadMany TEST 3 part 2 :
                ↪  PASSED");
187         end
188         else $error("WriteManyReadMany TEST 3 part 2 : FAILED
            ↪  (Got %h, Expected 1)", fb_readdata);
189
190         fb_addr_y = 5'b00000;
191         fb_addr_x = 6'b000000;
192         fb_writedata = 0;
193         fb_WE = 0;
194         repeat(2) @(posedge clk);
195
196         fb_addr_y = 5'b01000;
197         fb_addr_x = 6'b001000;
198         fb_writedata = 0;
199         fb_WE = 0;
200         repeat(2) @(posedge clk);
201
202         repeat(2)
203         @(posedge clk);
204         assert (fb_readdata == 1'h1) begin
205             $display ("WriteManyReadMany TEST 3 part 3 :
                ↪  PASSED");
206         end
207         else $error("WriteManyReadMany TEST 3 part 3 : FAILED
            ↪  (Got %h, Expected 1)", fb_readdata);
208
209         fb_addr_y = 5'b00000;
210         fb_addr_x = 6'b000000;
211         fb_writedata = 0;
212         fb_WE = 0;
213         repeat(2) @(posedge clk);
214
215         fb_addr_y = 5'b10000;
216         fb_addr_x = 6'b010000;
```

```
217         fb_writedata = 0;
218         fb_WE = 0;
219         repeat(2) @(posedge clk);
220 
221         repeat(2)
222         @(posedge clk);
223         assert (fb_readdata == 1'h1) begin
224             $display ("WriteManyReadMany TEST 3 part 4 :
                ↪  PASSED");
225         end
226         else $error("WriteManyReadMany TEST 3 part 4 : FAILED
            ↪  (Got %h, Expected 1)", fb_readdata);
227 
228         fb_addr_y = 5'b00000;
229         fb_addr_x = 6'b000000;
230         fb_writedata = 0;
231         fb_WE = 0;
232         repeat(2) @(posedge clk);
233 
234         fb_addr_y = 5'b10000;
235         fb_addr_x = 6'b100000;
236         fb_writedata = 0;
237         fb_WE = 0;
238         repeat(2) @(posedge clk);
239 
240         repeat(2)
241         @(posedge clk);
242         assert (fb_readdata == 1'h0) begin
243             $display ("WriteManyReadMany TEST 3 part 5 :
                ↪  PASSED");
244             total = total + 1;
245         end
246         else $error("OR TEST 3 part 5 : FAILED (Got %h,
            ↪  Expected 0)", fb_readdata);
247 
248     fb_addr_y = 6'h0000;
249     fb_addr_x = 5'h0000;
```

```
250     fb_writedata = 1'h0;
251     fb_WE = 1'h0;
252 
253     reset = 1'h1;
254     repeat(2) @(posedge clk);
255     reset = 1'h0;
256     repeat(2) @(posedge clk);
257     //testReset(clk, fb_addr_y, fb_addr_x, fb_writedata,
        ↪  fb_WE, reset);
258 endtask
259 
260 module fb_testbench();
261     logic clk;
262     logic reset;
263     logic [4:0] fb_addr_y;
264     logic [5:0] fb_addr_x;
265     logic fb_writedata;
266     logic fb_WE;
267     logic fb_readdata;
268     logic [7:0] VGA_R, VGA_G, VGA_B;
269     logic VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n;
270     logic VGA_SYNC_n;
271     int total;
272 
273     Chip8_framebuffer dut(.*);
274 
275     initial begin
276         clk = 0;
277         reset = 0;
278         fb_addr_y = 5'b00000;
279         fb_addr_x = 6'b000000;
280         fb_writedata = 0;
281         fb_WE = 0;
282         total = 0;
283         forever
284             #20ns clk = ~clk;
285     end
286 
```

```
287     initial begin
288 
289         $display("Starting test script...");
290         testWriteOne(clk, fb_addr_y, fb_addr_x, fb_writedata,
            ↪  fb_WE, fb_readdata, reset, total);
291         testWriteOneReadElse(clk, fb_addr_y, fb_addr_x,
            ↪  fb_writedata, fb_WE, fb_readdata, reset, total);
292         testWriteManyReadMany(clk, fb_addr_y, fb_addr_x,
            ↪  fb_writedata, fb_WE, fb_readdata, reset, total);
293 
294         repeat(2) @(posedge clk);
295 
296     end
297 
298 endmodule
299 
```

7.2.8 alu_testbench.sv

```
1  /*****************************************************************************
2   * alu_testbench.sv
3   *
4   * Contains tests for the following instructions:
5   *      - OR          - bitwise OR
6   *      - AND          - bitwise AND
7   *      - XOR         - bitwise XOR
8   *      - ADD         - Addition
9   *      - MINUS         - Subtract
10  *      - LSHIFT     - Shift left
11  *      - RSHIFT     - Shift right
12  *      - EQUALS      - Equals compare
13  *      - GREATER     - Greater than compare
14  *      - INC          - Increment
15  *
16  * This module is solely used by the Chip8_CPU module, &&
    ↪  relies on the ALU_f
17  * enum defined in enums.svh
```

```
18  *
19  * AUTHORS: David Watkins, Gabrielle Taylor
20  * Dependencies:
21  *     - enums.svh
22  *  - Chip8_CPU/Chip8_ALU.sv
23
    ↪  ********************************************************************************
24
25  `include "enums.svh"
26
27  task automatic testReset(logic [15:0] input1, input2,
28                           ALU_f alu_op);
29      input1 = 16'h0000;
30      input2 = 16'h0000;
31      alu_op = ALU_f_NOP;
32  endtask
33
34
35  /**
36   * Tests the OR instruction for the ALU
37   *
38   * @test 1
39   * @input input1 = 16'hF5A0
40   * @input input2 = 16'hFA50
41   * @input alu_op = ALU_f_OR
42   * @expected result = 16'FFF0
43   * @expected alu_carry = 1'b0
44   */
45  task automatic testOR(ref logic clk, alu_carry,
46                        ref logic [15:0] input1, input2, result,
47                        ref ALU_f alu_op,
48                        ref int total);
49      //Setup test 1
50      repeat(2)
51          @(posedge clk);
52      input1 = 16'hF5A0;
53      input2 = 16'hFA50;
54      alu_op = ALU_f_OR;
```

```
55
56      repeat(2)
57          @(posedge clk);
58      assert (result == 16'hFFF0 && alu_carry == 1'b0) begin
59          $display ("OR TEST 1 : PASSED");
60          total = total + 1;
61      end
62      else $error("OR TEST 1 : FAILED (Got %h, Expected fff0)",
        ↪  result);
63
64      testReset(input1, input2, alu_op);
65  endtask
66
67
68  /**
69   * Tests the AND instruction for the ALU
70   *
71   * @test 1
72   * @input input1 = 16'hF5A0
73   * @input input2 = 16'hFA50
74   * @input alu_op = ALU_f_AND
75   * @expected result = 16'F000
76   * @expected alu_carry = 1'b0
77   */
78  task automatic testAND(ref logic clk, alu_carry,
79                  ref logic [15:0] input1, input2, result,
80                  ref ALU_f alu_op,
81                  ref int total);
82      //Setup test 1
83      repeat(2)
84          @(posedge clk);
85      input1 = 16'hF5A0;
86      input2 = 16'hFA50;
87      alu_op = ALU_f_AND;
88
89      repeat(2)
90          @(posedge clk);
```

```
91      assert (result == 16'hF000 && alu_carry == 1'b0) begin
92          $display ("AND TEST 1 : PASSED");
93          total = total + 1;
94      end
95      else $error("AND TEST 1 : FAILED (Got %h, Expected f000)",
        ↪  result);
96
97      testReset(input1, input2, alu_op);
98  endtask
99
100
101 /**
102  * Tests the XOR instruction for the ALU
103  *
104  * @test 1
105  * @input input1 = 16'hF5A0
106  * @input input2 = 16'hFA50
107  * @input alu_op = ALU_f_XOR
108  * @expected result = 16'h0FF0
109  * @expected alu_carry = 1'b0
110  */
111 task automatic testXOR(ref logic clk, alu_carry,
112                 ref logic [15:0] input1, input2, result,
113                 ref ALU_f alu_op,
114                 ref int total);
115     //Setup test 1
116     repeat(2)
117         @(posedge clk);
118     input1 = 16'hF5A0;
119     input2 = 16'hFA50;
120     alu_op = ALU_f_XOR;
121
122     repeat(2)
123         @(posedge clk);
124     assert (result == 16'h0FF0 && alu_carry == 1'b0) begin
125         $display ("XOR TEST 1 : PASSED");
126         total = total + 1;
```

```
127     end
128     else $error("XOR TEST 1 : FAILED (Got %h, Expected 0ff0)",
        ↪  result);
129
130     testReset(input1, input2, alu_op);
131 endtask
132
133
134 /**
135  * Tests the MINUS instruction for the ALU
136  *
137  * @test 1
138  * @input input1 = 16'd180
139  * @input input2 = 16'd180
140  * @input alu_op = ALU_f_ADD
141  * @expected result = 16'd360
142  * @expected alu_carry = 1'b1
143  *
144  * @test 2
145  * @input input1 = 16'd5
146  * @input input2 = 16'd5
147  * @input alu_op = ALU_f_ADD
148  * @expected result = 16'd10
149  * @expected alu_carry = 1'b1
150  */
151 task automatic testADD(ref logic clk, alu_carry,
152                 ref logic [15:0] input1, input2, result,
153                 ref ALU_f alu_op,
154                 ref int total);
155     //Setup test 1
156     repeat(2)
157         @(posedge clk);
158     input1 = 16'd180;
159     input2 = 16'd180;
160     alu_op = ALU_f_ADD;
161
162     repeat(2)
```

```
163         @(posedge clk);
164     assert (result == 16'd360 && alu_carry == 1'b1) begin
165         $display ("ADD TEST 1 : PASSED");
166         total = total + 1;
167     end
168     else $error("ADD TEST 1 : FAILED (Got %d, Expected 360)
        ↪  (Got %d, Expected 1)", result, alu_carry);
169
170     //Setup test 2
171     repeat(2)
172         @(posedge clk);
173     input1 = 16'd5;
174     input2 = 16'd5;
175     alu_op = ALU_f_ADD;
176
177     repeat(2)
178         @(posedge clk);
179     assert (result == 16'd10 && alu_carry == 1'b0) begin
180         $display ("ADD TEST 2 : PASSED");
181         total = total + 1;
182     end
183     else $error("ADD TEST 2 : FAILED (Got %d, Expected 10)
        ↪  (Got %d, Expected 0)", result, alu_carry);
184
185     testReset(input1, input2, alu_op);
186 endtask
187
188
189 /**
190  * Tests the MINUS instruction for the ALU
191  *
192  * @test 1
193  * @input input1 = 16'hC3C3
194  * @input input2 = 16'hC3C3
195  * @input alu_op = ALU_f_MINUS
196  * @expected result = 16'h0000
197  * @expected alu_carry = 1'b0
```

```
198  *
199  * @test 2
200  * @input input1 = 16'hE0A5
201  * @input input2 = 16'h7003
202  * @input alu_op = ALU_f_MINUS
203  * @expected result = 16'h70A2
204  * @expected alu_carry = 1'b0
205  *
206  * @test 3
207  * @input input1 = 16'h7003
208  * @input input2 = 16'hE0A5
209  * @input alu_op = ALU_f_MINUS
210  * @expected result = 16'd36702
211  * @expected alu_carry = 1'b0
212  */
213 task automatic testMINUS(ref logic clk, alu_carry,
214                 ref logic [15:0] input1, input2, result,
215                 ref ALU_f alu_op,
216                 ref int total);
217
218     //Setup test 1
219     repeat(2)
220         @(posedge clk);
221     input1 = 16'hC3C3;
222     input2 = 16'hC3C3;
223     alu_op = ALU_f_MINUS;
224
225     repeat(2)
226         @(posedge clk);
227     assert (result == 16'h0000 && alu_carry == 1'b0) begin
228         $display ("MINUS TEST 1 : PASSED");
229         total = total + 1;
230     end
231     else $error("MINUS TEST 1 : FAILED (Got %h, Expected 0000)
     ↪  (Got %d, Expected 1)", result, alu_carry);
232
233     //Setup test 2
```

```
234     repeat(2)
235         @(posedge clk);
236     input1 = 16'hE0A5;
237     input2 = 16'h7003;
238     alu_op = ALU_f_MINUS;
239 
240     repeat(2)
241         @(posedge clk);
242     assert (result == 16'h70A2 && alu_carry == 1'b0) begin
243         $display ("MINUS TEST 2 : PASSED");
244         total = total + 1;
245     end
246     else $error("MINUS TEST 2 : FAILED (Got %h, Expected 70a2)
     ↪  (Got %d, Expected 1)", result, alu_carry);
247 
248     //Setup test 3
249     repeat(2)
250         @(posedge clk);
251     input1 = 16'h7003;
252     input2 = 16'hE0A5;
253     alu_op = ALU_f_MINUS;
254 
255     repeat(2)
256         @(posedge clk);
257     assert (result == 16'd36702 && alu_carry == 1'b1) begin
258         $display ("MINUS TEST 3 : PASSED");
259         total = total + 1;
260     end
261     else $error("MINUS TEST 3 : FAILED (Got %d, Expected
     ↪  36702) (Got %d, Expected 1)", result, alu_carry);
262 
263     testReset(input1, input2, alu_op);
264 endtask
265 
266 
267 /**
268  * Tests the lSHIFT instruction for the ALU
```

```
269  *
270  * @test 1
271  * @input input1 = 16'h0031
272  * @input input2 = 16'h0002
273  * @input alu_op = ALU_f_LSHIFT
274  * @expected result = 16'h00C4
275  * @expected alu_carry = 1'b0
276  *
277  * @test 2
278  * @input input1 = 16'h1111
279  * @input input2 = 16'h0001
280  * @input alu_op = ALU_f_LSHIFT
281  * @expected result = 16'h2222
282  * @expected alu_carry = 1'b0
283  */
284 task automatic testLSHIFT(ref logic clk, alu_carry,
285                 ref logic [15:0] input1, input2, result,
286                 ref ALU_f alu_op,
287                 ref int total);
288     //Setup test 1
289     repeat(2)
290         @(posedge clk);
291     input1 = 16'h0031;
292     input2 = 16'h0002;
293     alu_op = ALU_f_LSHIFT;
294 
295     repeat(2)
296         @(posedge clk);
297     assert (result == 16'h00C4 && alu_carry == 1'b0) begin
298         $display ("LSHIFT TEST 1 : PASSED");
299         total = total + 1;
300     end
301     else $error("LSHIFT TEST 1 : FAILED (Got %h, Expected
     ↪  00c4)", result);
302 
303     //Setup test 2
304     repeat(2)
```

```
305         @(posedge clk);
306     input1 = 16'h1111;
307     input2 = 16'h0001;
308     alu_op = ALU_f_LSHIFT;
309 
310     repeat(2)
311         @(posedge clk);
312     assert (result == 16'h2222 && alu_carry == 1'b0) begin
313         $display ("LSHIFT TEST 2 : PASSED");
314         total = total + 1;
315     end
316     else $error("LSHIFT TEST 2 : FAILED (Got %h, Expected
     ↪  2222)", result);
317 
318     testReset(input1, input2, alu_op);
319 endtask
320 
321 
322 /**
323  * Tests the RSHIFT instruction for the ALU
324  *
325  * @test 1
326  * @input input1 = 16'h0031
327  * @input input2 = 16'h0002
328  * @input alu_op = ALU_f_RSHIFT
329  * @expected result = 16'h000C
330  * @expected alu_carry = 1'b0
331  *
332  * @test 2
333  * @input input1 = 16'h1111
334  * @input input2 = 16'h0001
335  * @input alu_op = ALU_f_RSHIFT
336  * @expected result = 16'h0888
337  * @expected alu_carry = 1'b0
338  */
339 task automatic testRSHIFT(ref logic clk, alu_carry,
340                 ref logic [15:0] input1, input2, result,
```

```
341                 ref ALU_f alu_op,
342                 ref int total);
343     //Setup test 1
344     repeat(2)
345         @(posedge clk);
346     input1 = 16'h0031;
347     input2 = 16'h0002;
348     alu_op = ALU_f_RSHIFT;
349 
350     repeat(2)
351         @(posedge clk);
352     assert (result == 16'h000C && alu_carry == 1'b0) begin
353         $display ("RSHIFT TEST 1 : PASSED");
354         total = total + 1;
355     end
356     else $error("RSHIFT TEST 1 : FAILED (Got %h, Expected
     ↪  000c)", result);
357 
358     //Setup test 2
359     repeat(2)
360         @(posedge clk);
361     input1 = 16'h1111;
362     input2 = 16'h0001;
363     alu_op = ALU_f_RSHIFT;
364 
365     repeat(2)
366         @(posedge clk);
367     assert (result == 16'h0888 && alu_carry == 1'b0) begin
368         $display ("RSHIFT TEST 2 : PASSED");
369         total = total + 1;
370     end
371     else $error("RSHIFT TEST 2 : FAILED (Got %h, Expected
     ↪  0888)", result);
372 
373     testReset(input1, input2, alu_op);
374 endtask
375 
```

```
376
377 /**
378  * Tests the GREATER instruction for the ALU
379  *
380  * @test 1
381  * @input input1 = 16'd180
382  * @input input2 = 16'd15
383  * @input alu_op = ALU_f_GREATER
384  * @expected result = 16'd1
385  * @expected alu_carry = 1'b0
386  *
387  * @test 2
388  * @input input1 = 16'd15
389  * @input input2 = 16'd180
390  * @input alu_op = ALU_f_GREATER
391  * @expected result = 16'd0
392  * @expected alu_carry = 1'b0
393  *
394  * @test 3
395  * @input input1 = 16'd15
396  * @input input2 = 16'd15
397  * @input alu_op = ALU_f_GREATER
398  * @expected result = 16'd0
399  * @expected alu_carry = 1'b0
400  */
401 task automatic testGREATER(ref logic clk, alu_carry,
402                 ref logic [15:0] input1, input2, result,
403                 ref ALU_f alu_op,
404                 ref int total);
405     //Setup test 1
406     repeat(2)
407         @(posedge clk);
408     input1 = 16'd180;
409     input2 = 16'd15;
410     alu_op = ALU_f_GREATER;
411
412     repeat(2)
413         @(posedge clk);
```

```
414     assert (result == 16'd1 && alu_carry == 1'b0) begin
415         $display ("GREATER TEST 1 : PASSED");
416         total = total + 1;
417     end
418     else $error("GREATER TEST 1 : FAILED (Got %d, Expected
        ↪ 1)", result);
419
420     //Setup test 2
421     repeat(2)
422         @(posedge clk);
423     input1 = 16'd15;
424     input2 = 16'd180;
425     alu_op = ALU_f_GREATER;
426
427     repeat(2)
428         @(posedge clk);
429     assert (result == 16'd0 && alu_carry == 1'b0) begin
430         $display ("GREATER TEST 2 : PASSED");
431         total = total + 1;
432     end
433     else $error("GREATER TEST 2 : FAILED (Got %d, Expected
        ↪ 0)", result);
434
435     //Setup test 3
436     repeat(2)
437         @(posedge clk);
438     input1 = 16'd15;
439     input2 = 16'd15;
440     alu_op = ALU_f_GREATER;
441
442     repeat(2)
443         @(posedge clk);
444     assert (result == 16'd0 && alu_carry == 1'b0) begin
445         $display ("GREATER TEST 3 : PASSED");
446         total = total + 1;
447     end
```

```
448     else $error("GREATER TEST 3 : FAILED (Got %d, Expected
        ↪  0)", result);
449 
450     testReset(input1, input2, alu_op);
451 endtask
452 
453 
454 /**
455  * Tests the EQUALS instruction for the ALU
456  *
457  * @test 1
458  * @input input1 = 16'd8
459  * @input input2 = 16'd8
460  * @input alu_op = ALU_f_EQUALS
461  * @expected result = 16'd1
462  * @expected alu_carry = 1'b0
463  *
464  * @test 2
465  * @input input1 = 16'd8
466  * @input input2 = 16'd9
467  * @input alu_op = ALU_f_EQUALS
468  * @expected result = 16'd0
469  * @expected alu_carry = 1'b0
470  */
471 task automatic testEQUALS(ref logic clk, alu_carry,
472                 ref logic [15:0] input1, input2, result,
473                 ref ALU_f alu_op,
474                 ref int total);
475     //Setup test 1
476     repeat(2)
477         @(posedge clk);
478     input1 = 16'd8;
479     input2 = 16'd8;
480     alu_op = ALU_f_EQUALS;
481 
482     repeat(2)
483         @(posedge clk);
```

```
484     assert (result == 16'd1 && alu_carry == 1'b0) begin
485         $display ("EQUALS TEST 1 : PASSED");
486         total = total + 1;
487     end
488     else $error("EQUALS TEST 1 : FAILED (Got %d, Expected 1)",
        ↪  result);
489 
490     //Setup test 2
491     repeat(2)
492         @(posedge clk);
493     input1 = 16'd8;
494     input2 = 16'd9;
495     alu_op = ALU_f_EQUALS;
496 
497     repeat(2)
498         @(posedge clk);
499     assert (result == 16'd0 && alu_carry == 1'b0) begin
500         $display ("EQUALS TEST 2 : PASSED");
501         total = total + 1;
502     end
503     else $error("EQUALS TEST 2 : FAILED (Got %d, Expected 0)",
        ↪  result);
504 
505     testReset(input1, input2, alu_op);
506 endtask
507 
508 
509 /**
510  * Tests the INC instruction for the ALU
511  *
512  * @test 1
513  * @input input1 = 16'd8
514  * @input alu_op = ALU_f_INC
515  * @expected result = 16'd26
516  * @expected alu_carry = 1'b0
517  */
518 task automatic testINC(ref logic clk, alu_carry,
```

```
519                 ref logic [15:0] input1, input2, result,
520                 ref ALU_f alu_op,
521                 ref int total);
522     //Setup
523     repeat(2)
524         @(posedge clk);
525     input1 = 16'd8;
526     input2 = 16'h0000;
527     alu_op = ALU_f_INC;
528     repeat(16) begin
529         @(posedge clk);
530         input1 = result;
531     end
532 
533     //Check
534     repeat(2)
535         @(posedge clk);
536     assert (result == 16'd25 && alu_carry == 1'b0) begin
537         $display ("INC TEST : PASSED");
538         total = total + 1;
539     end
540     else $error("INC TEST : FAILED (Got %d, Expected 24)",
     ↪  result);
541 
542     testReset(input1, input2, alu_op);
543 endtask
544 
545 
546 module alu_testbench();
547     logic clk;
548     logic alu_carry;
549     logic [15:0] result;
550     logic[15:0] input1, input2;
551     ALU_f alu_op;
552     int total;
553 
554     Chip8_ALU dut(
```

```
555         .input1(input1),
556         .input2(input2),
557         .sel(alu_op),
558         .out(result),
559         .alu_carry(alu_carry));
560 
561     initial begin
562         clk = 0;
563         input1 = 16'h0000;
564         input2 = 16'h0000;
565         alu_op = ALU_f_NOP;
566         forever
567             #20ns clk = ~clk;
568     end
569 
570     initial begin
571         $display("Starting test script...");
572         testOR(clk, alu_carry, input1, input2, result, alu_op,
            ↪  total);
573         testAND(clk, alu_carry, input1, input2, result,
            ↪  alu_op, total);
574         testXOR(clk, alu_carry, input1, input2, result,
            ↪  alu_op, total);
575         testADD(clk, alu_carry, input1, input2, result,
            ↪  alu_op, total);
576         testMINUS(clk, alu_carry, input1, input2, result,
            ↪  alu_op, total);
577         testLSHIFT(clk, alu_carry, input1, input2, result,
            ↪  alu_op, total);
578         testRSHIFT(clk, alu_carry, input1, input2, result,
            ↪  alu_op, total);
579         testGREATER(clk, alu_carry, input1, input2, result,
            ↪  alu_op, total);
580         testEQUALS(clk, alu_carry, input1, input2, result,
            ↪  alu_op, total);
581         testINC(clk, alu_carry, input1, input2, result,
            ↪  alu_op, total);
582 
```

```
583         $display("TESTS PASSED : %d", total);
584     end
585
586 endmodule
587
```

7.3 Linux Code

7.3.1 chip8.c

```
1  /*
2   * Userspace program that communicates with the vga_ball
    ↪  device driver
3   * primarily through ioctls
4   *
5   * David Watkins (djw2146), Ashley Kling (ask2203)
6   * Columbia University
7   */
8
9  #include <stdio.h>
10 #include "chip8driver.h"
11 #include <sys/ioctl.h>
12 #include <sys/types.h>
13 #include <sys/stat.h>
14 #include <fcntl.h>
15 #include <string.h>
16 #include <unistd.h>
17 #include <signal.h>
18 #include <stdlib.h>
19 #include <pthread.h>
20
21 #include "usbkeyboard.h"
22
23 static int CHIP8_FONTSET[] =
24     {
25         0xF0, 0x90, 0x90, 0x90, 0xF0, //0
26         0x20, 0x60, 0x20, 0x20, 0x70, //1
```

```
        0xF0, 0x10, 0xF0, 0x80, 0xF0, //2
        0xF0, 0x10, 0xF0, 0x10, 0xF0, //3
        0x90, 0x90, 0xF0, 0x10, 0x10, //4
        0xF0, 0x80, 0xF0, 0x10, 0xF0, //5
        0xF0, 0x80, 0xF0, 0x90, 0xF0, //6
        0xF0, 0x10, 0x20, 0x40, 0x40, //7
        0xF0, 0x90, 0xF0, 0x90, 0xF0, //8
        0xF0, 0x90, 0xF0, 0x10, 0xF0, //9
        0xF0, 0x90, 0xF0, 0x90, 0x90, //A
        0xE0, 0x90, 0xE0, 0x90, 0xE0, //B
        0xF0, 0x80, 0x80, 0x80, 0xF0, //C
        0xE0, 0x90, 0x90, 0x90, 0xE0, //D
        0xF0, 0x80, 0xF0, 0x80, 0xF0, //E
        0xF0, 0x80, 0xF0, 0x80, 0x80  //F
    };

#define FONTSET_LENGTH 80
#define MEMORY_START 0x200
#define MEMORY_END 0x1000

int chip8_fd;
struct libusb_device_handle *keyboard;
uint8_t endpoint_address;
FILE *fp;

void quit_program(int signal) {
    printf("Chip8 is terminating\n");
    close(chip8_fd);
    exit(0);
}

void chip8_write(chip8_opcode *op) {
    if(ioctl(chip8_fd, CHIP8_WRITE_ATTR, op)) {
        perror("ioctl(CHIP8_WRITE_ATTR) failed");
        quit_program(0);
    }
}

```

```
65 void chip8_read(chip8_opcode *op) {
66     if(ioctl(chip8_fd, CHIP8_READ_ATTR, op)) {
67         perror("ioctl(CHIP8_READ_ATTR) failed");
68         printf("(%d, %d)\n", op->addr, op->data);
69         quit_program(0);
70     }
71 }
72
73 void setFramebuffer(int x, int y, int value) {
74     chip8_opcode op;
75     op.addr = FRAMEBUFFER_ADDR;
76     op.data = (1 << 12) | ((value & 0x1) << 11) | ((x & 0x3f)
       ↪  << 5) | (y & 0x1f);
77     chip8_write(&op);
78 }
79
80 int readFramebuffer(int x, int y) {
81     chip8_opcode op;
82     op.addr = FRAMEBUFFER_ADDR;
83     op.data = (0 << 12) | (0 << 11) | ((x & 0x3f) << 5) | (y &
       ↪  0x1f);
84     chip8_read(&op);
85     return op.readdata;
86 }
87
88 void flipPixel(int x, int y) {
89     int px = readFramebuffer(x, y);
90     setFramebuffer(x, y, !px);
91 }
92
93 void setMemory(int address, int data) {
94     chip8_opcode op;
95     op.addr = MEMORY_ADDR;
96     op.data = (1 << 20) | ((address & 0xfff) << 8) | (data &
       ↪  0xff);
97     chip8_write(&op);
98 }
99
```

```
100 int readMemory(int address) {
101     chip8_opcode op;
102     op.addr = MEMORY_ADDR;
103     op.data = (0 << 20) | ((address & 0xfff) << 8) | (0 &
        ↪  0xff);
104     chip8_read(&op);
105     return (op.readdata & 0xff);
106 }
107 
108 void setIRegister(int data) {
109     chip8_opcode op;
110     op.addr = I_ADDR;
111     op.data = (data & 0xffff);
112     chip8_write(&op);
113 }
114 
115 int readIRegister() {
116     chip8_opcode op;
117     op.addr = I_ADDR;
118     chip8_read(&op);
119     return op.readdata;
120 }
121 
122 int readRegister(int reg) {
123     chip8_opcode op;
124     op.addr = V0_ADDR + 4 * (reg & 0xf);
125     chip8_read(&op);
126     return op.readdata;
127 }
128 
129 void writeRegister(int reg, int value) {
130     chip8_opcode op;
131     op.addr = V0_ADDR + 4 * (reg & 0xf);
132     op.data = value & 0xff;
133     chip8_write(&op);
134 }
135 
136 /*
```

```
137 * Load the font set onto the chip8 sequentially
138 * Uses the op codes specified in chip8driver.h
139 */
140 void loadfontset() {
141     int i;
142     for(i = 0; i < FONTSET_LENGTH; ++i) {
143         setMemory(i, CHIP8_FONTSET[i]);
144         // int mem_val = readMemory(i);
145         // printf("(Address: %d) Wrote: %d, Read: %d\n", i,
            ↪  CHIP8_FONTSET[i], mem_val);
146         int got = readMemory(i);
147         if (CHIP8_FONTSET[i] != got) {
148             printf("Memory mismatch (expected: %d, got:
                ↪  %d)\n", CHIP8_FONTSET[i], got);
149         }
150     }
151 }
152
153 void refreshFrameBuffer() {
154     int x, y;
155     for(x = 0; x < 64; ++x) {
156         for(y = 0; y < 32; ++y) {
157
158             // int mem_val = readFramebuffer(x, y);
159             setFramebuffer(x, y, 0);
160
161             // printf("(x: %d, y: %d) Wrote: %d, Read: %d\n",
                ↪  x, y, !mem_val, mem_val);
162         }
163     }
164 }
165
166 /*
167 * Load a ROM file byte by byte onto the chip8
168 * Uses the op codes specified in chip8driver.h
169 */
170 void loadROM(const char* romfilename) {
171     FILE *romfile;
```

```
172     char buffer;
173     long filelen;
174     int i;
175 
176     romfile = fopen(romfilename, "rb");
177     fseek(romfile, 0, SEEK_END);
178     filelen = ftell(romfile);
179     rewind(romfile);
180 
181     for(i = 0; i < filelen && i < MEMORY_END - MEMORY_START;
        ↪ i++) {
182         fread((&buffer), 1, 1, romfile);
183 
184         setMemory(MEMORY_START + i, buffer);
185         int got = readMemory(MEMORY_START + i);
186         if (buffer != got) {
187             printf("Memory mismatch (expected: %d, got:
                ↪ %d)\n", buffer, got);
188         }
189     }
190 
191     for(i = i; i < MEMORY_END - MEMORY_START; ++i) {
192         setMemory(MEMORY_START + i, 0);
193     }
194 
195     fclose(romfile); // Close the file
196 }
197 
198 void resetMemory() {
199     int i;
200     for(i = 0; i < MEMORY_END; i++) {
201         setMemory(i, 0);
202     }
203 }
204 
205 void startChip8() {
206     chip8_opcode op;
207     op.addr = STATE_ADDR;
```

```
208     op.data = RUNNING_STATE;
209 
210     chip8_write(&op);
211 }
212 
213 void pauseChip8() {
214     chip8_opcode op;
215     op.addr = STATE_ADDR;
216     op.data = PAUSED_STATE;
217 
218     chip8_write(&op);
219 }
220 
221 void runInstructionChip8() {
222     chip8_opcode op;
223     op.addr = STATE_ADDR;
224     op.data = RUN_INSTRUCTION_STATE;
225 
226     chip8_write(&op);
227 }
228 
229 int chip8isRunning() {
230     chip8_opcode op;
231     op.addr = STATE_ADDR;
232     chip8_read(&op);
233     return op.readdata == RUNNING_STATE;
234 }
235 
236 int chip8isPaused() {
237     chip8_opcode op;
238     op.addr = STATE_ADDR;
239     chip8_read(&op);
240     return op.readdata == PAUSED_STATE;
241 }
242 
243 int chip8isRunInstruction() {
244     chip8_opcode op;
245     op.addr = STATE_ADDR;
```

```
246     chip8_read(&op);
247     return op.readdata == RUN_INSTRUCTION_STATE;
248 }
249 
250 int readPC() {
251     chip8_opcode op;
252     op.addr = PROGRAM_COUNTER_ADDR;
253     chip8_read(&op);
254     // fprintf(fp, "Instruction: %04x, PC: %d\n",
        ↪  (op.readdata & 0xfffff000) >> 12, (op.readdata &
        ↪  0xfff));
255     return (op.readdata & 0xfff);
256 }
257 
258 void writePC(int pc) {
259     chip8_opcode op;
260     op.addr = PROGRAM_COUNTER_ADDR;
261     op.data = pc;
262     chip8_write(&op);
263 }
264 
265 void printMemory() {
266     int i = 0;
267     for(i = 0; i < MEMORY_END; ++i) {
268         printf("%d ", readMemory(i));
269     }
270     printf("\n");
271 }
272 
273 void resetStack() {
274     chip8_opcode op;
275     op.addr = STACK_ADDR;
276     chip8_write(&op);
277 }
278 
279 int readSoundTimer() {
280     chip8_opcode op;
281     op.addr = SOUND_TIMER_ADDR;
```

```
282     chip8_read(&op);
283     return op.readdata;
284 }
285 
286 void writeSoundTimer(int value) {
287     chip8_opcode op;
288     op.addr = SOUND_TIMER_ADDR;
289     op.data = value;
290     chip8_write(&op);
291 }
292 
293 int readDelayTimer() {
294     chip8_opcode op;
295     op.addr = DELAY_TIMER_ADDR;
296     chip8_read(&op);
297     return op.readdata;
298 }
299 
300 void writeDelayTimer(int value) {
301     chip8_opcode op;
302     op.addr = DELAY_TIMER_ADDR;
303     op.data = value;
304     chip8_write(&op);
305 }
306 
307 void writeInstruction(int instruction) {
308     chip8_opcode op;
309     op.addr = INSTRUCTION_ADDR;
310     op.data = instruction;
311     chip8_write(&op);
312 
313     usleep(1000000);
314 }
315 
316 int readInstruction() {
317     chip8_opcode op;
318     op.addr = INSTRUCTION_ADDR;
319     chip8_read(&op);
```

```
320     return op.readdata;
321 }
322
323
324 void chip8writekeypress(char val, unsigned int ispressed) {
325     chip8_opcode op;
326     op.addr = KEY_PRESS_ADDR;
327     op.data = ((ispressed & 0x1) << 4) | (val & 0xf);
328     chip8_write(&op);
329 }
330
331 void printKeyState() {
332     chip8_opcode op;
333     op.addr = KEY_PRESS_ADDR;
334     chip8_read(&op);
335
336     printf("Is pressed: %d, Key val: %d, raw value: %d\n",
        ↪  (op.readdata & 0x10) >> 4, (op.readdata & 0xf),
        ↪  op.readdata);
337 }
338
339 void writeReset() {
340     chip8_opcode op;
341     op.addr = RESET_ADDR;
342     chip8_write(&op);
343 }
344
345 void printStatus(FILE *out, int index) {
346
347     fprintf(out, "Status %d\n", index);
348     if(chip8isPaused()) {
349         fprintf(out, "Paused\n");
350     } else if(chip8isRunning()) {
351         fprintf(out, "Running\n");
352     } else {
353         fprintf(out, "Run Instruction\n");
354     }
355     int pc = readPC();
```

```
356     int mem = readMemory(pc);
357     int mem2 = readMemory(pc + 1);
358     fprintf(out, "Program counter is: %d, instruction is: %04x
        ↪  / %04x\n", pc, mem << 4 | mem2, readInstruction());
359     fprintf(out, "I register: %d\n", readIRegister());
360     int i;
361     for(i = 0; i < 0x10; ++i) {
362     fprintf(out, "v%d: %d\n", i, readRegister(i));
363     }
364 
365     fprintf(out, "Sound timer: %d\n", readSoundTimer());
366     fprintf(out, "Delay timer: %d\n\n", readDelayTimer());
367 }
368 
369 
370 void resetChip8(const char* filename) {
371     //Need to write to registers and all
372     //Reload font set etc.
373     pauseChip8();
374     resetMemory();
375 
376     loadfontset();
377     if(filename != 0)
378         loadROM(filename);
379     refreshFrameBuffer();
380 
381     int i;
382     for(i = 0; i < 0x10; ++i) {
383         writeRegister(i, 0);
384     }
385 
386     // printMemory();
387     writePC(0x200);
388     setIRegister(0);
389     resetStack();
390     chip8writekeypress(0, 0);
391     writeSoundTimer(0);
392     writeDelayTimer(0);
```

```
393     printStatus(stdout, 0);
394 }
395 
396 /*
397 * Checks to see if a key is pressed, or depressed
398 * Then writes the associated action to the chip8 device
399 */
400 void checkforkeypress(const char *file) {
401     struct usb_keyboard_packet packet;
402     int transferred;
403     char keystate[12];
404 
405     libusb_interrupt_transfer(keyboard, endpoint_address,
     ↪  (unsigned char *) &packet, sizeof(packet),
     ↪  &transferred, 0);
406     if (transferred == sizeof(packet)) {
407         sprintf(keystate, "%02x %02x %02x", packet.modifiers,
         ↪  packet.keycode[0], packet.keycode[1]);
408         char val[1];
409         if (kbiskeypad(&packet, val)) {
410             chip8writekeypress(val[0], 1);
411         } else if(kbisstart(&packet)) {
412             startChip8();
413         } else if(kbispause(&packet)) {
414             pauseChip8();
415         } else if(kbisreset(&packet)) {
416             resetChip8(file);
417         } else {
418             chip8writekeypress(0, 0);
419         }
420     } else {
421         printf("Size mismatch %d %d\n", sizeof(packet),
         ↪  transferred);
422     }
423 }
424 
425 void *status_thread_f(void *ignored)
426 {
```

```
427     int index = 0;
428     while(1) {
429         printStatus(fp, index++);
430         usleep(4000);
431     }
432 
433     return NULL;
434 }
435 
436 int main(int argc, char** argv)
437 {
438     int runType = 0;
439     if(argc != 2 && argc != 3) {
440         printf("Usage: chip8 <romfilename>\n");
441         exit(1);
442     }
443 
444     if(argc == 3) runType = 1;
445 
446     /* Open the keyboard */
447     if ( (keyboard = openkeyboard(&endpoint_address)) == NULL
     ↪  ) {
448         fprintf(stderr, "Did not find a keyboard\n");
449         exit(1);
450     }
451 
452     static const char filename[] = "/dev/vga_led";
453     if ( (chip8_fd = open(filename, O_RDWR)) == -1) {
454         fprintf(stderr, "could not open %s\n", filename);
455         return -1;
456     }
457 
458     signal(SIGINT, quit_program);
459 
460     fp = fopen("log.txt", "w+");
461 
462     pthread_t status_thread;
463 
```

```
464     if(runType == 0) {
465         resetChip8(argv[1]);
466         // pthread_create(&status_thread, NULL,
            ↪  status_thread_f, NULL);
467 
468         while(chip8isRunning() || chip8isPaused()) {
469             // printStatus(stdout, 0);
470             checkforkeypress(argv[1]);
471             printKeyState();
472         }
473 
474         /* Terminate the status thread */
475         // pthread_cancel(status_thread);
476 
477         /* Wait for the status thread to finish */
478         // pthread_join(status_thread, NULL);
479     }
480 
481     fclose(fp);
482 
483     printf("Chip8 is terminating\n");
484     close(chip8_fd);
485     return 0;
486 }
```

7.3.2 chip8driver.c

```
1  /*
2   * Device driver for the CHIP8 SystemVerilog Emulator
3   *
4   * A Platform device implemented using the misc subsystem
5   *
6   * Columbia University
7   *
8   * References:
9   * Linux source: Documentation/driver-model/platform.txt
10  *               drivers/misc/arm-charlcd.c
```

```
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 *
 * "make" to build
 * insmod chip8driver.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree chip8driver.c
 */

#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 "chip8driver.h"

#define DRIVER_NAME "vga_led"

/*
 * Information about our device
 */
struct chip8_dev {
    struct resource res;        /* Resource: our registers
    ↪  */
    void __iomem *virtbase;     /* Where registers can be
    ↪  accessed in memory */
} dev;

/*
```

```
 * Writes an opcode (defined in chip8driver.h) to the device
 */
static void write_op(unsigned int addr, unsigned int instruction) {
    iowrite32(instruction, dev.virtbase + addr);
}

/*
 * Reads a value after sending a proper write opcode to the device
 */
static int read_value(unsigned int addr) {
    return ioread32(dev.virtbase + addr);
}

/*
* Checks to see if the address is validly formatted
*/
static int isValidInstruction(unsigned int addr, unsigned int instruction, int isWrite) {
    switch(addr) {
        //Register instructions are always okay
        case V0_ADDR: return 1;
        case V1_ADDR: return 1;
        case V2_ADDR: return 1;
        case V3_ADDR: return 1;
        case V4_ADDR: return 1;
        case V5_ADDR: return 1;
        case V6_ADDR: return 1;
        case V7_ADDR: return 1;
        case V8_ADDR: return 1;
        case V9_ADDR: return 1;
        case VA_ADDR: return 1;
        case VB_ADDR: return 1;
        case VC_ADDR: return 1;
        case VD_ADDR: return 1;
        case VE_ADDR: return 1;
        case VF_ADDR: return 1;
```

```
82          case I_ADDR:  return 1;
83
84          //Timer instructions are always okay
85          case SOUND_TIMER_ADDR: return 1;
86          case DELAY_TIMER_ADDR: return 1;
87
88          //Stack instructions are only valid if they conform
            ↪  to stack size
89          //Always looks at last three nibbles
90          case STACK_POINTER_ADDR: return !isWrite ||
            ↪  (instruction >= 0 && instruction < 64);
91          case STACK_ADDR:          return 1;
92
93          //Handle state transition
94          case STATE_ADDR: switch(instruction) {
95              case RUNNING_STATE: return 1;
96              case RUN_INSTRUCTION_STATE: return 1;
97              case PAUSED_STATE: return 1;
98              default: return !isWrite;
99          }
100
101         //Memory address
102         case MEMORY_ADDR:
103         //0000_0000_0001_AAAA_AAAA_AAAA_DDDD_DDDD
104         if(isWrite) return 1;
105         else return 2;
106
107         //Program Counter will always look at the last 3
            ↪  nibbles
108         case PROGRAM_COUNTER_ADDR: return 1;
109
110         //Always considers last nibble
111         case KEY_PRESS_ADDR: return 1;
112
113         //Make sure X, Y, and data values conform
114         //Data value will always be 1 byte
115         case FRAMEBUFFER_ADDR:
116         //0000_0000_0000_0000_0001_DXXX_XXXY_YYYY
```

```
117         if(isWrite) return 1;
118         else        return 2;
119 
120         case INSTRUCTION_ADDR: return 1;
121         case RESET_ADDR : return 1;
122 
123         default: break;
124     }
125 
126     return 0;
127 }
128 
129 
130 /*
131  * Handle ioctl() calls from userspace:
132  * Read or write the segments on single digits.
133  * Note extensive error checking of arguments
134  */
135 static long chip8_ioctl(struct file *f, unsigned int cmd,
    ↪ unsigned long arg)
136 {
137     chip8_opcode op;
138     int isWrite = 0;
139 
140     switch (cmd) {
141     case CHIP8_WRITE_ATTR:
142         if (copy_from_user(&op, (chip8_opcode *) arg,
            ↪ sizeof(chip8_opcode)))
143             return -EACCES;
144         if (!isValidInstruction(op.addr, op.data, 1))
145             return -EINVAL;
146         write_op(op.addr, op.data);
147         break;
148 
149     case CHIP8_READ_ATTR:
150         if (copy_from_user(&op, (chip8_opcode *) arg,
            ↪ sizeof(chip8_opcode)))
151             return -EACCES;
```

```
152         isWrite = isValidInstruction(op.addr, op.data, 0);
153         if(isWrite == 0)
154             return -EINVAL;
155 
156 
157         if(isWrite == 2)
158             write_op(op.addr, op.data);
159 
160         op.readdata = read_value(op.addr);
161         if (copy_to_user((chip8_opcode *) arg, &op,
            ↪  sizeof(chip8_opcode)))
162             return -EACCES;
163         break;
164 
165     default:
166         return -EINVAL;
167     }
168 
169     return 0;
170 }
171 
172 /* The operations our device knows how to do */
173 static const struct file_operations chip8_fops = {
174     .owner        = THIS_MODULE,
175     .unlocked_ioctl = chip8_ioctl,
176 };
177 
178 /* Information about our device for the "misc" framework --
    ↪  like a char dev */
179 static struct miscdevice chip8_misc_device = {
180     .minor        = MISC_DYNAMIC_MINOR,
181     .name         = DRIVER_NAME,
182     .fops         = &chip8_fops,
183 };
184 
185 /*
186  * Initialization code: get resources (registers) and
    ↪  display
```

```
 * a welcome message
 */
static int __init chip8_probe(struct platform_device *pdev)
{
    int ret;

    /* Register ourselves as a misc device: creates
     ↪  /dev/chip8 */
    ret = misc_register(&chip8_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;
    }

    /* Write paused state to the chip8 device */
    write_op(STATE_ADDR, PAUSED_STATE);

    return 0;

```

```
221 out_release_mem_region:
222     release_mem_region(dev.res.start,
        ↪  resource_size(&dev.res));
223 out_deregister:
224     misc_deregister(&chip8_misc_device);
225     return ret;
226 }
227 
228 /* Clean-up code: release resources */
229 static int chip8_remove(struct platform_device *pdev)
230 {
231     iounmap(dev.virtbase);
232     release_mem_region(dev.res.start,
        ↪  resource_size(&dev.res));
233     misc_deregister(&chip8_misc_device);
234     return 0;
235 }
236 
237 /* Which "compatible" string(s) to search for in the Device
    ↪  Tree */
238 #ifdef CONFIG_OF
239 static const struct of_device_id chip8_of_match[] = {
240     { .compatible = "altr,vga_led" },
241     {},
242 };
243 MODULE_DEVICE_TABLE(of, chip8_of_match);
244 #endif
245 
246 /* Information for registering ourselves as a "platform"
    ↪  driver */
247 static struct platform_driver chip8_driver = {
248     .driver    = {
249         .name    = DRIVER_NAME,
250         .owner    = THIS_MODULE,
251         .of_match_table = of_match_ptr(chip8_of_match),
252     },
253     .remove    = __exit_p(chip8_remove),
254 };
```

```
255
256 /* Called when the module is loaded: set things up */
257 static int __init chip8_init(void)
258 {
259     pr_info(DRIVER_NAME ": init\n");
260     return platform_driver_probe(&chip8_driver, chip8_probe);
261 }
262
263 /* Called when the module is unloaded: release resources */
264 static void __exit chip8_exit(void)
265 {
266     platform_driver_unregister(&chip8_driver);
267     pr_info(DRIVER_NAME ": exit\n");
268 }
269
270 module_init(chip8_init);
271 module_exit(chip8_exit);
272
273 MODULE_LICENSE("GPL");
274 MODULE_AUTHOR("The Chip8 Team");
275 MODULE_DESCRIPTION("Chip8 Emulator");
```

7.3.3 chip8driver.h

```
1  #ifndef __CHIP8_DRIVER_H__
2  #define __CHIP8_DRIVER_H__
3
4  #include <linux/ioctl.h>
5  #include <stdbool.h>
6
7  typedef struct {
8      unsigned int data;
9      unsigned int addr;
10     unsigned int readdata;
11 } chip8_opcode;
12
13 #define CHIP8_MAGIC 'q'
```

```
14
15 /* ioctls and their arguments */
16 #define CHIP8_WRITE_ATTR _IOW(CHIP8_MAGIC, 1, chip8_opcode
    ↪  *)
17 #define CHIP8_READ_ATTR  _IOWR(CHIP8_MAGIC, 2, chip8_opcode
    ↪  *)
18
19 /*
20 * To write data to a particular register, use iowrite with
21 * NNNNNNXX
22 * Where NNNNNN is ignored
23 * XX is the 8 bits to be written
24 *
25 * To read from a register use ioread with one of the
    ↪  following addresses
26 */
27 #define V0_ADDR 0x00
28 #define V1_ADDR 0x04
29 #define V2_ADDR 0x08
30 #define V3_ADDR 0x0C
31 #define V4_ADDR 0x10
32 #define V5_ADDR 0x14
33 #define V6_ADDR 0x18
34 #define V7_ADDR 0x1C
35 #define V8_ADDR 0x20
36 #define V9_ADDR 0x24
37 #define VA_ADDR 0x28
38 #define VB_ADDR 0x2C
39 #define VC_ADDR 0x30
40 #define VD_ADDR 0x34
41 #define VE_ADDR 0x38
42 #define VF_ADDR 0x3C
43
44 /*
45 * To write to the I index register
46 * NNNNDDDD
47 * DDDD is the 16 bits to write
48 *
```

```
49  * Use ioread to read from the I register
50  */
51  #define I_ADDR 0x40
52
53  /*
54  * To write to the sound timer
55  * NNNNNNDD
56  * Where DD is the number to write to the sound timer
57  *
58  * Use ioread to read from the sound timer
59  */
60  #define SOUND_TIMER_ADDR 0x44
61
62  /*
63  * To write to the delay timer
64  * NNNNNNDD
65  * Where DD is the number to write to the delay timer
66  *
67  * Use ioread to read from the delay timer
68  */
69  #define DELAY_TIMER_ADDR 0x48
70
71  /*
72  * To write to the stack pointer
73  * NNNNNNDD
74  * Where DD is the number to write to the stack pointer
75  * Only the last six bits are considered
76  *
77  * Use ioread to read from the stack pointer
78  */
79  #define STACK_POINTER_ADDR 0x60
80
81  /*
82  * To reset the stack, iowrite
83  */
84  #define STACK_ADDR 0x4C
85
86  /*
```

```
87  * To write to the program counter
88  * 0014DDDD
89  * Where DDDD is the number to write to the program counter
90  *
91  * Use ioread to read from the program counter
92  */
93  #define PROGRAM_COUNTER_ADDR 0x50
94
95  /*
96  * To write a keypress to the Chip8 control unit
97  * NNNNNNPD
98  * Where D is the number corresponding to a keypress 0-F
99  * Where P is whether a key is currently pressed or not (0x1,
    ↪  0x0)
100 */
101 #define KEY_PRESS_ADDR 0x54
102
103 /*
104 * To change the state of the Chip8
105 * 001600DD
106 * Where DD is an 8-bit number corresponding to varying
    ↪  states
107 * * 0x01 - Running
108 * * 0x02 - Loading ROM
109 * * 0x03 - Loading font set
110 * * 0x04 - Paused
111 * The state is initially set to loading font set
112 *
113 * Use ioread to read the state of the Chip8
114 */
115 #define STATE_ADDR 0x58
116 #define RUNNING_STATE 0x0
117 #define RUN_INSTRUCTION_STATE 0x1
118 #define PAUSED_STATE 0x2
119
120 /*
121 * To write to a location in memory
122 * 0000_0000_0001_AAAA_AAAA_AAAA_DDDD_DDDD
```

```
123 * Where DD is the 8-bit data that is to be written
124 * Where AAA is the 12-bit address to write the data
125 * Where W is a 1-bit value corresponding to a read or a
    ↪  write
126 *
127 * To read data from memory, use iowrite with
128 * 0000_0000_0000_AAAA_AAAA_AAAA_NNNN_NNNN
129 * Where AAA is the 12-bit address to read the data from
130 */
131 #define MEMORY_ADDR 0x64
132
133 /*
134 * In order to write data to the framebuffer
135 * 0000_0000_0000_0000_0001_DXXX_XXXY_YYYY
136 * Where XX is the x position (6 bits)
137 * Where YY is the y position (5 bits)
138 * Where DD is the 8-bits of data to write to the screen
139 *
140 * In order to read data from the framebuffer
141 * 0000_0000_0000_0000_0000_NXXX_XXXY_YYYY
142 * Where XX is the x position (6 bits)
143 * Where YY is the y position (5 bits)
144 * Where NN is ignored
145 */
146 #define FRAMEBUFFER_ADDR 0x5C
147
148 #define SCREEN_HEIGHT 480
149 #define SCREEN_WIDTH 640
150
151 #define MAX_FBX 32 //32/8
152 #define MAX_FBY 64
153
154 /*
155 * In order to write data to the instruction
156 * 0000_0000_0000_0000_IIII_IIII_IIII_IIII
157 * Where I corresponds to the 16 bits in the instruction
158 * The state must currently be in Chip8_RUN_INSTRUCTION
159 *
```

```
160 * In order to read data from the framebuffer
161 * 0000_0000_0000_0000_0000_NXXX_XXXY_YYYY
162 * Where XX is the x position (6 bits)
163 * Where YY is the y position (5 bits)
164 * Where NN is ignored
165 */
166 #define INSTRUCTION_ADDR 0x68
167
168 /**
169  * iowrite RESET_ADDR to reset all internal values
170  */
171 #define RESET_ADDR 0x6C
172
173 #endif //__CHIP8_DRIVER_H__
```

7.3.4 Makefile

```
1  ifneq (${KERNELRELEASE},)
2  # KERNELRELEASE defined: we are being compiled as part of
   ↪  the Kernel
3      obj-m := chip8driver.o
4  else
5
6  # We are being compiled as a module: use the Kernel build
   ↪  system
7
8  KERNEL_SOURCE := /usr/src/linux*
9  PWD := $(shell pwd)
10
11 CFLAGS = -Wall
12 OBJECTS = chip8.o usbkeyboard.o
13
14 default: module chip8
15
16 module:
17     ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} modules
18
```

```
chip8 : $(OBJECTS)
	cc $(CFLAGS) -o chip8 $(OBJECTS) -lusb-1.0 -pthread

lab2.o : lab2.c fbputchar.h usbkeyboard.h
usbkeyboard.o : usbkeyboard.c usbkeyboard.h

.PHONY : clean
clean:
	${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} clean
	${RM} chip8

socfpga.dtb : socfpga.dtb
	dtc -O dtb -o socfpga.dtb socfpga.dts

endif
```

7.3.5 socfpga.dts

```
/*
 *  Copyright (C) 2012 Altera Corporation <www.altera.com>
 *
 * This program is free software; you can redistribute it
 ↪  and/or modify
 * it under the terms of the GNU General Public License as
 ↪  published by
 * the Free Software Foundation; either version 2 of the
 ↪  License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be
 ↪  useful,
 * but WITHOUT ANY WARRANTY; without even the implied
 ↪  warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 ↪  the
 * GNU General Public License for more details.
 *
```

```
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 * dtc -O dtb -o socfpga.dtb socfpga.dts
 */

/dts-v1/;
/include/ "socfpga.dtsi"

/ {
    model = "Altera SOCFPGA Cyclone V";
    compatible = "altr,socfpga-cyclone5", "altr,socfpga";

    chosen {
        bootargs = "console=ttyS0,57600";
    };

    memory {
        name = "memory";
        device_type = "memory";
        reg = <0x0 0x40000000>; /* 1 GB */
    };

    aliases {
        /* this allow the ethaddr uboot environmnet variable contents
         * to be added to the gmac1 device tree blob.
         */
        ethernet0 = &gmac1;
    };

    soc {
        clkmgr@ffd04000 {
            clocks {
                osc1 {
```

```
                clock-frequency = <25000000>;
            };
        };
    };

    dcan0: d_can@ffc00000 {
        status = "disabled";
    };

    dcan1: d_can@ffc10000 {
        status = "disabled";
    };

    dwmmc0@ff704000 {
        num-slots = <1>;
        supports-highspeed;
        broken-cd;
        altr,dw-mshc-ciu-div = <4>;
        altr,dw-mshc-sdr-timing = <0 3>;

        slot@0 {
            reg = <0>;
            bus-width = <4>;
        };
    };

    ethernet@ff700000 {
        status = "disabled";
    };

    ethernet@ff702000 {
        phy-mode = "rgmii";
        phy-addr = <0xffffffff>; /* probe for phy addr */
    };

    i2c1: i2c@ffc05000 {
        status = "disabled";
```

```
85          };
86
87          i2c2: i2c@ffc06000 {
88              status = "disabled";
89          };
90
91          i2c3: i2c@ffc07000 {
92              status = "disabled";
93          };
94
95          qspi: spi@ff705000 {
96                  compatible = "cadence,qspi";
97                  #address-cells = <1>;
98                  #size-cells = <0>;
99                  reg = <0xff705000 0x1000>,
100                     <0xffa00000 0x1000>;
101                 interrupts = <0 151 4>;
102                 master-ref-clk = <400000000>;
103                 ext-decoder = <0>;  /* external decoder */
104                 num-chipselect = <4>;
105                 fifo-depth = <128>;
106                 bus-num = <2>;
107
108                 flash0: n25q00@0 {
109                     #address-cells = <1>;
110                     #size-cells = <1>;
111                     compatible = "n25q00";
112                     reg = <0>;    /* chip select */
113                     spi-max-frequency = <100000000>;
114                     page-size = <256>;
115                     block-size = <16>; /* 2^16, 64KB */
116                     quad = <1>;       /* 1-support quad */
117                     tshsl-ns = <200>;
118                     tsd2d-ns = <255>;
119                     tchsh-ns = <20>;
120                     tslch-ns = <20>;
121
```

```
                    partition@0 {
                        /* 8MB for raw data. */
                        label = "Flash 0 Raw Data";
                        reg = <0x0 0x800000>;
                    };
                    partition@800000 {
                        /* 8MB for jffs2 data. */
                        label = "Flash 0 jffs2 Filesystem";
                        reg = <0x800000 0x800000>;
                    };
                };

            };

        sysmgr@ffd08000 {
            cpu1-start-addr = <0xffd080c4>;
        };

        timer0@ffc08000 {
            clock-frequency = <100000000>;
        };

        timer1@ffc09000 {
            clock-frequency = <100000000>;
        };

        timer2@ffd00000 {
            clock-frequency = <25000000>;
        };

        timer3@ffd01000 {
            clock-frequency = <25000000>;
        };

        serial0@ffc02000 {
            clock-frequency = <100000000>;
        };
```

```
159
160         serial1@ffc03000 {
161             clock-frequency = <100000000>;
162         };
163
164         usb0: usb@ffb00000 {
165             status = "disabled";
166         };
167
168         usb1: usb@ffb40000 {
169             ulpi-ddr = <0>;
170         };
171
172         i2c0: i2c@ffc04000 {
173             speed-mode = <0>;
174         };
175
176         leds {
177             compatible = "gpio-leds";
178             hps0 {
179                 label = "hps_led0";
180                 gpios = <&gpio1 15 1>;
181             };
182
183             hps1 {
184                 label = "hps_led1";
185                 gpios = <&gpio1 14 1>;
186             };
187
188             hps2 {
189                 label = "hps_led2";
190                 gpios = <&gpio1 13 1>;
191             };
192
193             hps3 {
194                 label = "hps_led3";
195                 gpios = <&gpio1 12 1>;
```

```
196             };
197         };
198
199         lightweight_bridge: bridge@0xff200000 {
200             #address-cells = <1>;
201             #size-cells = <1>;
202             ranges = < 0x0 0xff200000 0x200000 >;
203
204             compatible = "simple-bus";
205
206             chip8: chip8@0 {
207                 compatible = "altr,chip8";
208                 reg = <0x0 0x2>;
209             };
210         };
211     };
212 };
213
214 &i2c0 {
215     lcd: lcd@28 {
216         compatible = "newhaven,nhd-0216k3z-nsw-bbw";
217         reg = <0x28>;
218         height = <2>;
219         width = <16>;
220         brightness = <8>;
221     };
222
223     eeprom@51 {
224         compatible = "atmel,24c32";
225         reg = <0x51>;
226         pagesize = <32>;
227     };
228
229     rtc@68 {
230         compatible = "dallas,ds1339";
231         reg = <0x68>;
232     };
```

```
};
```

7.3.6 usbkeyboard.c

```
#include "usbkeyboard.h"

#include <stdio.h>
#include <stdlib.h>

/* References on libusb 1.0 and the USB HID/keyboard protocol
 *
 * http://libusb.org
 *
 http://www.dreamincode.net/forums/topic/148707-introduction-to-using-libusb
 * http://www.usb.org/developers/devclass_docs/HID1_11.pdf
 * http://www.usb.org/developers/devclass_docs/Hut1_11.pdf
 */

/*
 * Find and return a USB keyboard device or NULL if not found
 * The argument con
 *
 */
struct libusb_device_handle *openkeyboard(uint8_t *endpoint_address) {
    libusb_device **devs;
    struct libusb_device_handle *keyboard = NULL;
    struct libusb_device_descriptor desc;
    ssize_t num_devs, d;
    uint8_t i, k;

    /* Start the library */
    if ( libusb_init(NULL) < 0 ) {
        fprintf(stderr, "Error: libusb_init failed\n");
        exit(1);
```

```
30      }
31
32      /* Enumerate all the attached USB devices */
33      if ( (num_devs = libusb_get_device_list(NULL, &devs)) < 0
        ↪  ) {
34          fprintf(stderr, "Error: libusb_get_device_list
            ↪  failed\n");
35          exit(1);
36      }
37
38      /* Look at each device, remembering the first HID device
        ↪  that speaks
39           the keyboard protocol */
40
41      for (d = 0 ; d < num_devs ; d++) {
42          libusb_device *dev = devs[d];
43          if ( libusb_get_device_descriptor(dev, &desc) < 0 ) {
44              fprintf(stderr, "Error:
                ↪  libusb_get_device_descriptor failed\n");
45              exit(1);
46          }
47
48          if (desc.bDeviceClass == LIBUSB_CLASS_PER_INTERFACE)
            ↪  {
49              struct libusb_config_descriptor *config;
50              libusb_get_config_descriptor(dev, 0, &config);
51              for (i = 0 ; i < config->bNumInterfaces ; i++)
52                  for ( k = 0 ; k <
                    ↪  config->interface[i].num_altsetting ; k++
                    ↪  ) {
53                      const struct libusb_interface_descriptor
                        ↪  *inter =
54                      config->interface[i].altsetting + k ;
55                      if ( inter->bInterfaceClass ==
                        ↪  LIBUSB_CLASS_HID &&
56                          inter->bInterfaceProtocol ==
                            ↪  USB_HID_KEYBOARD_PROTOCOL) {
57                          int r;
```

```
58                      if ((r = libusb_open(dev, &keyboard)) !=
                        ↪  0) {
59                          fprintf(stderr, "Error: libusb_open
                            ↪  failed: %d\n", r);
60                          exit(1);
61                      }
62                      if
                        ↪  (libusb_kernel_driver_active(keyboard,i))
63                          libusb_detach_kernel_driver(keyboard,
                            ↪  i);
64
                        ↪                          libusb_set_auto_detach_kernel_driver
                        ↪  i);
65                      if ((r = libusb_claim_interface(keyboard,
                        ↪  i)) != 0) {
66                          fprintf(stderr, "Error:
                            ↪  libusb_claim_interface failed:
                            ↪  %d\n", r);
67                          exit(1);
68                      }
69                      *endpoint_address =
                        ↪  inter->endpoint[0].bEndpointAddress;
70                      goto found;
71                  }
72              }
73          }
74      }
75
76      found:
77      libusb_free_device_list(devs, 1);
78
79      return keyboard;
80  }
81
82  /*
83  * Check to see if any value in the keypad is currently
    ↪  pressed
84  */
```

```
85  int kbiskeypad(struct usb_keyboard_packet* packet, char
    ↪  val[1]) {
86      uint8_t keycode = packet->keycode[0];
87
88      switch(keycode) {
89          case KEY1: val[0] = 0x1; return 1; break;
90          case KEY2: val[0] = 0x2; return 1; break;
91          case KEY3: val[0] = 0x3; return 1; break;
92          case KEYC: val[0] = 0xC; return 1; break;
93          case KEY4: val[0] = 0x4; return 1; break;
94          case KEY5: val[0] = 0x5; return 1; break;
95          case KEY6: val[0] = 0x6; return 1; break;
96          case KEYD: val[0] = 0xD; return 1; break;
97          case KEY7: val[0] = 0x7; return 1; break;
98          case KEY8: val[0] = 0x8; return 1; break;
99          case KEY9: val[0] = 0x9; return 1; break;
100         case KEYE: val[0] = 0xE; return 1; break;
101         case KEYA: val[0] = 0xA; return 1; break;
102         case KEY0: val[0] = 0x0; return 1; break;
103         case KEYB: val[0] = 0xB; return 1; break;
104         case KEYF: val[0] = 0xF; return 1; break;
105         default: break;
106     }
107
108     return 0;
109 }
110
111 int kbisstart(struct usb_keyboard_packet* packet) {
112     uint8_t keycode = packet->keycode[0];
113     return keycode == KEY_START;
114 }
115
116 int kbispause(struct usb_keyboard_packet* packet) {
117     uint8_t keycode = packet->keycode[0];
118     return keycode == KEY_PAUSE;
119 }
120
121 int kbisreset(struct usb_keyboard_packet* packet) {
```

```
122     uint8_t keycode = packet->keycode[0];
123     return keycode == KEY_RESET;
124 }
```

7.3.7 usbkeyboard.h

```
1  #ifndef _USBKEYBOARD_H
2  #define _USBKEYBOARD_H
3
4  #include <libusb-1.0/libusb.h>
5
6  #define USB_HID_KEYBOARD_PROTOCOL 1
7
8  /* Modifier bits */
9  #define USB_LCTRL  (1 << 0)
10 #define USB_LSHIFT (1 << 1)
11 #define USB_LALT   (1 << 2)
12 #define USB_LGUI   (1 << 3)
13 #define USB_RCTRL  (1 << 4)
14 #define USB_RSHIFT (1 << 5)
15 #define USB_RALT   (1 << 6)
16 #define USB_RGUI   (1 << 7)
17
18
19 /*
20 * Keyboard layout for the Chip8:
21 *     +---------+
22 *     | 1 2 3 C |
23 *     | 4 5 6 D |
24 *     | 7 8 9 E |
25 *     | A 0 B F |
26 *     +---------+
27 * In this program mapped to a qwerty keyboard:
28 *     +---------+
29 *     | 1 2 3 4 |
30 *     | Q W E R |
31 *     | A S D F |
```

```
32  *     | Z X C V |
33  *     +---------+
34  * Relying on the ascii mapping defined by the usb standard
35  */
36  #define KEY1 0x1E
37  #define KEY2 0x1F
38  #define KEY3 0x20
39  #define KEYC 0x21
40  #define KEY4 0x14
41  #define KEY5 0x1A
42  #define KEY6 0x08
43  #define KEYD 0x15
44  #define KEY7 0x04
45  #define KEY8 0x16
46  #define KEY9 0x07
47  #define KEYE 0x09
48  #define KEYA 0x1d
49  #define KEY0 0x1b
50  #define KEYB 0x06
51  #define KEYF 0x19
52
53  /*
54  * Three additional keys will be defined
55  * START - Enter key
56  * PAUSE - P key
57  * RESET - O key
58  */
59
60  #define KEY_START 0x28
61  #define KEY_PAUSE 0x13
62  #define KEY_RESET 0x12
63
64  struct usb_keyboard_packet {
65    uint8_t modifiers;
66    uint8_t reserved;
67    uint8_t keycode[6];
68  };
69
```

```
70 /* Find and open a USB keyboard device.  Argument should
   ↪  point to
71    space to store an endpoint address.  Returns NULL if no
   ↪  keyboard
72    device was found. */
73 extern struct libusb_device_handle *openkeyboard(uint8_t *);
74 int kbiskeypad(struct usb_keyboard_packet* packet, char
   ↪  val[1]);
75 int kbisstart(struct usb_keyboard_packet* packet);
76 int kbispause(struct usb_keyboard_packet* packet);
77 int kbisreset(struct usb_keyboard_packet* packet);
78
79 #endif
```

7.4 Git commit history

```
commit 9cbd68476bd8045e95e91276e283cbf7532033a8
Author: David Watkins <djw2146@columbia.edu>
Date:   Wed May 11 03:26:54 2016 -0400

Working version of Chip8 EMulator

commit 4cb0402e07bc27139bb2ec94ed30b98be097cab5
Author: David Watkins <djw2146@columbia.edu>
Date:   Wed May 11 02:34:55 2016 -0400

Added double buffering

commit 10e88a207f36cec3a484a047409808596004284c
Author: David Watkins <djw2146@columbia.edu>
Date:   Wed May 11 00:35:08 2016 -0400

Added +2 to stack when popping

commit 98026152ae7b99b62a3af3b9d7f37b16bcb71472
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Tue May 10 22:14:00 2016 -0400

Theoretical fix for infinite keypress waiting.

commit 680448658b65edfc694ef473da867d2d05be4fb3
Author: David Watkins <djw2146@columbia.edu>
Date:   Tue May 10 22:00:27 2016 -0400

Added stuff

commit b215d0325e08dff2d67519b258eb21cde202e898
Author: David Watkins <djw2146@columbia.edu>
Date:   Tue May 10 20:44:38 2016 -0400

Added correct PC_writedata case

commit 14fa0ec84b11bb1fcb3713d59a511428a3d529bf
Author: David Watkins <djw2146@columbia.edu>
```

```
39 Date:   Tue May 10 20:42:43 2016 -0400
40
41 Fixed enums
42
43 commit 1778fa04cc1ef9eedb8309aca2de91de227fa9c7
44 Merge: 1647c31 1b9a333
45 Author: David Watkins <djw2146@columbia.edu>
46 Date:   Tue May 10 20:36:59 2016 -0400
47
48 Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog
49
50 commit 1647c3141e4ed1cff44e8e7af2a0f07106b7bd5f
51 Author: David Watkins <djw2146@columbia.edu>
52 Date:   Tue May 10 20:36:37 2016 -0400
53
54 Added new changes to software
55
56 commit 1b9a333ffde12cf77340a763c1c69e3d6635252b
57 Merge: 6be3433 9aafb45
58 Author: lpo1234 <lpo1017@frontiernet.net>
59 Date:   Tue May 10 20:33:28 2016 -0400
60
61 Revamped CPU and Top after memory file findings. Pushing
    ↪ representative stack testbench.
62
63 commit 6be34339a828d7f32c80dc684393dbceac509f50
64 Author: lpo1234 <lpo1017@frontiernet.net>
65 Date:   Tue May 10 20:27:00 2016 -0400
66
67 Revamped CPU and some Top to fix with memory module. Adding
    ↪ representative stack testbench.
68
69 commit 9aafb4524c6b3e45f6acfe553fcd08cf8440b4c3
70 Author: David Watkins <djw2146@columbia.edu>
71 Date:   Tue May 10 18:51:22 2016 -0400
72
73 Added missing enums
```

```
74
75  commit b71d4db1b9469cad2e763d550fc0f46e0c6f8c8a
76  Author: David Watkins <djw2146@columbia.edu>
77  Date:   Tue May 10 15:59:10 2016 -0400
78
79  Added changed stuff
80
81  commit cc8bc08af9a933ca683d01b0b0b2fb45495490fa
82  Merge: e9f7318 e1cd8d1
83  Author: lpo1234 <lpo1017@frontiernet.net>
84  Date:   Mon May 9 20:19:39 2016 -0400
85
86  Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog
87
88  commit e9f73189e4d6b106bb55e6d1d9112f23d9938ed0
89  Author: lpo1234 <lpo1017@frontiernet.net>
90  Date:   Mon May 9 20:19:20 2016 -0400
91
92  Possible draw-sprite fix.
93
94  commit e1cd8d18d6439d85845f1f281d040a3163f5ccbd
95  Author: Ashley Kling <ask2203@columbia.edu>
96  Date:   Mon May 9 18:43:23 2016 -0400
97
98  added tests for 1 and 2, expanded a CPU instruction to
    ↪ accomodate stack
99
100 commit 67d9cb4db4bf466ad8e0c16fa2de5d1fcfaae741
101 Merge: d9eed9c 1bff01e
102 Author: Ashley Kling <ask2203@columbia.edu>
103 Date:   Mon May 9 18:40:22 2016 -0400
104
105 Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog
106
107 commit 1bff01e901e62f68ec67f276c111c94e0b75a832
108 Author: lpo1234 <lpo1017@frontiernet.net>
```

```
Date:   Mon May 9 18:39:38 2016 -0400

Updated big CPU testbench

commit 2b8da46248698d2952be6ae234e2613a03dccf1e
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Mon May 9 18:00:04 2016 -0400

Took out multiplier in led emulator. Small fixes for mem
 ↪  writing in Chip8_Top.

commit d9eed9ce7aae3db22a66af52f19c133b49498317
Author: Ashley Kling <ask2203@columbia.edu>
Date:   Mon May 9 16:31:30 2016 -0400

updated 00EE in CPU, added that test to testbench.

commit 839b32f08ebb3730f0e67d8a08aa03fe228b03e9
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Mon May 9 15:36:28 2016 -0400

Fixed simple typo. -_-

commit fc3d05aa7a2c74c29b18bb0b21b83751dcb6ea09
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Sun May 8 18:37:50 2016 -0400

Bug fixes in CPU-Top memory request interface. Switched instrs
 ↪  Fx65/Fx55 so they're right.

commit 99deaae0b4539d29b3ceb3ebee0f390a14ac34e0
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Sun May 8 16:58:37 2016 -0400

Ash found small bug in Dxyn mem-request pattern. Fixed.

commit d197eca1ba52494f9445ee34d2635c71a0e75b4b
Author: David Watkins <djw2146@columbia.edu>
```

```
Date:   Sun May 8 09:52:38 2016 -0400

Chip8 now outputs characters

commit 0058cc0831b800947b6b85f8b80f4f563a5a086a
Merge: d8cc820 2e1e52c
Author: David Watkins <djw2146@columbia.edu>
Date:   Sun May 8 05:45:09 2016 -0400

Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog

commit d8cc820ec98ee24a27b830187f76125895a1223d
Author: David Watkins <djw2146@columbia.edu>
Date:   Sun May 8 05:44:43 2016 -0400

Changed top level files

commit 2e1e52cfba28bce7b60bbc84277c0de3e4d46dd8
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Sun May 8 05:40:44 2016 -0400

Updated cpu big testbench to reflect Dxyn isDrawing flag.

commit 7aa045ae83ba86b0d7abdd8765947133d2055667
Author: David Watkins <djw2146@columbia.edu>
Date:   Sun May 8 05:28:17 2016 -0400

Added isDrawing flag

commit 9f51f4a9b46295363cf6bf1519678000f014be25
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Sun May 8 04:59:58 2016 -0400

Added a few testing instructions.

commit ac45ce0a7523b032def876b7aa6397256199528d
Merge: 77f28cb d94043e
```

```
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Sun May 8 02:54:36 2016 -0400

Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog

commit 77f28cb7988ed39885f5a3a29bfdda65e91636bd
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Sun May 8 02:52:11 2016 -0400

Added big testbench for CPU. Tested Dxyn draw-sprite::it works.

commit d94043ea52b16bf74ec6d5514691f516681d9008
Author: Ashley Kling <ask2203@columbia.edu>
Date:   Sun May 8 02:19:05 2016 -0400

added a cpu fix and tests for instructions 3-5 and 9-E
    ↪ (inclusive)

commit fa9819b7a998b5ff092a984cde2e2ba2132659d3
Author: Ashley Kling <ask2203@columbia.edu>
Date:   Sun May 8 01:16:38 2016 -0400

8-series cpu testbench + alu fix

commit 89203c16085cd8708c700188b375b7d85808d485
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Sun May 8 01:02:16 2016 -0400

Cleaned up Dxyn draw sprite cmd. Still untested.

commit d5049e1a6185356d3af2f74076267fd49ebb99eb
Merge: 85cd6a8 22030b6
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Sun May 8 00:56:13 2016 -0400

Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog
```

```
217
218 commit 85cd6a80b1e954573d7fa240c019834ef1b7b828
219 Author: lpo1234 <lpo1017@frontiernet.net>
220 Date:   Sun May 8 00:54:43 2016 -0400
221
222 Changed how draw-sprite-command Dxyn works. Untested.
223
224 commit 22030b63e776204af143883c4f288e194c65b43d
225 Author: David Watkins <djw2146@columbia.edu>
226 Date:   Sun May 8 00:09:25 2016 -0400
227
228 Added proper key reading
229
230 commit fe2f4e96b21c6db9acc523832381fa6d227c44c2
231 Merge: 85e2d54 d63d5cb
232 Author: David Watkins <djw2146@columbia.edu>
233 Date:   Sat May 7 21:49:22 2016 -0400
234
235 Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog
236
237 commit 85e2d54f55d3a089d181bc78796ff2ba45c787f1
238 Author: David Watkins <djw2146@columbia.edu>
239 Date:   Sat May 7 21:49:02 2016 -0400
240
241 Added reading and writing proper functionality
242
243 commit d63d5cb3591f7d481d0920abe76425f3113bc6db
244 Author: Ashley Kling <ask2203@columbia.edu>
245 Date:   Sat May 7 19:43:33 2016 -0400
246
247 basic framebuffer testbench w/rudimentary reset
248
249 commit e676f44becd5e6dd449422d014e027b4f064f495
250 Author: David Watkins <djw2146@columbia.edu>
251 Date:   Sat May 7 00:29:51 2016 -0400
252
253 Added change to ispressed
```

```
254
255 commit 25466202fab00ebc180bc529d0163dea458b7b95
256 Author: lpo1234 <lpo1017@frontiernet.net>
257 Date:   Sat May 7 00:27:30 2016 -0400
258
259 Properly addressed enums.svh.
260
261 commit 6e7ee698ebe3ef5588029f59c26d14b95288ba3a
262 Author: lpo1234 <lpo1017@frontiernet.net>
263 Date:   Sat May 7 00:15:20 2016 -0400
264
265 Forgot to push CPU.
266
267 commit cc7eb43fac64d27a259030851aa15aee4e854f10
268 Merge: e7e44fb ad62fb7
269 Author: lpo1234 <lpo1017@frontiernet.net>
270 Date:   Sat May 7 00:13:11 2016 -0400
271
272 Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog
273
274 commit e7e44fb5c1a6e543d3bce3a76a5e4a724b3d13b9
275 Author: lpo1234 <lpo1017@frontiernet.net>
276 Date:   Sat May 7 00:12:43 2016 -0400
277
278 Added stack in. Updated top-level to reflect I/O changes.
    ↪ Prepping for compile.
279
280 commit ad62fb76fe236d96018b981edbd80b1485dcb98e
281 Author: Ashley Kling <ask2203@columbia.edu>
282 Date:   Fri May 6 22:43:38 2016 -0400
283
284 moved some declarations so it works
285
286 commit 965d008c4f4b735c9c4c44989aca249a425112bd
287 Author: Ashley Kling <ask2203@columbia.edu>
288 Date:   Fri May 6 22:20:59 2016 -0400
289
```

```
290 updated stack stuff
291
292 commit 6ef322275e702af5f30edd9aacee10c5e494838f
293 Author: gabriellet <gat2118@columbia.edu>
294 Date:   Fri May 6 21:00:32 2016 -0400
295
296 stack testbench updated, stack passes tests
297
298 commit 858effa51dc45c746a19a737d797821fdc020183
299 Merge: ec6a883 960f694
300 Author: gabriellet <gat2118@columbia.edu>
301 Date:   Fri May 6 20:04:56 2016 -0400
302
303 Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog
304
305 commit ec6a8838916b47857443fde5bcd9191ed1a49f27
306 Author: gabriellet <gat2118@columbia.edu>
307 Date:   Fri May 6 20:04:28 2016 -0400
308
309 Stack testbench update
310
311 commit 960f6947b523d1a2c36a6aef02f214852aa27076
312 Author: David Watkins <djw2146@columbia.edu>
313 Date:   Fri May 6 19:39:13 2016 -0400
314
315 Added changes to makefile
316
317 commit 94932c1591f2e6bdcf5560a5600adb5e254749b6
318 Author: David Watkins <djw2146@columbia.edu>
319 Date:   Fri May 6 14:21:14 2016 -0400
320
321 Added changes to chip8_top
322
323 commit cdcfb9ff9fdecfc322e74b0b71f7db80ea6536eb
324 Author: David Watkins <djw2146@columbia.edu>
325 Date:   Fri May 6 14:17:36 2016 -0400
326
```

```
Added changes to stack to allow for multiple states

commit ff1990098338ff401861a4bd34074ebe104ea68f
Author: David Watkins <djw2146@columbia.edu>
Date:   Fri May 6 13:21:28 2016 -0400

Added compiling version of the driver

commit c7dd59eb78938c38f87a938f33492d28417971eb
Author: David Watkins <djw2146@columbia.edu>
Date:   Fri May 6 03:27:37 2016 -0400

Added skeleton code for new stack ops

commit 5a709b0648979746203101f37ad921df79a42c5a
Author: David Watkins <djw2146@columbia.edu>
Date:   Fri May 6 03:26:41 2016 -0400

Added Stack operations

commit 30e93e3845b365ce2722ca3011e4d71d6925a89c
Merge: c5845b6 4baf65a
Author: David Watkins <djw2146@columbia.edu>
Date:   Thu May 5 02:48:03 2016 -0400

Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog

commit c5845b6ca4bc6a12674a44a4f5ffb29fe53e560a
Author: David Watkins <djw2146@columbia.edu>
Date:   Thu May 5 02:47:24 2016 -0400

Added bash file that can run modelsim from command line

commit 4baf65abbe6e1da8dbe6479f7c47de365c859bd1
Author: lpo2105 <lpo2105@micro6.ilab.columbia.edu>
Date:   Thu May 5 01:43:37 2016 -0400

```

```
Added greater depth to testing approach.

commit 9f67225ddcb4bb9b0aa54204c612d6d1def0cdd8
Author: lpo2105 <lpo2105@micro6.ilab.columbia.edu>
Date:   Thu May 5 00:48:24 2016 -0400

Adding CPU testbench for 6xkk/7xkk. Template to be extended to
    ↪ most CPU cmds.

commit d47c1b3833026dffca2c7fd54dc3783989e3bbc6
Author: David Watkins <djw2146@columbia.edu>
Date:   Tue May 3 17:24:08 2016 -0400

Removed "Hello world"

commit 83bd4ce83c55a21172903bd2cbc6ead0f6e4077a
Author: David Watkins <djw2146@columbia.edu>
Date:   Tue May 3 17:23:08 2016 -0400

All tests pass

commit 761072efe624954cf5cfa0438fc8b8d2c18a632f
Author: David Watkins <djw2146@columbia.edu>
Date:   Tue May 3 16:48:15 2016 -0400

Added testbench update

commit a79a5420cbbaf24d86f84b92bfcf4a2aaf0627c7
Merge: 35dfb69 53d84f5
Author: David Watkins <djw2146@columbia.edu>
Date:   Tue May 3 16:14:01 2016 -0400

Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog

commit 53d84f54d5b3051834b034ad32f0cf4f930347d7
Merge: 22fe90c ed8e36c
Author: Ashley Kling <ask2203@columbia.edu>
```

```
Date:   Tue May 3 15:26:36 2016 -0400

Merge branch 'master' of
 ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog

commit 22fe90c4e442c5e6dee85b7eda89162f04ff6303
Author: Ashley Kling <ask2203@columbia.edu>
Date:   Tue May 3 15:26:03 2016 -0400

cleaned up stack testbench

commit ed8e36c7356e3f099418e2888de466e1d0eac4d9
Merge: 719a978 cf347a1
Author: gabriellet <gat2118@columbia.edu>
Date:   Tue May 3 15:14:54 2016 -0400

Merge branch 'master' of
 ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog

commit 719a978d4fcd00790ce49149ca0e4f075afddfc4
Author: gabriellet <gat2118@columbia.edu>
Date:   Tue May 3 15:14:28 2016 -0400

fb testbench update

commit cf347a195fa9023bc25c12e8f7a241f1549d48cb
Author: Ashley Kling <ask2203@columbia.edu>
Date:   Tue May 3 15:05:38 2016 -0400

Stack working. When reading, output will be available on second
 ↪ clock cycle. When enable = 00, it will read the value at
 ↪ the 0th place in the stack, ignore this.

commit cb46529016f72bf7410f10404df9fedf762b614f
Author: gabriellet <gat2118@columbia.edu>
Date:   Tue May 3 13:39:19 2016 -0400

tested ALU update
```

```
434
435 commit 87816742da290e4f79ade8934a370eed163c8f3d
436 Merge: 5a914f3 36e18d2
437 Author: gabriellet <gat2118@columbia.edu>
438 Date:   Tue May 3 13:37:37 2016 -0400
439
440 Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog
441
442 commit 5a914f300ad9a6a9580cf4ad60c51deba4863ffd
443 Author: gabriellet <gat2118@columbia.edu>
444 Date:   Tue May 3 13:37:15 2016 -0400
445
446 updated tested ALU and enum
447
448 commit 35dfb697a5c5c9c4f7c483846fc381ade5bfb5f0
449 Merge: 4165b3d 36e18d2
450 Author: David Watkins <djw2146@columbia.edu>
451 Date:   Tue May 3 13:33:03 2016 -0400
452
453 Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog
454
455 commit 36e18d283f432890f757638157050543925e3419
456 Author: Gabrielle A Taylor <gat2118@columbia.edu>
457 Date:   Tue May 3 13:26:56 2016 -0400
458
459 Update .gitignore
460
461 commit 1c7e5013d058406d0db3dc80bb155ca6892206ee
462 Author: Gabrielle A Taylor <gat2118@columbia.edu>
463 Date:   Tue May 3 13:22:56 2016 -0400
464
465 Fixed carry in ALU_f_ADD
466
467 commit 95b7b78a1768822c00fd4a3ef263b7bafde03535
468 Author: gabriellet <gat2118@columbia.edu>
469 Date:   Tue May 3 13:11:33 2016 -0400
```

```
470
471 final ALU files - confirmed working
472
473 commit 8275d351c7b51571213c949e39a756ebf58a140f
474 Author: gabriellet <gat2118@columbia.edu>
475 Date:   Sat Apr 30 23:34:40 2016 -0400
476
477 Stack testbench intermediate files:
478
479 commit 4c40781ff01ebe970ed4869a15cacf2832de7037
480 Author: gabriellet <gat2118@columbia.edu>
481 Date:   Sat Apr 30 23:32:19 2016 -0400
482
483 ALU testbench files updated
484
485 commit 7ed35963899dce144ed4292b45bb16eed78d9ce0
486 Author: gabriellet <gat2118@columbia.edu>
487 Date:   Sat Apr 30 17:21:20 2016 -0400
488
489 additional ALU test files
490
491 commit c6faa7efd3dc2d29ffaea21853acd405822b76f7
492 Author: gabriellet <gat2118@columbia.edu>
493 Date:   Sat Apr 30 17:19:56 2016 -0400
494
495 ALU testing underway, still buggy
496
497 commit 2eab7a62d7b455e38b324c723b461ec29e0c6d53
498 Author: gabriellet <gat2118@columbia.edu>
499 Date:   Thu Apr 28 16:06:36 2016 -0400
500
501 new testbench files
502
503 commit e4f8c1ef00d2144481f1f63f74faa7a4c06e643f
504 Author: gabriellet <gat2118@columbia.edu>
505 Date:   Thu Apr 28 15:49:32 2016 -0400
506
507 moved audio files in test to test/audio
```

```
508
509 commit 529cdcfb17e518f53d2c7b88caffa99ce6b3b1e6
510 Author: gabriellet <gat2118@columbia.edu>
511 Date:   Thu Apr 28 15:45:20 2016 -0400
512
513 push stack ram to test/stk
514
515 commit e19c5f02d734e5572249e492e79890ea1f8c978c
516 Author: gabriellet <gat2118@columbia.edu>
517 Date:   Thu Apr 28 13:32:43 2016 -0400
518
519 stack testbench update
520
521 commit e5998b9b50fadeecafb85a5abd9697b598b21bb2
522 Merge: 9a46db3 0b17dc7
523 Author: gabriellet <gat2118@columbia.edu>
524 Date:   Thu Apr 28 13:14:19 2016 -0400
525
526 Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog
527 Stack updated, merging to testbench
528
529 commit 9a46db322947f85a19d149a3fc9fdcfff0dbaaa6
530 Author: gabriellet <gat2118@columbia.edu>
531 Date:   Thu Apr 28 13:13:56 2016 -0400
532
533 stack_testbench update
534
535 commit 0b17dc7a2c1ce76f303790bab864614746189dba
536 Merge: 3ac5cff 6ddb751
537 Author: Ashley Kling <ask2203@columbia.edu>
538 Date:   Thu Apr 28 13:04:17 2016 -0400
539
540 Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog
541
542 commit 3ac5cfffa9b0ca446599d195381851d777b5cf4f
543 Author: Ashley Kling <ask2203@columbia.edu>
```

```
Date:   Thu Apr 28 13:04:04 2016 -0400

updated stack

commit 6ddb751b3ba98f4cf5a06c6f36ae5ccd3704771a
Author: gabriellet <gat2118@columbia.edu>
Date:   Thu Apr 28 12:51:14 2016 -0400

Stack testbench files

commit 889fbe39b2d2a638312fe15fdb16b5a94c2414ef
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Thu Apr 28 00:09:17 2016 -0400

Revamped framebuffer and CPU to match. Theoretically supports
    ↪ draw sprite and clear screen cmds. CHANGES NOT REFLECTED IN
    ↪ Chip8_Top

commit 18e996d09a296988bfc56018fc85abce34341a2a
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Wed Apr 27 18:17:44 2016 -0400

Adding MegaFunction Framebuffer memory.

commit 2913ad5cfc62beb48050c2705d375b981103d3d3
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Tue Apr 26 21:55:28 2016 -0400

Fixed CPU casex-ladder syntax problems for DC conditions.
    ↪ Cleaned up truncation warnings. Tested
    ↪ triple-ported-register-file. It seems to work.

commit 4165b3d491665156e77feb45618226961df4cdff
Author: David Watkins <djw2146@columbia.edu>
Date:   Tue Apr 26 14:52:57 2016 -0400

Added a python script for generating properly formatted MIF
    ↪ files
```

```
577
578 commit 5e6485faef80c816c1ad30c4b84635552b2e4b04
579 Author: David Watkins <djw2146@columbia.edu>
580 Date:   Tue Apr 26 08:07:17 2016 -0400
581
582 Updated code to support instructions at top level
583
584 commit dfd8770b0b2b77dca4b54fcedbc0f806b6e0ac87
585 Author: David Watkins <djw2146@columbia.edu>
586 Date:   Tue Apr 26 06:57:58 2016 -0400
587
588 Compiling version of entire project
589
590 commit 932855dd20090b7ad68be05b9c9009ae2d762ec2
591 Author: David Watkins <djw2146@columbia.edu>
592 Date:   Tue Apr 26 04:14:34 2016 -0400
593
594 Added updates to the sound controller
595
596 commit aa6f9247e0d2766b790dfafb3c8520e32ca3081b
597 Author: David Watkins <djw2146@columbia.edu>
598 Date:   Tue Apr 26 03:46:59 2016 -0400
599
600 Added header files
601
602 commit 7f698e00b0b09c2348541596c9b71568eb5bf98e
603 Author: David Watkins <djw2146@columbia.edu>
604 Date:   Tue Apr 26 03:38:25 2016 -0400
605
606 Fixed ALU and CPU to support all instruction (Draw NYI)
607
608 commit 2a1286c317fd71e37bfe17bb83903c56bb340d27
609 Author: lpo1234 <lpo1017@frontiernet.net>
610 Date:   Mon Apr 25 01:34:59 2016 -0400
611
612 Added most PC functionality to the CPU.
613
614 commit 0ddddfb9e4ef23b12aa804610d08ce9819a9ddde
```

```
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Sun Apr 24 23:55:18 2016 -0400

Added BCD functionality to CPU. Fixed oversized default value
 ↪  in random num generator.

commit b58aa9ef625b711d4180fedc31c5ce193eccb036
Merge: 9cbc6d7 3de4899
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Sun Apr 24 23:40:36 2016 -0400

Merge branch 'master' of
 ↪  https://github.com/DavidWatkins/Chip8-SystemVerilog

commit 9cbc6d726f251f8ef79a779751bd8a60dc902c99
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Sun Apr 24 23:39:50 2016 -0400

Updated CPU for top-level control. It does not deal with instrs
 ↪  regarding PC, framebuffer, or BCD.

commit 3de48994fb16a1fa8d2e4952da22f1527fe5153b
Author: ask2203 <ask2203@micro15.ilab.columbia.edu>
Date:   Sat Apr 23 18:13:12 2016 -0400

untested stack file

commit b172666ea7b2de978e5c5096882a788f4a0ea9a6
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date:   Fri Apr 22 18:22:09 2016 -0400

Duplicated files deleted

commit 2d0c61a7e144c8759127d99af262cbd7c1140d64
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date:   Fri Apr 22 18:21:28 2016 -0400

Duplicated files deleted
```

```
650
651 commit 34e241302a4ef0977bbdbe2e5394f0dc8f6556b6
652 Author: Gabrielle A Taylor <gat2118@columbia.edu>
653 Date:   Fri Apr 22 18:21:08 2016 -0400
654
655 Duplicated files deleted
656
657 commit f0a1dc7c772ed001e2073d755dd7f1d67ce37547
658 Author: Gabrielle A Taylor <gat2118@columbia.edu>
659 Date:   Fri Apr 22 18:20:58 2016 -0400
660
661 Duplicated files deleted
662
663 commit d56f6ab9c26e0d9fd880f727d6be6fbbec44be6e
664 Author: Gabrielle A Taylor <gat2118@columbia.edu>
665 Date:   Fri Apr 22 18:20:47 2016 -0400
666
667 Duplicated files deleted
668
669 commit 92890f725928acce4d7755c0784d69cca85062ac
670 Author: Gabrielle A Taylor <gat2118@columbia.edu>
671 Date:   Fri Apr 22 18:20:34 2016 -0400
672
673 Duplicated files deleted
674
675 commit be03d04f33b42f09ee3dfb45c2b5cd0cf547df4e
676 Author: Gabrielle A Taylor <gat2118@columbia.edu>
677 Date:   Fri Apr 22 18:17:14 2016 -0400
678
679 Script + samples + original wav
680
681 Script extracts samples from wav file. Samples for beep2.wav
    ↪  are in samples.txt
682
683 commit b705539fb6021353ac2ddb476142288cf3f267a1
684 Author: David Watkins <djw2146@columbia.edu>
685 Date:   Tue Apr 19 15:51:50 2016 -0400
686
```

```
687 Added files that wer emissing from before
688
689 commit 512823e2a46d09a3d8305699f60615b0a0988908
690 Author: David Watkins <djw2146@columbia.edu>
691 Date:   Sat Apr 16 18:11:21 2016 -0400
692
693 Moved testbench for delay timer
694
695 commit 844880913d18dee6e4c4cee38dd8a12c012a413f
696 Author: David Watkins <djw2146@columbia.edu>
697 Date:   Sat Apr 16 18:08:35 2016 -0400
698
699 Redid file directory
700
701 commit f55eb911738c27765f9c0a8f8158ba6393cd541b
702 Author: gabriellet <gat2118@columbia.edu>
703 Date:   Sat Apr 16 07:24:54 2016 -0400
704
705 removed mips directory from test, added delay timer and
    ↪ testbench to test
706
707 commit ef7706c9c01f65935ec1f8d8ba75900d9821b824
708 Merge: 6e306a2 fcb610e
709 Author: gabriellet <gat2118@columbia.edu>
710 Date:   Fri Apr 15 12:22:42 2016 -0400
711
712 Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog
713
714 commit 6e306a29a061cb4612c4e1c4c6160b731e7ec84a
715 Author: gabriellet <gat2118@columbia.edu>
716 Date:   Fri Apr 15 12:21:58 2016 -0400
717
718 delay timer implementation
719
720 commit fcb610e9bfb6face733099142e3a1d275106f871
721 Author: lpo1234 <lpo1017@frontiernet.net>
722 Date:   Wed Apr 13 23:50:14 2016 -0400
```

```
723
724 Here is the test waveform to show on Thursday Apr 14. I am the
    ↪ prettiest.
725
726 commit fa449ed041a88e713c1f133beb3033e2a5f48021
727 Author: David Watkins <djw2146@columbia.edu>
728 Date:   Wed Apr 13 23:02:56 2016 -0400
729
730 Forgot test
731
732 commit 050daf1090a867f8181af4f1db6bbaad9c6054f6
733 Merge: c37a15a ac284ac
734 Author: David Watkins <djw2146@columbia.edu>
735 Date:   Wed Apr 13 22:46:39 2016 -0400
736
737 Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog
738
739 commit c37a15a2e2f3ffd0e0ccdb502336571f20f50000
740 Author: David Watkins <djw2146@columbia.edu>
741 Date:   Wed Apr 13 22:46:23 2016 -0400
742
743 Cries
744
745 commit ac284ac58a27087d4766eb4ccd0061ee9e555917
746 Author: lpo1234 <lpo1017@frontiernet.net>
747 Date:   Wed Apr 13 22:40:15 2016 -0400
748
749 Added dual ported memory for register file and memory.
750
751 commit 07694cbe2ac21b44c806dc56c466470b44dfdc02
752 Author: David Watkins <djw2146@columbia.edu>
753 Date:   Wed Apr 13 22:30:14 2016 -0400
754
755 Added testbench
756
757 commit ade90c0e38478bbeb1ccd93f255fce8834adf321
758 Author: David Watkins <djw2146@columbia.edu>
```

```
Date:   Wed Apr 13 22:03:12 2016 -0400

Added test rom initialization file that tests basic
    ↪ instructions

commit 0e31d27cf7e437b01e5e055bb7d90e229c0de5db
Author: David Watkins <djw2146@columbia.edu>
Date:   Wed Apr 13 21:56:58 2016 -0400

Added a new top level module that runs over the pc

commit 2133d107428c3ad7ad74065bca527d4e7fb1f2a4
Merge: 77d791d 1443ecb
Author: David Watkins <djw2146@columbia.edu>
Date:   Wed Apr 13 21:31:27 2016 -0400

Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog

commit 77d791dc6a4f63a5a0ea4fecd13c111f1d2d1700
Author: David Watkins <djw2146@columbia.edu>
Date:   Wed Apr 13 21:31:10 2016 -0400

Added in progress version of chip8 top level

commit 1443ecb60c1d37b41f097d38155e532bc3c2d5c0
Author: Ashley Kling <ask2203@columbia.edu>
Date:   Wed Apr 13 20:42:55 2016 -0400

fixed dec-hex

commit 4ffe66bcffed6e9a54b3c44b7111ea0428327a8f
Author: David Watkins <DavidWatkins@users.noreply.github.com>
Date:   Wed Apr 13 19:02:48 2016 -0400

Delete SoCKit_Top.ipinfo

commit dbbdc521e65452464f9d20f66fe9b73828dac53a
```

```
Author: David Watkins <DavidWatkins@users.noreply.github.com>
Date:   Wed Apr 13 19:02:43 2016 -0400

Delete SoCKit_Top.sld_design_entry.sci

commit 4c5ce1ca7dd6541308ef7f923505f80bb075035f
Author: David Watkins <DavidWatkins@users.noreply.github.com>
Date:   Wed Apr 13 19:02:38 2016 -0400

Delete SoCKit_Top.db_info

commit 966343eeb5429e8624554017bb7ac7ba6e62a27c
Author: gat2118 <gat2118@micro18.ilab.columbia.edu>
Date:   Wed Apr 13 16:37:06 2016 -0400

copied files to test in order to implement MIPS-like processor
    ↪ design

commit 6c913287ef3c20a539423cbe0cd6715d7f0a2923
Merge: 8048a4a a67a547
Author: Ashley Kling <ask2203@columbia.edu>
Date:   Tue Apr 12 14:44:31 2016 -0400

Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog

commit a67a547f37b38d65657e96f8174ba7acf63dfdef
Merge: af3718e 9761ee9
Author: David Watkins <djw2146@columbia.edu>
Date:   Tue Apr 12 14:16:55 2016 -0400

Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog

commit af3718ee4f5a685290f46ec466e8562c2d9711ed
Author: David Watkins <djw2146@columbia.edu>
Date:   Tue Apr 12 14:16:27 2016 -0400

```

```
Added register file code

commit 8048a4a57a22b65b5a41c11d1851bbe225e8145a
Author: Ashley Kling <ask2203@columbia.edu>
Date:   Tue Apr 12 13:57:34 2016 -0400

updated CPU file, need to test, esp # cycles per instruction

commit 9761ee95f0496f077c326bee4aba1825aeedd73f
Author: David Watkins <DavidWatkins@users.noreply.github.com>
Date:   Tue Apr 12 13:48:02 2016 -0400

Update README.md

commit 5cd366cfed3ea15b0c9961b5ca37ee9bf98cb840
Author: David Watkins <DavidWatkins@users.noreply.github.com>
Date:   Tue Apr 12 12:19:30 2016 -0400

Update README.md

commit 553cfe6250d772c230278d92c4ec80d0fc230a93
Author: David Watkins <DavidWatkins@users.noreply.github.com>
Date:   Tue Apr 12 12:19:05 2016 -0400

Update README.md

commit 02e3c4b657d98c03c53c3f1d7d5bc501ad851dc4
Author: David Watkins <djw2146@columbia.edu>
Date:   Tue Apr 12 16:56:18 2016 +0100

dded u-boot script which is necessary for getting the sockit
↪  board to boot

commit dd7d50a9e71b0c4f1623d25148c4d96b4790e306
Merge: b15ad94 f9f3232
Author: David Watkins <djw2146@columbia.edu>
Date:   Tue Apr 12 03:58:11 2016 -0400

```

```
Merge branch 'master' of
    ↪ https://github.com/DavidWatkins/Chip8-SystemVerilog

commit b15ad94df32fd92cbf140653c3f5b44649d32daa
Author: David Watkins <djw2146@columbia.edu>
Date:   Tue Apr 12 03:57:31 2016 -0400

Added communication back and forth. Needs testing

commit f9f3232503cc2a16efe751c24054502fc8ed20c8
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date:   Mon Apr 11 14:29:13 2016 -0400

Update sockit_top.sv

commit 5ca73ee5a326b6b758c9046ed2cab6b02ed9aa6c
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date:   Mon Apr 11 14:15:56 2016 -0400

Changed implementation

Clock is now default low (no edge detection necessary for use
    ↪ in delay, sound timers).

commit 99ac7117f1e962f780f1ff26d8026510292eec02
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date:   Mon Apr 11 12:55:44 2016 -0400

clk_div.sv works

Seems to work in simulation. Will try a few more tests with
    ↪ edge cases to determine behavior.

commit 75aa22faed1b3fabc1f3439d8df2defcded46e95
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date:   Mon Apr 11 12:28:07 2016 -0400

bcd.sv now works
```

```
902
903 Tests indicate module works.
904
905 commit a744ba2ec0ca570e06cecc2770aedd176d044d2d
906 Author: Gabrielle A Taylor <gat2118@columbia.edu>
907 Date:   Sun Apr 10 13:30:36 2016 -0400
908
909 Fix compilation errors
910
911 Simulated in modelsim but still does not work. Loop in line 13
    ↪  only runs once. Will investigate why.
912
913 commit fd6b8e93cce11ec12061d7a328f912fbe85f1cad
914 Author: lpo1234 <lpo1017@frontiernet.net>
915 Date:   Sun Apr 10 03:32:07 2016 -0400
916
917 Adding memory module waveform test.
918
919 commit 4b5e6b9b12c7e6706af463a5324b2b65ba1626cb
920 Author: lpo1234 <lpo1017@frontiernet.net>
921 Date:   Sun Apr 10 03:18:46 2016 -0400
922
923 Added 4096byte memory module.
924
925 commit 90fd2f67f770b0eb25a8930198fbd8b05b8afa88
926 Author: Gabrielle A Taylor <gat2118@columbia.edu>
927 Date:   Sat Apr 9 22:52:47 2016 -0400
928
929 BCD module in systemverilog
930
931 commit 6de8348fd274c49e7564113fcf631f53b4cbfc32
932 Author: Gabrielle A Taylor <gat2118@columbia.edu>
933 Date:   Sat Apr 9 22:24:38 2016 -0400
934
935 Information relevant to audio test
936
937 commit df18feae1c60ffe03b53e1bf4117c52852969b2b
938 Author: Gabrielle A Taylor <gat2118@columbia.edu>
```

```
Date:   Sat Apr 9 22:19:38 2016 -0400

Rename sockit_top.sv to test/sockit_top.sv

commit 08b5170a0a56c9c93b32fbd4386313c09af62bc2
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date:   Sat Apr 9 22:19:18 2016 -0400

Rename i2c_controller.sv to test/i2c_controller.sv

commit f8d101603af378f5a6d530a1a924ba3a05a383e9
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date:   Sat Apr 9 22:19:06 2016 -0400

Rename i2c_av_config.sv to test/i2c_av_config.sv

commit 2566be1a0a3a9cc017a8ba3829e0894e12fbd3ff
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date:   Sat Apr 9 22:18:50 2016 -0400

Rename delay_timer.sv to test/delay_timer.sv

commit 7a3c6f0788411ac69e24c6680e1104ed22a31167
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date:   Sat Apr 9 22:18:37 2016 -0400

Rename audio_effects.sv to test/audio_effects.sv

commit e4ebe895dc5205aaf655a51849df03bb5f08268c
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date:   Sat Apr 9 22:18:17 2016 -0400

Rename audio_codec.sv to test/audio_codec.sv

commit 070b16f32c9820b0c0d15aca5b321020e9d181fa
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date:   Sat Apr 9 22:17:52 2016 -0400

```

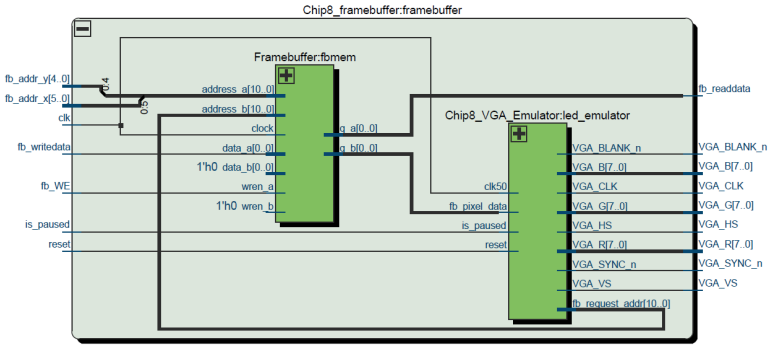
```
Test for SoCKit Sound

Tests whether SoCKit board will output 440Hz sine wave

commit 1d325b6e8e083bfcd199f182fa298fa31fde876d
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date:   Sat Apr 9 22:12:39 2016 -0400

Clock divider

Converts from 50MHz clock to 60Hz clock. To be used in delay
 ↪  and sound timers.

commit 51f1409a2ae329f78aa28986d65e4fae7b306df0
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Thu Apr 7 13:51:30 2016 -0400

ACTUALLY added initial CPU vwf test.

commit 174d5f5228965d9067ab7cd326f82fb7ca7ba5d0
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Thu Apr 7 13:49:56 2016 -0400

Comb instr decode works. Cleaned code. Adding CPU vwf test.

commit 260a661a7c625be088986997ee5c417edda4ca01
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Thu Apr 7 13:29:00 2016 -0400

Made instruction decode combinational. I believe it works, but
 ↪  not all instrs implemented were tested.

commit 0da757bd8e26d569a3ee7682fbac164fe97339b8
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Tue Apr 5 23:58:29 2016 -0400

Started CPU. Data is not appearing as expected. See
 ↪  CPU_initial_test.vwf waveform.
```

```
commit 146d5c3b0dcc7412d0853dc1258e1d5ab239a3a2
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Tue Apr 5 21:07:43 2016 -0400

Created register module to control V0-VF. Also created a folder
 ↪  to hold test waveforms.

commit 904fba10987ee571bae75211cf3992935e10ec45
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Thu Mar 31 02:54:44 2016 -0400

Added 16b random number generator.

commit e3410e4fb4c763867bdb6e503751ed155e06830c
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Thu Mar 31 00:07:47 2016 -0400

Added ALU and memory-to-screen VGA emulator

commit 9b629ea8e3d0dde730a90f39806208222fde5965
Author: David Watkins <djw2146@columbia.edu>
Date:   Wed Mar 30 16:12:46 2016 -0400

Fixed key press values

commit 6520961eb44e7fb8d902c80adb6ce8c38807276c
Merge: ef35672 ad878f1
Author: David Watkins <djw2146@columbia.edu>
Date:   Wed Mar 30 04:03:29 2016 -0400

Merge branch 'master' of
 ↪  https://github.com/DavidWatkins/Chip8-SystemVerilog

commit ef35672328ff42b0b4fcde7d7391555aaa3574da
Author: David Watkins <djw2146@columbia.edu>
Date:   Wed Mar 30 04:02:55 2016 -0400
```

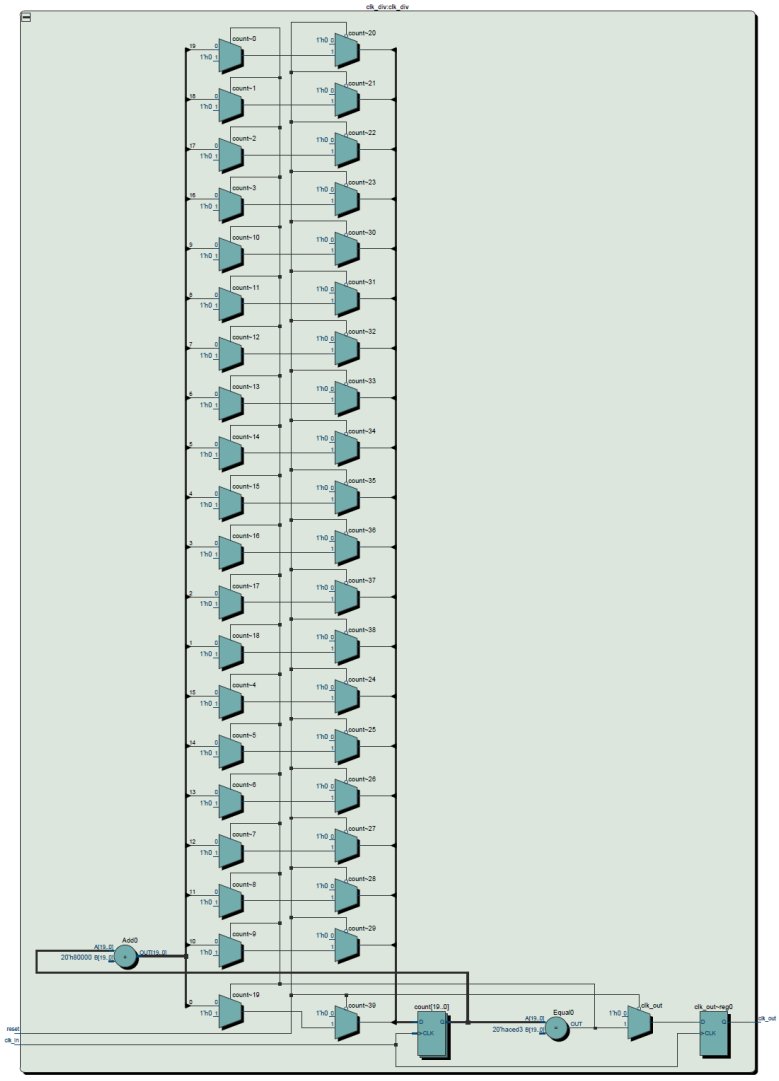
```
New status for stuff. Added a bunch of local stuff

commit ad878f13b861ec9b4084a9eaf8d94309be5f1590
Author: David Watkins <DavidWatkins@users.noreply.github.com>
Date:   Tue Mar 29 23:50:36 2016 -0400

Update Chip8_VGA_Emulator.sv

commit 7b43eb8ec84468304dbae5eba2a739278eb2fc16
Author: David Watkins <djw2146@columbia.edu>
Date:   Tue Mar 29 23:34:08 2016 -0400

Added changes to framebuffer code

commit cab0f99bfc540cd1a64a27a29ba18d5c68818b32
Author: David Watkins <djw2146@columbia.edu>
Date:   Mon Mar 28 23:34:20 2016 -0400

Initial commit

commit 7e436e72239b592fb76c50e6d126497341210cea
Author: David Watkins <DavidWatkins@users.noreply.github.com>
Date:   Mon Mar 28 17:23:05 2016 -0400

Initial commit
```

7.5 Schematics

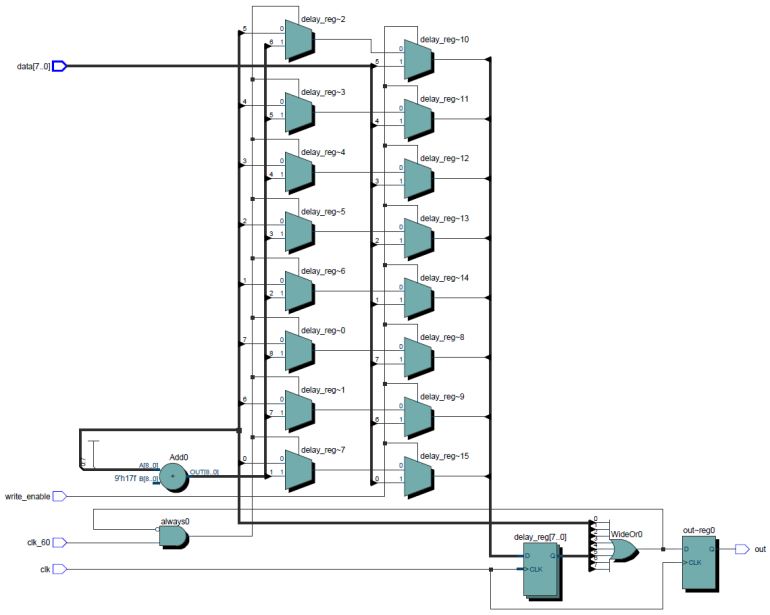
7.5.1 Chip8_framebuffer.sv



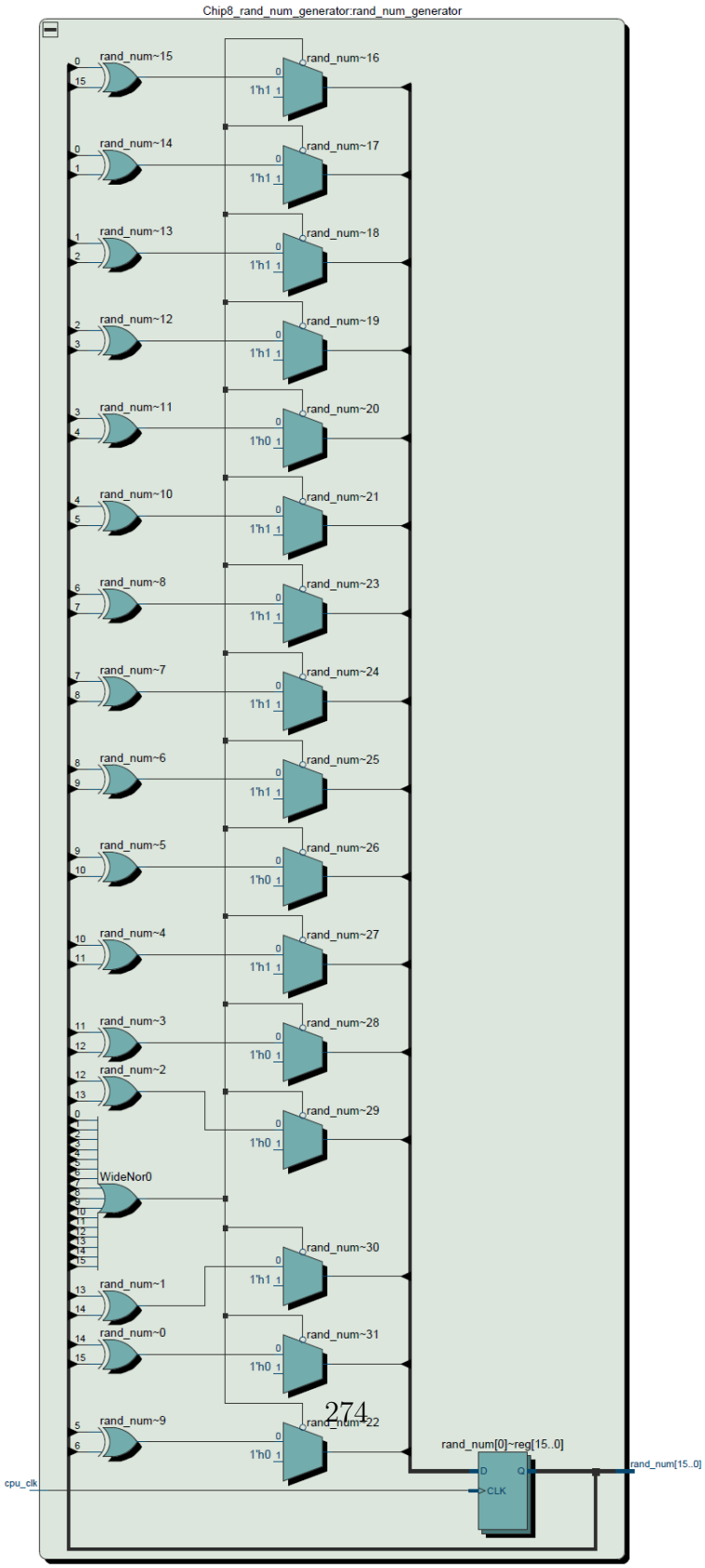
7.5.2 clk_div.sv


clk_div.clk_div
count-0
count-1
count-2
count-3
count-10
count-11
count-12
count-13
count-14
count-15
count-16
count-17
count-18
count-4
count-5
count-6
count-7
count-8
count-9
count-19
count-20
count-21
count-22
count-23
count-30
count-31
count-32
count-33
count-34
count-35
count-36
count-37
count-38
count-24
count-25
count-26
count-27
count-28
count-29
count-39
Add0
Equal0
count[19..0]
clk_out
clk_out-reg0
reset
clk_in

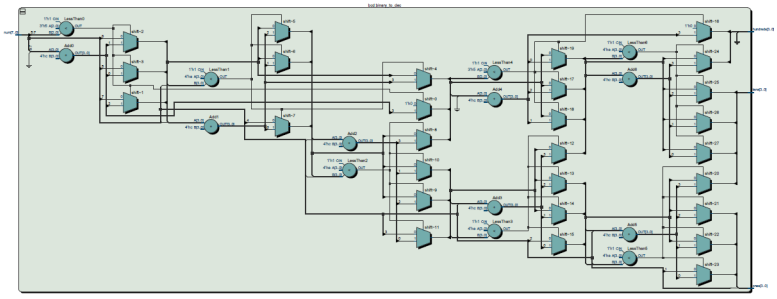
7.5.3 timer.sv



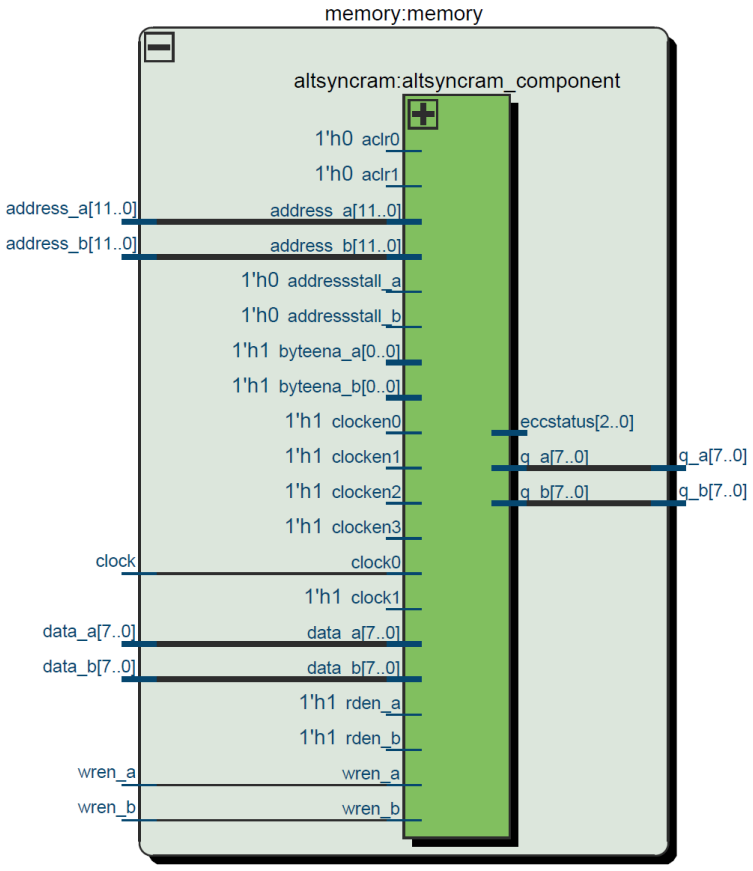
7.5.4 Chip8_rand_num_generator.sv


Chip8_rand_num_generator:rand_num_generator
rand_num~15
rand_num~16
rand_num~14
rand_num~17
rand_num~13
rand_num~18
rand_num~12
rand_num~19
rand_num~11
rand_num~20
rand_num~10
rand_num~21
rand_num~8
rand_num~23
rand_num~7
rand_num~24
rand_num~6
rand_num~25
rand_num~5
rand_num~26
rand_num~4
rand_num~27
rand_num~3
rand_num~28
rand_num~2
rand_num~29
WideNor0
rand_num~30
rand_num~1
rand_num~0
rand_num~31
rand_num~9
rand_num~22
1'h1
1'h0
rand_num[0]~reg[15..0]
D
Q
CLK
cpu_clk
rand_num[15..0]

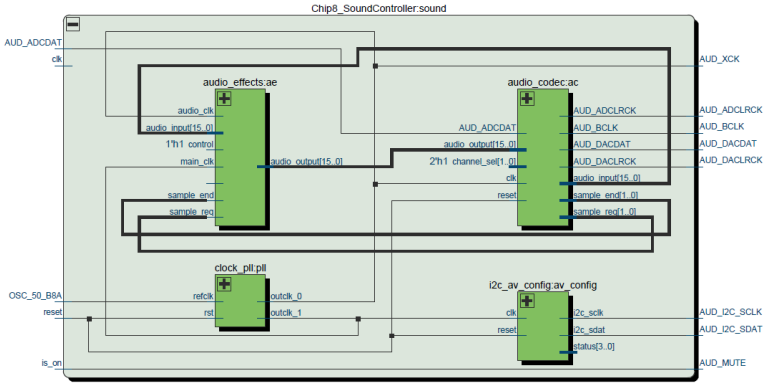
7.5.5 bcd.sv



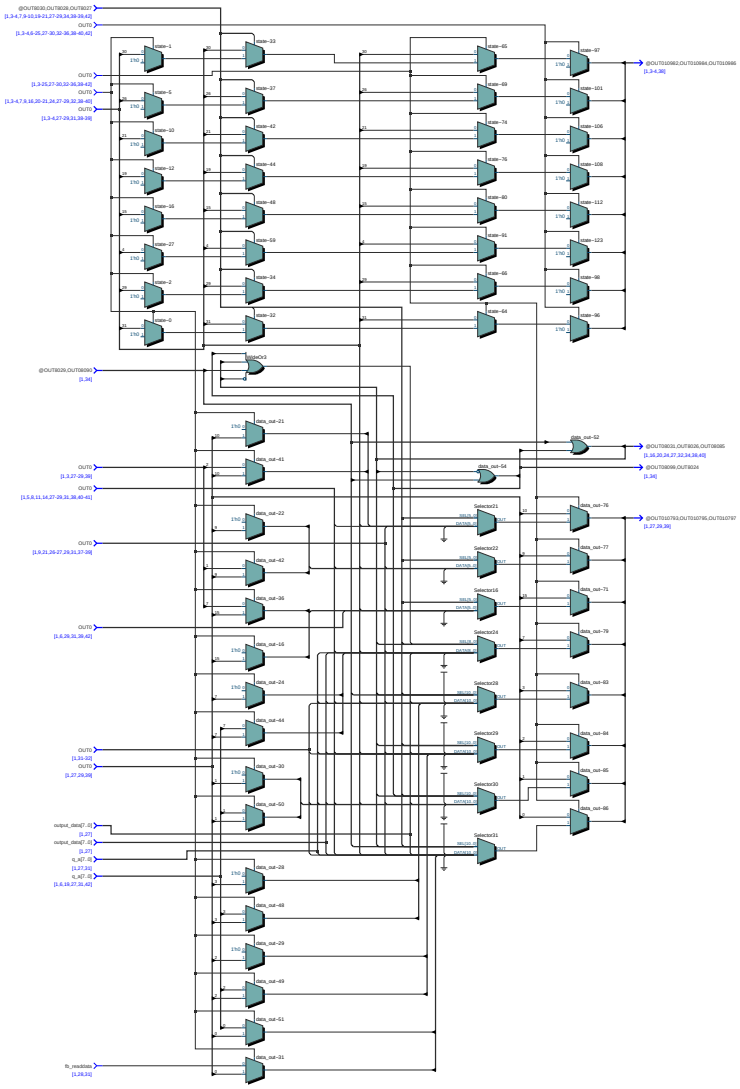
7.5.6 memory.sv

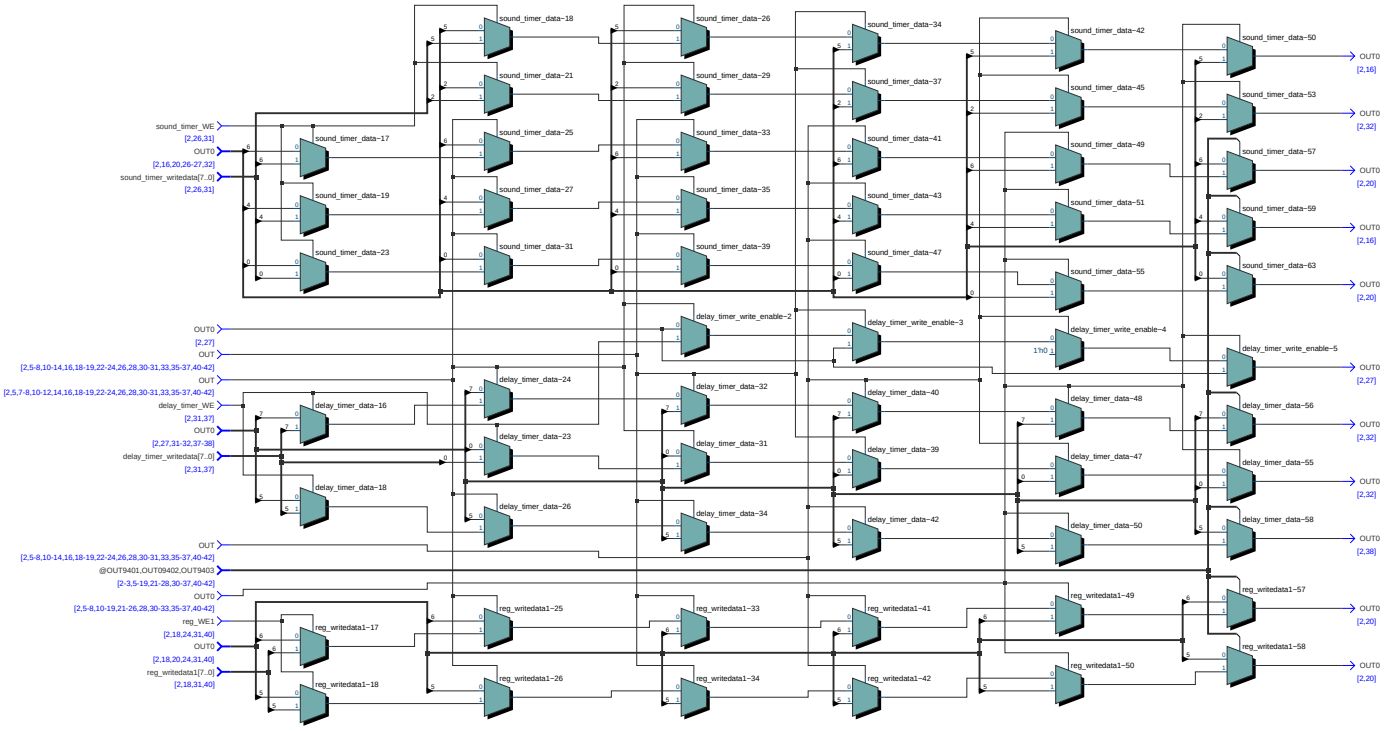


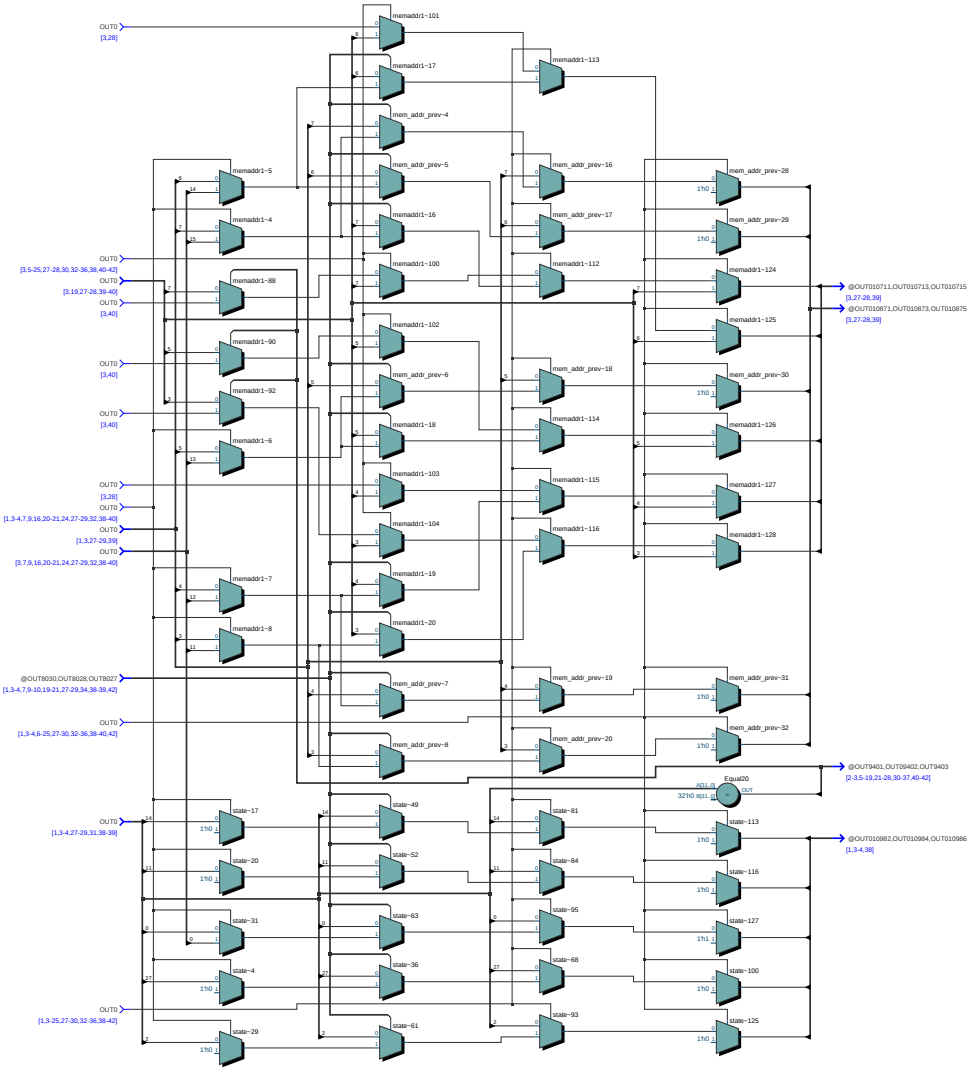
7.5.7 Chip8_SoundController.sv

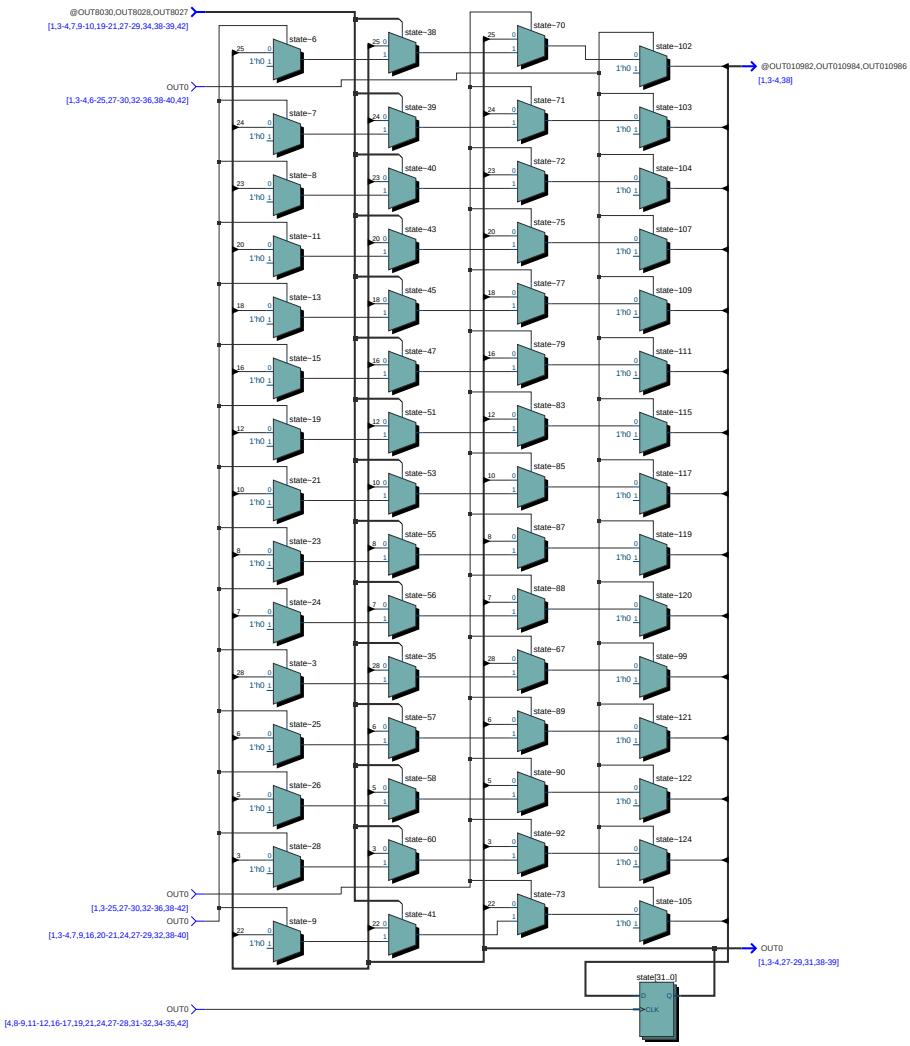


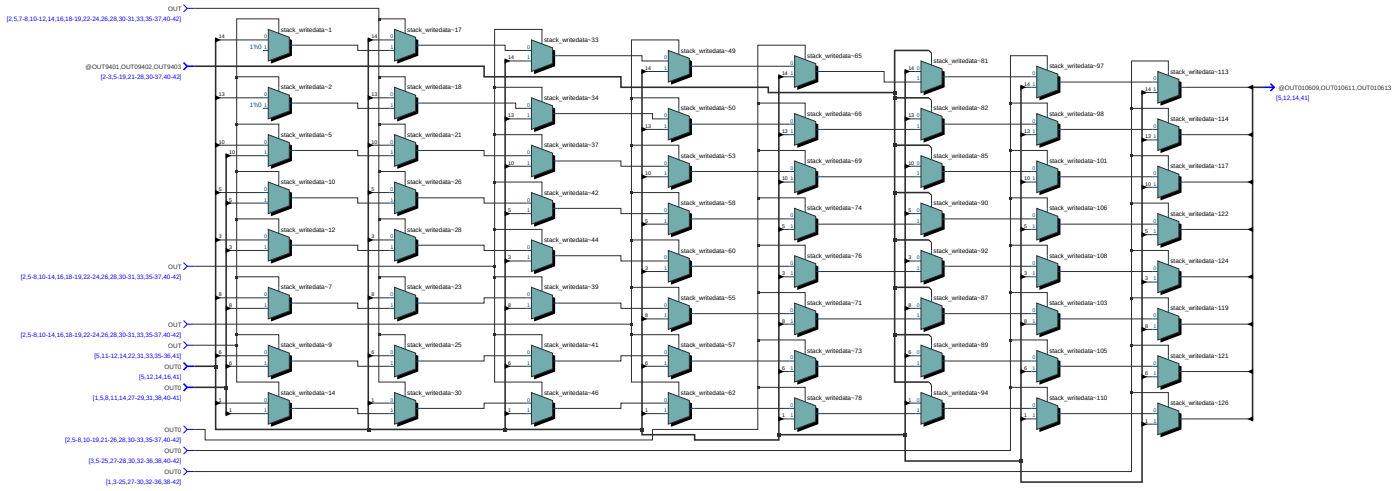
7.5.8 Entire Design

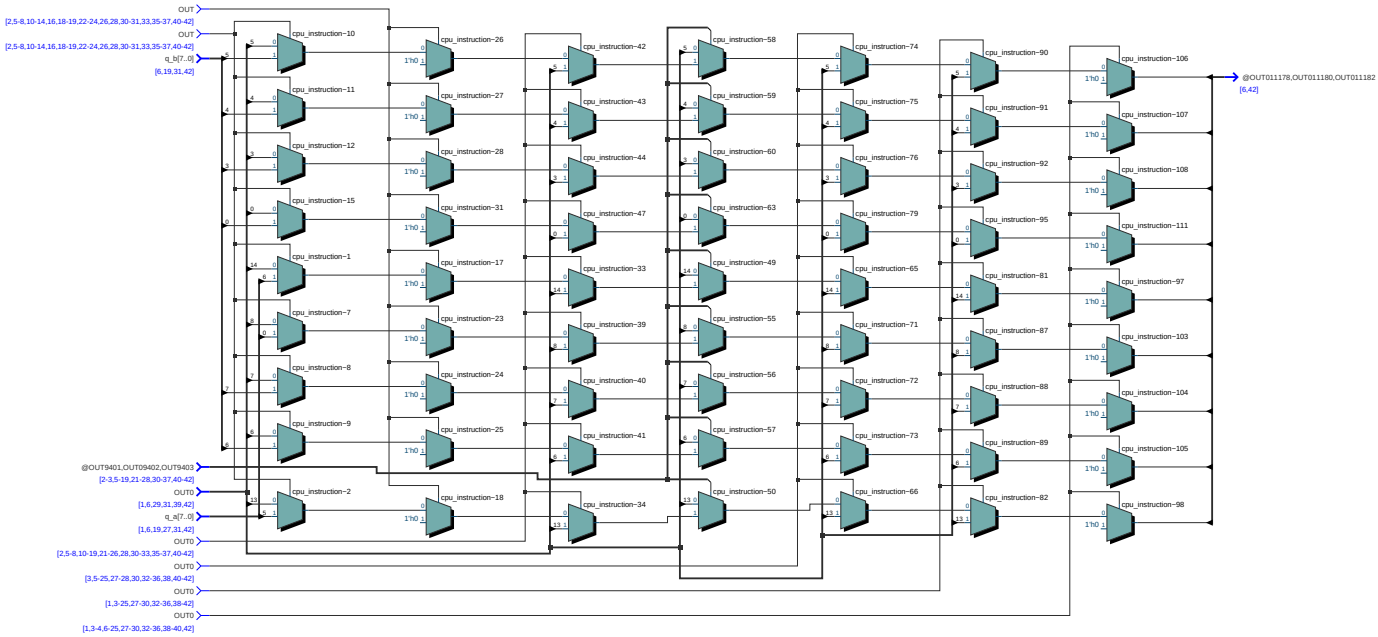

@OUT08030,OUT08026,OUT08027
[1,3-4,7,9-10,19-21,27-29,34,39-39,43]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT0
[1,3-25,27-30,32-36,39-42]
OUT0
[1,3-4,7,9,16,20-21,24,27-29,32,38-40]
OUT0
[1,3-4,27-29,31,39-39]
state--1
state--33
state--5
state--37
state--10
state--42
state--12
state--44
state--16
state--46
state--27
state--59
state--2
state--34
state--0
state--32
@OUT08029,OUT08000
[1,34]
data_out--21
data_out--41
OUT0
[1,3,27-29,39]
OUT0
[1,5,9,11,14,27-29,31,38,40-41]
data_out--22
data_out--42
OUT0
[1,9,21,26-27,29,31,37-39]
data_out--36
data_out--16
OUT0
[1,6,29,31,39,42]
data_out--24
data_out--44
OUT0
[1,31-33]
OUT0
[1,27,29,39]
data_out--30
data_out--50
output_data[7..0]
[1,27]
output_data[7..0]
[1,27]
o_a[7..0]
[1,27,31]
o_a[7..0]
[1,6,18,27,31,40]
data_out--28
data_out--48
data_out--29
data_out--49
data_out--51
data_out--31
fb_readdata
[1,28,31]
state--65
state--97
@OUT07100982,OUT0100984,OUT0100986
[1,3-4,36]
state--69
state--101
state--74
state--106
state--76
state--109
state--80
state--112
state--91
state--123
state--66
state--98
state--64
state--96
data_out--52
@OUT08031,OUT08026,OUT08085
[1,10,20,24,27,32,34,38,43]
data_out--54
@OUT08099,OUT08024
[1,34]
Selector21
Selector22
Selector16
Selector24
Selector28
Selector29
Selector30
Selector31
data_out--76
@OUT0100793,OUT0100795,OUT0100797
[1,27,29,39]
data_out--77
data_out--71
data_out--79
data_out--83
data_out--84
data_out--85
data_out--86

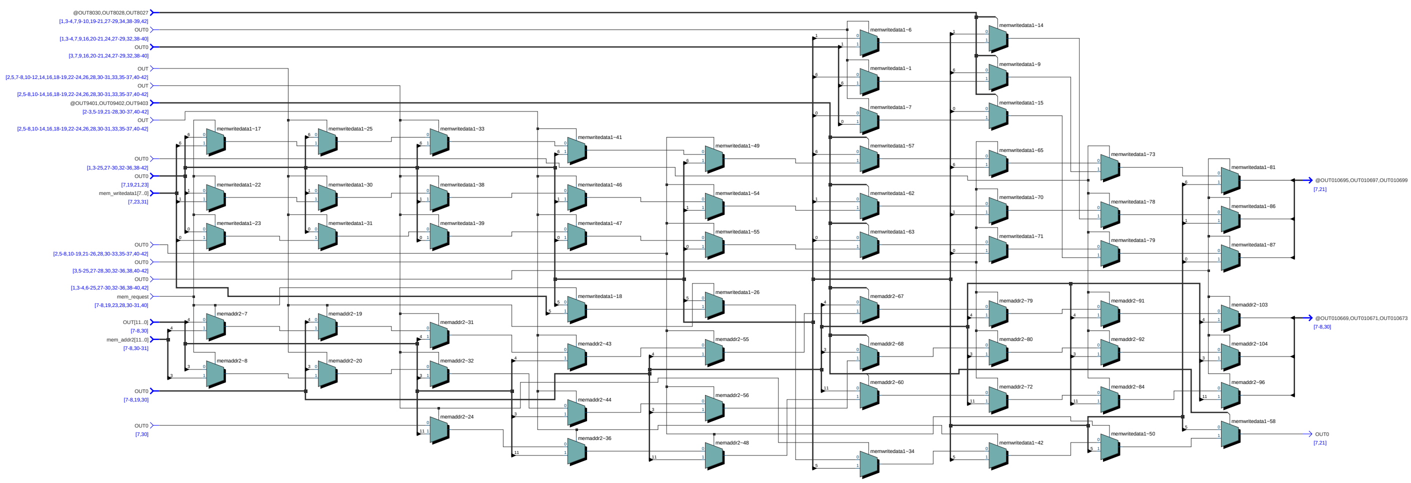

sound_timer_WE
[2,26,31]
OUT0
[2,16,20,26-27,32]
sound_timer_writedata[7..0]
[2,26,31]
sound_timer_data~17
sound_timer_data~18
sound_timer_data~19
sound_timer_data~21
sound_timer_data~23
sound_timer_data~25
sound_timer_data~27
sound_timer_data~31
sound_timer_data~26
sound_timer_data~29
sound_timer_data~33
sound_timer_data~35
sound_timer_data~39
sound_timer_data~34
sound_timer_data~37
sound_timer_data~41
sound_timer_data~43
sound_timer_data~47
sound_timer_data~42
sound_timer_data~45
sound_timer_data~49
sound_timer_data~51
sound_timer_data~55
sound_timer_data~50
sound_timer_data~53
sound_timer_data~57
sound_timer_data~59
sound_timer_data~63
OUT0
[2,16]
OUT0
[2,32]
OUT0
[2,20]
OUT0
[2,16]
OUT0
[2,20]
OUT0
[2,27,31-32,37-38]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5,7-8,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
delay_timer_WE
[2,31,37]
OUT0
[2,27,31-32,37-38]
delay_timer_writedata[7..0]
[2,31,37]
delay_timer_data~16
delay_timer_data~18
delay_timer_data~24
delay_timer_data~23
delay_timer_data~26
delay_timer_write_enable~2
delay_timer_data~32
delay_timer_data~31
delay_timer_data~34
delay_timer_write_enable~3
delay_timer_data~40
delay_timer_data~39
delay_timer_data~42
delay_timer_write_enable~4
1'h0
delay_timer_data~48
delay_timer_data~47
delay_timer_data~50
delay_timer_write_enable~5
delay_timer_data~56
delay_timer_data~55
delay_timer_data~58
OUT0
[2,27]
OUT0
[2,32]
OUT0
[2,32]
OUT0
[2,38]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
reg_WE1
[2,18,24,31,40]
OUT0
[2,18,20,24,31,40]
reg_writedata1[7..0]
[2,18,31,40]
reg_writedata1~17
reg_writedata1~18
reg_writedata1~25
reg_writedata1~26
reg_writedata1~33
reg_writedata1~34
reg_writedata1~41
reg_writedata1~42
reg_writedata1~49
reg_writedata1~50
reg_writedata1~57
reg_writedata1~58
OUT0
[2,20]
OUT0
[2,20]


OUT0
[3,28]
memaddr1~101
memaddr1~17
memaddr1~113
mem_addr_prev~4
mem_addr_prev~5
mem_addr_prev~16
mem_addr_prev~28
memaddr1~5
memaddr1~16
mem_addr_prev~17
mem_addr_prev~29
memaddr1~4
memaddr1~100
memaddr1~112
memaddr1~124
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
memaddr1~88
@OUT10711,OUT10713,OUT10715
[3,27-28,39]
OUT0
[3,19,27-28,39-40]
@OUT10871,OUT10873,OUT10875
[3,27-28,39]
OUT0
[3,40]
memaddr1~102
memaddr1~125
memaddr1~90
mem_addr_prev~18
mem_addr_prev~30
OUT0
[3,40]
mem_addr_prev~6
memaddr1~92
memaddr1~114
memaddr1~126
OUT0
[3,40]
memaddr1~18
memaddr1~6
memaddr1~103
memaddr1~115
memaddr1~127
OUT0
[3,28]
memaddr1~104
memaddr1~116
memaddr1~128
OUT0
[1,3-4,7,9,16,20-21,24,27-29,32,38-40]
OUT0
[1,3,27-29,39]
OUT0
[3,7,9,16,20-21,24,27-29,32,38-40]
memaddr1~19
memaddr1~7
memaddr1~20
memaddr1~8
@OUT8030,OUT8028,OUT8027
[1,3-4,7,9-10,19-21,27-29,34,38-39,42]
mem_addr_prev~7
mem_addr_prev~19
mem_addr_prev~31
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
mem_addr_prev~8
mem_addr_prev~20
mem_addr_prev~32
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
Equal20
A[31..0]
32'h0
B[31..0]
OUT
state~17
state~49
state~81
state~113
OUT0
[1,3-4,27-29,31,38-39]
@OUT010982,OUT010984,OUT010986
[1,3-4,38]
state~20
state~52
state~84
state~116
state~31
state~63
state~95
state~127
state~4
state~36
state~68
state~100
OUT0
[1,3-25,27-30,32-36,38-42]
state~29
state~61
state~93
state~125

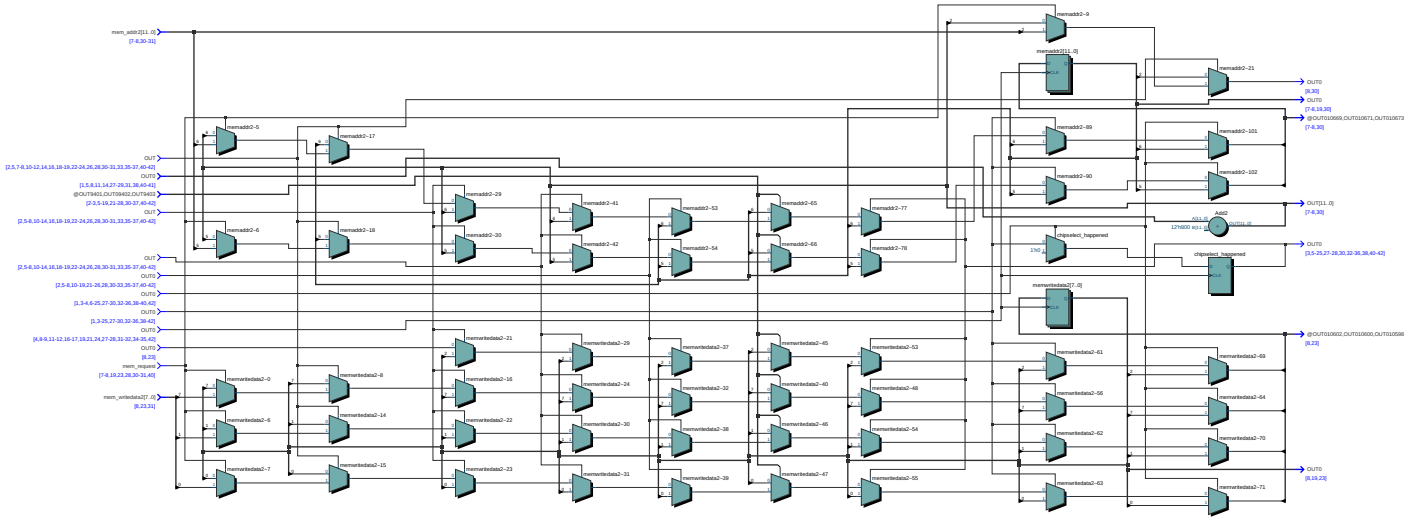

@OUT8030,OUT8028,OUT8027
[1,3-4,7,9-10,19-21,27-29,34,38-39,42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT0
[1,3-4,7,9,16,20-21,24,27-29,32,38-40]
OUT0
[4,8-9,11-12,16-17,19,21,24,27-28,31-32,34-35,42]
@OUT010982,OUT010984,OUT010986
[1,3-4,38]
OUT0
[1,3-4,27-29,31,38-39]
state[31..0]
CLK
state~6
state~7
state~8
state~11
state~13
state~15
state~19
state~21
state~23
state~24
state~3
state~25
state~26
state~28
state~9
state~38
state~39
state~40
state~43
state~45
state~47
state~51
state~53
state~55
state~56
state~35
state~57
state~58
state~60
state~41
state~70
state~71
state~72
state~75
state~77
state~79
state~83
state~85
state~87
state~88
state~67
state~89
state~90
state~92
state~73
state~102
state~103
state~104
state~107
state~109
state~111
state~115
state~117
state~119
state~120
state~99
state~121
state~122
state~124
state~105

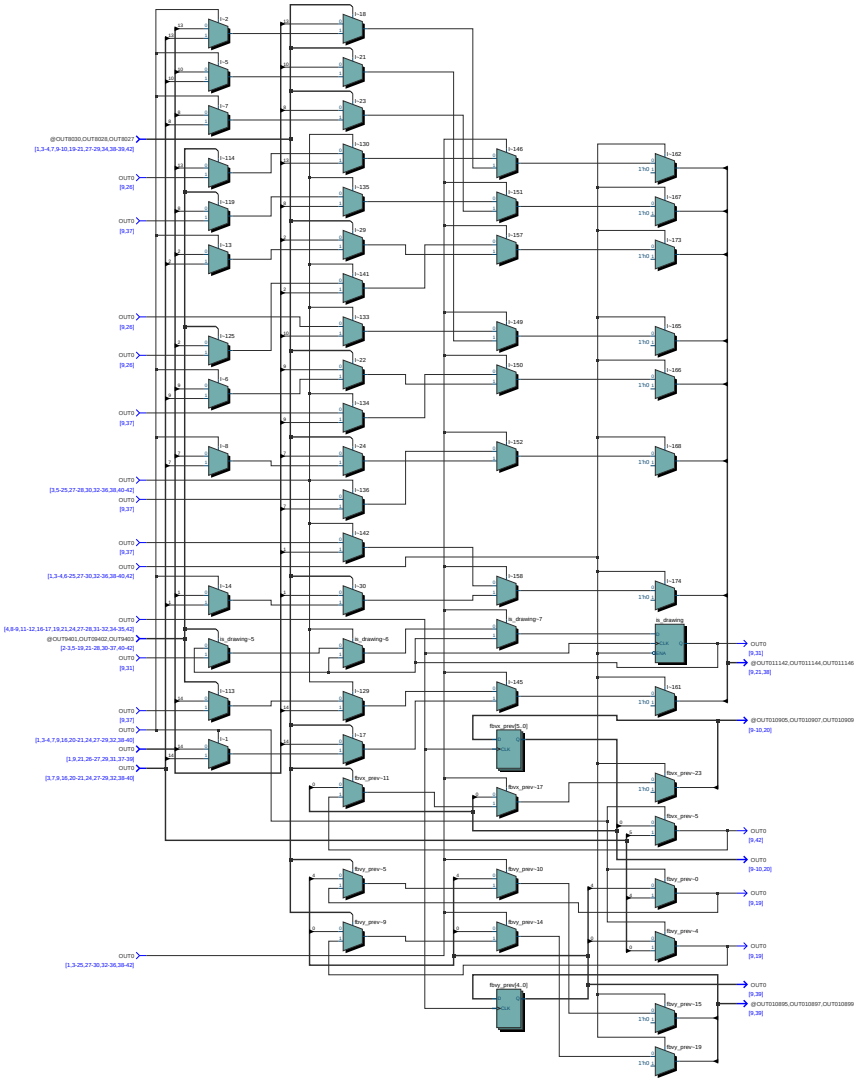

OUT
[2,5,7-8,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[5,11-12,14,22,31,33,35-36,41]
OUT0
[5,12,14,16,41]
OUT0
[1,5,8,11,14,27-29,31,38,40-41]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[1,3-25,27-30,32-36,38-42]
stack_writedata~1
stack_writedata~2
stack_writedata~5
stack_writedata~10
stack_writedata~12
stack_writedata~7
stack_writedata~9
stack_writedata~14
stack_writedata~17
stack_writedata~18
stack_writedata~21
stack_writedata~26
stack_writedata~28
stack_writedata~23
stack_writedata~25
stack_writedata~30
stack_writedata~33
stack_writedata~34
stack_writedata~37
stack_writedata~42
stack_writedata~44
stack_writedata~39
stack_writedata~41
stack_writedata~46
stack_writedata~49
stack_writedata~50
stack_writedata~53
stack_writedata~58
stack_writedata~60
stack_writedata~55
stack_writedata~57
stack_writedata~62
stack_writedata~65
stack_writedata~66
stack_writedata~69
stack_writedata~74
stack_writedata~76
stack_writedata~71
stack_writedata~73
stack_writedata~78
stack_writedata~81
stack_writedata~82
stack_writedata~85
stack_writedata~90
stack_writedata~92
stack_writedata~87
stack_writedata~89
stack_writedata~94
stack_writedata~97
stack_writedata~98
stack_writedata~101
stack_writedata~106
stack_writedata~108
stack_writedata~103
stack_writedata~105
stack_writedata~110
stack_writedata~113
stack_writedata~114
stack_writedata~117
stack_writedata~122
stack_writedata~124
stack_writedata~119
stack_writedata~121
stack_writedata~126
@OUT010609,OUT010611,OUT010613
[5,12,14,41]

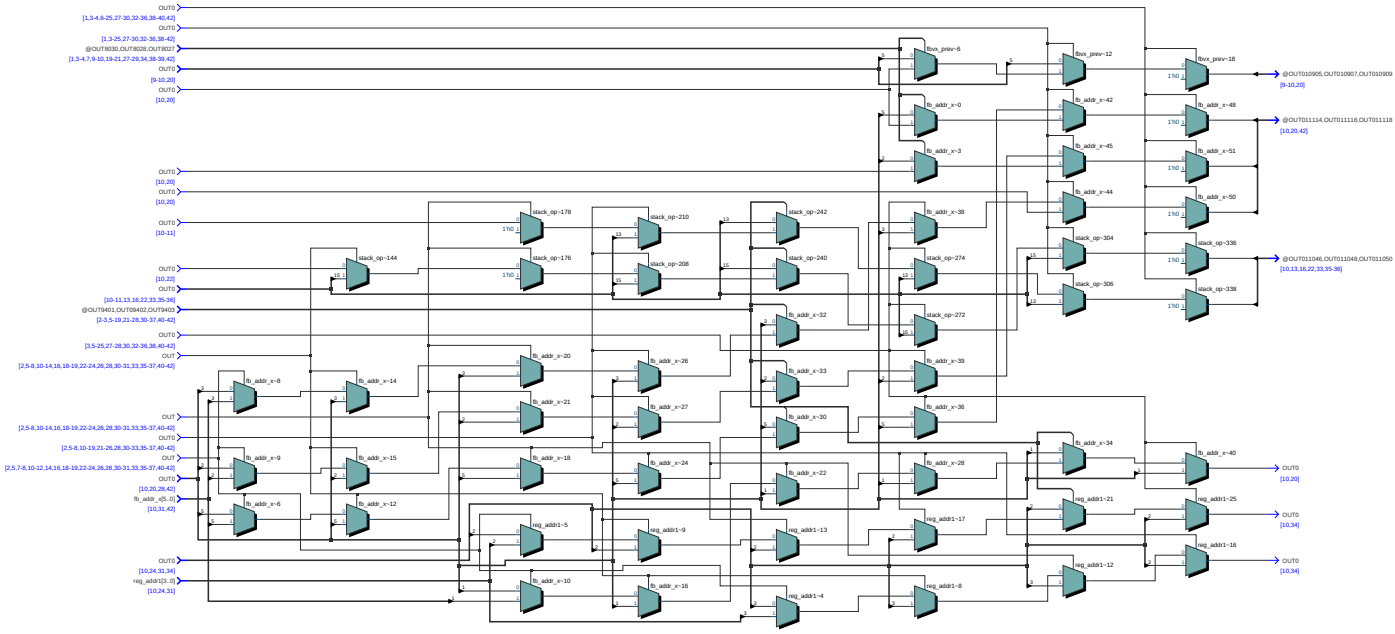

OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
q_b[7..0]
[6,19,31,42]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT0
[1,6,29,31,39,42]
q_a[7..0]
[1,6,19,27,31,42]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
cpu_instruction~10
cpu_instruction~11
cpu_instruction~12
cpu_instruction~15
cpu_instruction~1
cpu_instruction~7
cpu_instruction~8
cpu_instruction~9
cpu_instruction~2
cpu_instruction~26
cpu_instruction~27
cpu_instruction~28
cpu_instruction~31
cpu_instruction~17
cpu_instruction~23
cpu_instruction~24
cpu_instruction~25
cpu_instruction~18
cpu_instruction~42
cpu_instruction~43
cpu_instruction~44
cpu_instruction~47
cpu_instruction~33
cpu_instruction~39
cpu_instruction~40
cpu_instruction~41
cpu_instruction~34
cpu_instruction~58
cpu_instruction~59
cpu_instruction~60
cpu_instruction~63
cpu_instruction~49
cpu_instruction~55
cpu_instruction~56
cpu_instruction~57
cpu_instruction~50
cpu_instruction~74
cpu_instruction~75
cpu_instruction~76
cpu_instruction~79
cpu_instruction~65
cpu_instruction~71
cpu_instruction~72
cpu_instruction~73
cpu_instruction~66
cpu_instruction~90
cpu_instruction~91
cpu_instruction~92
cpu_instruction~95
cpu_instruction~81
cpu_instruction~87
cpu_instruction~88
cpu_instruction~89
cpu_instruction~82
cpu_instruction~106
cpu_instruction~107
cpu_instruction~108
cpu_instruction~111
cpu_instruction~97
cpu_instruction~103
cpu_instruction~104
cpu_instruction~105
cpu_instruction~98
@OUT011178,OUT011180,OUT011182
[6,42]

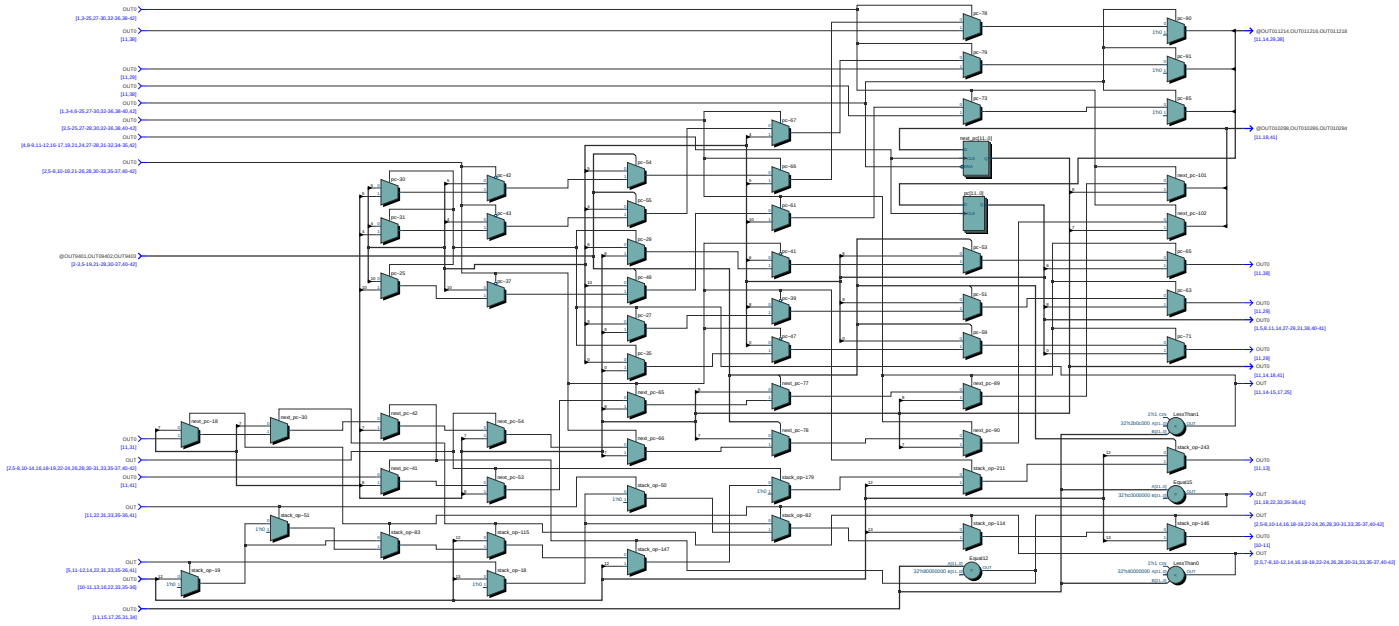


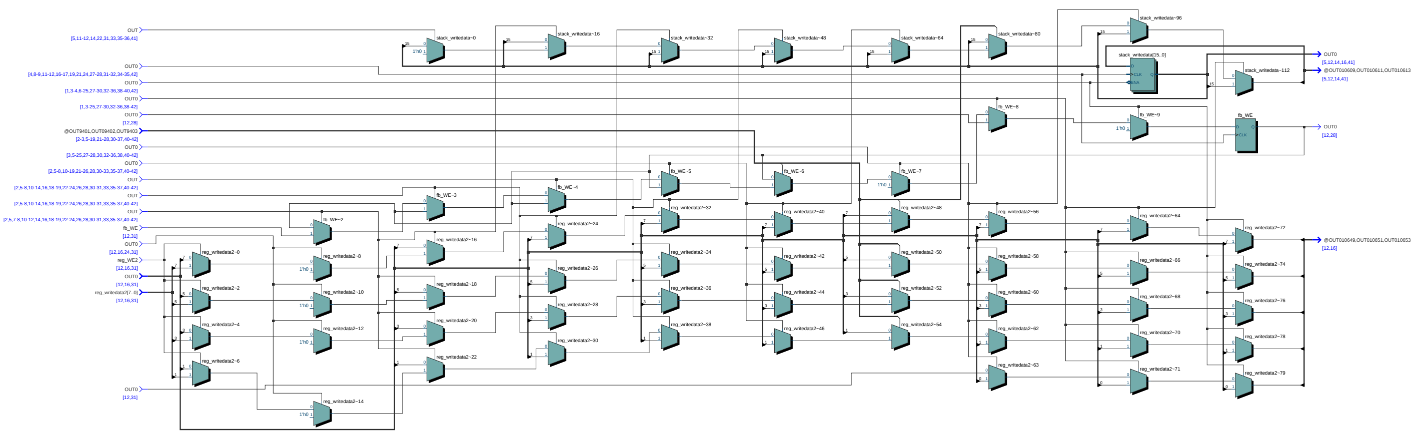
@OUT8030,OUT8028,OUT8027
[1,3-4,7,9-10,19-21,27-29,34,38-39,42]
OUT0
[1,3-4,7,9,16,20-21,24,27-29,32,38-40]
OUT0
[3,7,9,16,20-21,24,27-29,32,38-40]
OUT
[2,5,7-8,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT0
[7,19,21,23]
mem_writedata1[7..0]
[7,23,31]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
mem_request
[7-8,19,23,28,30-31,40]
OUT[11..0]
[7-8,30]
mem_addr2[11..0]
[7-8,30-31]
OUT0
[7-8,19,30]
OUT0
[7,30]
memwritedata1~17
memwritedata1~25
memwritedata1~33
memwritedata1~41
memwritedata1~22
memwritedata1~30
memwritedata1~38
memwritedata1~46
memwritedata1~23
memwritedata1~31
memwritedata1~39
memwritedata1~47
memwritedata1~18
memaddr2~7
memaddr2~19
memaddr2~31
memaddr2~43
memaddr2~8
memaddr2~20
memaddr2~32
memaddr2~44
memaddr2~24
memaddr2~36
memwritedata1~49
memwritedata1~54
memwritedata1~55
memwritedata1~26
memaddr2~55
memaddr2~56
memaddr2~48
memwritedata1~6
memwritedata1~1
memwritedata1~7
memwritedata1~57
memwritedata1~62
memwritedata1~63
memaddr2~67
memaddr2~68
memaddr2~60
memwritedata1~34
memwritedata1~14
memwritedata1~9
memwritedata1~15
memwritedata1~65
memwritedata1~70
memwritedata1~71
memaddr2~79
memaddr2~80
memaddr2~72
memwritedata1~42
memwritedata1~73
memwritedata1~78
memwritedata1~79
memaddr2~91
memaddr2~92
memaddr2~84
memwritedata1~50
memwritedata1~81
memwritedata1~86
memwritedata1~87
memaddr2~103
memaddr2~104
memaddr2~96
memwritedata1~58
@OUT010695,OUT010697,OUT010699
[7,21]
@OUT010669,OUT010671,OUT010673
[7-8,30]
OUT0
[7,21]


mem_addr2[11..0]
[7-8,30-31]
OUT
[2,5-7,9,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT0
[1,5,9,11,14,27-29,31,38,40-41]
@OUT10431,OUT10432,OUT10433
[2-3,5-19,21-28,30-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT0
[4,8-9,11-12,16-17,19,21,24,27-28,31-32,34-35,42]
OUT0
[8,23]
mem_request
[7-8,19,23,28,30-31,40]
mem_writedata2[7..0]
[8,23,31]
memaddr2~9
memaddr2[11..0]
memaddr2~21
OUT0
[8,30]
OUT0
[7-8,19,30]
@OUT010669,OUT010671,OUT010673
[7-8,30]
memaddr2~89
memaddr2~101
memaddr2~90
memaddr2~102
OUT[11..0]
[7-8,30]
Add2
A[11..0]
127h800
B[11..0]
OUT[11..0]
chipselect_happened
1'h0
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
chipselect_happened
memwritedata2[7..0]
@OUT010602,OUT010600,OUT010598
[8,23]
memwritedata2~61
memwritedata2~69
memwritedata2~56
memwritedata2~64
memwritedata2~62
memwritedata2~70
OUT0
[8,19,23]
memwritedata2~63
memwritedata2~71
memaddr2~5
memaddr2~17
memaddr2~6
memaddr2~18
memaddr2~29
memaddr2~41
memaddr2~53
memaddr2~65
memaddr2~77
memaddr2~30
memaddr2~42
memaddr2~54
memaddr2~66
memaddr2~78
memwritedata2~0
memwritedata2~8
memwritedata2~6
memwritedata2~14
memwritedata2~7
memwritedata2~15
memwritedata2~21
memwritedata2~29
memwritedata2~37
memwritedata2~45
memwritedata2~53
memwritedata2~16
memwritedata2~24
memwritedata2~32
memwritedata2~40
memwritedata2~48
memwritedata2~22
memwritedata2~30
memwritedata2~38
memwritedata2~46
memwritedata2~54
memwritedata2~23
memwritedata2~31
memwritedata2~39
memwritedata2~47
memwritedata2~55

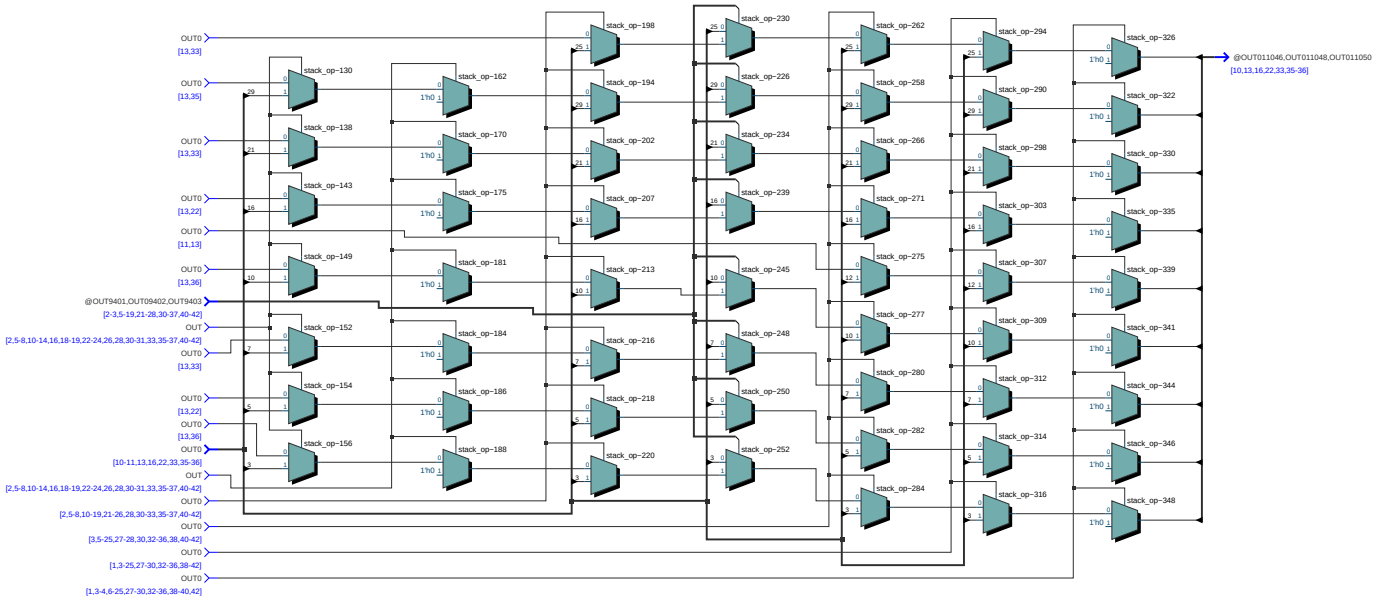

@OUT8030,OUT8028,OUT8027
[1,3-4,7,9-10,19-21,27-29,34,38-39,42]
OUT0
[9,26]
OUT0
[9,37]
OUT0
[9,26]
OUT0
[9,26]
OUT0
[9,37]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[9,37]
OUT0
[9,37]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT0
[4,6-9,11-12,16-17,19,21,24,27-28,31-32,34-35,42]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT0
[9,31]
OUT0
[9,37]
OUT0
[1,3-4,7,9,16,20-21,24,27-29,32,38-40]
OUT0
[1,9,21,26-27,29,31,37-38]
OUT0
[3,7,9,16,20-21,24,27-29,32,38-40]
OUT0
[1,3-25,27-30,32-36,38-42]
fbvx_prev[5..0]
CLK
fbvy_prev[4..0]
CLK
is_drawing
CLK
OUT0
[9,31]
@OUT011142,OUT011144,OUT011146
[9,21,38]
@OUT010905,OUT010907,OUT010909
[9-10,20]
OUT0
[9,42]
OUT0
[9-10,20]
OUT0
[9,19]
OUT0
[9,19]
OUT0
[9,39]
@OUT010895,OUT010897,OUT010899
[9,39]

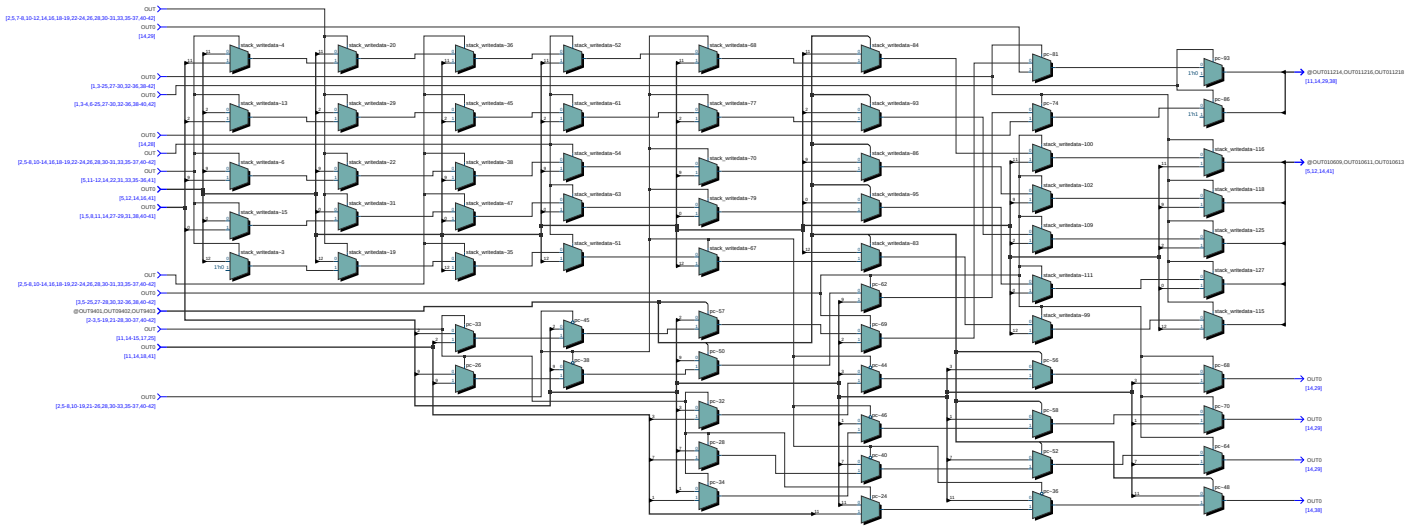

OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT0
[1,3-25,27-30,32-36,38-42]
@OUT8030,OUT8028,OUT8027
[1,3-4,7,9-10,19-21,27-29,34,38-39,42]
OUT0
[9-10,20]
OUT0
[10,20]
OUT0
[10,20]
OUT0
[10,20]
OUT0
[10-11]
OUT0
[10,22]
OUT0
[10-11,13,16,22,33,35-36]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
OUT
[2,5,7-8,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT0
[10,20,28,42]
fb_addr_x[5..0]
[10,31,42]
OUT0
[10,24,31,34]
req_addr1[3..0]
[10,24,31]
fbvx_prev~6
fbvx_prev~12
fbvx_prev~18
@OUT010905,OUT010907,OUT010909
[9-10,20]
fb_addr_x~0
fb_addr_x~3
fb_addr_x~42
fb_addr_x~45
fb_addr_x~44
fb_addr_x~48
fb_addr_x~51
fb_addr_x~50
@OUT011114,OUT011116,OUT011118
[10,20,42]
stack_op~144
stack_op~178
stack_op~176
stack_op~210
stack_op~208
stack_op~242
stack_op~240
stack_op~304
stack_op~306
stack_op~336
stack_op~338
@OUT011046,OUT011048,OUT011050
[10,13,16,22,33,35-36]
fb_addr_x~38
stack_op~274
stack_op~272
fb_addr_x~32
fb_addr_x~33
fb_addr_x~30
fb_addr_x~39
fb_addr_x~36
fb_addr_x~8
fb_addr_x~14
fb_addr_x~20
fb_addr_x~21
fb_addr_x~18
fb_addr_x~26
fb_addr_x~27
fb_addr_x~24
fb_addr_x~22
fb_addr_x~28
fb_addr_x~34
fb_addr_x~40
OUT0
[10,20]
fb_addr_x~9
fb_addr_x~15
fb_addr_x~6
fb_addr_x~12
req_addr1~5
fb_addr_x~10
req_addr1~9
fb_addr_x~16
req_addr1~13
req_addr1~4
req_addr1~17
req_addr1~8
req_addr1~21
req_addr1~12
req_addr1~25
OUT0
[10,34]
req_addr1~16
OUT0
[10,34]

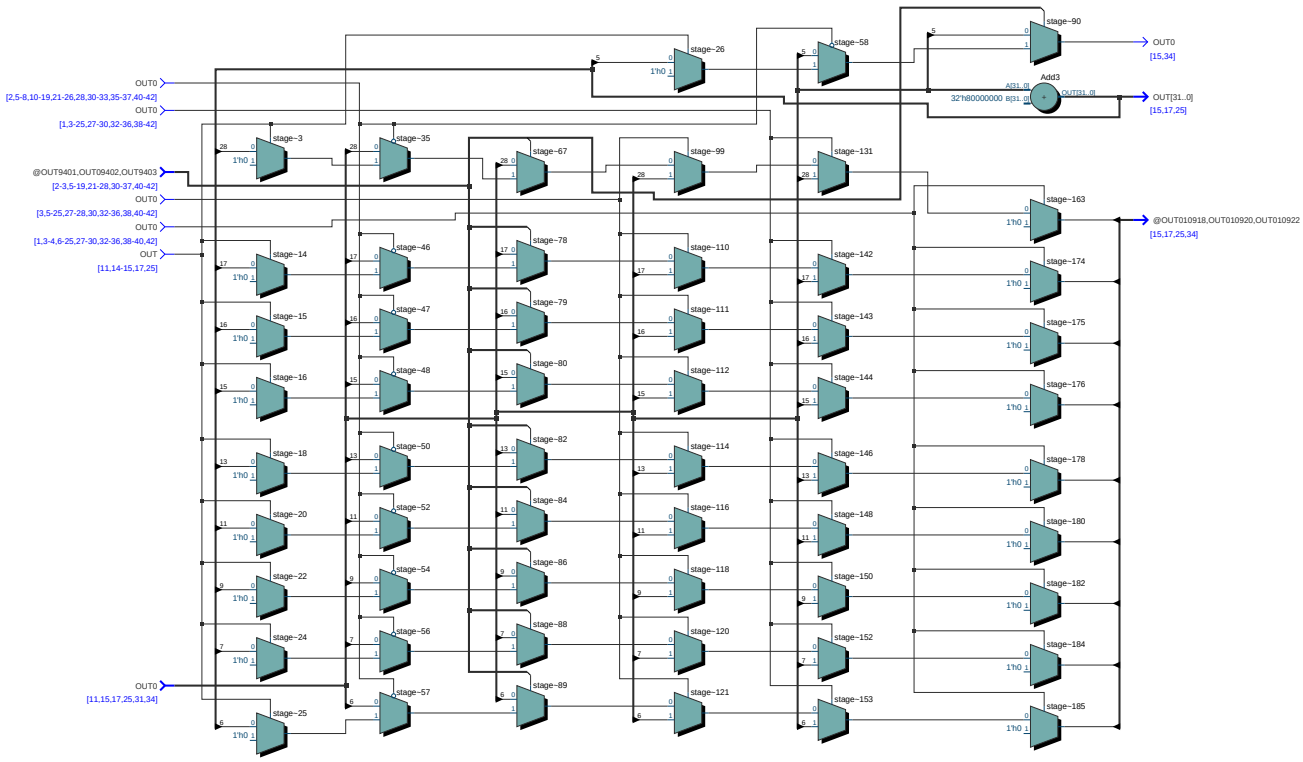


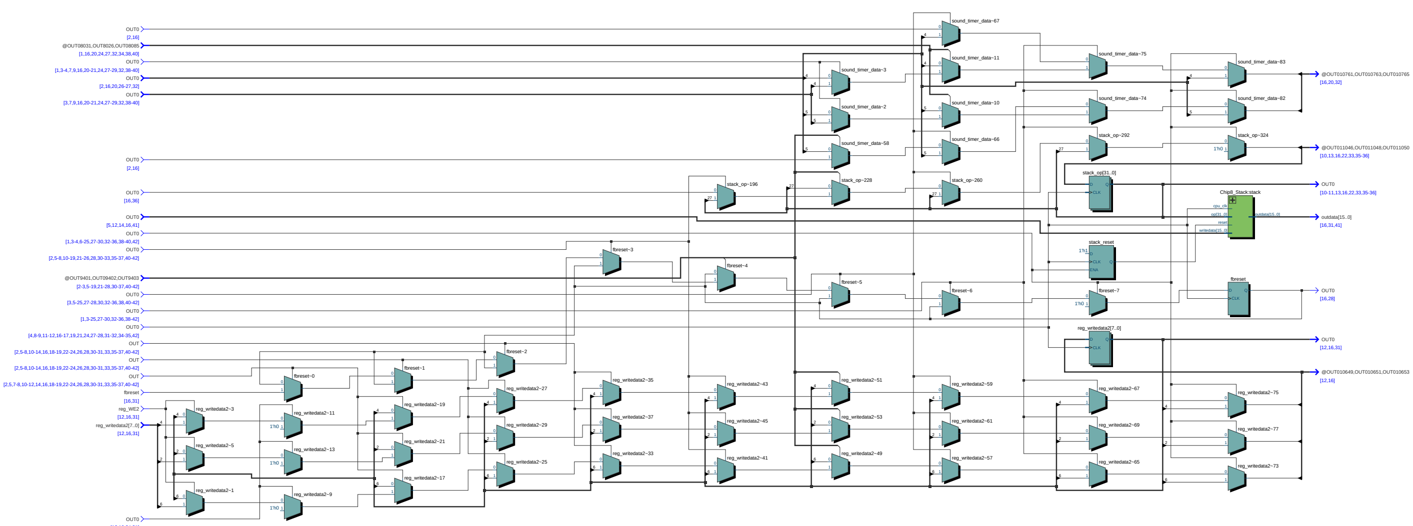


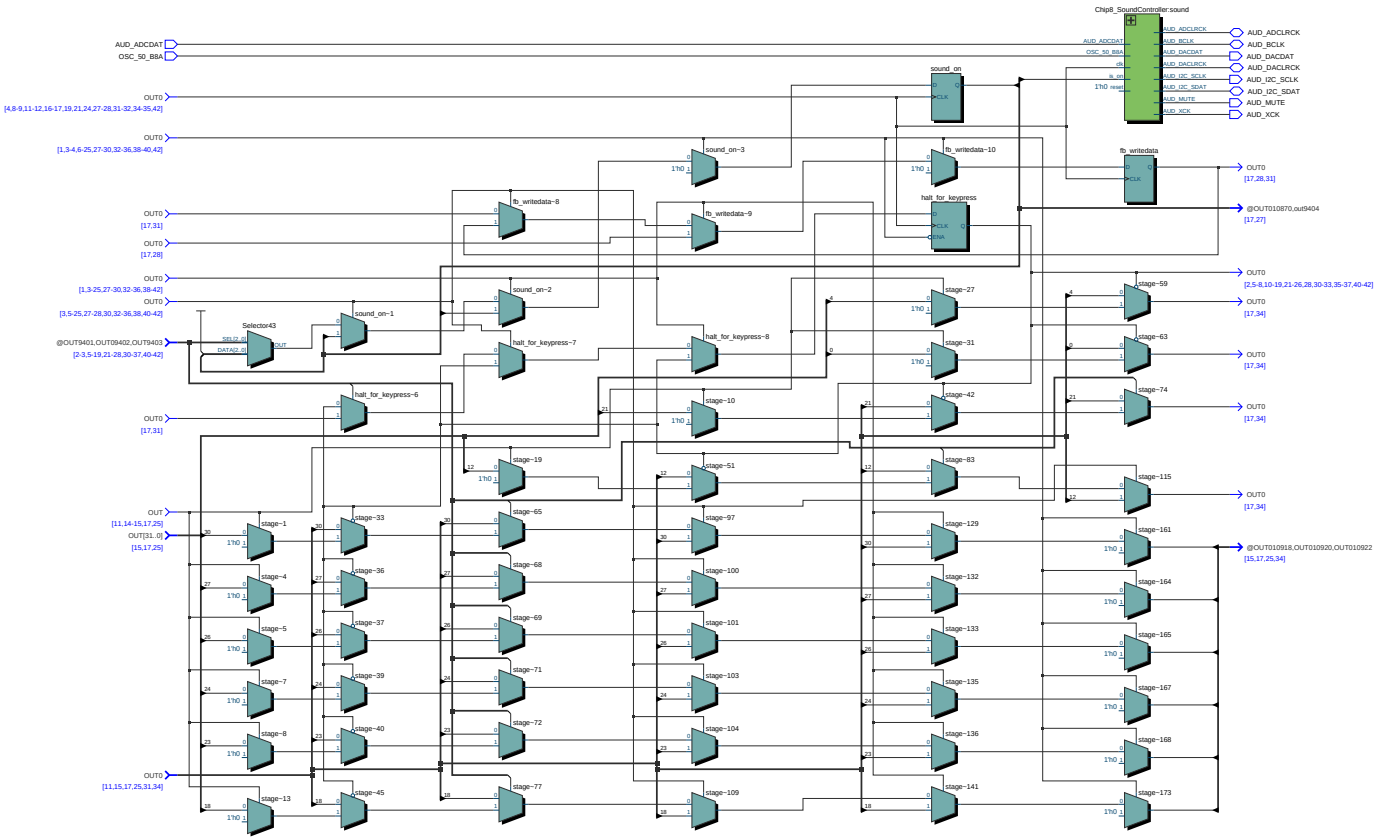
OUT
[5,11-12,14,22,31,33,35-36,41]
OUT0
[4,8-9,11-12,16-17,19,21,24,27-28,31-32,34-35,42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT0
[12,28]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5,7-8,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
fb_WE
[12,31]
OUT0
[12,16,24,31]
reg_WE2
[12,16,31]
OUT0
[12,16,31]
reg_writedata2[7..0]
[12,16,31]
OUT0
[12,31]
stack_writedata~0
stack_writedata~16
stack_writedata~32
stack_writedata~48
stack_writedata~64
stack_writedata~80
stack_writedata~96
stack_writedata~112
stack_writedata[15..0]
fb_WE~2
fb_WE~3
fb_WE~4
fb_WE~5
fb_WE~6
fb_WE~7
fb_WE~8
fb_WE~9
fb_WE
reg_writedata2~0
reg_writedata2~2
reg_writedata2~4
reg_writedata2~6
reg_writedata2~8
reg_writedata2~10
reg_writedata2~12
reg_writedata2~14
reg_writedata2~16
reg_writedata2~18
reg_writedata2~20
reg_writedata2~22
reg_writedata2~24
reg_writedata2~26
reg_writedata2~28
reg_writedata2~30
reg_writedata2~32
reg_writedata2~34
reg_writedata2~36
reg_writedata2~38
reg_writedata2~40
reg_writedata2~42
reg_writedata2~44
reg_writedata2~46
reg_writedata2~48
reg_writedata2~50
reg_writedata2~52
reg_writedata2~54
reg_writedata2~56
reg_writedata2~58
reg_writedata2~60
reg_writedata2~62
reg_writedata2~63
reg_writedata2~64
reg_writedata2~66
reg_writedata2~68
reg_writedata2~70
reg_writedata2~71
reg_writedata2~72
reg_writedata2~74
reg_writedata2~76
reg_writedata2~78
reg_writedata2~79
1'h0
D
CLK
ENA
Q
OUT0
[5,12,14,16,41]
@OUT010609,OUT010611,OUT010613
[5,12,14,41]
OUT0
[12,28]
@OUT010649,OUT010651,OUT010653
[12,16]

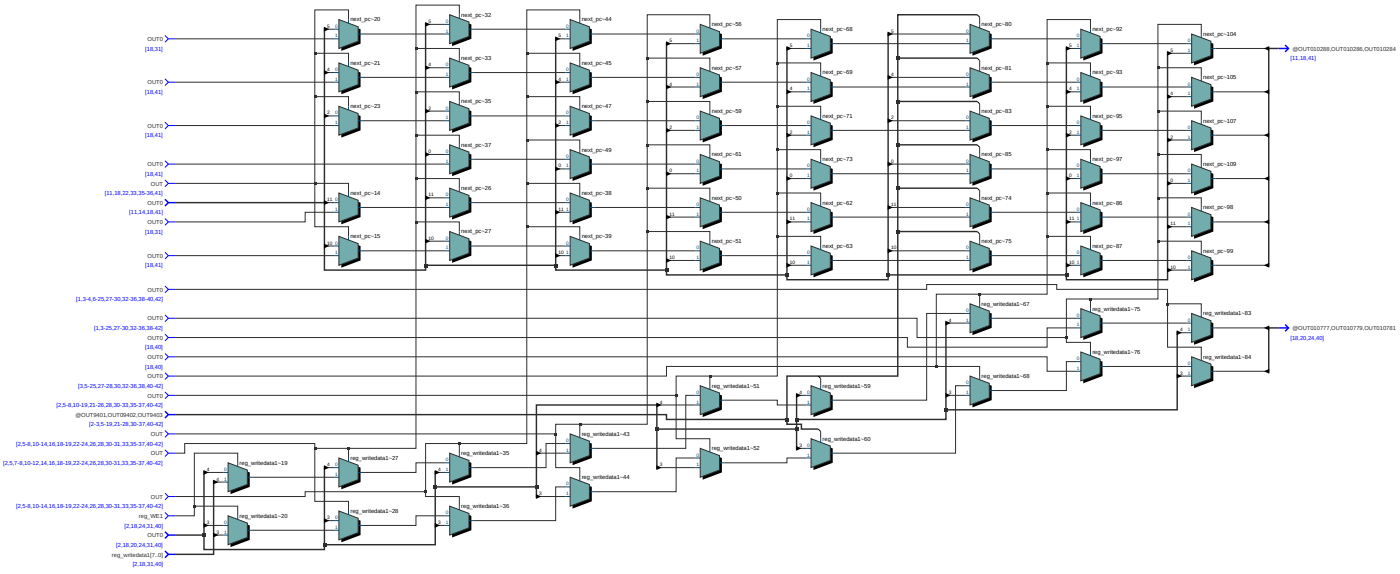

OUT0
[13,33]
OUT0
[13,35]
OUT0
[13,33]
OUT0
[13,22]
OUT0
[11,13]
OUT0
[13,36]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT0
[13,33]
OUT0
[13,22]
OUT0
[13,36]
OUT0
[10-11,13,16,22,33,35-36]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
stack_op~130
stack_op~138
stack_op~143
stack_op~149
stack_op~152
stack_op~154
stack_op~156
stack_op~162
stack_op~170
stack_op~175
stack_op~181
stack_op~184
stack_op~186
stack_op~188
stack_op~198
stack_op~194
stack_op~202
stack_op~207
stack_op~213
stack_op~216
stack_op~218
stack_op~220
stack_op~230
stack_op~226
stack_op~234
stack_op~239
stack_op~245
stack_op~248
stack_op~250
stack_op~252
stack_op~262
stack_op~258
stack_op~266
stack_op~271
stack_op~275
stack_op~277
stack_op~280
stack_op~282
stack_op~284
stack_op~294
stack_op~290
stack_op~298
stack_op~303
stack_op~307
stack_op~309
stack_op~312
stack_op~314
stack_op~316
stack_op~326
stack_op~322
stack_op~330
stack_op~335
stack_op~339
stack_op~341
stack_op~344
stack_op~346
stack_op~348
1'h0
@OUT011046,OUT011048,OUT011050
[10,13,16,22,33,35-36]

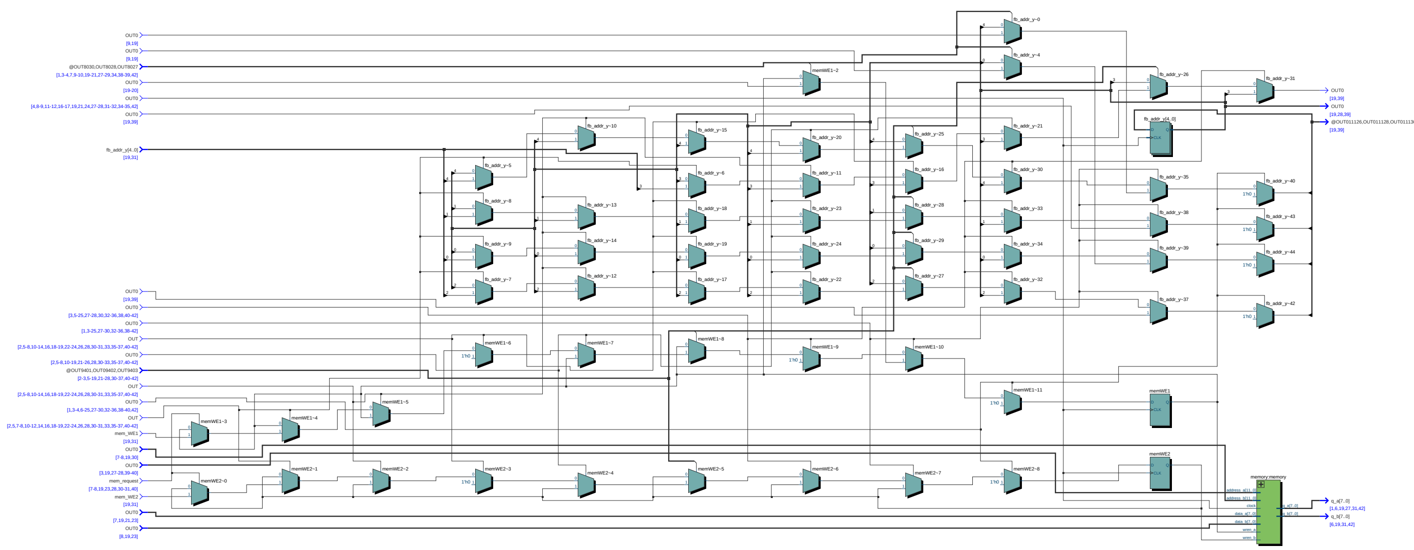

OUT
[2,5-7,9,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT0
[14,29]
OUT0
[1,3-25,27-30,32-36,38-43]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT0
[14,28]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[5,11-12,14,22,31,33,35-36,41]
OUT0
[5,12,14,16,41]
OUT0
[1,5,8,11,14,27-29,31,38,40-41]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
@OUT9401,OUT09402,OUT9403
[2,3,5-18,21-28,30-37,40-42]
OUT
[11,14-15,17,25]
OUT0
[11,14,18,41]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
stack_writedata~4
stack_writedata~20
stack_writedata~36
stack_writedata~52
stack_writedata~68
stack_writedata~84
sc~81
sc~93
@OUT011214,OUT011216,OUT011218
[11,14,29,38]
stack_writedata~13
stack_writedata~29
stack_writedata~45
stack_writedata~61
stack_writedata~77
stack_writedata~93
sc~74
sc~86
stack_writedata~6
stack_writedata~22
stack_writedata~38
stack_writedata~54
stack_writedata~70
stack_writedata~86
stack_writedata~100
stack_writedata~116
@OUT010609,OUT010611,OUT010613
[5,12,14,41]
stack_writedata~15
stack_writedata~31
stack_writedata~47
stack_writedata~63
stack_writedata~79
stack_writedata~95
stack_writedata~102
stack_writedata~118
stack_writedata~3
stack_writedata~19
stack_writedata~35
stack_writedata~51
stack_writedata~67
stack_writedata~83
stack_writedata~109
stack_writedata~125
stack_writedata~111
stack_writedata~127
stack_writedata~99
stack_writedata~115
sc~62
sc~69
sc~44
sc~46
sc~40
sc~24
sc~33
sc~45
sc~26
sc~38
sc~57
sc~50
sc~32
sc~28
sc~34
sc~56
sc~58
sc~52
sc~36
sc~68
sc~70
sc~64
sc~48
OUT0
[14,29]
OUT0
[14,29]
OUT0
[14,29]
OUT0
[14,38]


OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
OUT0
[1,3-25,27-30,32-36,38-42]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT
[11,14-15,17,25]
OUT0
[11,15,17,25,31,34]
stage-3
stage-14
stage-15
stage-16
stage-18
stage-20
stage-22
stage-24
stage-25
stage-35
stage-46
stage-47
stage-48
stage-50
stage-52
stage-54
stage-56
stage-57
stage-67
stage-78
stage-79
stage-80
stage-82
stage-84
stage-86
stage-88
stage-89
stage-26
stage-99
stage-110
stage-111
stage-112
stage-114
stage-116
stage-118
stage-120
stage-121
stage-58
stage-131
stage-142
stage-143
stage-144
stage-146
stage-148
stage-150
stage-152
stage-153
stage-90
Add3
32'h80000000
A[31..0]
B[31..0]
OUT[31..0]
OUT0
[15,34]
OUT[31..0]
[15,17,25]
stage-163
stage-174
stage-175
stage-176
stage-178
stage-180
stage-182
stage-184
stage-185
@OUT010918,OUT010920,OUT010922
[15,17,25,34]

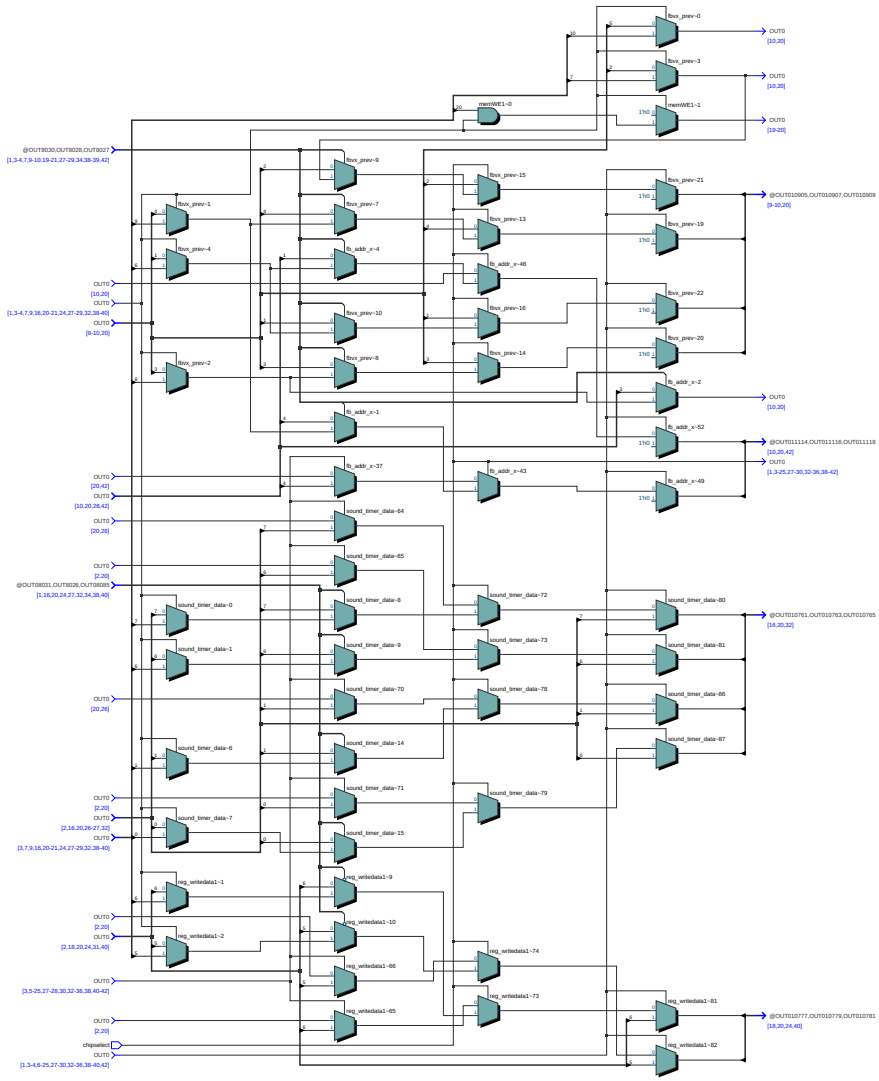

OUT0
[2,16]
@OUT08031,OUT8026,OUT08085
[1,16,20,24,27,32,34,38,40]
OUT0
[1,3-4,7,9,16,20-21,24,27-29,32,38-40]
OUT0
[2,16,20,26-27,32]
OUT0
[3,7,9,16,20-21,24,27-29,32,38-40]
OUT0
[2,16]
OUT0
[16,36]
OUT0
[5,12,14,16,41]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT0
[4,8-9,11-12,16-17,19,21,24,27-28,31-32,34-35,42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5,7-8,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
fbreset
[16,31]
reg_WE2
[12,16,31]
reg_writedata2[7..0]
[12,16,31]
OUT0
[12,16,24,31]
sound_timer_data~67
sound_timer_data~11
sound_timer_data~3
sound_timer_data~2
sound_timer_data~10
sound_timer_data~58
sound_timer_data~66
sound_timer_data~75
sound_timer_data~74
sound_timer_data~83
sound_timer_data~82
stack_op~196
stack_op~228
stack_op~260
stack_op~292
stack_op~324
stack_op[31..0]
Chip8_Stack:stack
cpu_clk
op[31..0]
reset
writedata[15..0]
outdata[15..0]
stack_reset
fbreset
fbreset~0
fbreset~1
fbreset~2
fbreset~3
fbreset~4
fbreset~5
fbreset~6
fbreset~7
reg_writedata2[7..0]
reg_writedata2~1
reg_writedata2~3
reg_writedata2~5
reg_writedata2~9
reg_writedata2~11
reg_writedata2~13
reg_writedata2~17
reg_writedata2~19
reg_writedata2~21
reg_writedata2~25
reg_writedata2~27
reg_writedata2~29
reg_writedata2~33
reg_writedata2~35
reg_writedata2~37
reg_writedata2~41
reg_writedata2~43
reg_writedata2~45
reg_writedata2~49
reg_writedata2~51
reg_writedata2~53
reg_writedata2~57
reg_writedata2~59
reg_writedata2~61
reg_writedata2~65
reg_writedata2~67
reg_writedata2~69
reg_writedata2~73
reg_writedata2~75
reg_writedata2~77
@OUT010761,OUT010763,OUT010765
[16,20,32]
@OUT011046,OUT011048,OUT011050
[10,13,16,22,33,35-36]
OUT0
[10-11,13,16,22,33,35-36]
outdata[15..0]
[16,31,41]
OUT0
[16,28]
OUT0
[12,16,31]
@OUT010649,OUT010651,OUT010653
[12,16]

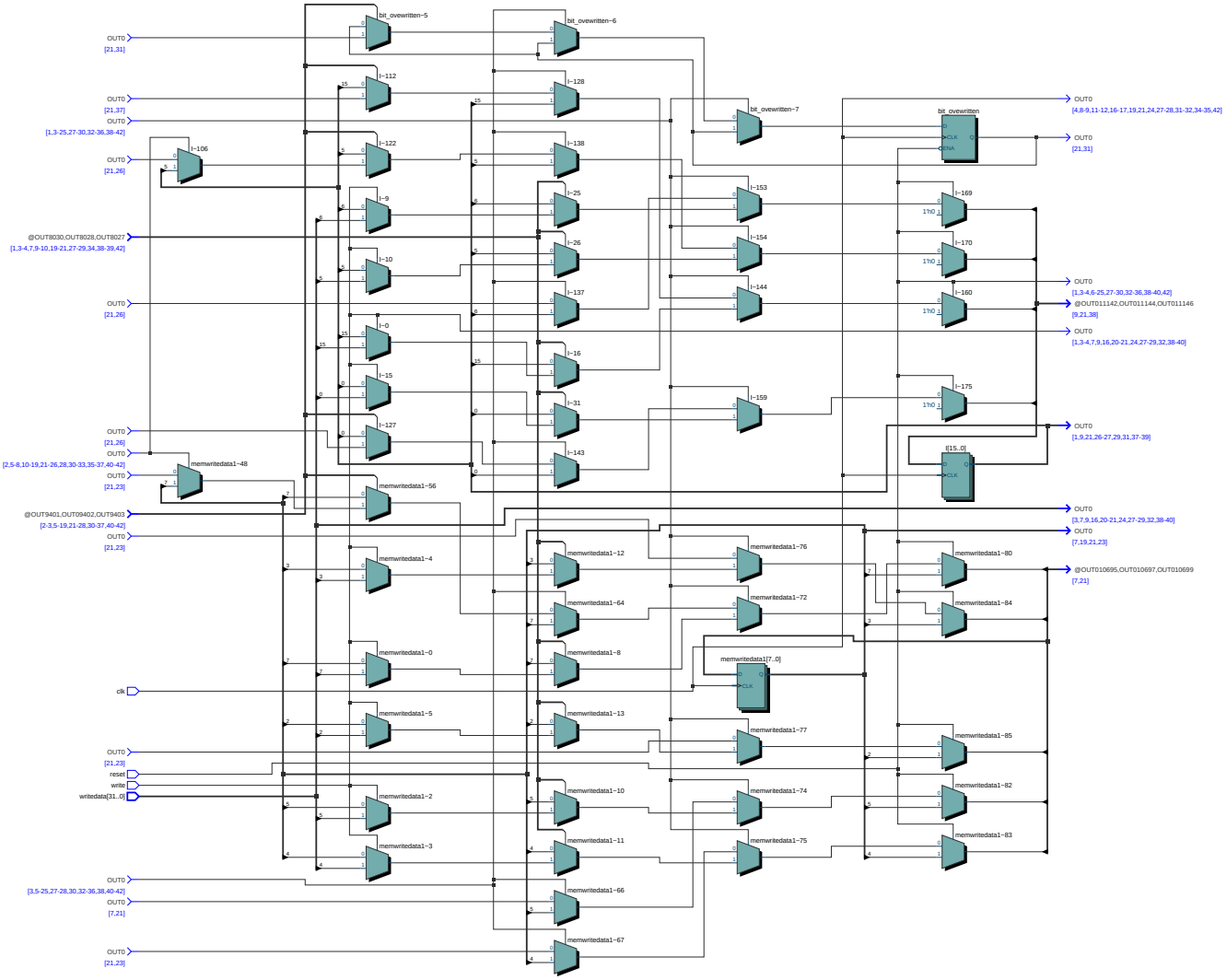

Chip8_SoundController:sound
AUD_ADCDAT
OSC_50_BBA
OUT0
[4,8-9,11-12,16-17,19,21,24,27-28,31-32,34-35,42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT0
[17,31]
OUT0
[17,28]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT0
[17,31]
OUT
[11,14-15,17,25]
OUT[31..0]
[15,17,25]
OUT0
[11,15,17,25,31,34]
AUD_ADCLRCK
AUD_BCLK
AUD_DACDAT
AUD_DACLRCK
AUD_I2C_SCLK
AUD_I2C_SDAT
AUD_MUTE
AUD_XCK
sound_on
fb_writedata~10
fb_writedata
halt_for_keypress
sound_on~3
fb_writedata~8
fb_writedata~9
sound_on~2
sound_on~1
Selector43
halt_for_keypress~7
halt_for_keypress~8
halt_for_keypress~6
stage~1
stage~4
stage~5
stage~7
stage~8
stage~13
stage~33
stage~36
stage~37
stage~39
stage~40
stage~45
stage~19
stage~65
stage~68
stage~69
stage~71
stage~72
stage~77
stage~10
stage~51
stage~97
stage~100
stage~101
stage~103
stage~104
stage~109
stage~27
stage~31
stage~42
stage~83
stage~129
stage~132
stage~133
stage~135
stage~136
stage~141
stage~59
stage~63
stage~74
stage~115
stage~161
stage~164
stage~165
stage~167
stage~168
stage~173
OUT0
[17,28,31]
@OUT010870,out9404
[17,27]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
OUT0
[17,34]
OUT0
[17,34]
OUT0
[17,34]
OUT0
[17,34]
@OUT010918,OUT010920,OUT010922
[15,17,25,34]

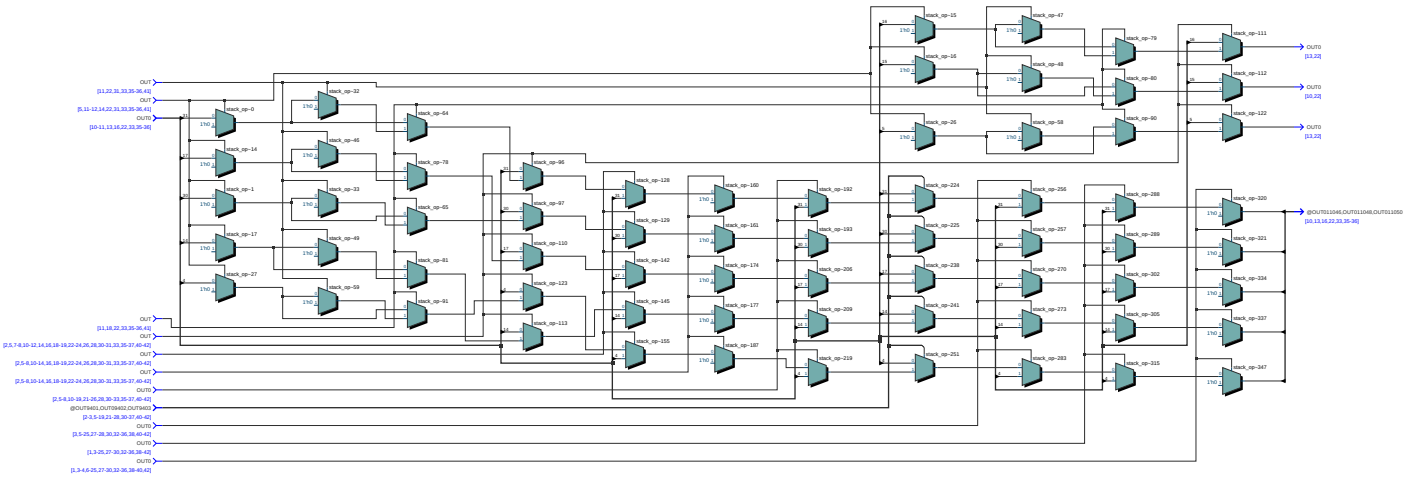

next_pc~20
next_pc~32
next_pc~44
next_pc~56
next_pc~68
next_pc~80
next_pc~92
next_pc~104
@OUT010288,OUT010286,OUT010284
[11,18,41]
OUT0
[18,33]
next_pc~21
next_pc~33
next_pc~45
next_pc~57
next_pc~69
next_pc~81
next_pc~93
next_pc~105
OUT0
[18,41]
next_pc~23
next_pc~35
next_pc~47
next_pc~59
next_pc~71
next_pc~83
next_pc~95
next_pc~107
OUT0
[18,41]
next_pc~37
next_pc~49
next_pc~61
next_pc~73
next_pc~85
next_pc~97
next_pc~109
OUT0
[18,41]
OUT
[11,18,22,33,35-36,41]
next_pc~14
next_pc~26
next_pc~38
next_pc~50
next_pc~62
next_pc~74
next_pc~86
next_pc~98
OUT0
[11,14,18,41]
OUT0
[18,33]
next_pc~15
next_pc~27
next_pc~39
next_pc~51
next_pc~63
next_pc~75
next_pc~87
next_pc~99
OUT0
[18,41]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT0
[18,40]
OUT0
[18,40]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[2,5-8,10-19,21-26,28,30-32,35-37,40-42]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-43]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5,7-8,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
reg_writedata1~19
reg_writedata1~27
reg_writedata1~35
reg_writedata1~43
reg_writedata1~51
reg_writedata1~59
reg_writedata1~67
reg_writedata1~75
reg_writedata1~83
@OUT010777,OUT010779,OUT010781
[18,20,24,40]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
reg_writedata1~20
reg_writedata1~28
reg_writedata1~36
reg_writedata1~44
reg_writedata1~52
reg_writedata1~60
reg_writedata1~68
reg_writedata1~76
reg_writedata1~84
reg_WE1
[2,18,24,31,40]
OUT0
[2,18,20,24,31,40]
reg_writedata17..0]
[2,18,31,40]

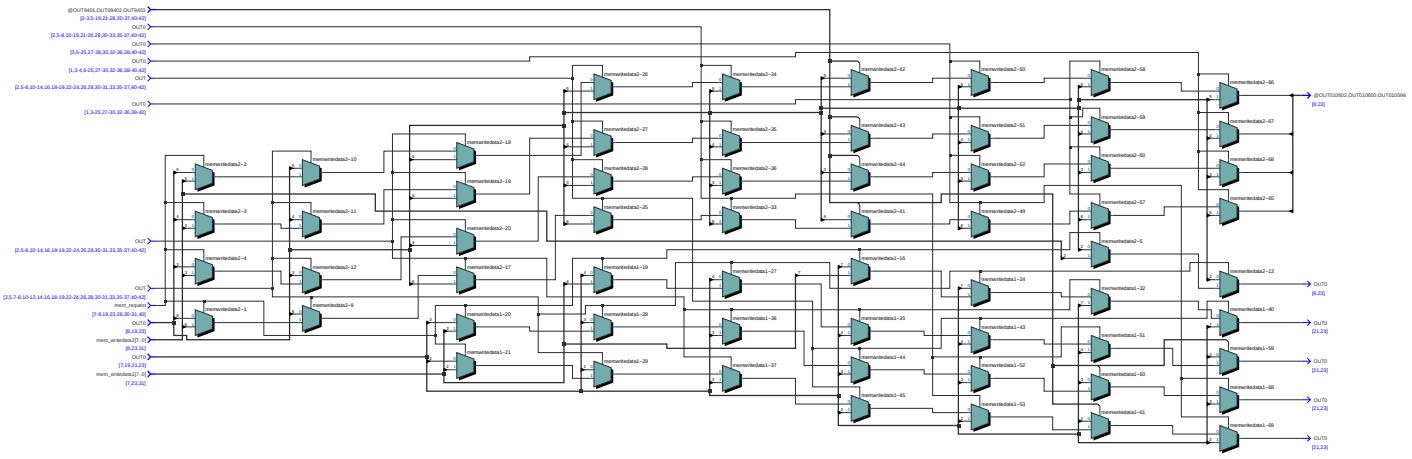


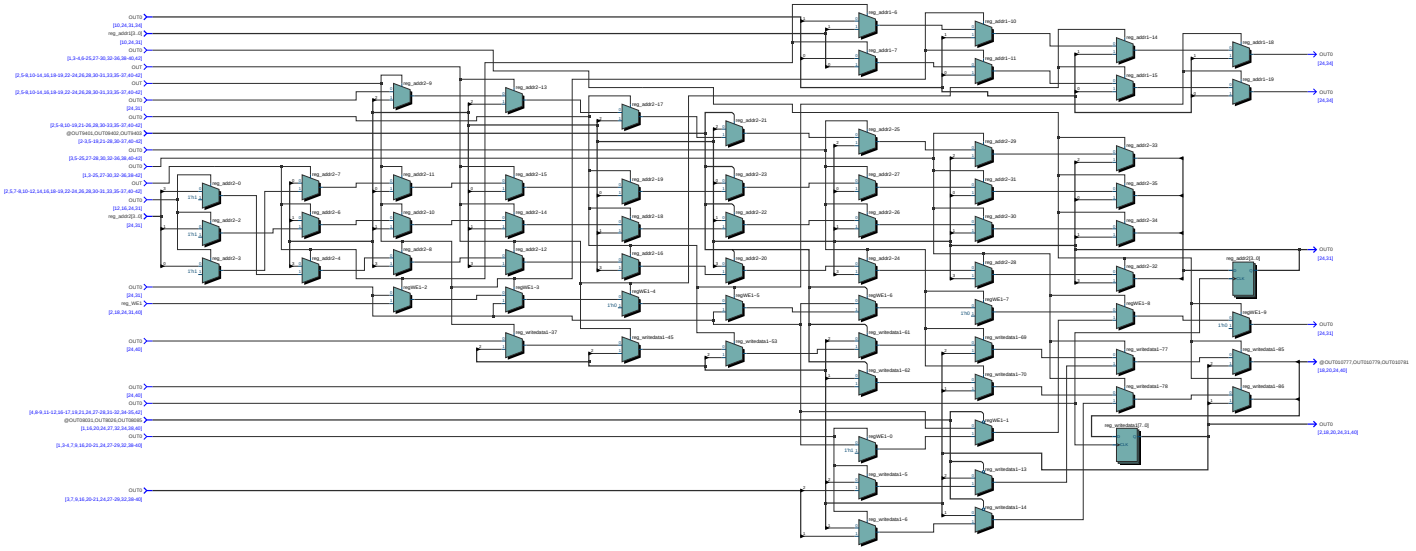
OUT0
[9,19]
OUT0
[9,19]
@OUT8030,OUT8028,OUT8027
[1,3-4,7,9-10,19-21,27-29,34,38-39,42]
OUT0
[19-20]
OUT0
[4,8-9,11-12,16-17,19,21,24,27-28,31-32,34-35,42]
OUT0
[19,39]
fb_addr_y[4..0]
[19,31]
OUT0
[19,39]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT
[2,5,7-8,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
mem_WE1
[19,31]
OUT0
[7-8,19,30]
OUT0
[3,19,27-28,39-40]
mem_request
[7-8,19,23,28,30-31,40]
mem_WE2
[19,31]
OUT0
[7,19,21,23]
OUT0
[8,19,23]
fb_addr_y~0
fb_addr_y~4
memWE1~2
fb_addr_y~26
fb_addr_y~31
fb_addr_y[4..0]
fb_addr_y~10
fb_addr_y~15
fb_addr_y~20
fb_addr_y~25
fb_addr_y~21
fb_addr_y~5
fb_addr_y~6
fb_addr_y~11
fb_addr_y~16
fb_addr_y~30
fb_addr_y~35
fb_addr_y~40
fb_addr_y~8
fb_addr_y~13
fb_addr_y~18
fb_addr_y~23
fb_addr_y~28
fb_addr_y~33
fb_addr_y~38
fb_addr_y~43
fb_addr_y~9
fb_addr_y~14
fb_addr_y~19
fb_addr_y~24
fb_addr_y~29
fb_addr_y~34
fb_addr_y~39
fb_addr_y~44
fb_addr_y~7
fb_addr_y~12
fb_addr_y~17
fb_addr_y~22
fb_addr_y~27
fb_addr_y~32
fb_addr_y~37
fb_addr_y~42
memWE1~6
memWE1~7
memWE1~8
memWE1~9
memWE1~10
memWE1~11
memWE1
memWE1~3
memWE1~4
memWE1~5
memWE2~0
memWE2~1
memWE2~2
memWE2~3
memWE2~4
memWE2~5
memWE2~6
memWE2~7
memWE2~8
memWE2
memory:memory
address_a[11..0]
address_b[11..0]
clock
data_a[7..0]
data_b[7..0]
wren_a
wren_b
q_a[7..0]
q_b[7..0]
OUT0
[19,39]
OUT0
[19,28,39]
@OUT011126,OUT011128,OUT011130
[19,39]
q_a[7..0]
[1,6,19,27,31,42]
q_b[7..0]
[6,19,31,42]
1'h0

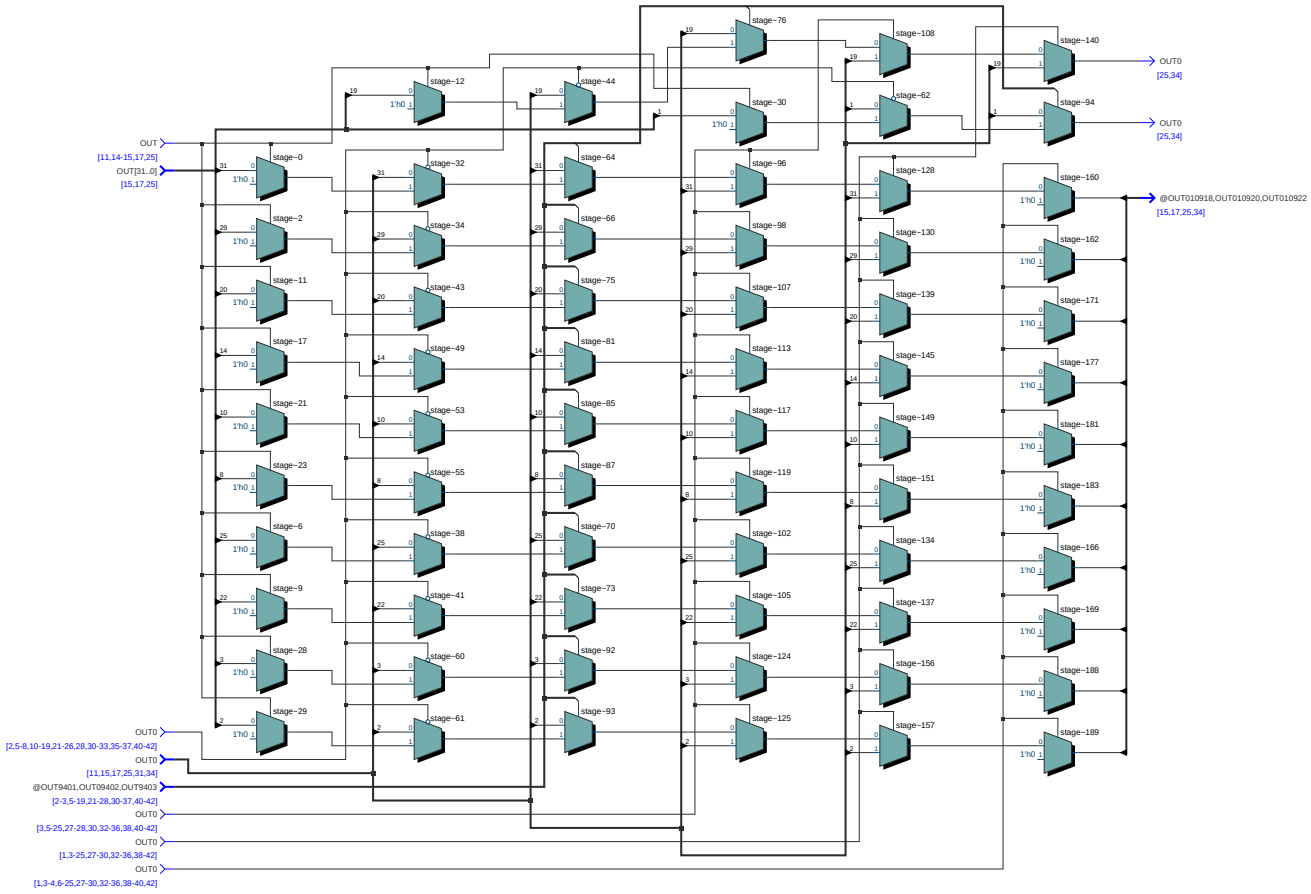

@OUT8030,OUT8028,OUT8027
[1,3-4,7,9-10,19-21,27-29,34,38-39,42]
OUT0
[10,20]
OUT0
[1,3-4,7,9,16,20-21,24,27-29,32,38-40]
OUT0
[9-10,20]
OUT0
[20,42]
OUT0
[10,20,26,42]
OUT0
[20,26]
OUT0
[2,20]
@OUT08031,OUT8026,OUT08085
[1,16,20,24,27,32,34,38,40]
OUT0
[20,26]
OUT0
[2,20]
OUT0
[2,16,20,26-27,32]
OUT0
[3,7,9,16,20-21,24,27-29,32,38-40]
OUT0
[2,20]
OUT0
[2,18,20,24,31,40]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[2,20]
chipselect
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
fbvx_prev-0
fbvx_prev-3
memWE1~0
memWE1~1
OUT0
[10,20]
OUT0
[10,20]
OUT0
[10-20]
fbvx_prev-9
fbvx_prev-15
fbvx_prev-21
@OUT010905,OUT010907,OUT010909
[9-10,20]
fbvx_prev-1
fbvx_prev-7
fbvx_prev-13
fbvx_prev-19
fbvx_prev-4
fb_addr_x-4
fb_addr_x-46
fbvx_prev-22
fbvx_prev-10
fbvx_prev-16
fbvx_prev-20
fbvx_prev-2
fbvx_prev-8
fbvx_prev-14
fb_addr_x-2
OUT0
[10,20]
fb_addr_x-1
fb_addr_x-52
@OUT011114,OUT011116,OUT011118
[10,20,42]
fb_addr_x-37
fb_addr_x-43
fb_addr_x-49
OUT0
[1,3-25,27-30,32-36,38-42]
sound_timer_data-64
sound_timer_data-65
sound_timer_data-72
sound_timer_data-80
@OUT010761,OUT010763,OUT010765
[16,20,32]
sound_timer_data-0
sound_timer_data-8
sound_timer_data-73
sound_timer_data-81
sound_timer_data-1
sound_timer_data-9
sound_timer_data-70
sound_timer_data-78
sound_timer_data-86
sound_timer_data-6
sound_timer_data-14
sound_timer_data-87
sound_timer_data-71
sound_timer_data-79
sound_timer_data-7
sound_timer_data-15
reg_writedata1-1
reg_writedata1-9
reg_writedata1-2
reg_writedata1-10
reg_writedata1-74
reg_writedata1-66
reg_writedata1-73
reg_writedata1-81
@OUT010777,OUT010779,OUT010781
[18,20,24,40]
reg_writedata1-65
reg_writedata1-82

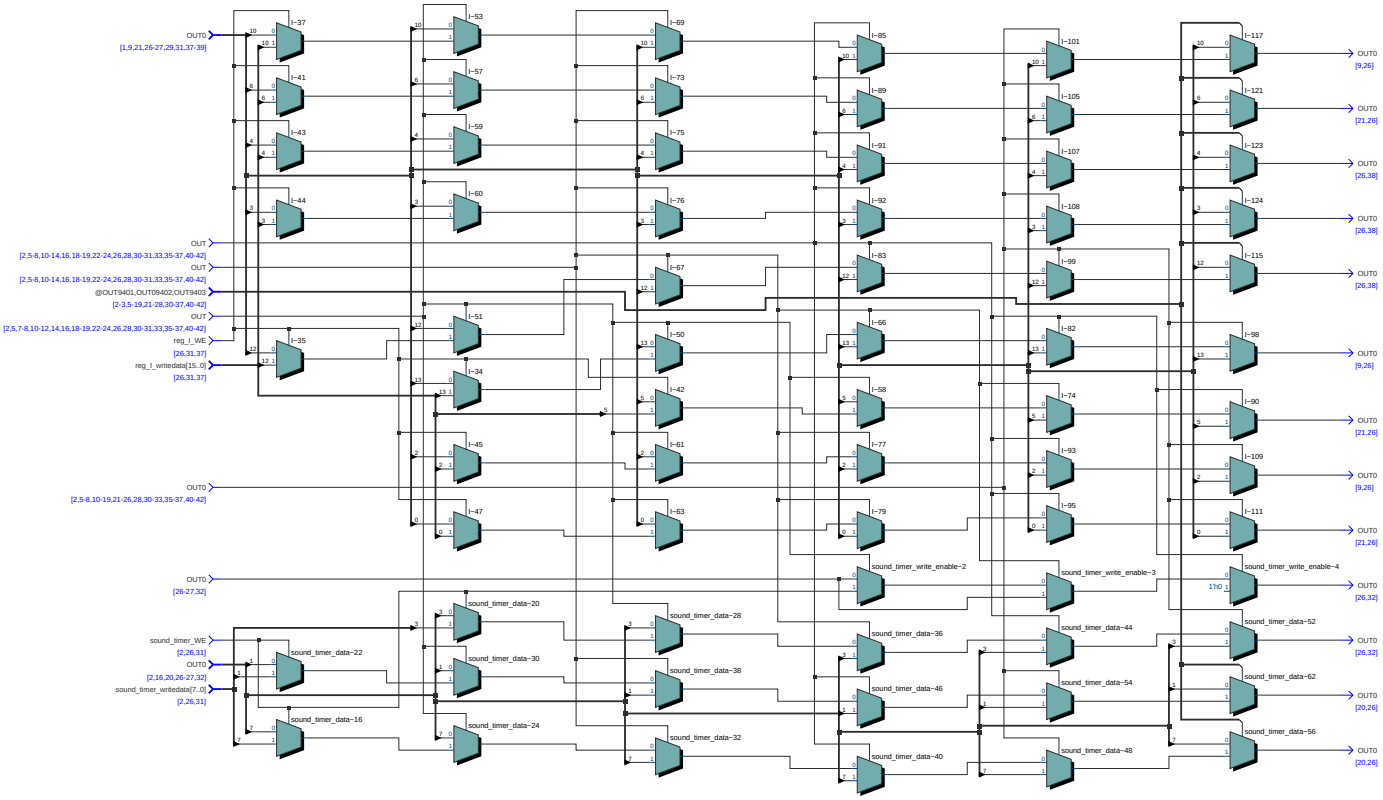

bit_ovewritten~5
bit_ovewritten~6
bit_ovewritten~7
bit_ovewritten
OUT0
[21,31]
OUT0
[4,8-9,11-12,16-17,19,21,24,27-28,31-32,34-35,42]
OUT0
[21,37]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT0
[21,26]
@OUT8030,OUT8028,OUT8027
[1,3-4,7,9-10,19-21,27-29,34,38-39,42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
@OUT011142,OUT011144,OUT011146
[9,21,38]
OUT0
[1,3-4,7,9,16,20-21,24,27-29,32,38-40]
OUT0
[1,9,21,26-27,29,31,37-39]
OUT0
[21,26]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
OUT0
[21,23]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT0
[3,7,9,16,20-21,24,27-29,32,38-40]
OUT0
[7,19,21,23]
OUT0
[21,23]
@OUT010695,OUT010697,OUT010699
[7,21]
clk
OUT0
[21,23]
reset
write
writedata[31..0]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[7,21]
OUT0
[21,23]
I~106
I~112
I~122
I~9
I~10
I~0
I~15
I~127
memwritedata1~48
memwritedata1~56
memwritedata1~4
memwritedata1~0
memwritedata1~5
memwritedata1~2
memwritedata1~3
I~128
I~138
I~25
I~26
I~137
I~16
I~31
I~143
memwritedata1~12
memwritedata1~64
memwritedata1~8
memwritedata1~13
memwritedata1~10
memwritedata1~11
memwritedata1~66
memwritedata1~67
I~153
I~154
I~144
I~159
memwritedata1~76
memwritedata1~72
memwritedata1[7..0]
memwritedata1~77
memwritedata1~74
memwritedata1~75
I~169
I~170
I~160
I~175
I[15..0]
memwritedata1~80
memwritedata1~84
memwritedata1~85
memwritedata1~82
memwritedata1~83

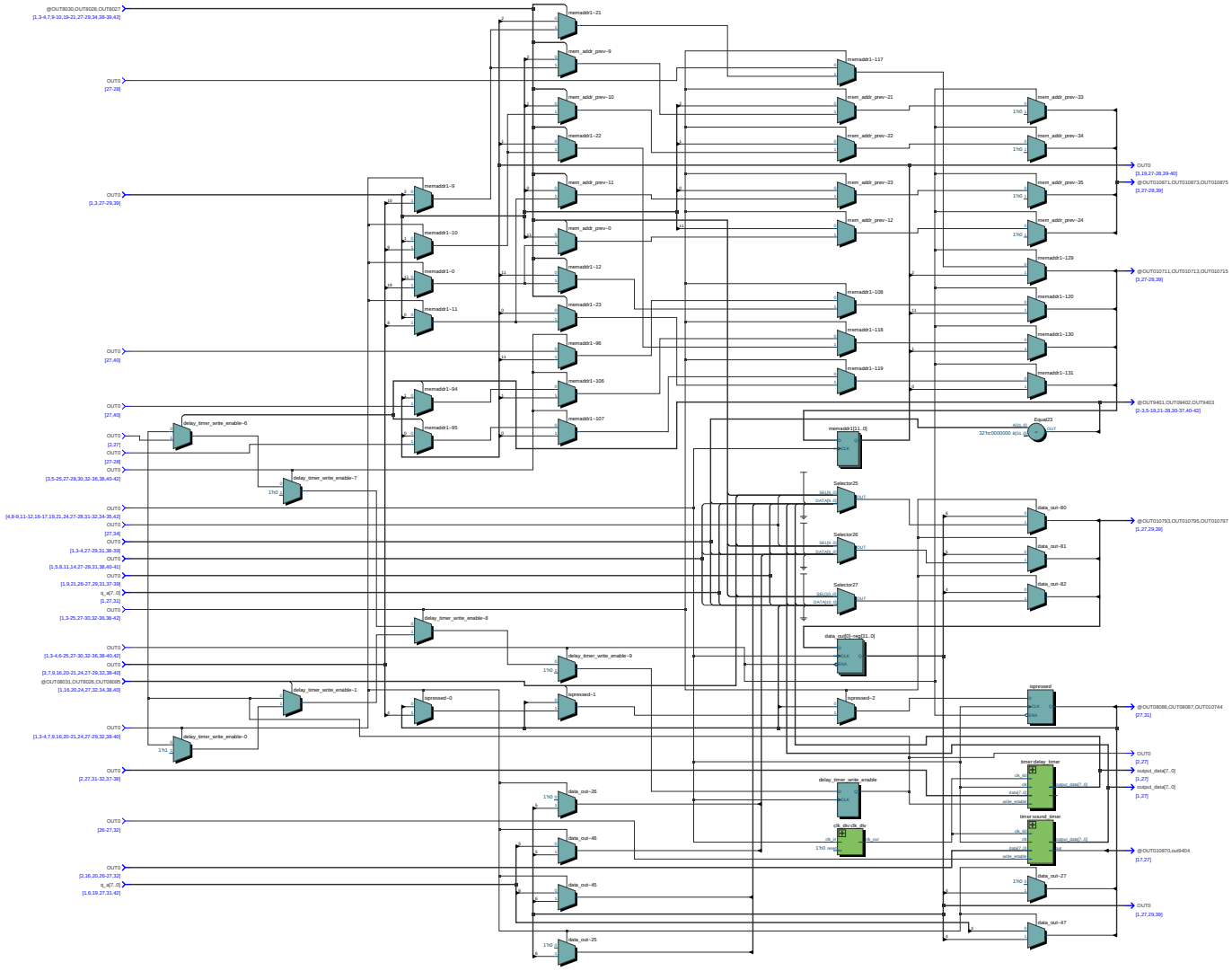

OUT
[11,22,31,33,35-36,43]
OUT
[5,11-12,14,22,31,33,35-36,43]
OUT0
[10-11,13,16,22,33,35-36]
OUT
[11,18,22,33,35-36,43]
OUT
[2,5,7-8,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-43]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-43]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT0
[2,5-8,10-18,21-26,28,30-33,35-37,40-42]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-43]
OUT0
[3,5-25,27-28,30,32-36,38,40-43]
OUT0
[1,3-25,27-30,32-36,38-43]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,43]
stack_op-0
stack_op-14
stack_op-1
stack_op-17
stack_op-27
stack_op-32
stack_op-46
stack_op-33
stack_op-49
stack_op-59
stack_op-64
stack_op-78
stack_op-65
stack_op-81
stack_op-91
stack_op-96
stack_op-97
stack_op-110
stack_op-123
stack_op-113
stack_op-128
stack_op-129
stack_op-142
stack_op-145
stack_op-155
stack_op-160
stack_op-161
stack_op-174
stack_op-177
stack_op-187
stack_op-192
stack_op-193
stack_op-206
stack_op-209
stack_op-219
stack_op-15
stack_op-16
stack_op-26
stack_op-47
stack_op-48
stack_op-58
stack_op-79
stack_op-80
stack_op-90
stack_op-111
stack_op-112
stack_op-122
stack_op-224
stack_op-225
stack_op-238
stack_op-241
stack_op-251
stack_op-256
stack_op-257
stack_op-270
stack_op-273
stack_op-283
stack_op-288
stack_op-289
stack_op-302
stack_op-305
stack_op-315
stack_op-320
stack_op-321
stack_op-334
stack_op-337
stack_op-347
OUT0
[11,22]
OUT0
[10,22]
OUT0
[13,22]
@OUT011045,OUT011048,OUT011050
[10,13,16,22,33,35-36]

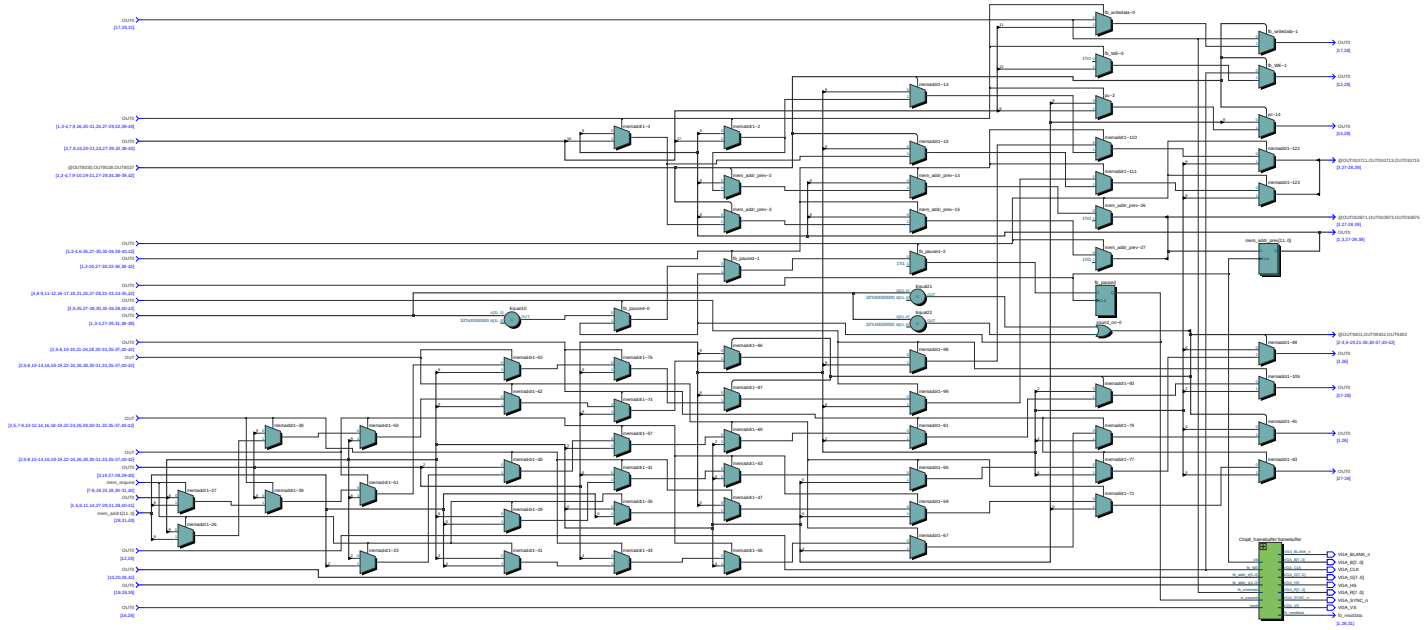

@OUT7401,OUT7402,OUT7403
[2,13,19,21,28,30,37,40,42]
OUT0
[2,5-8,10-19,21-28,28,30-33,35-37,40-42]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5,7-8,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
mem_request
[7-8,19,23,28,30-31,40]
OUT0
[8,19,23]
mem_writedata[27..0]
[8,23,31]
OUT0
[7,19,21,23]
mem_writedata[127..0]
[7,23,31]
memwritedata2~2
memwritedata2~3
memwritedata2~4
memwritedata2~1
memwritedata2~10
memwritedata2~11
memwritedata2~12
memwritedata2~9
memwritedata3~18
memwritedata3~19
memwritedata3~20
memwritedata3~17
memwritedata1~20
memwritedata1~21
memwritedata2~26
memwritedata2~27
memwritedata2~28
memwritedata2~25
memwritedata1~19
memwritedata1~28
memwritedata1~29
memwritedata3~34
memwritedata3~35
memwritedata3~36
memwritedata3~33
memwritedata1~27
memwritedata1~36
memwritedata1~37
memwritedata2~42
memwritedata2~43
memwritedata2~44
memwritedata2~41
memwritedata1~16
memwritedata1~35
memwritedata1~44
memwritedata1~45
memwritedata2~50
memwritedata2~51
memwritedata2~52
memwritedata2~49
memwritedata1~24
memwritedata1~43
memwritedata1~52
memwritedata1~53
memwritedata2~58
memwritedata2~59
memwritedata2~60
memwritedata2~57
memwritedata2~5
memwritedata1~32
memwritedata1~51
memwritedata1~60
memwritedata1~61
memwritedata2~66
memwritedata2~67
memwritedata2~68
memwritedata2~65
memwritedata2~13
memwritedata1~40
memwritedata1~59
memwritedata1~68
memwritedata1~69
@OUT7010602,OUT7010600,OUT7010598
[8,23]
OUT0
[8,23]
OUT0
[21,23]
OUT0
[21,23]
OUT0
[21,23]
OUT0
[21,23]

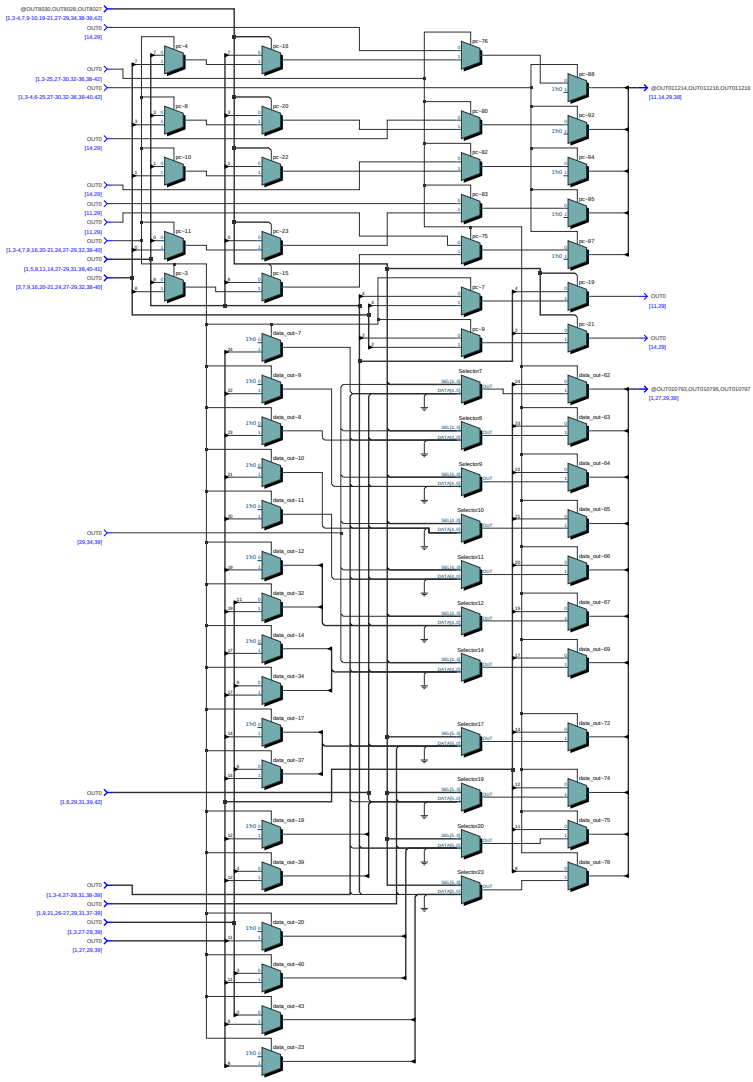


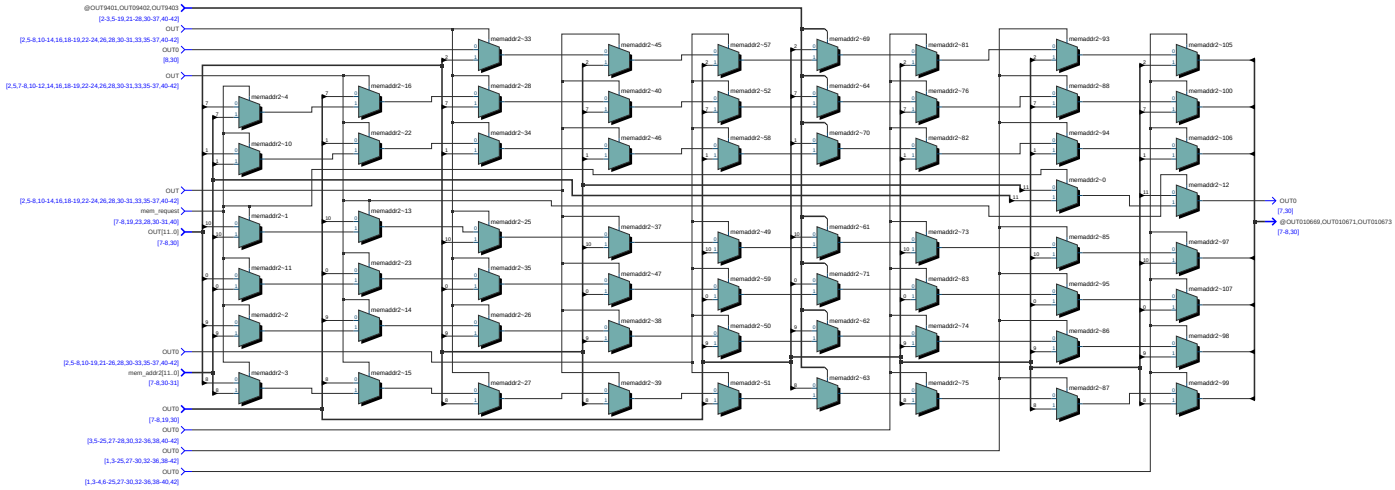

OUT
[11,14-15,17,25]
OUT[31..0]
[15,17,25]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
OUT0
[11,15,17,25,31,34]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT0
[25,34]
OUT0
[25,34]
@OUT010918,OUT010920,OUT010922
[15,17,25,34]

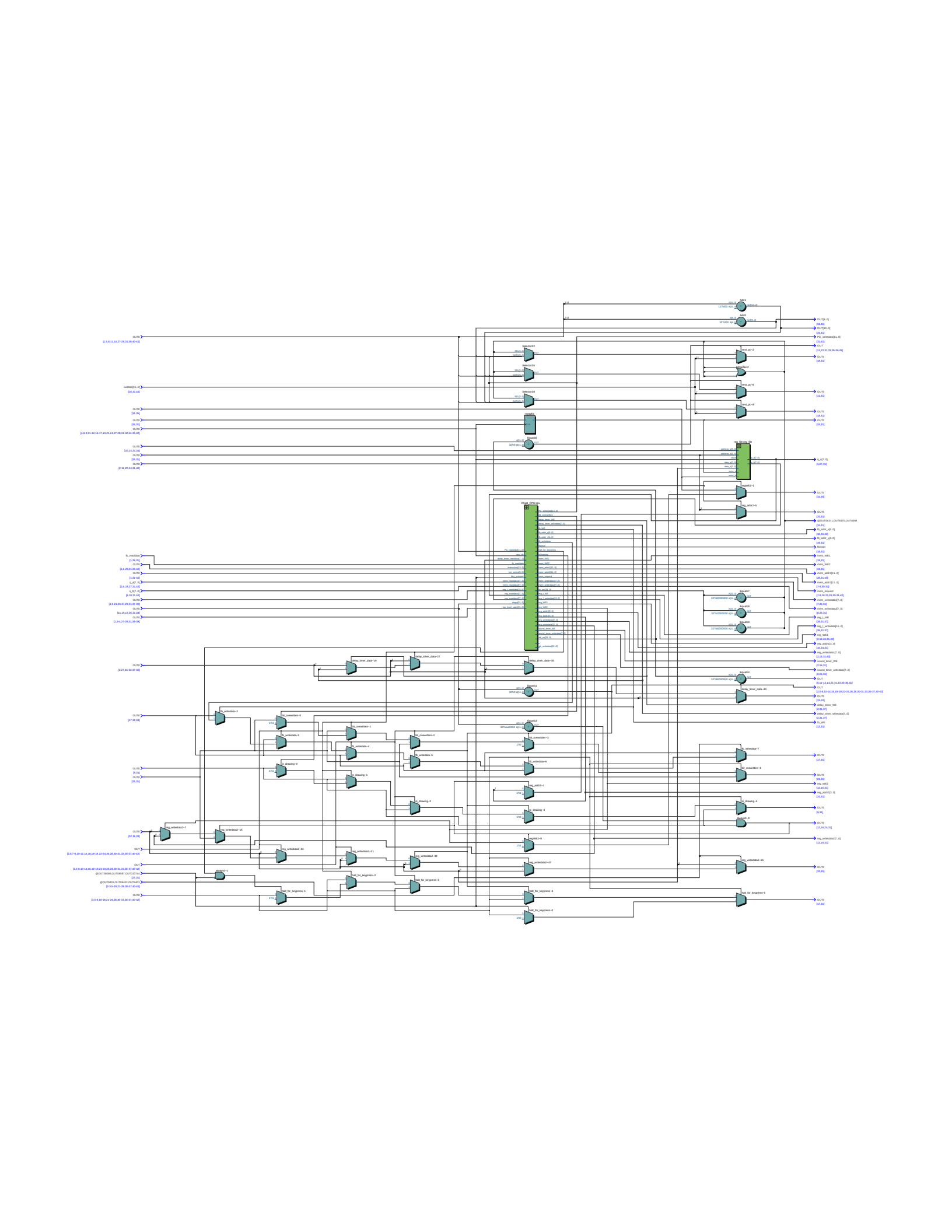

OUT0
[1,9,21,26-27,29,31,37-39]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT
[2,5,7-8,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
reg_I_WE
[26,31,37]
reg_i_writedata[15..0]
[26,31,37]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
OUT0
[26-27,32]
sound_timer_WE
[2,26,31]
OUT0
[2,16,20,26-27,32]
sound_timer_writedata[7..0]
[2,26,31]
I~37
I~41
I~43
I~44
I~35
I~53
I~57
I~59
I~60
I~51
I~34
I~45
I~47
I~69
I~73
I~75
I~76
I~67
I~50
I~42
I~61
I~63
I~85
I~89
I~91
I~92
I~83
I~66
I~58
I~77
I~79
I~101
I~105
I~107
I~108
I~99
I~82
I~74
I~93
I~95
I~117
I~121
I~123
I~124
I~115
I~98
I~90
I~109
I~111
sound_timer_data~22
sound_timer_data~16
sound_timer_data~20
sound_timer_data~30
sound_timer_data~24
sound_timer_data~28
sound_timer_data~38
sound_timer_data~32
sound_timer_write_enable~2
sound_timer_data~36
sound_timer_data~46
sound_timer_data~40
sound_timer_write_enable~3
sound_timer_data~44
sound_timer_data~54
sound_timer_data~48
sound_timer_write_enable~4
sound_timer_data~52
sound_timer_data~62
sound_timer_data~56
OUT0
[9,26]
OUT0
[21,26]
OUT0
[26,38]
OUT0
[26,38]
OUT0
[26,38]
OUT0
[9,26]
OUT0
[21,26]
OUT0
[9,26]
OUT0
[21,26]
OUT0
[26,32]
OUT0
[26,32]
OUT0
[20,26]
OUT0
[20,26]

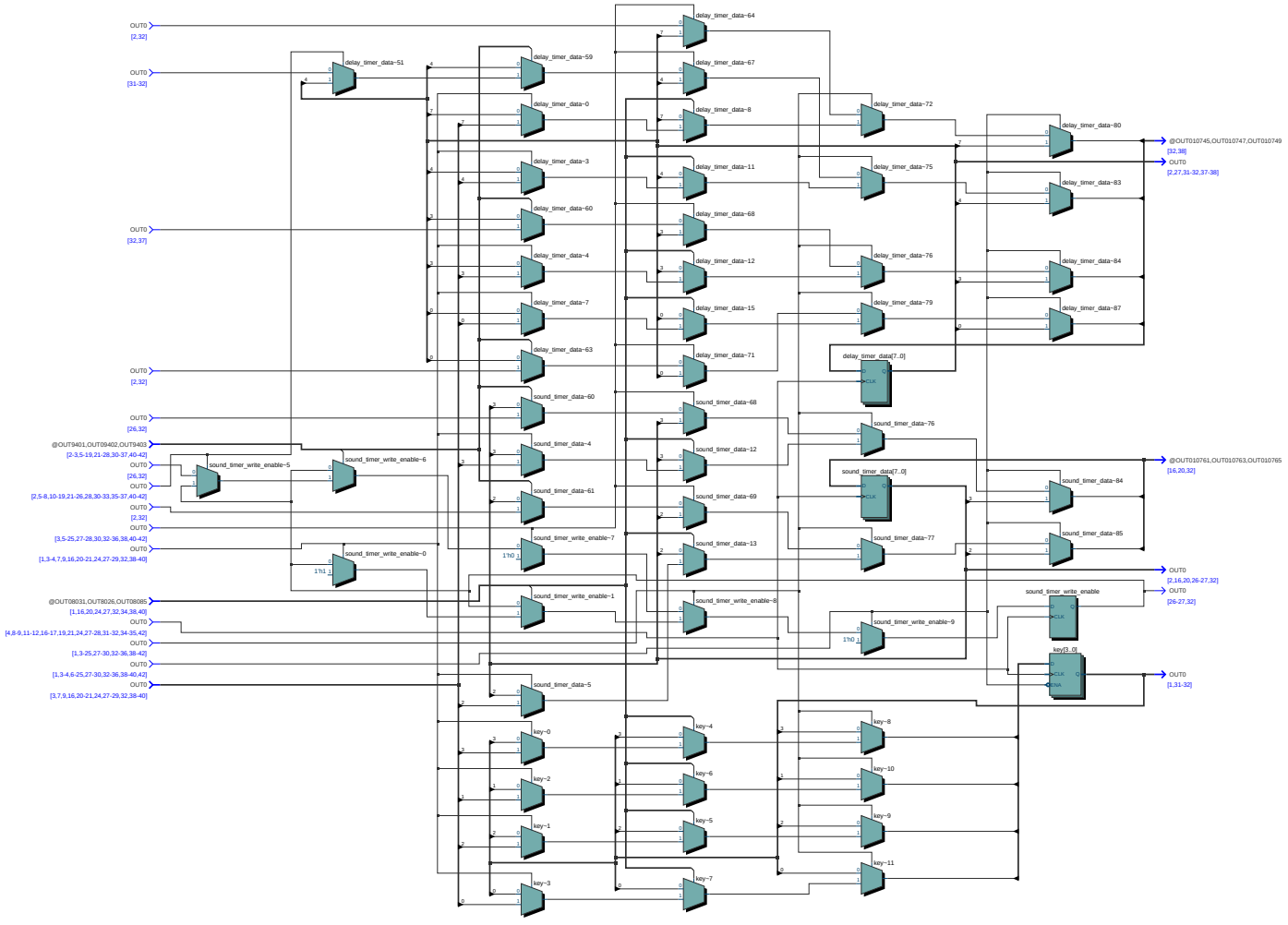


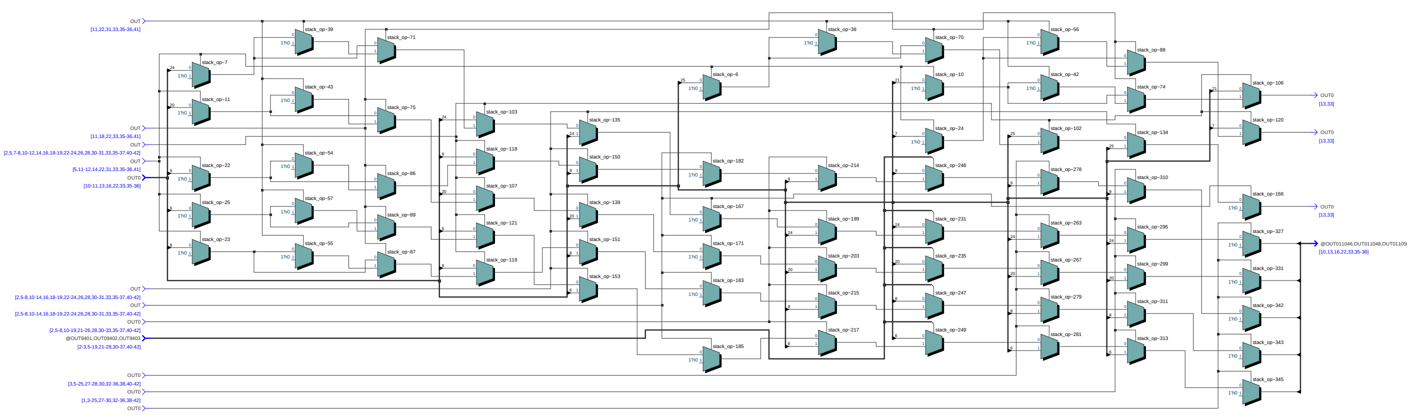


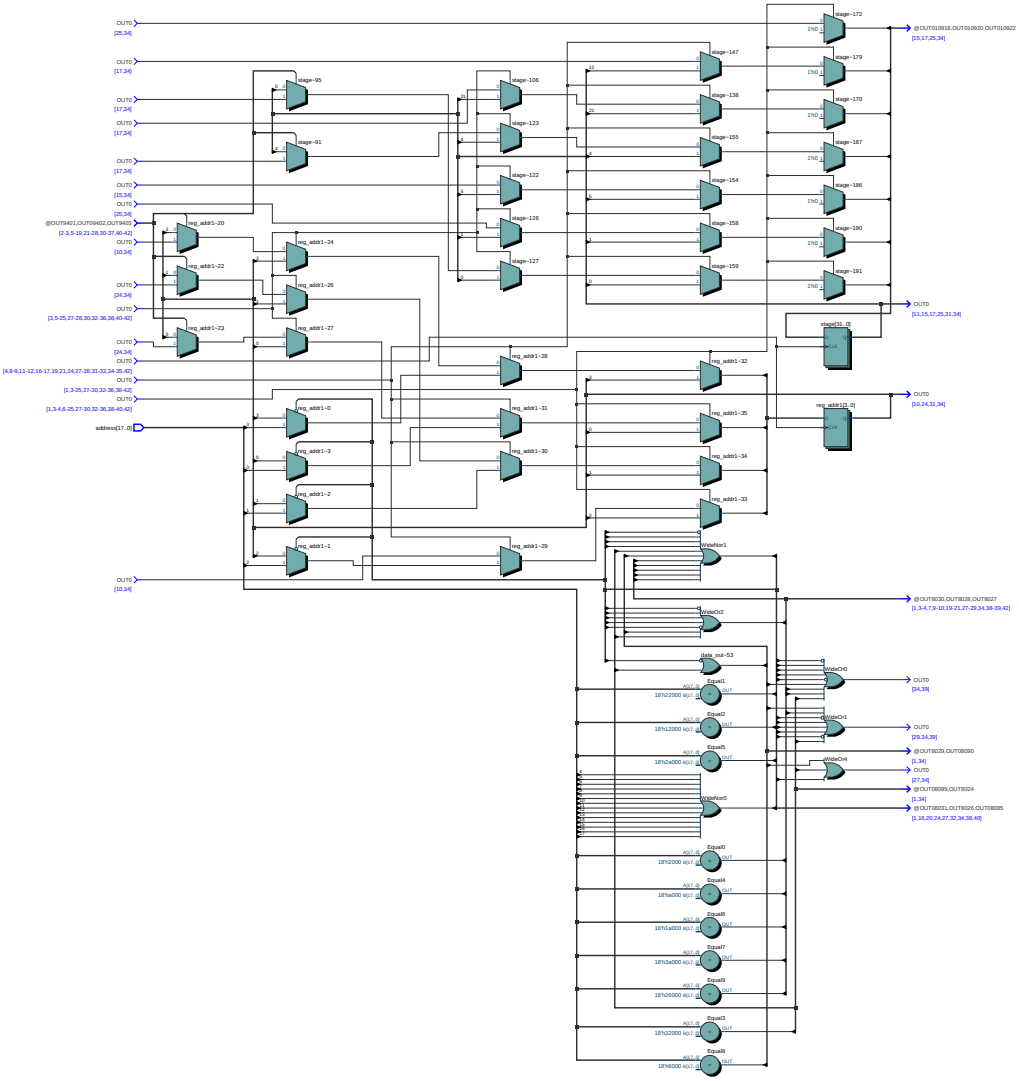

@OUT8030,OUT8028,OUT8027
[1,2-4,7,9-10,19-21,27-29,34,38-39,42]
OUT0
[14,29]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT0
[14,29]
OUT0
[14,29]
OUT0
[11,29]
OUT0
[11,29]
OUT0
[1,3-4,7,9,16,20-21,24,27-29,32,38-40]
OUT0
[1,5,8,11,14,27-29,31,38,40-41]
OUT0
[3,7,9,16,20-21,24,27-29,32,38-40]
OUT0
[29,34,39]
OUT0
[1,6,29,31,39,42]
OUT0
[1,3-4,27-29,31,38-39]
OUT0
[1,9,21,26-27,29,31,37-38]
OUT0
[1,3,27-29,39]
OUT0
[1,27,29,39]
sc-4
sc-16
sc-8
sc-20
sc-10
sc-22
sc-11
sc-23
sc-3
sc-15
sc-76
sc-80
sc-82
sc-83
sc-75
sc-7
sc-9
sc-88
sc-92
sc-94
sc-95
sc-97
sc-19
sc-21
@OUT011214,OUT011216,OUT011218
[11,14,29,38]
OUT0
[11,29]
OUT0
[14,29]
data_out~7
data_out~9
data_out~8
data_out~10
data_out~11
data_out~12
data_out~32
data_out~14
data_out~34
data_out~17
data_out~37
data_out~19
data_out~39
data_out~20
data_out~40
data_out~43
data_out~23
Selector7
Selector8
Selector9
Selector10
Selector11
Selector12
Selector14
Selector17
Selector19
Selector20
Selector23
data_out~62
data_out~63
data_out~64
data_out~65
data_out~66
data_out~67
data_out~69
data_out~72
data_out~74
data_out~75
data_out~78
@OUT010793,OUT010795,OUT010797
[1,27,29,39]

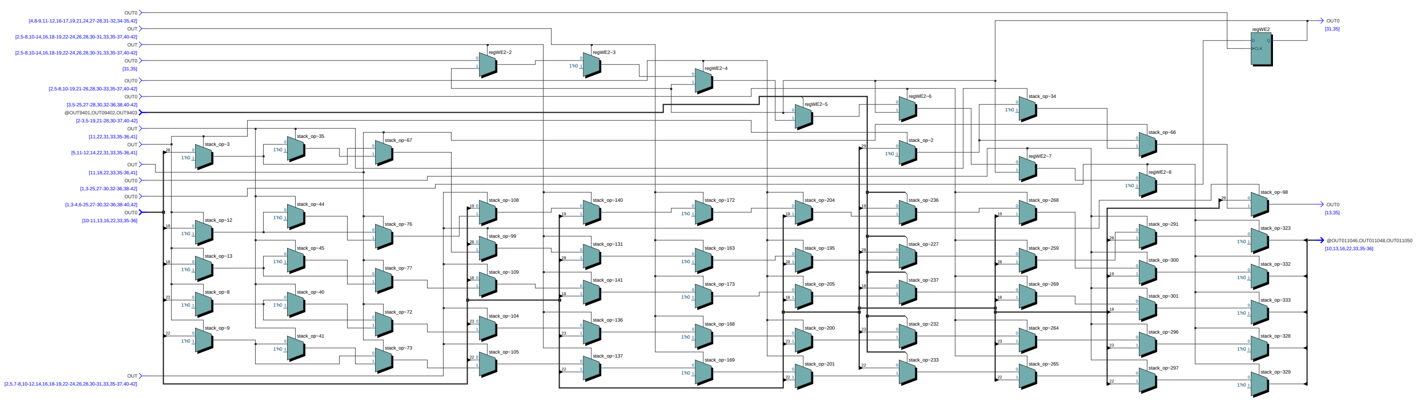

@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT0
[8,30]
OUT
[2,5,7-8,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
mem_request
[7-8,19,23,28,30-31,40]
OUT[11:0]
[7-8,30]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
mem_addr2[11:0]
[7-8,30-31]
OUT0
[7-8,19,30]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
memaddr2~4
memaddr2~10
memaddr2~1
memaddr2~11
memaddr2~2
memaddr2~3
memaddr2~16
memaddr2~22
memaddr2~13
memaddr2~23
memaddr2~14
memaddr2~15
memaddr2~33
memaddr2~28
memaddr2~34
memaddr2~25
memaddr2~35
memaddr2~26
memaddr2~27
memaddr2~45
memaddr2~40
memaddr2~46
memaddr2~37
memaddr2~47
memaddr2~38
memaddr2~39
memaddr2~57
memaddr2~52
memaddr2~58
memaddr2~49
memaddr2~59
memaddr2~50
memaddr2~51
memaddr2~69
memaddr2~64
memaddr2~70
memaddr2~61
memaddr2~71
memaddr2~62
memaddr2~63
memaddr2~81
memaddr2~76
memaddr2~82
memaddr2~73
memaddr2~83
memaddr2~74
memaddr2~75
memaddr2~93
memaddr2~88
memaddr2~94
memaddr2~0
memaddr2~85
memaddr2~95
memaddr2~86
memaddr2~87
memaddr2~105
memaddr2~100
memaddr2~106
memaddr2~12
memaddr2~97
memaddr2~107
memaddr2~98
memaddr2~99
OUT0
[7,30]
@OUT010669,OUT010671,OUT010673
[7-8,30]

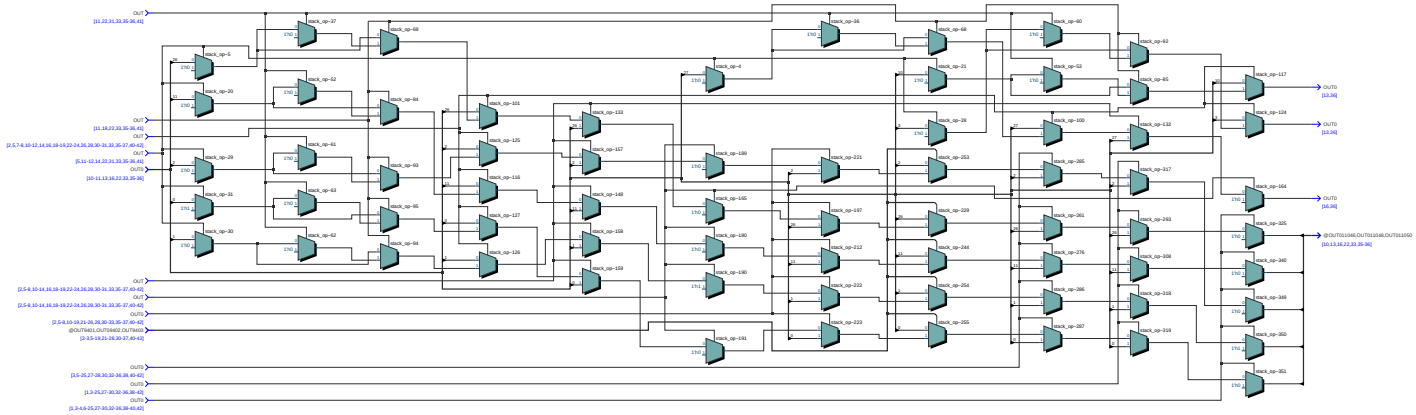


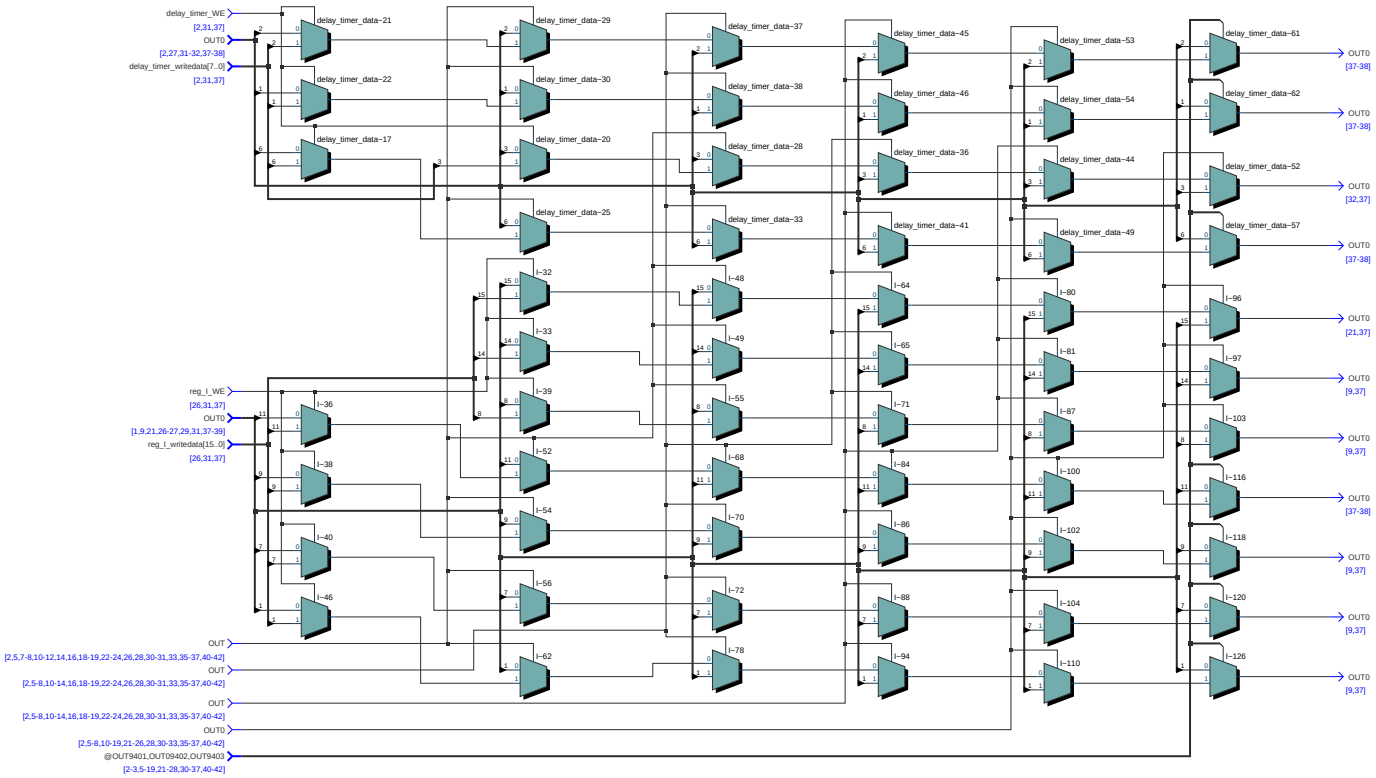

OUT0
[2,32]
OUT0
[31-32]
OUT0
[32,37]
OUT0
[2,32]
OUT0
[26,32]
@OUT09401,OUT09402,OUT09403
[2-3,5-19,21-28,30-37,40-42]
OUT0
[26,32]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
OUT0
[2,32]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[1,3-4,7,9,16,20-21,24,27-29,32,38-40]
@OUT08031,OUT8026,OUT08085
[1,16,20,24,27,32,34,38,40]
OUT0
[4,8-9,11-12,16-17,19,21,24,27-28,31-32,34-35,42]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT0
[3,7,9,16,20-21,24,27-29,32,38-40]
delay_timer_data~51
delay_timer_data~59
delay_timer_data~0
delay_timer_data~3
delay_timer_data~60
delay_timer_data~4
delay_timer_data~7
delay_timer_data~63
sound_timer_data~60
sound_timer_data~4
sound_timer_data~61
sound_timer_write_enable~7
sound_timer_write_enable~1
sound_timer_data~5
key~0
key~2
key~1
key~3
sound_timer_write_enable~5
sound_timer_write_enable~6
sound_timer_write_enable~0
delay_timer_data~64
delay_timer_data~67
delay_timer_data~8
delay_timer_data~11
delay_timer_data~68
delay_timer_data~12
delay_timer_data~15
delay_timer_data~71
sound_timer_data~68
sound_timer_data~12
sound_timer_data~69
sound_timer_data~13
sound_timer_write_enable~8
key~4
key~6
key~5
key~7
delay_timer_data~72
delay_timer_data~75
delay_timer_data~76
delay_timer_data~79
delay_timer_data[7..0]
sound_timer_data~76
sound_timer_data[7..0]
sound_timer_data~77
sound_timer_write_enable~9
key~8
key~10
key~9
key~11
delay_timer_data~80
delay_timer_data~83
delay_timer_data~84
delay_timer_data~87
@OUT010745,OUT010747,OUT010749
[32,38]
OUT0
[2,27,31-32,37-38]
sound_timer_data~84
sound_timer_data~85
@OUT010761,OUT010763,OUT010765
[16,20,32]
OUT0
[2,16,20,26-27,32]
sound_timer_write_enable
OUT0
[26-27,32]
key[3..0]
OUT0
[1,31-32]

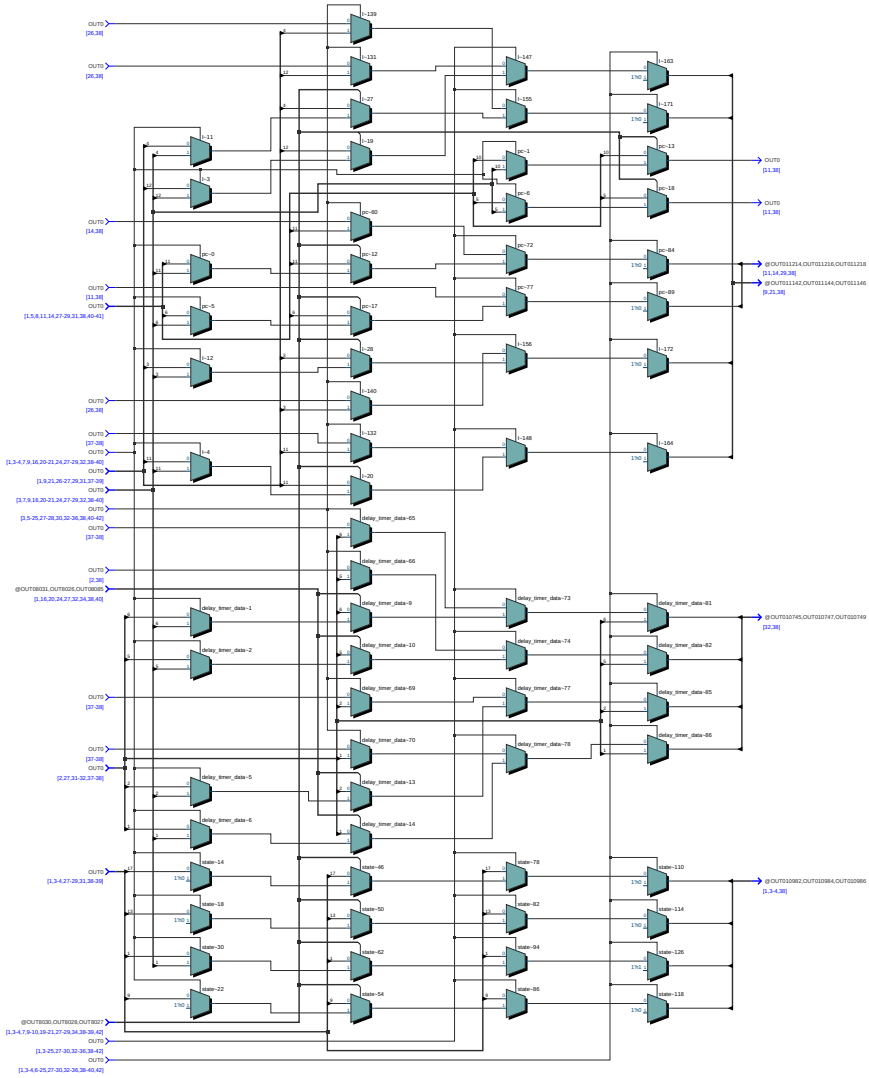

OUT
[11,22,31,33,35-36,41]
OUT
[11,18,22,33,35-36,41]
OUT
[2,5,7-8,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[5,11-12,14,22,31,33,35-36,41]
OUT0
[10-11,13,16,22,33,35-36]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
stack_op~39
stack_op~71
stack_op~7
stack_op~11
stack_op~43
stack_op~75
stack_op~22
stack_op~54
stack_op~86
stack_op~25
stack_op~57
stack_op~89
stack_op~23
stack_op~55
stack_op~87
stack_op~103
stack_op~118
stack_op~107
stack_op~121
stack_op~119
stack_op~135
stack_op~150
stack_op~139
stack_op~151
stack_op~153
stack_op~6
stack_op~182
stack_op~167
stack_op~171
stack_op~183
stack_op~185
stack_op~38
stack_op~214
stack_op~199
stack_op~203
stack_op~215
stack_op~217
stack_op~70
stack_op~10
stack_op~24
stack_op~246
stack_op~231
stack_op~235
stack_op~247
stack_op~249
stack_op~56
stack_op~42
stack_op~102
stack_op~278
stack_op~263
stack_op~267
stack_op~279
stack_op~281
stack_op~88
stack_op~74
stack_op~134
stack_op~310
stack_op~295
stack_op~299
stack_op~311
stack_op~313
stack_op~106
stack_op~120
stack_op~166
stack_op~327
stack_op~331
stack_op~342
stack_op~343
stack_op~345
1'h0
OUT0
[13,33]
OUT0
[13,33]
OUT0
[13,33]
@OUT011046,OUT011048,OUT011050
[10,13,16,22,33,35-36]

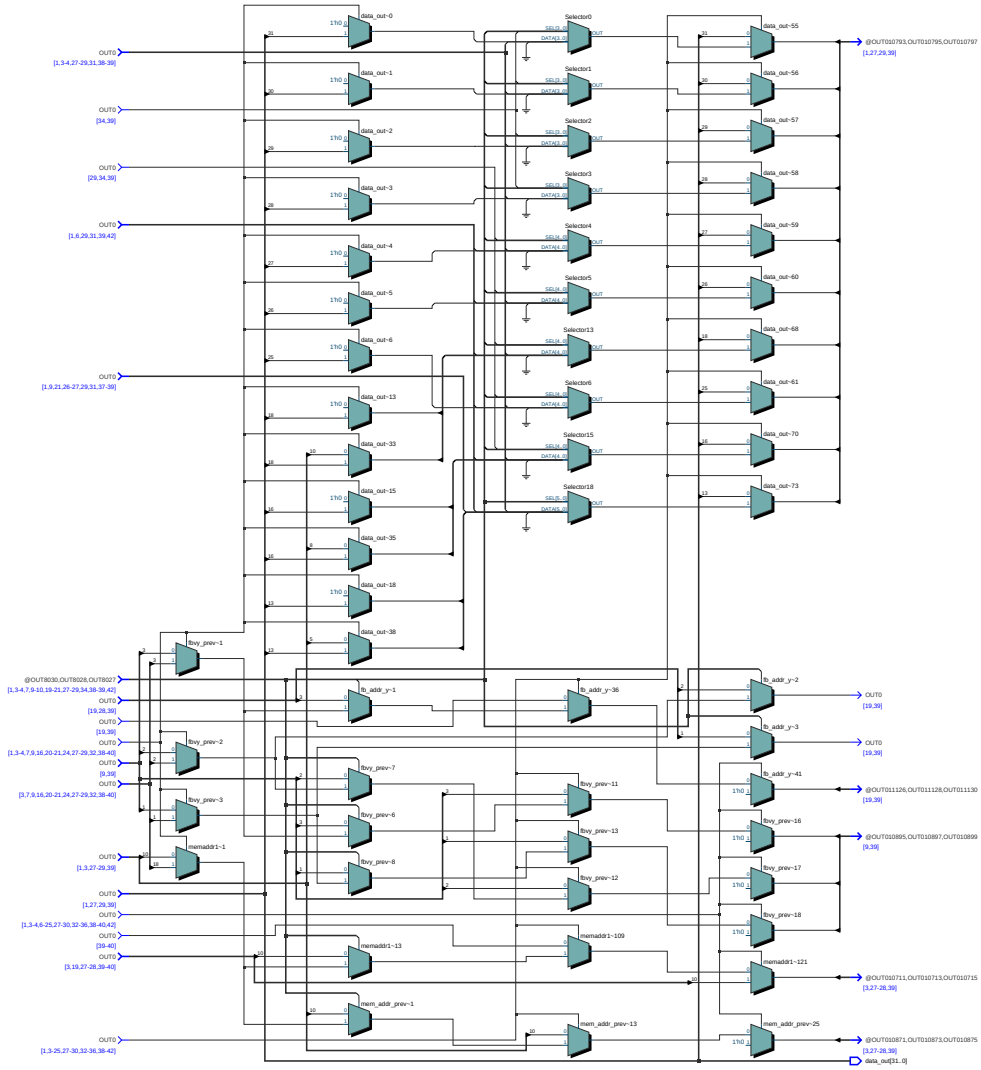

OUT0
[25,34]
OUT0
[17,34]
OUT0
[17,34]
OUT0
[17,34]
OUT0
[17,34]
OUT0
[15,34]
OUT0
[25,34]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT0
[10,34]
OUT0
[24,34]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[24,34]
OUT0
[4,6-9,11-12,16-17,19,21,24,27-28,31-32,34-35,42]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
address[17..0]
OUT0
[10,34]
stage~95
stage~91
stage~106
stage~123
stage~122
stage~126
stage~127
req_addr1~20
req_addr1~24
req_addr1~22
req_addr1~26
req_addr1~23
req_addr1~27
req_addr1~28
req_addr1~31
req_addr1~30
req_addr1~29
req_addr1~0
req_addr1~3
req_addr1~2
req_addr1~1
stage~147
stage~138
stage~155
stage~154
stage~158
stage~159
stage~172
stage~179
stage~170
stage~187
stage~186
stage~190
stage~191
@OUT010918,OUT010920,OUT010922
[15,17,25,34]
OUT0
[11,15,17,25,31,34]
stage[31..0]
req_addr1~32
req_addr1~35
req_addr1~34
req_addr1~33
req_addr1[31..0]
OUT0
[10,24,31,34]
WideNor1
@OUT8030,OUT8028,OUT8027
[1,3-4,7,9-10,19-21,27-29,34,38-39,42]
WideOr2
data_out~53
WideOr0
OUT0
[34,39]
Equal1
A[17..0]
18'h20000
Equal2
18'h12000
WideOr1
OUT0
[29,34,39]
Equal5
18'h2a000
@OUT8029,OUT08090
[1,34]
WideOr4
OUT0
[27,34]
@OUT08099,OUT8024
[1,34]
WideNor0
@OUT08031,OUT8026,OUT08085
[1,16,20,24,27,32,34,38,40]
Equal0
18'h2000
Equal4
18'ha000
Equal6
18'h1a000
Equal7
18'h3a000
Equal9
18'h26000
Equal3
18'h32000
Equal8
18'h6000

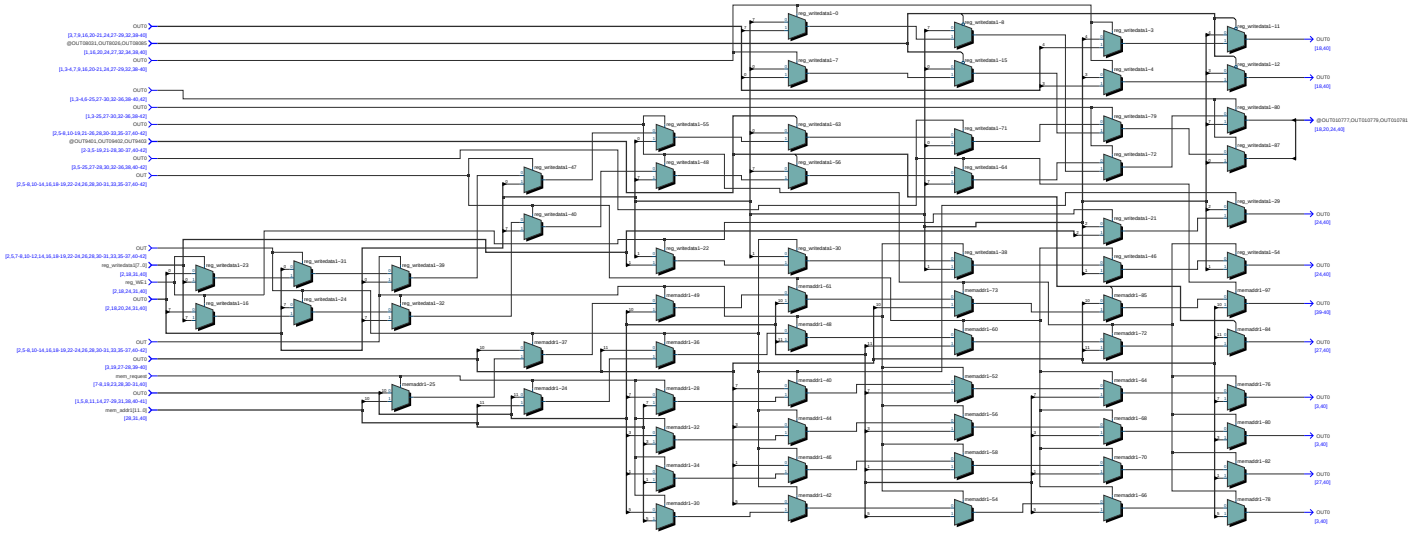

OUT0
[4,8-9,11-12,16-17,19,21,24,27-28,31-32,34-35,42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT0
[31,35]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT
[11,22,31,33,35-36,41]
OUT
[5,11-12,14,22,31,33,35-36,41]
OUT
[11,18,22,33,35-36,41]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT0
[10-11,13,16,22,33,35-36]
OUT
[2,5,7-8,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
regWE2~2
regWE2~3
regWE2~4
regWE2~5
regWE2~6
regWE2~7
regWE2~8
regWE2
D
Q
CLK
stack_op~3
stack_op~35
stack_op~67
stack_op~12
stack_op~44
stack_op~76
stack_op~13
stack_op~45
stack_op~77
stack_op~8
stack_op~40
stack_op~72
stack_op~9
stack_op~41
stack_op~73
stack_op~108
stack_op~99
stack_op~109
stack_op~104
stack_op~105
stack_op~140
stack_op~131
stack_op~141
stack_op~136
stack_op~137
stack_op~172
stack_op~163
stack_op~173
stack_op~168
stack_op~169
stack_op~204
stack_op~195
stack_op~205
stack_op~200
stack_op~201
stack_op~2
stack_op~236
stack_op~227
stack_op~237
stack_op~232
stack_op~233
stack_op~34
stack_op~268
stack_op~259
stack_op~269
stack_op~264
stack_op~265
stack_op~66
stack_op~291
stack_op~300
stack_op~301
stack_op~296
stack_op~297
stack_op~98
stack_op~323
stack_op~332
stack_op~333
stack_op~328
stack_op~329
1'h0
OUT0
[31,35]
OUT0
[13,35]
@OUT011046,OUT011048,OUT011050
[10,13,16,22,33,35-36]

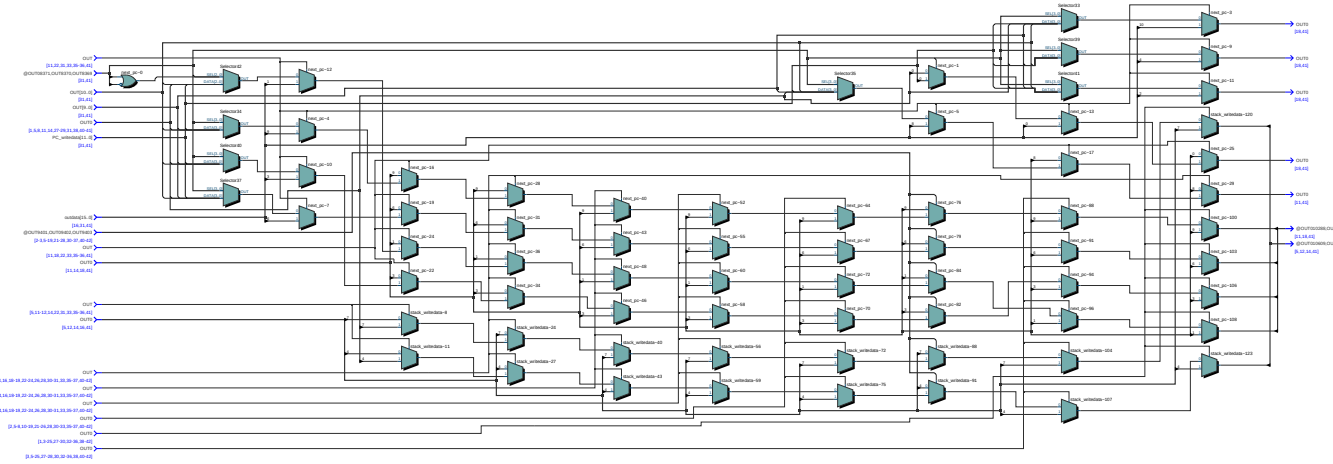

OUT
[11,22,31,33,35-36,41]
OUT
[11,18,22,33,35-36,43]
OUT
[2,5,7-8,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-43]
OUT
[5,11-12,14,22,31,33,35-36,43]
OUT
[10-11,13,16,22,31,35-36]
OUT
[2,5-6,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-43]
OUT
[2,5-6,10-14,16,18-19,22-24,26,28,30-31,33,35-37,43-43]
OUT
[2,5-6,10-19,21-26,28,30-33,35-37,40-43]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-43]
OUT
[1,5-25,27-28,30,32-36,38,40-43]
OUT
[1,3-25,27-30,32-36,38-43]
OUT
[1,3-4,6-25,27-30,32-36,38-43,43]
stack_op-27
stack_op-69
stack_op-5
stack_op-20
stack_op-52
stack_op-84
stack_op-101
stack_op-133
stack_op-4
stack_op-36
stack_op-68
stack_op-60
stack_op-92
stack_op-117
OUT0
[13,36]
stack_op-41
stack_op-93
stack_op-125
stack_op-157
stack_op-189
stack_op-21
stack_op-53
stack_op-85
stack_op-124
OUT0
[13,36]
stack_op-29
stack_op-116
stack_op-148
stack_op-28
stack_op-100
stack_op-132
stack_op-43
stack_op-95
stack_op-127
stack_op-158
stack_op-165
stack_op-221
stack_op-253
stack_op-285
stack_op-317
stack_op-164
OUT0
[16,36]
stack_op-31
stack_op-42
stack_op-94
stack_op-126
stack_op-159
stack_op-180
stack_op-197
stack_op-229
stack_op-261
stack_op-293
stack_op-325
@OUT0710461,OUT0710481,OUT0710503
[10,13,16,22,33,35-36]
stack_op-30
stack_op-190
stack_op-212
stack_op-244
stack_op-276
stack_op-308
stack_op-340
stack_op-191
stack_op-222
stack_op-254
stack_op-286
stack_op-318
stack_op-349
stack_op-223
stack_op-255
stack_op-287
stack_op-319
stack_op-350
stack_op-351

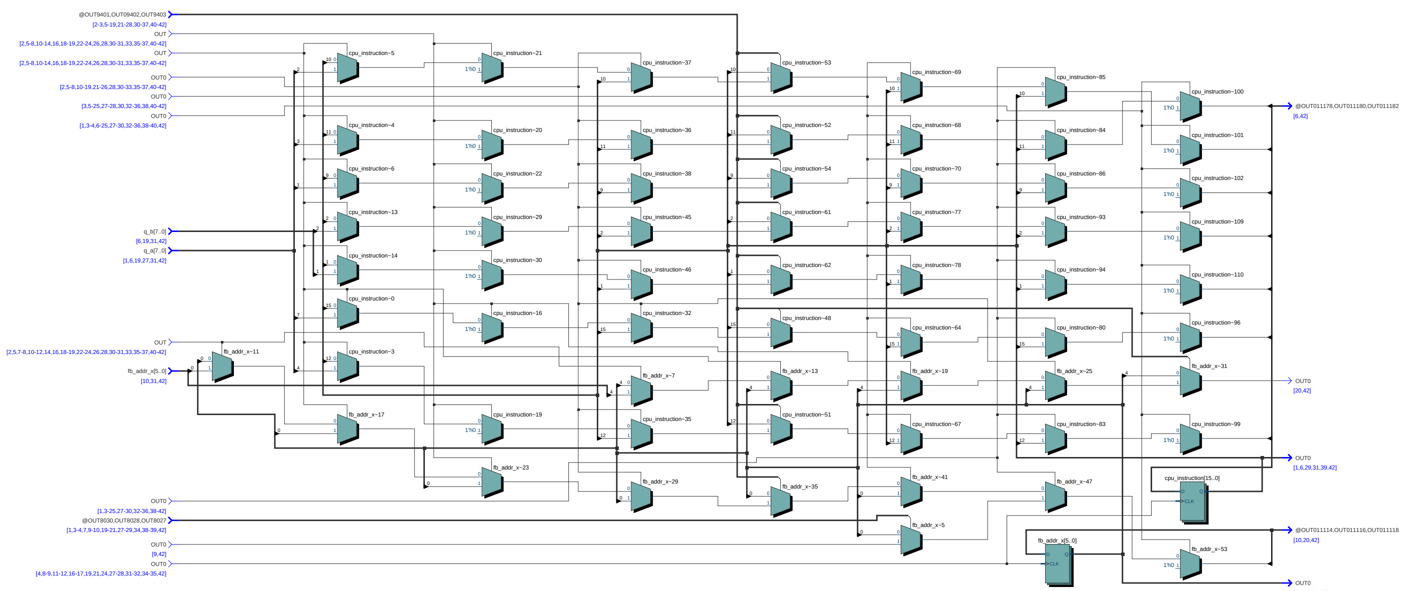

delay_timer_WE
[2,31,37]
OUT0
[2,27,31-32,37-38]
delay_timer_writedata[7..0]
[2,31,37]
delay_timer_data~21
delay_timer_data~22
delay_timer_data~17
delay_timer_data~29
delay_timer_data~30
delay_timer_data~20
delay_timer_data~25
delay_timer_data~37
delay_timer_data~38
delay_timer_data~28
delay_timer_data~33
delay_timer_data~45
delay_timer_data~46
delay_timer_data~36
delay_timer_data~41
delay_timer_data~53
delay_timer_data~54
delay_timer_data~44
delay_timer_data~49
delay_timer_data~61
delay_timer_data~62
delay_timer_data~52
delay_timer_data~57
OUT0
[37-38]
OUT0
[37-38]
OUT0
[32,37]
OUT0
[37-38]
reg_l_WE
[26,31,37]
OUT0
[1,9,21,26-27,29,31,37-39]
reg_l_writedata[15..0]
[26,31,37]
I~36
I~38
I~40
I~46
I~32
I~33
I~39
I~52
I~54
I~56
I~62
I~48
I~49
I~55
I~68
I~70
I~72
I~78
I~64
I~65
I~71
I~84
I~86
I~88
I~94
I~80
I~81
I~87
I~100
I~102
I~104
I~110
I~96
I~97
I~103
I~116
I~118
I~120
I~126
OUT0
[21,37]
OUT0
[9,37]
OUT0
[9,37]
OUT0
[37-38]
OUT0
[9,37]
OUT0
[9,37]
OUT0
[9,37]
OUT
[2,5,7-8,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]


OUT0
[26,38]
OUT0
[26,38]
OUT0
[14,38]
OUT0
[11,38]
OUT0
[1,5,8,11,14,27-29,31,38,40-41]
OUT0
[26,38]
OUT0
[37-38]
OUT0
[1,3-4,7,9,16,20-21,24,27-29,32,38-40]
OUT0
[1,9,21,26-27,29,31,37-38]
OUT0
[1,7,9,16,20-21,24,27-29,32,38-40]
OUT0
[1,5-25,27-28,30,32-36,38,40-42]
OUT0
[37-38]
OUT0
[2,38]
@OUT08031,OUT8026,OUT08035
[1,16,20,24,27,32,34,38,40]
OUT0
[37-38]
OUT0
[37-38]
OUT0
[2,27,31-32,37-38]
OUT0
[1,3-4,27-29,31,38-39]
@OUT8030,OUT8028,OUT8027
[1,3-4,7,9-10,19-21,27-29,34,38-39,42]
OUT0
[1,3-25,27-30,32-36,38-42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT0
[11,38]
OUT0
[11,38]
@OUT011214,OUT011216,OUT011218
[11,14,29,38]
@OUT011142,OUT011144,OUT011146
[9,21,38]
@OUT010745,OUT010747,OUT010749
[32,38]
@OUT010982,OUT010984,OUT010986
[1,3-4,38]


data_out~0
data_out~1
data_out~2
data_out~3
data_out~4
data_out~5
data_out~6
data_out~13
data_out~33
data_out~15
data_out~35
data_out~18
data_out~38
data_out~55
data_out~56
data_out~57
data_out~58
data_out~59
data_out~60
data_out~68
data_out~61
data_out~70
data_out~73
Selector0
Selector1
Selector2
Selector3
Selector4
Selector5
Selector13
Selector6
Selector15
Selector18
fbvy_prev~1
fb_addr_y~1
fbvy_prev~2
fbvy_prev~7
fbvy_prev~3
fbvy_prev~6
memaddr1~1
fbvy_prev~8
memaddr1~13
mem_addr_prev~1
fb_addr_y~36
fbvy_prev~11
fbvy_prev~13
fbvy_prev~12
memaddr1~109
mem_addr_prev~13
fb_addr_y~2
fb_addr_y~3
fb_addr_y~41
fbvy_prev~16
fbvy_prev~17
fbvy_prev~18
memaddr1~121
mem_addr_prev~25
OUT0
[1,3-4,27-29,31,38-39]
OUT0
[34,39]
OUT0
[29,34,39]
OUT0
[1,6,29,31,39,42]
OUT0
[1,9,21,26-27,29,31,37-39]
@OUT8030,OUT8028,OUT8027
[1,3-4,7,9-10,19-21,27-29,34,38-39,42]
OUT0
[19,28,39]
OUT0
[19,39]
OUT0
[1,3-4,7,9,16,20-21,24,27-29,32,38-40]
OUT0
[9,39]
OUT0
[3,7,9,16,20-21,24,27-29,32,38-40]
OUT0
[1,3,27-29,39]
OUT0
[1,27,29,39]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
OUT0
[39,40]
OUT0
[3,19,27-28,39-40]
OUT0
[1,3-25,27-30,32-36,38-42]
@OUT010793,OUT010795,OUT010797
[1,27,29,39]
OUT0
[19,39]
OUT0
[19,39]
@OUT011126,OUT011128,OUT011130
[19,39]
@OUT010895,OUT010897,OUT010899
[9,39]
@OUT010711,OUT010713,OUT010715
[3,27-28,39]
@OUT010871,OUT010873,OUT010875
[3,27-28,39]
data_out[31..0]






@OUT9401,OUT09402,OUT9403
[2-3,5-19,21-28,30-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT
[2,5-8,10-14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
OUT0
[2,5-8,10-19,21-26,28,30-33,35-37,40-42]
OUT0
[3,5-25,27-28,30,32-36,38,40-42]
OUT0
[1,3-4,6-25,27-30,32-36,38-40,42]
q_b[7..0]
[6,19,31,42]
q_a[7..0]
[1,6,19,27,31,42]
OUT
[2,5,7-8,10-12,14,16,18-19,22-24,26,28,30-31,33,35-37,40-42]
fb_addr_x[5..0]
[10,31,42]
OUT0
[1,3-25,27-30,32-36,38-42]
@OUT8030,OUT8028,OUT8027
[1,3-4,7,9-10,19-21,27-29,34,38-39,42]
OUT0
[9,42]
OUT0
[4,8-9,11-12,16-17,19,21,24,27-28,31-32,34-35,42]
cpu_instruction~5
cpu_instruction~4
cpu_instruction~6
cpu_instruction~13
cpu_instruction~14
cpu_instruction~0
cpu_instruction~3
fb_addr_x~11
fb_addr_x~17
cpu_instruction~21
cpu_instruction~20
cpu_instruction~22
cpu_instruction~29
cpu_instruction~30
cpu_instruction~16
cpu_instruction~19
fb_addr_x~23
cpu_instruction~37
cpu_instruction~36
cpu_instruction~38
cpu_instruction~45
cpu_instruction~46
cpu_instruction~32
fb_addr_x~7
cpu_instruction~35
fb_addr_x~29
cpu_instruction~53
cpu_instruction~52
cpu_instruction~54
cpu_instruction~61
cpu_instruction~62
cpu_instruction~48
fb_addr_x~13
cpu_instruction~51
fb_addr_x~35
cpu_instruction~69
cpu_instruction~68
cpu_instruction~70
cpu_instruction~77
cpu_instruction~78
cpu_instruction~64
fb_addr_x~19
cpu_instruction~67
fb_addr_x~41
fb_addr_x~5
cpu_instruction~85
cpu_instruction~84
cpu_instruction~86
cpu_instruction~93
cpu_instruction~94
cpu_instruction~80
fb_addr_x~25
cpu_instruction~83
fb_addr_x~47
fb_addr_x[5..0]
cpu_instruction~100
cpu_instruction~101
cpu_instruction~102
cpu_instruction~109
cpu_instruction~110
cpu_instruction~96
fb_addr_x~31
cpu_instruction~99
cpu_instruction[15..0]
fb_addr_x~53
@OUT011178,OUT011180,OUT011182
[6,42]
OUT0
[20,42]
OUT0
[1,6,29,31,39,42]
@OUT011114,OUT011116,OUT011118
[10,20,42]
OUT0
[10,20,28,42]