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
2.3.22 Bnnn - JP V0, addr .......................... 13
2.3.23 Cxkk - RND Vx, byte ....................... 13
2.3.24 Dxyn - DRW Vx, Vy, nibble ................. 13
2.3.25 Ex9E - SKP Vx ............................... 14
2.3.26 ExA1 - SKNP Vx ............................. 14
2.3.27 Fx07 - LD Vx, DT ............................ 14
2.3.28 Fx0A - LD Vx, K ............................. 14
2.3.29 Fx15 - LD DT, Vx ............................ 14
2.3.30 Fx18 - LD ST, Vx ............................. 14
2.3.31 Fx1E - ADD I, Vx .............................. 14
2.3.32 Fx29 - LD F, Vx .............................. 14
2.3.33 Fx33 - LD B, Vx .............................. 15
2.3.34 Fx55 - LD [I], Vx .............................. 15
2.3.35 Fx65 - LD Vx, [I] ............................. 15
2.4 Keyboard Input .................................. 15
2.5 Sound ........................................... 16
2.6 Screenshots ..................................... 16

3 Hardware Design .................................. 18
  3.1 CPU, Top and Module Access Control ............... 18
  3.2 ALU ........................................... 20
  3.3 Random Number Generator ........................ 20
  3.4 BCD ........................................... 20
  3.5 Graphics and Framebuffer ........................ 21
  3.6 Memory and Registers ............................ 22
  3.7 Sound ......................................... 23
  3.8 Return Address Stack and Program Counter .......... 23
  3.9 Timers ........................................ 24

4 Software Overview ................................ 25
  4.1 Chip8.c ...................................... 25
  4.2 Chip8driver.c .................................. 26
  4.3 Data Transfer ISA ................................ 26

5 Project Plan ...................................... 30
  5.1 Lessons Learned ................................ 30
  5.2 Timeline ...................................... 31
## 6 Debugging

## 7 Code Listing

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>SystemVerilog Code</td>
<td>34</td>
</tr>
<tr>
<td>7.1.1</td>
<td>bcd.sv</td>
<td>34</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Chip8_CPU.sv</td>
<td>35</td>
</tr>
<tr>
<td>7.1.3</td>
<td>Chip8_rand_num_generator.sv</td>
<td>60</td>
</tr>
<tr>
<td>7.1.4</td>
<td>Chip8_Stack.sv</td>
<td>61</td>
</tr>
<tr>
<td>7.1.5</td>
<td>Chip8_ALU.sv</td>
<td>63</td>
</tr>
<tr>
<td>7.1.6</td>
<td>Chip8_framebuffer.sv</td>
<td>65</td>
</tr>
<tr>
<td>7.1.7</td>
<td>Chip8_VGA_Emulator.sv</td>
<td>69</td>
</tr>
<tr>
<td>7.1.8</td>
<td>audio_codec.sv</td>
<td>75</td>
</tr>
<tr>
<td>7.1.9</td>
<td>audio_effects.sv</td>
<td>78</td>
</tr>
<tr>
<td>7.1.10</td>
<td>Chip8_SoundController.sv</td>
<td>82</td>
</tr>
<tr>
<td>7.1.11</td>
<td>i2c_av_config.sv</td>
<td>84</td>
</tr>
<tr>
<td>7.1.12</td>
<td>i2c_controller.sv</td>
<td>87</td>
</tr>
<tr>
<td>7.1.13</td>
<td>clk_div.sv</td>
<td>90</td>
</tr>
<tr>
<td>7.1.14</td>
<td>timer.sv</td>
<td>91</td>
</tr>
<tr>
<td>7.1.15</td>
<td>Chip8_Top.sv</td>
<td>92</td>
</tr>
<tr>
<td>7.1.16</td>
<td>SoCKit_top.sv</td>
<td>113</td>
</tr>
<tr>
<td>7.1.17</td>
<td>utils.svh</td>
<td>140</td>
</tr>
<tr>
<td>7.1.18</td>
<td>enums.svh</td>
<td>140</td>
</tr>
<tr>
<td>7.2</td>
<td>Testbenches</td>
<td>143</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Chip8_CPU_6xkk_7xkk.sv</td>
<td>143</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Chip8_CPU_big_testbench.sv</td>
<td>149</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Chip8_CPU_testbench.sv</td>
<td>168</td>
</tr>
<tr>
<td>7.2.4</td>
<td>Chip8_Top_test.sv</td>
<td>169</td>
</tr>
<tr>
<td>7.2.5</td>
<td>delay_timer_testbench.sv</td>
<td>170</td>
</tr>
<tr>
<td>7.2.6</td>
<td>Triple_port_reg_file_test.sv</td>
<td>171</td>
</tr>
<tr>
<td>7.2.7</td>
<td>fb_testbench.sv</td>
<td>173</td>
</tr>
<tr>
<td>7.2.8</td>
<td>alu_testbench.sv</td>
<td>182</td>
</tr>
<tr>
<td>7.3</td>
<td>Linux Code</td>
<td>199</td>
</tr>
<tr>
<td>7.3.1</td>
<td>chip8.c</td>
<td>199</td>
</tr>
<tr>
<td>7.3.2</td>
<td>chip8driver.c</td>
<td>212</td>
</tr>
<tr>
<td>7.3.3</td>
<td>chip8driver.h</td>
<td>220</td>
</tr>
<tr>
<td>7.3.4</td>
<td>Makefile</td>
<td>225</td>
</tr>
<tr>
<td>7.3.5</td>
<td>socfpga.dts</td>
<td>226</td>
</tr>
<tr>
<td>7.3.6</td>
<td>usbkeyboard.c</td>
<td>233</td>
</tr>
</tbody>
</table>
7.3.7 usbkeyboard.h ........................................... 237
7.4 Git commit history ........................................ 239
7.5 Schematics .................................................... 270
  7.5.1 Chip8_framebuffer.sv .................................... 270
  7.5.2 clk_div.sv ............................................... 270
  7.5.3 timer.sv ................................................ 271
  7.5.4 Chip8_rand_num_generator.sv .......................... 272
  7.5.5 bcd.sv .................................................. 274
  7.5.6 memory.sv .............................................. 274
  7.5.7 Chip8_SoundController.sv ............................. 275
  7.5.8 Entire Design .......................................... 276
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, an 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.
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.  

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,

---

1http://devernay.free.fr/hacks/chip8/C8TECH10HTM

---
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
Figure 2.3: Chip8 character sprite specification

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

![Chip8 16-key keyboard specification](image)

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.  

2.6 Screenshots

![Tapeworm game running on the device](http://devernay.free.fr/hacks/chip8/C8TECH10.HTM)

Figure 2.6: Tapeworm game running on the device
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 $2^{13} \approx 2048$). 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 combinatorially.

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.

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, Chip8Framebuffer.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\(^{2}\)C) Protocol. The I\(^{2}\)C controller and I\(^{2}\)C 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 \texttt{pc\_src} == \texttt{PRC\_SRC\_SKIP}, \texttt{next\_pc} is set to \texttt{pc+4}. This effectively skips the next instruction -if \texttt{pc\_src} == \texttt{PC\_SRC\_NEXT}, \texttt{next\_pc} is set to \texttt{pc+2}. This is the default, which increments the PC by one instruction.

There is also a special case for setting \texttt{next\_pc}. The CPU sets flags regarding the return address stack (RAS). This flag, \texttt{stk\_op}, acts as follows:
- if the CPU sets \texttt{stk\_op} to \texttt{STACK\_HOLD}, the PC is unaffected by the RAS
- if the CPU sets \texttt{stk\_op} to \texttt{STACK\_PUSH}, \texttt{pc+2} is pushed to the top of the stack
- if the CPU sets \texttt{stk\_op} to \texttt{STACK\_POP}, \texttt{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 \texttt{stk\_op} == \texttt{STACK\_PUSH}, nor does it pop every time it equals \texttt{STACK\_POP}. These operations only happen when \texttt{stk\_op} changes from \texttt{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

![Overview of the software design](image)

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 NNNNNNNXX 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 NNNNNNNXX 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 NNNNNNNXX 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 NNNNNNNXX 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 NNNNNNNXX 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 NNNNNNNXX 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_AAAA 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_XXY_YYY

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

```verilog
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)
        if(shift[15:12] >= 3’d5)
        if(shift[19:16] >= 3’d5)
        // Shift entire register left once
        shift = shift << 1;
    end
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 occurred. */
/* 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 */

end
hundreds = shift[19:16];
tens = shift[15:12];
ones = shift[11:8];
end
module Chip8_CPU(
    input logic cpu_clk,
    input logic[15:0] instruction,
    input logic[7:0] reg_readdata1, reg_readdata2,
    mem_readdata1, mem_readdata2,
    input logic[15:0] reg_I_readdata,
    input logic[7:0] delay_timer_readdata,

    input logic key_pressed,
    input logic[3:0] key_press,

    input logic[11:0] PC_readdata,

    input logic[31:0] stage,

    input logic fb_readdata,

    input Chip8_STATE top_level_state,

    output logic delay_timer_WE, sound_timer_WE,
    output logic[7:0] delay_timer_writedata,
    → sound_timer_writedata,

    output PC_SRC pc_src,
    output logic[11:0] PC_writedata,

    output logic reg_WE1, reg_WE2,
    output logic[3:0] reg_addr1, reg_addr2,
    output logic[7:0] reg_writedata1, reg_writedata2,

    output logic mem_WE1, mem_WE2,
    output logic[11:0] mem_addr1, mem_addr2,
    output logic[ 7:0] mem_writedata1, mem_writedata2,
    output logic mem_request,
)
output logic reg_I_WE,
output logic[15:0] reg_I_writedata,
output logic stk_reset,
output STACK_OP stk_op,
output logic[15:0] stk_writedata,
output logic [4:0] fb_addr_y, //max val = 31
output logic [5:0] fb_addr_x, //max val = 63
output logic fb_writedata, //data to write to
output logic fb_WE, //enable writing to address
output logic fbreset,
output logic bit_overwritten, //VF overwritten
output logic isDrawing, //Draw instruction where VF could be overwritten
output logic halt_for_keypress
);

logic[15:0] alu_in1, alu_in2, alu_out;
ALU_f alu_cmd;
logic alu_carry;
wire[15:0] rand_num;
logic[7:0] to_bcd;
wire[3:0] bcd_hundreds, bcd_tens, bcd_ones;
wire[31:0] stage_shifted_by4_minus1 = (stage >> 32'h4) - 32'h1;
logic[31:0] num_rows_written; //used for sprite writing
logic [31:0] stageminus16;
Chip8_rand_num_generator rand_num_generator(cpu_clk,
=> rand_num);

38
bcd binary_to_dec(to_bcd, bcd_hundreds, bcd_tens, bcd_ones);
Chip8_ALU alu(alu_in1, alu_in2, alu_cmd, alu_out, alu_carry);

always_comb begin
    /*DEFAULT WIRE VALUES BEGIN*/
    delay_timer_WE = 1'b0;
sound_timer_WE = 1'b0;
delay_timer_writedata = 8'b0;
sound_timer_writedata = 8'b0;
    pc_src = PC_SRC_NEXT;
    PC_writedata = 12'b0;
    reg_WE1 = 1'b0;
    reg_WE2 = 1'b0;
    reg_addr1 = 4'b0;
    reg_addr2 = 4'b0;
    reg_writedata1 = 8'b0;
    reg_writedata2 = 8'b0;
    mem_WE1 = 1'b0;
    mem_WE2 = 1'b0;
    mem_addr1 = 12'h0;
    mem_addr2 = 12'h0;
    mem_request = 1'b0;
    mem_writedata1 = 8'h0;
    mem_writedata2 = 8'h0;
    reg_I_WE = 1'b0;
    reg_I_writedata = 16'h0;
    fb_addr_y = 5'h0;
    fb_addr_x = 6'h0;
    fb_writedata = 1'b0;
    fb_WE = 1'b0;
    fbreset = 1'b0;
    num_rows_written = 4'h0;
    bit_overwritten = 1'b0;
    halt_for_keypress = 1'b0;
    alu_in1 = 16'h0;
    alu_in2 = 16'h0;
alu_cmd = ALU_f_NOP;
to_bcd = 8'h0;
stk_op = STACK_HOLD;
stk_reset = 1'b0;
stk_writedata = 16'b0;
isDrawing = 1'b0;
stageminus16 = stage - 32'd16;

/*END DEFAULT VALUES*/

/*BEGIN INSTRUCTION DECODE*/
if(top_level_state == Chip8_RUNNING && stage != 32'h0)
  begin
    casex (instruction)
      // 16'h???: begin
      //This instruction is only used on the old
      // computers on which Chip-8
      // was originally implemented. It is ignored by
      // modern interpreters.
      // end
      16'h00E0: begin //00E0 - CLS
        //Clear the screen
        if(stage == 32'h2) begin
          fbreset = 1'b1;
        end else if (stage > 32'h2 & stage < 32'd8189)
          begin
            fb_addr_x = stage[7:2];
            fb_addr_y = stage[12:8];
            fb_WE = 1'b1;
            fb_writedata = 1'b0;
            //CPU DONE
          end
      end

      //memory module fix May 10
      16'h00EE: begin //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.
if (stage >= 32'h3 & stage <= NEXT_PC_WRITE_STAGE) begin // two stages b/c stack takes two cycles
  stk_op = STACK_POP;
  pc_src = PC_SRC_STACK;
end else begin
  // CPU DONE
end

// memory module fix May 10
16'h1xxx: begin // 1nnn - JP addr
  // The interpreter sets the program counter to nnn.
  if (stage >= 32'h3 & stage <= NEXT_PC_WRITE_STAGE) begin
    pc_src = PC_SRC_ALU;
    PC_writedata = instruction[11:0];
  end else begin
    // CPU DONE
  end
end

// memory module fix May 10
16'h2xxx: begin // 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.
  if (stage >= 32'h3 & stage <= NEXT_PC_WRITE_STAGE) begin
    stk_op = STACK_PUSH;
  end else begin
    // CPU DONE
  end
end
stk_writedata = PC_readdata + 12'h2;
pc_src = PC_SRC_ALU;
PC_writedata = instruction[11:0];
end else begin
  //CPU DONE
end

//memory module fix May 10
16'h3xxx: begin //3kkk - 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.

  if(stage >= 32'h3 & stage <=
    NEXT_PC_WRITE_STAGE) begin
    reg_addr1 = instruction[11:8];
    if(reg_readdata1 == instruction[7:0])
      pc_src = PC_SRC_SKIP;
    else pc_src = PC_SRC_NEXT;
  end
end

//memory module fix May 10
16'h4xxx: begin //4kkk - 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.

  if(stage >= 32'h3 & stage <=
    NEXT_PC_WRITE_STAGE) begin
    reg_addr1 = instruction[11:8];
    if(reg_readdata1 != instruction[7:0])
      pc_src = PC_SRC_SKIP;
  end
else pc_src = PC_SRC_NEXT;

eend

//memory module fix May 10
16'h5xx0: begin //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.
  if(stage >= 32'h3 & stage <=
    NEXT_PC_WRITE_STAGE) begin
    reg_addr1 = instruction[11:8];
    reg_addr2 = instruction[7:4];
    if(reg_readdata1 == reg_readdata2) pc_src
      = PC_SRC_SKIP;
    else pc_src = PC_SRC_NEXT;
  end
end

//memory module fix May 10
16'h6xxx: begin //6xkk - LD Vx, byte
  //Set Vx = kk.
  //The interpreter puts the value kk into
  // register Vx.

  if(stage == 32'h2 || stage == 32'h3) begin
    reg_addr1 = instruction[11:8];
    reg_writedata1 = instruction[7:0];
    reg_WE1 = 1'b1;
  end else begin
    //CPU DONE
  end
  //CPU DONE
end

//memory module fix May 10
16'h7xxx: begin //7xkk - ADD Vx, byte
//Set Vx = Vx + kk.
//Adds the value kk to the value of register Vx, then stores the result in Vx.
if (stage >= 32'h2 & stage <= 32'h6) begin
  reg_addr1 = instruction[11:8];
end else if (stage == 32'h7) begin
  reg_addr1 = instruction[11:8];
  reg_writedata1 = alu_out[7:0];
  reg_WE1 = 1'b1;

  alu_in1 = reg_readdata1;
  alu_in2 = instruction[7:0];
  alu_cmd = ALU_f_ADD;
end else begin
  //CPU DONE
end
end

//memory module fix May 10
//Arithmetic operators
16'h8xxx: begin //8xyk
  if (stage >= 32'h2 && stage <= 32'h7) begin
    reg_addr1 = instruction[11:8];
    reg_addr2 = instruction[7:4];
  end else if (stage >= 32'h2 && stage <= 32'h8) begin
    case (instruction[3:0])
      4'h0: begin //8xy0 - LD Vx, Vy
        //Set Vx = Vy.
        //Stores the value of register Vy in register Vx.
        reg_addr1 = instruction[11:8];
        reg_addr2 = instruction[7:4];
        reg_writedata1 = reg_readdata2;
        reg_WE1 = 1'b1;
      end

end

44
4'h1: begin //8xy1 - OR Vx, Vy
  //Set Vx = Vx OR Vy.
  //Perform 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.

  alu_cmd = ALU_f_OR;
  alu_in1 = reg_readdata1;
  alu_in2 = reg_readdata2;

  reg_addr1 = instruction[11:8];
  reg_addr2 = instruction[7:4];
  reg_WE1 = 1'b1;
  reg_writedata1 = alu_out[7:0];
end

4'h2: begin //8xy2 - AND Vx, Vy
  //Set Vx = Vx AND Vy.
  //Perform 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.

  alu_cmd = ALU_f_AND;
  alu_in1 = reg_readdata1;

  reg_addr1 = instruction[11:8];
  reg_addr2 = instruction[7:4];
  reg_WE1 = 1'b1;
  reg_writedata1 = alu_out[7:0];
  reg_writedata2 = alu_out[7:0];
end
alu_in2 = reg_readdata2;

reg_addr1 = instruction[11:8];
reg_addr2 = instruction[7:4];
reg_WE1 = 1'b1;
reg_writedata1 = alu_out[7:0];
end

4'h3: begin //8xy3 - XOR Vx, Vy
    //Set Vx = Vx XOR Vy.
    //Perform 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.

alu_cmd = ALU_f_XOR;
alu_in1 = reg_readdata1;
alu_in2 = reg_readdata2;
reg_addr1 = instruction[11:8];
reg_addr2 = instruction[7:4];
reg_WE1 = 1'b1;
reg_writedata1 = alu_out[7:0];
end

4'h4: begin //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.

alu_cmd = ALU_f_ADD;
alu_in1 = reg_readdata1;
alu_in2 = reg_readdata2;

reg_addr1 = instruction[11:8];
reg_WE1 = 1'b1;
reg_writedata1 = alu_out[7:0];

reg_addr2 = 4'hF;
reg_WE2 = 1'b1;
reg_writedata2 = alu_carry;
end

4'h5: begin //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.

alu_cmd = ALU_f_MINUS;
alu_in1 = reg_readdata1;
alu_in2 = reg_readdata2;

reg_addr1 = instruction[11:8];
reg_WE1 = 1'b1;
reg_writedata1 = alu_out[7:0];

reg_addr2 = 4'hF;
reg_WE2 = 1'b1;
reg_writedata2 = alu_carry;
end

4'h6: begin //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.

    reg_addr1 = instruction[11:8];
    reg_WE2 = 1'b1;
    reg_writedata2 = {7'h0,
                       reg_readdata1[0]};

    alu_cmd = ALU_f_RSHIFT;
    alu_in1 = reg_readdata1;
    alu_in2 = 1;

    reg_addr2 = 4'hF;
    reg_WE1 = 1'b1;
    reg_writedata1 = alu_out[7:0];
end

4'h7: begin //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.

    alu_cmd = ALU_f_MINUS;
    alu_in1 = reg_readdata2;
    alu_in2 = reg_readdata1;
end
reg_addr1 = instruction[11:8];
reg_WE1 = 1'b1;
reg_writedata1 = alu_out[7:0];

reg_addr2 = 4'hF;
reg_WE2 = 1'b1;
reg_writedata2 = alu_carry;
end

4'hE: begin //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.

  reg_addr2 = 4'hF;
  reg_WE2 = 1'b1;
  reg_writedata2 = {7'h0,
                    reg_readdata1[7]};

  alu_cmd = ALU_f_LSHIFT;
  alu_in1 = reg_readdata1;
  alu_in2 = 1;

  reg_addr1 = instruction[11:8];
  reg_WE1 = 1'b1;
  reg_writedata1 = alu_out[7:0];
end

  default : /* default */;
endcase
  end else begin
    //CPU DONE
    end
end
//memory module fix May 10
16'h9xx0: begin //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.
    if(stage >= 32'h3 & stage <=
      NEXT_PC_WRITE_STAGE) begin
        reg_addr1 = instruction[11:8];
        reg_addr2 = instruction[7:4];
        if(reg_readdata1 != reg_readdata2) pc_src
            = PC_SRC_SKIP;
        else pc_src = PC_SRC_NEXT;
    end
end

//memory module fix May 10
16'hAxxx: begin //Annn - LD I, addr
    //Set I = nnn.
    //The value of register I is set to nnn.
    if(stage == 32'h2 || stage == 32'h3) begin
        reg_I_WE = 1'b1;
        reg_I_writedata = {4'h0,
                        instruction[11:0]};
    end else begin
        //CPU DONE
    end
end

//memory module fix May 10
16'hBxxx: begin //Bnnn - JP V0, addr
    //Jump to location nnn + V0.
    //The program counter is set to nnn plus the
    //value of V0.
if (stage >= 32'h2 & stage <= NEXT_PC_WRITE_STAGE) begin
    reg_addr1 = 4'h0;
    PC_write_data = instruction[11:0] + {4'h0, // reg_readdata1};
    pc_src = PC_SRC_ALU;
end

//memory module fix May 10
16'hCxxx: begin //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.
    if (stage >= 32'h3 & stage <= NEXT_PC_WRITE_STAGE) begin
        alu_cmd = ALU_f_AND;
        alu_in1 = rand_num[7:0];
        alu_in2 = instruction[7:0];

        reg_addr1 = instruction[11:8];
        reg_WE1 = 1'b1;
        reg_write_data1 = alu_out[7:0];
    end else begin
        // CPU DONE
        end
end

//memory module fix May 10
16'hDxxx: begin //Dxyn - DRW Vx, Vy, nibble
    // Display n-byte sprite starting at memory
    // location I at
    end
// (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 XORed 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. See instruction 8xy3 for more information on XOR, and section 2.4, Display, for more information on the Chip-8 screen and sprites.

if (stage > 4'b1111) begin
    reg_addr1 = instruction[11:8];
    reg_addr2 = instruction[7:4];
    num_rows_written = {7'b0, stageminus16[31:7]};
    mem_addr1 = num_rows_written + reg_I_readdata;
    mem_request = 1'b1;
    fb_addr_x = reg_readdata1 + (5'b0, stageminus16[6:4]);
    fb_addr_y = reg_readdata2 + (4'b0, num_rows_written[3:0]);
    fb_writedata = mem_readdata1[3'h7 - stageminus16[6:4]] ^ fb_readdata;
fb_WE = (num_rows_written < {28'h0, instruction[3:0]}) & (stage[3:0]);
bit_overwritten = (mem_readdatal[3'h7 - stageminus16[6:4]]) & (fb_readdata) & fb_WE;
isDrawing = 1'b1;
end
/*
if(stage >= 32'h2) begin
    reg_addr1 = instruction[11:8];
    reg_addr2 = instruction[ 7:4];

    if(stage <= 32'h15) num_rows_written =
    4'b0;
    else num_rows_written =
    stage_shifted_by4_minus1[3:0];//((stage >> 32'h4) - 32'h1);

    mem_addr1 = reg_I_readdata[11:0] +
    {8'b0,num_rows_written};
    mem_request = (stage >= 32'd16) &
    (stage_shifted_by4_minus1 < instruction[3:0]) &
    !(stage[0]);
    fb_WE = (stage >= 32'd16) &
    (stage_shifted_by4_minus1 < instruction[3:0]) &
    (stage[0]);
    fb_addr_x = reg_readdata1 + ({5'b0, stage[3:1]});
    fb_addr_y = reg_readdata2 + ({4'b0,
    num_rows_written});
    fb_writedata = mem_readdatal[stage[3:1]]
^ fb_readdata;
    bit_overwritten =
    (mem_readdatal[stage[3:1]]) & (fb_readdata) & fb_WE;
    //bit_overwritten goes high whenever
    a pixel is set from 1 to 0
    isDrawing = 1'b1;
end
16’hEx9E: begin //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.

    if(stage >= 32’h2 & stage <=
        NEXT_PC_WRITE_STAGE) begin
        reg_addr1 = instruction[11:8];
        if(key_pressed && key_press ==
                reg_readdata1) begin
            pc_src = PC_SRC_SKIP;
        end
    end else begin
        //CPU DONE
    end
end

16’hExA1: begin //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.

    if(stage >= 32’h2 & stage <=
        NEXT_PC_WRITE_STAGE) begin
        reg_addr1 = instruction[11:8];
if(~key_pressed || key_press != reg_readdata1) begin
    pc_src = PC_SRC_SKIP;
end
end else begin
    //CPU DONE
end

//memory module fix May 10
//F Instructions
16’hFx07: begin //Fx07 - LD Vx, DT
    //Set Vx = delay timer value.
    //The value of DT is placed into Vx.
    if(stage >= 32’h2 & stage <= 32’h6) begin
        reg_addr1 = instruction[11:8];
        reg_writedata1 = delay_timer_readdata;
        reg_WE1 = 1’b1;
    end else begin
        //CPU DONE
    end
end

16’hFx0A: begin //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.
    if((stage >= 32’h2) && (NEXT_PC_WRITE_STAGE >= stage) && !halt_for_keypress) begin
        halt_for_keypress = 1’b1;
    end else if(key_pressed) begin
        halt_for_keypress = 1’b0;
        reg_addr1 = instruction[11:8];
        reg_writedata1 = key_press;
end
reg_WE1 = 1'b1;
end else begin
  //CPU DONE
end

//memory module fix May 10
16'hFx15: begin //Fx15 - LD DT, Vx
  //Set delay timer = Vx.
  //DT is set equal to the value of Vx.

  if(stage >= 32'h2 & stage <= 32'h6) begin
    reg_addr1 = instruction[11:8];
  end else if(stage <= NEXT_PC_WRITE_STAGE)
  begin
    delay_timer_writedata = reg_readdata1;
    delay_timer_WE = 1'b1;
  end else begin
    //CPU DONE
end
end

//memory module fix May 10
16'hFx18: begin //Fx18 - LD ST, Vx
  //Set sound timer = Vx.
  //ST is set equal to the value of Vx.

  if(stage >= 32'h2 & stage <= 32'h6) begin
    reg_addr1 = instruction[11:8];
  end else if(stage == 32'h7) begin
    sound_timer_writedata = reg_readdata1;
    sound_timer_WE = 1'b1;
  end else begin
    //CPU DONE
  end
end

//memory module fix May 10
16’hFx1E: begin //Fx1E - ADD I, Vx
    //Set I = I + Vx.
    //The values of I and Vx are added, and the
    results are stored
    //in I.

    if(stage >= 32'h2 & stage <= 32'h5) begin
        reg_addr1 = instruction[11:8];
    end else if(stage == 32'h6) begin
        alu_cmd = ALU_f_ADD;
        alu_in1 = reg_I_readdata;
        alu_in2 = reg_readdata1;

        reg_I_writedata = alu_out;
        reg_I_WE = 1'b1;
    end else begin
        //CPU DONE
    end

end

//memory module fix May 10

16’hFx29: begin //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.

    //The chip8 fontset has each character
    //starting from 0 to 80,
    //where each character takes 5 bytes each.

    if(stage >= 32'h2 & stage <= 32'h5) begin
        reg_addr1 = instruction[11:8];
    end else if(stage == 32'h6) begin
reg_I_writedata = {10'h0,
                 reg_readdata1[3:0], 2'h0} +
                 reg_readdata1[3:0];

reg_I_WE = 1'b1;
end else begin
  //CPU DONE
end

//memory module fix May 10
16'hFx33: begin //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.

  if(stage >= 32'h2 & stage <= 32'h5) begin
    reg_addr1 = instruction[11:8];
  end else if(stage >= 32'h6 & stage <= 32'h9)
    begin
      to_bcd = reg_readdata1;
      reg_addr1 = instruction[11:8];

      mem_addr1 = reg_I_readdata[11:0];
      mem_request = 1'b1;
      mem_writedata1 = bcd_hundreds;
      mem_WE1 = 1'b1;
  end else if(stage >= 32'hA & stage <= 32'hD)
    begin
      to_bcd = reg_readdata1;
      reg_addr1 = instruction[11:8];

      mem_addr1 = reg_I_readdata + 12'h1;
mem_request = 1'b1;
mem_writedata1 = bcd_tens;
mem_WE1 = 1'b1;

end else if(stage >= 32'hE & stage <= 32'hF)
    begin
        to_bcd = reg_readdata1;
        reg_addr1 = instruction[11:8];

        mem_addr1 = reg_I_readd + 12'h2;
        mem_request = 1'b1;
        mem_writedata1 = bcd_ones;
        mem_WE1 = 1'b1;
    end else begin
    //CPU DONE
    end

end

//memory module fix May 10
16'hFx55: begin //Fx55 - LD [I], Vx
    //Store registers V0 through Vx in memory
    //starting at location I
    //The interpreter copies the values of
    //registers V0 through Vx
    //into memory, starting at the address in I.
    if(stage >= 32'h7 & (stage[18:3] <=
        instruction[11:8])) begin
        reg_addr1 = stage[6:3];

        alu_cmd = ALU_f_ADD;
        alu_in1 = reg_I_readdata;
        alu_in2 = stage[18:3];

        mem_addr1 = alu_out[11:0];
        mem_request = 1'b1;
        mem_writedata1 = reg_readdata1;
        mem_WE1 = &(stage[2:0]);
    end

end
end

//memory module fix May 10
16’hF6x5: begin //Fx65 - LD Vx, [I]
  //Read registers V0 through Vx from memory
  //starting at location I.
  //The interpreter reads values from memory
  //starting at location I into registers V0 through Vx.
  if(stage >= 32’h7 & (stage[18:3] <= instruction[11:8])) begin
    reg_addr1 = stage[6:3];

    alu_cmd = ALU_f_ADD;
    alu_in1 = reg_I_readdata;
    alu_in2 = stage[18:3];

    mem_addr1 = alu_out[11:0];
    mem_request = 1'b1;

    reg_writedata1 = mem_readdata1[7:0];
    reg_WE1 = &(stage[2:0]);
  end

  default : /* default */;
  endcase
end

endmodule

/*END INSTRUCTION DECODE*/
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[14] <= rand_num[1] ^ rand_num[0];
        rand_num[15] <= rand_num[0] ^ rand_num[15];
    end
end
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
    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;
    end
end
```
end else begin
  case (op)
    STACK_PUSH: begin
      if(~hold) begin
        address <= stackptr;
data <= writedata;
        if(secondcycle == 1'h0) begin
          wren <= 1'h1;
          secondcycle <= 1'h1;
        end else begin
          wren <= 1'h0;
          stackptr <= stackptr + 4'b0001;
          secondcycle <= 1'h0;
          hold <= 1'b1;
        end
      end
    end
  STACK_POP: begin
    if(~hold) begin
      address <= stackptr - 4'b0001;
wren <= 1'h0;
      outdata <= q;
      if(secondcycle == 1'h0) begin
        secondcycle <= 1'h1;
        stackptr <= stackptr - 4'b0001;
      end else begin
        secondcycle <= 1'h0;
        hold <= 1'b1;
      end
    end
  endcase
STACK_HOLD: hold = 1'b0;
default: /* default */;
  end
end
endmodule
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;}
always_comb begin
    case (sel)

        ALU_f_OR : begin
            alu_carry = 0;
            out = input1 | input2;
        end

        ALU_f_AND : begin
            alu_carry = 0;
            out = input1 & input2;
        end

        ALU_f_XOR : begin
            alu_carry = 0;
            out = input1 ^ input2;
        end

        ALU_f_ADD : begin
            out = input1 + input2;
            alu_carry = |(out[15:8]);
        end

        ALU_f_MINUS : begin
            alu_carry = input1 > input2;
            out = input1 - input2;
        end

        ALU_f_LSHIFT : begin
            alu_carry = 0;
            out = input1 << input2;
        end

        ALU_f_RSHIFT : begin
            alu_carry = 0;
            out = input1 >> input2;
        end

    endcase
end
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
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 output video
* Built off of Stephen Edwards’s VGA_LED code
* AUTHORS: David Watkins, Levi Oliver, Ashley Kling, Gabrielle Taylor
* Dependencies:
module Chip8_framebuffer(
    input logic clk,
    input logic reset,
    //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")
    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
    input logic fb_WE, //enable writing to address
    input logic is_paused,
    output logic fb_readdata, //data to write to
    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
);
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;
assign fb_readdata = fb_readdata_general;

logic[10:0] counter;

initial begin
    counter = 11'b0;
end

wire[10:0] copy_from_addr;
wire[10:0] copy_to_addr;
wire copy_data;
wire copyWE;
wire deadwire;

logic[31:0] fb_stage;
logic[31:0] time_since_last_copy;

initial begin
    time_since_last_copy <= 32'h0;
    fb_stage <= 32'h0;
end

always_ff @(posedge clk) begin
    if(fb_WE) begin
        fb_stage <= 32'h0;
    end else if(fb_stage < FRAMEBUFFER_REFRESH_HOLD) begin
        fb_stage <= fb_stage;
    end else begin
        fb_stage <= fb_stage + 32'h1;
    end
end

always_ff @(posedge clk) begin
    if(fb_stage >= FRAMEBUFFER_REFRESH_HOLD ||
    time_since_last_copy > COPY_THRESHOLD) begin
        counter <= counter + 1;
    end
    copyWE <= 1'b1;
copy_from_addr <= counter + 11'h1;
copy_to_addr <= counter;

if(counter == 11'b111_1111_1111)
  → time_since_last_copy <= 32'h0;
end else begin
  copyWE <= 1'b0;
  time_since_last_copy <= time_since_last_copy +
  → 32'h1;
  counter <= 11'h0;
end
end

Chip8_VGA_Emulator led_emulator(
  .clk50(clk),
  .reset(reset),
  .fb_pixel_data(fb_readdata_vga),
  .fb_request_addr(fb_addr_vga),
  .is_paused(is_paused),
  .VGA_R(VGA_R),
  .VGA_G(VGA_G),
  .VGA_B(VGA_B),
  .VGA_CLK(VGA_CLK),
  .VGA_HS(VGA_HS),
  .VGA_VS(VGA_VS),
  .VGA_BLANK_n(VGA_BLANK_n),
  .VGA_SYNC_n(VGA_SYNC_n)
);

Framebuffer from_cpu (  
  .clock(clk),
  .address_a(fb_addr_general),
  .address_b(copy_from_addr),
  .data_a(fb_writedata_general),
  .data_b(fb_writedata_vga),
  .wren_a(fb_WE_general),
  .wren_b(fb_WE_vga),
  .wren_b(fb_WE_vga),
module Chip8_VGA_Emulator(
    input logic clk50, reset,
    //input logic [2047:0] framebuffer,
    input logic fb_pixel_data,
    input logic is_paused,

   .q_a(fb_readdata_general),
   .q_b(copy_data)
);

Framebuffer toscreen (  
   .clock(clk),
   .address_a(fb_addr_vga),
   .address_b(copy_to_addr),
   .data_a(fb_writedata_vga),
   .data_b(copy_data),
   .wren_a(fb_WE_vga),
   .wren_b(copyWE),
   .q_a(fb_readdata_vga),
   .q_b(deadwire)
);
endmodule

7.1.7 Chip8_VGA_Emulator.sv

/* Chip8-Framebuffer to VGA module.
   * Adjusts the dimensions of the screen so that it appears 8 times larger.
   * Developed by Levi and Ash
   * Built off of Stephen Edwards’s code
   * Columbia University
*/

module Chip8_VGA_Emulator(
    input logic clk50, reset,
   //input logic [2047:0] framebuffer,
    input logic fb_pixel_data,
    input logic is_paused,
output logic[10:0] fb_request_addr,
output logic [7:0] VGA_R, VGA_G, VGA_B,
output logic VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n,
     VGA_SYNC_n);

/*
* 640 X 480 VGA timing for a 50 MHz clock: one pixel
every other cycle
*
* HCOUNT 1599 0 1279 1599 0
* _______________ ________
* | Video | Video |
* |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
* _______________________ _____________
* |____| VGA_HS |____|
*/

// Parameters for hcount
parameter HACTIVE = 11’d 1280,
HFRONT_PORCH = 11’d 32,
HSYNC = 11’d 192,
HBACK_PORCH = 11’d 96,
HTOTAL = HACTIVE + HFRONT_PORCH + HSYNC +
     HBACK_PORCH; // 1600

// Parameters for vcount
parameter VACTIVE = 10’d 480,
VFRONT_PORCH = 10’d 10,
VSYNC = 10’d 2,
VBACK_PORCH = 10’d 33,
VTOTAL = VACTIVE + VFRONT_PORCH + VSYNC +
     VBACK_PORCH; // 525

logic [10:0] hcount; // Horizontal counter //
     Hcount[10:1] indicates pixel column (0-639)
logic endOfLine;
always_ff @(posedge clk50 or posedge reset)
if (reset) hcount <= 0;
else if (endOfLine) hcount <= 0;
else hcount <= hcount + 11'd 1;

assign endOfLine = hcount == HTOTAL - 1;

// Vertical counter
logic [9:0] vcount;
logic endOfField;

always_ff @(posedge clk50 or posedge reset)
if (reset) vcount <= 0;
else if (endOfLine)
if (endOfField) vcount <= 0;
else vcount <= vcount + 10'd 1;

assign endOfField = vcount == VTOTAL - 1;

// Horizontal sync: from 0x520 to 0x5DF (0x57F)
// 101 0010 0000 to 101 1101 1111
assign VGA_HS = !( (hcount[10:8] == 3'b101) &
                   ! (hcount[7:5] == 3'b111));
assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

assign VGA_SYNC_n = 1; // For adding sync to video
                        // signals; not used for VGA

// Horizontal active: 0 to 1279     Vertical active: 0 to
// 479
// 101 0000 0000 1280 01 1110 0000 480
// 110 0011 1111 1599 10 0000 1100 524
 !( vcount[9] | (vcount[8:5] == 4'b1111) );

/* VGA_CLK is 25 MHz */
assign VGA_CLK = hcount[0]; // 25 MHz clock: pixel latched on rising edge

parameter chip_hend = 7'd 64;
parameter chip_vend = 6'd 32;

parameter left_bound = 7'd 64;
parameter right_bound = 10'd 576;
parameter top_bound = 7'd 112;
parameter bottom_bound = 9'd 368;

logic[11:0] fb_pos;
assign fb_request_addr = ((((vcount[8:0] - top_bound) & ~ (8'b1111_1000)) << (4'd 3)) + ((hcount[10:1] - ~ left_bound) >> (4'd 3)));

logic inChip;
// 120 <= Y-dim < 360
// 64 <= X-dim < 576
/*
 * +--------------------------------+
 * | VGA Screen (640x480)       |
 * | 64 576                    |
 * | +------------------------+ |
 * | | Chip8 Screen           |
 * | | (64x8x32x8)             |
 * | | +------------------------|
 * | |                          |
 * +--------------------------------+
 */

assign inChip = (((hcount[10:1]) >= (chip_hend * (5'd8) + 7'd64)) &
                 ((hcount[10:1]) < (chip_hend * (5'd8) + 10'd576)) &
                 ((vcount[8:0]) >= (chip_vend * (5'd8) + 7'd112)) &
                 ((vcount[8:0]) < (chip_vend * (5'd8) + 10'd368));

assign inChip = (((hcount[10:1]) > (left_bound)) &
                 ((hcount[10:1]) < (right_bound)) &
                 ((vcount[8:0]) > (top_bound)) &
                 ((vcount[8:0]) < (bottom_bound));

/**
 * 16 columns, 8 rows
 * +---------------------------------------------------------+
 * / px24                                                 |
 * / px48 === === === === === === === px432          |
 * / = = = = = = = = = ==                               |
 * / === === = = === === === = =                        |
 * / = = = = = = = = = ==                              |
 * / = = === === === === px88                          |
 */
parameter paused_left = 10’d64;
parameter paused_right = 10’d576;
parameter paused_top = 10’d24;
parameter paused_bottom = 10’d88;

logic [9:0] hcount_offseted, vcount_offseted;
assign hcount_offseted = (hcount[10:1] - paused_left) >> 4;
assign vcount_offseted = (vcount[8:0] - paused_top) >> 3;

reg [31:0] romdata [7:0];

initial begin
  romdata[7] = 32’b0000_1110_1110_1010_1110_1110_1110_0000;
  romdata[6] = 32’b0000_1010_1010_1010_1000_1000_1011_0000;
  romdata[5] = 32’b0000_1110_1110_1010_1110_1110_1001_0000;
  romdata[4] = 32’b0000_1000_1010_1010_0010_1000_1011_0000;
  romdata[3] = 32’b0000_1000_1010_1110_1110_1110_1110_0000;
  romdata[2] = 32’b0;
  romdata[1] = 32’b0;
  romdata[0] = 32’b0;
end

assign inPaused = is_paused & (
  ((hcount[10:1]) >= (paused_left)) &
  ((hcount[10:1]) < (paused_right)) &
  ((vcount[8:0]) >= (paused_top)) &
  ((vcount[8:0]) < (paused_bottom)) &
  romdata[vcount_offseted][hcount_offseted]
);
always_comb begin
    {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0}; // Black
    if (inChip & fb_pixel_data) begin
        // White to show on-pixel
        {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
    end else if(inChip) begin
        // Purple to show general area
        {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'hFF};
    end else if(inPaused) begin
        {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
    end
end
endmodule // VGA_LED_Emulator

7.1.8 audio_codec.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_codec.sv
Sends samples to the audio codec ssm 2603 at audio clock rate.
*/

// Audio codec interface
module audio_codec (
    input clk,
    input reset,
    output [1:0] sample_end,
    output [1:0] sample_req,

endmodule // audio_codec

76
input [15:0] audio_output,
output [15:0] audio_input,
  // 1 - left, 0 - right
input [1:0] channel_sel,

output AUD_ADCLRCK,
input AUD_ADCDAT,
output AUD_DACLRCK,
output AUD_DACDAT,
output AUD_BCLK
);

reg [7:0] lrck_divider;
reg [1:0] bclk_divider;

reg [15:0] shift_out;
reg [15:0] shift_temp;
reg [15:0] shift_in;

wire lrck = !lrck_divider[7];

assign AUD_ADCLRCK = lrck;
assign AUD_DACLRCK = lrck;
assign AUD_BCLK = bclk_divider[1];
assign AUD_DACDAT = shift_out[15];

always @(posedge clk) begin
    if (reset) begin
        lrck_divider <= 8'hff;
        bclk_divider <= 2'b11;
    end else begin
        lrck_divider <= lrck_divider + 1'b1;
        bclk_divider <= bclk_divider + 1'b1;
    end
end

assign sample_end[1] = (lrck_divider == 8'h40);
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
  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;
parameter SINE_LAST = 7’d99;

initial begin
  romdata[0] = 16’h0000;
  romdata[1] = 16’h0805;
  romdata[2] = 16’h1002;
  romdata[3] = 16’h17ee;
  romdata[4] = 16’h1fc3;
  romdata[5] = 16’h2777;
  romdata[6] = 16’h2f04;
  romdata[7] = 16’h3662;
  romdata[8] = 16’h3d89;
  romdata[9] = 16’h4472;
  romdata[10] = 16’h4b16;
  romdata[11] = 16’h516f;
  romdata[12] = 16’h5776;
  romdata[13] = 16’h5d25;
  romdata[14] = 16’h6276;
  romdata[15] = 16’h6764;
  romdata[16] = 16’h6bea;
  romdata[17] = 16’h7004;
  romdata[18] = 16’h73ad;
  romdata[19] = 16’h76e1;
  romdata[20] = 16’h799e;
  romdata[21] = 16’h7be1;
  romdata[22] = 16’h7da7;
  romdata[23] = 16’h7eeef;
  romdata[24] = 16’h7fb7;
  romdata[25] = 16’h7fff;
  romdata[26] = 16’h7fc6;
  romdata[27] = 16’h7f0c;
  romdata[28] = 16’h7dd3;
  romdata[29] = 16’h7c1b;
  romdata[30] = 16’h79e6;
  romdata[31] = 16’h7737;
  romdata[32] = 16’h7410;
  romdata[33] = 16’h7074;
  romdata[34] = 16’h6c67;
romdata[35] = 16'h67ed;
romdata[36] = 16'h630a;
romdata[37] = 16'h5dc4;
romdata[38] = 16'h5820;
romdata[39] = 16'h5222;
romdata[40] = 16'h4bd3;
romdata[41] = 16'h4537;
romdata[42] = 16'h3e55;
romdata[43] = 16'h3735;
romdata[44] = 16'h2fdd;
romdata[45] = 16'h2855;
romdata[46] = 16'h20a5;
romdata[47] = 16'h18d3;
romdata[48] = 16'h10e9;
romdata[49] = 16'h08ee;
romdata[50] = 16'h00e9;
romdata[51] = 16'hf8e4;
romdata[52] = 16'hf0e6;
romdata[53] = 16'he8f7;
romdata[54] = 16'he120;
romdata[55] = 16'hd967;
romdata[56] = 16'hd1d5;
romdata[57] = 16'hca72;
romdata[58] = 16'hc344;
romdata[59] = 16'hbc54;
romdata[60] = 16'hbb5a;
romdata[61] = 16'haf46;
romdata[62] = 16'ha935;
romdata[63] = 16'ha37c;
romdata[64] = 16'h9e20;
romdata[65] = 16'h9926;
romdata[66] = 16'h9494;
romdata[67] = 16'h906e;
romdata[68] = 16'h8cb8;
romdata[69] = 16'h8976;
romdata[70] = 16'h86ab;
romdata[71] = 16'h845a;
romdata[72] = 16'h8286;
romdata[73] = 16'h8130;
romdata[74] = 16'h8059;
romdata[75] = 16'h8003;
romdata[76] = 16'h802d;
romdata[77] = 16'h80d8;
romdata[78] = 16'h8203;
romdata[79] = 16'h83ad;
romdata[80] = 16'h85d3;
romdata[81] = 16'h8875;
romdata[82] = 16'h8b8f;
romdata[83] = 16'h8f1d;
romdata[84] = 16'h931e;
romdata[85] = 16'h978c;
romdata[86] = 16'h9c63;
romdata[87] = 16'ha19e;
romdata[88] = 16'ha738;
romdata[89] = 16'ha2d2b;
romdata[90] = 16'hc0df;
romdata[91] = 16'hc7f9;
romdata[92] = 16'hcf4b;
romdata[93] = 16'hd6ce;
romdata[94] = 16'he648;
romdata[95] = 16'hee30;
romdata[96] = 16'hf629;
end
always @(*) begin
  if (control)
    dat <= romdata[index];
  else
    dat <= 16'd0;
end
always @(posedge audio_clk) begin

if (sample_req) begin
    if (index == SINE_LAST)
        index <= 7'd00;
    else
        index <= index + 1'b1;
end
endmodule

7.1.10 Chip8_SoundController.sv

module Chip8_SoundController (  
    input OSC_50_B8A, //reference clock  
    inout AUD_ADCLRCK, //Channel clock for ADC  
    input AUD_ADCDAT,  
    inout AUD_DACLRCK, //Channel clock for DAC
    //...
output AUD_DACDAT,  
output AUD_XCK,  
inout AUD_BCLK,  
output AUD_I2C_SCLK,  
inout AUD_I2C_SDAT,  
output AUD_MUTE,  

input logic clk,  
input logic is_on,  
input logic reset  
);

wire main_clk;  
wire audio_clk;  

wire [1:0] sample_end;  
wire [1:0] sample_req;  
wire [15:0] audio_output;  
wire [15:0] audio_input;  
wire [3:0] status;  

clock_pll pll (  
  .refclk (OSC_50_B8A),  
  .rst (reset),  
  .outclk_0 (audio_clk),  
  .outclk_1 (main_clk)  
);

i2c_av_config av_config (  
  .clk (main_clk),  
  .reset (reset),  
  .i2c_sclk (AUD_I2C_SCLK),  
  .i2c_sdat (AUD_I2C_SDAT),  
  .status (status)  
);

assign AUD_XCK = audio_clk;
assign AUD_MUTE = is_on;

audio_codec ac (  
  .clk (audio_clk),  
  .reset (reset),  
  .sample_end (sample_end),  
  .sample_req (sample_req),  
  .audio_output (audio_output),  
  .audio_input (audio_input),  
  .channel_sel (2'b10),  
  .AUD_ADCLRCK (AUD_ADCLRCK),  
  .AUD_ADCDAT (AUD_ADCDAT),  
  .AUD_DACLRCK (AUD_DACLRCK),  
  .AUD_DACDAT (AUD_DACDAT),  
  .AUD_BCLK (AUD_BCLK)
);

audio_effects ae (  
  .audio_clk (audio_clk),  
  .main_clk (main_clk),  
  .sample_end (sample_end[1]),  
  .sample_req (sample_req[1]),  
  .audio_output (audio_output),  
  .audio_input (audio_input),  
  .control (1'b1)
);

endmodule

7.1.11 i2c_av_config.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
// configure Audio codec using the I2C protocol
module i2c_av_config (
    input clk,
    input reset,
    output i2c_sclk,
    inout i2c_sdat,
    output [3:0] status
);

reg [23:0] i2c_data;
reg [15:0] lut_data;
reg [3:0] lut_index = 4'd0;

parameter LAST_INDEX = 4'ha;

reg i2c_start = 1'b0;
wire i2c_done;
wire i2c_ack;

i2c_controller control (
    .clk (clk),
    .i2c_sclk (i2c_sclk),
    .i2c_sdat (i2c_sdat),
    .i2c_data (i2c_data),
    .start (i2c_start),
    .done (i2c_done),
    .ack (i2c_ack)
);

always @(*) begin
    case (lut_index)
        4'h0: lut_data <= 16'h0c10; // power on everything except out
        4'h1: lut_data <= 16'h0017; // left input
        4'h2: lut_data <= 16'h0217; // right input
        4'h3: lut_data <= 16'h0479; // left output
    endcase
end
4'h4: lut_data <= 16'h0679;  // right output
4'h5: lut_data <= 16'h08d4;  // analog path
4'h6: lut_data <= 16'h0a04;  // digital path
4'h7: lut_data <= 16'h0e01;  // digital IF
4'h8: lut_data <= 16'h1020;  // sampling rate
4'h9: lut_data <= 16'h0c00;  // power on everything
4'ha: lut_data <= 16'h1201;  // activate
default: lut_data <= 16'h0000;
endcase
end

reg [1:0] control_state = 2'b00;
assign status = lut_index;

always @(posedge clk) begin
  if (reset) begin
    lut_index <= 4'd0;
i2c_start <= 1'b0;
control_state <= 2'b00;
  end else begin
  case (control_state)
  2'b00: begin
    i2c_start <= 1'b1;
i2c_data <= {8'h34, lut_data};
control_state <= 2'b01;
  end
  2'b01: begin
    i2c_start <= 1'b0;
control_state <= 2'b10;
  end
  2'b10: if (i2c_done) begin
    if (i2c_ack) begin
      if (lut_index == LAST_INDEX)
        control_state <= 2'b11;
      else begin
       lut_index <= lut_index + 1'b1;
control_state <= 2'b00;
      end
    end
  end
end
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;
// don’t toggle the clock unless we’re sending data
// clock will also be kept high when sending START and STOP
assign i2c_sclk = (!clock_en) || sclk_divider[6];
wire midlow = (sclk_divider == 7’h1f);
reg sdat = 1’b1;
// rely on pull-up resistor to set SDAT high
assign i2c_sdat = (sdat) ? 1’bz : 1’b0;

reg [2:0] acks;

parameter LAST_STAGE = 5’d29;
assign ack = (acks == 3’b000);
assign done = (stage == LAST_STAGE);

//implementing I2C protocol
always @(posedge clk) begin
  if (start) begin
    sclk_divider <= 7’d0;
    stage <= 5’d0;
    clock_en = 1’b0;
    sdat <= 1’b1;
    acks <= 3’b111;
    data <= i2c_data;
  end else begin
    if (sclk_divider == 7’d127) begin
      sclk_divider <= 7’d0;
    end
    if (stage != LAST_STAGE)
      stage <= stage + 1’b1;
    case (stage)
      // after start
        5’d0: clock_en <= 1’b1;
  end
end
// receive acks
5'd9: acks[0] <= i2c_sdat;
5'd18: acks[1] <= i2c_sdat;
5'd27: acks[2] <= i2c_sdat;

// before stop
5'd28: clock_en <= 1'b0;
endcase
end else
sclk_divider <= sclk_divider + 1'b1;

if (midlow) begin
  case (stage)
    // start
    5'd0: sdat <= 1'b0;
    // byte 1
    5'd1: sdat <= data[23];
    5'd2: sdat <= data[22];
    5'd3: sdat <= data[21];
    5'd4: sdat <= data[20];
    5'd5: sdat <= data[19];
    5'd6: sdat <= data[18];
    5'd7: sdat <= data[17];
    5'd8: sdat <= data[16];
    // ack 1
    5'd9: sdat <= 1'b1;
    // byte 2
    5'd10: sdat <= data[15];
    5'd11: sdat <= data[14];
    5'd12: sdat <= data[13];
    5'd13: sdat <= data[12];
    5'd14: sdat <= data[11];
    5'd15: sdat <= data[10];
    5'd16: sdat <= data[9];
    5'd17: sdat <= data[8];
    // ack 2
    5'd18: sdat <= 1'b1;
    // byte 3
    5'd19: sdat <= data[7];
7.1.13  clk_div.sv

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

endmodule
// 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:

```vhdl
module timer (write_enable, clk, clk_60, data, out, 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
* Top level Chip8 module that controls all other modules

* AUTHORS: David Watkins, Levi Oliver, Ashley Kling, Gabrielle Taylor

* Dependencies:

  * - Chip8_SoundController.sv
  * - Chip8_framebuffer.sv
  * - timer.sv
  * - clk_div.sv
  * - memory.v
  * - reg_file.v
  * - enums.svh
  * - utils.svh
  * - Chip8_CPU.sv

}}

'include "enums.svh"
'include "utils.svh"

module Chip8_Top(
    input logic clk,
    input logic reset,
    input logic [31:0] writedata,
    input logic write,
    input chipselect,
    input logic [17:0] address,

    output logic [31:0] data_out,

    //VGA Output
    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,

    //Audio Output
    input OSC_50_B8A, //reference clock
    inout AUD_ADCLRCK, //Channel clock for ADC
input  AUD_ACDAT,  
input  AUD_DAClark,  //Channel clock for DAC
output AUD_DACDAT,  //DAC data
output AUD_XCK,  
inout AUD_BCLK,  //Bit clock
output AUD_I2C_SCLK,  //I2C clock
inout AUD_I2C_SDAT,  //I2C data
output AUD_MUTE  //Audio mute
);

//Index register
logic [15:0] I;

//Program counter
logic [7:0] pc_state;
logic [11:0] pc = 12'h200;
logic [11:0] next_pc;
logic [31:0] stage;
logic halt_for_keypress;
logic [31:0] last_stage;

//Framebuffer values
logic fbreset;
logic [4:0] fb_addr_y; //max val = 31
logic [5:0] fb_addr_x; //max val = 63
logic fb_writedata; //data to write to address
logic fb_WE; //enable writing to address
logic fb_readdata; //data to write to address
logic fb_paused;

//Keyboard
logic ispressed;
logic [3:0] key;

//Memory
logic [7:0] memwritedata1, memwritedata2;
logic memWE1, memWE2;
logic [11:0] memaddr1, memaddr2;
logic [7:0] memreaddata1, memreaddata2;
// Reg file
logic [7:0] reg_writedata1, reg_writedata2;
logic regWE1, regWE2;
logic [3:0] reg_addr1, reg_addr2;
logic [7:0] reg_readdata1, reg_readdata2;

// CPU
logic [15:0] cpu_instruction;
logic cpu_delay_timer_WE, cpu_sound_timer_WE;
logic [7:0] cpu_delay_timer_writedata,
            cpu_sound_timer_writedata;
PC_SRC cpu_pc_src;
logic [11:0] cpu_PC_writedata;
logic cpu_reg_WE1, cpu_reg_WE2;
logic [3:0] cpu_reg_addr1, cpu_reg_addr2;
logic [7:0] cpu_reg_writedata1, cpu_reg_writedata2;
logic cpu_mem_WE1, cpu_mem_WE2;
logic [11:0] cpu_mem_addr1, cpu_mem_addr2;
logic cpu_mem_request;
logic [7:0] cpu_mem_writedata1, cpu_mem_writedata2;
logic cpu_reg_I_WE;
logic [15:0] cpu_reg_I_writedata;
logic cpu_fbreset;
logic [4:0] cpu_fb_addr_y;
logic [5:0] cpu_fb_addr_x;
logic cpu_fb_writedata;
logic cpu_fb_WE;
logic cpu_halt_for_keypress;
logic cpu_stk_reset;
STACK_OP cpu_stk_op;
logic[15:0] cpu_stk_writedata;
logic cpu_bit_overwritten;
logic cpu_is_drawing;

// Sound
logic sound_on;
logic sound_reset;

//Timers
clock clk_div_reset, clk_div_clk_out;
logic delay_timer_write_enable, delay_timer_out;
logic [7:0] delay_timer_data, delay_timer_output_data;
logic sound_timer_write_enable, sound_timer_out;
logic [7:0] sound_timer_data, sound_timer_output_data;

//Stack
logic stack_reset;
STACK_OP stack_op;
logic [15:0] stack_writedata;
logic [15:0] stack_outdata;

//State
Chip8_STATE state = Chip8_PAUSED;
logic bit_overwritten;
logic is_drawing;

//ARM Registers
logic [5:0] fbvx_prev;
logic [4:0] fbvy_prev;
logic [11:0] mem_addr_prev;
logic chipselect_happened;

//Chipselect temporary values
logic [3:0] chip_reg_addr1_prev;
logic chip_regWE1_prev;
logic [15:0] chip_reg_writedata1_prev;
logic [5:0] chip_fb_addr_x_prev;
logic [4:0] chip_fb_addr_y_prev;
logic chip_fb_writedata_prev;
logic chip_fb_WE_prev;
logic [11:0] chip_memaddr1_prev;
logic chip_memWE1_prev;
logic [7:0] chip_memwritedata1_prev;
logic chip_sound_timer_write_enable_prev;
logic    chip_delay_timer_write_enable_prev;

initial begin
    pc <= 12'h200;
    last_stage <= 32'h0;
    memWE1 <= 1'b0;
    memWE2 <= 1'b0;
    cpu_instruction <= 16'h0;
    delay_timer_write_enable <= 1'b0;
    sound_timer_write_enable <= 1'b0;
    I <= 16'h0;
    regWE1 <= 1'b0;
    regWE2 <= 1'b0;
    fbreset <= 1'b0;
    fb_addr_y <= 5'b0;
    fb_addr_x <= 6'b0;
    fb_writedata <= 1'b0;
    fb_WE <= 1'b0;
    stack_op <= STACK_HOLD;
    state <= Chip8_PAUSED;
    cpu_instruction <= 16'h0;
    stage <= 32'h0;
    stack_reset <= 1'b1;
    fbvx_prev <= 6'h0;
    fbvy_prev <= 5'h0;
    mem_addr_prev <= 12'h0;
    sound_on <= 1'b0;
    chipselect_happened <= 1'b0;
    fb_paused <= 1'b1;
halt_for_keypress <= 1'b0;
end

always_ff @(posedge clk) begin
    if(reset) begin
        // Add initial values for code
        pc <= 12'h200;

        memWE1 <= 1'b0;
        memWE2 <= 1'b0;
        cpu_instruction <= 16'h0;
        delay_timer_write_enable <= 1'b0;
        sound_timer_write_enable <= 1'b0;
        I <= 16'h0;
        regWE1 <= 1'b0;
        regWE2 <= 1'b0;

        fbreset <= 1'b0;
        fb_addr_y <= 5'b0;
        fb_addr_x <= 6'b0;
        fb_writedata <= 1'b0;
        fb_WE <= 1'b0;

        stack_op <= STACK_HOLD;

        state <= Chip8_PAUSED;
        cpu_instruction <= 16'h0;
        stage <= 32'h0;

        stack_reset <= 1'b1;

        fbvx_prev <= 6'h0;
        fbvy_prev <= 5'h0;
        mem_addr_prev <= 12'h0;

        sound_on <= 1'b0;
        chipselect_happened <= 1'b0;
fb_paused <= 1'b1;

halt_for_keypress <= 1'b0;

//Handle input from the ARM processor
end else if(chipselect) begin

chipselect_happened <= 1'b1;

if(!chipselect_happened) begin
    chip_sound_timer_write_enable_prev <= sound_timer_write_enable;
    chip_delay_timer_write_enable_prev <= delay_timer_write_enable;
    chip_reg_addr1_prev <= reg_addr1;
    chip_regWE1_prev <= regWE1;
    chip_reg_writedata1_prev <= reg_writedata1;
    chip_fb_addr_x_prev <= fb_addr_x;
    chip_fb_addr_y_prev <= fb_addr_y;
    chip_fb_writedata_prev <= fb_writedata;
    chip_fb_WE_prev <= fb_WE;
    chip_memaddr1_prev <= memaddr1;
    chip_memWE1_prev <= memWE1;
    chip_memwritedata1_prev <= memwritedata1;
end

casex (address)

//Read/write from register
18'b00_0000_0000_0000_0000_0000_xxxx: begin
    reg_addr1 <= address[3:0];
    data_out <= {24'h0, reg_readdata1};
    if(write) begin
        regWE1 <= 1'b1;
        reg_writedata1 <= writedata[7:0];
    end
end
18’h10 : begin
  if(write)
    I <= writedata[15:0];
  data_out <= {16'b0, I};
end

//Read/write to sound_timer
18’h11 : begin
  data_out <= {24’h0, sound_timer_output_data};
  if(write) begin
    sound_timer_write_enable <= 1’b1;
    sound_timer_data <= writedata[7:0];
  end
end

//Read/write to delay_timer
18’h12 : begin
  data_out <= {24’h0, delay_timer_output_data};
  if(write) begin
    delay_timer_write_enable <= 1’b1;
    delay_timer_data <= writedata[7:0];
  end
end

//Reset stack
18’h13 : begin
  if(write) stack_reset <= 1’b1;
  data_out <= 32’h13;
end

//Read/write to program counter
18’h14 : begin
  data_out <= {4’h0, cpu_instruction, pc};
  if(write) pc <= writedata[11:0]; //0-4095
end
// Read/write key presses
18'h15 : begin
  data_out <= {27'b0, ispressed, key};
  if(write) begin
    ispressed <= wriedata[4];
    key <= wriedata[3:0];
  end
end

// Read/write the state of the emulator
18'h16 : begin
  data_out <= state;
  if(write) begin
    case (wriedata[1:0])
      2'h0: state <= Chip8_RUNNING;
      2'h1: state <= Chip8_RUN_INSTRUCTION;
      2'h2: state <= Chip8_PAUSED;
      default : state <= Chip8_PAUSED;
    endcase
  end
end

// Modify framebuffer
18'h17 : begin
  if(write) begin
    fbvx_prev <= wriedata[10:5];
    fbvy_prev <= wriedata[4:0];

    fb_addr_x <= wriedata[10:5];
    fb_addr_y <= wriedata[4:0];
    fb_writedata <= wriedata[11];
    fb_WE <= wriedata[12];
  end else begin
    data_out <= {31'h0, fb_readdata};
    fb_addr_x <= fbvx_prev;
    fb_addr_y <= fbvy_prev;
    fb_WE <= 1'b0;
  end
end
end
end

//Read/write stack
18’h18 : begin
    $display("READ/WRITE STACK NOT IMPLEMENTED");
    data_out <= 32’h18;
end

//MODIFY MEMORY
18’h19 : begin
    if(write) begin
        memaddr1 <= wriedata[19:8];
        memWE1 <= wriedata[20] & write;
        memwritedata1 <= wriedata[7:0];

        mem_addr_prev <= wriedata[19:8];
    end else begin
        data_out <= {12’h0, mem_addr_prev, memreaddata1};
        memWE1 <= 1’h0;
        memaddr1 <= mem_addr_prev;
    end
end

//Load single instruction
18’h1A : begin
    if(write)
        cpu_instruction <= wriedata[15:0];
    data_out <= {stage[15:0], cpu_instruction};
    stage <= 32’h0;
end

18’h1B : begin
    //Add initial values for code
    pc <= 12’h200;
end
memWE1 <= 1'b0;
memWE2 <= 1'b0;
cpu_instruction <= 16'h0;
delay_timer_write_enable <= 1'b0;
sound_timer_write_enable <= 1'b0;
I <= 16'h0;
regWE1 <= 1'b0;
regWE2 <= 1'b0;

fbreset <= 1'b0;
fb_addr_y <= 5'b0;
fb_addr_x <= 6'b0;
fb_writedata <= 1'b0;
fb_WE <= 1'b0;

stack_op <= STACK_HOLD;

state <= Chip8_PAUSED;
cpu_instruction <= 16'h0;
stage <= 32'h0;

stack_reset <= 1'b1;

fbvx_prev <= 6'h0;
fbvy_prev <= 5'h0;
mem_addr_prev <= 12'h0;

sound_on <= 1'b0;
chipselect_happened <= 1'b0;

fb_paused <= 1'b1;

halt_for_keypress <= 1'b0;

end

default: begin

data_out <= 32'd101;
end

case

end else if(chipselect_happened) begin
    // sound_timer_write_enable <=
    → chip_sound_timer_write_enable_prev;
    // delay_timer_write_enable <=
    → chip_delay_timer_write_enable_prev;
    // reg_addr1 <= chip_reg_addr1_prev;
    // regWE1 <= chip_regWE1_prev;
    // reg.writedata1 <= chip_reg.writedata1_prev;
    // fb_addr_x <= chip_fb_addr_x_prev;
    // fb_addr_y <= chip_fb_addr_y_prev;
    // fb.writedata <= chip_fb.writedata_prev;
    // fb_WE <= chip_fb_WE_prev;
    // memaddr1 <= chip_memaddr1_prev;
    // memWE1 <= chip_memWE1_prev;
    // memwritedata1 <= chip_memwritedata1_prev;

    chipselect_happened <= 1'b0;
    stack_reset <= 1'b0;
end else begin
    fb_paused <= state == Chip8_PAUSED;

case (state)
    Chip8_RUNNING: begin
        sound_on <= sound_timer_out;

        if(halt_for_keypress) begin
            if(ispressed) begin
                halt_for_keypress <= 1'b0;
            end
        end else if(stage == 32'h0) begin
            memaddr1 <= pc;
            memaddr2 <= pc + 12'h1;
            cpu_instruction <= 16'h0;
    end

105
bit_overwritten <= 1'b0;

is_drawing <= 1'b0;

delay_timer_write_enable <= 1'b0;
sound_timer_write_enable <= 1'b0;
regWE1 <= 1'b0;
regWE2 <= 1'b0;
memWE1 <= 1'b0;
memWE2 <= 1'b0;
stack_op <= STACK_HOLD;
end else if (stage == 32'h1) begin
memaddr1 <= pc;
memaddr2 <= pc + 12'h1;
cpu_instruction <= {memreaddata1,
memreaddata2};
last_stage <= stage;
end else if (stage >= 32'h2) begin
last_stage <= stage;
if(cpu_delay_timer_WE) begin
delay_timer_write_enable <= 1'b1;
delay_timer_data <=
cpu_delay_timer_writedata;
end else begin
delay_timer_write_enable <= 1'b0;
end

if(cpu_sound_timer_WE) begin
sound_timer_write_enable <= 1'b1;
sound_timer_data <=
cpu_sound_timer_writedata;
end else begin
sound_timer_write_enable <= 1'b0;
end

if(cpu_reg_WE1) begin
regWE1 <= 1'b1;
reg_writedata1 <=
cpu_reg_writedata1;
end else begin
    regWE1 <= 1'b0;
end

if(cpu_is_drawing) begin
    is_drawing <= 1'b1;
end

if(cpu_bit_overwritten) begin
    bit_ovewritten <= 1'b1;
end

if(stage == 32'd30000 && is_drawing)
begin
    regWE2 <= 1'b1;
    reg_writedata2 <= {7'h0,
                      bit_overwritten};
    reg_addr2 <= 4'hF; //Setting VF
                      register to write
end else if(cpu_reg_WE2) begin
    regWE2 <= 1'b1;
    reg_writedata2 <=
                   cpu_reg_writedata2;
    reg_addr2 <= cpu_reg_addr2;
end else begin
    regWE2 <= 1'b0;
    reg_addr2 <= cpu_reg_addr2;
end

if(cpu_reg_I_WE) begin
    I <= cpu_reg_I_writedata;
end

if(cpu_stk_op == STACK_PUSH) begin
    stack_op <= STACK_PUSH;
    stack_writedata <=
                   cpu_stk_writedata;
end
//next_pc final modification on
// NEXT_PC_WRITE_STAGE
if((stage >= NEXT_PC_WRITE_STAGE - 32'h3) & (stage <= NEXT_PC_WRITE_STAGE)) begin
  if(cpu_stk_op == STACK_POP) begin
    stack_op <= STACK_POP;
    next_pc <= stack_outdata[11:0];
  end else begin
    case (cpu_pc_src)
      PC_SRC_ALU : next_pc <= cpu_PC_writedata;
      PC_SRC_SKIP : next_pc <= pc + 12'd4;
      PC_SRC_NEXT : next_pc <= pc + 12'd2;
      default : next_pc <= pc /*default next_pc <= pc + 12'd2*/;
    endcase
  end
end

if(cpu_fb_WE) begin
  fb_writedata <= cpu_fb_writedata;
  fb_WE <= cpu_fb_WE;
end else begin
  fb_WE <= 1'b0;
end

if(cpu_halt_for_keypress & !ispressed) begin
  halt_for_keypress <= 1'b1;
end

if(cpu_mem_request) begin
memaddr1 <= cpu_mem_addr1;
memaddr2 <= cpu_mem_addr2;
memwritedata1 <=
   cpu_mem_writedata1;
memwritedata2 <=
   cpu_mem_writedata2;
memWE1 <= cpu_mem_WE1;
memWE2 <= cpu_mem_WE2;
end

//Always
reg_addr1 <= cpu_reg_addr1;
fb_addr_x <= cpu_fb_addr_x;
fb_addr_y <= cpu_fb_addr_y;
// memaddr1 <= cpu_mem_addr1;
// memaddr2 <= cpu_mem_addr2;
end

//Cap of 50000, since 1000 instructions/sec is reasonable
if(!halt_for_keypress) begin
  if(stage >= CPU_CYCLE_LENGTH) begin
    stage <= 32'h0;
    pc <= next_pc;
  end
  else if (stage == 32'h1) begin
    if(stage == last_stage) stage <=
       32'h2;
    else stage <= 32'h1;
  end else if(stage == 32'h2) begin
    if(stage == last_stage) stage <=
       32'h3;
    else stage <= 32'h2;
  end
  else begin
    stage <= stage + 32'h1;
  end
end
Chip8_RUN_INSTRUCTION: begin
// sound_on <= 1'b1;
end
Chip8_PAUSED: begin
// sound_on <= 1'b1;
end
default: /* default */;
endcase
end

Chip8_framebuffer framebuffer(
.clk(clk),
.reset(fbreset),
.fb_addr_y(fb_addr_y),
.fb_addr_x(fb_addr_x),
.fb_writedata(fb_writedata),
.fb_WE(fb_WE),
.fb_readdata(fb_readdata),
.is_paused(fb_paused),
.VGA_R(VGA_R),
.VGA_G(VGA_G),
.VGA_B(VGA_B),
.VGA_CLK(VGA_CLK),
.VGA_HS(VGA_HS),
.VGA_VS(VGA_VS),
.VGA_BLANK_n(VGA_BLANK_n),
.VGA_SYNC_n(VGA_SYNC_n)
);

memory memory(
.address_a(memaddr1),
.address_b(memaddr2),
.clock(clk),
.data_a(memwritedata1),
.data_b(memwritedata2),
reg_file reg_file(
    .clock(clk),
    .address_a(reg_addr1),
    .address_b(reg_addr2),
    .data_a(reg_writedata1),
    .data_b(reg_writedata2),
    .wren_a(regWE1),
    .wren_b(regWE2),
    .q_a(reg_readdata1),
    .q_b(reg_readdata2)
);

Chip8_CPU cpu(
    .cpu_clk(clk),
    .instruction(cpu_instruction),
    .reg_readdata1(reg_readdata1),
    .reg_readdata2(reg_readdata2),
    .mem_readdata1(memreaddata1),
    .mem_readdata2(memreaddata2),
    .reg_I_readdata(I),
    .delay_timer_readdata(delay_timer_output_data),
    .key_pressed(ispressed),
    .key_press(key),
    .PC_readdata(pc),
    .stage(stage),
    .top_level_state(state),
    .delay_timer_WE(cpu_delay_timer_WE),
    .sound_timer_WE(cpu_sound_timer_WE),
    .delay_timer_writedata(cpu_delay_timer_writedata),
    .sound_timer_writedata(cpu_sound_timer_writedata),
    .pc_src(cpu_pc_src),
    .PC_writedata(cpu_PC_writedata),
    .wren_a(memWE1),
    .wren_b(memWE2),
    .q_a(memreaddata1),
    .q_b(memreaddata2)
);
.reg_WE1(cpu_reg_WE1),
.reg_WE2(cpu_reg_WE2),
.reg_addr1(cpu_reg_addr1),
.reg_addr2(cpu_reg_addr2),
.reg_writedata1(cpu_reg_writedata1),
.reg_writedata2(cpu_reg_writedata2),
.mem_WE1(cpu_mem_WE1),
.mem_WE2(cpu_mem_WE2),
.mem_addr1(cpu_mem_addr1),
.mem_addr2(cpu_mem_addr2),
.mem_request (cpu_mem_request),
.mem_writedata1(cpu_mem_writedata1),
.mem_writedata2(cpu_mem_writedata2),
.reg_I_WE(cpu_reg_I_WE),
.reg_I_writedata(cpu_reg_I_writedata),
.fbrset(cpu_fbrset),
.fb_addr_y(cpu_fb_addr_y),
.fb_addr_x(cpu_fb_addr_x),
.fb_writedata(cpu_fb_writedata),
.fb_WE(cpu_fb_WE),
.fb_readdata(fb_readdata),
.bit_overwritten(cpu_bit_overwritten),
.isDrawing(cpu_is_drawing),
.stk_reset(cpu_stk_reset),
.stk_op(cpu_stk_op),
.stk_writedata(cpu_stk_writedata),
.halt_for_keypress(cpu_halt_for_keypress) );

Chip8_SoundController sound(
  .OSC_50_B8A(OSC_50_B8A),
  .AUD_ADCLRCK(AUD_ADCLRCK),
  .AUD_ADCDAT(AUD_ADCDAT),
  .AUD_DACLRCK(AUD_DACLRCK),
);
.AUD_DACDAT(AUD_DACDAT),
.AUD_XCK(AUD_XCK),
.AUD_BCLK(AUD_BCLK),
.AUD_I2C_SCLK(AUD_I2C_SCLK),
.AUD_I2C_SDAT(AUD_I2C_SDAT),
.AUD_MUTE(AUD_MUTE),
.clk(clk),
.is_on(sound_on),
.reset(sound_reset)
);

clk_div clk_div(
 .clk_in(clk),
 .reset(clk_div_reset),
 .clk_out(clk_div_clk_out)
);

timer delay_timer(
 .clk(clk),
 .clk_60(clk_div_clk_out),
 .write_enable(delay_timer_write_enable),
 .data(delay_timer_data),
 .out(delay_timer_out),
 .output_data (delay_timer_output_data)
);

timer sound_timer(
 .clk(clk),
 .clk_60(clk_div_clk_out),
 .write_enable(sound_timer_write_enable),
 .data(sound_timer_data),
 .out(sound_timer_out),
 .output_data (sound_timer_output_data)
);

Chip8_Stack stack (  
 .reset(stack_reset),
7.1.16 SoCKit_top.sv

// ------------------
// Copyright (c) 2013 by Terasic Technologies Inc.
// ------------------
// Permission:
// ------------------
// Terasic grants permission to use and modify this code
// for use
// in synthesis for all Terasic Development Boards and
// Altera Development
// Kits made by Terasic. Other use of this code,
// including the selling
// duplication, or modification of any portion is
// strictly prohibited.
// ------------------
// Disclaimer:
// ------------------
// This VHDL/Verilog or C/C++ source code is intended as a
// design reference
// which illustrates how these types of functions can be
// implemented.
// It is the user’s responsibility to verify their design
// for
// consistency and functionality through the use of formal
// verification methods. Terasic provides no warranty regarding the use
// or functionality of this code.

//============================================================================
// Terasic Technologies Inc
// 9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070. Taiwan
//
// web: http://www.terasic.com/
// email: support@terasic.com
//
//============================================================================
// Major Functions: SoCKit_Default
//
// Revision History:

// Ver  Author       Mod. Date   Changes Made:
// V1.0  xinxian     04/02/13  Initial Revision

//define ENABLE_DDR3
//define ENABLE_HPS
//define ENABLE_HSMC_XCVR
module SoCKit_top(

    /////////////AUD/////////////
    AUD_ADCDAT,
    AUD_ADCLRCK,
    AUD_BCLK,
    AUD_DACDAT,
    AUD_DACLRCK,
    AUD_I2C_SCLK,
    AUD_I2C_SDAT,
    AUD_MUTE,
    AUD_XCK,

    ‘ifdef ENABLE_DDR3

    /////////////DDR3///////////
    DDR3_A,
    DDR3_BA,
    DDR3_CAS_n,
    DDR3_CKE,
    DDR3_CK_n,
    DDR3_CK_p,
    DDR3_CS_n,
    DDR3_DM,
    DDR3_DQ,
    DDR3_DQS_n,
    DDR3_DQS_p,
    DDR3_ODT,
    DDR3_RAS_n,
    DDR3_RESET_n,
    DDR3_RZQ,
    DDR3_WE_n,

    ‘endif /*ENABLE_DDR3*/

          /////////////FAN///////////
    FAN_CTRL,

    ‘ifdef ENABLE_HPS

            /////////////HPS///////////
HPS_CLOCK_25,
HPS_CLOCK_50,
HPS_CONV_USB_n,
HPS_DDR3_A,
HPS_DDR3_BA,
HPS_DDR3_CAS_n,
HPS_DDR3_CKE,
HPS_DDR3_CLK_n,
HPS_DDR3_CLK_p,
HPS_DDR3_CS_n,
HPS_DDR3_DM,
HPS_DDR3_DQ,
HPS_DDR3_DQS_n,
HPS_DDR3_DQS_p,
HPS_DDR3_ODT,
HPS_DDR3_RAS_n,
HPS_DDR3_RESET_n,
HPS_DDR3_RZQ,
HPS_DDR3_WE_n,
HPS_ENET_GTX_CLK,
HPS_ENET_INT_n,
HPS_ENET_MDC,
HPS_ENET_MDIO,
HPS_ENET_RESET_n,
HPS_ENET_RX_CLK,
HPS_ENET_RX_DATA,
HPS_ENET_RX_DV,
HPS_ENET_TX_DATA,
HPS_ENET_TX_EN,
HPS_FLASH_DATA,
HPS_FLASH_DCLK,
HPS_FLASH_NCSO,
HPS_GSENSOR_INT,
HPS_I2C_CLK,
HPS_I2C_SDA,
HPS_KEY,
HPS_LCM_D_C,
HPS_LCM_RST_N,
HPS_LCM_SPIM_CLK,
HPS_LCM_SPIM_MISO,
HPS_LCM_SPIM_MOSI,
HPS_LCM_SPIM_SS,
HPS_LED,
HPS_LTC_GPIO,
HPS_RESET_n,
HPS_SD_CLK,
HPS_SD_CMD,
HPS_SD_DATA,
HPS_SPIM_CLK,
HPS_SPIM_MISO,
HPS_SPIM_MOSI,
HPS_SPIM_SS,
HPS_SW,
HPS_UART_RX,
HPS_UART_TX,
HPS_USB_CLKOUT,
HPS_USB_DATA,
HPS_USB_DIR,
HPS_USB_NXT,
HPS_USB_RESET_PHY,
HPS_USB_STP,
HPS_WARM_RST_n,

`endif /*ENABLE_HPS*/`

`ifdef ENABLE_HSMC_XCVR

HSMC_GXB_RX_p,
HSMC_GXB_TX_p,
HSMC_REF_CLK_p,
'endif
HSMC_RX_n,
HSMC_RX_p,
HSMC_SCL,
HSMC_SDA,
HSMC_TX_n,
HSMC_TX_p,

///////////IRDA///////////
IRDA_RXD,

///////////KEY///////////
KEY,

///////////LED///////////
LED,

///////////OSC///////////
OSC_50_B3B,
OSC_50_B4A,
OSC_50_B5B,
OSC_50_B8A,

///////////PCIE///////////
PCIE_PERST_n,
PCIE_WAKE_n,

///////////RESET///////////
RESET_n,

///////////SI5338///////////
SI5338_SCL,
SI5338_SDA,

///////////Sw///////////
SW,
TEMP_CS_n,
TEMP_DIN,
TEMP_DOUT,
TEMP_SCLK,

USB_B2_CLK,
USB_B2_DATA,
USB_EMPTY,
USB_FULL,
USB_OE_n,
USB_RD_n,
USB_RESET_n,
USB_SCL,
USB_SDA,
USB_WR_n,

VGA_B,
VGA_BLANK_n,
VGA_CLK,
VGA_G,
VGA_HS,
VGA_R,
VGA_SYNC_n,
VGA_VS,

memory_mem_a,
memory_mem_ba,
memory_mem_ck,
memory_mem_ck_n,
memory_mem_cke,
memory_mem_cs_n,
memory_mem_ras_n,
memory_mem_cas_n,
memory_mem_we_n,
memory_mem_reset_n,
memory_mem_dq,
memory_mem_dqs,
memory_mem_dqs_n,
memory_mem_odt,
memory_mem_dm,
memory_oct_rzqin,
hps_io_hps_io_emac1_inst_TX_CLK,
hps_io_hps_io_emac1_inst_TXD0,
hps_io_hps_io_emac1_inst_TXD1,
hps_io_hps_io_emac1_inst_TXD2,
hps_io_hps_io_emac1_inst_TXD3,
hps_io_hps_io_emac1_inst_RXD0,
hps_io_hps_io_emac1_inst_MDI0,
hps_io_hps_io_emac1_inst_MDC,
hps_io_hps_io_emac1_inst_RX_CTL,
hps_io_hps_io_emac1_inst_TX_CTL,
hps_io_hps_io_emac1_inst_RX_CLK,
hps_io_hps_io_emac1_inst_RXD1,
hps_io_hps_io_emac1_inst_RXD2,
hps_io_hps_io_emac1_inst_RXD3,
hps_io_hps_io_qspi_inst_IO0,
hps_io_hps_io_qspi_inst_IO1,
hps_io_hps_io_qspi_inst_IO2,
hps_io_hps_io_qspi_inst_IO3,
hps_io_hps_io_qspi_inst_SS0,
hps_io_hps_io_qspi_inst_CLK,
hps_io_hps_io_sdio_inst_CMD,
hps_io_hps_io_sdio_inst_D0,
hps_io_hps_io_sdio_inst_D1,
hps_io_hps_io_sdio_inst_CLK,
hps_io_hps_io_sdio_inst_D2,
hps_io_hps_io_sdio_inst_D3,
hps_io_hps_io_usb1_inst_D0,
hps_io_hps_io_usb1_inst_D1,
hps_io_hps_io_usb1_inst_CLK,
hps_io_hps_io_usb1_inst_D2,
hps_io_hps_io_usb1_inst_D3,
hps_io_hps_io_usb1_inst_D4,
hps_io_hps_io_usb1_inst_D5,
hps_io_hps_io_usb1_inst_D6,
hps_io_hps_io_usb1_inst_D7,
hps_io_hps_io_usb1_inst_CLK,
hps_io_hps_io_usb1_inst_STP,
hps_io_hps_io_usb1_inst_DIR,
hps_io_hps_io_usb1_inst_NXT,
hps_io_hps_io_spim0_inst_CLK,
hps_io_hps_io_spim0_inst_MOSI,
hps_io_hps_io_spim0_inst_MISO,
hps_io_hps_io_spim0_inst_SS0,
hps_io_hps_io_spim1_inst_CLK,
hps_io_hps_io_spim1_inst_MOSI,
hps_io_hps_io_spim1_inst_MISO,
hps_io_hps_io_spim1_inst_SS0,
hps_io_hps_io_uart0_inst_RX,
hps_io_hps_io_uart0_inst_TX,
hps_io_hps_io_i2c1_inst_SDA,
hps_io_hps_io_i2c1_inst_SCL,
hps_io_hps_io_gpio_inst_GPIO00
);

//=== PORT declarations ====

// AUD ///////////

input
    AUD_ADCDAT;
input
    AUD_ADCLRCK;
inout
    AUD_BCLK;
output
    AUD_DACDAT;
inout
    AUD_DACLRCK;
output
→ AUD_I2C_SCLK;
inout
→ AUD_I2C_SDAT;
output
→ AUD_MUTE;
output
→ AUD_XCK;

ifndef ENABLE_DDR3
      ////////// DDR3 //////////
output [14:0] DDR3_A;
output [2:0] DDR3_BA;
output
→ DDR3_CAS_n;
output
→ DDR3_CKE;
output
→ DDR3 CK_n;
output
→ DDR3 CK_p;
output
→ DDR3 CS n;
output [3:0] DDR3 DM;
inout [31:0] DDR3 DQ;
inout [3:0] DDR3 DQS_n;
inout [3:0] DDR3 DQS_p;
output
→ DDR3 ODT;
output
→ DDR3 RAS_n;
output
→ DDR3 RESET_n;
input
→ DDR3 RZQ;
output
→ DDR3 WE n;
endif /*ENABLE_DDR3*/
'ifdef ENABLE_HPS

input
  → HPS_CLOCK_25;
input
  → HPS_CLOCK_50;
input
  → HPS_CONV_USB_n;
output [14:0]          HPS_DDR3_A;
output [2:0]           HPS_DDR3_BA;
output
  → HPS_DDR3_CAS_n;
output
  → HPS_DDR3_CKE;
output
  → HPS_DDR3_CK_n;
output
  → HPS_DDR3_CK_p;
output
  → HPS_DDR3_CS_n;
output [3:0]           HPS_DDR3_DM;
inout [31:0]           HPS_DDR3_DQ;
inout [3:0]            HPS_DDR3_DQS_n;
inout [3:0]            HPS_DDR3_DQS_p;
output
  → HPS_DDR3_ODT;
output
  → HPS_DDR3_RAS_n;
output
  → HPS_DDR3_RESET_n;
input
  → HPS_DDR3_RZQ;
output
  → HPS_DDR3_WE_n;

'else

output
  → FAN_CTRL;

'endif
input  HPS_ENET_GTX_CLK;
input  HPS_ENET_INT_n;
output HPS_ENET_MDC;
inout  HPS_ENET_MDIO;
output HPS_ENET_RESET_n;
input  HPS_ENET_RX_CLK;
inout [3:0] HPS_ENET_RX_DATA;
in  HPS_ENET_RX_DV;
output [3:0] HPS_ENET_TX_DATA;
inout [3:0] HPS_FLASH_DATA;
output HPS_FLASH_DCLK;
output HPS_FLASH_NCSO;
in  HPS_GSENSOR_INT;
inout  HPS_I2C_CLK;
inout  HPS_I2C_SDA;
inout [3:0] HPS_KEY;
output HPS_LCM_D_C;
output HPS_LCM_RST_N;
in  HPS_LCM_SPIM_CLK;
inout  HPS_LCM_SPIM_MISO;
output
  └→ HPS_LCM_SPIM_MOSI;
output
  └→ HPS_LCM_SPIM_SS;
output [3:0] HPS_LED;
inout
  └→ HPS_LTC_GPIO;
input
  └→ HPS_RESET_n;
output
  └→ HPS_SD_CLK;
inout
  └→ HPS_SD_CMD;
inout [3:0] HPS_SD_DATA;
output
  └→ HPS_SPIM_CLK;
input
  └→ HPS_SPIM_MISO;
output
  └→ HPS_SPIM_MOSI;
output
  └→ HPS_SPIM_SS;
inout [3:0] HPS_SW;
input
  └→ HPS_UART_RX;
output
  └→ HPS_UART_TX;
input
  └→ HPS_USB_CLKOUT;
inout [7:0] HPS_USB_DATA;
input
  └→ HPS_USB_DIR;
input
  └→ HPS_USB_NXT;
output
  └→ HPS_USB_RESET_PHY;
output
  └→ HPS_USB_STP;
input  HPS_WARM_RST_n;
'endif /*ENABLE_HPS*/

///////// HSMC ///////////
input [2:1] HSMC_CLKIN_n;
input [2:1] HSMC_CLKIN_p;
output [2:1] HSMC_CLKOUT_n;
output [2:1] HSMC_CLKOUT_p;
inout  HSMC_CLK_IN0;
output  HSMC_CLK_OUT0;
inout [3:0] HSMC_D;
'ifdef ENABLE_HSMC_XCVR
input [7:0] HSMC_GXB_RX_p;
output [7:0] HSMC_GXB_TX_p;
inout  HSMC_REF_CLK_p;
'endif
ininout [16:0] HSMC_RX_n;
inininout [16:0] HSMC_RX_p;
ininoutput HSMC_SCL;
inininout HSMC_SDA;
inininout [16:0] HSMC_TX_n;
inininout [16:0] HSMC_TX_p;

///////// IRDA ///////////
ininput  IRDA_RXD;

///////// KEY ///////////
ininput [3:0] KEY;

///////// LED ///////////
output [3:0] LED;
 OSC
input
   OSC_50_B3B;
input
   OSC_50_B4A;
input
   OSC_50_B5B;
input
   OSC_50_B8A;

 PCIE
input
   PCIE_PERST_n;
input
   PCIE_WAKE_n;

 RESET
input

 SI5338
inout
   SI5338_SCL;
inout
   SI5338_SDA;

 SW
input [3:0]

 TEMP
output
   TEMP_CS_n;
output
   TEMP_DIN;
inout
   TEMP_DOUT;
output
   TEMP_SCLK;
input  USB_B2_CLK;
inout [7:0]  USB_B2_DATA;
output  USB_EMPTY;
output  USB_FULL;
in  USB_OE_n;
in  USB_RD_n;
in  USB_RESET_n;
inout  USB_SCL;
inout  USB_SDA;
in  USB_WR_n;

output [7:0]  VGA_B;
output  VGA_BLANK_n;
output  VGA_CLK;
output [7:0]  VGA_G;
output  VGA_HS;
output [7:0]  VGA_R;
output  VGA_SYNC_n;
output  VGA_VS;

output wire [14:0]  memory_mem_a;
output wire [2:0]  memory_mem_ba;
output wire  memory_mem_clk;
output wire  memory_mem_clk_n;
output wire  memory_mem_ck;
output wire  memory_mem_cke;
output wire memory_mem_cs_n;
output wire memory_mem_ras_n;
output wire memory_mem_cas_n;
output wire memory_mem_we_n;
output wire memory_mem_reset_n;
inout wire [31:0] memory_mem_dq;
inout wire [3:0] memory_mem_dqs;
inout wire [3:0] memory_mem_dqs_n;
output wire memory_mem_odt;
output wire [3:0] memory_mem_dm;
inout wire hps_io_hps_io_emac1_inst_TX_CLK;
output wire hps_io_hps_io_emac1_inst_TXD0;
output wire hps_io_hps_io_emac1_inst_TXD1;
output wire hps_io_hps_io_emac1_inst_TXD2;
output wire hps_io_hps_io_emac1_inst_TXD3;
input wire hps_io_hps_io_emac1_inst_RXD0;
inout wire hps_io_hps_io_emac1_inst_MDI0;
output wire hps_io_hps_io_emac1_inst_MDC;
input wire hps_io_hps_io_emac1_inst_RX_CTL;
output wire hps_io_hps_io_emac1_inst_TX_CTL;
input wire hps_io_hps_io_emac1_inst_RX_CLK;
inout wire hps_io_hps_io_emac1_inst_RXD1;
inout wire hps_io_hps_io_emac1_inst_RXD2;
input wire
                hps_io_hps_io_emac1_inst_RXD3;
inout wire
                hps_io_hps_io_qspi_inst_IO0;
inout wire
                hps_io_hps_io_qspi_inst_IO1;
inout wire
                hps_io_hps_io_qspi_inst_IO2;
inout wire
                hps_io_hps_io_qspi_inst_IO3;
inout wire
                hps_io_hps_io_qspi_inst_SS0;
output wire
                hps_io_hps_io_qspi_inst_CLK;
inout wire
                hps_io_hps_io_sdio_inst_CMD;
inout wire
                hps_io_hps_io_sdio_inst_D0;
inout wire
                hps_io_hps_io_sdio_inst_D1;
output wire
                hps_io_hps_io_sdio_inst_CLK;
inout wire
                hps_io_hps_io_sdio_inst_D2;
inout wire
                hps_io_hps_io_sdio_inst_D3;
inout wire
                hps_io_hps_io_usb1_inst_D0;
inout wire
                hps_io_hps_io_usb1_inst_D1;
inout wire
                hps_io_hps_io_usb1_inst_D2;
inout wire
                hps_io_hps_io_usb1_inst_D3;
inout wire
                hps_io_hps_io_usb1_inst_D4;
inout wire
                hps_io_hps_io_usb1_inst_D5;
inout wire
  -> hps_io_hps_io_usb1_inst_D6;
inout wire
  -> hps_io_hps_io_usb1_inst_D7;
input wire
  -> hps_io_hps_io_usb1_inst_CLK;
output wire
  -> hps_io_hps_io_usb1_inst_STP;
input wire
  -> hps_io_hps_io_usb1_inst_DIR;
input wire
  -> hps_io_hps_io_usb1_inst_NXT;
output wire
  -> hps_io_hps_io_spim0_inst_CLK;
output wire
  -> hps_io_hps_io_spim0_inst_MOSI;
inout wire
  -> hps_io_hps_io_spim0_inst_MISO;
output wire
  -> hps_io_hps_io_spim0_inst_SS0;
output wire
  -> hps_io_hps_io_spim1_inst_CLK;
output wire
  -> hps_io_hps_io_spim1_inst_MOSI;
inout wire
  -> hps_io_hps_io_spim1_inst_MISO;
output wire
  -> hps_io_hps_io_spim1_inst_SS0;
inout wire
  -> hps_io_hps_io_uart0_inst_RX;
output wire
  -> hps_io_hps_io_uart0_inst_TX;
inout wire
  -> hps_io_hps_io_i2c1_inst_SDA;
inout wire
  -> hps_io_hps_io_i2c1_inst_SCL;
inout wire
  -> hps_io_hps_io_gpio_inst_GPIO00;
//=================================================================================================
// REG/WIRE declarations
//=================================================================================================

// For Audio CODEC
wire AUD_CTRL_CLK;  // For Audio Controller

reg [31:0] Cont;
wire VGA_CTRL_CLK;
wire [9:0] mVGA_R;
wire [9:0] mVGA_G;
wire [9:0] mVGA_B;
wire [19:0] mVGA_ADDR;
wire DLY_RST;

// For VGA Controller
wire mVGA_CLK;
wire [9:0] mRed;
wire [9:0] mGreen;
wire [9:0] mBlue;
wire VGA_Read;  // VGA data request
wire [9:0] recon_VGA_R;
wire [9:0] recon_VGA_G;
wire [9:0] recon_VGA_B;

// For Down Sample
wire [3:0] Remain;
wire [9:0] Quotient;
wire AUD_MUTE;

// Drive the LEDs with the switches
assign LED = SW;

// Make the FPGA reset cause an HPS reset
133
reg [19:0] hps_reset_counter = 20'h0;
reg hps_fpga_reset_n = 0;

always @(posedge OSC_50_B4A) begin
    if (hps_reset_counter == 20'h ffffff) hps_fpga_reset_n <= 1;
    hps_reset_counter <= hps_reset_counter + 1;
end

Chip8 u0 (  
    .clk_clk  
        (OSC_50_B4A), //
        .clk_clk  
    .reset_reset_n  
        (hps_fpga_reset_n), //
        .reset.reset_n  
    .memory_mem_a  
        (memory_mem_a), //
        .memory.mem_a  
    .memory_mem_ba  
        (memory_mem_ba), //
        .mem_ba  
    .memory_mem_ck  
        (memory_mem_ck), //
        .mem_ck  
    .memory_mem_ck_n  
        (memory_mem_ck_n), //
        .mem_ck_n  
    .memory_mem_cke  
        (memory_mem_cke), //
        .mem_cke  
    .memory_mem_cs_n  
        (memory_mem_cs_n), //
        .mem_cs_n  
    .memory_mem_ras_n  
        (memory_mem_ras_n), //
        .mem_ras_n  
)
.memory_mem_cas_n
→ (memory_mem_cas_n), //
→ .mem_cas_n

.memory_mem_we_n
→ (memory_mem_we_n), //
→ .mem_we_n

.memory_mem_reset_n
→ (memory_mem_reset_n), //
→ .mem_reset_n

.memory_mem_dq
→ (memory_mem_dq), //
→ .mem_dq

.memory_mem_dqs
→ (memory_mem_dqs), //
→ .mem_dqs

.memory_mem_dqs_n
→ (memory_mem_dqs_n), //
→ .mem_dqs_n

.memory_mem_odt
→ (memory_mem_odt), //
→ .mem_odt

.memory_mem_dm
→ (memory_mem_dm), //
→ .mem_dm

.memory_oct_rzqin
→ (memory_oct_rzqin), //
→ .oct_rzqin

.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

.hps_io_hps_io_emac1_inst_TXD0
→ (hps_io_hps_io_emac1_inst_TXD0), //
→ .hps_to_emac1_inst_TXD0

.hps_io_hps_io_emac1_inst_TXD1
→ (hps_io_hps_io_emac1_inst_TXD1), //
→ .hps_to_emac1_inst_TXD1
.hps_io_hps_io_emac1_inst_TXD2
→ (hps_io_hps_io_emac1_inst_TXD2), //
→ .hps_io_emac1_inst_TXD2
.hps_io_hps_io_emac1_inst_TXD3
→ (hps_io_hps_io_emac1_inst_TXD3), //
→ .hps_io_emac1_inst_TXD3
.hps_io_hps_io_emac1_inst_RXD0
→ (hps_io_hps_io_emac1_inst_RXD0), //
→ .hps_io_emac1_inst_RXD0
.hps_io_hps_io_emac1_inst_MDIO
→ (hps_io_hps_io_emac1_inst_MDIO), //
→ .hps_io_emac1_inst_MDIO
.hps_io_hps_io_emac1_inst_MDC
→ (hps_io_hps_io_emac1_inst_MDC), //
→ .hps_io_emac1_inst_MDC
.hps_io_hps_io_emac1_inst_RX_CTL
→ (hps_io_hps_io_emac1_inst_RX_CTL), //
→ .hps_io_emac1_inst_RX_CTL
.hps_io_hps_io_emac1_inst_TX_CTL
→ (hps_io_hps_io_emac1_inst_TX_CTL), //
→ .hps_io_emac1_inst_TX_CTL
.hps_io_hps_io_emac1_inst_RX_CLK
→ (hps_io_hps_io_emac1_inst_RX_CLK), //
→ .hps_io_emac1_inst_RX_CLK
.hps_io_hps_io_emac1_inst_RXD1
→ (hps_io_hps_io_emac1_inst_RXD1), //
→ .hps_io_emac1_inst_RXD1
.hps_io_hps_io_emac1_inst_RXD2
→ (hps_io_hps_io_emac1_inst_RXD2), //
→ .hps_io_emac1_inst_RXD2
.hps_io_hps_io_emac1_inst_RXD3
→ (hps_io_hps_io_emac1_inst_RXD3), //
→ .hps_io_emac1_inst_RXD3
.hps_io_hps_io_qspi_inst_IO0
→ (hps_io_hps_io_qspi_inst_IO0), //
→ .hps_io_qspi_inst_IO0
.hps_io_hps_io_qspi_inst_I01
→ (hps_io_hps_io_qspi_inst_I01),  //
→ .hps_io_qspi_inst_I01

.hps_io_hps_io_qspi_inst_I02
→ (hps_io_hps_io_qspi_inst_I02),  //
→ .hps_io_qspi_inst_I02

.hps_io_hps_io_qspi_inst_I03
→ (hps_io_hps_io_qspi_inst_I03),  //
→ .hps_io_qspi_inst_I03

.hps_io_hps_io_qspi_inst_SS0
→ (hps_io_hps_io_qspi_inst_SS0),  //
→ .hps_io_qspi_inst_SS0

.hps_io_hps_io_qspi_inst_CLK
→ (hps_io_hps_io_qspi_inst_CLK),  //
→ .hps_io_qspi_inst_CLK

.hps_io_hps_io_sdio_inst_CMD
→ (hps_io_hps_io_sdio_inst_CMD),  //
→ .hps_io_sdio_inst_CMD

.hps_io_hps_io_sdio_inst_D0
→ (hps_io_hps_io_sdio_inst_D0),  //
→ .hps_io_sdio_inst_D0

.hps_io_hps_io_sdio_inst_D1
→ (hps_io_hps_io_sdio_inst_D1),  //
→ .hps_io_sdio_inst_D1

.hps_io_hps_io_sdio_inst_CLK
→ (hps_io_hps_io_sdio_inst_CLK),  //
→ .hps_io_sdio_inst_CLK

.hps_io_hps_io_sdio_inst_D2
→ (hps_io_hps_io_sdio_inst_D2),  //
→ .hps_io_sdio_inst_D2

.hps_io_hps_io_sdio_inst_D3
→ (hps_io_hps_io_sdio_inst_D3),  //
→ .hps_io_sdio_inst_D3

.hps_io_hps_io_usb1_inst_D0
→ (hps_io_hps_io_usb1_inst_D0),  //
→ .hps_io_usb1_inst_D0
.hps_io_hps_io_usb1_inst_D1
→ (hps_io_hps_io_usb1_inst_D1), //
→ .hps_io_usb1_inst_D1

.hps_io_hps_io_usb1_inst_D2
→ (hps_io_hps_io_usb1_inst_D2), //
→ .hps_io_usb1_inst_D2

.hps_io_hps_io_usb1_inst_D3
→ (hps_io_hps_io_usb1_inst_D3), //
→ .hps_io_usb1_inst_D3

.hps_io_hps_io_usb1_inst_D4
→ (hps_io_hps_io_usb1_inst_D4), //
→ .hps_io_usb1_inst_D4

.hps_io_hps_io_usb1_inst_D5
→ (hps_io_hps_io_usb1_inst_D5), //
→ .hps_io_usb1_inst_D5

.hps_io_hps_io_usb1_inst_D6
→ (hps_io_hps_io_usb1_inst_D6), //
→ .hps_io_usb1_inst_D6

.hps_io_hps_io_usb1_inst_D7
→ (hps_io_hps_io_usb1_inst_D7), //
→ .hps_io_usb1_inst_D7

.hps_io_hps_io_usb1_inst_CLK
→ (hps_io_hps_io_usb1_inst_CLK), //
→ .hps_io_usb1_inst_CLK

.hps_io_hps_io_usb1_inst_STP
→ (hps_io_hps_io_usb1_inst_STP), //
→ .hps_io_usb1_inst_STP

.hps_io_hps_io_usb1_inst_DIR
→ (hps_io_hps_io_usb1_inst_DIR), //
→ .hps_io_usb1_inst_DIR

.hps_io_hps_io_usb1_inst_NXT
→ (hps_io_hps_io_usb1_inst_NXT), //
→ .hps_io_usb1_inst_NXT

.hps_io_hps_io_spim0_inst_CLK
→ (hps_io_hps_io_spim0_inst_CLK), //
→ .hps_io_spim0_inst_CLK
.hps_io_hps_io_spim0_inst_MOSI
→ (hps_io_hps_io_spim0_inst_MOSI), //
→ .hps_io_spim0_inst_MOSI

.hps_io_hps_io_spim0_inst_MISO
→ (hps_io_hps_io_spim0_inst_MISO), //
→ .hps_io_spim0_inst_MISO

.hps_io_hps_io_spim0_inst_SS0
→ (hps_io_hps_io_spim0_inst_SS0), //
→ .hps_io_spim0_inst_SS0

.hps_io_hps_io_spim1_inst_CLK
→ (hps_io_hps_io_spim1_inst_CLK), //
→ .hps_io_spim1_inst_CLK

.hps_io_hps_io_spim1_inst_MOSI
→ (hps_io_hps_io_spim1_inst_MOSI), //
→ .hps_io_spim1_inst_MOSI

.hps_io_hps_io_spim1_inst_MISO
→ (hps_io_hps_io_spim1_inst_MISO), //
→ .hps_io_spim1_inst_MISO

.hps_io_hps_io_spim1_inst_SS0
→ (hps_io_hps_io_spim1_inst_SS0), //
→ .hps_io_spim1_inst_SS0

.hps_io_hps_io_uart0_inst_RX
→ (hps_io_hps_io_uart0_inst_RX), //
→ .hps_io_uart0_inst_RX

.hps_io_hps_io_uart0_inst_TX
→ (hps_io_hps_io_uart0_inst_TX), //
→ .hps_io_uart0_inst_TX

.hps_io_hps_io_i2c1_inst_SDA
→ (hps_io_hps_io_i2c1_inst_SDA), //
→ .hps_io_i2c1_inst_SDA

.hps_io_hps_io_i2c1_inst_SCL
→ (hps_io_hps_io_i2c1_inst_SCL), //
→ .hps_io_i2c1_inst_SCL

.chip8_device_VGA_R
→ (VGA_R),

.chip8_device_VGA_G
→ (VGA_G),
.chip8_device_VGA_B
  -> (VGA_B),
.chip8_device_VGA_CLK
  -> (VGA_CLK),
.chip8_device_VGA_HS
  -> (VGA_HS),
.chip8_device_VGA_VS
  -> (VGA_VS),
.chip8_device_VGA_BLANK_n
  -> (VGA_BLANK_n),
.chip8_device_OSC_50_B8A
  -> (OSC_50_B8A),
.chip8_device_AUD_ADCLRCK
  -> (AUD_ADCLRCK),
.chip8_device_AUD_DACLRCK
  -> (AUD_DACLRCK),
.chip8_device_AUD_ADCLRCK
  -> (AUD_DACLRCK),
.chip8_device_AUD_DACDAT
  -> (AUD_DACDAT),
.chip8_device_AUD_XCK
  -> (AUD_XCK),
.chip8_device_AUD_BCLK
  -> (AUD_BCLK),
.chip8_device_AUD_I2C_SCLK
  -> (AUD_I2C_SCLK),
.chip8_device_AUD_I2C_SDAT
  -> (AUD_I2C_SDAT),
.chip8_device_AUD_MUTE
  -> (AUD_MUTE),
.chip8_device_VGA_SYNC_n
  -> (VGA_SYNC_n)
);
7.1.17  utils.svh

```verilog
ifdef 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

```verilog
/****************************************************************************
* 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

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

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

```
* - ALU_f_LSHIFT : Shift left
* - ALU_f_RSHIFT : Shift right
* - ALU_f_EQUALS : Equals compare
* - ALU_f_GREATER : Greater than compare
* - ALU_f_INC : Increment
*/
typedef enum {
    ALU_f_OR,
    ALU_f_AND,
    ALU_f_XOR,
    ALU_f_ADD,
    ALU_f_MINUS,
    ALU_f_LSHIFT,
    ALU_f_RSHIFT,
    ALU_f_EQUALS,
    ALU_f_GREATER,
    ALU_f_INC,
    ALU_f_NOP
} ALU_f ;

/**
 * PC_SRC defines the behavior of the program counter from
 * output from the CPU
 *
 * * PC_SRC_STACK : Read from the current stack pointer
 * * PC_SRC_ALU : Read the output from the processor
 * * PC_SRC_DEVICE: Read from linux input
 * * PC_SRC_SKIP : Assign PC = PC + 4
 * * PC_SRC_HOLD : Assign PC = PC
 * * PC_SRC_NEXT : Assign PC = PC + 2
 */
typedef enum {
    PC_SRC_STACK,
    PC_SRC_ALU,
    PC_SRC_DEVICE,
    PC_SRC_SKIP,
    PC_SRC_HOLD,
    PC_SRC_NEXT
*/
typedef enum {
    Chip8_RUNNING,
    Chip8_RUN_INSTRUCTION,
    Chip8_PAUSED
} Chip8_STATE;

/**
 * STACK_OP defines the behavior of the stack
 *
 * STACK_POP   : Pop the stack and write out value
 * STACK_PUSH  : Push the input onto the stack
 * STACK_HOLD  : Do nothing
 */
typedef enum {
    STACK_POP,
    STACK_PUSH,
    STACK_HOLD
} STACK_OP;

parameter NEXT_PC_WRITE_STAGE = 32'd12;
parameter CPU_CYCLE_LENGTH = 32'd50000;
parameter FRAMEBUFFER_REFRESH_HOLD = 32'd200000;
parameter COPY_THRESHOLD = 32'd500000;
'endif
7.2 Testbenches

7.2.1 Chip8_CPU_6xkk_7xkk.sv

```verilog
'timescale 1ns/100ps

#include "../enums.svh"

/**
 * Test to make sure that after an instruction happens,
 * all output values get reset to their defaults
 */

task automatic test_resets(ref logic[31:0] stage, ref int total, ref int failed,
                          ref logic delay_timer_WE, sound_timer_WE,
                          ref logic[7:0] delay_timer_writedata,
                          ref logic[11:0] PC_writedata,
                          ref logic reg_WE1, reg_WE2,
                          ref logic[3:0] reg_addr1, reg_addr2,
                          ref logic[7:0] reg_writedata1, reg_writedata2,
                          ref logic mem_WE1, mem_WE2,
                          ref logic[11:0] mem_addr1, mem_addr2,
                          ref logic[7:0] mem_writedata1, mem_writedata2,
                          ref logic reg_I_WE,
                          ref logic[15:0] reg_I_writedata,
                          ref logic sp_push, sp_pop,
                          ref logic [4:0] fb_addr_y, // max val = 31
                          ref logic [5:0] fb_addr_x, // max val = 63
                          ref logic fb_writedata, fb_WE, fbreset,
                          ref logic halt_for_keypress);
#1ns;
assert(
    delay_timer_WE == 1'b0 &
    sound_timer_WE == 1'b0 &
    delay_timer_writedata == 8'b0 &
```

144
sound_timer_writedata == 8'b0 &
//
   pc_src == PC_SRC_NEXT &
PC_writedata == 12'b0 &
reg_WE1 == 1'b0 &
reg_WE2 == 1'b0 &
reg_addr1 == 4'b0 &
reg_addr2 == 4'b0 &
reg_writedata1 == 8'b0 &
reg_writedata2 == 8'b0 &
mem_WE1 == 1'b0 &
mem_WE2 == 1'b0 &
mem_addr1 == 12'h0 &
mem_addr2 == 12'h0 &
mem_writedata1 == 8'h0 &
mem_writedata2 == 8'h0 &
reg_I_WE == 1'b0 &
reg_I_writedata == 16'h0 &
sp_push == 1'b0 &
sp_pop == 1'b0 &
fb_addr_y == 5'h0 &
fb_addr_x == 6'h0 &
fb_writedata == 1'b0 &
freset == 1'b0 &
halt_for_keypress == 1'b0) begin
   $display("All outputs reset to their defaults");
total = total + 1;
end else begin
   $display("Outputs were NOT reset to their defaults. Current stage: %d", stage);
   failed = failed + 1;
end
endtask

/**
  * 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;
    $display("Instruction 7xkk failed with instruction %h
    in stage %d", instruction, stage);
end
wait(stage == 32'h4);

endtask

/**
 * Tests to make sure instruction 6xkk works
 * by testing 61F0.
 */

task automatic test6xkk(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);
repeat (2) @(posedge cpu_clk);
stage = 32'h0;
repeat (1) @(posedge cpu_clk);

instruction = 16'h61F0;

wait(instruction && 16'h61F0 && stage == 32'h2);
#ns;
assert((reg_addr1 == 4'h1) && (reg_writedata1 == 8'hF0) &&
  ↦ (reg_WE1 == 1'h1))begin
$display("6xkk passed with instr %h", instruction);
total = total + 1;
end else begin
$display("6xkk FAILED with instr %h", instruction);
failed = failed + 1;
end
wait(stage == 32'h3);
endtask
module Chip8_CPU_6xkk_7xkk( );

logic cpu_clk;
logic[15:0] instruction;
logic[7:0] reg_readdata1, reg_readdata2,
mem_readdata1, mem_readdata2;
logic[15:0] reg_I_readdata;
logic[7:0] delay_timer_readdata;

logic key_pressed;
logic[3:0] key_press;

logic[11:0] PC_readdata;
logic[31:0] stage;
logic fb_readdata;

Chip8_STATE top_level_state;

logic delay_timer_WE, sound_timer_WE;
logic[7:0] delay_timer_writedata, sound_timer_writedata;

PC_SRC pc_src;
logic[11:0] PC_writedata;

logic reg_WE1, reg_WE2;
logic[3:0] reg_addr1, reg_addr2;
logic[7:0] reg_writedata1, reg_writedata2;

logic mem_WE1, mem_WE2;
logic[11:0] mem_addr1, mem_addr2;
logic[7:0] mem_writedata1, mem_writedata2;

logic reg_I_WE;
logic[15:0] reg_I_writedata;
logic sp_push, sp_pop;
logic [4:0]  fb_addr_y; // max val = 31
logic [5:0]  fb_addr_x; // max val = 63
logic   fb_writedata, // data to write to address
         fb_WE, // enable writing to
         fbreset;
logic    halt_for_keypress;
int total = 0;
int failed = 0;

Chip8_CPU dut(.);

initial begin
  cpu_clk = 0;
  stage = 32'b0;
  forever begin
    #20ns cpu_clk = 1;
    stage = stage + 1;
    #20ns cpu_clk = 0;
  end
end

initial begin
  $display("Starting test tasks.");
  test6xkk(cpu_clk, instruction,
           stage, total, failed, reg_WE1, reg_WE2,
           reg_addr1, reg_addr2, reg_writedata1,
           reg_writedata2);
  test_resets(stage, total, failed, delay_timer_WE,
              sound_timer_WE,
              delay_timer_writedata, sound_timer_writedata,
              /*PC_SRC pc_src,*/
*/
PC_writedata, reg_WE1, reg_WE2, reg_addr1,
  \rightarrow reg_addr2, reg_writedata1, reg_writedata2,
mem_WE1, mem_WE2, mem_addr1,
  \rightarrow mem_addr2, mem_writedata1, mem_writedata2,
reg_I_WE, reg_I_writedata, sp_push,
  \rightarrow sp_pop, fb_addr_y, fb_addr_x,
fb_writedata, fb_WE, fbreset, halt_for_keypress);

test7xkk(cpu_clk, instruction,
  \rightarrow stage, total, failed, reg_WE1, reg_WE2,
reg_addr1, reg_addr2, reg_writedata1,
  \rightarrow reg_writedata2, reg_readdata1);

test_resets(stage, total, failed, delay_timer_WE,
  \rightarrow sound_timer_WE,
delay_timer_writedata, sound_timer_writedata,
  \rightarrow /*PC_SRC pc_src,*/
PC_writedata, reg_WE1, reg_WE2, reg_addr1,
  \rightarrow reg_addr2, reg_writedata1, reg_writedata2,
mem_WE1, mem_WE2, mem_addr1,
  \rightarrow mem_addr2, mem_writedata1, mem_writedata2,
reg_I_WE, reg_I_writedata, sp_push,
  \rightarrow sp_pop, fb_addr_y, fb_addr_x,
fb_writedata, fb_WE, fbreset, halt_for_keypress);

$display("Total number of tests passed: %d", total);
$display("Total number of tests failed: %d", failed);
end
endmodule

7.2.2 Chip8_CPU_big_testbench.sv

/**
* Author: Levi Oliver
* This code tests instructions
* 00e0 -- clear screen
* 6xkk -- load kk into Vx
* 7xkk -- sets Vx = Vx + kk
* Dxy,n -- draws sprite! see instruction description in Chip8_CPU.sv
* Fx29 -- sets I to memory address of sprite representing value in Vx
* Fx33 -- stores BCD value for Vx in memory at I/I+1/I+2 :: h/t/o
*/

`timescale 1ns/100ps

#include "../enums.svh"

/**
 * Test to make sure that after an instruction happens,
 * all output values get reset to their defaults
 */

task automatic test_resets(ref logic cpu_clk, ref logic[31:0] stage, ref int total, ref int failed,
  ref logic delay_timer_WE, sound_timer_WE,
  ref logic[7:0] delay_timer_writedata,
  // sound_timer_writedata,
  // ref PC_SRC pc_src,
  ref logic[11:0] PC_writedata,
  ref logic reg_WE1, reg_WE2,
  ref logic[3:0] reg_addr1, reg_addr2,
  ref logic[7:0] reg_writedata1, reg_writedata2,
  ref logic mem_WE1, mem_WE2,
  ref logic[11:0] mem_addr1, mem_addr2,
  ref logic[7:0] mem_writedata1, mem_writedata2,
  ref logic reg_I_WE,
  ref logic[15:0] reg_I_writedata,
  ref logic sp_push, sp_pop,
  ref logic [4:0] fb_addr_y, // max val = 31
ref logic [5:0] fb_addr_x, \(// max \text{ val } = 63\)
ref logic fb_writedata, fb_WE, fbreset,
ref logic halt_for_keypress);
#3ns;
wait(cpu_clk == 1'b0);
assert(
  delay_timer_WE == 1'b0 &
  sound_timer_WE == 1'b0 &
  delay_timer_writedata == 8'b0 &
  sound_timer_writedata == 8'b0 &
  // pc_src == PC_SRC_NEXT &
  PC_writedata == 12'b0 &
  reg_WE1 == 1'b0 &
  reg_WE2 == 1'b0 &
  reg_addr1 == 4'b0 &
  reg_addr2 == 4'b0 &
  reg_writedata1 == 8'b0 &
  reg_writedata2 == 8'b0 &
  mem_WE1 == 1'b0 &
  mem_WE2 == 1'b0 &
  mem_addr1 == 12'h0 &
  mem_addr2 == 12'h0 &
  mem_writedata1 == 8'h0 &
  mem_writedata2 == 8'h0 &
  reg_I_WE == 1'b0 &
  reg_I_writedata == 16'h0 &
  fb_addr_y == 5'h0 &
  fb_addr_x == 6'h0 &
  fb_writedata == 1'b0 &
  fb_WE == 1'b0 &
  fbreset == 1'b0 &
  halt_for_keypress == 1'b0) begin
$display("All outputs reset to their defaults");
  total = total + 1;
end else begin
$display("Outputs were NOT reset to their defaults. Current stage: %d", stage);
  failed = failed + 1;
end
repeat (2) @(posedge cpu_clk);
endtask
task automatic testFx33(ref logic cpu_clk,
ref logic[15:0] instruction,
ref logic[31:0] stage,
ref int total, failed,
ref logic[ 3:0] reg_addr1,
ref logic[ 7:0] reg_readdata1,
ref logic[11:0] mem_addr1,
ref logic[ 7:0] mem_writedata1,
ref logic mem_WE1,
ref logic[15:0] reg_I_readdata);

instruction = 16'hFe33;
stage = 32'b0;

wait(stage == 32'h2);#1ns;
assert(reg_addr1==instruction[11:8]);
reg_readdata1 = 8'd195;//1100 0011
reg_I_readdata = 16'h03F2;

wait(stage == 32'h3);#1ns;
assert(reg_addr1==instruction[11:8] &
      mem_addr1==reg_I_readdata[11:0] &
      mem_addr1==reg_I_readdata[11:0] &
      mem_writedata1==8'd1& mem_WE1)
else begin
$display("Improper BCD conversion. \n\tGiven value: \
\t%d\n\tCalculated hundreds place: \
\t%d",reg_readdata1, mem_writedata1);
end

wait(stage == 32'h4);#1ns;
assert(reg_addr1==instruction[11:8] &
        mem_addr1==(1+reg_I_readdata[11:0]) &
        mem_writedata1==8'd9 & mem_WE1)
else begin
    $display("Improper BCD conversion. \n    Given value: %d\n    Calculated tens place: %d",reg_readdatal, mem_writedata1);
end

wait(stage == 32'h5);#1ns;
assert(mem_addr1==(2+reg_I_readdata[11:0]) &
        mem_writedata1==8'd5 & mem_WE1) begin
    total = total + 1;
    $display("Instruction Fx33 (store Vx as BCD in I/+1/+2
    in mem) is a success.");
end else begin
    $display("Improper BCD conversion. \n    Given value: %d\n    Calculated ones place: %d",reg_readdatal, mem_writedata1);
end

wait(stage == 32'h6); #1ns;

endtask
task automatic testFx29(ref logic cpu_clk,
                        ref logic[15:0] instruction,
                        ref logic[31:0] stage,
                        ref int total, failed,
                        ref logic[ 3:0] reg_addr1,
                        ref logic[ 7:0] reg_readdata1,
                        ref logic[15:0] reg_I_writedata,
                        ref logic reg_I_WE);

stage = 32'b0;
instruction = 16'hFC29;
wait(stage == 32'h2);#1ns;
assert(reg_addr1==instruction[11:8])
else begin
    failed = failed + 1;
    $display("FX29 FAILED IN STAGE 2.");
end

reg_readdat1 = 8'h0E;

wait(stage == 32'h3);#1ns;
assert(reg_I_writedata == 16'd70 & reg_I_WE == 1'b1)
begin
    $display("Fx29 (set I to Font by Vx) works!");
    total = total + 1;
end else begin
    $display("Fx29 failed in stage 3");
end

wait(stage == 32'h4); #1ns;

endtask

task automatic testDxyn(ref logic cpu_clk,
    ref logic[15:0] instruction,
    ref logic[31:0] stage,
    ref int total,
    ref int failed,
    ref logic[3:0] reg_addr1, reg_addr2,
    ref logic reg_WE1, reg_WE2,
    ref logic[7:0] reg_readdata1,
        reg_readdata2,
    ref logic[7:0] mem_readdata1,
        mem_readdata2,
    ref logic mem_WE1,
    ref logic[11:0] mem_addr1,
    ref logic[5:0] fb_addr_x,
    ref logic[4:0] fb_addr_y,
ref logic fb_writedata, fb_readdata,
    <- fb_WE,
ref logic[15:0] reg_I_readdata,
ref logic bit_overwritten, isDrawing);

instruction = 16'hd392;
stage = 32'h0;
reg_I_readdata = 16'h0F0F;

if((stage == 32'h0) || (stage == 32'h1)) begin
    assert(reg_WE1==1'b0 & reg_WE2==1'b0 &
          mem_WE1==1'b0 & fb_WE==1'b0)
else begin
    $display("INSTR DXYN HAS INVALID WRITE_ENABLE
          VALS BEFORE STAGE 2");
    failed = failed + 1;
end
end

wait(stage == 32'h2); #1ns;
assert(reg_addr1==instruction[11:8] &
    <- reg_addr2==instruction[7:4] &
        mem_addr1==reg_I_readdata & reg_WE1==1'b0 &
    <- reg_WE2==1'b0 &
        mem_WE1==1'b0 & fb_WE==1'b0 & isDrawing)
else begin
    $display("INSTR DXYN HAS FAILED TO SET INITIAL OUTPUT
          VALS IN STAGE 3");
    failed = failed + 1;
end

fb_readdata = 1'b0;
reg_readdata1 = 8'd3; //write to x = 3
reg_readdata2 = 8'd9; //write to y = 9
    //that is position x + 64y =
        576
mem_readdata1 = 8'hAF;
wait(stage == 32'h3); #1ns;
assert(reg_addr1==instruction[11:8] &
  reg_addr2==instruction[7:4] &
  mem_addr1==reg_I_readdata & reg_WE1==1'b0 &
  mem_WE1==1'b0 & fb_WE==1'b0 & isDrawing)
else begin
  $display("INSTR DXYN HAS FAILED TO HOLD VALS FROM
STATE 2 IN STATE 3
\t\tmem_addr1=%h",mem_addr1);
  failed = failed + 1;
end

wait(stage == 32'd15); #1ns;
assert(reg_addr1==instruction[11:8] &
  reg_addr2==instruction[7:4] &
  mem_addr1==reg_I_readdata & reg_WE1==1'b0 &
  mem_WE1==1'b0 & fb_WE==1'b0 & isDrawing)
else begin
  $display("INSTR DXYN HAS FAILED TO HOLD VALS FROM
STATE 2 IN STATE 15
\t\tmem_addr1=%h",mem_addr1);
  failed = failed + 1;
end

wait(stage == 32'd17); #1ns;
assert(reg_addr1==instruction[11:8] &
  reg_addr2==instruction[7:4] &
  mem_addr1==reg_I_readdata & fb_WE==1'b1 &
  fb_addr_x==reg_readdata1 &
  fb_addr_y==reg_readdata2 &
  fb_writedata==1'b1 & bit_overwritten==1'b0 &
  isDrawing)
else begin
$display("INSTR DXYN FAILED IN ITS FIRST WRITE STAGE");
failed = failed + 1;
end

wait(stage == 32’d18); #1ns;
assert(reg_addr1==instruction[11:8] &
        reg_addr2==instruction[7:4] &
        mem_addr1==reg_I_readdata & fb_WE==1’d0 &
        fb_addr_x==(reg_readdata1+1) &
        fb_addr_y==reg_readdata2 &
        bit_overwritten==1’d0 & isDrawing)
else begin
   $display("INSTR DXYN FAILED IN STAGE 18");
   failed = failed + 1;
end

fb_readdata = 1’b1;
wait(stage == 32’d31); #1ns;
assert(reg_addr1==instruction[11:8] &
        reg_addr2==instruction[7:4] &
        mem_addr1==reg_I_readdata & fb_WE==1’d1 &
        fb_addr_x==(reg_readdata1+7) &
        fb_addr_y==reg_readdata2 &
        bit_overwritten==1’d1 & isDrawing)
else begin
   $display("INSTR DXYN FAILED IN STAGE 31");
   failed = failed + 1;
end

fb_readdata = 1’b0;

wait(stage == 32’d32); #3ns;
assert(reg_addr1==instruction[11:8] &
        reg_addr2==instruction[7:4] &
        mem_addr1==(reg_I_readdata+1) & fb_WE==1’d0 &
        fb_addr_x==(reg_readdata1) &
        fb_addr_y==(reg_readdata2+1) &
        bit_overwritten==1’d0 & isDrawing)
else begin
$display("INSTR DXYN FAILED IN STAGE
    32\n\treg_readdata2=%h\n\tfb_addr_y=%h",reg_readdata2,fb_addr_y);
$display("\tmem_addr1=%h\n\treg_I_readdata=%h",mem_addr1,reg_I_readdata);
failed = failed + 1;
end

mem_readdata1 = 8'b1101000;

wait(stage == 32'd33); #1ns;
assert(reg_addr1==instruction[11:8] &
    reg_addr2==instruction[7:4] &
    mem_addr1==(reg_I_readdata+1) & fb_WE==1'b1 &
    fb_addr_x==(reg_readdata1) &
    fb_addr_y==(reg_readdata2+1) &
    bit_overwritten==1'b0 & isDrawing)
else begin
    $display("INSTR DXYN FAILED IN STAGE
        33\n\treg_readdata2=%h\n\tfb_addr_y=%h",reg_readdata2,fb_addr_y);
    $display("\tmem_addr1=%h\n\treg_I_readdata=%h",mem_addr1,reg_I_readdata);
    failed = failed + 1;
end

wait(stage == 32'd39); #1ns;
assert(reg_addr1==instruction[11:8] &
    reg_addr2==instruction[7:4] &
    mem_addr1==(reg_I_readdata+1) & fb_WE==1'b1 &
    fb_addr_x==(reg_readdata1+8'h3) &
    fb_addr_y==(reg_readdata2+1) &
    fb_writedata==1'b1 & bit_overwritten==1'b0 &
    isDrawing)
else begin
    $display("INSTR DXYN FAILED IN STAGE
        39\n\treg_readdata2=%h\n\tfb_addr_y=%h",reg_readdata2,fb_addr_y);
    $display("\tmem_addr1=%h\n\treg_I_readdata=%h",mem_addr1,reg_I_readdata);
    failed = failed + 1;
end

wait(stage == 32'd47); #1ns;
assert(reg_addr1==instruction[11:8] &
    reg_addr2==instruction[7:4] &
    mem_addr1=(reg_I_readdata+1) & fb_WE==1'b1 &
    fb_addr_x=(reg_readdata1+8'h7) &
    fb_addr_y=(reg_readdata2+1) &
    fb_writedata==1'b0 & bit_overwritten==1'b0 &
    isDrawing)
else begin
    $display("INSTR DXYN FAILED IN STAGE
        47\n\treg_readdata2=%h\n\tfb_addr_y=%h",reg_readdata2,fb_addr_y);
    $display("\tmem_addr1=%h\n\treg_I_readdata=%h",mem_addr1,reg_I_readdata);
    failed = failed + 1;
end

wait(stage == 32'd48); #1ns;
assert(fb_WE==1'b0 & bit_overwritten==1'b0 & isDrawing)
else begin
    failed = failed + 1;
    $display("INSTR DXYN FAILED IN STAGE: %d", stage);
end

wait(stage == 32'd61); #1ns;
assert(fb_WE==1'b0 & bit_overwritten==1'b0 & isDrawing)
    begin
        $display("Dxyn draw sprite works!! :D");
        total = total + 1;
    end
else begin
    $display("INSTR DXYN FAILED IN STAGE: %d", stage);
    failed = failed + 1;
end

$display("WOO0000BL");
wait(stage == 32'd100); #1ns;
stage = 32'b0;
endtask
/**
Test to make sure CPU outputs proper request signals for 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);
end

358 wait(stage == 32'h4);
endtask

/**
* Tests to make sure instruction 6xkk works
* by testing 61F0.
*/

361 task automatic test6xkk(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);
repeat (2) @(posedge cpu_clk);
stage = 32'h0;
repeat (1) @(posedge cpu_clk);

379 instruction = 16'h61F0;
wait(instruction && 16'h61F0 && stage == 32'h2);
#1ns;
assert((reg_addr1 == 4'h1) && (reg_writedata1 == 8'hF0) &&
    (reg_WE1 == 1'h1))begin
    $display("6xkk passed with instr %h", instruction);
total = total + 1;
end else begin
    $display("6xkk FAILED with instr %h", instruction);
failed = failed + 1;
end
```verilog
wait(stage == 32'h3);
endtask

task automatic test00E0(ref logic cpu_clk,
            ref logic[15:0] instruction,
            ref logic[31:0] stage,
            ref int total, failed,
            ref logic fb_WE, fb_writedata, fbreset,
            ref logic[5:0] fb_addr_x,
            ref logic[4:0] fb_addr_y);

stage = 32'h0;
instruction = 16'h00e0;

wait(stage == 32'h2); #1ns;
assert(fbreset == 1'b1)
else begin
  failed = failed + 1;
  $display("INSTR 00E0: fbreset NEVER SET HIGH.");
end

wait(stage == 32'h3); #1ns;
assert(fb_WE==1'b1 & fb_writedata==1'b0 & fb_addr_x==0 & fb_addr_y==0 & fbreset==1'b0)
else begin
  failed = failed + 1;
  $display("INSTR 00E0: Did not start clearing at (x,y)=(0, 0). stage: %h", stage);
end

if(stage > 32'h2 & stage < 32'd28189 & fbreset==1'b0)
  begin
    assert(fb_WE==1'b1 & fb_writedata==1'b0)
  end
else begin
  failed = failed + 1;
  $display("INSTR 00E0: failed in stage: %h", stage);
```

163
wait(stage==32’d8188);#1ns;
assert(fb_WE==1’b1 & fb_writedata==1’b0 &
  (!fb_addr_x) & (!fb_addr_y) & fbreset==1’b0)
  begin
    $display("00e0 clear screen success!);
    total = total + 1;
  end else begin
    failed = failed + 1;
    $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,
stage)
  end
end

wait(stage == 32’d8189); #1ns;
endtask

module Chip8_CPU_big_testbench( ) ;

  logic cpu_clk;
  logic[15:0] instruction;
  logic[7:0]  reg_readdata1, reg_readdata2,
              mem_readdata1, mem_readdata2;
  logic[15:0] reg_I_readdata;
  logic[7:0] delay_timer_readdata;

  logic key_pressed;
  logic[3:0] key_press;

  logic[11:0] PC_readdata;
  logic[31:0] stage;

  logic fb_readdata;
Chip8_STATE top_level_state;

logic delay_timer_WE, sound_timer_WE;
logic[7:0] delay_timer_writedata, sound_timer_writedata;

PC_SRC pc_src;
logic[11:0] PC_writedata;

logic reg_WE1, reg_WE2;
logic[3:0] reg_addr1, reg_addr2;
logic[7:0] reg_writedata1, reg_writedata2;

logic mem_WE1, mem_WE2;
logic[11:0] mem_addr1, mem_addr2;
logic[ 7:0] mem_writedata1, mem_writedata2;

logic reg_I_WE;
logic[15:0] reg_I_writedata;
logic sp_push, sp_pop;

logic [4:0] fb_addr_y; // max val = 31
logic [5:0] fb_addr_x; // max val = 63
logic fb_writedata, // data to write to address
        fb_WE, // enable writing to
        fbreset, isDrawing;

logic halt_for_keypress;

logic stk_reset;
STACK_OP stk_op;
logic[15:0] stk_writedata;
logic bit_overwritten;
logic mem_request;

int total = 0;
int failed = 0;
Chip8_CPU dut(.*);

initial begin
  cpu_clk = 0;
  stage = 32'b0;
forever begin
  #20ns cpu_clk = 1;
  stage = stage + 1;
  #20ns cpu_clk = 0;
end
end

initial begin
  $display("Starting test tasks.");
test6xkk(cpu_clk, instruction,
  stage,total,failed,reg_WE1, reg_WE2,
  reg_addr1, reg_addr2,reg_writedata1,
  reg_writedata2);
test_resets(cpu_clk, stage,
  total,failed,delay_timer_WE, sound_timer_WE,
  delay_timer_writedata, sound_timer_writedata,
  PC_writedata,reg_WE1, reg_WE2,reg_addr1,
  reg_addr2,reg_writedata1, reg_writedata2,
  mem_WE1, mem_WE2, mem_addr1,
  mem_addr2,mem_writedata1, mem_writedata2,
  reg_I_WE,reg_I_writedata,sp_push,
  sp_pop,fb_addr_y,fb_addr_x,
  fb_writedata,fb_WE, fbreset,halt_for_keypress);
test7xkk(cpu_clk, instruction,
  stage,total,failed,reg_WE1, reg_WE2,
reg_addr1, reg_addr2, reg_writedata1,
   ← reg_writedata2, reg_readdata1);

test_resets(cpu_clk, stage,
   ← total, failed, delay_timer_WE, sound_timer_WE,
   delay_timer_writedata, sound_timer_writedata,
   → /*PC_SRC pc_src,*/
   PC_writedata, reg_WE1, reg_WE2, reg_addr1,
   → reg_addr2, reg_writedata1, reg_writedata2,
   mem_WE1, mem_WE2, mem_addr1,
   → mem_addr2, mem_writedata1, mem_writedata2,
   reg_I_WE, reg_I_writedata, sp_push,
   → sp_pop, fb_addr_y, fb_addr_x,
   fb_writedata, fb_WE, fbreset, halt_for_keypress);

testDxyn(cpu_clk, instruction, stage, total, failed,
   reg_addr1, reg_addr2, reg_WE1, reg_WE2,
   reg_readdata1,
   ← reg_readdata2, mem_readdata1,
   mem_writedata, mem_WE1, mem_addr1,
   → fb_addr_x, fb_addr_y,
   fb_writedata, fb_readdata,
   ← fb_WE, reg_I_readdata,
   → bit_overwritten, isDrawing);

test_resets(cpu_clk, stage,
   ← total, failed, delay_timer_WE, sound_timer_WE,
   delay_timer_writedata, sound_timer_writedata,
   → /*PC_SRC pc_src,*/
   PC_writedata, reg_WE1, reg_WE2, reg_addr1,
   → reg_addr2, reg_writedata1, reg_writedata2,
   mem_WE1, mem_WE2, mem_addr1,
   → mem_addr2, mem_writedata1, mem_writedata2,
   reg_I_WE, reg_I_writedata, sp_push,
   → sp_pop, fb_addr_y, fb_addr_x,
   fb_writedata, fb_WE, fbreset, halt_for_keypress);
test00E0(cpu_clk, 
  → instruction, stage, total, failed, fb_WE, 
  → fb_writedata, fbreset, fb_addr_x, fb_addr_y);

test_resets(cpu_clk, stage, 
  → total, failed, delay_timer_WE, sound_timer_WE, 
  delay_timer_writedata, sound_timer_writedata,  
  → /*PC_SRC pc_src,* /
   PC_writedata, reg_WE1, reg_WE2, reg_addr1,  
   → reg_addr2, reg_writedata1, reg_writedata2, 
   mem_WE1, mem_WE2, mem_addr1,  
   → mem_addr2, mem_writedata1, mem_writedata2, 
   reg_I_WE, reg_I_writedata, sp_push,  
   → sp_pop, fb_addr_y, fb_addr_x,  
   fb_writedata, fb_WE, fbreset, halt_for_keypress);

testFx29(  
  → cpu_clk, instruction, stage, total, failed, reg_addr1, reg_readdata1, reg_I_readdata);

test_resets(cpu_clk, stage, 
  → total, failed, delay_timer_WE, sound_timer_WE, 
  delay_timer_writedata, sound_timer_writedata,  
  → /*PC_SRC pc_src,* /
   PC_writedata, reg_WE1, reg_WE2, reg_addr1, 
   → reg_addr2, reg_writedata1, reg_writedata2, 
   mem_WE1, mem_WE2, mem_addr1,  
   → mem_addr2, mem_writedata1, mem_writedata2, 
   reg_I_WE, reg_I_writedata, sp_push,  
   → sp_pop, fb_addr_y, fb_addr_x,  
   fb_writedata, fb_WE, fbreset, halt_for_keypress);

testFx33(cpu_clk, instruction, stage, total, 
  → failed, reg_addr1, reg_readdata1, mem_addr1, mem_writedata1, mem_WE1,  
  → reg_I_readdata);

test_resets(cpu_clk, stage, 
  → total, failed, delay_timer_WE, sound_timer_WE,
module Chip8_TESTBENCH;

logic clk;
logic[15:0] instruction;
logic[3:0] testIn1, testIn2;
wire[7:0] testOut1, testOut2;

//Initialize module here
Chip8_CPU cpu (.cpu_clk(clk), .*);

initial begin
    clk = 0;
    forever
        #20ns clk = ~clk;
    end

initial begin
    instruction = 16'h6122;
    repeat (2)
end

endmodule
```verilog
 @(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

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

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

initial begin
    clk_60 = 1'b0;
    reset = 0;
    data = 8'b0000_0000;
    write_enable = 0;

    repeat (8) begin
        @(posedge clk);
        clk_60 = ~clk_60;
    end

    clk_60 = 1'b1;
    data = 8'b0000_1000;
    write_enable = 1;

    repeat (2)
        @(posedge clk);
    write_enable = 0;

    repeat (64) begin
        @(posedge clk);
        clk_60 = ~clk_60;
    end
end
endmodule

7.2.6  Triple_port_reg_file_test.sv

`timescale 10ns/10ns
```
module Triple_port_reg_file_test ();
logic cpu_clk; //system clock that controls
  writing data  
logic[7:0] writedata1, writedata2, VFwritedata; //data
to be written to corresponding addresses
logic WE1, WE2, WEVF; //enable writing on
  addressed registers
logic[3:0] addr1, addr2; //addresses to write to and
  read from
logic[7:0] readdata1, readdata2, VFreaddata; //data
  output from addressed registers

//Initialize module here
Chip8_register_file tprf (.cpu_clk(cpu_clk), .*);

initial begin
  cpu_clk = 0;
  forever
    #20ns cpu_clk = ~cpu_clk;
end

initial begin
  WE1 = 1;
  writedata1 = 8’hEE;
  addr1 = 4’h0;
  repeat (2)
    @(posedge cpu_clk);
  WE2 = 1;
  writedata2 = 8’h44;
  addr2 = 4’h1;
  repeat (2)
    @(posedge cpu_clk);
  WEVF = 1;
  VFwritedata = 8’hFF;
  repeat (2)
    @(posedge cpu_clk);
  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*/
*/
```
//task automatic testReset(logic clk,
// logic[4:0] fb_addr_y,
// logic[5:0] fb_addr_x,
// logic fb_writedata,
// logic fb_WE,
// logic reset);

// 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);
//endtask

task automatic testWriteOne(ref logic clk,
ref logic[4:0] fb_addr_y,
ref logic[5:0] fb_addr_x,
ref logic fb_writedata,
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);

fb_addr_y = 5'b00001;
fb_addr_x = 6'b00001;
fb_writedata = 0;
fb_WE = 0;
repeat(2) @(posedge clk);

repeat(2) @(posedge clk);
assert (fb_readdata == 1'h1) begin
  $display("WriteOne TEST 1 : PASSED");
total = total + 1;
end
else $error("OR TEST 1 : FAILED (Got %h, Expected 1)",
          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

task automatic testWriteOneReadElse(ref logic clk,
                                      ref logic[4:0] fb_addr_y,
                                      ref logic[5:0] fb_addr_x,
                                      ref logic fb_writedata,
                                      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
task automatic testWriteManyReadMany(ref logic clk,
    ref logic[4:0] fb_addr_y,
    ref logic[5:0] fb_addr_x,
    ref logic fb_writedata,
    ref logic fb_WE,
    ref logic fb_readdata,
    ref logic reset,
    ref int total);
repeat(4) @(posedge clk);

    fb_addr_y = 5'b00010;
    fb_addr_x = 6'b000010;
    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);

    fb_addr_y = 5'b00100;
    fb_addr_x = 6'b001000;
    fb_writedata = 1;
    fb_WE = 1;
    repeat(2) @(posedge clk);

    fb_addr_y = 5'b01000;
    fb_addr_x = 6'b010000;
    fb_writedata = 1;
    fb_WE = 1;
    repeat(2) @(posedge clk);

    fb_addr_y = 5'b01000;
    fb_addr_x = 6'b010000;
    fb_writedata = 1;
    fb_WE = 1;
    repeat(2) @(posedge clk);

    fb_addr_y = 5'b10000;
    fb_addr_x = 6'b010000;
    fb_writedata = 1;
repeat(2) @(posedge clk);

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

fb_addr_y = 5'b000010;
fb_addr_x = 6'b000010;
fb_writedata = 0;
fb_WE = 0;
repeat(2) @(posedge clk);

repeat(2)
@ (posedge clk);
assert (fb_readdata == 1'h1) begin
   $display ("WriteManyReadMany TEST 3 part 1 : PASSES");
end
else $error("WriteManyReadMany TEST 3 part 1 : FAILED (Got %h, Expected 1)", fb_readdata);

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

fb_addr_y = 5'b00010;
fb_addr_x = 6'b000100;
fb_writedata = 0;
fb_WE = 0;
repeat(2) @(posedge clk);

repeat(2)
@posedge clk;
assert (fb_readdata == 1'h1) begin
   $display ("WriteManyReadMany TEST 3 part 2 : 
      → PASSED");
end
else $error("WriteManyReadMany TEST 3 part 2 : FAILED
      → (Got %h, Expected 1", fb_readdata);

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

fb_addr_y = 5'b01000;
fb_addr_x = 6'b001000;
fb_writedata = 0;
fb_WE = 0;
repeat(2) @posedge clk;

repeat(2)
@posedge clk;
assert (fb_readdata == 1'h1) begin
   $display ("WriteManyReadMany TEST 3 part 3 : 
      → PASSED");
end
else $error("WriteManyReadMany TEST 3 part 3 : FAILED
      → (Got %h, Expected 1", fb_readdata);

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

fb_addr_y = 5'b10000;
fb_addr_x = 6'b010000;
fb_writedata = 0;
fb_WE = 0;
repeat(2) @(posedge clk);

repeat(2)
 @(posedge clk);
assert (fb_readdata == 1'h1) begin
   $display("WriteManyReadMany TEST 3 part 4 :
   \rightarrow PASSED");
end
else $error("WriteManyReadMany TEST 3 part 4 : FAILED
\rightarrow (Got %h, Expected 1)", fb_readdata);

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

fb_addr_y = 5'b10000;
fb_addr_x = 6'b100000;
fb_writedata = 0;
fb_WE = 0;
repeat(2) @(posedge clk);

repeat(2)
 @(posedge clk);
assert (fb_readdata == 1'h0) begin
   $display("WriteManyReadMany TEST 3 part 5 :
   \rightarrow PASSED");
   total = total + 1;
end
else $error("OR TEST 3 part 5 : FAILED (Got %h,
\rightarrow 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

module fb_testbench();
  logic clk;
  logic reset;
  logic [4:0] fb_addr_y;
  logic [5:0] fb_addr_x;
  logic fb_writedata;
  logic fb_WE;
  logic fb_readdata;
  logic [7:0] VGA_R, VGA_G, VGA_B;
  logic VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n;
  logic VGA_SYNC_n;
  int total;

  Chip8_framebuffer dut(.);

  initial begin
    clk = 0;
    reset = 0;
    fb_addr_y = 5'b00000;
    fb_addr_x = 6'b000000;
    fb_writedata = 0;
    fb_WE = 0;
    total = 0;
    forever
      #20ns clk = ~clk;
    end
```
initial begin

$display("Starting test script...");
testWriteOne(clk, fb_addr_y, fb_addr_x, fb_writedata, fb_WE, fb_readdata, reset, total);
testWriteOneReadElse(clk, fb_addr_y, fb_addr_x, fb_writedata, fb_WE, fb_readdata, reset, total);
testWriteManyReadMany(clk, fb_addr_y, fb_addr_x, fb_writedata, fb_WE, fb_readdata, reset, total);

repeat(2) @(posedge clk);

end

endmodule

7.2.8  alu_testbench.sv

/*****************************************************************************/
/* alu_testbench.sv */
/* Contains tests for the following 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, Gabrielle Taylor
* Dependencies:
  * - enums.svh
  * - Chip8_CPU/Chip8_ALU.sv

'include "enums.svh"

task automatic testReset(logic [15:0] input1, input2,
       ALU_f alu_op);
   input1 = 16’h0000;
   input2 = 16’h0000;
   alu_op = ALU_f_NOP;
endtask

/**
* Tests the OR instruction for the ALU
*
* @test 1
* @input input1 = 16’hF5A0
* @input input2 = 16’hFA50
* @input alu_op = ALU_f_OR
* @expected result = 16’FFF0
* @expected alu_carry = 1’b0
*/
task automatic testOR(ref logic clk, alu_carry,
       ref logic [15:0] input1, input2, result,
       ref ALU_f alu_op,
       ref int total);
  //Setup test 1
  repeat(2)
      @(posedge clk);
      input1 = 16’hF5A0;
      input2 = 16’hFA50;
      alu_op = ALU_f_OR;
repeat(2)
    @(posedge clk);
assert (result == 16'hFFF0 && alu_carry == 1'b0) begin
    $display("OR TEST 1 : PASSED");
    total = total + 1;
end
else error("OR TEST 1 : FAILED (Got %h, Expected fff0)",
              result);

    testReset(input1, input2, alu_op);
endtask

/**
* Tests the AND instruction for the ALU
 *
* @test 1
* @input input1 = 16'hF5A0
* @input input2 = 16'hFA50
* @input alu_op = ALU_f_AND
* @expected result = 16'F000
* @expected alu_carry = 1'b0
*/
task automatic testAND(ref logic clk, alu_carry,
                      ref logic [15:0] input1, input2, result,
                      ref ALU_f alu_op,
                      ref int total);
//Setup test 1
repeat(2)
    @(posedge clk);
input1 = 16'hF5A0;
input2 = 16'hFA50;
alu_op = ALU_f_AND;
repeat(2)
    @(posedge clk);
assert (result == 16'hF000 && alu_carry == 1'b0) begin
  $display("AND TEST 1 : PASSED");
  total = total + 1;
end
else $error("AND TEST 1 : FAILED (Got %h, Expected f000", result);

  testReset(input1, input2, alu_op);
endtask

/**
 * Tests the XOR instruction for the ALU
 *
 * @test 1
 * @input input1 = 16'hF5A0
 * @input input2 = 16'hFA50
 * @input alu_op = ALU_f_XOR
 * @expected result = 16'h0FF0
 * @expected alu_carry = 1'b0
 */
task automatic testXOR(ref logic clk, alu_carry,
  ref logic [15:0] input1, input2, result,
  ref ALU_f alu_op,
  ref int total);
//Setup test 1
repeat(2)
  @(posedge clk);
input1 = 16'hF5A0;
input2 = 16'hFA50;
alu_op = ALU_f_XOR;
repeat(2)
  @(posedge clk);
assert (result == 16'h0FF0 && alu_carry == 1'b0) begin
  $display("XOR TEST 1 : PASSED");
  total = total + 1;
end
```verilog
end

else $error("XOR TEST 1 : FAILED (Got %h, Expected 0ff0)",
                   result);

testReset(input1, input2, alu_op);
endtask

/**
* Tests the MINUS instruction for the ALU
*
* @test 1
* @input input1 = 16’d180
* @input input2 = 16’d180
* @input alu_op = ALU_f_ADD
* @expected result = 16’d360
* @expected alu_carry = 1’b1
* 
* @test 2
* @input input1 = 16’d5
* @input input2 = 16’d5
* @input alu_op = ALU_f_ADD
* @expected result = 16’d10
* @expected alu_carry = 1’b1
*/
task automatic testADD(ref logic clk, alu_carry,
                        ref logic [15:0] input1, input2, result,
                        ref ALU_f alu_op,
                        ref int total);

//Setup test 1
repeat(2)
  @(posedge clk);
input1 = 16’d180;
input2 = 16’d180;
alu_op = ALU_f_ADD;
repeat(2)
```
assert (result == 16'd360 && alu_carry == 1'b1) begin
  $display ("ADD TEST 1 : PASSED");
total = total + 1;
end
else $error("ADD TEST 1 : FAILED (Got %d, Expected 360)
  \(\sim\) (Got %d, Expected 1)", result, alu_carry);

//Setup test 2
repeat(2)
  @(posedge clk);
input1 = 16'd5;
input2 = 16'd5;
alu_op = ALU_f_ADD;
repeat(2)
  @(posedge clk);
assert (result == 16'd10 && alu_carry == 1'b0) begin
  $display ("ADD TEST 2 : PASSED");
total = total + 1;
end
else $error("ADD TEST 2 : FAILED (Got %d, Expected 10)
  \(\sim\) (Got %d, Expected 0)", result, alu_carry);

  testReset(input1, input2, alu_op);
endtask

/**
* Tests the MINUS instruction for the ALU
*
* @test 1
* @input input1 = 16'hC3C3
* @input input2 = 16'hC3C3
* @input alu_op = ALU_f_MINUS
* @expected result = 16'h0000
* @expected alu_carry = 1'b0
* @test 2
* @input input1 = 16’hE0A5
* @input input2 = 16’h7003
* @input alu_op = ALU_f_MINUS
* @expected result = 16’h70A2
* @expected alu_carry = 1'b0
*
* @test 3
* @input input1 = 16’h7003
* @input input2 = 16’hE0A5
* @input alu_op = ALU_f_MINUS
* @expected result = 16’d36702
* @expected alu_carry = 1'b0
*/
task automatic testMINUS(ref logic clk, alu_carry,
    ref logic [15:0] input1, input2, result,
    ref ALU_f alu_op,
    ref int total);

    //Setup test 1
repeat(2)
    @(posedge clk);
input1 = 16’hC3C3;
input2 = 16’hC3C3;
alu_op = ALU_f_MINUS;
repeat(2)
    @(posedge clk);
assert (result == 16’h0000 && alu_carry == 1'b0) begin
    $display ("MINUS TEST 1 : PASSED");
total = total + 1;
end
else $error("MINUS TEST 1 : FAILED (Got %h, Expected 0000)
            (Got %d, Expected 1)", result, alu_carry);

    //Setup test 2
repeat(2)
 @(posedge clk);
 input1 = 16'hE0A5;
 input2 = 16'h7003;
 alu_op = ALU_f_MINUS;

repeat(2)
 @(posedge clk);
 assert (result == 16'h70A2 && alu_carry == 1'b0) begin
   $display("MINUS TEST 2 : PASSED");
   total = total + 1;
 end
 else $error("MINUS TEST 2 : FAILED (Got %h, Expected 70a2)
                     (Got %d, Expected 1)", result, alu_carry);

//Setup test 3
repeat(2)
 @(posedge clk);
 input1 = 16'h7003;
 input2 = 16'hE0A5;
 alu_op = ALU_f_MINUS;

repeat(2)
 @(posedge clk);
 assert (result == 16'd36702 && alu_carry == 1'b1) begin
   $display("MINUS TEST 3 : PASSED");
   total = total + 1;
 end
 else $error("MINUS TEST 3 : FAILED (Got %d, Expected 36702)
                     (Got %d, Expected 1)", result, alu_carry);

testReset(input1, input2, alu_op);
endtask

/**
 * Tests the lSHIFT instruction for the ALU
*/
* @test 1
* @input input1 = 16'h0031
* @input input2 = 16'h0002
* @input alu_op = ALU_f_LSHIFT
* @expected result = 16'h00C4
* @expected alu_carry = 1'b0
*
* @test 2
* @input input1 = 16'h1111
* @input input2 = 16'h0001
* @input alu_op = ALU_f_LSHIFT
* @expected result = 16'h2222
* @expected alu_carry = 1'b0
*/

task automatic testLSHIFT(ref logic clk, alu_carry,
    ref logic [15:0] input1, input2, result,
    ref ALU_f alu_op,
    ref int total);

  //Setup test 1
  repeat(2)
    @(posedge clk);
    input1 = 16'h0031;
    input2 = 16'h0002;
    alu_op = ALU_f_LSHIFT;
  repeat(2)
    @(posedge clk);
    assert (result == 16'h00C4 && alu_carry == 1'b0) begin
      $display("LSHIFT TEST 1 : PASSED");
      total = total + 1;
    end
  else
    $error("LSHIFT TEST 1 : FAILED (Got %h, Expected \00c4)", result);

  //Setup test 2
  repeat(2)
@posedge clk;
input1 = 16'h1111;
input2 = 16'h0001;
alu_op = ALU_f_LSHIFT;

repeat(2)
  @posedge clk;
  assert (result == 16'h2222 && alu_carry == 1'b0) begin
    $display("LSHIFT TEST 2 : PASSED");
    total = total + 1;
  end
else $error("LSHIFT TEST 2 : FAILED (Got %h, Expected ~ 2222)", result);

  testReset(input1, input2, alu_op);
endtask

/**************************************************************************
  * Tests the RSHIFT instruction for the ALU
  *
  * @test 1
  * @input input1 = 16'h0031
  * @input input2 = 16'h0002
  * @input alu_op = ALU_f_RSHIFT
  * @expected result = 16'h000C
  * @expected alu_carry = 1'b0
  *
  * @test 2
  * @input input1 = 16'h1111
  * @input input2 = 16'h0001
  * @input alu_op = ALU_f_RSHIFT
  * @expected result = 16'h0888
  * @expected alu_carry = 1'b0
  */
task automatic testRSHIFT(ref logic clk, alu_carry, 
  ref logic [15:0] input1, input2, result,
ref ALU_f alu_op,
    ref int total);

//Setup test 1
repeat(2)
    @(posedge clk);
input1 = 16'h0031;
input2 = 16'h0002;
alu_op = ALU_f_RSHIFT;

repeat(2)
    @(posedge clk);
assert (result == 16'h000C && alu_carry == 1'b0) begin
    $display("RSHIFT TEST 1 : PASSED");
    total = total + 1;
end
else $error("RSHIFT TEST 1 : FAILED (Got %h, Expected ^ 000c)", result);

//Setup test 2
repeat(2)
    @(posedge clk);
input1 = 16'h1111;
input2 = 16'h0001;
alu_op = ALU_f_RSHIFT;

repeat(2)
    @(posedge clk);
assert (result == 16'h0888 && alu_carry == 1'b0) begin
    $display("RSHIFT TEST 2 : PASSED");
    total = total + 1;
end
else $error("RSHIFT TEST 2 : FAILED (Got %h, Expected ^ 0888)", result);

testReset(input1, input2, alu_op);
endtask
/**
 * Tests the GREATER instruction for the ALU
 *
 * @test 1
 * @input input1 = 16’d180
 * @input input2 = 16’d15
 * @input alu_op = ALU_f_GREATER
 * @expected result = 16’d1
 * @expected alu_carry = 1'b0
 *
 * @test 2
 * @input input1 = 16’d15
 * @input input2 = 16’d180
 * @input alu_op = ALU_f_GREATER
 * @expected result = 16’d0
 * @expected alu_carry = 1'b0
 *
 * @test 3
 * @input input1 = 16’d15
 * @input input2 = 16’d15
 * @input alu_op = ALU_f_GREATER
 * @expected result = 16’d0
 * @expected alu_carry = 1'b0
 */

task automatic testGREATER(ref logic clk, alu_carry,
    ref logic [15:0] input1, input2, result,
    ref ALU_f alu_op,
    ref int total);

    //Setup test 1
    repeat(2)
        @(posedge clk);
        input1 = 16’d180;
        input2 = 16’d15;
        alu_op = ALU_f_GREATER;
        repeat(2)
            @(posedge clk);
assert (result == 16'd1 && alu_carry == 1'b0) begin
    $display ("GREATER TEST 1 : PASSED");
    total = total + 1;
end
else $error("GREATER TEST 1 : FAILED (Got %d, Expected ~ 1)", result);

//Setup test 2
repeat(2)
    @(posedge clk);
    input1 = 16'd15;
    input2 = 16'd180;
    alu_op = ALU_f_GREATER;
repeat(2)
    @(posedge clk);
assert (result == 16'd0 && alu_carry == 1'b0) begin
    $display ("GREATER TEST 2 : PASSED");
    total = total + 1;
end
else $error("GREATER TEST 2 : FAILED (Got %d, Expected ~ 0)", result);

//Setup test 3
repeat(2)
    @(posedge clk);
    input1 = 16'd15;
    input2 = 16'd15;
    alu_op = ALU_f_GREATER;
repeat(2)
    @(posedge clk);
assert (result == 16'd0 && alu_carry == 1'b0) begin
    $display ("GREATER TEST 3 : PASSED");
    total = total + 1;
end
```verilog
testReset(input1, input2, alu_op);
endtask

/**
 * Tests the EQUALS instruction for the ALU
 * @test 1
 * @input input1 = 16'd8
 * @input input2 = 16'd8
 * @input alu_op = ALU_f_EQUALS
 * @expected result = 16'd1
 * @expected alu_carry = 1'b0
 * @test 2
 * @input input1 = 16'd8
 * @input input2 = 16'd9
 * @input alu_op = ALU_f_EQUALS
 * @expected result = 16'd0
 * @expected alu_carry = 1'b0
 */
task automatic testEQUALS(ref logic clk, alu_carry,
                          ref logic [15:0] input1, input2, result,
                          ref ALU_f alu_op,
                          ref int total);
  //Setup test 1
  repeat(2)
    @(posedge clk);
  input1 = 16'd8;
  input2 = 16'd8;
  alu_op = ALU_f_EQUALS;
  repeat(2)
    @(posedge clk);
```
assert (result == 16'd1 && alu_carry == 1'b0) begin
    $display ("EQUALS TEST 1 : PASSED");
    total = total + 1;
end
else $error("EQUALS TEST 1 : FAILED (Got %d, Expected 1)",
               result);

//Setup test 2
repeat(2)
    @(posedge clk);
    input1 = 16'd8;
    input2 = 16'd9;
    alu_op = ALU_f_EQUALS;
repeat(2)
    @(posedge clk);
assert (result == 16'd0 && alu_carry == 1'b0) begin
    $display ("EQUALS TEST 2 : PASSED");
    total = total + 1;
end
else $error("EQUALS TEST 2 : FAILED (Got %d, Expected 0)",
              result);

    testReset(input1, input2, alu_op);
endtask

/**
 * Tests the INC instruction for the ALU
 * @test 1
 * @input input1 = 16'd8
 * @input alu_op = ALU_f_INC
 * @expected result = 16'd26
 * @expected alu_carry = 1'b0
 */
task automatic testINC(ref logic clk, alu_carry,
ref logic [15:0] input1, input2, result,
ref ALU_f alu_op,
ref int total);

//Setup
repeat(2)
  @(posedge clk);
input1 = 16’d8;
input2 = 16’h0000;
alu_op = ALU_f_INC;
repeat(16) begin
  @(posedge clk);
  input1 = result;
end

//Check
repeat(2)
  @(posedge clk);
assert (result == 16’d25 && alu_carry == 1’b0) begin
  $display (“INC TEST : PASSED”);
  total = total + 1;
end
else $error(“INC TEST : FAILED (Got %d, Expected 24)",
             result);

testReset(input1, input2, alu_op);
endtask

module alu_testbench();
  logic clk;
  logic alu_carry;
  logic [15:0] result;
  logic[15:0] input1, input2;
  ALU_f alu_op;
  int total;

  Chip8_ALU dut(
.input1(input1),
.input2(input2),
.sel(alu_op),
.out(result),
.alu_carry(alu_carry));

initial begin
  clk = 0;
  input1 = 16'h0000;
  input2 = 16'h0000;
  alu_op = ALU_f_NOP;
  forever
    #20ns clk = ~clk;
end

initial begin
  $display("Starting test script...");
testOR(clk, alu_carry, input1, input2, result, alu_op, total);
testAND(clk, alu_carry, input1, input2, result, alu_op, total);
testXOR(clk, alu_carry, input1, input2, result, alu_op, total);
testADD(clk, alu_carry, input1, input2, result, alu_op, total);
testMINUS(clk, alu_carry, input1, input2, result, alu_op, total);
testLSHIFT(clk, alu_carry, input1, input2, result, alu_op, total);
testRSHIFT(clk, alu_carry, input1, input2, result, alu_op, total);
testGREATER(clk, alu_carry, input1, input2, result, alu_op, total);
testEQUALS(clk, alu_carry, input1, input2, result, alu_op, total);
testINC(clk, alu_carry, input1, input2, result, alu_op, total);
$display("TESTS PASSED : %d", total);
end
endmodule

7.3 Linux Code

7.3.1 chip8.c

/*
 * Userspace program that communicates with the vga_ball
device driver
* primarily through ioctls
*
* David Watkins (djw2146), Ashley Kling (ask2203)
* Columbia University
*/

#include <stdio.h>
#include "chip8driver.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
#include <pthread.h>

#include "usbkeyboard.h"

static int CHIP8_FONTSET[] =
{
  0xF0, 0x90, 0x90, 0x90, 0xF0, //0
  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);
    }
}
```c
void chip8_read(chip8_opcode *op) {
    if(ioctl(chip8_fd, CHIP8_READ_ATTR, op)) {
        perror("ioctl(CHIP8_READ_ATTR) failed");
        printf("(%d, %d)\n", op->addr, op->data);
        quit_program(0);
    }
}

void setFramebuffer(int x, int y, int value) {
    chip8_opcode op;
    op.addr = FRAMEBUFFER_ADDR;
    op.data = (1 << 12) | ((value & 0x1) << 11) | ((x & 0x3f) << 5) | (y & 0x1f);
    chip8_write(&op);
}

int readFramebuffer(int x, int y) {
    chip8_opcode op;
    op.addr = FRAMEBUFFER_ADDR;
    op.data = (0 << 12) | (0 << 11) | ((x & 0x3f) << 5) | (y & 0x1f);
    chip8_read(&op);
    return op.readdata;
}

void flipPixel(int x, int y) {
    int px = readFramebuffer(x, y);
    setFramebuffer(x, y, !px);
}

void setMemory(int address, int data) {
    chip8_opcode op;
    op.addr = MEMORY_ADDR;
    op.data = (1 << 20) | ((address & 0xffffff) << 8) | (data & 0xff);
    chip8_write(&op);
}
```
int readMemory(int address) {
    chip8_opcode op;
    op.addr = MEMORY_ADDR;
    op.data = ((address & 0xfff) << 8) | (0 & ~ 0xff);
    chip8_read(&op);
    return (op.readdata & 0xff);
}

void setIRegister(int data) {
    chip8_opcode op;
    op.addr = I_ADDR;
    op.data = (data & 0xffff);
    chip8_write(&op);
}

int readIRegister() {
    chip8_opcode op;
    op.addr = I_ADDR;
    chip8_read(&op);
    return op.readdata;
}

int readRegister(int reg) {
    chip8_opcode op;
    op.addr = V0_ADDR + 4 * (reg & 0xf);
    chip8_read(&op);
    return op.readdata;
}

void writeRegister(int reg, int value) {
    chip8_opcode op;
    op.addr = V0_ADDR + 4 * (reg & 0xf);
    op.data = value & 0xff;
    chip8_write(&op);
}
* Load the font set onto the chip8 sequentially
* Uses the op codes specified in chip8driver.h
*/

void loadfontset() {
    int i;
    for(i = 0; i < FONTSET_LENGTH; ++i) {
        setMemory(i, CHIP8_FONTSET[i]);
        // int mem_val = readMemory(i);
        // printf("(Address: %d) Wrote: %d, Read: %d\n", i,
                CHIP8_FONTSET[i], mem_val);
        int got = readMemory(i);
        if (CHIP8_FONTSET[i] != got) {
            printf("Memory mismatch (expected: %d, got:
                    %d)\n", CHIP8_FONTSET[i], got);
        }
    }
}

void refreshFrameBuffer() {
    int x, y;
    for(x = 0; x < 64; ++x) {
        for(y = 0; y < 32; ++y) {
            // int mem_val = readFramebuffer(x, y);
            setFramebuffer(x, y, 0);

            // printf("(x: %d, y: %d) Wrote: %d, Read: %d\n",
                    x, y, !mem_val, mem_val);
        }
    }
}

/*
* Load a ROM file byte by byte onto the chip8
* Uses the op codes specified in chip8driver.h
*/
void loadROM(const char* romfilename) {
    FILE *romfile;
char buffer;
long filelen;
int i;

romfile = fopen(romfilename, "rb");
ffile(filelen, 0, SEEK_END);
filelen = ftell(romfile);
rewind(romfile);

for(i = 0; i < filelen && i < MEMORY_END - MEMORY_START;
    i++) {
    fread(&buffer, 1, 1, romfile);
    setMemory(MEMORY_START + i, buffer);
    int got = readMemory(MEMORY_START + i);
    if (buffer != got) {
        printf("Memory mismatch (expected: %d, got: %d)\n", buffer, got);
    }
}

fclose(romfile); // Close the file

void resetMemory() {
    int i;
    for(i = 0; i < MEMORY_END; i++) {
        setMemory(i, 0);
    }
}

void startChip8() {
    chip8_opcode op;
    op.addr = STATE_ADDR;
void pauseChip8() {
    chip8_opcode op;
    op.addr = STATE_ADDR;
    op.data = PAUSED_STATE;
    chip8_write(&op);
}

void runInstructionChip8() {
    chip8_opcode op;
    op.addr = STATE_ADDR;
    op.data = RUN_INSTRUCTION_STATE;
    chip8_write(&op);
}

int chip8isRunning() {
    chip8_opcode op;
    op.addr = STATE_ADDR;
    chip8_read(&op);
    return op.readdata == RUNNING_STATE;
}

int chip8isPaused() {
    chip8_opcode op;
    op.addr = STATE_ADDR;
    chip8_read(&op);
    return op.readdata == PAUSED_STATE;
}

int chip8isRunInstruction() {
    chip8_opcode op;
    op.addr = STATE_ADDR;
    op.data = PAUSED_STATE;
}
chip8_read(&op);
return op.readdata == RUN_INSTRUCTION_STATE;
}

int readPC() {
    chip8_opcode op;
    op.addr = PROGRAM_COUNTER_ADDR;
    chip8_read(&op);
    // fprintf(fp, "Instruction: %04x, PC: %d\n",
    → (op.readdata & 0xffffffff) >> 12, (op.readdata &
    → 0xff));
    return (op.readdata & 0xffff);
}

void writePC(int pc) {
    chip8_opcode op;
    op.addr = PROGRAM_COUNTER_ADDR;
    op.data = pc;
    chip8_write(&op);
}

void printMemory() {
    int i = 0;
    for(i = 0; i < MEMORY_END; ++i) {
        printf("%d ", readMemory(i));
    }
    printf("\n");
}

void resetStack() {
    chip8_opcode op;
    op.addr = STACK_ADDR;
    chip8_write(&op);
}

int readSoundTimer() {
    chip8_opcode op;
    op.addr = SOUND_TIMER_ADDR;

    return op.readdata & 0xff;
}
chip8_read(&op);
return op.readdata;
}

void writeSoundTimer(int value) {
    chip8_opcode op;
    op.addr = SOUND_TIMER_ADDR;
    op.data = value;
    chip8_write(&op);
}

int readDelayTimer() {
    chip8_opcode op;
    op.addr = DELAY_TIMER_ADDR;
    chip8_read(&op);
    return op.readdata;
}

void writeDelayTimer(int value) {
    chip8_opcode op;
    op.addr = DELAY_TIMER_ADDR;
    op.data = value;
    chip8_write(&op);
}

void writeInstruction(int instruction) {
    chip8_opcode op;
    op.addr = INSTRUCTION_ADDR;
    op.data = instruction;
    chip8_write(&op);
    usleep(1000000);
}

int readInstruction() {
    chip8_opcode op;
    op.addr = INSTRUCTION_ADDR;
    chip8_read(&op);
return op.readdata;
}

void chip8Writekeypress(char val, unsigned int ispressed) {
    chip8_opcode op;
    op.addr = KEY_PRESS_ADDR;
    op.data = ((ispressed & 0x1) << 4) | (val & 0xf);
    chip8_write(&op);
}

void printKeyState() {
    chip8_opcode op;
    op.addr = KEY_PRESS_ADDR;
    chip8_read(&op);
    printf("Is pressed: %d, Key val: %d, raw value: %d\n",
           (op.readdata & 0x10) >> 4, (op.readdata & 0xf),
           op.readdata);
}

void writeReset() {
    chip8_opcode op;
    op.addr = RESET_ADDR;
    chip8_write(&op);
}

void printStatus(FILE *out, int index) {
    printf(out, "Status %d\n", index);
    if(chip8isPaused()) {
        printf(out, "Paused\n");
    } else if(chip8isRunning()) {
        printf(out, "Running\n");
    } else {
        printf(out, "Run Instruction\n");
    }
    int pc = readPC();
}
int mem = readMemory(pc);
int mem2 = readMemory(pc + 1);
fprintf(out, "Program counter is: %d, instruction is: %04x
\rightarrow / %04x\n", pc, mem << 4 | mem2, readInstruction());
fprintf(out, "I register: %d\n", readIRegister());
int i;
for(i = 0; i < 0x10; ++i) {
    fprintf(out, "v%d: %d\n", i, readRegister(i));
}

fprintf(out, "Sound timer: %d\n", readSoundTimer());
fprintf(out, "Delay timer: %d\n\n", readDelayTimer());
}

void resetChip8(const char* filename) {
    // Need to write to registers and all
    // Reload font set etc.
    pauseChip8();
    resetMemory();

    loadfontset();
    if(filename != 0) {
        loadROM(filename);
        refreshFrameBuffer();
    }

    int i;
    for(i = 0; i < 0x10; ++i) {
        writeRegister(i, 0);
    }

    // printMemory();
    writePC(0x200);
    setIRegister(0);
    resetStack();
    chip8writekeypress(0, 0);
    writeSoundTimer(0);
    writeDelayTimer(0);
printStatus(stdout, 0);
}

/*
* Checks to see if a key is pressed, or depressed
* Then writes the associated action to the chip8 device
*/

void checkforkeypress(const char *file) {
    struct usb_keyboard_packet packet;
    int transferred;
    char keystate[12];

    libusb_interrupt_transfer(keyboard, endpoint_address,
    (unsigned char *) &packet, sizeof(packet),
    &transferred, 0);

    if (transferred == sizeof(packet)) {
        sprintf(keystate, "%02x %02x %02x", packet.modifiers,
        packet.keycode[0], packet.keycode[1]);
        char val[1];
        if (kbiskeypad(&packet, val)) {
            chip8writekeypress(val[0], 1);
        } else if (kbisstart(&packet)) {
            startChip8();
        } else if (kbispause(&packet)) {
            pauseChip8();
        } else if (kbisreset(&packet)) {
            resetChip8(file);
        } else {
            chip8writekeypress(0, 0);
        }
    }
    else {
        printf("Size mismatch %d %d\n", sizeof(packet),
        transferred);
    }
}

void *status_thread_f(void *ignored) {

int index = 0;
while(1) {
    printStatus(fp, index++);
    usleep(4000);
}

return NULL;

int main(int argc, char** argv)
{
    int runType = 0;
    if(argc != 2 && argc != 3) {
        printf("Usage: chip8 <romfilename>\n");
        exit(1);
    }

    if(argc == 3) runType = 1;

    /* Open the keyboard */
    if ( (keyboard = openkeyboard(&endpoint_address)) == NULL ) {
        fprintf(stderr, "Did not find a keyboard\n");
        exit(1);
    }

    static const char filename[] = "/dev/vga_led";
    if ( (chip8_fd = open(filename, O_RDWR)) == -1 ) {
        fprintf(stderr, "could not open %s\n", filename);
        return -1;
    }

    signal(SIGINT, quit_program);

    fp = fopen("log.txt", "w+");

    pthread_t status_thread;
if(runType == 0) {
    resetChip8(argv[1]);
    // pthread_create(&status_thread, NULL,
    -> status_thread_f, NULL);

    while(chip8isRunning() || chip8isPaused()) {
        // printStatus(stdout, 0);
        checkforkeypress(argv[1]);
        printKeyState();
    }

    /* Terminate the status thread */
    // pthread_cancel(status_thread);

    /* Wait for the status thread to finish */
    // pthread_join(status_thread, NULL);
}

fclose(fp);

printf("Chip8 is terminating\n");
close(chip8_fd);
return 0;
}

7.3.2 chip8driver.c

/*
 * Device driver for the CHIP8 SystemVerilog Emulator
 *
 * A Platform device implemented using the misc subsystem
 *
 * Columbia University
 *
 * References:
 * Linux source: Documentation/player-model/platform.txt
 * drivers/misc/arm-charlcd.c
 */
"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;
    }
}
case I_ADDR: return 1;

// Timer instructions are always okay
case SOUND_TIMER_ADDR: return 1;
case DELAY_TIMER_ADDR: return 1;

// Stack instructions are only valid if they conform to stack size
// Always looks at last three nibbles
case STACK_POINTER_ADDR: return !isWrite || (instruction >= 0 && instruction < 64);
case STACK_ADDR: return 1;

// Handle state transition
case STATE_ADDR: switch(instruction) {
    case RUNNING_STATE: return 1;
    case RUN_INSTRUCTION_STATE: return 1;
    case PAUSED_STATE: return 1;
    default: return !isWrite;
}

// Memory address
case MEMORY_ADDR:
    /0000_0000_0001_AAAA_AAAA_AAAA_DDDD_DD
if(isWrite) return 1;
else return 2;

// Program Counter will always look at the last 3 nibbles
case PROGRAM_COUNTER_ADDR: return 1;

// Always considers last nibble
case KEY_PRESS_ADDR: return 1;

// Make sure X, Y, and data values conform
// Data value will always be 1 byte
case FRAMEBUFFER_ADDR:
    /0000_0000_0000_0001_DXXX_XXY_YYY
if(isWrite) return 1;
else return 2;

case INSTRUCTION_ADDR: return 1;
case RESET_ADDR: return 1;

default: break;
}

return 0;
}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long chip8_ioctl(struct file *f, unsigned int cmd,
                         unsigned long arg)
{
    chip8_opcode op;
    int isWrite = 0;

    switch (cmd) {
    case CHIP8_WRITE_ATTR:
        if (copy_from_user(&op, (chip8_opcode *) arg,
                           sizeof(chip8_opcode)))
            return -EACCES;
        if (!isValidInstruction(op.addr, op.data, 1))
            return -EINVAL;
        write_op(op.addr, op.data);
        break;

    case CHIP8_READ_ATTR:
        if (copy_from_user(&op, (chip8_opcode *) arg,
                           sizeof(chip8_opcode)))
            return -EACCES;
isWrite = isValidInstruction(op.addr, op.data, 0);
if(isWrite == 0)
    return -EINVAL;

if(isWrite == 2)
    write_op(op.addr, op.data);

op.readdata = read_value(op.addr);
if (copy_to_user((chip8_opcode *) arg, &op,
                 sizeof(chip8_opcode)))
    return -EACCES;
    break;

default:
    return -EINVAL;
}

return 0;

/* The operations our device knows how to do */
static const struct file_operations chip8_fops = {
    .owner = THIS_MODULE,
    .unlocked_ioctl = chip8_ioctl,
};

/* Information about our device for the "misc" framework --
like a char dev */
static struct miscdevice chip8_misc_device = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = DRIVER_NAME,
    .fops = &chip8_fops,
};

/ *
* Initialization code: get resources (registers) and
display
`static int __init chip8_probe(struct platform_device *pdev)`

```c
  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;
```
out_release_mem_region:
    release_mem_region(dev.res.start,
    resource_size(&dev.res));
out_deregister:
    misc_deregister(&chip8_misc_device);
    return ret;
}

/* Clean-up code: release resources */
static int chip8_remove(struct platform_device *pdev)
{
    iounmap(dev.virtbase);
    release_mem_region(dev.res.start,
    resource_size(&dev.res));
    misc_deregister(&chip8_misc_device);
    return 0;
}

/* Which "compatible" string(s) to search for in the Device
   Tree */
#ifdef CONFIG_OF
static const struct of_device_id chip8_of_match[] = {
    { .compatible = "altr,vga_led" },
    {}},
};
MODULE_DEVICE_TABLE(of, chip8_of_match);
#endif

/* Information for registering ourselves as a "platform"
   driver */
static struct platform_driver chip8_driver = {
    .driver = {
        .name = DRIVER_NAME,
        .owner = THIS_MODULE,
        .of_match_table = of_match_ptr(chip8_of_match),
    },
    .remove = __exit_p(chip8_remove),
};
220
/* Called when the module is loaded: set things up */
static int __init chip8_init(void)
{
    pr_info(DRIVER_NAME " : init\n");
    return platform_driver_probe(&chip8_driver, chip8_probe);
}

/* Called when the module is unloaded: release resources */
static void __exit chip8_exit(void)
{
    platform_driver_unregister(&chip8_driver);
    pr_info(DRIVER_NAME " : exit\n");
}

module_init(chip8_init);
module_exit(chip8_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("The Chip8 Team");
MODULE_DESCRIPTION("Chip8 Emulator");

7.3.3  chip8driver.h

#ifndef __CHIP8_DRIVER_H__
#define __CHIP8_DRIVER_H__

#include <linux/ioctl.h>
#include <stdbool.h>

typedef struct {
    unsigned int data;
    unsigned int addr;
    unsigned int readdata;
} chip8_opcode;

#define CHIP8_MAGIC 'q'
/* ioctl's and their arguments */
#define CHIP8_WRITE_ATTR _IOW(CHIP8_MAGIC, 1, chip8_opcode *)
#define CHIP8_READ_ATTR _IOWR(CHIP8_MAGIC, 2, chip8_opcode *)

*/
* To write data to a particular register, use iowrite with
* NNNNNNX
* Where NNNNNN is ignored
* XX is the 8 bits to be written
* 
* To read from a register use ioread with one of the
* following addresses
*/
#define VO_ADDR 0x00
#define V1_ADDR 0x04
#define V2_ADDR 0x08
#define V3_ADDR 0x0C
#define V4_ADDR 0x10
#define V5_ADDR 0x14
#define V6_ADDR 0x18
#define V7_ADDR 0x1C
#define V8_ADDR 0x20
#define V9_ADDR 0x24
#define VA_ADDR 0x28
#define VB_ADDR 0x2C
#define VC_ADDR 0x30
#define VD_ADDR 0x34
#define VE_ADDR 0x38
#define VF_ADDR 0x3C

/* To write to the I index register
* NNNNNDDD
* DDDD is the 16 bits to write
*/
* Use ioread to read from the I register
*/
#define I_ADDR 0x40
/*
* 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
*/
#define SOUND_TIMER_ADDR 0x44
/*
* 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
*/
#define DELAY_TIMER_ADDR 0x48
/*
* 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
*/
#define STACK_POINTER_ADDR 0x60
/*
* To reset the stack, iowrite
*/
#define STACK_ADDR 0x4C
/*
* To write to the program counter
  * 0014DDDD
  * Where DDDD is the number to write to the program counter
  *
  * Use ioread to read from the program counter
  */
#define PROGRAM_COUNTER_ADDR 0x50

  /*
  To write a keypress to the Chip8 control unit
  * NNNNNND
  * Where D is the number corresponding to a keypress 0-F
  * Where P is whether a key is currently pressed or not (0x1, 0x0)
  */
#define KEY_PRESS_ADDR 0x54

  /*
  To change the state of the Chip8
  * 001600DD
  * Where DD is an 8-bit number corresponding to varying states
  * * 0x01 - Running
  * * 0x02 - Loading ROM
  * * 0x03 - Loading font set
  * * 0x04 - Paused
  * The state is initially set to loading font set
  *
  * Use ioread to read the state of the Chip8
  */
#define STATE_ADDR 0x58
#define RUNNING_STATE 0x0
#define RUN_INSTRUCTION_STATE 0x1
#define PAUSED_STATE 0x2

  /*
  To write to a location in memory
  * 0000_0000_0001_AAAA_AAAA_AAAA_DDDD_DDDD
  */
* 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
*/
#define MEMORY_ADDR 0x64

/* In order to write data to the framebuffer
  * 0000_0000_0000_0000_0001_DXXX_XXXY_YYYY
  * Where XX is the x position (6 bits)
  * Where YY is the y position (5 bits)
  * Where DD is the 8-bits of data to write to the screen
  *
  * 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
*/
#define FRAMEBUFFER_ADDR 0x5C
#define SCREEN_HEIGHT 480
#define SCREEN_WIDTH 640
#define MAX_FBX 32 //32/8
#define MAX_FBY 64

/* In order to write data to the instruction
  * 0000_0000_0000_0000_INNN_INNN_INNN_INNN
  * 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
*/
#define INSTRUCTION_ADDR 0x68
/**
* iowrite RESET_ADDR to reset all internal values
*/
#define RESET_ADDR 0x6C
#endif //__CHIP8_DRIVER_H__

7.3.4 Makefile

ifeq ($(KERNELRELEASE),)
# KERNELRELEASE defined: we are being compiled as part of the Kernel
    obj-m := chip8driver.o
else
# We are being compiled as a module: use the Kernel build system
  KERNEL_SOURCE := /usr/src/linux*
PWD := $(shell pwd)
CFLAGS = -Wall
OBJECTS = chip8.o usbkeyboard.o

default: module chip8
module:
    $(MAKE) -C $(KERNEL_SOURCE) SUBDIRS=$(PWD) modules
chip8 : $(OBJECTS)
c
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:

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

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.dtisi"

{
    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 environment variable contents * to be added to the gmac1 device tree blob. */
        ethernet0 = &gmac1;
    };

    soc {
        clkmgr@ffd04000 {
            clocks {
                osc1 { 

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

dcan0: d_can0 @ ffcc00000 {
    status = "disabled";
};

dcan1: d_can1 @ ffcc10000 {
    status = "disabled";
};
dwmmc0 @ ff704000 {
    num-slots = <1>;
    supports-highspeed;
    broken-cd;
    altr,dw-mshc-ciu-div = <4>;
    altr,dw-mshc-sdr-timing = <0 3>;

    slot0 {
        reg = <0>;
        bus-width = <4>;
    }
};

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

ethernet @ ff702000 {
    phy-mode = "rgmii";
    phy-addr = <0xffffffff>; /* probe for phy addr */
};
i2c1: i2c0 @ ffcc05000 {
    status = "disabled";
i2c2: i2c0ffe06000 {
    status = "disabled";
};

i2c3: i2c0ffe07000 {
    status = "disabled";
};

qspi: spi0ffe705000 {
    compatible = "cadence,qspi";
    #address-cells = <1>;
    #size-cells = <0>;
    reg = <0xff705000 0x1000>,
        <0xffa00000 0x1000>;
    interrupts = <0 151 4>;
    master-ref-clk = <400000000>;
    ext-decoder = <0>; /* external decoder */
    num-chipselect = <4>;
    fifo-depth = <128>;
    bus-num = <2>;
}

flash0: n25q00000 {
    #address-cells = <1>;
    #size-cells = <1>;
    compatible = "n25q00";
    reg = <0>; /* chip select */
    spi-max-frequency = <100000000>;
    page-size = <256>;
    block-size = <16>; /* 2^16, 64KB */
    quad = <1>; /* 1-support quad */
    tshsl-ns = <200>;
    tsd2d-ns = <255>;
    tchsh-ns = <20>;
    tslch-ns = <20>;
}
partition0 { 
    /* 8MB for raw data. */ 
    label = "Flash 0 Raw Data"; 
    reg = <0x0 0x800000>; 
};

partition0800000 { 
    /* 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>; 
};
serial1@ffc03000 {
    clock-frequency = <100000000>;
};

usb0: usb@ffb00000 {
    status = "disabled";
};

usb1: usb@ffb40000 {
    ulpi-ddr = <0>;
};

i2c0: i2c@ffc04000 {
    speed-mode = <0>;
};

leds {
    compatible = "gpio-leds";
    hps0 {
        label = "hps_led0";
        gpios = <&gpio1 15 1>;
    };

    hps1 {
        label = "hps_led1";
        gpios = <&gpio1 14 1>;
    };

    hps2 {
        label = "hps_led2";
        gpios = <&gpio1 13 1>;
    };

    hps3 {
        label = "hps_led3";
        gpios = <&gpio1 12 1>;
    };
}
lightweight_bridge: bridge0x0ff200000 {
    #address-cells = <1>;
    #size-cells = <1>;
    ranges = < 0x0 0xff200000 0x200000 >;

    compatible = "simple-bus";
}

chip8: chip80 {
    compatible = "altr,chip8";
    reg = <0x0 0x2>;
};
}

&i2c0 {
    lcd: lcd828 {
        compatible = "newhaven,nhd-0216k3z-nsw-bbw";
        reg = <0x28>;
        height = <2>;
        width = <16>;
        brightness = <8>;
    };}

eeprom051 {
    compatible = "atmel,24c32";
    reg = <0x51>;
    pagesize = <32>;
};

rtc068 {
    compatible = "dallas,ds1339";
    reg = <0x68>;
};

233
# 7.3.6 usbkeyboard.c

```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 */

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

if ( (num_devs = libusb_get_device_list(NULL, &devs)) < 0 ) {
    fprintf(stderr, "Error: libusb_get_device_list failed\n");
    exit(1);
}

/* Look at each device, remembering the first HID device that speaks the keyboard protocol */

for (d = 0 ; d < num_devs ; d++) {
    libusb_device *dev = devs[d];
    if ( libusb_get_device_descriptor(dev, &desc) < 0 ) {
        fprintf(stderr, "Error: libusb_get_device_descriptor failed\n");
        exit(1);
    }

    if (desc.bDeviceClass == LIBUSB_CLASS_PER_INTERFACE) {
        struct libusb_config_descriptor *config;
        libusb_get_config_descriptor(dev, 0, &config);
        for (i = 0 ; i < config->bNumInterfaces ; i++)
            for ( k = 0 ; k < config->interface[i].num_altsetting ; k++)
                const struct libusb_interface_descriptor *
inter =
config->interface[i].altsetting + k ;
        if ( inter->bInterfaceClass == 
            LIBUSB_CLASS_HID &
inter->bInterfaceProtocol == 
            USB_HID_KEYBOARD_PROTOCOL) {
            int r;
if ((r = libusb_open(dev, &keyboard)) != 0) {
    fprintf(stderr, "Error: libusb_open failed: %d\n", r);
    exit(1);
}

if (libusb_kernel_driver_active(keyboard, i))
    libusb_detach_kernel_driver(keyboard, i);

libusb_set_auto_detach_kernel_driver(keyboard, i);

if ((r = libusb_claim_interface(keyboard, i)) != 0) {
    fprintf(stderr, "Error: libusb_claim_interface failed: %d\n", r);
    exit(1);
}

*endpoint_address = inter->endpoint[0].bEndpointAddress;

    goto found;
}

found:
libusb_free_device_list(devs, 1);

return keyboard;

/*
 * Check to see if any value in the keypad is currently pressed
 */
int kbiskeypad(struct usb_keyboard_packet* packet, char val[1]) {
    uint8_t keycode = packet->keycode[0];

    switch(keycode) {
    case KEY1: val[0] = 0x1; return 1; break;
    case KEY2: val[0] = 0x2; return 1; break;
    case KEY3: val[0] = 0x3; return 1; break;
    case KEYC: val[0] = 0xC; return 1; break;
    case KEY4: val[0] = 0x4; return 1; break;
    case KEY5: val[0] = 0x5; return 1; break;
    case KEY6: val[0] = 0x6; return 1; break;
    case KEYD: val[0] = 0xD; return 1; break;
    case KEY7: val[0] = 0x7; return 1; break;
    case KEY8: val[0] = 0x8; return 1; break;
    case KEY9: val[0] = 0x9; return 1; break;
    case KEYE: val[0] = 0xE; return 1; break;
    case KEYA: val[0] = 0xA; return 1; break;
    case KEY0: val[0] = 0x0; return 1; break;
    case KEYB: val[0] = 0xB; return 1; break;
    case KEYF: val[0] = 0xF; return 1; break;
    default: break;
    }

    return 0;
}

int kbisstart(struct usb_keyboard_packet* packet) {
    uint8_t keycode = packet->keycode[0];
    return keycode == KEY_START;
}

int kbispause(struct usb_keyboard_packet* packet) {
    uint8_t keycode = packet->keycode[0];
    return keycode == KEY_PAUSE;
}

int kbrisreset(struct usb_keyboard_packet* packet) {

uint8_t keycode = packet->keycode[0];
return keycode == KEY_RESET;
}

7.3.7 usbkeyboard.h

#ifndef _USBKEYBOARD_H
#define _USBKEYBOARD_H
#include <libusb-1.0/libusb.h>
#define USB_HID_KEYBOARD_PROTOCOL 1

/* Modifier bits */
#define USB_LCTRL (1 << 0)
#define USB_LSHIFT (1 << 1)
#define USB_LALT (1 << 2)
#define USB_LGUI (1 << 3)
#define USB_RCTRL (1 << 4)
#define USB_RSHIFT (1 << 5)
#define USB_RALT (1 << 6)
#define USB_RGUI (1 << 7)

/* Keyboard layout for the Chip8:
   +---------+
   | 1 2 3 C |
   | 4 5 6 D |
   | 7 8 9 E |
   | A 0 B F |
   +---------+
   In this program mapped to a qwerty keyboard:
   +---------+
   | 1 2 3 4 |
   | Q W E R |
   | A S D F |
   +---------+
Relying on the ascii mapping defined by the usb standard

```
#define KEY1 0x1E
#define KEY2 0x1F
#define KEY3 0x20
#define KEYC 0x21
#define KEY4 0x14
#define KEY5 0x1A
#define KEY6 0x08
#define KEY7 0x04
#define KEY8 0x16
#define KEY9 0x07
#define KEYE 0x09
#define KEYA 0x1d
#define KEY0 0x1b
#define KEYB 0x06
#define KEYF 0x19
```

```c
struct usb_keyboard_packet {
    uint8_t modifiers;
    uint8_t reserved;
    uint8_t keycode[6];
};
```
/* Find and open a USB keyboard device. Argument should
    point to space to store an endpoint address. Returns NULL if no
    keyboard device was found. */
extern struct libusb_device_handle *openkeyboard(uint8_t *);
int kbiskeypad(struct usb_keyboard_packet* packet, char val[1]);
int kbisstart(struct usb_keyboard_packet* packet);
int kbispause(struct usb_keyboard_packet* packet);
int kbisreset(struct usb_keyboard_packet* packet);
#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 4cb0402e07bc27139bb2ec94ed30b98be97cab5
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>
Fixed enums

commit 1778fa04cc1ef9e6eb8309aca2de91de227fa9c7
Merge: 1647c31 1b9a333
Author: David Watkins <djw2146@columbia.edu>
Date: Tue May 10 20:36:59 2016 -0400


commit 1647c3141e4ed1c7f44e8e7af2a0f07106b7bd5f
Author: David Watkins <djw2146@columbia.edu>
Date: Tue May 10 20:36:37 2016 -0400

Added new changes to software

commit 1b9a333ffde12c77340a763c1c69e3d6635252b
Merge: 6be3433 9aafb45
Author: lpo1234 <lpo1017@frontiernet.net>
Date: Tue May 10 20:33:28 2016 -0400

Revamped CPU and Top after memory file findings. Pushing representative stack testbench.

commit 6be34339a828d7f32c80dc684393dbceac509f50
Author: lpo1234 <lpo1017@frontiernet.net>
Date: Tue May 10 20:27:00 2016 -0400

Revamped CPU and some Top to fix with memory module. Adding representative stack testbench.

commit 9aafbb4524c6b3e45f6acfe553fcd08cf8440b4c3
Author: David Watkins <djw2146@columbia.edu>
Date: Tue May 10 18:51:22 2016 -0400

Added missing enums
commit b71d4db1b9469cad2e763d550fc0f46e0c6f8c8a
Author: David Watkins <djw2146@columbia.edu>
Date: Tue May 10 15:59:10 2016 -0400

Added changed stuff

commit cc8bc08af9a933ca683d01b0b0b2fb45495490fa
Merge: e9f7318 e1cd8d1
Author: lpo1234 <lpo1017@frontiernet.net>
Date: Mon May 9 20:19:39 2016 -0400

Merge branch 'master' of

commit e9f73189e4d6b106bb55e6d1d9112f23d9938ed0
Author: lpo1234 <lpo1017@frontiernet.net>
Date: Mon May 9 20:19:20 2016 -0400

Possible draw-sprite fix.

commit e1cd8d18d6439d85845f1f281d040a3163f5ccbd
Author: Ashley Kling <ask2203@columbia.edu>
Date: Mon May 9 18:43:23 2016 -0400

added tests for 1 and 2, expanded a CPU instruction to
  accommodate stack

commit 67d9cb4db4bf466ad8e0c16fa2de5d1fcfaae741
Merge: d9eed9c 1bff01e
Author: Ashley Kling <ask2203@columbia.edu>
Date: Mon May 9 18:40:22 2016 -0400

Merge branch 'master' of

commit 1bff01e901e62f68ec67f276c111c94e0b75a832
Author: lpo1234 <lpo1017@frontiernet.net>
Date: Mon May 9 18:39:38 2016 -0400

Updated big CPU testbench

commit 2b8da46248698d2952be6ae234e2613a03dccc1e
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 d9eed9ce7a3e3db22a66af52f19c133b49498317
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 fc3d05aa7a2c74c29b18bb0b21b83751dc6ea09
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>
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

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

Changed top level files

commit 2e1e52cfba28bce7b60b6c84277c0de3e4d46dd8
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

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

commit 85cd6a80b1e954573d7fa240c019834ef1b7b828
Author: lpo1017 <lpo1017@frontiernet.net>
Date:  Sun May 8 00:54:43 2016 -0400

Changed how draw-sprite-command Dxy works. Untested.

commit 22030b63e776204af143883c4f288e194c65b43d
Author: David Watkins <djw2146@columbia.edu>
Date:  Sun May 8 00:09:25 2016 -0400

Added proper key reading

commit fe2f4e96b21c6db9acc523832381fa6d227c44c2
Merge: 85e2d54 d63d5cb
Author: David Watkins <djw2146@columbia.edu>
Date:  Sat May 7 21:49:22 2016 -0400


commit 85e2d54f55d3a089d181bc78796ff2ba45c787f1
Author: David Watkins <djw2146@columbia.edu>
Date:  Sat May 7 21:49:02 2016 -0400

Added reading and writing proper functionality

commit d63d5cb3591f7d481d0920abe76425f3113bc6db
Author: Ashley Kling <ask2203@columbia.edu>
Date:  Sat May 7 19:43:33 2016 -0400

basic framebuffer testbench w/rudimentary reset

commit e676f44becd5e6dd449422d014e027b4f064f495
Author: David Watkins <djw2146@columbia.edu>
Date:  Sat May 7 00:29:51 2016 -0400

Added change to ispressed
commit 25466202fab00ebc180bc529d0163dea458b7b95
Author: lpo1234 <lpo1017@frontiernet.net>
Date: Sat May 7 00:27:30 2016 -0400

Properly addressed enums.svh.

commit 6e7ee698ebe3ef5588029f59c26d14b95288ba3a
Author: lpo1234 <lpo1017@frontiernet.net>
Date: Sat May 7 00:15:20 2016 -0400

Forgot to push CPU.

commit cc7eb43fac64d27a259030851aa15ae4e854f10
Merge: e7e44fb ad62fb7
Author: lpo1234 <lpo1017@frontiernet.net>
Date: Sat May 7 00:13:11 2016 -0400

Merge branch 'master' of

commit e7e44fb5c1a6e543d3bce3a76a5e4a724b3d13b9
Author: lpo1234 <lpo1017@frontiernet.net>
Date: Sat May 7 00:12:43 2016 -0400

Added stack in. Updated top-level to reflect I/O changes.
  Prepping for compile.

commit ad62fb76fe236d96018b981edbd80b1485dc98e
Author: Ashley Kling <ask2203@columbia.edu>
Date: Fri May 6 22:43:38 2016 -0400

moved some declarations so it works

commit 965d008c4f4b735c9c4c44989aca249a425112bd
Author: Ashley Kling <ask2203@columbia.edu>
Date: Fri May 6 22:20:59 2016 -0400
updated stack stuff

commit 6ef322275e702af5f30edd9aace10c5e494838f
Author: gabriellet <gat2118@columbia.edu>
Date: Fri May 6 21:00:32 2016 -0400

stack testbench updated, stack passes tests

commit 858effa51dc45c746a19a737d797821f0c020183
Merge: ec6a883 960f694
Author: gabriellet <gat2118@columbia.edu>
Date: Fri May 6 20:04:56 2016 -0400


commit ec6a8838916b4785743fde5b0cd9191ed1a49f27
Author: gabriellet <gat2118@columbia.edu>
Date: Fri May 6 20:04:28 2016 -0400

Stack testbench update

commit 960f6947b523d1a2c36a6aef02f214852aa27076
Author: David Watkins <djw2146@columbia.edu>
Date: Fri May 6 19:39:13 2016 -0400

Added changes to makefile

commit 94932c1591f2e6bd3f5560a5600adb5e254749b6
Author: David Watkins <djw2146@columbia.edu>
Date: Fri May 6 14:21:14 2016 -0400

Added changes to chip8_top

commit cdcf9ff9fdecfc322e74b0b71f7db80ea6536eb
Author: David Watkins <djw2146@columbia.edu>
Date: Fri May 6 14:17:36 2016 -0400
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


commit c5845b6ca4bc6a12674a44a4f5ffbb9fe53e560a
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 83bd4ce83c55a21172903bd2c6ead0f6e4077a
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

commit 53d84f54d5b3051834b034ad32f0cf4f930347d7
Merge: 22fe90c ed8e36c
Author: Ashley Kling <ask2203@columbia.edu>
Date: Tue May 3 15:26:36 2016 -0400


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


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

252
commit 87816742da290e4f79ade8934a370eed163c8f3d
Merge: 5a914f3 36e18d2
Author: gabriellet <gat2118@columbia.edu>
Date: Tue May 3 13:37:37 2016 -0400

Merge branch 'master' of

commit 5a914f300ad9a6a9580cf4ad60c51deba4863ffd
Author: gabriellet <gat2118@columbia.edu>
Date: Tue May 3 13:37:15 2016 -0400

updated tested ALU and enum

commit 35dfb697a5c5c9c4f7c483846fc381ade5bf5f0
Merge: 4165b3d 36e18d2
Author: David Watkins <djw2146@columbia.edu>
Date: Tue May 3 13:33:03 2016 -0400

Merge branch 'master' of

commit 36e18d283f432890f757638157050543925e3419
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date: Tue May 3 13:26:56 2016 -0400

Update .gitignore

commit 1c7e5013d058406d0db3dc80bb155ca6892206ee
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date: Tue May 3 13:22:56 2016 -0400

Fixed carry in ALU_f_ADD

commit 95b7b78a1768822c00fd4a3ef263b7bafde03535
Author: gabriellet <gat2118@columbia.edu>
Date: Tue May 3 13:11:33 2016 -0400
final ALU files - confirmed working

commit 8275d351c7b51571213c949e39a756ebf58a140f
Author: gabriellet <gat2118@columbia.edu>
Date: Sat Apr 30 23:34:40 2016 -0400

Stack testbench intermediate files:

commit 4c40781ff01ebe970ed4869a15cacf2832de7037
Author: gabriellet <gat2118@columbia.edu>
Date: Sat Apr 30 23:32:19 2016 -0400

ALU testbench files updated

commit 7ed35963899dce144ed4292b45bb16eed78d9ce0
Author: gabriellet <gat2118@columbia.edu>
Date: Sat Apr 30 17:21:20 2016 -0400

additional ALU test files

commit c6faa7efd3dc2d29ffaee21853acd405822b76f7
Author: gabriellet <gat2118@columbia.edu>
Date: Sat Apr 30 17:19:56 2016 -0400

ALU testing underway, still buggy

commit 2eab7a62d7b455e38b324c723b461ec29e0c6d53
Author: gabriellet <gat2118@columbia.edu>
Date: Thu Apr 28 16:06:36 2016 -0400

new testbench files

commit e4f8c1ef00d2144481f1f63f74faa7a4c06e643f
Author: gabriellet <gat2118@columbia.edu>
Date: Thu Apr 28 15:49:32 2016 -0400

moved audio files in test to test/audio
commit 529cdcfb17e518f53d2c7b88caaffa99ce6b3b1e6
Author: gabriellet <gat2118@columbia.edu>
Date: Thu Apr 28 15:45:20 2016 -0400

push stack ram to test/stk

commit e19c5f02d734e5572249e492e79890ea1f8c978c
Author: gabriellet <gat2118@columbia.edu>
Date: Thu Apr 28 13:32:43 2016 -0400

stack testbench update

commit e5998b9b50fadeecaf8b85a5abd9697b598b21bb2
Merge: 9a46db3 0b17dc7
Author: gabriellet <gat2118@columbia.edu>
Date: Thu Apr 28 13:14:19 2016 -0400

Merge branch 'master' of
Stack updated, merging to testbench

commit 9a46db322947f85a19d149a3fc9fdcf0dbaaa6
Author: gabriellet <gat2118@columbia.edu>
Date: Thu Apr 28 13:13:56 2016 -0400

stack_testbench update

commit 0b17dc7a2c1ce76f303790bab864614746189dba
Merge: 3ac5cff 6ddb751
Author: Ashley Kling <ask2203@columbia.edu>
Date: Thu Apr 28 13:04:17 2016 -0400

Merge branch 'master' of

commit 3ac5cfff9b0ca446599d195381851d777b5cf4f
Author: Ashley Kling <ask2203@columbia.edu>
Date: Thu Apr 28 13:04:04 2016 -0400

updated stack

commit 6ddb751b3ba98f4cf5a06c6f36ae5cccd3704771a
Author: gabriellet <gat2118@columbia.edu>
Date: Thu Apr 28 12:51:14 2016 -0400

Stack testbench files

commit 889fbe39b2d2a638312fe15fbd16b5a94c2414ef
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
commit 5e6485faef80c816c1ad30c4b84635552b2e4b04
Author: David Watkins <djw2146@columbia.edu>
Date: Tue Apr 26 08:07:17 2016 -0400

Updated code to support instructions at top level

commit dfd8770b0b2b77dca4b54fcedbc0f806b6e0ac87
Author: David Watkins <djw2146@columbia.edu>
Date: Tue Apr 26 06:57:58 2016 -0400

Compiling version of entire project

commit 932855dd20090b7ad68be05b9c9009ae2d762ec2
Author: David Watkins <djw2146@columbia.edu>
Date: Tue Apr 26 04:14:34 2016 -0400

Added updates to the sound controller

commit aa6f9247e0d2766b790dfafbb3c8520e32ca3081b
Author: David Watkins <djw2146@columbia.edu>
Date: Tue Apr 26 03:46:59 2016 -0400

Added header files

commit 7f698e00b0b09c2348541596c9b71568eb5bf98e
Author: David Watkins <djw2146@columbia.edu>
Date: Tue Apr 26 03:38:25 2016 -0400

Fixed ALU and CPU to support all instruction (Draw NYI)

commit 2a1286c317fd71e37bfe17bb83903c56bb340d27
Author: lpo1234 <lpo1017@frontiernet.net>
Date: Mon Apr 25 01:34:59 2016 -0400

Added most PC functionality to the CPU.

commit 0ddddefb9e4ef23b12aa804610d08ce9819a9ddd
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 b58aa9ef625b711d4180fecd31c5ce193eccb036
Merge: 9cbc6d7 3de4899
Author: lpo1234 <lpo1017@frontiernet.net>
Date: Sun Apr 24 23:40:36 2016 -0400

Merge branch 'master' of

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
commit 34e241302a4ef0977bbdbe2e5394f0dc8f6556b6
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date: Fri Apr 22 18:21:08 2016 -0400

Duplicated files deleted

commit f0a1dc7c772ed001e2073d755dd7f1d67ce37547
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date: Fri Apr 22 18:20:58 2016 -0400

Duplicated files deleted

commit d56f6ab9c26e0d9fd880f727d6be6fbbec44be6e
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date: Fri Apr 22 18:20:47 2016 -0400

Duplicated files deleted

commit 92890f725928acce4d7755c0784d69cca85062ac
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date: Fri Apr 22 18:20:34 2016 -0400

Duplicated files deleted

commit be03d04f33b42f09ee3dfb45c2b5cd0cf547df4e
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date: Fri Apr 22 18:17:14 2016 -0400

Script + samples + original wav

Script extracts samples from wav file. Samples for beep2.wav are in samples.txt

commit b705539fb6021353ac2ddb476142288cf3f267a1
Author: David Watkins <djw2146@columbia.edu>
Date: Tue Apr 19 15:51:50 2016 -0400
Added files that were missing from before

commit 512823e2a46d09a3d8305699f60615b0a0988908
Author: David Watkins <djw2146@columbia.edu>
Date: Sat Apr 16 18:11:21 2016 -0400

Moved testbench for delay timer

commit 844880913d18dee6e4c4cee38dd8a12c012a413f
Author: David Watkins <djw2146@columbia.edu>
Date: Sat Apr 16 18:08:35 2016 -0400

Redid file directory

commit f55eb911738c27765f9c0a8f8158ba6393cd541b
Author: gabriellet <gat2118@columbia.edu>
Date: Sat Apr 16 07:24:54 2016 -0400

removed mips directory from test, added delay timer and testbench to test

commit ef7706c9c01f65935ec1f8d8ba75900d9821b824
Merge: 6e306a2 fcb610e
Author: gabriellet <gat2118@columbia.edu>
Date: Fri Apr 15 12:22:42 2016 -0400


commit 6e306a29a061cb4612c4e1c4c6160b731e7ec84a
Author: gabriellet <gat2118@columbia.edu>
Date: Fri Apr 15 12:21:58 2016 -0400

delay timer implementation

commit fcb610e9bf6b6face733099142e3a1d275106f871
Author: lpo1234 <lpo1017@frontiernet.net>
Date: Wed Apr 13 23:50:14 2016 -0400
Here is the test waveform to show on Thursday Apr 14. I am the prettiest.

commit fa449ed041a88e713c1f133beb3033e2a5f48021
Author: David Watkins <djw2146@columbia.edu>
Date:   Wed Apr 13 23:02:56 2016 -0400

Forgot test

commit 050daf1090a867f8181af4f1db6bbaad9c6054f6
Merge: c37a15a ac284ac
Author: David Watkins <djw2146@columbia.edu>
Date:   Wed Apr 13 22:46:39 2016 -0400


commit c37a15a2e2f3ffdd0e0ccdb502336571f20f50000
Author: David Watkins <djw2146@columbia.edu>
Date:   Wed Apr 13 22:46:23 2016 -0400

Cries

commit ac284ac58a27087d476eb4ccd0061ee9e555917
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Wed Apr 13 22:40:15 2016 -0400

Added dual ported memory for register file and memory.

commit 07694cbe2ac21b44c806dc56c466470b44dfdc02
Author: David Watkins <djw2146@columbia.edu>
Date:   Wed Apr 13 22:30:14 2016 -0400

Added testbench

commit ade90c0e38478bbee1cccd93f255fce8834adf321
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: 77d791d1 1443ecb
Author: David Watkins <djw2146@columbia.edu>
Date: Wed Apr 13 21:31:27 2016 -0400


commit 77d791dc6a4f63a5a0ea4fedc13c11f1d2d1700
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 4ffe66bcafed6e9a54b3c44b7111ea0428327a8f
Author: David Watkins <DavidWatkins@users.noreply.github.com>
Date: Wed Apr 13 19:02:48 2016 -0400

Delete SoCKit_Top.ipinfo

commit dbbdc521e65452464f9d20f66fe9b73828dac53a
Delete SoCKit_Top.sld_design_entry.sci

commit 4c5ce1ca7dd6541308ef7f923505f80b075035f
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

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

Merge branch 'master' of

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 9761ee95f0496f077c326bee4aba1825aeed73f
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

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 75aa22faed1b3fabc1f3439d8df2defced46e95
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date:  Mon Apr 11 12:28:07 2016 -0400

bcd.sv now works
Tests indicate module works.

commit a744ba2ec0ca570e06cecc2770aedd176d044d2d
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date:   Sun Apr 10 13:30:36 2016 -0400

Fix compilation errors
Simulated in modelsim but still does not work. Loop in line 13 only runs once. Will investigate why.

commit fd6b8e93cce11ec12061d7a328f912fbe85f1cad
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Sun Apr 10 03:32:07 2016 -0400

Adding memory module waveform test.

commit 4b5e6b9b12c7e6706af463a5324b2b65ba1626cb
Author: lpo1234 <lpo1017@frontiernet.net>
Date:   Sun Apr 10 03:18:46 2016 -0400

Added 4096byte memory module.

commit 90fd2f67f770b0eb25a8930198fbd8b05b8afa88
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date:   Sat Apr 9 22:52:47 2016 -0400

BCD module in systemverilog

commit 6de8348fd274c49e7564113f6f631f53b4cbfc32
Author: Gabrielle A Taylor <gat2118@columbia.edu>
Date:   Sat Apr 9 22:24:38 2016 -0400

Information relevant to audio test

commit df18f8e1c60ffe03b53e1bf4117c52852969b2b
Author: Gabrielle A Taylor <gat2118@columbia.edu>
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 e3410e4fb4c763867b1b6e503751ed155e06830c
Author: lpo1234 <lpo1017@frontiernet.net>
Date: Thu Mar 31 00:07:47 2016 -0400

Added ALU and memory-to-screen VGA emulator

commit 9b629ea8e3d0ddee730a90f39806208222fde5965
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


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
7.5.3 timer.sv
7.5.4 Chip8_rand_num_generator.sv
7.5.5  bcd.sv

7.5.6  memory.sv
7.5.7 Chip8_SoundController.sv
7.5.8 Entire Design