CSEE 4840 Embedded Systems: CHIP8 Hardware Emulator

Professor Steven Edwards

Mar 30, 2022

Elysia Witham(ew2632)
Daniel Indictor(di2215)
Yuhang Zhu(yz4136)
Xin Gao(xg2376)

Table of Contents

1. Introduction

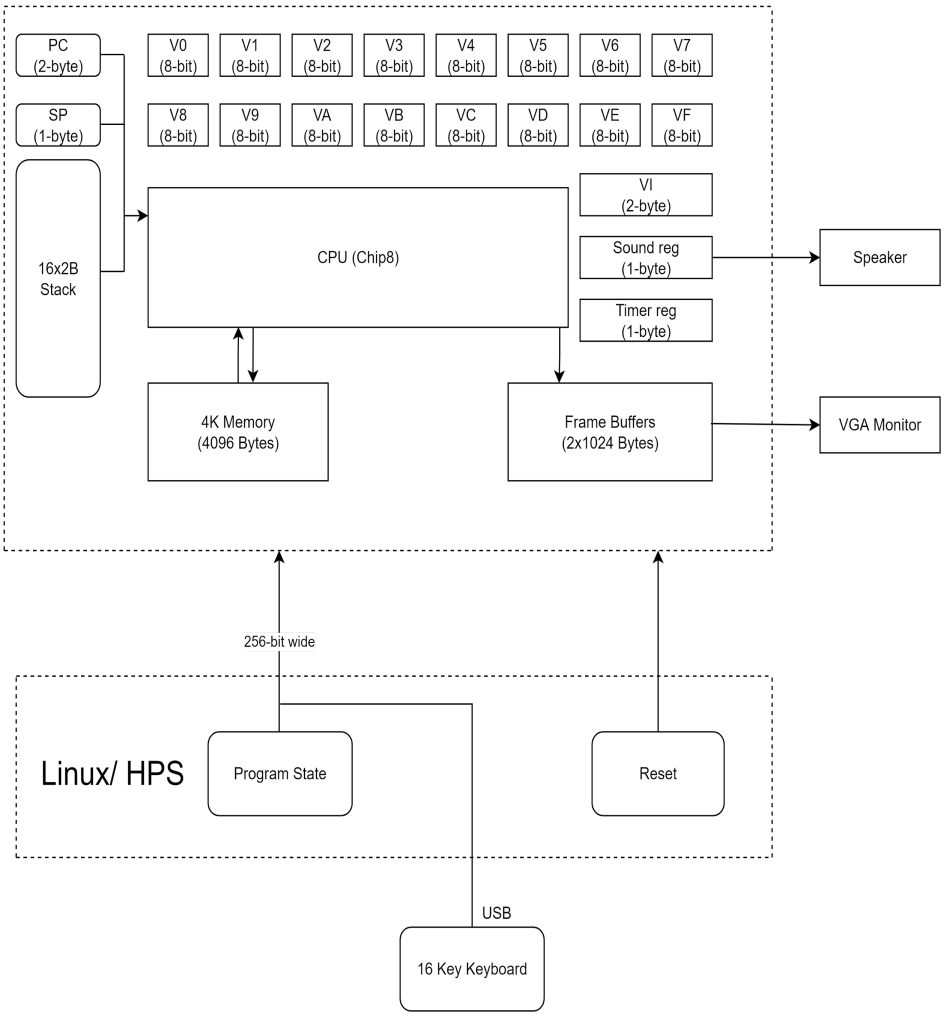
Our goal is to create a CHIP8 emulator. CHIP8 is a super simple interpreted programming language that runs in a virtual machine which provides it access to virtualized peripherals such as a keyboard, low resolution display, and buzzer. Many toy programs and games have already been written in this language, which supports a low-resolution display, a hexadecimal number pad, and a rudimentary sound output.

Because creating a CHIP8 emulator in software is almost trivial, we opted to make a hardware emulator instead using SystemVerilog. The entire CHIP8 system will be recreated in hardware on our DE1-SoC FPGA. As for software, we will be creating a game selection menu with a few pre-selected games which will be shown on the VGA display. Keyboard presses will be captured by software, with the appropriate data being communicated to hardware. Once a game is selected via the keyboard, the hardware emulator will take over and a user will be able to play the selected game using the keyboard.

2. Block Diagram

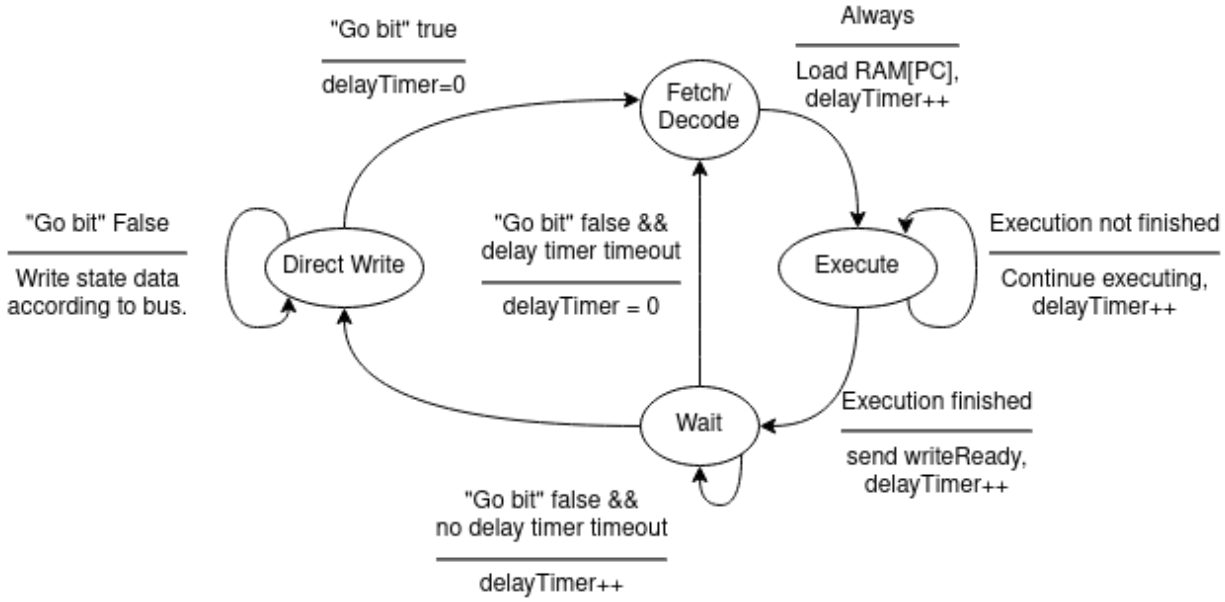
The design of the CHIP8 will have five major parts, which includes the 4K memory, the framebuffer, the CPU and controller, the speaker, and the VGA monitor.

The Framebuffer module will be responsible for providing a buffer for both the CHIP8 program and the Linux machine, either of which can write to the “working” framebuffer. Either may also request a “display update”, or that the working buffer is copied to the displayed buffer. It will additionally scale and center the 128x64-bit image to fit on a normal-sized monitor.



3. Algorithms

Due to the complexity of certain CHIP8 instructions, a single cycle computer is infeasible. For instance, while a load instruction may be executed in a single clock cycle, a sprite-rendering instruction may take upwards of 100 clock cycles. To accommodate this design, we instead use a fetch-execute-wait cycle, where the “execute” phase may take anywhere between a single and a hundred clock cycles.



Since the FPGA can execute programs far faster than the original CHIP-8 programs were meant to be run, we have to artificially slow down our program using a delay timer. This timer starts counting every time an instruction is fetched and counts up to some constant, “Delay Timer Max”, preventing the execution of the next instruction until it finishes. The value of this maximum depends on the FPGA clock speed and the target instruction clock. For instance, if the FPGA is running at 50MHz and we’re targeting 700 instructions per second, we would set our Delay Timer Max to about 71428.

We are using a double buffering method to display graphics, which means we will not be editing the frame buffer that is being shown in real time. We instead edit a secondary frame buffer which is the same size, and then copy it to the displayed buffer in its entirety, rather than one pixel at a time. This helps prevent flickering, and also reduces the amount of writes through the VGA.

4. Hardware-Software Interface

As per CHIP8 spec, all multi-byte values are MSB-first.

We will be using a bus size of 256 bits, as this is the largest port size available in Quartus's megafunction wizard for RAM. Thus, we can write to the memory and frame buffer 256 bits (or 32 bytes) at a time.

Bus Address	End	(hex)	Start	(hex)	Size (bytes)
128x32B (4K) Memory	127	(7F)	0	(0)	4096
Framebuffer	159	(9F)	128	(80)	1024
16x2B Stack	160	(A0)	160	(A0)	32
Keyboard State	161	(A1)	161	(A1)	32
Control Data	162	(A2)	162	(A2)	32

We dedicate a special address to smaller state and control data.

Misc Address	End	(hex)	Start	(hex)	Size (bytes)
V0-VF	15	(F)	0	(0)	16
Delay Timer Max	19	(13)	16	(10)	4
Reg I	21	(15)	20	(14)	2
Reg PC	23	(17)	22	(16)	2
Sound	24	(18)	24	(18)	1
Timer	25	(19)	25	(19)	1
Stack Pointer	26	(1A)	26	(1A)	1
Unused	30	(1E)	27	(1B)	4
Control Byte	31	(1F)	31	(1F)	1

The "Control Byte" has a "Go bit", which instructs the FPGA to begin or continue its work. Additionally, the IsWrite bit is checked to make sure that the HPS wants to write the other data in the Misc Address. The DiplayUpdate bit is set to request that the framebuffer copy the working buffer to the displayed buffer.

Control Byte Bits	End		Start
Go	1		1
IsWrite	2		2
DisplayEnabled	3		3
DisplayUpdate	4		4

5. Resources Requirements

The CHIP8 specification requires the use of sixteen 1-byte registers (V0-VF), a 2-byte index register, a 32-byte stack with 1-byte stack pointer, an 1-byte timer register, an 1-byte sound register, a 8192-bit display (in our case we will need twice that for the framebuffer), and a 2-byte 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 registers are supposed to produce a triangle wave 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 framebuffers are (x, y) addressable memory arrays which designate whether a pixel is currently on or off. We will be using a double-buffering scheme as described above.
• 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 needs to be drawn on the screen.
• The stack will be a 32-byte memory block that only reads out the current value stored in the stack pointer, which can be either increasing or decreasing by the current instruction being evaluated.
• The keyboard presses will be synchronized written to the control unit from the ARM processor. That enables the control units to properly handle varying input without the potential affect to resulting code.

Category	Size(bits)	# of items	Total size(bytes)
registers(V0-VF)	8-bit	16	16
Index register(VI)	16-bit	1	2
stack	128	1	16
Stack Pointer(SP)	8	1	1
Timer register	8-bit	1	1
Sound register	8-bit	1	1
Frame buffers	2x8192 bit	2	2048

Program Counter	16-bit	1	2

6. Timeline

- Milestone I (17th April): Proof-of-concepts. By this milestone, we will have created a demo showing how each hardware and software component interacts with the outside world.
    - Proof-of-concept demo of communication from the FPGA to the HPS
    - Keyboard interface code finished (libusb)
    - Finish and test software prototype.
    - Write and test framebuffer module.
    - Make speakers work
- Milestone II (1st May): Integration Hell. By this milestone, we will have written interfaces and a dummy CHIP8-emulator to control them.
    - Completion of program-selection user interface.
    - All the SystemVerilog module interfaces should be finalized at this point, and all the peripherals implemented. All Linux code for interacting with the FPGA should be completed.
- Milestone III (7th May): Integration of completed CHIP8 emulator.
    - Implementing the integration of hardware and software into CHIP8. Verify its functionality and fix bugs.
    - Choose a handful of CHIP8 games to load into the emulator and tune them (such as timer delay, for example)

7. Reference

Cowgod's Chip-8 Technical Reference. [Online].

Available:http://devernay.free.fr/hacks/chip8/C8TECH10.HTM#0.0.

JohnEarnest, “JohnEarnest/Octo: A Chip8 IDE,” *GitHub*. [Online]. Available:

https://github.com/JohnEarnest/Octo/.

Pwmarcz, “pwmarcz/fpga-chip8: CHIP-8 console on FPGA,” *GitHub*. [Online]. Available:

https://github.com/pwmarcz/fpga-chip8.