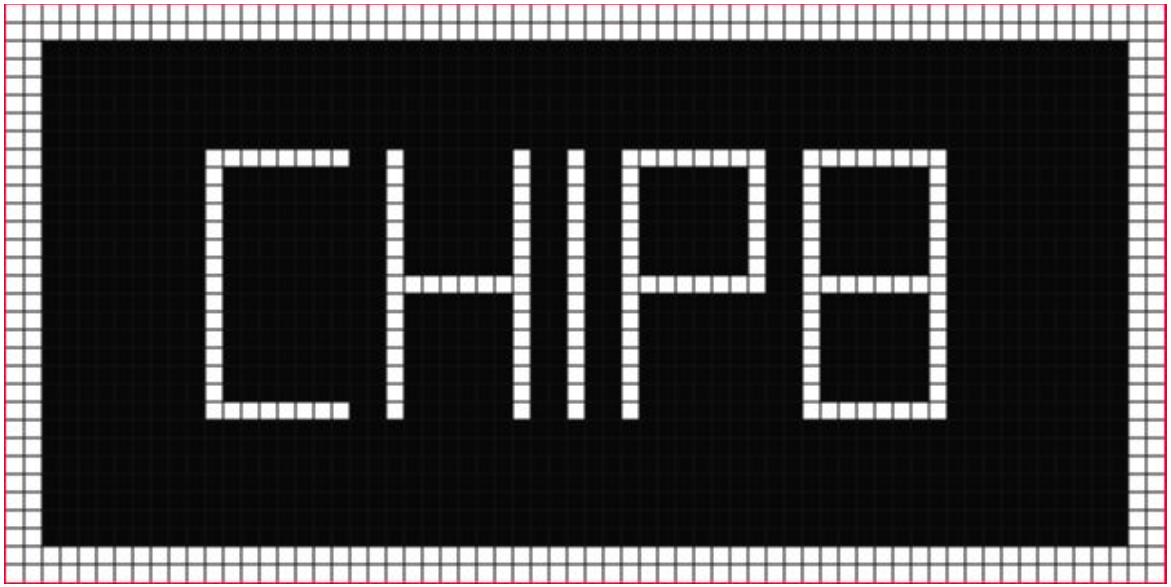
Chip-8 Emulator

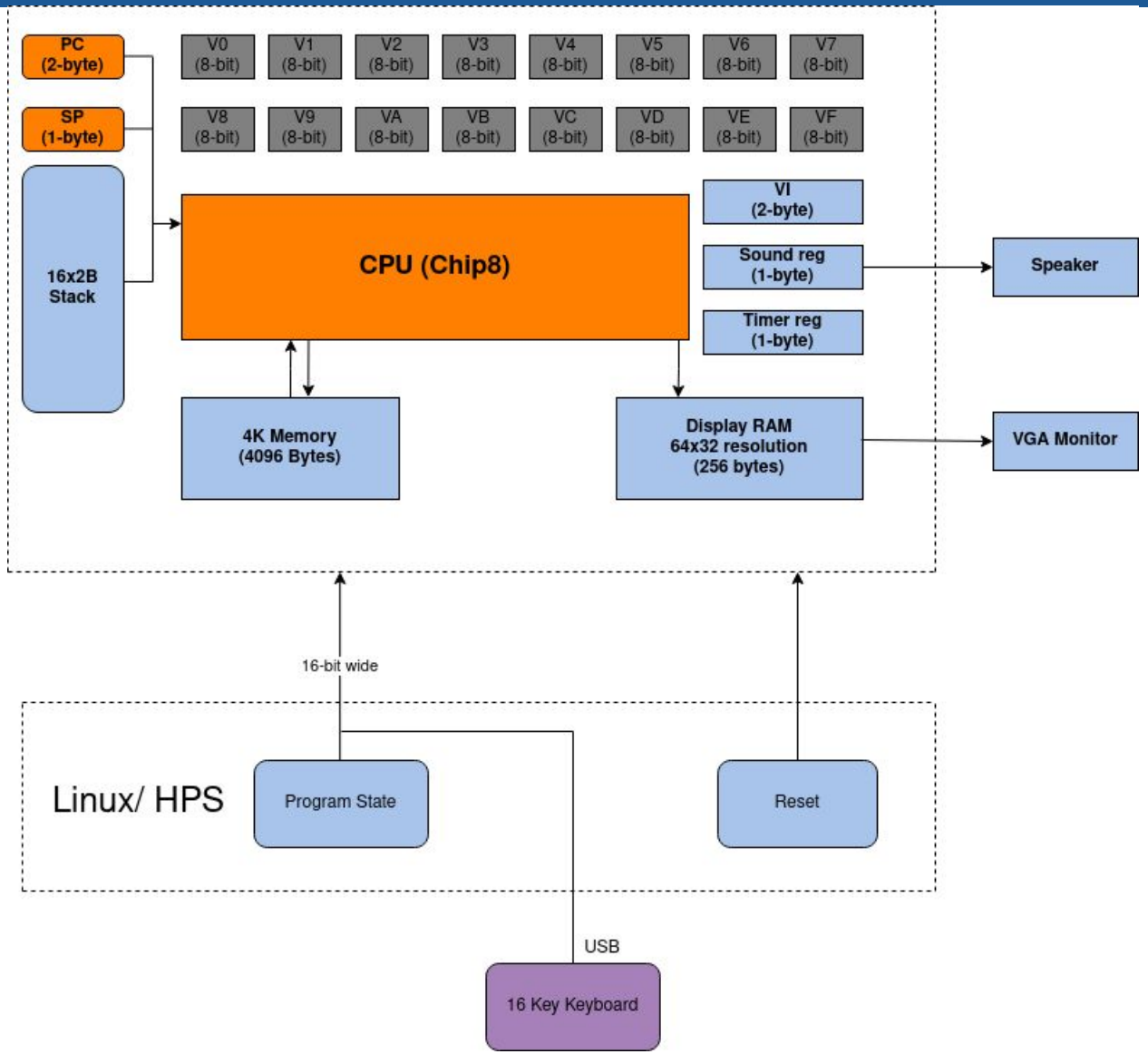
Team:

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

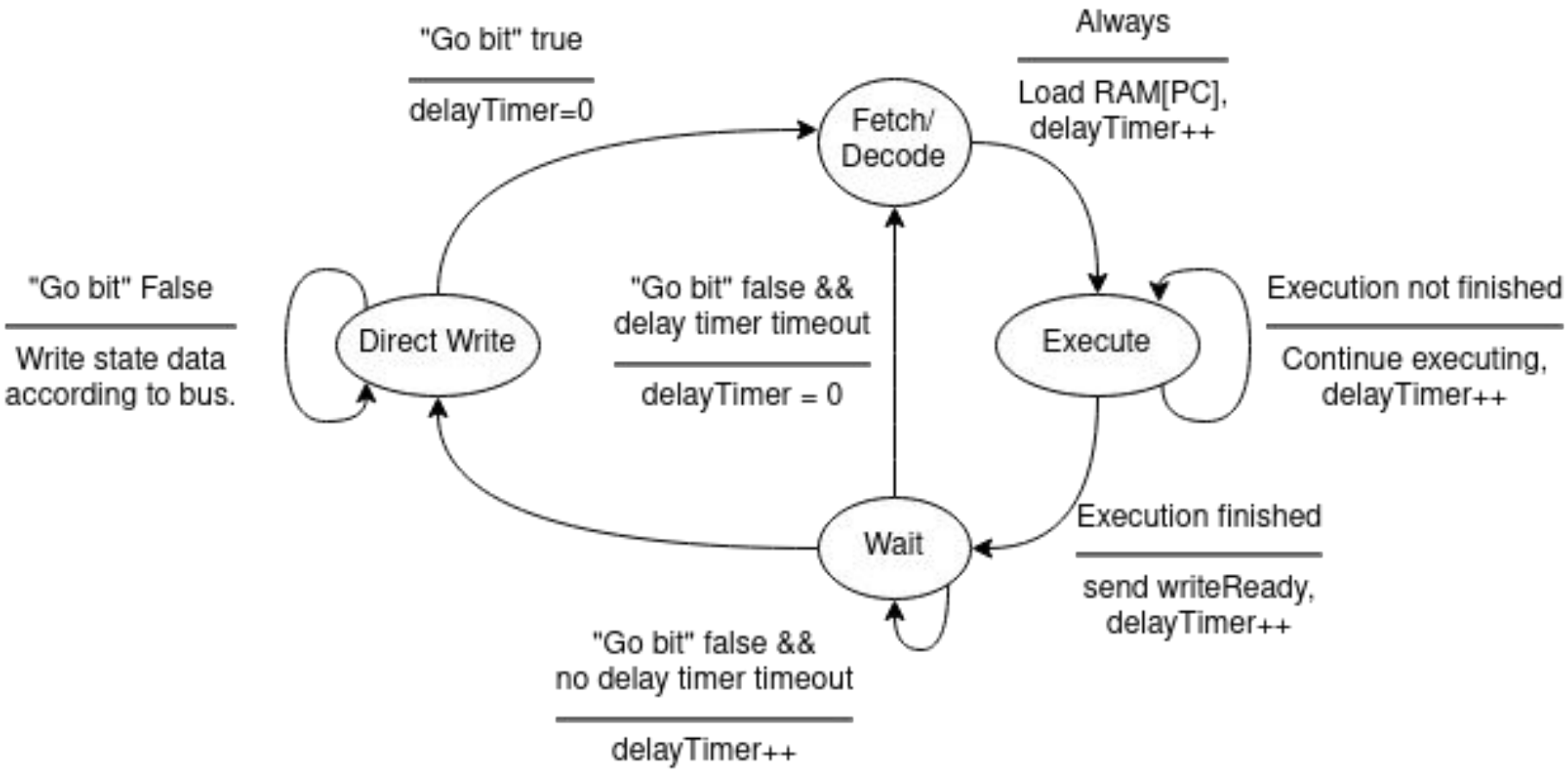
COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK


CHIP8

Block Diagram



FSM Diagram


"Go bit" true
delayTimer=0
Always
Load RAM[PC],
delayTimer++
Fetch/
Decode
"Go bit" False
Write state data
according to bus.
Direct Write
"Go bit" false &&
delay timer timeout
delayTimer = 0
Execute
Execution not finished
Continue executing,
delayTimer++
Execution finished
send writeReady,
delayTimer++
Wait
"Go bit" false &&
no delay timer timeout
delayTimer++

Code Layout

Chip-8

- **hw/**
  - Contains SystemVerilog modules (next slide)
  - Contains module-level unit tests
- **sw/**
  - Contains kernel driver (chip8.c)
  - Contains user-space application
- **swproto/**
  - Contains software prototype
  - Contains unit tests

Chip-8 Hardware

- **display_ram**: A simple 256-byte two-port read-write RAM to be read by the display module and written by either the software driver or the CPU module.
- **chip_ram**: A 4096-byte two-port read-write RAM to be read by the CPU and written by the CPU or the software driver.
- **display**: This module does the job of reading data from the display ram and scaling/fitting the 64x32 pixel display native to CHIP8 to a 640x480 pixel VGA display. It thus also handles setting VGA protocol signals.
- **cpu**: This module is incomplete (as you shall see) but in theory it should house the finite state machine given earlier and a huge switch statement to handle the opcodes.
- **sound**: This module is complete (but not integrated). In theory, it should just make the FPGA make a sound on its AUX port given the signal to do so.
- **yachip**: This is the top-level module (completed but not tested) that delegates access to the display and chip8 rams to the CPU and display module. As discussed, it gives priority to the software driver.
- **timer**: Timers are necessary in three parts in our code: first, for two special purpose registers which, when set to a non-zero value tick down to 0 at 60Hz, and another to slow down the CPU.

VGA Display

Ram module
Chip8 only had 64x32 pixel display
Interface with CPU.

```
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00011111001110010010001111110000,
00000011110110111010011000010000,
00011111011110111010101001100000,
00000011000111101111001110000000,
00011111000111000110001011110000,
00000000000000000000000000000000,
00000000000000011000000000000000,
00000000000011111111100000000000,
00000000000011111111110000000000,
00000000000011111111100000000000,
00000000000001111100000000000000,
00000000000011111111000111100000,
00000000000011111110111101000000,
00000000000111111111111011100000,
00000000000111111111110111110000,
00000000001111111111110011100000,
00000000001111111111010001000000,
00000000000111111111010000000000,
00000000000011111111000000000000,
00000000000011111111000000000000,
00000000000011111110000000000000,
00000000000011111110000000000000,
00000000000011111110000000000000,
00000000000011111110000000000000,
00000000000011111110000000000000,
00000000001111111111100000000000,
```

Audio

- Generate a memory initialization file (.mif) for audio effect
- Single-port ROM memory blocks
- Use 3 IPs in Qsys:
  - Audio and Video Config
  - Audio Clock for DE-series Board
  - Audio.

Keyboard Layout

We opted to use a USB keyboard rather than a dedicated pin pad for CHIP8 inputs. Many online CHIP8 emulators support a keyboard with the same keys as the ones we are using mapped to the same inputs. The mapping is as follows:

1 2 3 4		1 2 3 C
q w e r		4 5 6 D
a s d f	-->	7 8 9 E
z x c v		A 0 B F

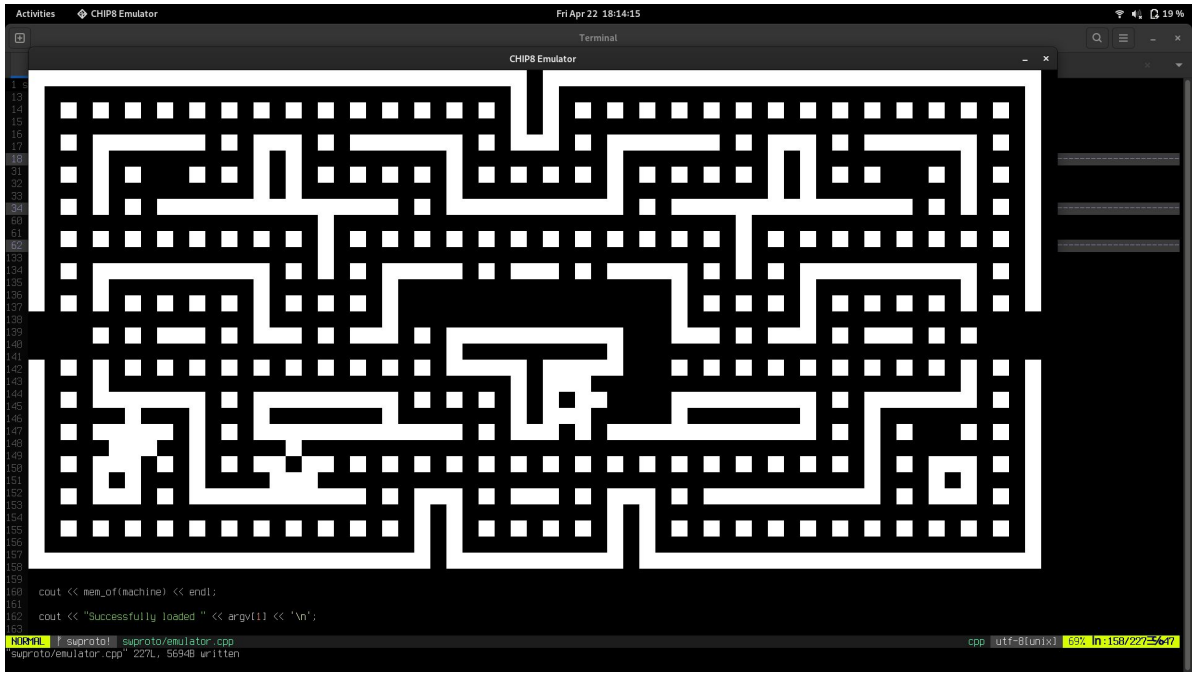
The left side represents a standard keyboard, with the right side representing what it is being translated as for the CHIP8 device.

Game Selection

We created a userspace application with a few pre-selected games which can be chosen with 5-9 number keys. Keyboard presses are 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.

Software Prototype

```
15:49 desktop daniel build$ ninja && ./swproto/statemachine_test
[4/4] Linking CXX executable swproto/statemachine_test
Running main() from /home/daniel/Documents/college/embedded_systems_cs
[==========] Running 23 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 23 tests from StateMachineTest
[ RUN      ] StateMachineTest.Test00EE_2nnn
[       OK ] StateMachineTest.Test00EE_2nnn (0 ms)
[ RUN      ] StateMachineTest.Test6xkk_7xkk
[       OK ] StateMachineTest.Test6xkk_7xkk (0 ms)
[ RUN      ] StateMachineTest.Test1nnn
[       OK ] StateMachineTest.Test1nnn (0 ms)
[ RUN      ] StateMachineTest.Test3xkk_4xkk
[       OK ] StateMachineTest.Test3xkk_4xkk (0 ms)
[ RUN      ] StateMachineTest.Test5xy0_9xy0
[       OK ] StateMachineTest.Test5xy0_9xy0 (0 ms)
[ RUN      ] StateMachineTest.Test8xy0
[       OK ] StateMachineTest.Test8xy0 (0 ms)
[ RUN      ] StateMachineTest.Test8xy1_7xy2_8xy3
[       OK ] StateMachineTest.Test8xy1_7xy2_8xy3 (0 ms)
[ RUN      ] StateMachineTest.Test8xy4
[       OK ] StateMachineTest.Test8xy4 (0 ms)
[ RUN      ] StateMachineTest.Test8xy5
[       OK ] StateMachineTest.Test8xy5 (0 ms)
[ RUN      ] StateMachineTest.Test8xy6
[       OK ] StateMachineTest.Test8xy6 (0 ms)
[ RUN      ] StateMachineTest.Test8xy7
[       OK ] StateMachineTest.Test8xy7 (0 ms)
[ RUN      ] StateMachineTest.Test8xyE
[       OK ] StateMachineTest.Test8xyE (0 ms)
[ RUN      ] StateMachineTest.TestEx9E
[       OK ] StateMachineTest.TestEx9E (0 ms)
[ RUN      ] StateMachineTest.TestExA1
[       OK ] StateMachineTest.TestExA1 (0 ms)
[ RUN      ] StateMachineTest.TestAnnn_Fx1E
/home/daniel/Documents/college/embedded_systems_cseew4840/yachip
Expected equality of these values:
  machine.reg_I()
    Which is: 4304
  0x800
    Which is: 200
[  FAILED  ] StateMachineTest.TestAnnn_Fx1E (0 ms)
[ RUN      ] StateMachineTest.TestFx0A
```



COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

Testing

chip_ram_test.cpp

```
#include "hw_test.hpp"

#include "Vchip_ram.h"

using ChipRamTest = VTest<Vchip_ram>;

TEST_F(ChipRamTest, TestSimpleWriteRead) {
  eval();
  dut.wa = 1;
  dut.wb = 1;
  // Write mem[0x01] = 0x23; mem[0x45] = 0x67;
  dut.aa = 0x01;
  dut.da = 0x23;
  dut.ab = 0x45;
  dut.db = 0x67;
  dut.wa = 1;
  dut.wb = 1;
  ticktock();
  ASSERT_EQ(dut.qa, 0x23);
  ASSERT_EQ(dut.qb, 0x67);

  // Make sure it does nothing with write enable off on port a.
  dut.wa = 0;
  dut.da = 0xff;
  // Write mem[0x89] = 0xAB;
  dut.ab = 0x89;
  dut.db = 0xAB;
  ticktock();
  ASSERT_EQ(dut.qa, 0x23);
  ASSERT_EQ(dut.qb, 0xAB);

  // Read address 0x01 with port b.
  dut.wb = 0;
  dut.ab = 0x01;
  ticktock();
  ASSERT_EQ(dut.qa, 0x23);
  ASSERT_EQ(dut.qb, 0x23);
}
```

Observations & Advice

- Decision to use C++ rather than C complicated the compilation process on the FPGA itself and our ability to use libraries that we learned about in labs, such as libusb and ioctl. We wound up having to cross compile C++ and C code at the end, which certainly increased the debugging time frame. Note to future students: getting cmake to work on Linux on the FPGA is a pain!
- Writing rigorous tests for all the modules was a good decision. Completing the modules would have been a better decision still