Title:

"Design and Implementation of a CHIP-8 Emulator using C and SDL2"

Introduction:

The CHIP-8 is a simple interpreted programming language that was initially used on COSMAC VIP and Telmac 1800 microcomputers in the mid-1970s. It is commonly used for writing games and demos due to its simplicity and ease of implementation. This report presents the design and implementation of a CHIP-8 emulator using the C programming language and the SDL2 library.

Background:

- Provide an overview of the CHIP-8 programming language, its history, and its significance in retro gaming and emulation communities.
- Discuss the motivation behind creating a CHIP-8 emulator, including its educational value and the opportunity to explore retro computing concepts.

Objectives:

- To design and implement a functional CHIP-8 emulator capable of running CHIP-8 ROMs.
- To gain hands-on experience with low-level programming concepts such as memory management, bitwise operations, and emulation techniques.
- To provide a platform for learning and experimentation with retro gaming and emulation.

Implementation Details:

- **Language and Libraries:** The emulator is implemented in the C programming language, leveraging the SDL2 library for graphics and user input handling.
- **Data Structures:** Utilizes various data structures such as enums and structs to organize CHIP-8 related data and SDL-related data.
- **Memory Management:** Manages memory for CHIP-8 RAM, display, stack, and registers.
- **Instruction Emulation:** Implements functions to emulate CHIP-8 instructions, including opcode decoding, memory access, and updating CPU state.
- **User Input Handling:** Incorporates functionality to handle user input events using SDL, allowing users to control the emulator.
- **Graphics Rendering:** Utilizes SDL's rendering capabilities to render the CHIP-8 display on a window, updating it based on the emulator's state.

Features:

- **ROM Loading:** Supports loading external CHIP-8 ROMs into the emulator's memory.
- **Instruction Emulation:** Implements core CHIP-8 instructions for CPU emulation.
- **Graphics Display:** Renders CHIP-8 display output using SDL, allowing users to visualize the execution of ROMs.
- **User Input:** Allows users to interact with the emulator using keyboard input, enabling gameplay and control.

Testing and Validation:

- Discusses the testing methodology employed to validate the correctness and functionality of the emulator.
- Includes details on unit testing individual components, integration testing the entire system, and manual testing with various CHIP-8 ROMs.

Performance Optimization:

- Analyzes potential performance bottlenecks and optimizations for improving the emulator's speed and efficiency.
- Discusses techniques such as opcode caching, parallelization, and memory management strategies to enhance performance.

Results and Evaluation:

- Provides an evaluation of the emulator's performance, compatibility, and usability.
- Discusses any limitations or challenges encountered during development and usage.
- Compares the emulator's functionality with existing CHIP-8 emulators and discusses areas for improvement.

Conclusion:

- Summarizes the achievements and contributions of the project in implementing a functional CHIP-8 emulator.
- Reflects on the learning outcomes and insights gained from the development process.
- Suggests potential future directions for enhancing the emulator, such as adding audio support, improving compatibility, or optimizing performance further.

References:

- Includes a list of references to relevant documentation, tutorials, papers, and resources used during the development of the emulator.

Appendices:

- Provides supplementary information, such as source code listings, additional diagrams, and sample CHIP-8 ROMs used for testing.

Conclusion:

This report presents the design and implementation of a CHIP-8 emulator, demonstrating the capabilities of the emulator and its potential for learning and experimentation in retro gaming and emulation. By providing a detailed analysis of the implementation process, testing methodology, and performance evaluation, this report serves as a valuable resource for enthusiasts and developers interested in retro computing and emulation.


Home / sreenithi
Save
CHIP-8 Emulator

```c
/*************************************/
/*           CHIP8 EMULATOR          */
/*                                   */
/*  AUTHOR  : SREE NITHI S V         */
/*  LICENSE : MPL                    */
/*                                   */
/*************************************/

#include <stddef.h>
#include<stdio.h>
#include<stdbool.h>
#include<SDL2/SDL.h>
#include<string.h>

#define DISCARD_UNUSED(var) (void)(var)     /* BOILERPLATE -- IGNORE */
#define println(var) printf("[INFO] %s\n", var);   // Defines a Macro which adds "\n" to the String

// Enum to Indicate the Status of the CHIP8 Emulator
typedef enum {
    QUIT,
    RUNNING,
    PAUSED
} EMU_State;

typedef struct {
    uint16_t OP;
    uint16_t NNN;
    uint8_t NN;
    uint8_t N;
    uint8_t X;
    uint8_t Y;
} CH8_Instruction;

// A Container for CHIP8 Related Data
typedef struct {
    EMU_State EM_State;
    uint8_t RAM[4096];
    bool DISPLAY[64*32];
    uint16_t STACK[12];
    uint16_t *STACK_PTR;
    uint16_t I;
    uint8_t V[16];
    uint16_t PC;
    uint8_t T_Delay;
    uint8_t T_Sound;
    bool Keypad[16];
    char *ROM_NAME;
    CH8_Instruction Inst;
} CHIP_Struct;

// A Container for SDL Related Data
typedef struct {
    SDL_Window *Window;
    SDL_Renderer *Renderer;
} SM_Struct;

// SDL Configuration
typedef struct {
```

```
    uint16_t W_Width;
    uint16_t W_Height;
    uint32_t C_Foreground;
    uint32_t C_Background;
    uint8_t Scale_Factor;
} CFG_Struct;

CFG_Struct* init_config(CFG_Struct *cfgStruct){
    if(!cfgStruct){
        SDL_Log("[ERROR] Cannot Initialize the Configuration");
        return cfgStruct;
    }

    cfgStruct->W_Height = 32;
    cfgStruct->W_Width = 64;
    cfgStruct->C_Background = 0x00000000;
    cfgStruct->C_Foreground = 0xFFFF00FF;
    cfgStruct->Scale_Factor = 20;
    return cfgStruct;

}

void handle_input(CHIP_Struct *CHIP8){
    SDL_Event Event;
    while(SDL_PollEvent(&Event)){
        switch (Event.type){
            case SDL_QUIT:
                CHIP8->EM_State = QUIT;
                return;
            case SDL_KEYDOWN:
                switch(Event.key.keysym.sym){
                    case SDLK_SPACE:
                        if(CHIP8->EM_State == RUNNING){
                            CHIP8->EM_State = PAUSED;
                            println("Emulation Paused");
                        }else{
                            CHIP8->EM_State = RUNNING;
                            println("Emulation Resumed");
                        }
                }
            case SDL_KEYUP:
                break;
            default:
                break;
        }
    }
}

void clrscrn(CFG_Struct *cfgStruct, SM_Struct *mainStruct){
    const uint8_t RED = (cfgStruct->C_Background >> 24) & 0xFF;
    const uint8_t GREEN = (cfgStruct->C_Background >> 16) & 0xFF;
    const uint8_t BLUE = (cfgStruct->C_Background >> 8) & 0xFF;
    const uint8_t ALPHA = (cfgStruct->C_Background >> 0) & 0xFF;

    SDL_SetRenderDrawColor(mainStruct->Renderer, RED, GREEN, BLUE, ALPHA);
    SDL_RenderClear(mainStruct->Renderer);
}

bool init_sdl(SM_Struct *mainStruct, CFG_Struct *cfgStruct){
    if(SDL_Init(SDL_INIT_VIDEO|SDL_INIT_AUDIO|SDL_INIT_TIMER) != 0){
```

```
119         SDL_Log("[ERROR] Initialization Failed\n[LOG] %s", SDL_GetError());
120         return SDL_FALSE;
121     }else{
122         mainStruct->Window = SDL_CreateWindow("CHIP-8 Emulator",
SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED, cfgStruct->W_Width * cfgStruct-
>Scale_Factor, cfgStruct->W_Height * cfgStruct->Scale_Factor, 0);
123 
124         if(!mainStruct->Window){
125             SDL_Log("[ERROR] Failed to create a Window\n[LOG] %s", SDL_GetError());
126             return SDL_FALSE;
127         }
128 
129         mainStruct->Renderer = SDL_CreateRenderer(mainStruct->Window, -1,
SDL_RENDERER_ACCELERATED);
130 
131         if(!mainStruct->Renderer){
132             SDL_Log("[ERROR] Failed to create a Renderer\n[LOG] %s",
SDL_GetError());
133             return SDL_FALSE;
134         }
135         return SDL_TRUE;
136     }
137 }
138 
139 bool init_chip(CHIP_Struct *chipStruct, char ROM_NAME[]){
140     const uint32_t entryPoint = 0x200;
141     const uint8_t chipFont[] = {
142         0xF0, 0x90, 0x90, 0x90, 0xF0,   // 0
143         0x20, 0x60, 0x20, 0x20, 0x70,   // 1
144         0xF0, 0x10, 0xF0, 0x80, 0xF0,   // 2
145         0xF0, 0x10, 0xF0, 0x10, 0xF0,   // 3
146         0x90, 0x90, 0xF0, 0x10, 0x10,   // 4
147         0xF0, 0x80, 0xF0, 0x10, 0xF0,   // 5
148         0xF0, 0x80, 0xF0, 0x90, 0xF0,   // 6
149         0xF0, 0x10, 0x20, 0x40, 0x40,   // 7
150         0xF0, 0x90, 0xF0, 0x90, 0xF0,   // 8
151         0xF0, 0x90, 0xF0, 0x10, 0xF0,   // 9
152         0xF0, 0x90, 0xF0, 0x90, 0x90,   // A
153         0xE0, 0x90, 0xE0, 0x90, 0xE0,   // B
154         0xF0, 0x80, 0x80, 0x80, 0xF0,   // C
155         0xE0, 0x90, 0x90, 0x90, 0xE0,   // D
156         0xF0, 0x80, 0xF0, 0x80, 0xF0,   // E
157         0xF0, 0x80, 0xF0, 0x80, 0x80,   // F
158     };
159 
160     memset(chipStruct, 0, sizeof(CHIP_Struct));
161 
162     // Loading the Font into the RAM
163     memcpy(&chipStruct->RAM[0], chipFont, sizeof(chipFont));
164 
165     // Load ROM
166     FILE *romFile = fopen(ROM_NAME, "rb");
167     if(!romFile){
168         SDL_Log("[ERROR] %s ROM Not Found\n", ROM_NAME);
169         return false;
170     }
171     fseek(romFile, 0, SEEK_END);
172     const size_t romSize = ftell(romFile);
173     rewind(romFile);
174 
```

```
    if(romSize > sizeof(chipStruct->RAM) - entryPoint){
        SDL_Log("[ERROR] %s ROM Size Exceeds the Maximum Allowed Memory -- ROM SIZE: %ld -- MEM SIZE: %lu\n", ROM_NAME, romSize, sizeof(chipStruct->RAM) - entryPoint);
        return SDL_FALSE;
    }

    if(fread(&chipStruct->RAM[entryPoint], romSize, 1, romFile) != 1){
        SDL_Log("[ERROR] Unable to Read ROM: %s into CHIP8 Memory", ROM_NAME);
        return false;
    }
    fclose(romFile);

    // Setting the CHIP8 Default State
    chipStruct->EM_State = RUNNING;
    chipStruct->PC = entryPoint;
    chipStruct->ROM_NAME = ROM_NAME;
    chipStruct->STACK_PTR = &chipStruct->STACK[0];
    return SDL_TRUE;
}

#ifdef DEBUG
void print_debug_info(CHIP_Struct *chipStruct){
    printf("[DEBUG] ADDR: 0x%04X, OP: 0x%04X, DESC: ", (chipStruct->PC)-2, chipStruct->Inst.OP);
    switch((chipStruct->Inst.OP >> 12) & 0x0F){
        case 0x0:
            switch(chipStruct->Inst.NN){
                case 0xE0:
                   printf("Screen Cleared\n");
                   break;
                case 0xEE:
                    printf("Return from Subroutine to Address 0x%04X\n", (chipStruct->STACK_PTR - 1));
                    break;
                default:
                    printf("OPCode Unimplemented\n");
                    break;
            }
            break;
        case 0x01:
            printf("Jump to Address NNN 0x%04X\n", (chipStruct->Inst.NNN));
            break;
        case 0x02:
            printf("Started Subroutine of Address 0x%04X\n", (chipStruct->STACK_PTR));
            break;
        case 0x0A:
            printf("Set I to NNN\n");
            break;
        case 0x06:
            printf("Set Register V%X = NN (0x%02X)\n", chipStruct->Inst.X, chipStruct->Inst.NN);
            break;
        case 0x07:
            printf("Set Register V%X (0x%02X) += NN (0x%02X)-- Result: (0x%02X)\n", chipStruct->Inst.X,chipStruct->V[chipStruct->Inst.X] ,chipStruct->Inst.NN, chipStruct->V[chipStruct->Inst.X], chipStruct->Inst.NN);
            break;
        case 0x0D:
```

```
            printf("Drawing N(Height): %u COORDS -- V%X: (0x%02X) V%X: (0x%02X) --
MEM I: (0x%02X)\n", chipStruct->Inst.N, chipStruct->Inst.X, chipStruct-
>V[chipStruct->Inst.X], chipStruct->Inst.Y, chipStruct->V[chipStruct->Inst.Y],
chipStruct->I);
            break;
        default:
            printf("OPCode Unimplemented\n");
            break;
    }
}
#endif

void emulate_inst(CHIP_Struct *chipStruct, CFG_Struct *configStruct){
    chipStruct->Inst.OP = (chipStruct->RAM[chipStruct->PC] << 8) | chipStruct-
>RAM[chipStruct->PC + 1];
    chipStruct->PC += 2;
    chipStruct->Inst.NNN = chipStruct->Inst.OP & 0x0FFF;
    chipStruct->Inst.NN = chipStruct->Inst.OP & 0x0FF;
    chipStruct->Inst.N = chipStruct->Inst.OP & 0x0F;
    chipStruct->Inst.X = (chipStruct->Inst.OP >> 8) & 0x0F;
    chipStruct->Inst.Y = (chipStruct->Inst.OP >> 4) & 0x0F;

#ifdef DEBUG
    print_debug_info(chipStruct);
#endif

    switch((chipStruct->Inst.OP >> 12) & 0x0F){
        case 0x00:
            switch(chipStruct->Inst.NN){
                case 0xE0:
                   memset(&chipStruct->DISPLAY[0], false, sizeof chipStruct-
>DISPLAY);
                   break;
                case 0xEE:
                   chipStruct->PC = *--chipStruct->STACK_PTR;
                   break;
            }
            break;
        case 0X01:
            chipStruct->PC = chipStruct->Inst.NNN;
            break;
        case 0x0A:
            chipStruct->I = chipStruct->Inst.NNN;
            break;
        case 0x02:
            *chipStruct->STACK_PTR++ = chipStruct->PC;
            chipStruct->PC = chipStruct->Inst.NNN;
            break;
        case 0x07:
            chipStruct->V[chipStruct->Inst.X] += chipStruct->Inst.NN;
            break;
        case 0x06:
            chipStruct->V[chipStruct->Inst.X] = chipStruct->Inst.NN;
            break;
        case 0x0D: {
            // 0xDXYN: Draw N-height sprite at coords X,Y; Read from memory location
I;
            //   Screen pixels are XOR'd with sprite bits,
            //   VF (Carry flag) is set if any screen pixels are set off; This is
useful
```

```
            //   for collision detection or other reasons.
            uint8_t X_coord = chipStruct->V[chipStruct->Inst.X] % configStruct->W_Width;
            uint8_t Y_coord = chipStruct->V[chipStruct->Inst.Y] % configStruct->W_Height;
            const uint8_t orig_X = X_coord; // Original X value

            chipStruct->V[0xF] = 0;  // Initialize carry flag to 0

            // Loop over all N rows of the sprite
            for (uint8_t i = 0; i < chipStruct->Inst.N; i++) {
                // Get next byte/row of sprite data
                const uint8_t sprite_data = chipStruct->RAM[chipStruct->I + i];
                X_coord = orig_X;   // Reset X for next row to draw

                for (int8_t j = 7; j >= 0; j--) {
                    // If sprite pixel/bit is on and display pixel is on, set carry flag
                    bool *pixel = &chipStruct->DISPLAY[Y_coord * configStruct->W_Width + X_coord];
                    const bool sprite_bit = (sprite_data & (1 << j));

                    if (sprite_bit && *pixel) {
                        chipStruct->V[0xF] = 1;
                    }

                    // XOR display pixel with sprite pixel/bit to set it on or off
                    *pixel ^= sprite_bit;

                    // Stop drawing this row if hit right edge of screen
                    if (++X_coord >= configStruct->W_Width) break;
                }

                // Stop drawing entire sprite if hit bottom edge of screen
                if (++Y_coord >= configStruct->W_Height) break;
            }
            break;
        }
        default:
            break;
    }
}

// void update_screen(const SM_Struct sdl, const CFG_Struct config, CHIP_Struct *chip8) {
//     SDL_Rect rect = {.x = 0, .y = 0, .w = config.Scale_Factor, .h = config.Scale_Factor};

//     // Grab bg color values to draw outlines
//     const uint8_t bg_r = (config.C_Background >> 24) & 0xFF;
//     const uint8_t bg_g = (config.C_Background >> 16) & 0xFF;
//     const uint8_t bg_b = (config.C_Background >>  8) & 0xFF;
//     const uint8_t bg_a = (config.C_Background >>  0) & 0xFF;

//     const uint8_t fg_r = (config.C_Foreground >> 24) & 0xFF;
//     const uint8_t fg_g = (config.C_Foreground >> 16) & 0xFF;
//     const uint8_t fg_b = (config.C_Foreground >>  8) & 0xFF;
//     const uint8_t fg_a = (config.C_Foreground >>  0) & 0xFF;

//     // Loop through display pixels, draw a rectangle per pixel to the SDL window
```

```
//     for (uint32_t i = 0; i < sizeof chip8->DISPLAY; i++) {
//         // Translate 1D index i value to 2D X/Y coordinates
//         // X = i % window width
//         // Y = i / window width
//         rect.x = (i % config.W_Width) * config.Scale_Factor;
//         rect.y = (i / config.W_Width) * config.Scale_Factor;

//         if (chip8->DISPLAY[i]) {
//             SDL_SetRenderDrawColor(sdl.Renderer,fg_r , fg_g, fg_b, fg_a);
//             SDL_RenderFillRect(sdl.Renderer, &rect);
//         } else {
//             SDL_SetRenderDrawColor(sdl.Renderer, bg_r, bg_g, bg_b, bg_a);
//             SDL_RenderFillRect(sdl.Renderer, &rect);
//         }
//     }

//     SDL_RenderPresent(sdl.Renderer);
// }

void update_screen(const SM_Struct *mainStruct, const CFG_Struct *configStruct,
CHIP_Struct *chipStruct){
    SDL_Rect rectVar = {0, 0, configStruct->Scale_Factor, configStruct-
>Scale_Factor};

    const uint8_t BG_RED = (configStruct->C_Background >> 24) & 0xFF;
    const uint8_t BG_GREEN = (configStruct->C_Background >> 16) & 0xFF;
    const uint8_t BG_BLUE = (configStruct->C_Background >> 8) & 0xFF;
    const uint8_t BG_ALPHA = (configStruct->C_Background >> 0) & 0xFF;

    const uint8_t FG_RED = (configStruct->C_Foreground >> 24) & 0xFF;
    const uint8_t FG_GREEN = (configStruct->C_Foreground >> 16) & 0xFF;
    const uint8_t FG_BLUE = (configStruct->C_Foreground >> 8) & 0xFF;
    const uint8_t FG_ALPHA = (configStruct->C_Foreground >> 0) & 0xFF;

    for (uint32_t i = 0; i < sizeof chipStruct->DISPLAY; i++) {
        rectVar.x = (i % configStruct->W_Width) * configStruct->Scale_Factor;
        rectVar.y = (i / configStruct->W_Width) * configStruct->Scale_Factor;

        if (chipStruct->DISPLAY[i]) {
            SDL_SetRenderDrawColor(mainStruct->Renderer,FG_RED , FG_GREEN, FG_BLUE,
FG_ALPHA);
            SDL_RenderFillRect(mainStruct->Renderer, &rectVar);
        } else {
            SDL_SetRenderDrawColor(mainStruct->Renderer, BG_RED, BG_GREEN, BG_BLUE,
BG_ALPHA);
            SDL_RenderFillRect(mainStruct->Renderer, &rectVar);
        }
    }

    SDL_RenderPresent(mainStruct->Renderer);
}

int main(int argc, char **argv){
    DISCARD_UNUSED(argc);
    DISCARD_UNUSED(argv);

    /* END OF BOILER PLATE CODE */

    SM_Struct mainStruct;
    CFG_Struct cfgStruct;
```

```
390     CHIP_Struct chipStruct;
391 
392     (!init_chip(&chipStruct, "IBM_Logo.ch8")) ? exit(EXIT_FAILURE) : println("
[SUCCESS] CHIP8 Initialized");
393 
394     // INITIALIZE SDL AND EXIT IF IT FAILS
395     (!init_sdl(&mainStruct, init_config(&cfgStruct))) ? exit(EXIT_FAILURE) :
println("[SUCCESS] Initialized SDL2");
396 
397     clrscrn(&cfgStruct, &mainStruct);
398 
399     //EMULATOR LOOP
400     while(chipStruct.EM_State != QUIT){
401         handle_input(&chipStruct);
402         if(chipStruct.EM_State == PAUSED) continue;
403         emulate_inst(&chipStruct, &cfgStruct);
404         SDL_Delay(16);
405         update_screen(&mainStruct, &cfgStruct, &chipStruct);
406     }
407 
408     //DEINITIALIZE SDL
409     SDL_DestroyRenderer(mainStruct.Renderer);
410     SDL_DestroyWindow(mainStruct.Window);
411     SDL_Quit();
412 }
413 
```