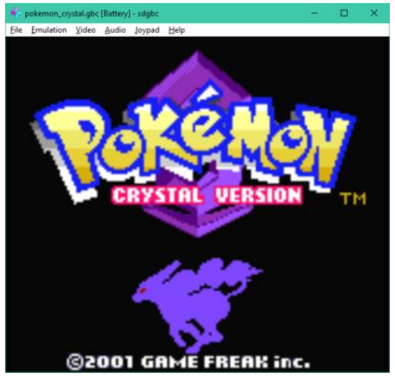
UNIVERSITY OF LEICESTER

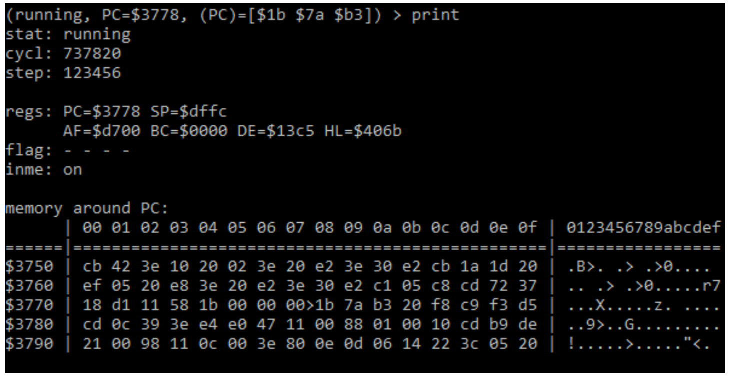
Software Emulator for the Nintendo Game Boy Color

Sean Dewar

Introduction

- Developed software that can run Game Boy and Game Boy Color programs on modern systems
  - Reads programs from ROM dumps (.gb, .gbc files)
- Uses a low-level emulation (LLE) approach
  - Replicates the operation of the GBC’s hardware in code
  - Essentially runs a virtual GBC guest machine
- Two modes of operation
  - GUI for interacting with programs (using wxWidgets)
  - Command-line interface for debugging CPU emulation





So, what *is* a Game Boy Color?

- Handheld video game console made by Nintendo
- Released in 1998
- Successor to the Game Boy
- Both systems sold a combined total of ~120 million units worldwide
- Contributed to the success of media franchises such as *Pokémon*, which saw its first few games released for the system

So, what *is* a Game Boy Color?

- 8-bit Sharp LR35902 CPU
  - Based on the Zilog Z80 and Intel 8080
  - Clock speed of ~4.2 or ~8.4 MHz
- 32 KB WRAM
- 16 KB VRAM
- 160 x 144-pixel colour LCD
- Joypad
  - Has a directional pad and four extra buttons
- Mono speaker
  - 2-channel stereo output possible with headphones

GAME BOY™ COLOR

So, what *is* a Game Boy Color?

- The GBC is backwards compatible with its predecessor, the Game Boy
  - Monochrome dot-matrix screen *(vs colour LCD)*
  - CPU locked at a 4.2 MHz clock speed *(vs 8.4 MHz max)*
  - 8 KB WRAM *(vs 32 KB)*
  - 8 KB VRAM *(vs 16 KB)*

- GB and GBC programs distributed in cartridges
  - 8 MB ROM maximum
  - 128 KB RAM maximum
    - Sometimes bundled with a battery and used for storing persistent data (leader board scores, player saves, etc.)

Why a Game Boy Color emulator?

- Interest in computer systems and video games
  - Emulation of video game consoles combines these two interests!
- Increase my knowledge of low-level programming and C++
  - C++ provides speed of native code and abstractions of low-level features such as pointers
- In a general, social context: hardware eventually fails
  - Emulation is necessary in the preservation of their history
  - In the case of the GBC, production of units stopped in 2003

Project Objectives

- Key Objectives *(musts)*
  - Emulation of important hardware components ✓
  - Implementation of a basic GUI ✓
  - Extensive testing ✓

- Optional Objectives *(mostly mays)*
  - Implementation of debugging tools ✓
  - Implementation of save states ?
  - Implementation of key remapping ✗
  - Implementation of input playback ✗

Legend:

✓ Fully met

? Partially met

✗ Not met

Testing Strategy

- Critical Hardware Emulation Tests
  - Automated unit tests that verify important hardware aspects
  - Failing these tests greatly affects emulated program compatibility
  - Many of these tests verify aspects of the emulated CPU

- Miscellaneous Hardware Emulation Tests
  - Typically automated unit tests that verify non-critical or obscure hardware aspects
  - Failing these tests produce superficial or no visible issues (e.g. minor graphical bugs)
  - Not a huge priority to pass all these tests

- Manual Program Compatibility Tests
  - Involves manually running real games released for the GB and GBC
  - Programs typically utilise all emulated hardware components to some degree
  - Mainly acts as full system tests

A Few Important Hardware & Emulation Aspects

Emulation

- All emulated hardware components are represented using classes
  - Contain methods for updating and hard resetting their states

- Emulation is driven by a loop that is run on a separate thread to the GUI
  - Emulates the GBC's clock speed via `sleep()` and frame skipping
  - Supports pausing and fast-forwarding
  - The `Emulator` class contains methods for interacting with the emulated system in a thread-safe manner

- The emulator automatically detects whether the loaded program should be ran in GB backwards compatibility mode
  - Achieved by parsing header data included inside of every program's ROM

Memory

- Emulated memory is represented using arrays
  - Dynamic sized arrays (`std::vectors`) are used for storing program ROM data and cartridge RAM
  - Fixed size arrays are used for all other memory
- All GBC memory is mapped within a single 16-bit address space
  - Emulated memory is mapped using simple if-else statements that test against each mapped address range
- Mapped I/O registers are used to configure and query the status of hardware components

Address Range	Memory Area
$0000 to $7FFF	Cartridge ROM
$8000 to $9FFF	VRAM
$A000 to $BFFF	Cartridge RAM *(i*
$C000 to $DFFF	WRAM
$FE00 to $FE9F	OAM-RAM
$FF00 to $FF2F	I/O Registers
$FF30 to $FF3F	Wave RAM
$FF40 to $FF7F	I/O Registers
$FF80 to $FFFE	HRAM
$FFFF	I/O Registers

CPU

- The emulated CPU is responsible for determining how many clock cycles other hardware components run for every time it is updated
- Implements 5 types of maskable interrupt
  - Requested by the program or other hardware when specific time sensitive conditions are met (e.g. when a frame of video has finished rendering)
- Program instructions are represented as machine code in ROM
  - 500 opcodes exist for representing all instruction and addressing mode configurations
  - Because opcodes are 8 bits, a special opcode (`0xCB`) is used as a prefix for representing 256 opcodes in an extended instruction set
  - Instruction decoding is implemented using a large switch statement, which compiles into a jump table

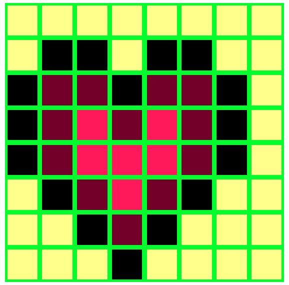
CPU

- The CPU exposes numerous program registers
  - Six 8-bit General Purpose registers (B, C, D, E, H, L)
  - An 8-bit Accumulator (A) stores the result of arithmetic and logical operations
  - An 8-bit Status Flag (F) stores the result of the previously executed arithmetic or logical operation
  - A 16-bit Program Counter (PC)
  - A 16-bit Stack Pointer (SP)

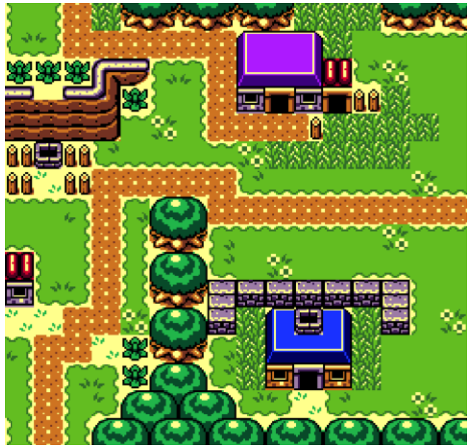
- Many 8-bit registers can be addressed as 16-bit register pairs
  - AF, BC, DE & HL

- The Curiously Recurring Template Pattern (CRTP) C++ idiom is used to represent the emulated CPU registers in code
  - This optimization allows emulated registers to inherit behaviour without virtual function call overhead
  - Requires concrete types to be known at compile-time

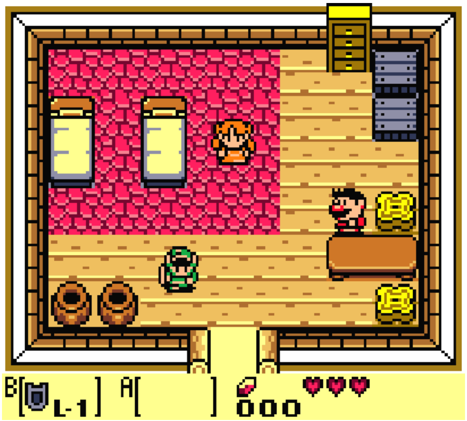
LCD Controller (PPU)



- All graphics to be rendered are stored in VRAM as individual 8 x 8-pixel bitmap tiles



- Assortments of 32 x 32 tiles, known as tile maps, are also stored in VRAM
- 3 tile layers are combined together to produce an output image
  - Background layer *(back)*
    - Uses a tile map for BG graphics
  - Window layer *(middle)*
    - Uses a tile map for UI graphics
  - Object layer *(front)*
    - Uses object attribute memory (OAM-RAM) for representing graphics that move independently from the other layers (e.g. projectiles)



LCD Controller (PPU)

- Emulated LCD output rendered using the SFML library
  - Provides cross-platform hardware acceleration for rendering (via OpenGL)

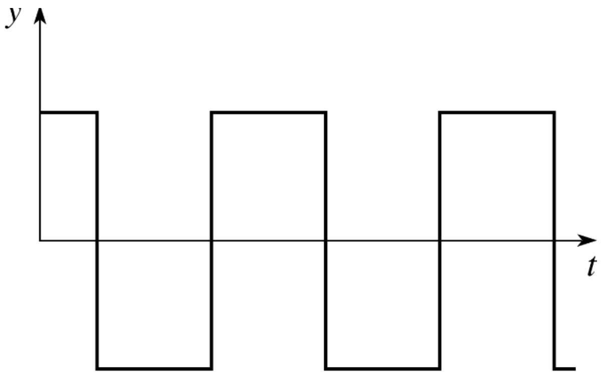
- The PPU renders on a per-scanline basis, alternating between 4 different modes of operation. The most important are:
  - H-Blank: when a single scanline has rendered
  - V-Blank: when a full frame of video has rendered (occurs at ~59.7 Hz)

- Implements direct memory access (DMA) circuitry
  - Allows programs to quickly copy data to VRAM and OAM-RAM

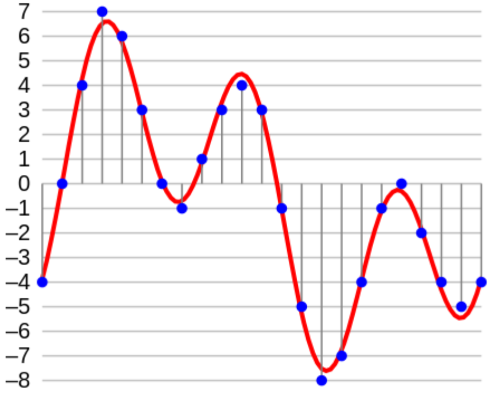
Audio Controller (APU)

- 2-channel stereo sound output streamed using SFML (via OpenAL)
  - Emulated output down-sampled to 44,100 Hz for compatibility with modern sound cards

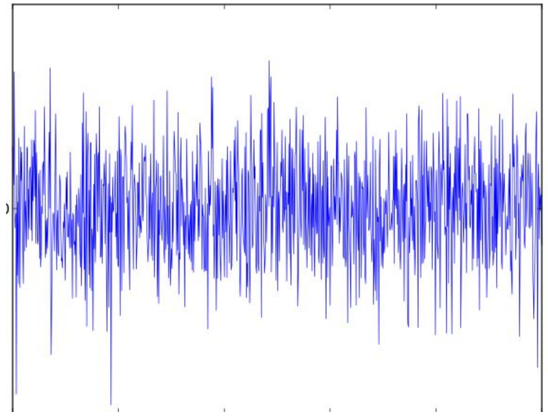
- Implements 4 waveform generation channels
  1. Square wave with sweep and envelope
  2. Square wave with envelope
  3. Voluntary waveforms (from 4-bit PCM samples in wave RAM)
  4. Pseudorandom white noise with envelope



Square wave



Voluntary wave

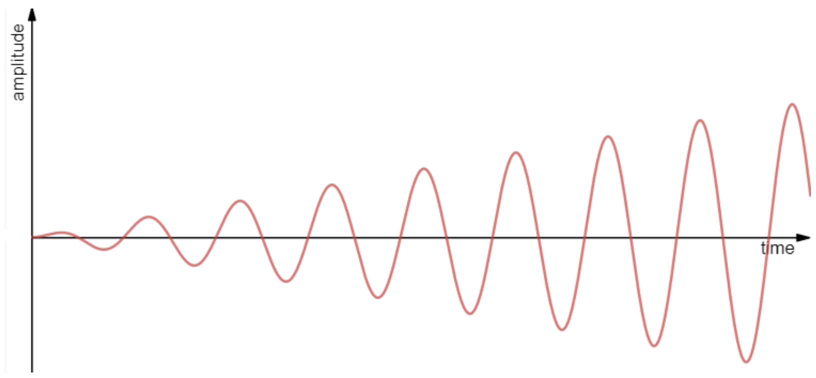


White noise wave

Audio Controller (APU)

- Volume envelope units
  - Gradually increases or decreases waveform volume over time, typically by modifying amplitude
- Frequency sweep units
  - Gradually increases or decreases waveform frequency over time
- Additionally, all channels include a length counter
  - Silences a channel after a specific amount of time
- These units are updated by a frame sequencer
  - Volume envelope units are updated at 64 Hz
  - Frequency sweep units are updated at 128 Hz
  - Length counters are updated at 256 Hz

Increasing amplitude



Increasing frequency