NES Programming in Rust

Sydney Rust Meetup 2023-03-01

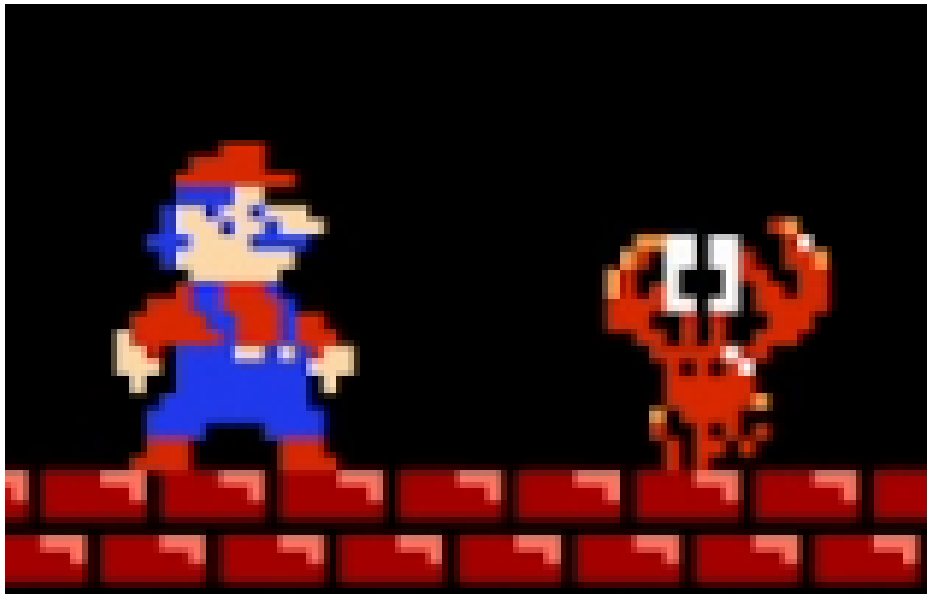
Stephen Sherratt (@gridbugs)

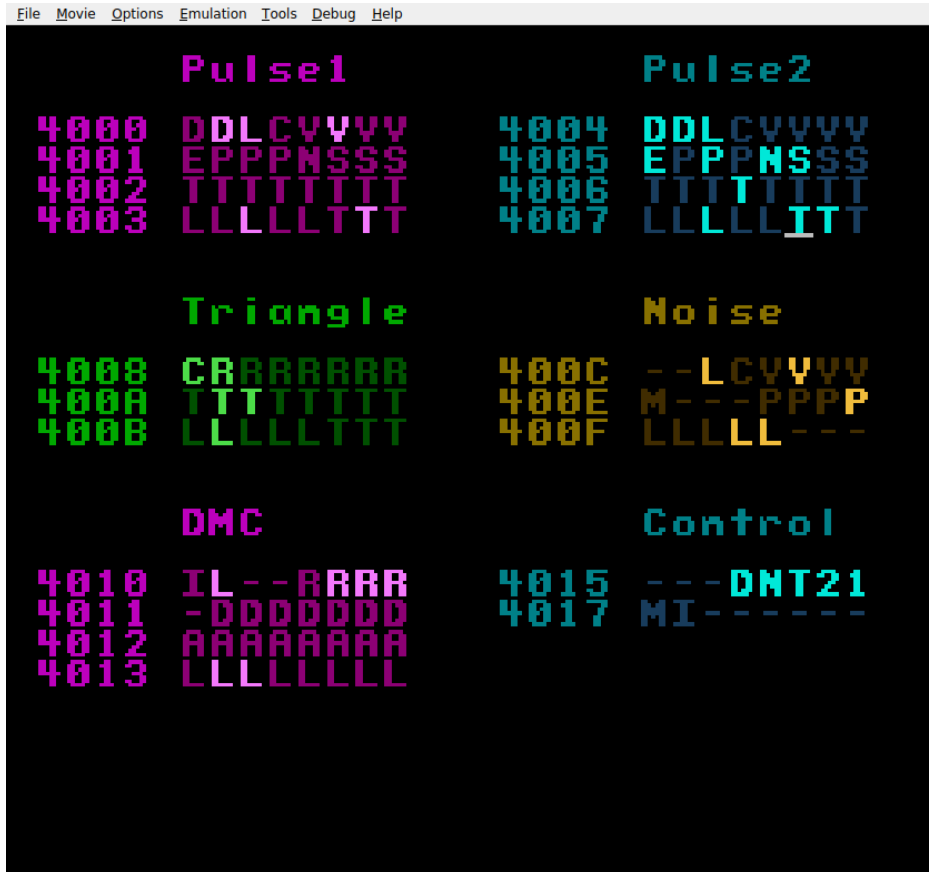
gridbugs.org

github.com/gridbugs

hachyderm.io/@gridbugs

twitch.tv/gridbugs





Demo (video) 🤞🤞🤞

https://youtu.be/QHoISiWdPXo

gridbugs / **nes-audio-playground** Public

Notifications | Fork 0 | Star 0

Code | Issues | Pull requests | Actions | Projects | Security | Insights

main | 1 branch | 0 tags | Go to file | Code

gridbugs Initial commit — 95811d9 — 16 hours ago — 1 commit

images	Initial commit	16 hours ago
src	Initial commit	16 hours ago
.gitignore	Initial commit	16 hours ago
Cargo.lock	Initial commit	16 hours ago
Cargo.toml	Initial commit	16 hours ago
README.md	Initial commit	16 hours ago
shell.nix	Initial commit	16 hours ago

README.md

NES Audio Playground

A tool for generating NES ROM files give access to the Audio Processing Unit's registers. Move the cursor with the d-pad, press A to flip a bit, and hold B to defer any bit flips until after B is released. Releasing B also has the side effect of rewriting the current value of the register under the cursor.

About

Tool for generating NES ROM files giving control over the bits in the APU's registers

Readme

0 stars

1 watching

0 forks

Releases

No releases published

Packages

No packages published

Languages

Rust 99.7% Nix 0.3%

gridbugs / **nes-audio-playground** Public

Notifications | Fork 0 | Star 0

Code | Issues | Pull requests | Actions | Projects | Security | Insights

main | 1 branch | 0 tags | Go to file | Code

gridbugs Initial commit — 95811d9 — 16 hours ago — 1 commit

images	Initial commit	16 hours ago
src	Initial commit	16 hours ago
.gitignore	Initial commit	16 hours ago
Cargo.lock	Initial commit	16 hours ago
Cargo.toml	Initial commit	16 hours ago
README	Initial commit	16 hours ago
shell.nix	Initial commit	16 hours ago

About

Tool for generating NES ROM files giving control over the bits in the APU's registers

Readme

0 stars

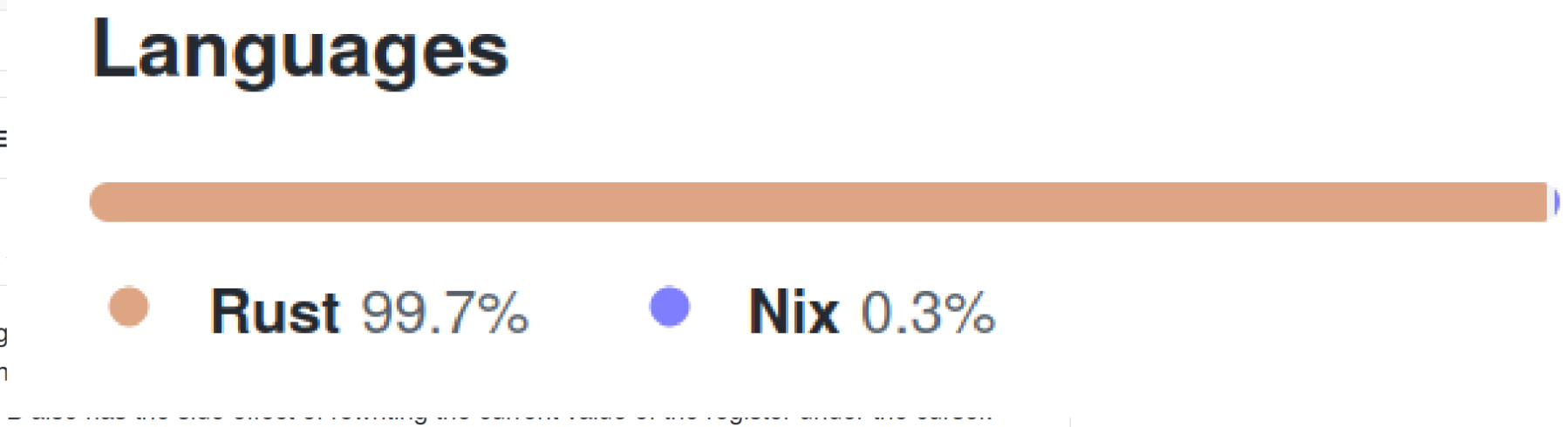
1 watching

0 forks

README

NES

A tool for g
cursor with
Releasing



Usage

```
cargo run -- -o playground.nes  # generate ROM file
fcoux playground.nes            # run ROM in NES emulator
```

Usage

```rust
use std::io::Write;
use ines::{Ines, Header};

let ines = Ines {
    header: Header { ... },
    chr_rom: chr_rom(), // tiles and sprites
    prg_rom: prg_rom(), // code and static data
};

let mut data = Vec::new();
ines.encode(&mut data);

let mut file = std::fs::File::create(output_path).unwrap();
file.write_all(&data).expect("Failed to write ROM file");
```

Character ROM

```
...
// 24: A
0b00111100,
0b01100110,
0b01100110,
0b01111110,
0b01100110,
0b01100110,
0b01100110,
...
```

Program ROM: Block

```
use mos6502_assembler::Block;

fn prg_rom() -> Vec<u8> {
    // A Block is an intermediate representation that keeps track of labels
    // and a cursor so you can put code/data at specific addresses.
    let mut b = Block::new();

    ...
}
```

ROM info from fceux NES emulator

```
Loading Sesame Street - Big Bird's Hide & Speak (USA).nes...

 PRG ROM: 16 x 16KiB = 256 KiB
 CHR ROM: 16 x  8KiB = 128 KiB
 ROM CRC32: 0xfde1c7ed
 ROM MD5:  0xe11377293fff45358d99aee90f98cbd6
 Mapper #: 1
 Mapper name: MMC1
 Mirroring: Horizontal
 Battery-backed: No
 Trained: No
```

Program ROM: Block

```
use mos6502_assembler::Block;

fn prg_rom() -> Vec<u8> {
    // A Block is an intermediate representation that keeps track of labels
    // and a cursor so you can put code/data at specific addresses.
    let mut b = Block::new();

    ...
}
```

Program ROM: Code/Data in EDSL

```
use mos6502_assembler::Block;

fn prg_rom() -> Vec<u8> {
    // A Block is an intermediate representation that keeps track of labels
    // and a cursor so you can put code/data at specific addresses.
    let mut b = Block::new();

    // describe program with EDSL
    b.inst(...);
    b.label(...);
    b.literal_byte(...);
    // ...etc


    ...
}
```

Program ROM: Assemble

```
use mos6502_assembler::Block;

fn prg_rom() -> Vec<u8> {
    // A Block is an intermediate representation that keeps track of labels
    // and a cursor so you can put code/data at specific addresses.
    let mut b = Block::new();

    // describe program with EDSL
    b.inst(...);
    b.label(...);
    b.literal_byte(...);
    // ...etc

    // convert from intermediate representation to byte array
    // (this pass is needed to resolve labels)
    let mut prg_rom = Vec::new();
    b.assemble(/* start address */ 0x8000, /* ROM bank size */ 0x4000, &mut prg_rom)
        .expect("Failed to assemble");
    prg_rom
}
```

6502 Assembler Rust EDSL

Defining and calling a function with string labels:

```
b.label("set_cursor_to_tile_coord"); // define a function with a label
b.inst(Txa, ());                     // x component passed in X register
b.inst(Asl(Accumulator), ());  // multiply by 8 (width of tile)
b.inst(Asl(Accumulator), ());
b.inst(Asl(Accumulator), ());
b.inst(Sta(Absolute), Addr(var::cursor::X));
b.inst(Tya, ());                     // y component passed in Y register
...
b.inst(Rts, ());                     // Return from subroutine
...
// call a function
b.inst(Ldx(ZeroPage), var::bit_table_entry::TILE_X);
b.inst(Ldy(ZeroPage), var::bit_table_entry::TILE_Y);
b.inst(Jsr(Absolute), "set_cursor_to_tile_coord");
```

6502 Assembler Rust EDSL

Static data:

```
b.label("blink_colour_table");
const BLINK_COLOURS: [u8; 8] = [
    0x20,
    0x20,
    0x10,
    0x10,
    0x00,
    0x00,
    0x10,
    0x10,
];
for c in BLINK_COLOURS {
    b.literal_byte(c);
}
...
b.inst(Tax, ()); // transfer the blink index into X register
b.inst(Ldy(AbsoluteXIndexed), "blink_colour_table"); // read current blink colour
b.write_ppu_address(0x3F11); // write the blink colour to the palette
b.inst(Sty(Absolute), Addr(0x2007));
```

6502 Assembler Rust EDSL

Platform-specific extension:

```
trait BlockNes {
    fn init_ppu(&mut self);
    fn write_ppu_address(&mut self, addr: u16);
    fn write_ppu_value(&mut self, value: u8);
    fn set_ppu_nametable_coord(&mut self, col: u8, row: u8);
    fn set_ppu_palette_universal_background(&mut self, value: u8);
    ...
}

impl BlockNes for Block { ... }

fn program(b: &mut Block) {
    b.inst(...);
    ...
}
```

6502 Assembler Rust EDSL

Rust is a macro language!

```
// Read 8 consecutive bytes from a little-endian address stored
// at var::bit_table_address::LO into a buffer beginning at
// var::bit_table_entry::START.
b.inst(Ldx(Immediate), 0);
for i in 0..8 {
    b.inst(Lda(XIndexedIndirect), var::bit_table_address::LO);
    b.inst(Sta(ZeroPage), var::bit_table_entry::START + i);
    b.inst(Inc(ZeroPage), var::bit_table_address::LO);
}
```

6502 Assembler Rust EDSL

Addressing mode errors are type errors:

```
b.inst(Inc(AbsoluteYIndexed), 0x0000);
```

```
error[E0277]: the trait bound
`AbsoluteYIndexed: instruction::inc::AddressingMode`
is not satisfied
```

INC

Operation: M + 1 → M

Addressing Mode
Zero Page
Zero Page, X
Absolute
Absolute, X

6502 Assembler Rust EDSL

How addressing mode errors are caught at compile time:

```rust
pub mod inc {
    pub trait AddressingMode: ReadData + WriteData { ... }

    impl AddressingMode for Absolute { ... }
    impl AddressingMode for AbsoluteXIndexed { ... }
    impl AddressingMode for ZeroPage { ... }
    impl AddressingMode for ZeroPageXIndexed { ... }

    pub struct Inst<A: AddressingMode>(pub A);

    pub fn interpret<A: AddressingMode, M: Memory>(
        _: A, cpu: &mut Cpu,
        memory: &mut M,
    ) -> u8 {
        let data = A::read_data(cpu, memory).wrapping_add(1);
        A::write_data(cpu, memory, data);
        cpu.status.set_negative_from_value(data);
        cpu.status.set_zero_from_value(data);
        cpu.pc = cpu.pc.wrapping_add(A::instruction_bytes());
        A::num_cycles()
    }
}
pub use inc::Inst as Inc;
```

INC

Operation: M + 1 → M

Addressing Mode
Zero Page
Zero Page, X
Absolute
Absolute, X

Addressing Mode Ergonomics

```
// immediate argument (1 byte)
b.inst(Lda(Immediate), 1);

// implied argument
b.inst(Lsr(Accumulator), ());

// address argument via string label
b.inst(Jsr(Absolute), "set_cursor_to_tile_coord");

// address argument literal
b.inst(Bit(Absolute), Addr(0x2002));

// relative offset via label (single byte so destination must be within +/- 127 bytes)
b.inst(Beq, LabelRelativeOffset("end_set_tile_offset"));
```

Addressing Mode Ergonomics

```
pub trait ArgOperand {
    type Operand: operand::Trait;
    fn program(self, block: &mut Block);
}

impl ArgOperand for u8 {
    type Operand = operand::Byte;
    fn program(self, block: &mut Block) { ... }
}

impl ArgOperand for () {
    type Operand = operand::None;
    fn program(self, _block: &mut Block) {}
}

impl ArgOperand for &'static str {
    type Operand = operand::Address;
    fn program(self, block: &mut Block) { ... }
}

pub struct Addr(pub Address);
impl ArgOperand for Addr {
    type Operand = operand::Address;
    fn program(self, block: &mut Block) { ... }
}

pub struct LabelRelativeOffset(pub &'static str);
impl ArgOperand for LabelRelativeOffset {
    type Operand = operand::Byte;
    fn program(self, block: &mut Block) { ... }
}
```

Real Example: Reading the controller button states

```
b.label("copy_controller_state_to_zp");
const CONTROLLER_REG: Addr = Addr(0x4016);

// copy the current controller state
b.inst(Lda(ZeroPage), var::controller::CURR);
b.inst(Sta(ZeroPage), var::controller::PREV);

// toggle the controller strobe bit to copy its current value into shift register
b.inst(Lda(Immediate), 1);
b.inst(Sta(Absolute), CONTROLLER_REG); // set controller strobe
b.inst(Sta(ZeroPage), var::controller::CURR); // store a 1 at destination
b.inst(Lsr(Accumulator), ()); // clear accumulator
b.inst(Sta(Absolute), CONTROLLER_REG); // clear controller strobe
                                       // shift each of the 8 bits of controller state from the shift register into address 0
b.label("copy_controller_state_to_zp_loop");
b.inst(Lda(Absolute), CONTROLLER_REG); // load single bit into LBS of acculumator
b.inst(Lsr(Accumulator), ()); // shift bit into carry flag
b.inst(Rol(ZeroPage), var::controller::CURR); // shift carry flag into 0, and MSB of 0 into carry flag

// if that set the carry flag, this was the 8th iteration
b.inst(Bcc, LabelRelativeOffset("copy_controller_state_to_zp_loop"));

b.inst(Lda(ZeroPage), var::controller::PREV);
b.inst(Eor(Immediate), 0xFF);
b.inst(And(ZeroPage), var::controller::CURR);
b.inst(Sta(ZeroPage), var::controller::PRESS_DELTA);

b.inst(Lda(ZeroPage), var::controller::CURR);
b.inst(Eor(Immediate), 0xFF);
b.inst(And(ZeroPage), var::controller::PREV);
b.inst(Sta(ZeroPage), var::controller::RELEASE_DELTA);

b.inst(Rts, ());
```

More NES shenanigans at gridbugs.org/tags/#nes

- Reverse-Engineering NES Tetris to add Hard Drop
- Conway's Game of Life on the NES in Rust
- Zelda Screen Transitions are Undefined Behaviour
- NES Emulator Debugging