Arduino: Anatomy of a Blink

Jack Leightcap

neuwireless.slack.com

leightcap.j@northeastern.edu

November 5, 2020

Outline

Languages

- Arduino IDE
- Bare C
- AVR Assembly
- Rust (new & fancy)
- Building steps for each language

Topics Along the Way

- Boolean Logic
- Systems Software
- Computer Architecture
- Comparison and Classifications of Programming Languages

Arduino IDE: Blink

```
void setup() {
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
    digitalWrite(LED_BUILTIN, HIGH);
    delay(1000);
    digitalWrite(LED_BUILTIN, LOW);
    delay(1000);
}
```

Figure: `blink.ino`

Arduino IDE: Analysis

Arduino Language

- Not *really* a programming language per se?
- Subset of C++ with Arduino-specific libraries/defaults
- Provides `pinMode()`, `LED_BUILTIN`, `HIGH`, etc.

Running

- Just press "Verify" and "Upload"
- Hides most build steps unless error
- Full output: `File` → `Preferences` → `verbose output`

Masks: Setting/Resetting Bits

Masking Bits to 1:

- $Y + 1 = 1$ and $Y + 0 = Y$ ($+$ is bitwise OR)
x	mask m	$x + m$
00001001	10000001	10001001
- `x = x | 0b10000001` → `x |= 0b10000001`

Masking Bits to 0:

- $Y * 0 = 0$ and $Y * 1 = Y$ ($*$ is bitwise AND)
x	mask m	$x * m$
00001001	01111110	00001000
- `x = x & 0b01111110` → `x &= 0b01111110` → `x &= ~0b10000001`

Arduino Port Registers

Data Direction Registers, `DDR*`

- 0 at position *i* indicates pin *i* is read only
- 1 at position *i* indicates pin *i* is write only

Data Registers, `PORT*`

- writing to read only undefined, reading from write only undefined

PORT B	PORT C	PORT D
digital pins 8-15	analog pins	digital pins 0-7
last two pins map to clock	ADCs	first two pins serial communication

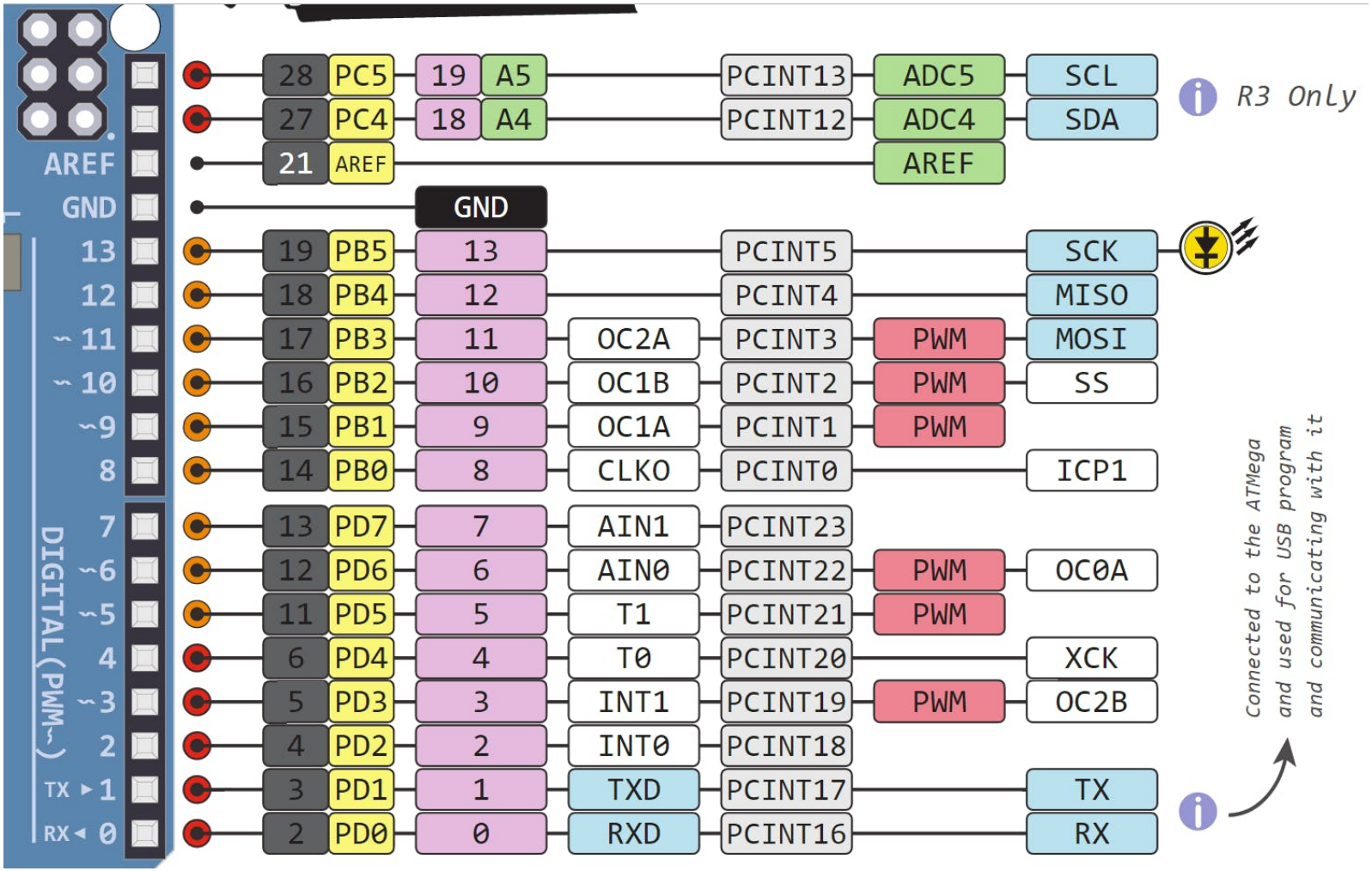

Figure: Arduino Uno Digital Pins, CC BY-SA

C: Blink

```c
int
main(void)
{
    DDRB |= _BV(DDB5);
    while(1) {
        PORTB |= _BV(PORTB5);
        _delay_ms(1000);
        PORTB &= ~_BV(PORTB5);
        _delay_ms(1000);
    }
    return 0;
}
```

Figure: `blink.c`

Systems Software: *Compiler*

- Transform the text of one programming language into another

$$\text{Source Language} \xrightarrow{\textit{compiler}} \text{Target Language}$$

- *Compiler* usually implies transforming into a lower level language
- *Decompiler* transforms language into something of higher level
- *Transpiler* transforms language into something of the same level

For the Arduino specifically, we'll be using two compilers:

- C to Arduino: `avr-gcc`
- Rust to Arduino: `LLVM` through `rustc`
- Target language is Intel HEX formatted for the Arduino CPU

C: Compiling and Deploying

```
# C -> ELF
$ avr-gcc -Os -DF_CPU=16000000UL -mmcu=atmega328p -o blink blink.c
# ELF -> HEX
$ avr-objcopy -O ihex -R .eeprom blink blink.hex
```

Figure: C $\xrightarrow{\texttt{avr-\{gcc,objcopy\}}}$ Intel HEX

```
# HEX -> Arduino
$ avrdude -F -V -c arduino -p ATMEGA328P -P /dev/ttyACM0 -b 115200 \
          -U flash:w:blink.hex
```

Figure: Intel HEX $\xrightarrow{\texttt{avrdude}}$ Arduino

Quick Intro to Assembly

Assembly 'Language'

- Not a programming language, but a family of languages
- CPU and Architecture specific!
- AVR Assembly, ARM Assembly, x86_64 Assembly, etc.
- Direct mapping to CPU instructions
- Oldest family of languages

Arduino Uno CPU and Architecture

- ATmega328P AVR microcontroller
- 8-bit RISC Instruction Set Architecture (ISA)
- 16MHz clock speed
- Harvard Architecture
- 32KB Flash memory store instructions/data, 2KB SRAM store data

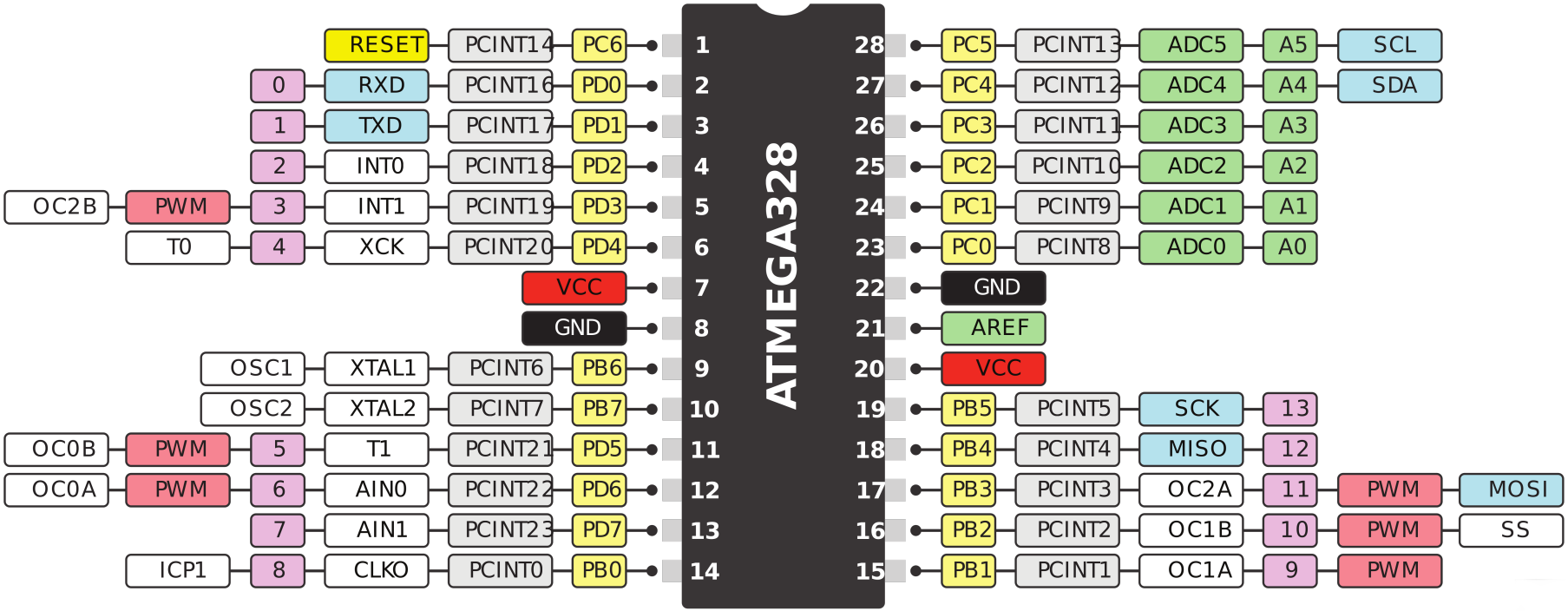

Figure: ATmega328P Pinout, CC BY-SA

Timings in Assembly

- How many clock cycles in 1s?

$$\underbrace{1\text{s}}_{\text{time}} \times \underbrace{\frac{16 * 10^6 \text{cycles}}{1\text{s}}}_{\text{clock speed}} = 16 * 10^6 \text{cycles}$$

- How many instructions is $16 * 10^6$ cycles?
- AVR is a RISC: instructions take exactly one cycle
- Use 3 registers (8-bits),

$$16 * 10^6 \text{cycles} \approx 256\text{cycles} \times 256\text{cycles} \times 244\text{cycles}$$

- Overflow two registers ($255 \rightarrow 0$), count down from 244 in another

AVR Assembly: Blink

```
.equ DDRB, 0x04
.equ PORTB, 0x05
.org 0x0000
rjmp RESET
RESET:
    ldi R16, 0x20
    out DDRB, R16
    ldi R18, 0x00
    ldi R17, 0x00
    ldi R20, 0x20
; cont...
```

```
Loop:
    ldi R19, 0xF4
delay:
    inc R17
    cpi R17, 0x00
    brne delay
    inc R18
    cpi R18, 0x00
    brne delay
    inc R19
    cpi R19, 0x00
    brne delay
    eor R16, R20
    out PORTB, R16
    rjmp Loop
```

Figure: `blink.asm`

Systems Software: *Assembler*

- Tranform Assembly into machine code

$$\text{Assembly} \xrightarrow{\textit{assembler}} \text{Machine Code}$$

- Maybe a type of compiler, is machine code a language?
- Some optimizations (jump length)
- Definitions: variable names (`DDRB`, `PORTB`), locations (`RESET`, `delay`)
- Directives: specify location with `.org 0x0000`
- Compiler calls Assembler in background

For the Arduino specifically:

- `avr-as`, what `avr-gcc` calls in the background
- Rust targeting AVR still requires `avr-as`, but choice of assembler for other architectures

AVR Assembly: Assembling and Deploying

```
# AVR Assembly -> Object file
$ avr-as blink.asm -o blink.o
# Object file -> ELF
$ avr-ld -Ttext 0 -nostdlib -nostartfiles  blink.o -o blink.elf
# ELF -> HEX
$ avr-objcopy -O ihex blink.elf blink.hex
```

Figure: AVR Assembly $\xrightarrow{\texttt{avr-\{as,ld,objcopy\}}}$ Intel HEX

```
# HEX -> Arduino
$ avrdude -F -V -c arduino -p ATMEGA328P -P /dev/ttyACM0 -b 115200 \
          -U flash:w:blink.hex
```

Figure: Intel HEX $\xrightarrow{\texttt{avrdude}}$ Arduino

Quick Intro to Rust

Language	First Appeared
Assembly	Ada Lovelace (1843), Kathleen Booth (1947) Atmel (1996)
C	Dennis Ritchie and Ken Thompson (1972)
C++	Bjarne Stroustrup (1985) Arduino (2005)
Rust	Graydon Hoare (2010)

- Very new (comparatively)
- Lots of benefits a small example won't show
- Memory safety, concurrency, high-level abstraction facilities
- "Most loved programming language" every year since 2016, Stack Overflow Developer Survey

Rust: Blink

```rust
#![feature(llvm_asm)]
#![no_std]
#![no_main]
use ruduino::cores::atmega328 as avr_core;
use ruduino::Register;
use avr_core::{DDRB, PORTB};
fn small_delay() {
    for _ in 0..400000 { unsafe{llvm_asm!("" :::: "volatile")} }
}
#[no_mangle]
pub extern fn main() {
    DDRB::set_mask_raw(0xFFu8);
    loop {
        PORTB::set_mask_raw(0xFF);   small_delay();
        PORTB::unset_mask_raw(0xFF); small_delay();
    }
}
```

Figure: `blink.rs`

Systems Software: *Build Systems*

- Saw the guts of how "Verify" and "Upload" work
- As complexity increases, building software becomes its own task
- Rust provides `cargo`
- `Makefile` common for C, `CMake` for C++; not provided by language
- Dependency management, build steps

```
[package]
name = "blink"
version = "0.1.0"
authors = ["Dylan McKay <me@dylanmckay.io>"]
edition = '2018'
[dependencies]
ruduino = "0.2"
[profile.dev]
panic = "abort"
```

Figure: `Cargo.toml` for `blink.rs`

Rust: Building and Deploying

```
# Rust -> ELF
$ cargo build -Z build-std=core --target avr-atmega328p.json
```

Figure: AVR Assembly $\xrightarrow{\texttt{avr-\{as,ld,objcopy\}}}$ Intel HEX

```
# ELF -> Arduino
$ avrdude -D -F -V -c arduino -p ATMEGA328P -P /dev/ttyACM0 -b 115200 \
          -U flash:w:$(BIN):e
```

Figure: ELF $\xrightarrow{\texttt{avrdude}}$ Arduino