Microcontrollers the Hard Way

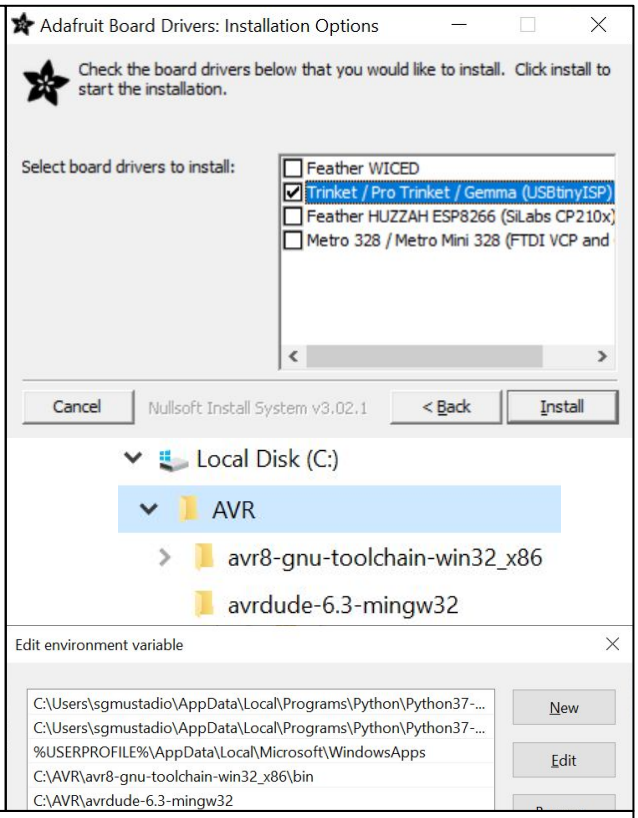
Worksheet by Shawn Hymel (License: CC BY 4.0)
Presented by DigiKey Electronics

Name:_____
Date:_____

Toolchain Setup

Installation (Windows)

- Install driver: bit.ly/adafruit-drivers-install
  - Select **Trinket…(USBtinyISP)**
  - Now plug in AVR Programmer
- Create folder C:\AVR
- Download AVR 8-bit Toolchain for Windows (v3.62) from: bit.ly/avr-toolchain-install
  - Unzip into C:\AVR
- Download *avrdude-6.3-mingw32.zip* from: http://bit.ly/avrdude-install
  - Unzip into C:\AVR\avrdude-6.3-mingw32
- Update Path
  - Control Panel > System and Security > System > Advanced System Settings > Advanced tab > Environment Variables
  - *User Variables*, select **Path**, click **Edit**
  - Click **New**, Click **Browse…**, add C:\AVR\avr8-gnu-toolchain-win32_x86\bin
  - Click **New**, Click **Browse…**, add C:\AVR\avrdude-6.3-mingw32



Installation (Mac)

- Install Homebrew:

```
$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

- Install AVR toolchain:

```
$ brew tap osx-cross/avr
$ brew install avr-gcc
```

- Install avrdude:

```
$ brew install avrdude
```

Installation (Linux)

- Install AVR toolchain and avrdude:

```
$ sudo apt update
$ sudo apt install -y gcc-avr binutils-avr avr-libc avrdude
```

Test It!

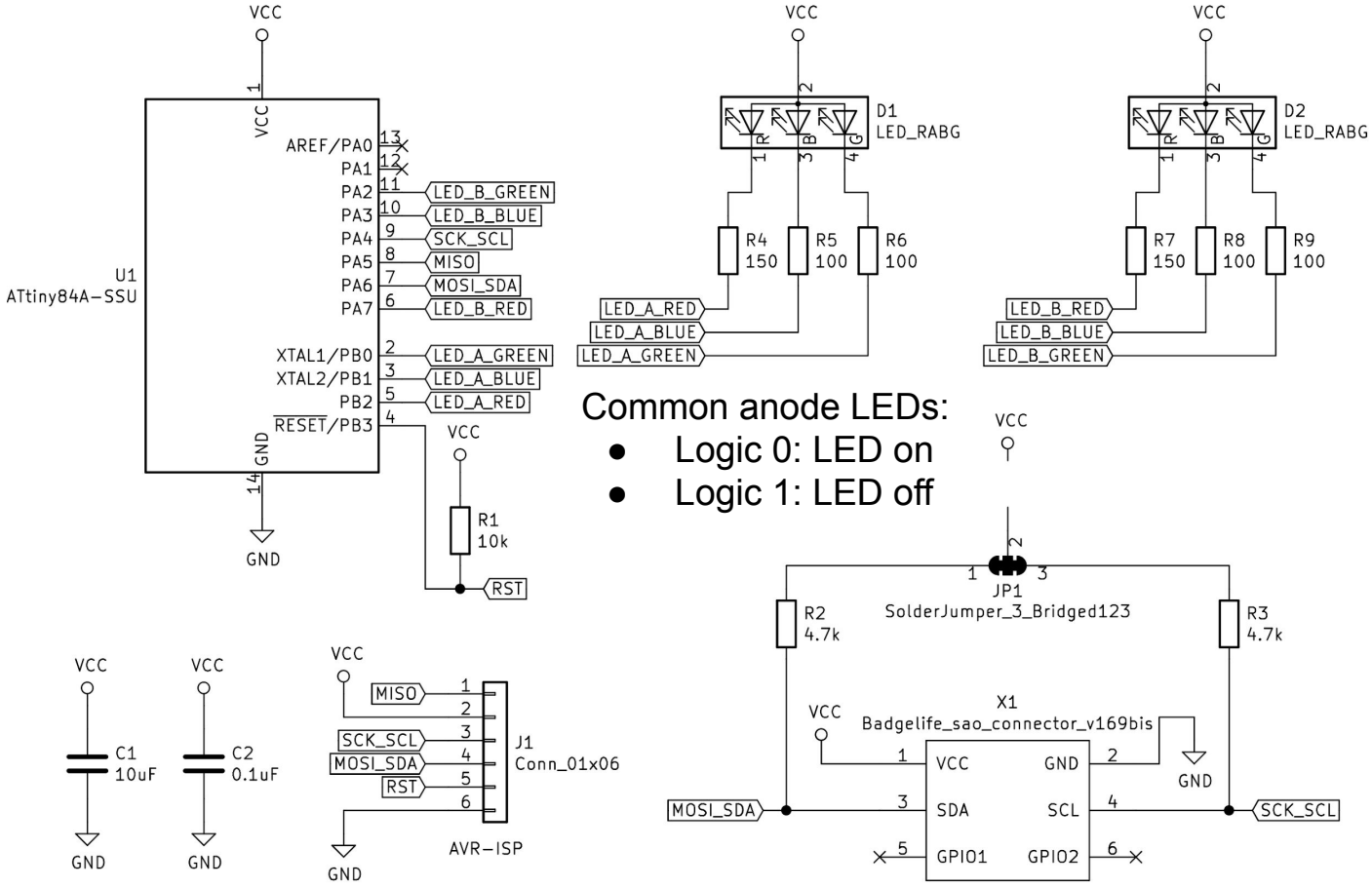
- In a terminal, enter `$ avr-gcc`
  - You should see an error: avr-gcc: fatal error: no input files
- Next, enter `$ avrdude`
  - You should see an error: `Usage: avrdude [options]`

Project Files

- Download ZIP from github.com/ShawnHymel/wenk-sao
- Unzip somewhere on your computer

References

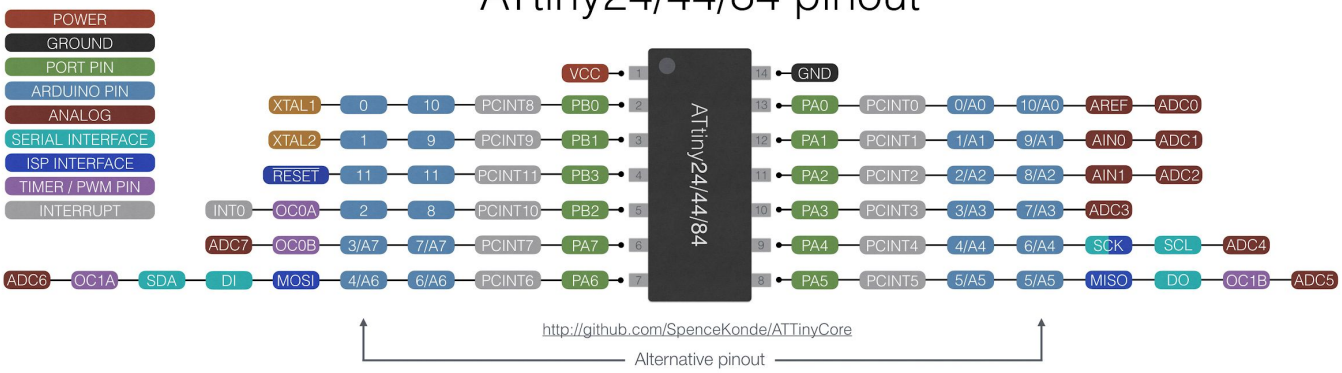
- Datasheet: **datasheet-avr-attiny-24a-44a-84a.pdf** in *wenk-sao/Documentation*
- Schematic:



Common anode LEDs:

- Logic 0: LED on
- Logic 1: LED off

ATtiny24/44/84 pinout




```
PORTA |= (1 << 7);
```

Sets bit 7 to 1 while keeping rest

`(1 << 7) = b10000000`

```
PORTA → 00011010
 OR     10000000
        10011010
```

```
PORTA &= ~(1 << 7);
```

Clears bit 7 to 0 while keeping rest

`~(1 << 7) = b01111111`

```
PORTA → 10111010
 AND    01111111
        00111010
```

```
PORTA ^= (1 << 7);
```

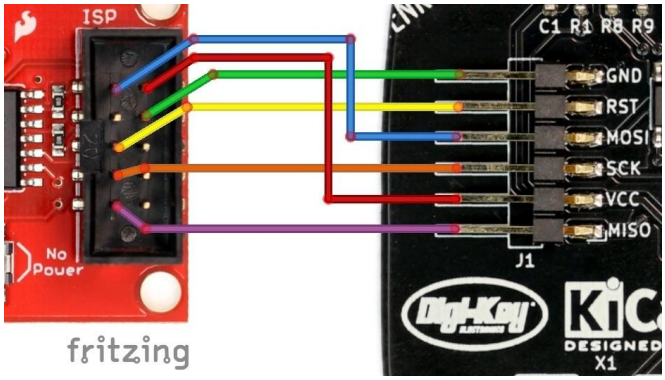
Toggles just bit 7

`(1 << 7) = b10000000`

```
PORTA → 10011010
 XOR    10000000
        00011010
```

Exercise 01 - Blinky!

Hardware Connections



Write Code

- Create AVR Projects folder somewhere on your computer
- In it, create *blinky.c*
- Open *blinky.c* with your favorite editor (not WordPad)
- In *blinky.c*, enter the code on the right
- Save

```c
#ifndef F_CPU
#define F_CPU 1000000UL // 1 MHz clock speed
#endif

#include <avr/io.h>
#include <util/delay.h>

int main(void) {

    // Make Ports A and B all output
    DDRA = 0b11111111;
    DDRB = 0b11111111;

    // Infinite loop
    while(1) {

        // Turn on all LEDs
        PORTA = 0x00;
        PORTB = 0x00;

        // Wait 1 second
        _delay_ms(1000);

        // Turn off all LEDs
        PORTA = 0xFF;
        PORTB = 0xFF;

        // Wait 1 second
        _delay_ms(1000);
    }
}
```

Compile and Upload

Open new terminal, enter commands:

```
$ cd <location of your AVR Projects directory>
$ avr-gcc -Os -mmcu=attiny84 -o blinky.elf blinky.c
$ avr-objcopy -j .text -j .data -O ihex blinky.elf blinky.hex
$ avrdude -c usbtiny -p t84 -U flash:w:blinky.hex
```

Or if you have Python (from wenk-sao directory):

```
$ python upload.py <path/to/blinky.c>
```

Hint: up arrow is your friend in the terminal to replay commands

Challenge: Flash only the green LED instead of all 3 LEDs

Exercise 02 - Blinky with Timer 1

Write Code

- Create *timer-blinky.c*
- Open *timer-blinky.c* with your favorite editor (not WordPad)
- In *timer-blinky.c*, enter the code on the right
- Save

Important Datasheet Sections

- 9.1 - Interrupt Vectors
- 10.3 - Register Descriptions for I/O Ports
- 12.11 - Register Descriptions for Timer/Counter 1

```c
#include <avr/io.h>
#include <avr/interrupt.h>

// Timer values (1 sec with prescaler of 64)
const uint16_t t1_load = 0;
const uint16_t t1_comp = 15625;

int main(void) {

    // Make only red LED pins output (PA7, PB2)
    DDRA = (1 << 7);
    DDRB = (1 << 2);

    // Set Timer 1 to normal operation
    TCCR1A = 0;
    TCCR1B = 0;

    // Set prescaler to 64
    TCCR1B |= (1 << CS11) | (1 << CS10);

    // Reset Timer 1 and set compare values
    TCNT1 = t1_load;
    OCR1A = t1_comp;

    // Enable Timer 1 compare interrupt
    TIMSK1 = (1 << OCIE1A);

    // Enable global interrupts
    sei();

    // Infinite loop
    while(1) {
        // Do nothing
    }
}

// Interrupt service routine
ISR(TIM1_COMPA_vect) {

    // Reset Timer 1
    TCNT1 = t1_load;

    // Toggle red LEDs
    PORTA ^= (1 << 7);
    PORTB ^= (1 << 2);
}
```

Compile and Upload

```
$ avr-gcc -Os -mmcu=attiny84 -o timer-blinky.elf timer-blinky.c
$ avr-objcopy -j .text -j .data -O ihex timer-blinky.elf timer-blinky.hex
$ avrdude -c usbtiny -p t84 -U flash:w:timer-blinky.hex
```

Or if you have Python (from wenk-sao directory):

```
$ python upload.py <path/to/timer-blinky.c>
```

Challenge: Flash green LED with a delay of 500 ms instead of 1 second.

Exercise 03 - Hardware PWM

Write Code

- Create *hardware-pwm.c*
- Open *hardware-pwm.c* with your favorite editor (not WordPad)
- In *hardware-pwm.c*, enter the code on the right
- Save

Important Datasheet Sections

- 10.3 - Register Descriptions for I/O Ports
- 11.9 - Register Descriptions for Timer/Counter 0

```c
// Need to define clock speed for delay functions
#ifndef F_CPU
#define F_CPU 1000000UL // 1 MHz clock speed
#endif

#include <avr/io.h>
#include <util/delay.h>

int main(void) {

    // Make only red LED pins output (PA7, PB2)
    DDRA = (1 << 7);
    DDRB = (1 << 2);

    // Set Timer 0 to fast PWM
    TCCR0A = (1 << WGM01) | (1 << WGM00);

    // Set Compare Output modes to inverting PWM
    TCCR0A |= (1 << COM0A1) | (1 << COM0A0);
    TCCR0A |= (1 << COM0B1) | (1 << COM0B0);

    // Set prescaler to 1 (no prescaling)
    TCCR0B = (1 << CS00);

    // Infinite loop
    while(1) {

        int16_t d;

        // Increase left eye, decrease right eye
        for ( d = 0; d <= 255; d++ ) {
            OCR0A = (uint8_t)d;         // Left eye
            OCR0B = (uint8_t)255 - d;   // Right eye
            _delay_ms(2);
        }

        // Decrease left eye, increase right eye
        for ( d = 255; d >= 0; d-- ) {
            OCR0A = (uint8_t)d;         // Left eye
            OCR0B = (uint8_t)255 - d;   // Right eye
            _delay_ms(2);
        }
    }
}
```

Compile and Upload

```
$ avr-gcc -Os -mmcu=attiny84 -o hardware-pwm.elf hardware-pwm.c
$ avr-objcopy -j .text -j .data -O ihex hardware-pwm.elf hardware-pwm.hex
$ avrdude -c usbtiny -p t84 -U flash:w:hardware-pwm.hex
```

Or if you have Python (from wenk-sao directory):

```
$ python upload.py <path/to/hardware-pwm.c>
```

Challenge: Notice that a duty cycle of 0% does not completely turn off LED. This is because in Fast PWM mode, at least 1 clock cycle happens while pin is low (LED on). Fix this by using Phase Correct PWM.

Hint: Read about the WGM0 bits in the TCCR0A register (datasheet section 11.9.1)

Exercise 04 - Software PWM

This is more for fun! Open `wenk-sao/Firmware/04-software-pwm/software-pwm.c` and play around with it.

Compile and Upload

```
$ avr-gcc -Os -mmcu=attiny84 -o software-pwm.elf software-pwm.c
$ avr-objcopy -j .text -j .data -O ihex software-pwm.elf software-pwm.hex
$ avrdude -c usbtiny -p t84 -U flash:w:software-pwm.hex
```

Or if you have Python (from wenk-sao directory):

```
$ python upload.py <path/to/software-pwm.c>
```

Challenge: Make some fun color patterns with the eyes. Check out `rainbow-pattern.c` to see how to remove LED flicker.