Microcontrollers the Hard Way

Blink Like a Pro

Please clone or download repo from bit.ly/wenk-sao

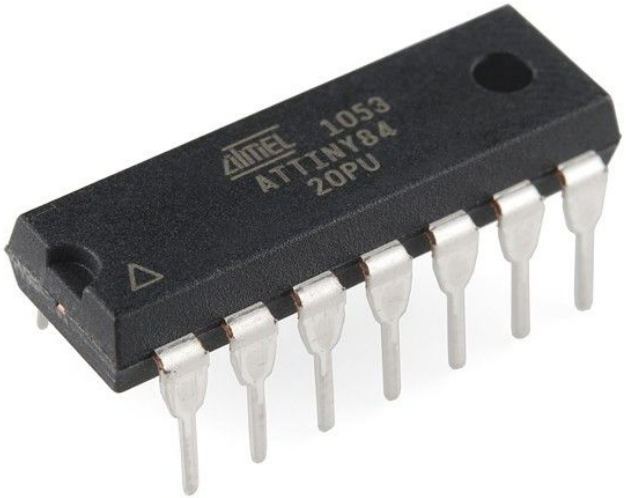
Install instructions in Documentation/ Worksheet PDF

Many thanks to the amazing folks at Digi-Key ELECTRONICS

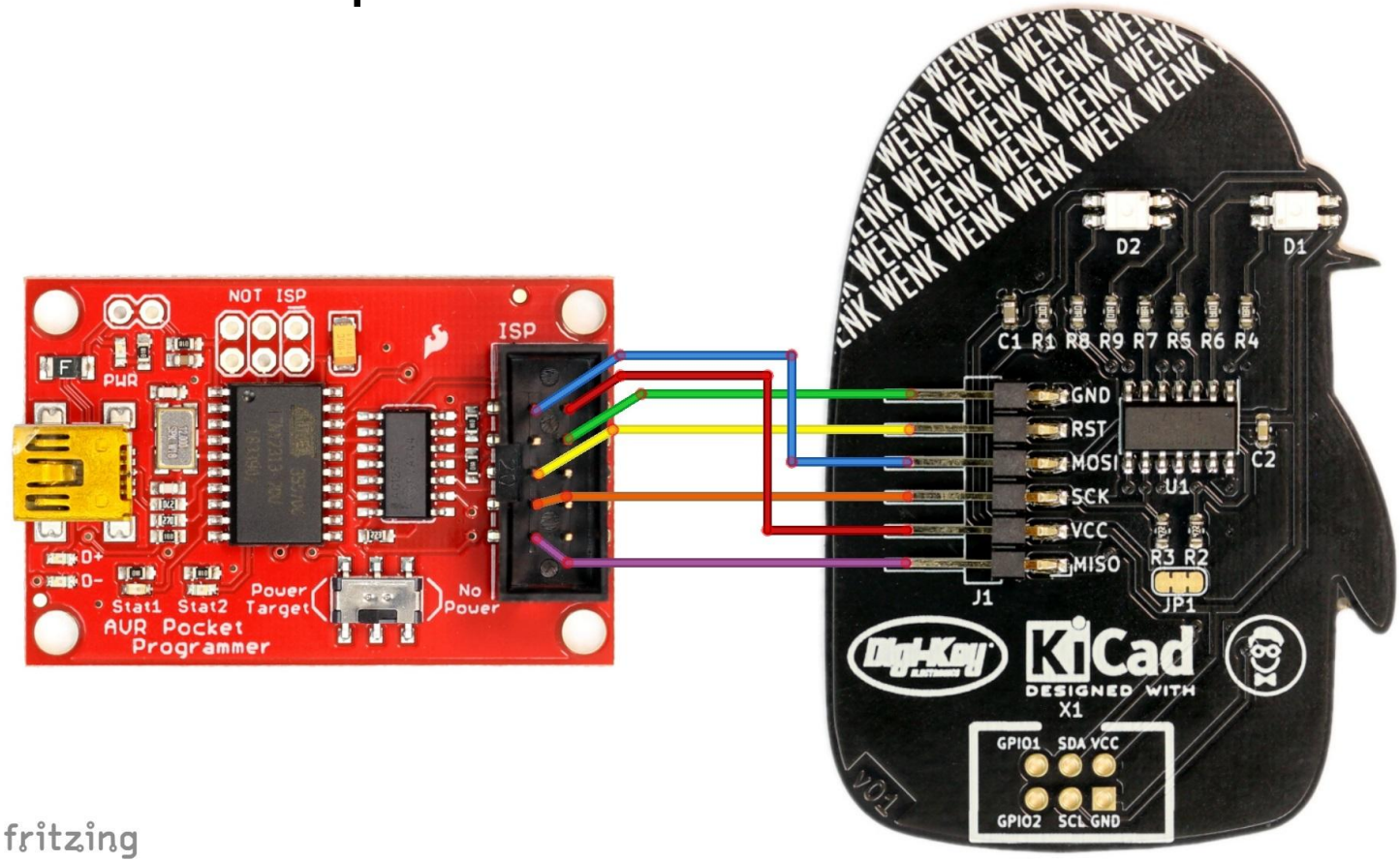
Why?

Please install AVR toolchain and AVRDUDE while I blab (bit.ly/wenk-sao)

- Arduino, Python, MicroPython, CircuitPython are much easier! (Calculator without math)
- For product development…
  - C is usually faster and more space efficient
  - Need to understand architecture
- But AVR is old! 32-bit is the new hotness
  - Registers
  - Interrupts
  - Timers
  - Peripherals

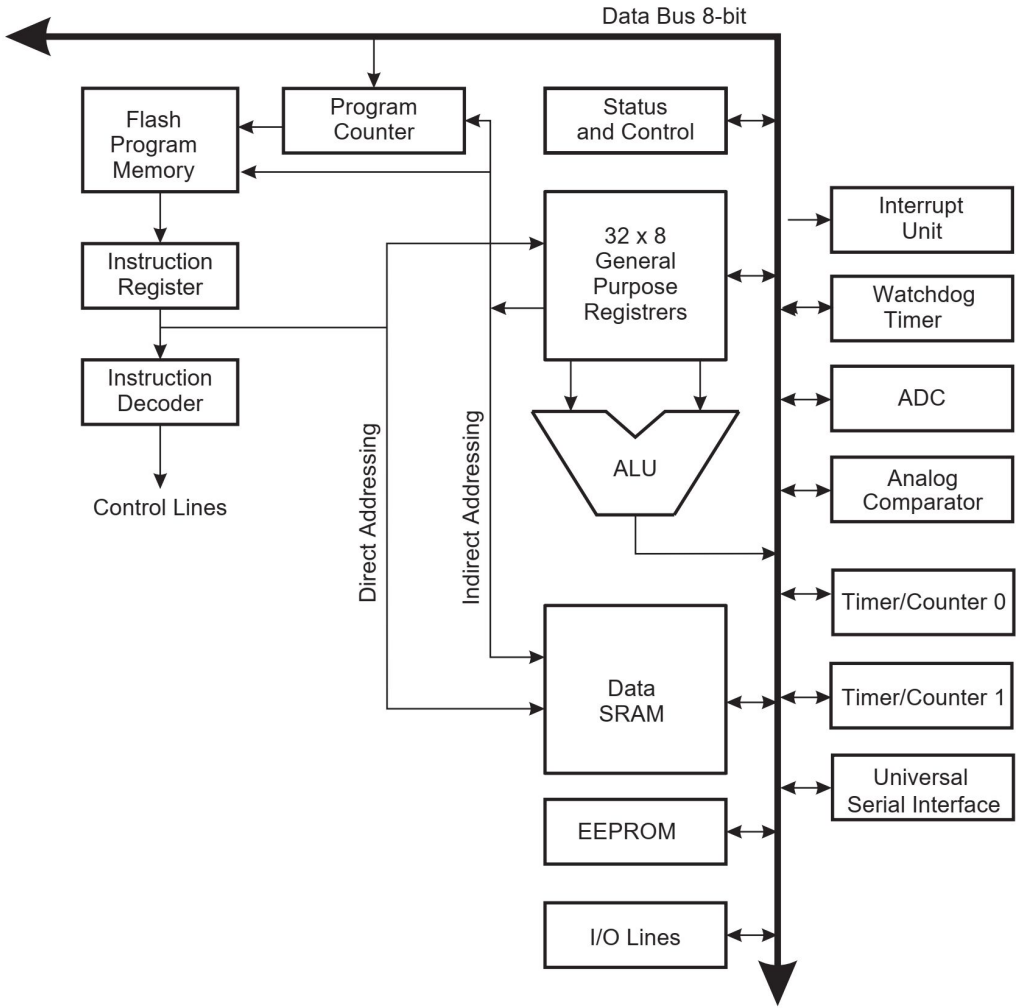


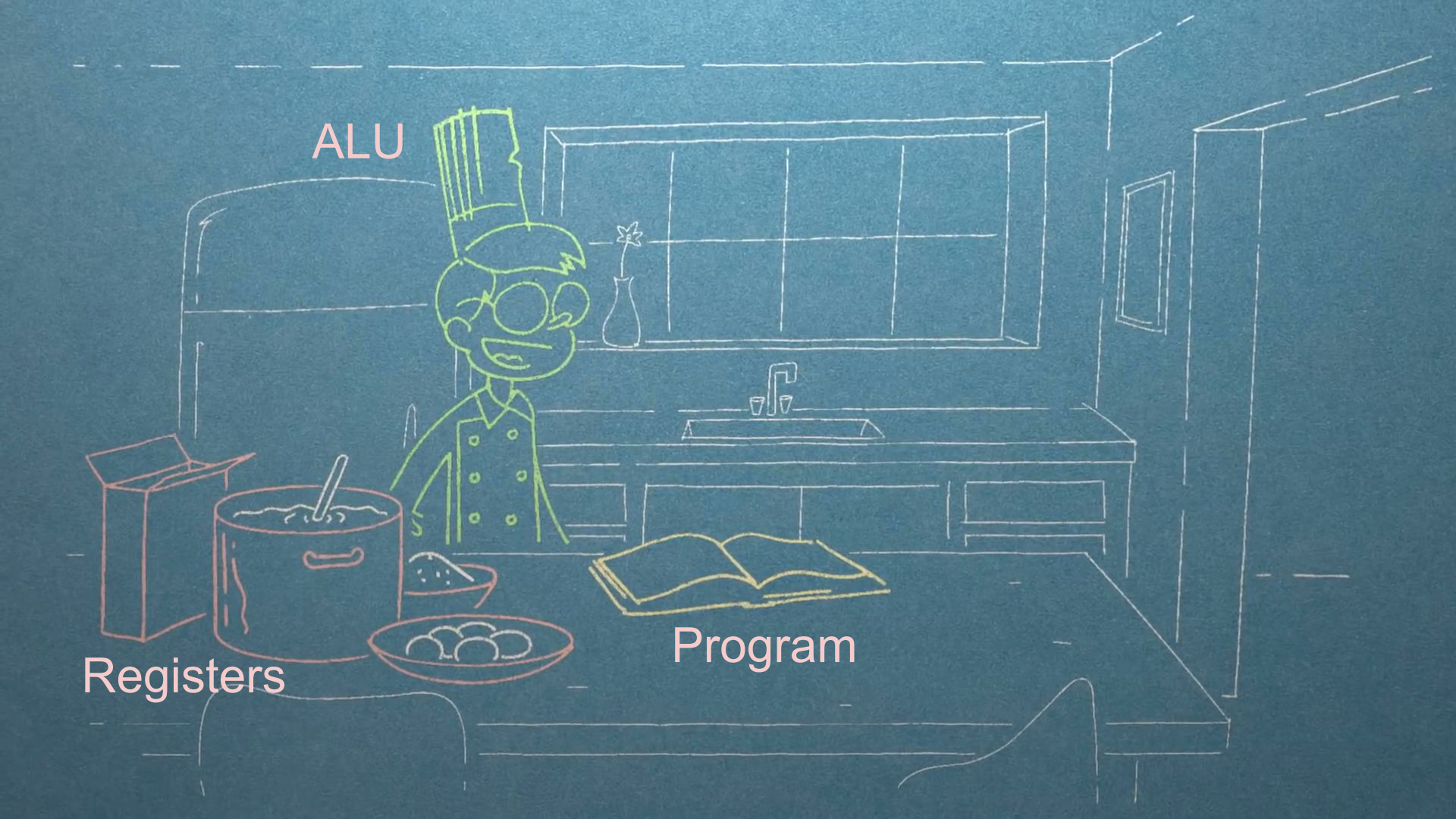
Hardware Hookup

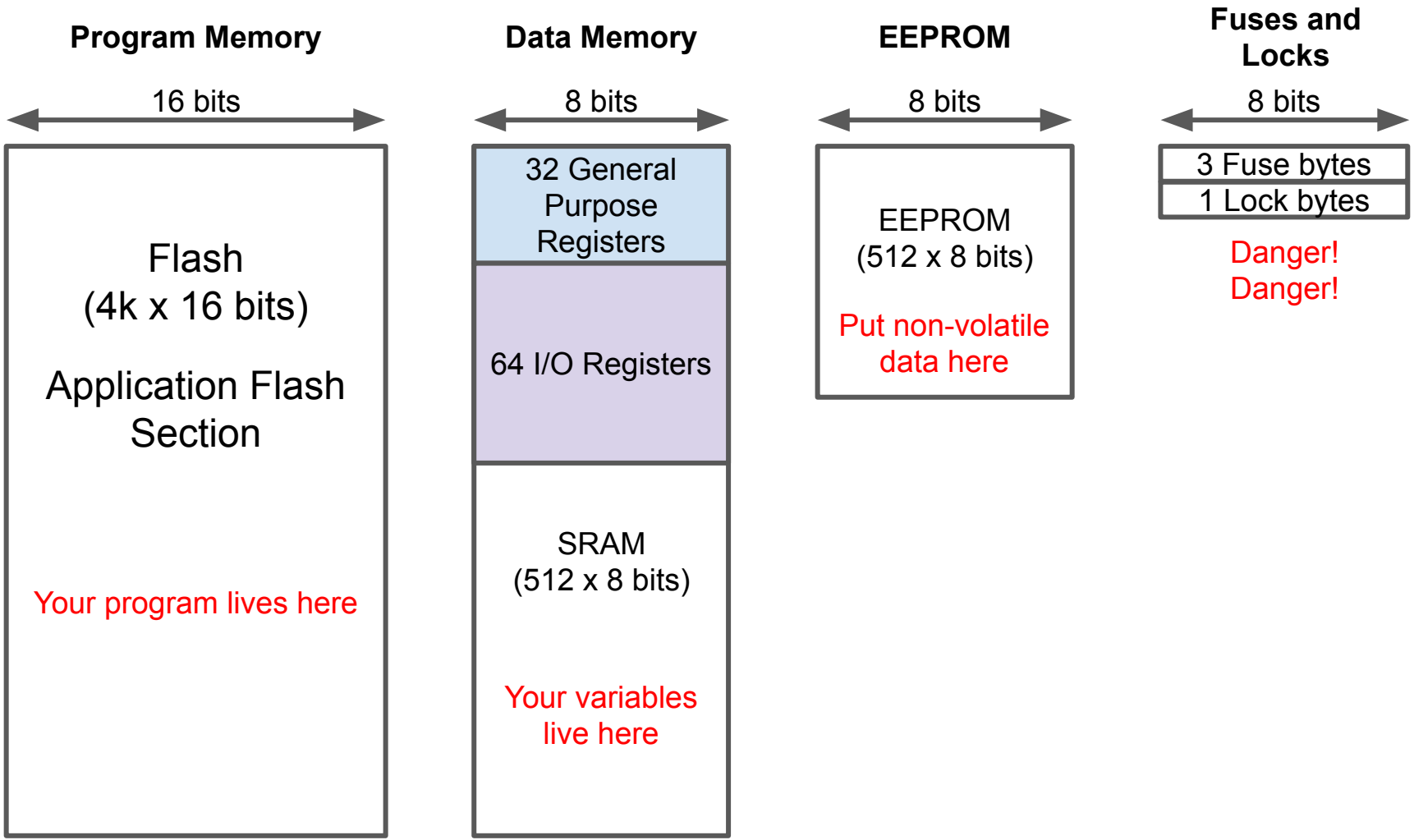


Hints

- Lots of information
- Type in exercises (at least open them from the repo)
- Try the challenges
- Have the datasheet open in another window


Data Bus 8-bit
Flash Program Memory
Program Counter
Status and Control
Interrupt Unit
32 x 8 General Purpose Registrers
Watchdog Timer
Instruction Register
Instruction Decoder
ADC
ALU
Analog Comparator
Control Lines
Direct Addressing
Indirect Addressing
Timer/Counter 0
Data SRAM
Timer/Counter 1
Universal Serial Interface
EEPROM
I/O Lines


ALU
Registers
Program


Program Memory
16 bits
Flash
(4k x 16 bits)
Application Flash Section
Your program lives here
Data Memory
8 bits
32 General Purpose Registers
64 I/O Registers
SRAM
(512 x 8 bits)
Your variables live here
EEPROM
8 bits
EEPROM
(512 x 8 bits)
Put non-volatile data here
Fuses and Locks
8 bits
3 Fuse bytes
1 Lock bytes
Danger!
Danger!

10.3 Register Description

From datasheet

10.3.1 MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	BODS	PUD	SE	SM1	SM0	BODSE	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 6 – PUD: Pull-up Disable**

When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn Registers are configured to enable the pull-ups ({DDxn, PORTxn} = 0b01). See “Configuring the Pin” on page 54 for more details about this feature.

10.3.2 PORTA – Port A Data Register

Step 2: Set PORT pins to 1 for logic high

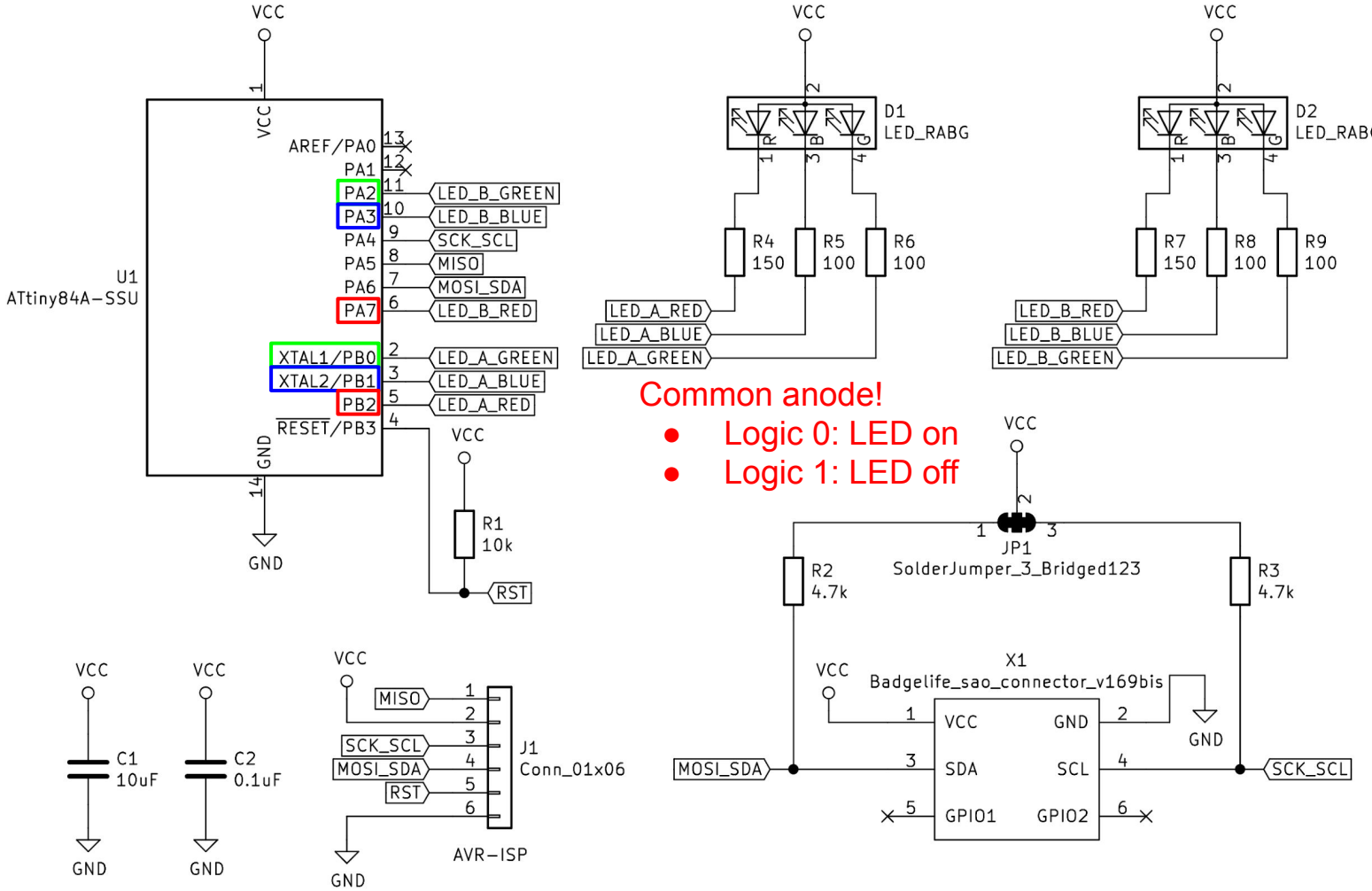
Use these names with avr/io.h

Bit	7	6	5	4	3	2	1	0	
0x1B (0x3B)	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

10.3.3 DDRA – Port A Data Direction Register

Step 1: Set DDR pins to 1 for output

Bit	7	6	5	4	3	2	1	0	
0x1A (0x3A)	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	


VCC
U1
ATtiny84A-SSU
AREF/PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
XTAL1/PB0
XTAL2/PB1
PB2
RESET/PB3
GND
LED_B_GREEN
LED_B_BLUE
SCK_SCL
MISO
MOSI_SDA
LED_B_RED
LED_A_GREEN
LED_A_BLUE
LED_A_RED
R1
10k
RST
D1
LED_RABG
R4
150
R5
100
R6
100
D2
R7
150
R8
100
R9
100
Common anode!
Logic 0: LED on
Logic 1: LED off
JP1
SolderJumper_3_Bridged123
R2
4.7k
R3
4.7k
X1
Badgelife_sao_connector_v169bis
C1
10uF
C2
0.1uF
J1
Conn_01x06
AVR-ISP

Exercise 01 - Blinky!

- Create AVR Projects folder somewhere on your computer
- In it, create blinky.c
- In blinky.c, enter the code on the right
- Save
- Open new terminal, enter commands:

```
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 from wenk-sao main directory:

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

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

Challenge: Only flash the green LED instead of all 3 LEDs

```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);
    }
}
```

Setting/Clearing Bits

```
PORTA = 0xFF;
```

Sets all bits in register PORTA to 1

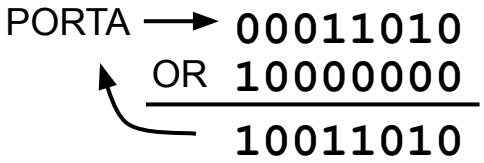
```
PORTA = 0b10000000;
```

Sets bit 7 to 1 and all others to 0

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

Sets bit 7 to 1 while keeping rest

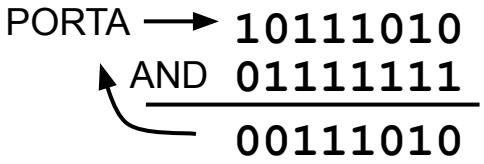
`(1 << 7) = b10000000`




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

Clears bit 7 to 0 while keeping rest

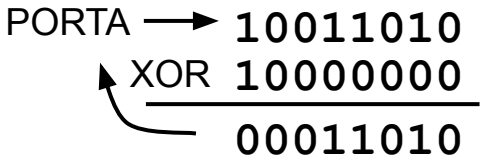
`~(1 << 7) = b01111111`



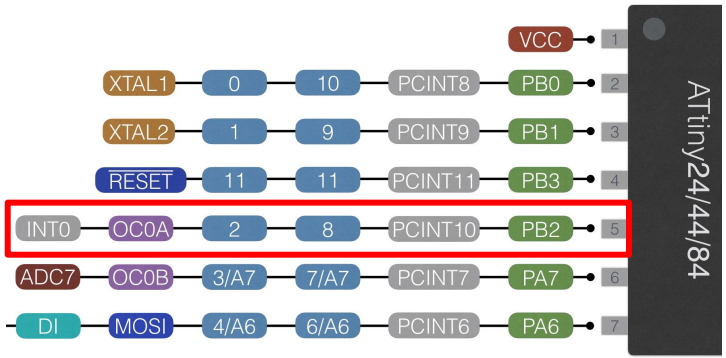

```
PORTA ^= (1 << 7);
```

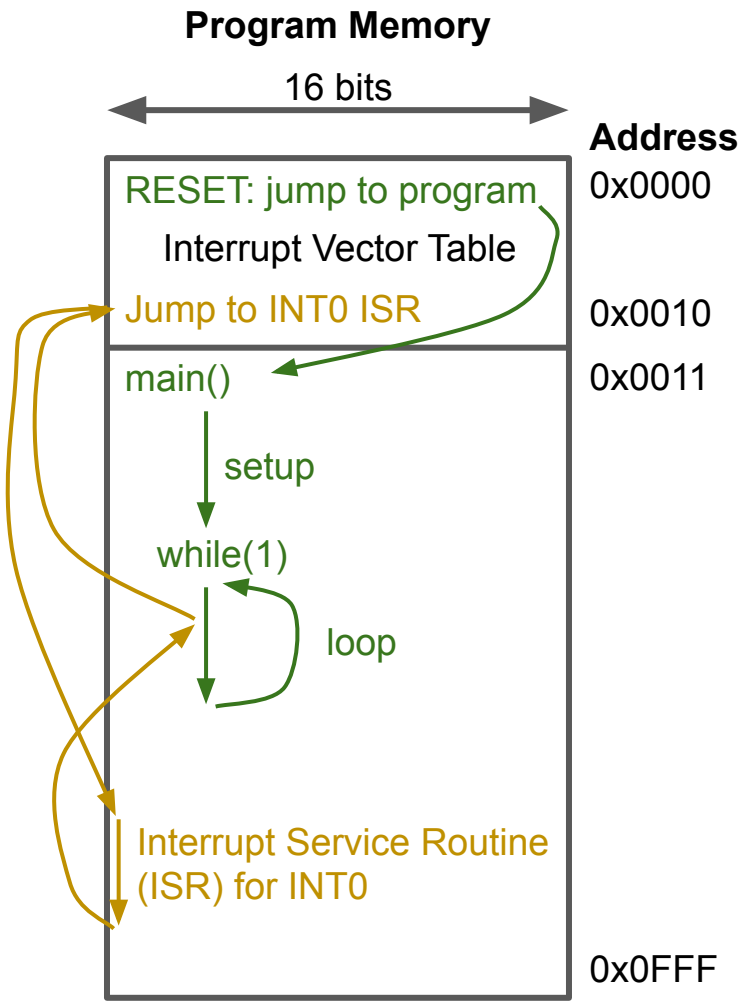
Toggles just bit 7

`(1 << 7) = b10000000`



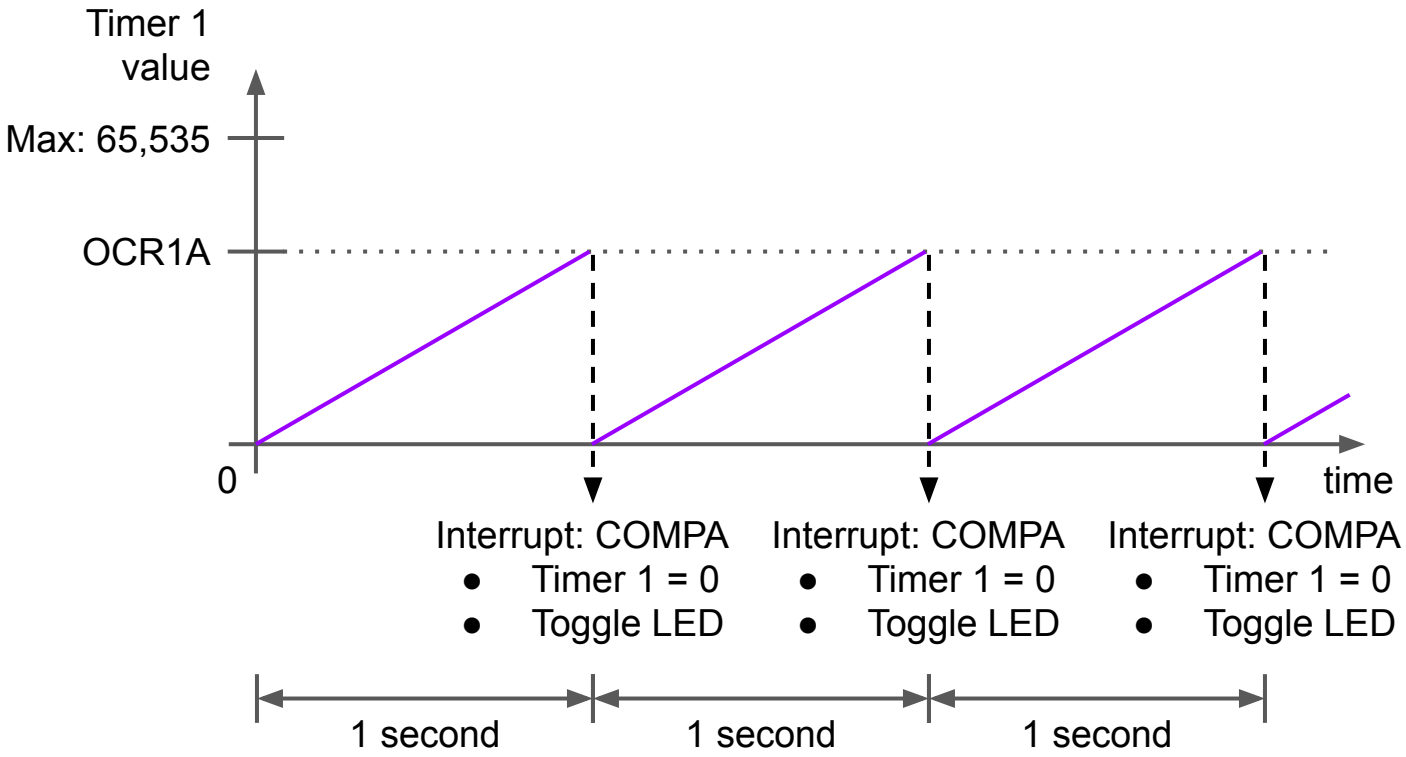
Intro to Interrupts





Intro to Timers

- Always counting, regardless of code
- Can be connected to pins, clocked by pins, used to trigger interrupts
- Timer 0: 8 bits, Timer 1: 16 bits



Timer Calculations

Period: $T = \frac{1}{f} = \frac{1}{1MHz} = 1\mu s$

Prescaler $\frac{8}{1MHz} = 8\mu s$

12.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
0x2E (0x4E)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 12-6. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Possible prescaler values

Try different prescaler values

Timer 1 (16 bit) max value: 65,535

$$\frac{1s \cdot 1MHz}{1} = 1,000,000$$

$$\frac{1s \cdot 1MHz}{8} = 125,000$$

$$\frac{1s \cdot 1MHz}{64} = 15,625$$

$$\frac{1s \cdot 1MHz}{256} = 3906.25$$

$$\frac{1s \cdot 1MHz}{1024} = 976.5625$$

How to Set Up Timer 1

1. Set WGM1 bits to choose Normal mode in TCCR1A
2. Set CS1 bits to choose prescaler in TCCR1B
3. Load reset value (0) into TCNT1 (Timer 1)
4. Load target value (15625) into OCR1A
5. Set OCIE1A bit in TIMSK1 to enable Timer 1 compare interrupt
6. Enable global interrupts

Important sections in the datasheet:

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

12.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
0x2F (0x4F)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 12-5. Waveform Generation Modes

Mode	WGM1 [3:0]	Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0000	Normal	0xFFFF	Immediate	MAX
1	0001	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0010	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0011	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0100	CTC (Clear Timer on Compare)	OCR1A	Immediate	MAX

12.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
0x2E (0x4E)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 12-6. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge

12.11.5 OCR1AH and OCR1AL – Output Compare Register 1 A

Bit	7	6	5	4	3	2	1	0	
0x2B (0x4B)	OCR1A[15:8]								OCR1AH
0x2A (0x4A)	OCR1A[7:0]								OCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

12.11.8 TIMSK1 – Timer/Counter Interrupt Mask Register 1

Bit	7	6	5	4	3	2	1	0	
0x0C (0x2C)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Exercise 02 - Blinky with Timer 1

Create timer-blinky.c with code below

```
$ 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
```

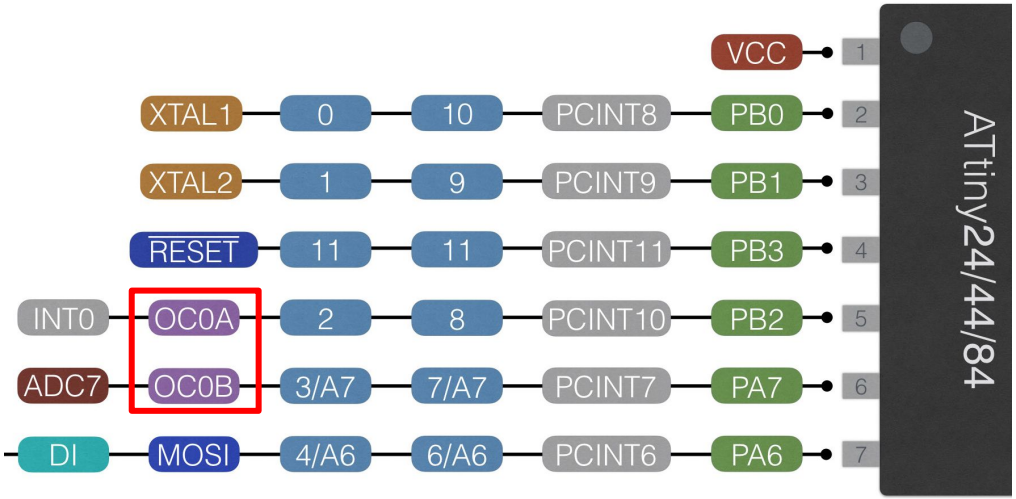
Or:

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

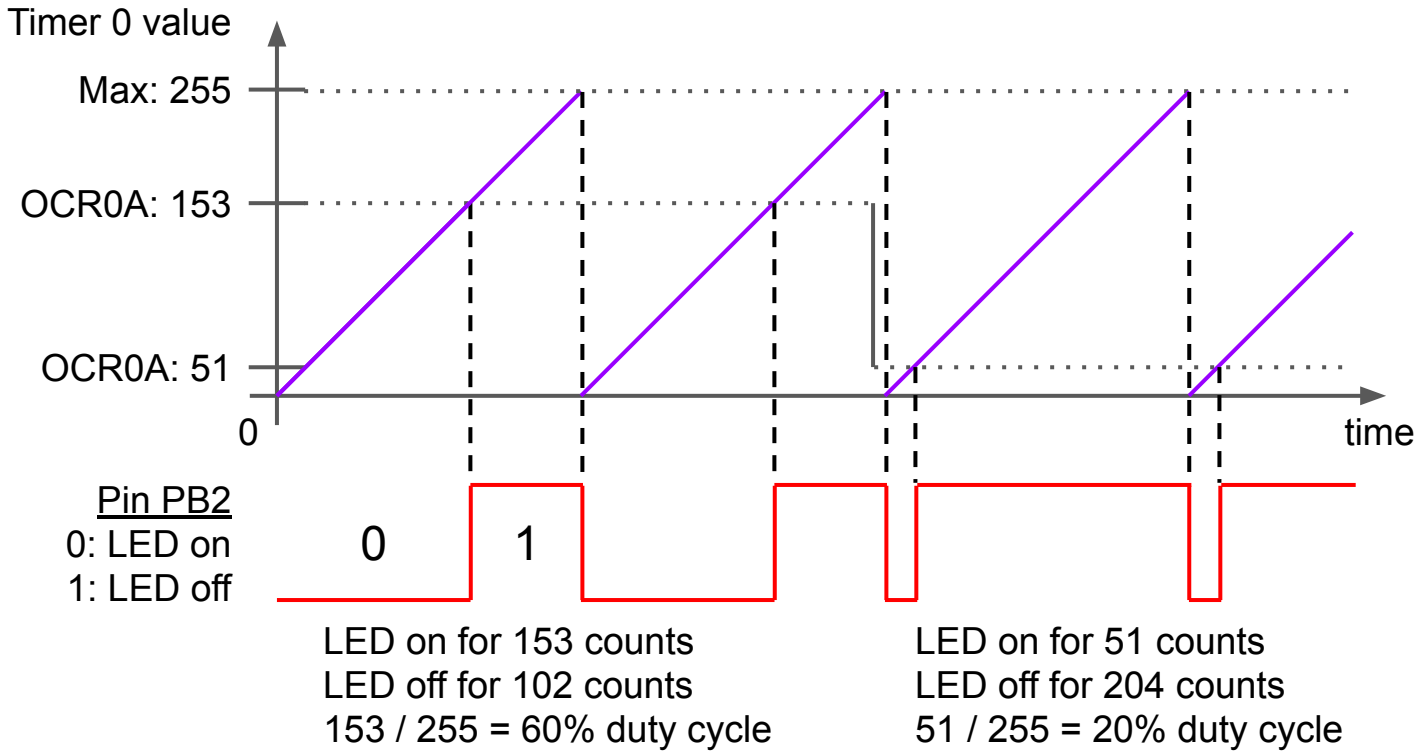
Challenge: Flash green LED with a delay of 500 ms instead of 1 second.

```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);
}
```

Intro to Pulse Width Modulation



Intro to Hardware PWM (Output Compare)



How to Set Up PWM

1. Set WGM0 bits in TCCR0A and TCCR0B
2. Set COM0A and COM0B bits in TCCR0A
3. Set CCS0 bits in TCCR0B for prescaler of 1
4. Set OCRA with desired duty cycle (0..255)

Important sections in the datasheet:

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

11.9.1 TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x30 (0x50)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 11-8. Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Table 11-3. Compare Output Mode, Fast PWM Mode(1)

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected
0	1	WGM02 = 0: Normal Port Operation, OC0A Disconnected WGM02 = 1: Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match Set OC0A at BOTTOM (non-inverting mode)
1	1	Set OC0A on Compare Match Clear OC0A at BOTTOM (inverting mode)

Exercise 03 - Hardware PWM

Create hardware-pwm.c with code to the right

```
$ 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:

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

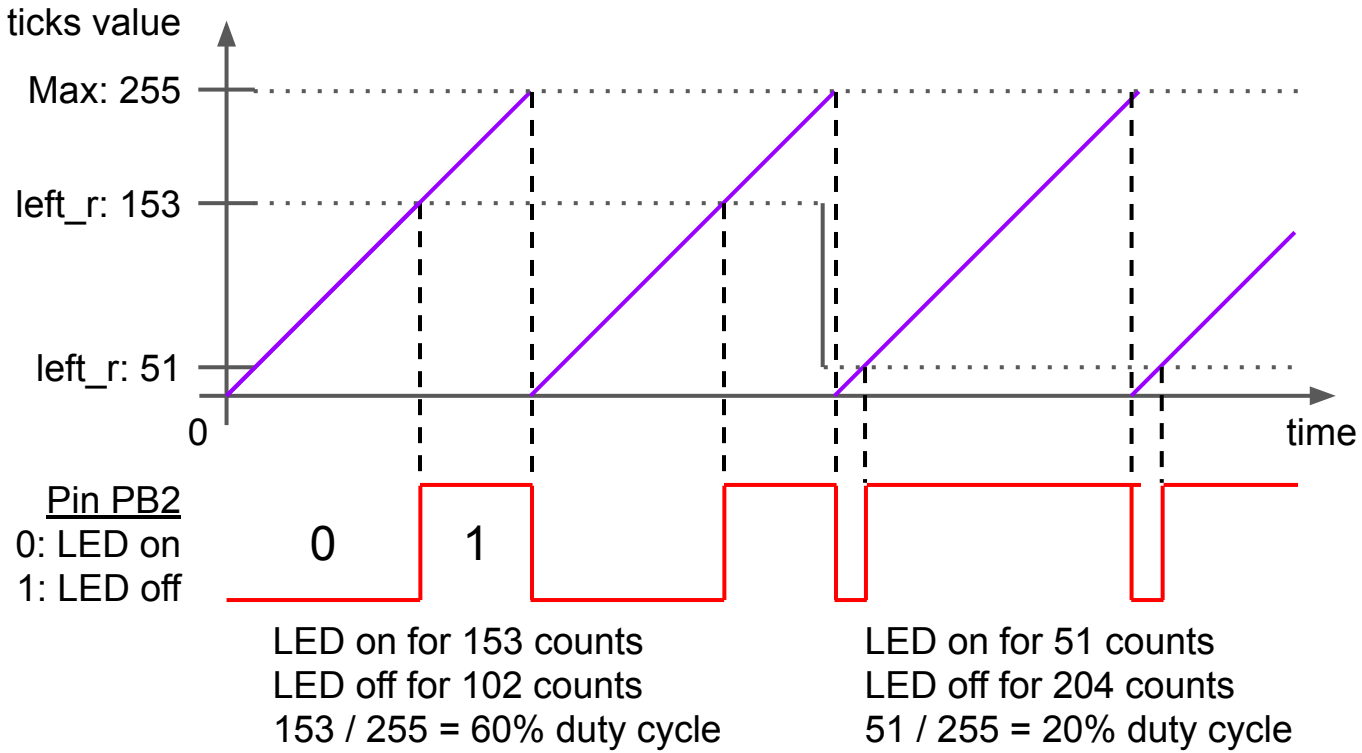
Challenge: Notice that a duty cycle of 0% does not completely turn off LED. This is because in Fast PWM mode, at least 1 clock cycle happens while pin is low (LED on). Fix this by using Phase Correct PWM.

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

```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 for A and B 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);
        }
    }
}
```

Software PWM

- Configure Timer 0 in normal mode
- Whenever Timer 0 reaches OCR0A, `ticks++`
- `ticks` is unsigned 8-bit
- Let `ticks` roll over from 255 to 0
- Continually compare `ticks` to desired value



PWM Frequency

CPU clock: 1 MHz

Timer 0 prescaler: 1

OCR0A (comp): ???

Ticks counting from 0 to 255 is one period

50 Hz LED refresh looks decent (and it's the best we can do)

$$\frac{1,000,000Hz}{255 \cdot comp} = 50Hz$$

$$comp \approx 78$$

Exercise 04 - Software PWM

Open `wenk-sao/Firmware/04-software-pwm/software-pwm.c`

Compile and upload it to the microcontroller.

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

Twitter: ShawnHymel

Instagram: shawn_hymel

LinkedIn: https://www.linkedin.com/in/ShawnHymel/


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

// Timer values
const uint8_t t0_load = 0;
const uint8_t t0_comp = 78; // ~50 Hz PWM

// LED pins (left = PORTB, right = PORTA)
const uint8_t p_lr = 2;
const uint8_t p_lg = 0;
const uint8_t p_lb = 1;
const uint8_t p_rr = 7;
const uint8_t p_rg = 2;
const uint8_t p_rb = 3;

// Tick counter
volatile uint8_t ticks = 0;

// Duty cycle per color
volatile uint8_t left_r = 0;
volatile uint8_t left_g = 0;
volatile uint8_t left_b = 0;
volatile uint8_t right_r = 0;
volatile uint8_t right_g = 0;
volatile uint8_t right_b = 0;

// Program entry point
int main(void) {

    // Make LED pins output
    DDRA = (1 << p_rr) | (1 << p_rg) | (1 << p_rb);
    DDRB = (1 << p_lr) | (1 << p_lg) | (1 << p_lb);

    // Set Timer 0 to normal operation
    TCCR0A = 0;
    TCCR0B = 0;

    // Set prescaler to 1
    TCCR0B |= (1 << CS00);

    // Reset Timer 0 and set compare values
    TCNT0 = t0_load;
    OCR0A = t0_comp;

    // Enable Timer 0 compare interrupt
    TIMSK0 = (1 << OCIE0A);

    // Enable global interrupts
    sei();

    // Infinite loop
    while(1) {

        // Set left eye: orange
        left_r = 255;
        left_g = 30;
        left_b = 0;

        // Set right eye: teal
        right_r = 0;
        right_g = 128;
        right_b = 128;
    }
}

// Interrupt service routine
ISR(TIM0_COMPA_vect) {

    // Reset Timer 0
    TCNT0 = t0_load;

    // Increment counter (just let roll over)
    ticks++;

    // Set left LED based on duty cycle
    if ( ticks < left_r ) {
        PORTB &= ~(1 << p_lr); // LED on
    } else {
        PORTB |= (1 << p_lr);  // LED off
    }
    if ( ticks < left_g ) {
        PORTB &= ~(1 << p_lg); // LED on
    } else {
        PORTB |= (1 << p_lg);  // LED off
    }
    if ( ticks < left_b ) {
        PORTB &= ~(1 << p_lb); // LED on
    } else {
        PORTB |= (1 << p_lb);  // LED off
    }

    // Set right LED based on duty cycle
    if ( ticks < right_r ) {
        PORTA &= ~(1 << p_rr); // LED on
    } else {
        PORTA |= (1 << p_rr);  // LED off
    }
    if ( ticks < right_g ) {
        PORTA &= ~(1 << p_rg); // LED on
    } else {
        PORTA |= (1 << p_rg);  // LED off
    }
    if ( ticks < right_b ) {
        PORTA &= ~(1 << p_rb); // LED on
    } else {
        PORTA |= (1 << p_rb);  // LED off
    }
}
```