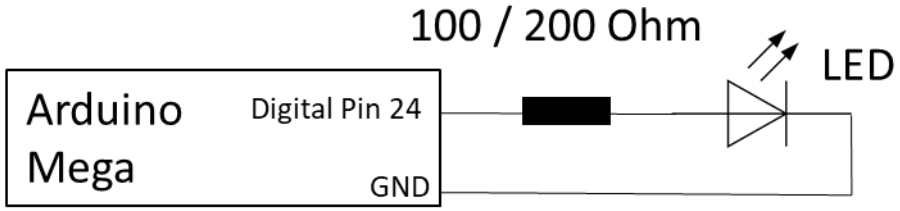
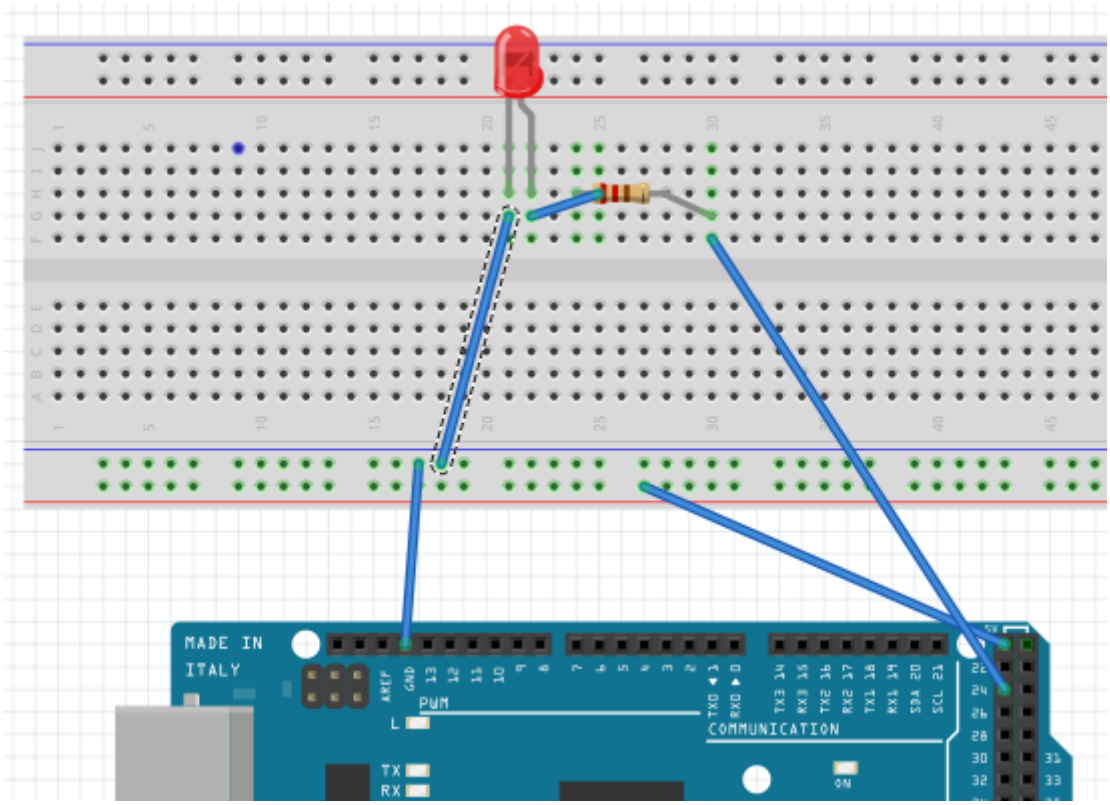
Lösung 04: Timer

Aufgabe 1: Vorbereitung

Achten Sie unbedingt darauf, dass die LED nicht einfach über einen Stromkreis von +5V zu Masse angeschlossen ist, sondern über den Digital Port 24 angesteuert wird. Schaltung und Verkabelung, siehe unten.
Digital Pin 24 entspricht PA2, also dem Pin Nr. 2 (3. Pin) des Ports A. Dieses Wissen ist später wichtig, wenn wir Interrupts über die AVR-Libc direkt über Register programmieren.





Aufgabe 2: LED Blinkfrequenz mit Prescaler

a) Dateiblatt Seite 164: Kein Prescaling, 1/8, 1/64, 1/256 und 1/1024.

b) Da bei jedem Overflow die LED wechselweise an- bzw. ausgeschaltet wird, entspricht eine „Blinkperiode“ der Zeitdauer bis **2** Overflow-Interrupts eingetreten sind. Man muss also die Frequenz nochmal durch 2 teilen. Nach 2^{16} Ticks findet bei einem 16-Bit Zähler jeweils ein Overflow statt. Es ergeben sich die folgenden Blinkfrequenzen:

- Kein Prescaler: $f = 16\,MHz \div (2^{16} * \mathbf{2}) = 122\,Hz$
- Clk/8-Prescaler: $f = 16\,MHz \div (2^{16} * 2 * 8) = 15\,Hz$

- Clk/64-Prescaler: $f = 16\,MHz \div (2^{16} * 2 * 64) = 1{,}9\,Hz$
- Clk/256-Prescaler: $f = 16\,MHz \div (2^{16} * 2 * 256) = 0{,}48\,Hz$
- Clk/1024-Prescaler: $f = 16\,MHz \div (2^{16} * 2 * 1024) = 0{,}12\,Hz$

c) Siehe Source Code! Achtung: Stellt man keinen Prescaler ein, so blinkt die LED so schnell, dass das menschliche Auge das gar nicht wahrnehmen kann.

```
#include <Arduino.h>

// Aufgabe 2
void setup()
{
	DDRA |= (1 << DDA2);

	// initialize timer-related registers
	TCCR4A = 0;
	TCCR4B = 0;
	TCNT4  = 0;

	//TCCR4B |= (1 << CS42) | (1 << CS40);   // prescaler 1024
	TCCR4B |= (1 << CS41) | (1 << CS40);     // prescaler 64
	//TCCR4B |= (1 << CS41);                 // prescaler 8
	//TCCR4B |= (1 << CS40);                 // no prescaler
	TIMSK4 |= (1 << TOIE4);                  // enable timer overflow interrupt

	sei();                                   // enable all interrupts
}


ISR(TIMER4_OVF_vect)      // ISR, called when timer overflow occurs
{
	PINA |= (1 << PINA2);     // trick for toggling LED, see manual p68
	// Alternative XOR for toggling:       PORTA ^= (1 << PA2);
}

void loop() {

}
```

d) Falls gar kein Prescaler verwendet wird, leuchtet die LED dauerhaft, wenn auch vielleicht etwas schwächer. Der Grund liegt in der Trägheit der LED bzw. das menschliche Auge könnte solche hohen Frequenzen überhaupt nicht wahrnehmen. Beim größten Prescaler ist die Blinkfrequenz sehr langsam.

Aufgabe 3: LED Blinkfrequenz ohne Prescaler

a) Alle $2^{16} \div 16\,MHz = 4ms$ tritt ein Overflow auf. Bzw. in 1 Sekunde treten $16MHz \div 2^{16} = 244{,}1 \approx 244$ Overflow-Ereignisse ein. In 2 Sekunden also ca. 488 Overflow-Ereignisse.

b) Siehe Programmcode! Zu beachten ist, dass für eine Frequenz von 0,5 Hz, jede Sekunde der Zustand der LED umgeschaltet werden muss. Man muss also nur bis 244-Overflows abwarten, bevor man die LED umschaltet / toggelt.

Sollte es bei einem Mikrocontroller keine Prescaler geben, kann man sich auf die folgende Weise behelfen:

```
volatile unsigned int overflow_counter = 0;
// counts number of overflows since last second

void setup()
{
	DDRA |= (1 << DDA2);

	// initialize timer4
	TCCR4A = 0;
	TCCR4B = 0;
	TCNT4  = 0;
	TCCR4B |= (1 << CS40);    // activate clock, but don't use a prescaler
	TIMSK4 |= (1 << TOIE4);   // enable timer overflow interrupt

	sei();                   // enable all interrupts
}


ISR(TIMER4_OVF_vect)            // ISR, called when timer overflow occurs
{
	if (overflow_counter == 244) {
	// 16 MHz / 2^16 = 244,1 -> during 1 second approx. 244 overflow events
		overflow_counter = 0;
		PINA |= (1 << PINA2);           // toggle LED
	}
	overflow_counter++;
}

void loop() {
	// do nothing, endless loop
}
```

Achtung: `Serial.println()` Statements kosten sehr viel Zeit. Verwendet man ein solches Statement z.B. in der ISR (was ohnehin nicht zu empfehlen ist), kann das zur Verlängerung der gewünschten Blinkperiode führen.

Aufgabe 4: LED Blinkfrequenz mit Output Compare

a) Der Zustand der LED muss jede Sekunde geändert werden (An- und Ausschalten). Bei einem clk/256 Prescaler hat der Zähler innerhalb einer Sekunde bis zum Wert $16000000 \div 256 = 62500$ hochgezählt. Der Zähler soll also von 0 bis 62500 zählen und dann per Interrupt (vorzeitig) auf 0 zurückgesetzt werden. Dazu muss der Wert 62500 in `OCR4A` konfiguriert werden.

b) Hinweis: Auch die folgende Methode verwenden einen Interrupt, jedoch statt eines Overflow-Interrupts einen Output-Compare Interrupt. Alle 3 bisher vorgestellten Verfahren funktionieren. Es ist mehr oder weniger Geschmackssache.

```
void setup()
{
	DDRA |= (1 << DDA2);

	// initialize timer4
	TCCR4A = 0;
	TCCR4B = 0;
```

```
      TCNT4  = 0;

      OCR4A = 62500;            // compare match register 16Mhz/256
      TCCR4B |= (1 << WGM42);   // CTC mode
      TCCR4B |= (1 << CS42);    // 256 prescaler
      TIMSK4 |= (1 << OCIE4A);  // enable timer compare interrupt
      sei();                    // enable all interrupts
}

ISR(TIMER4_COMPA_vect)          // timer compare interrupt service routine
{
      PINA |= (1 << PINA2);     // toggle LED
}

void loop() {
}
```