Lab 4: Gregor Karetka

Link to your `Digital-electronics-2` GitHub repository:

https://github.com/gkaretka/Digital-electronics-2

Preparation tasks (done before the lab at home)

Consider an n-bit number that we increment based on the clock signal. If we reach its maximum value and try to increase it, the value will be reset. We call this state an **overflow**. The overflow time depends on the frequency of the clock signal, the number of bits, and on the prescaler value:

Calculation for 16 MHz, 8 bit, /1 prescaler

$$t_{ovf} = \frac{1}{f_{CPU}} \cdot 2^n \cdot N = \frac{1}{16\ \mathrm{MHz}} \cdot 2^8 \cdot 1 = 16\ \mathrm{us}$$

Calculation for 16 MHz, 16 bit, /1 prescaler

$$t_{ovf} = \frac{1}{f_{CPU}} \cdot 2^n \cdot N = \frac{1}{16\ \mathrm{MHz}} \cdot 2^{16} \cdot 1 = 4.096\ \mathrm{ms}$$

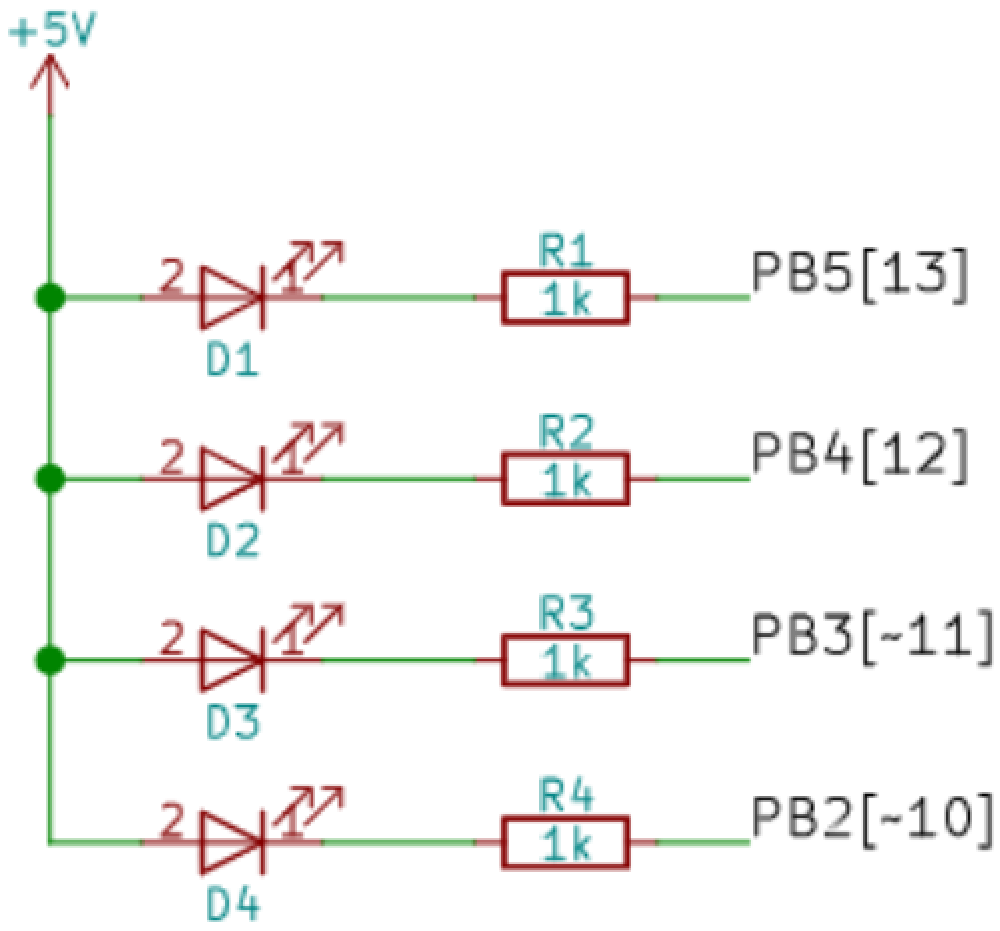
Overflow times

1. Complete table with overflow times.

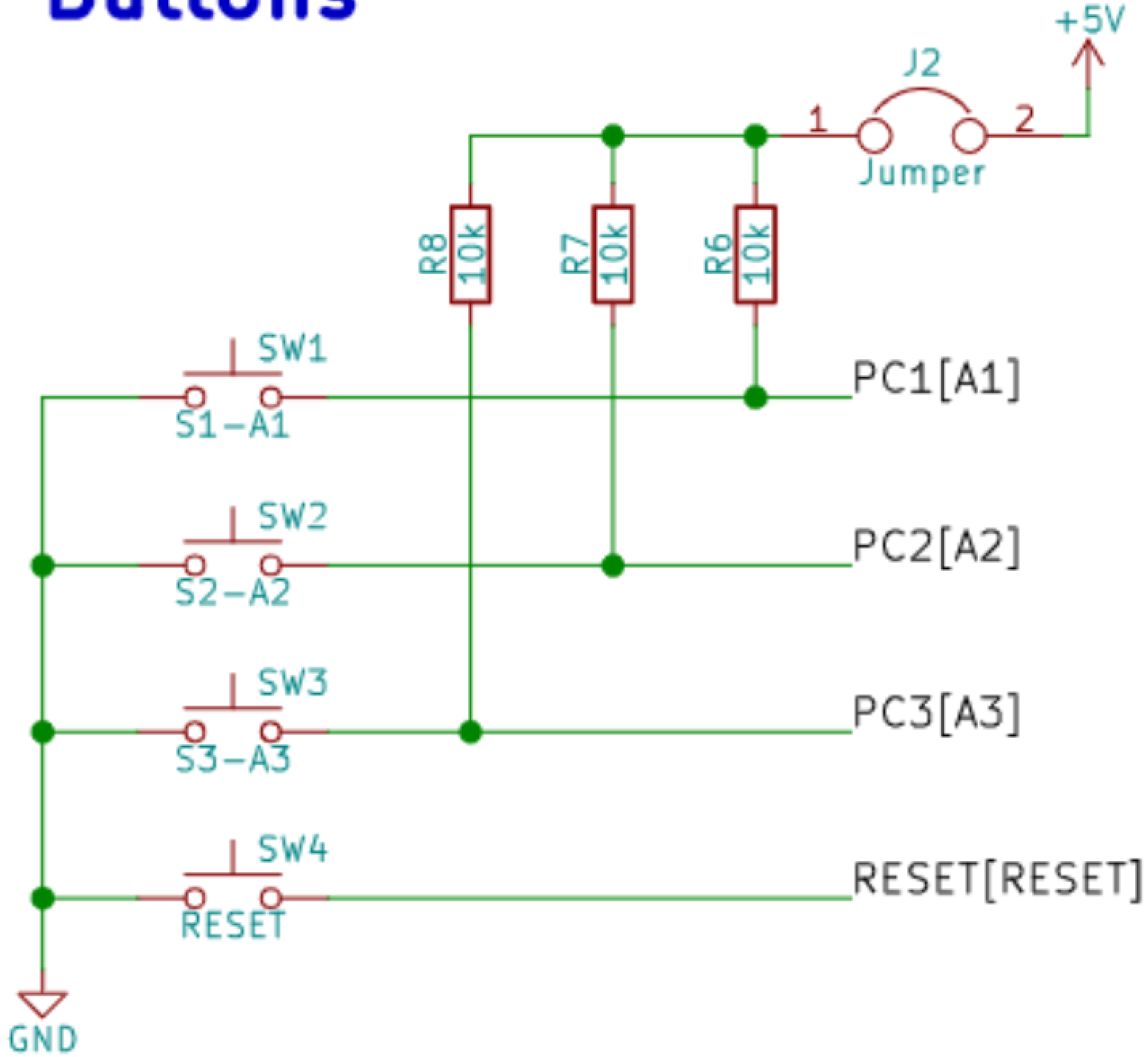
Module	Number of bits	1	8	32	64	128	256	1024
Timer/Counter0	8	16 us	128 us	--	1024 us	--	4.096 ms	16.384 ms
Timer/Counter1	16	4.096 ms	32.768 ms	--	262.144 ms	--	1.048576 s	4.194304 s
Timer/Counter2	8	16 us	128 us	512 us	1024 us	2048 us	4.096 ms	16.384 ms

2. Shields are boards that can be attached to an Arduino board, significantly expand its capabilities, and makes prototyping much faster. See schematic of Multi-function shield and find out the connection of four LEDs (D1, D2, D3, D4) and three push buttons (S1-A1, S2-A2, S3-A3).

LEDs



Buttons


+5V
J2
1
2
Jumper
R8
10k
R7
10k
R6
10k
SW1
S1-A1
PC1[A1]
SW2
S2-A2
PC2[A2]
SW3
S3-A3
PC3[A3]
SW4
RESET
RESET[RESET]
GND

Part 2: Timers

A timer (or counter) is a hardware block within an MCU and can be used to measure time events. ATmega328P has three timers, called:

- Timer/Counter0,
- Timer/Counter1, and
- Timer/Counter2.

T/C0 and T/C2 are 8-bit timers, where T/C1 is a 16-bit timer. The counter counts in synchronization with microcontroller clock from 0 up to 255 (for 8-bit counter) or 65,535 (for 16-bit). Different clock sources can be selected for each timer using a CPU frequency divider with fixed prescaler values, such as 8, 64, 256, 1024, and others.

The timer modules can be configured with several special purpose registers. According to the ATmega328P datasheet (eg in the **8-bit Timer/Counter0 with PWM > Register Description** section), which I/O registers and which bits configure the timer operations?

Module	**Operation**	**I/O register(s)**	**Bit(s)**
Timer/Counter0	Prescaler 8-bit data value Overflow interrupt enable	TCCR0B TCNT0 TIMSK0	CS02 CS01 CS00 (000: stopped, 001: 1, 010: 8, 011: 64, 100: 256, 101: 1024) TCNT0[7:0] TOIE0 (timer enable; 1: enable, 0: disable)

Module	Operation	I/O register(s)	Bit(s)
Timer/Counter1	Prescaler 16-bit data value Overflow interrupt enable	TCCR1B TCNT1H, TCNT1L TIMSK1	CS12, CS11, CS10 (000: stopped, 001: 1, 010: 8, 011: 64, 100: 256, 101: 1024) TCNT1[15:0] TOIE1 (1: enable, 0: disable)
Timer/Counter2	Prescaler 8-bit data value Overflow interrupt enable	TCCR2B TCNT2 TIMSK2	CS22 CS21 CS20 (000: stopped, 001: 1, 010: 8, 011: 32, 100: 64, 101: 128, 110: 256, 111: 1024) TCNT2[7:0] TOIE2 (timer enable; 1: enable, 0: disable)

Part 3: Polling and Interrupts

The state of continuous monitoring of any parameter is called **polling**. The microcontroller keeps checking the status of other devices; and while doing so, it does no other operation and consumes all its processing time for monitoring [3].

Interrupts can be established for events such as a counter's value, a pin changing state, serial communication receiving of information, or the Analog to Digital Converted has finished the conversion process.

See the ATmega328P datasheet (section **Interrupts > Interrupt Vectors in ATmega328 and ATmega328P**) for sources of interruptions that can occur on ATmega328P. Complete the selected interrupt sources in the following table. The names of the interrupt vectors in C can be found in C library manual.

Table 11-1. Reset and Interrupt Vectors in ATmega328P

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x0012	TIMER2 OVF	Timer/Counter2 overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x0016	TIMER1 COMPA	Timer/Counter1 compare match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 compare match B
14	0x001A	TIMER1 OVF	Timer/Counter1 overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 compare match B
17	0x0020	TIMER0 OVF	Timer/Counter0 overflow
18	0x0022	SPI, STC	SPI serial transfer complete
19	0x0024	USART, RX	USART Rx complete
20	0x0026	USART, UDRE	USART, data register empty
21	0x0028	USART, TX	USART, Tx complete
22	0x002A	ADC	ADC conversion complete
23	0x002C	EE READY	EEPROM ready

24	0x002E	ANALOG COMP	Analog comparator
25	0x0030	TWI	2-wire serial interface
26	0x0032	SPM READY	Store program memory ready

Timer library

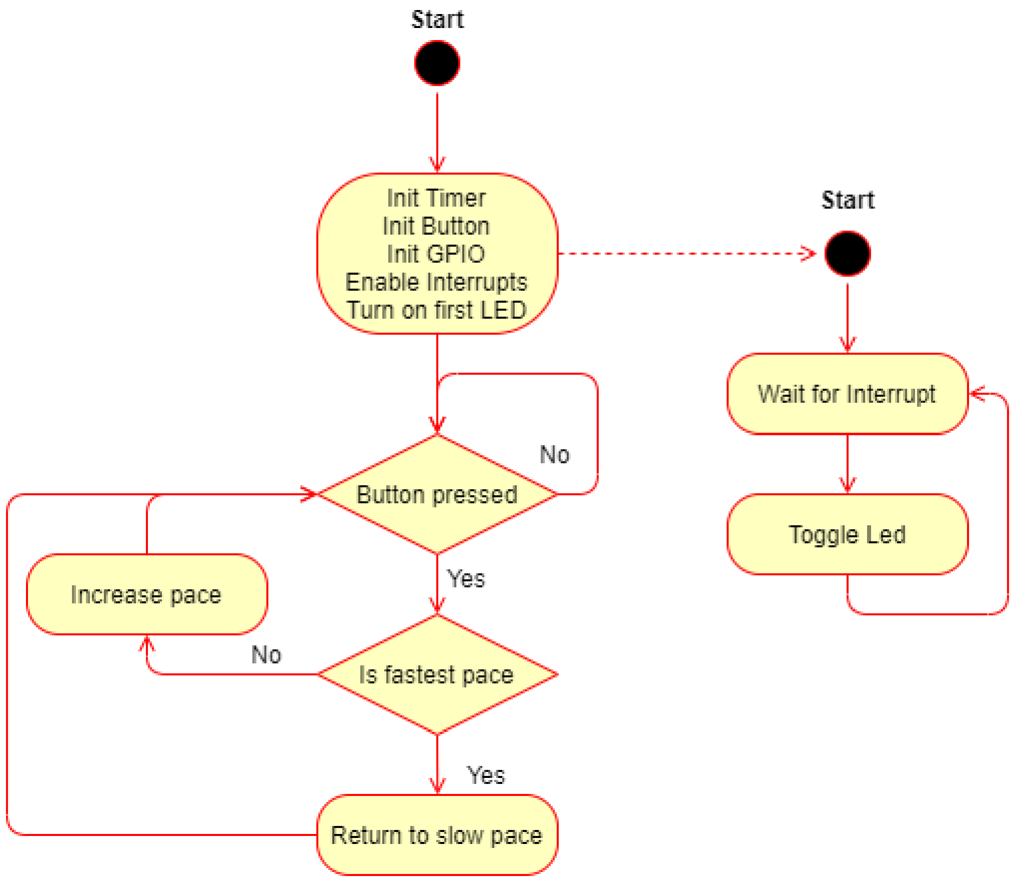
1. In your words, describe the difference between common C function and interrupt service routine.
   - Function - generic block of code we want to reuse. This block can be called from anywhere in our program.
   - Interrupt service routine - function to which we jump when interrupt occures, usually short. Main program is halted and after ISR is executed resumed from the same point it was halted in

2. Part of the header file listing with syntax highlighting, which defines settings for Timer/Counter0:

```c
/**
 * @name  Definitions of Timer/Counter0
 * @note  F_CPU = 16 MHz
 */
/**
 * @name  Definitions for 8-bit Timer/Counter0
 * @note  t_OVF = 1/F_CPU * prescaler * 2^n where n = 8, F_CPU = 16 MHz
 */
#define TIM0_stop()                 TCCR0B &= ~((1<<CS02) | (1<<CS01) | (1<<CS00));
/** @brief Set overflow 16us, prescaler 001 --> 1 */
#define TIM0_overflow_16us()        TCCR0B &= ~((1<<CS02) | (1<<CS01)); TCCR0B |= (1<<CS00);
/** @brief Set overflow 128us, prescaler 010 --> 8 */
#define TIM0_overflow_128us()       TCCR0B &= ~((1<<CS02) | (1<<CS00)); TCCR0B |= (1<<CS01);
/** @brief Set overflow 1024 us, prescaler 011 --> 64 */
#define TIM0_overflow_1024us()      TCCR0B &= ~(1<<CS02); TCCR0B |= (1<<CS01) | (1<<CS00);
/** @brief Set overflow 4096us, prescaler 100 --> 256 */
#define TIM0_overflow_4096us()      TCCR0B &= ~((1<<CS01) | (1<<CS00)); TCCR0B |= (1<<CS02);
/** @brief Set overflow 16384 us, prescaler // 101 --> 1024 */
#define TIM0_overflow_16384us()     TCCR0B &= ~(1<<CS01); TCCR0B |= (1<<CS02) | (1<<CS00);
```

```c
/** @brief Enable overflow interrupt, 1 --> enable */
#define TIM0_overflow_interrupt_enable()    TIMSK0 |= (1<<TOIE0);
/** @brief Disable overflow interrupt, 0 --> disable */
#define TIM0_overflow_interrupt_disable()   TIMSK0 &= ~(1<<TOIE0);
```

3. Flowchart figure for function `main()` and interrupt service routine `ISR(TIMER1_OVF_vect)` of application that ensures the flashing of one LED in the timer interruption. When the button is pressed, the blinking is faster, when the button is released, it is slower. Use only a timer overflow and not a delay library.




```c
static inline void knight_rider();
```

```c
/* Function definitions ----------------------------------------------*/
/**********************************************************************
 * Function: Main function where the program execution begins
 * Purpose:  Toggle one LED on the Multi-function shield using
 *           the internal 8- or 16-bit Timer/Counter.
 * Returns:  none
 **********************************************************************/
int main(void)
{
    // Configuration of LED(s) at port B - active low
    GPIO_config_output(&DDRB, LED_D1);
    GPIO_write_high(&PORTB, LED_D1); // high

    GPIO_config_input_pullup(&DDRC, BUTTON);

    // Configuration of 16-bit Timer/Counter1 for LED blinking
    // Set the overflow prescaler to 262 ms and enable interrupt
    TIM1_overflow_262ms();
    TIM1_overflow_interrupt_enable();

    // Enables interrupts by setting the global interrupt mask
    sei();

    enum speed_states {SPEED_4MS = 0, SPEED_33MS = 1, SPEED_262MS = 2, SPEED_1S = 3, SPEED_4S = 4};
    uint8_t speed_state = SPEED_262MS;

    // Infinite loop
    while (1)
    {
        /* Empty loop. All subsequent operations are performed exclusively
         * inside interrupt service routines ISRs */

        if (GPIO_read(&PINC, BUTTON) == 0) {
            switch(speed_state) {
                case (SPEED_4S):
                    speed_state = SPEED_1S;
                    TIM1_overflow_1s();
```

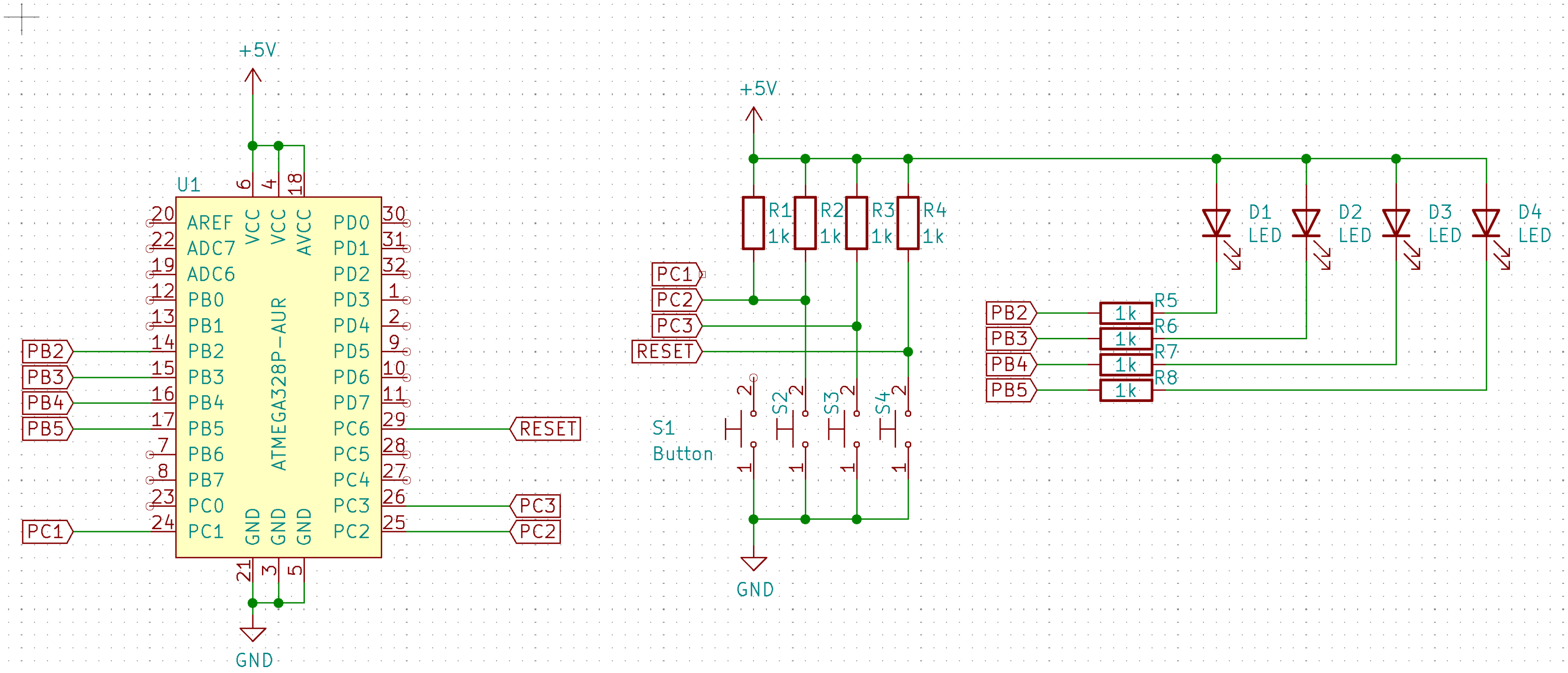
```c
                break;

            case (SPEED_1S):
                speed_state = SPEED_262MS;
                TIM1_overflow_262ms();
                break;

            case (SPEED_262MS):
                speed_state = SPEED_33MS;
                TIM1_overflow_33ms();
                break;

            case (SPEED_33MS):
                speed_state = SPEED_4MS;
                TIM1_overflow_4ms();
                break;

            case (SPEED_4MS):
                speed_state = SPEED_4S;
                TIM1_overflow_4s();
                break;

            default:
                speed_state = SPEED_4S;
                TIM1_overflow_4s();
                break;
            }

            _delay_ms(500);
        }
    }

    // Will never reach this
    return 0;
}

/* Interrupt service routines ----------------------------------------*/
```

```c
/**********************************************************************
 * Function: Timer/Counter1 overflow interrupt
 * Purpose:  Toggle D1 LED on Multi-function shield.
 **********************************************************************/
ISR(TIMER1_OVF_vect)
{
   GPIO_toggle(&PORTB, LED_D1);
}
```

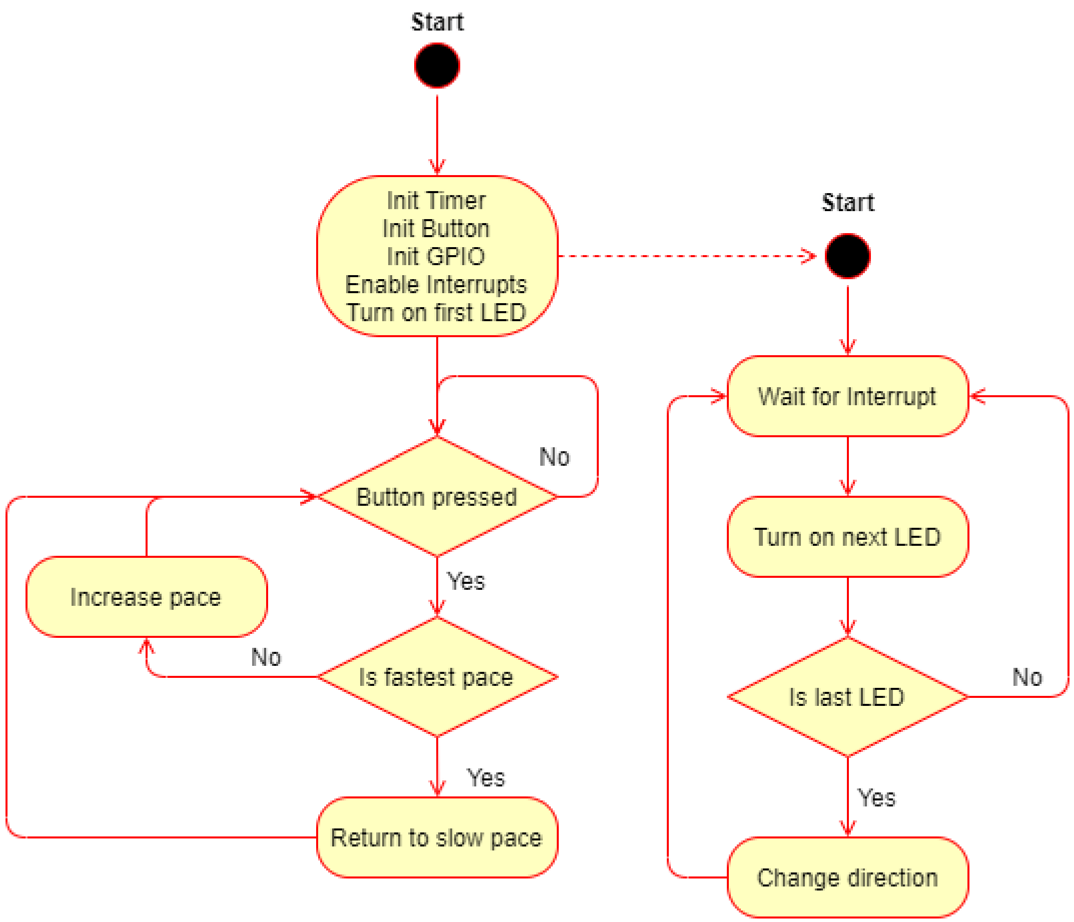
Knight Rider

1. Scheme of Knight Rider application with four LEDs and a push button, connected according to Multi-function shield. Connect AVR device, LEDs, resistors, push button, and supply voltage. The image can be drawn on a computer or by hand. Always name all components and their values!

Schematic



Flow diagram


Start
Init Timer
Init Button
Init GPIO
Enable Interrupts
Turn on first LED
Start
Wait for Interrupt
No
Button pressed
Turn on next LED
Increase pace
Yes
No
Is fastest pace
Is last LED
No
Yes
Yes
Return to slow pace
Change direction