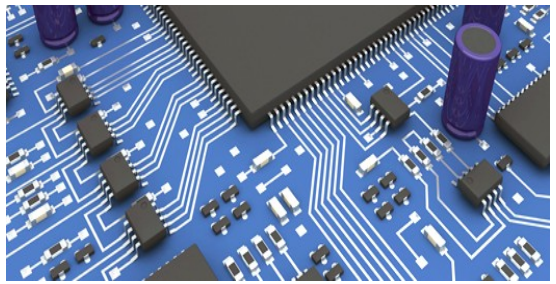
Aplikace Embedded systémů v Mechatronice



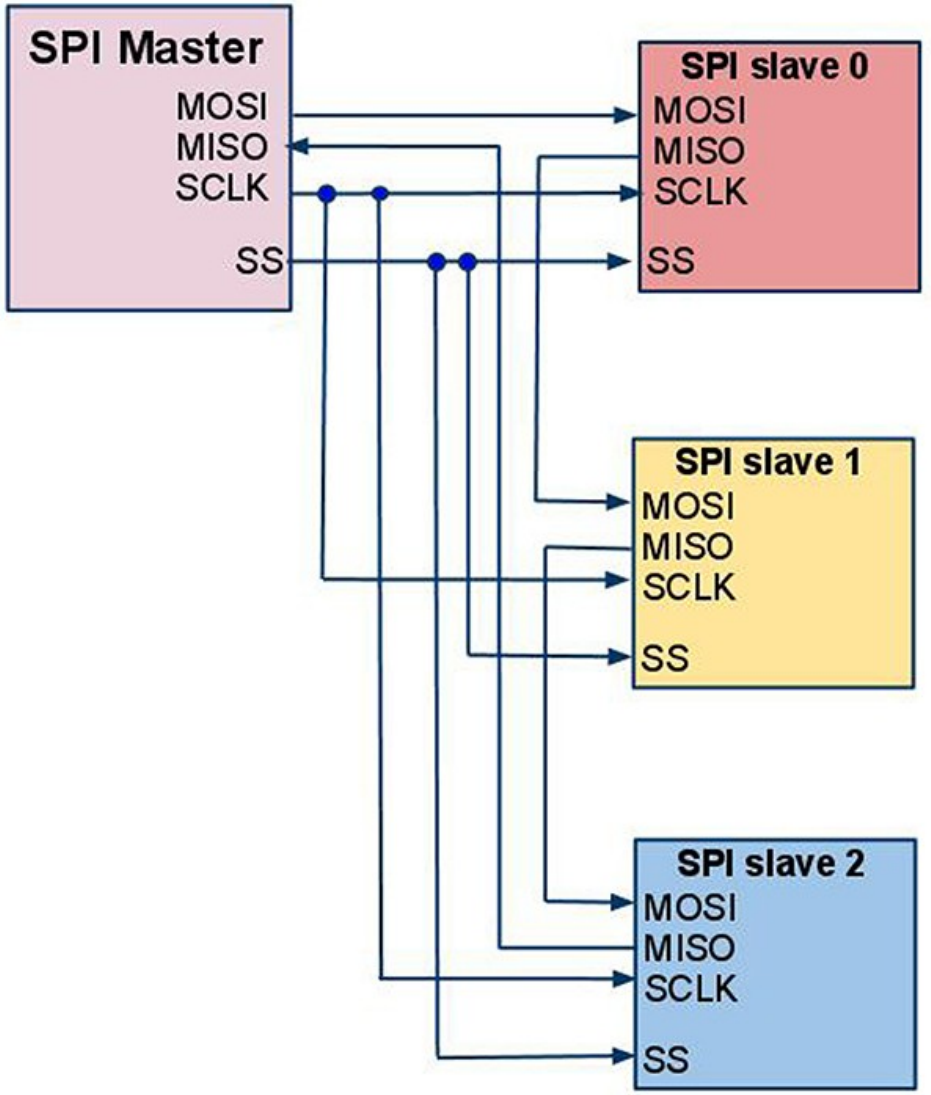
MPLAB® X IDE

Michal Bastl

SPI

Serial pheripheral interface:

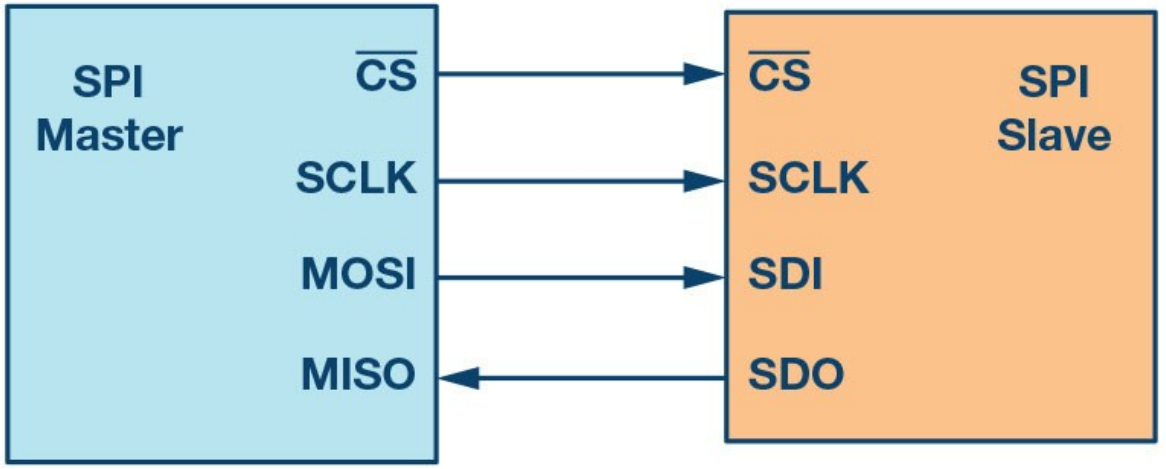
- Sběrnice v embedded systémech
- Topologicky jednoduchá synchronní (má sdílený CLK)
- Master-Slave
- Typicky komunikace v rámci DPS
- Paměti, ADC, DAC, SD karty atd.
- Rychlá (desítky MHz běžně)



SPI

Serial pheripheral interface:

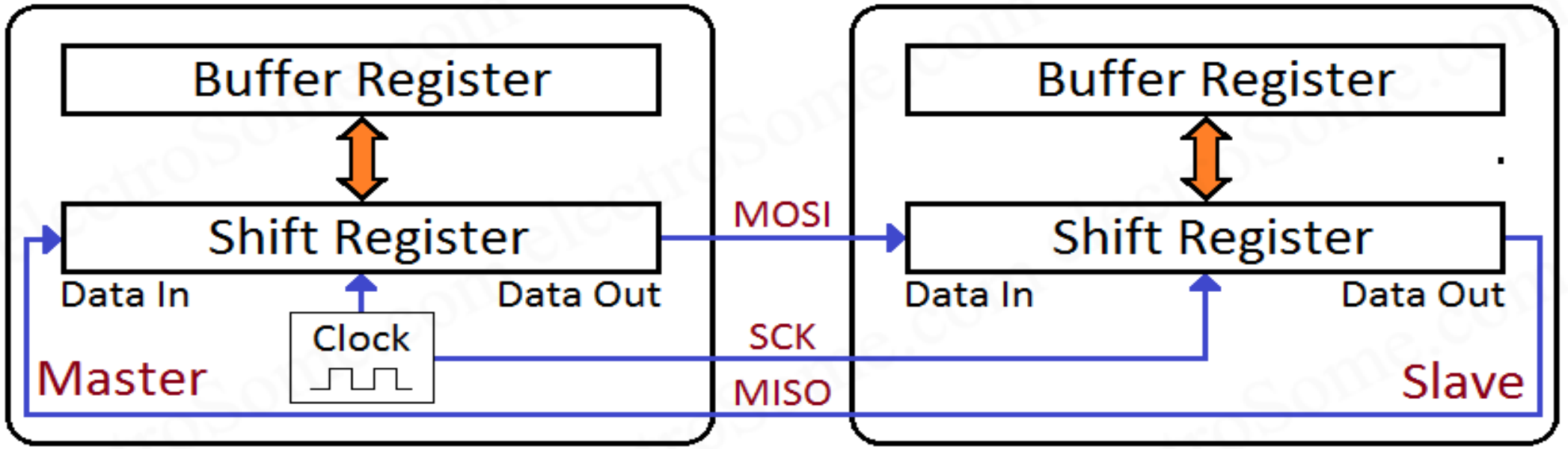
- Názvosloví není zcela konzistentní
- CLK - je vodič hodinového signálu (synchronizace)
- MOSI – master out slave in; MISO – master in slave out
- Ale také SDI – seriál data in; SDO – seriál data out
- CS (SS) – chip select, nebo slave select
- Komunikuji vždy s jedním zařízením (volím pomocí CS)
- SPI ma tedy 3 komunikační vodiče
- A pak n CS dle počtu slave jednotek



SPI

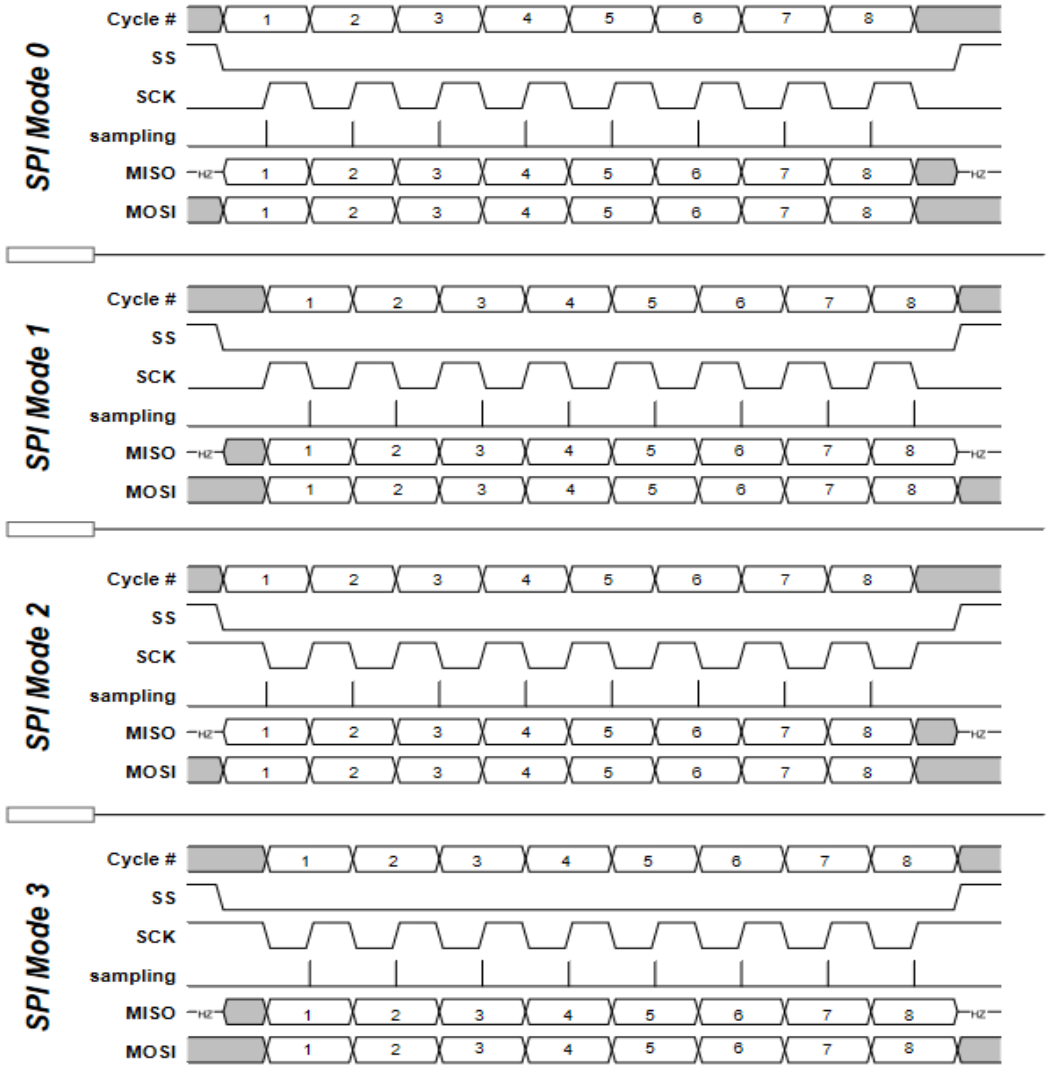
Serial pheripheral interface:

- **Princip je mimořádně jednoduchý**
- **Zařízení obsahují posuvné registry, které plní příchozími znaky (0/1)**
- **Vše je synchronizováno CLK z mástra**



SPI

- Existují 4 módy SPI
- V podstatě lze vybrat polaritu clk (normální vs. Invertovaná)
- Také mohu číst data na nástupnou, nbo vzestupnou hranu clk
- Toto spolu tvoří 4 kombinace viz obr.



- Master sends useful data and slave sends dummy data.
- Master sends useful data and slave sends useful data.
- Master sends dummy data and slave sends useful data

SPI PIC18

REGISTER 15-2: SSPxSTAT: SSPx STATUS REGISTER

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
bit 7							bit 0

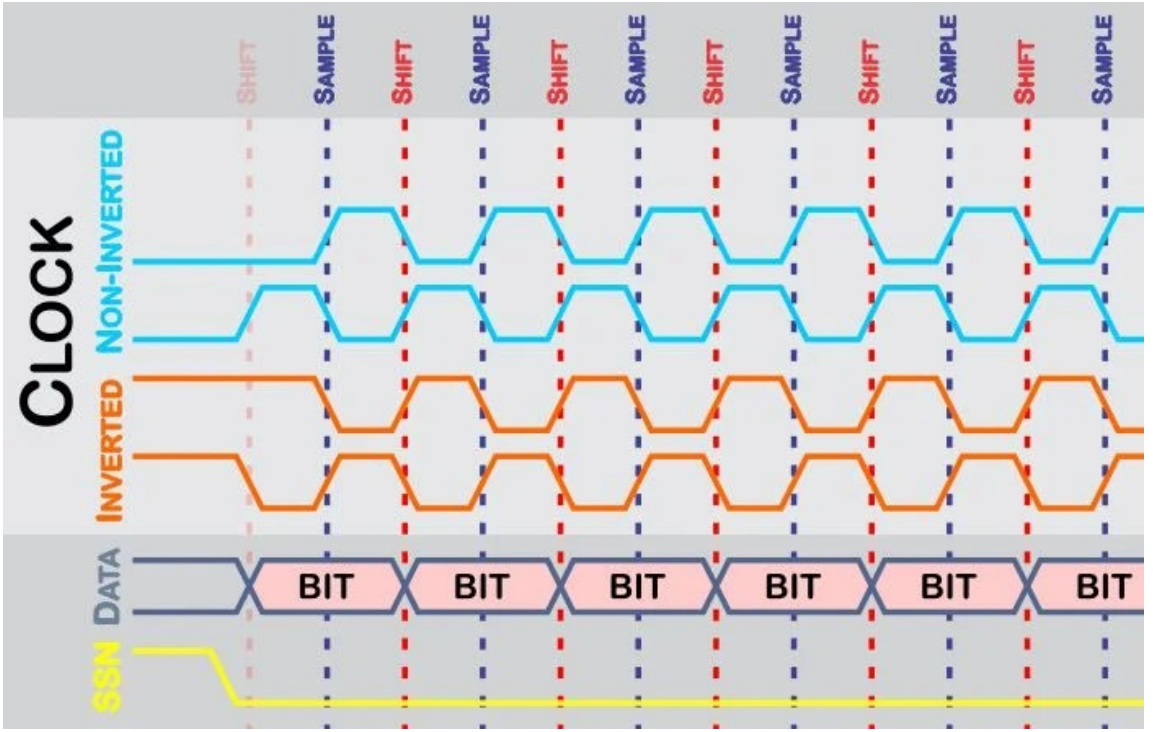
Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
'1' = Bit is set	'0' = Bit is cleared	

REGISTER 15-3: SSPxCON1: SSPx CONTROL REGISTER 1

R/C/HS-0	R/C/HS-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPxOV	SSPxEN	CKP	SSPxM<3:0>			
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets	
'1' = Bit is set	'0' = Bit is cleared	HS = Bit is set by hardware	C = User cleared

```
SSP1CON1bits.SSPM = 0b0010; // SPI clock
SSP1STATbits.CKE = 1;
SSP1CON1bits.SSPEN = 1;     // SPI zapnuto
```

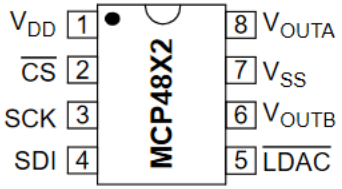


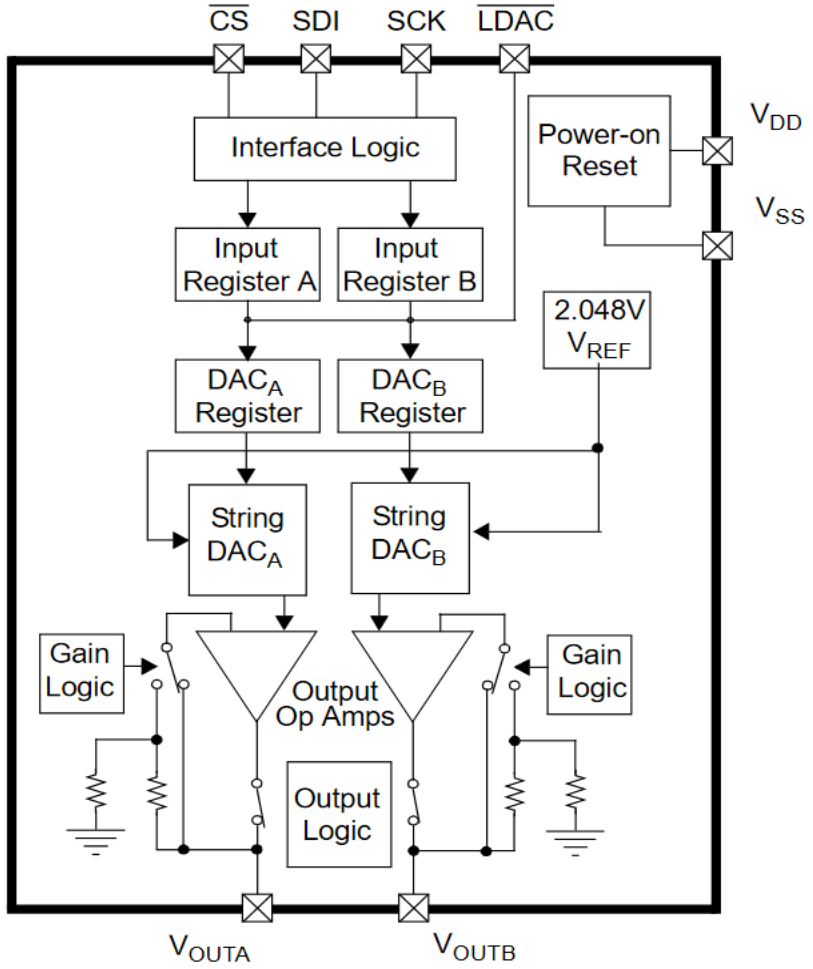
SPI PIC18

```
void SPIWrite(uint8_t channel ,uint8_t data){

    uint8_t msb, lsb, flush;
    msb = (channel | (data>>4));          // prvni bajt
    lsb = (data<<4) & 0xF0;               // druhy bajt
    DAC_SS = 0;                           // slave select
    PIR1bits.SSPIF = 0;                   // vynulovani priznaku SPI
    SSPBUF = msb;                         // zapis do bufferu
    while(PIR1bits.SSPIF == 0)NOP();      // pockat nez SPI posle prvni bajt

    PIR1bits.SSPIF = 0;                   // vynulovani priznaku SPI
    SSPBUF = lsb;                         // zapis do bufferu
    while(PIR1bits.SSPIF == 0)NOP();      // pockat nez SPI posle druhy bajt

    DAC_SS = 1;                           // vypnout slave select
    flush = SSPBUF;                       // vycteni bufferu

}
```

SPI DAC MCP4802/12/22

- SPI DAC (Pouze Sdi, tedy zapíši příkaz)
- 8/10/12 bit verze
- Dva kanály
- Vnitřní napěťová reference
- Nastavitelné zesílení 1x,2x
- String DAC obvod





REGISTER 5-3: WRITE COMMAND REGISTER FOR MCP4802 (8-BIT DAC)

W-x	W-x	W-x	W-0	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x
$\overline{A}$/B	—	$\overline{GA}$	$\overline{SHDN}$	D7	D6	D5	D4	D3	D2	D1	D0	x	x	x	x
bit 15															bit 0

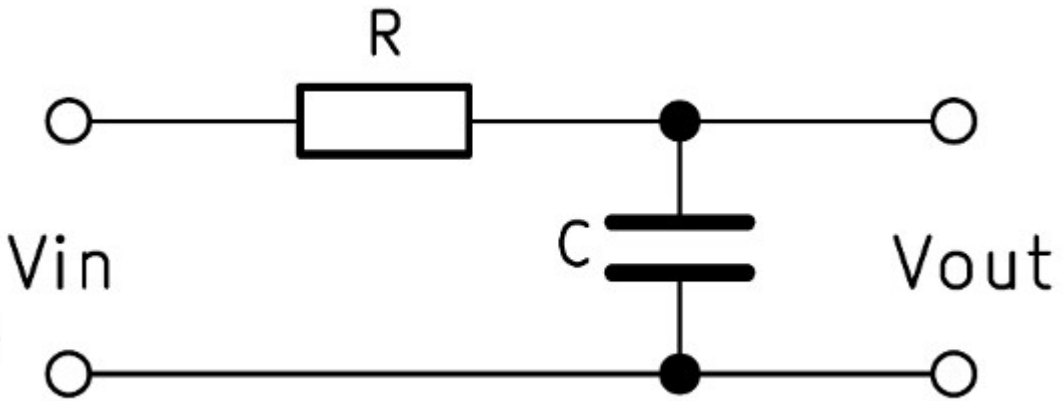
FILTR typu IIR

- Z rovnic RC filtru lze přímo odvodit tzv. Exponenciální filtr
- Jedná se o odhad plovoucího průměru se zapomínáním
- Váha předchozích hodnot klesá exponenciálně
- Po diskretizaci (euler) vidím, že nová hodnota je násobena konstantou a přičtena ke staré, která má také konstantu. Mohu i zvolit např. 0.1 a 0.9 tedy k*Vin* a (1-k)*Vout*

$$\frac{Vin - Vout}{R} = C\frac{\mathrm{d}Vout}{\mathrm{d}t}$$

$$Vout + RC\frac{\mathrm{d}Vout}{\mathrm{d}t} = Vin$$

$$V_{out}[n] = \frac{T}{T+RC}V_{in}[n] + \frac{RC}{T+RC}V_{out}[n-1]$$



- Vystačím si tedy s jednou konstantou α a mohu ještě upravit na běžný tvar exp. filtru

$$out_{[k]} = out_{[k-1]} + \alpha(new_{[k]} - out_{[k-1]})$$

FILTR typu IIR realizace

```
typedef struct {
    uint8_t alpha;
    int32_t avg;
}filter_t;

void filter_init(filter_t * self, uint8_t alpha){
    self->alpha = alpha;
    self->avg = 0;
}

int16_t filter_step(filter_t * self, int16_t in){
    int32_t mes = (int32_t)(in)<<10;
    self->avg += ((mes - self->avg)>>self->alpha);
    return (int16_t)(self->avg>>10);
}
```

- Při práci s float, nebo double použiji přímo zápis z předchozí strany.
- Při práci s int, také není žádný problém.
- Násobení konstantou mohu nahradit dělením $2^{n.}$
- Přesnost získám použitím větších čísel.