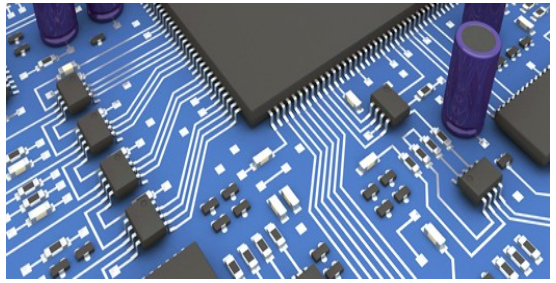
Aplikace Embedded systémů v Mechatronice



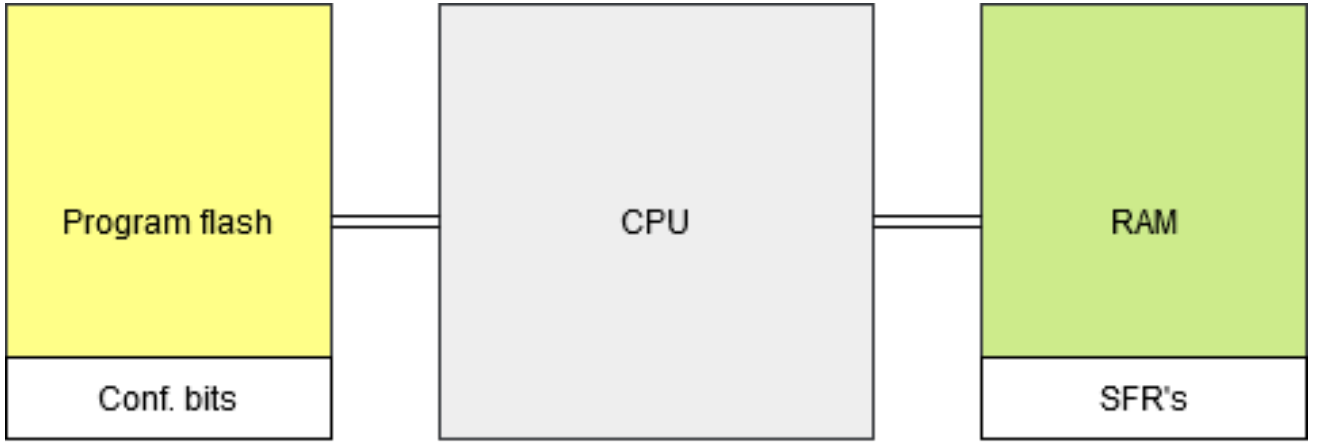
MPLAB® X IDE

Michal Bastl

PIC18

PIC18 obsahuje:
Paměť flash: slouží pro nahrání programu (pro váš program)
sekce configuration bits

Paměť RAM: (random access):
GPR General Purpose Registersl (pro vaše data)
sekce SFR special function registers (periferie)



Nastavení konfiguračních bitů

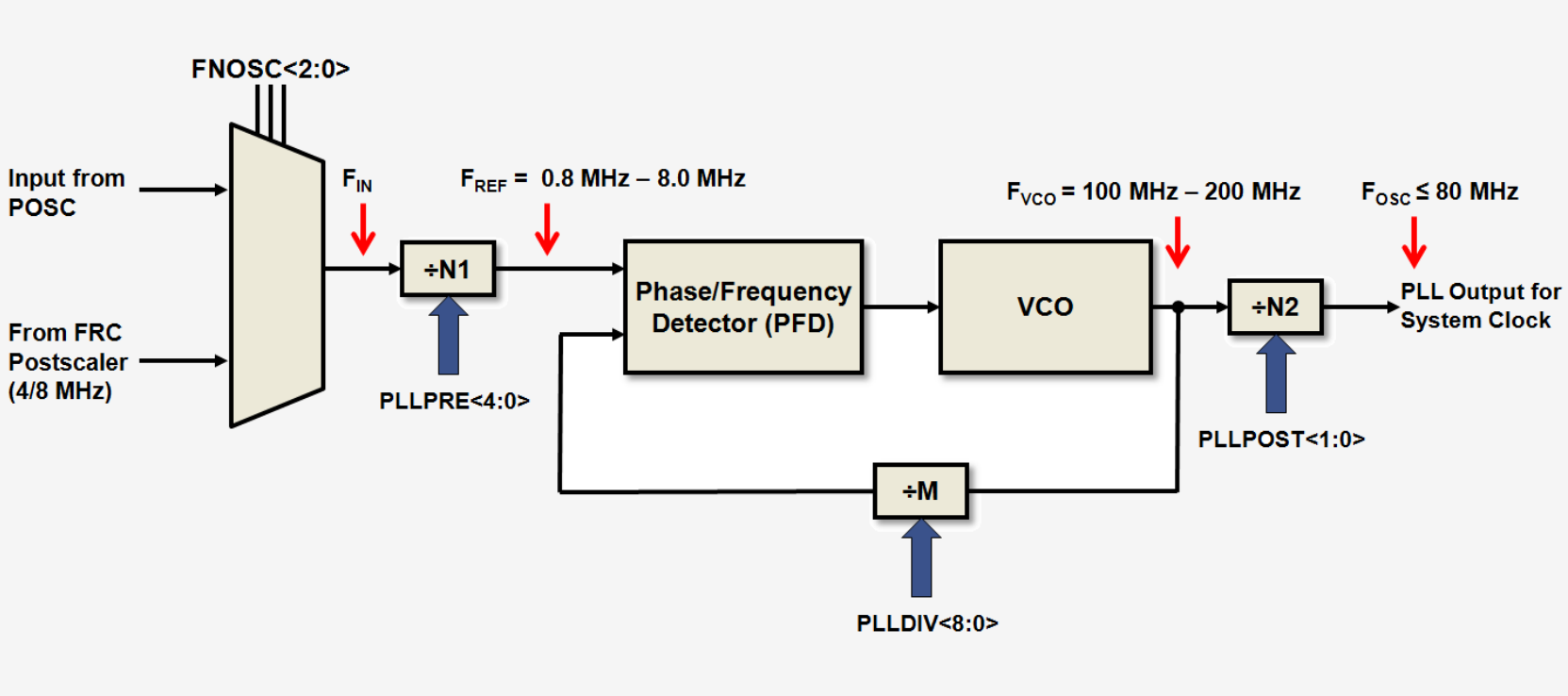
- Nastavení konfiguračních bitů se provádí během zavedení programu do MCU
- Nastavení konfiguračních bitů se potom standardně nemění
- V konfiguračních bitech jsou informace o základním nastavení MCU jako zdroj clocku, wdt, fail-safe apod
- Nastavení konfiguračních bitů se provádí pomocí direktivy #pragma

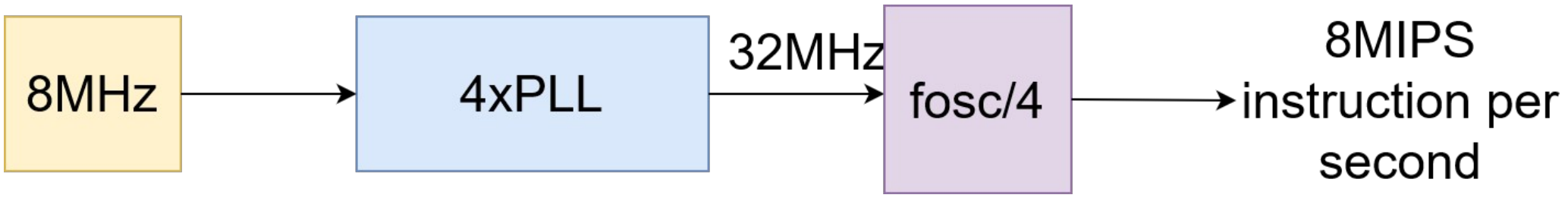
Minimalistické nastavení: (zbytek podle defaultní hodnoty)

```
#pragma config FOSC = HSMP
#pragma config PLLCFG = ON
#pragma config WDTEN = OFF
```

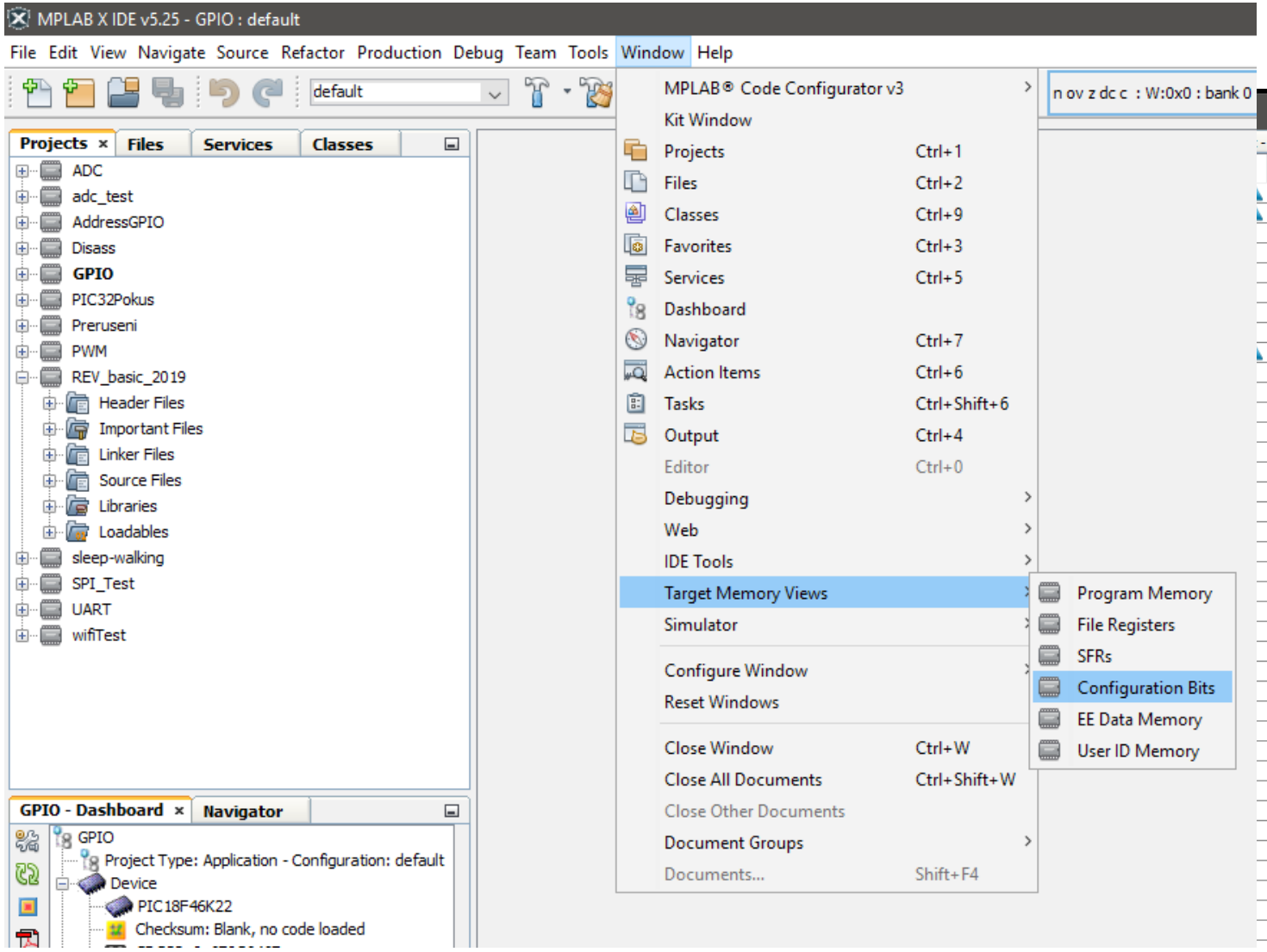
PLL

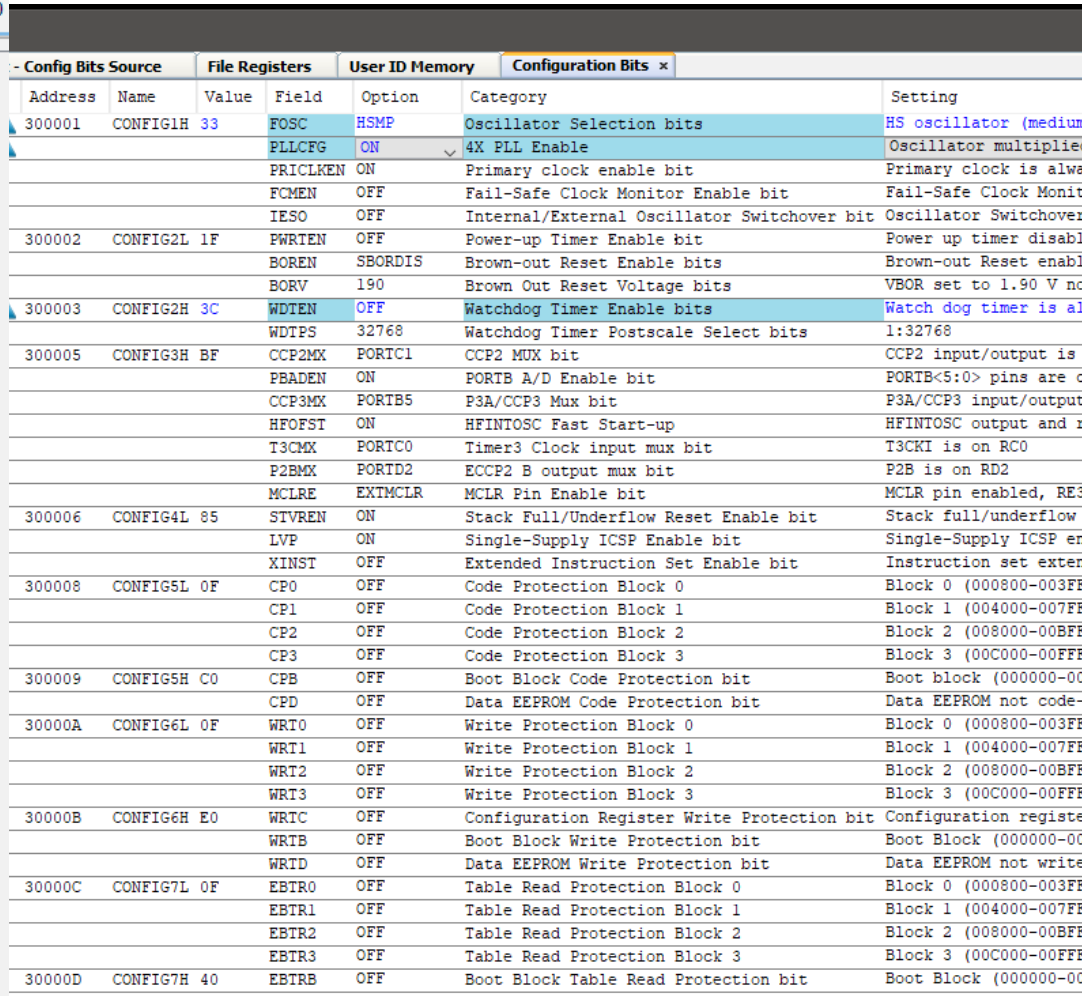
Schema PLL:





Snadné nastavení v MPLAB





Práce s dokumentaci

- Datasheet je strukturovaný a najdeme zde kapitoly podle jednotlivých periferii. GPIO, timer, ADC apod.
- Datasheet rozhodne není beletrie a nečte se tak!
- Není nutné znát přesně nastavení z hlavy. K tomu právě slouží datasheet

Datasheet:

- https://ww1.microchip.com/downloads/en/DeviceDoc/40001412G.pdf

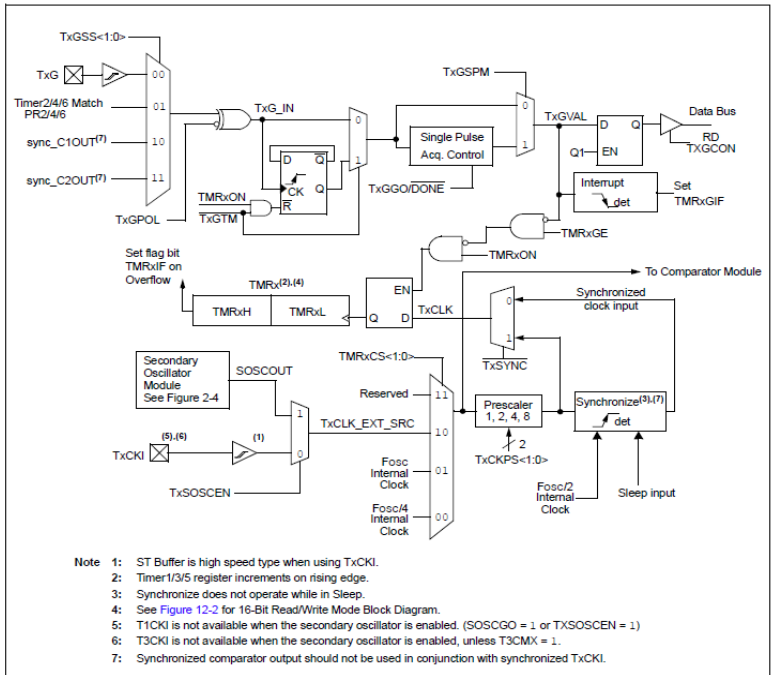
12.0 TIMER1/3/5 MODULE WITH GATE CONTROL

The Timer1/3/5 module is a 16-bit timer/counter with the following features:

- 16-bit timer/counter register pair (TMRxH:TMRxL)
- Programmable internal or external clock source
- 2-bit prescaler
- Dedicated Secondary 32 kHz oscillator circuit
- Optionally synchronized comparator out
- Multiple Timer1/3/5 gate (count enable) sources
- Interrupt on overflow
- Wake-up on overflow (external clock, Asynchronous mode only)
- 16-Bit Read/Write Operation
- Time base for the Capture/Compare function
- Special Event Trigger (with CCP/ECCP)
- Selectable Gate Source Polarity
- Gate Toggle mode
- Gate Single-pulse mode
- Gate Value Status
- Gate Event Interrupt

Figure 12-1 is a block diagram of the Timer1/3/5 module.

FIGURE 12-1: TIMER1/3/5 BLOCK DIAGRAM



Note 1: ST Buffer is high speed type when using TxCKI.
2: Timer1/3/5 register increments on rising edge.
3: Synchronize does not operate while in Sleep.
4: See Figure 12-2 for 16-Bit Read/Write Mode Block Diagram.
5: T1CKI is not available when the secondary oscillator is enabled. (SOSCGO = 1 or TXSOSCEN = 1)
6: T3CKI is not available when the secondary oscillator is enabled, unless T3CMX = 1.
7: Synchronized comparator output should not be used in conjunction with synchronized TxCKI.

SFR

SFR – special function registers

- Jsou speciální registry, které slouží pro práci s periferiemi zařízení
- Každá periferie má své registry, ty mají svoji adresu v paměti
- Mají většinou intuitivní název např. ANSEL (analog selectio), PIE (peripheral interrupt enabled), TxCON (Timer configuration)

- V hlavičkovém souboru xc.h jsou zavedeny makra, které může programátor používat pro práci s periferiemi

- Je však možné pracovat přímo s adresou v dokumentaci na str.78 je mapa SFR

- Napriklad registr ANSELA (analog-selection for port A) má adresu F38hex

TABLE 5-1: SPECIAL FUNCTION REGISTER MAP FOR PIC18(L)F2X/4XK22 DEVICES

Address	Name	Address	Name	Address	Name	Address	Name	Address	Name
FFFh	TOSU	FD7h	TMR0H	FAFh	SPBRG1	F87h	—(2)	F5Fh	CCPR3H
FFEh	TOSH	FD6h	TMR0L	FAEh	RCREG1	F86h	—(2)	F5Eh	CCPR3L
FFDh	TOSL	FD5h	T0CON	FADh	TXREG1	F85h	—(2)	F5Dh	CCP3CON
FFCh	STKPTR	FD4h	—(2)	FACh	TXSTA1	F84h	PORTE	F5Ch	PWM3CON
FFBh	PCLATU	FD3h	OSCCON	FABh	RCSTA1	F83h	PORTD(3)	F5Bh	ECCP3AS
FFAh	PCLATH	FD2h	OSCCON2	FAAh	EEADRH(4)	F82h	PORTC	F5Ah	PSTR3CON
FF9h	PCL	FD1h	WDTCON	FA9h	EEADR	F81h	PORTB	F59h	CCPR4H
FF8h	TBLPTRU	FD0h	RCON	FA8h	EEDATA	F80h	PORTA	F58h	CCPR4L
FF7h	TBLPTRH	FCFh	TMR1H	FA7h	EECON2(1)	F7Fh	IPR5	F57h	CCP4CON
FF6h	TBLPTRL	FCEh	TMR1L	FA6h	EECON1	F7Eh	PIR5	F56h	CCPR5H
FF5h	TABLAT	FCDh	T1CON	FA5h	IPR3	F7Dh	PIE5	F55h	CCPR5L
FF4h	PRODH	FCCh	T1GCON	FA4h	PIR3	F7Ch	IPR4	F54h	CCP5CON
FF3h	PRODL	FCBh	SSP1CON3	FA3h	PIE3	F7Bh	PIR4	F53h	TMR4
FF2h	INTCON	FCAh	SSP1MSK	FA2h	IPR2	F7Ah	PIE4	F52h	PR4
FF1h	INTCON2	FC9h	SSP1BUF	FA1h	PIR2	F79h	CM1CON0	F51h	T4CON
FF0h	INTCON3	FC8h	SSP1ADD	FA0h	PIE2	F78h	CM2CON0	F50h	TMR5H
FEFh	INDF0(1)	FC7h	SSP1STAT	F9Fh	IPR1	F77h	CM2CON1	F4Fh	TMR5L
FEEh	POSTINC0(1)	FC6h	SSP1CON1	F9Eh	PIR1	F76h	SPBRGH2	F4Eh	T5CON
FEDh	POSTDEC0(1)	FC5h	SSP1CON2	F9Dh	PIE1	F75h	SPBRG2	F4Dh	T5GCON
FECh	PREINC0(1)	FC4h	ADRESH	F9Ch	HLVDCON	F74h	RCREG2	F4Ch	TMR6
FEBh	PLUSW0(1)	FC3h	ADRESL	F9Bh	OSCTUNE	F73h	TXREG2	F4Bh	PR6
FEAh	FSR0H	FC2h	ADCON0	F9Ah	—(2)	F72h	TXSTA2	F4Ah	T6CON
FE9h	FSR0L	FC1h	ADCON1	F99h	—(2)	F71h	RCSTA2	F49h	CCPTMRS0
FE8h	WREG	FC0h	ADCON2	F98h	—(2)	F70h	BAUDCON2	F48h	CCPTMRS1
FE7h	INDF1(1)	FBFh	CCPR1H	F97h	—(2)	F6Fh	SSP2BUF	F47h	SRCON0
FE6h	POSTINC1(1)	FBEh	CCPR1L	F96h	TRISE	F6Eh	SSP2ADD	F46h	SRCON1
FE5h	POSTDEC1(1)	FBDh	CCP1CON	F95h	TRISD(3)	F6Dh	SSP2STAT	F45h	CTMUCONH
FE4h	PREINC1(1)	FBCh	TMR2	F94h	TRISC	F6Ch	SSP2CON1	F44h	CTMUCONL
FE3h	PLUSW1(1)	FBBh	PR2	F93h	TRISB	F6Bh	SSP2CON2	F43h	CTMUICON
FE2h	FSR1H	FBAh	T2CON	F92h	TRISA	F6Ah	SSP2MSK	F42h	VREFCON0
FE1h	FSR1L	FB9h	PSTR1CON	F91h	—(2)	F69h	SSP2CON3	F41h	VREFCON1
FE0h	BSR	FB8h	BAUDCON1	F90h	—(2)	F68h	CCPR2H	F40h	VREFCON2
FDFh	INDF2(1)	FB7h	PWM1CON	F8Fh	—(2)	F67h	CCPR2L	F3Fh	PMD0
FDEh	POSTINC2(1)	FB6h	ECCP1AS	F8Eh	—(2)	F66h	CCP2CON	F3Eh	PMD1
FDDh	POSTDEC2(1)	FB5h	—(2)	F8Dh	LATE(3)	F65h	PWM2CON	F3Dh	PMD2
FDCh	PREINC2(1)	FB4h	T3GCON	F8Ch	LATD(3)	F64h	ECCP2AS	F3Ch	ANSELE
FDBh	PLUSW2(1)	FB3h	TMR3H	F8Bh	LATC	F63h	PSTR2CON	F3Bh	ANSELD
FDAh	FSR2H	FB2h	TMR3L	F8Ah	LATB	F62h	IOCB	F3Ah	ANSELC
FD9h	FSR2L	FB1h	T3CON	F89h	LATA	F61h	WPUB	F39h	ANSELB
FD8h	STATUS	FB0h	SPBRGH1	F88h	—(2)	F60h	SLRCON	F38h	ANSELA

Použití SFR v C

- Klíčové slovo volatile ještě uvidíme
- Použití tohoto slova v definici/deklaraci proměnné znamená, že zakazujeme optimalizace této proměnné

```
// REV GPIO
#pragma config FOSC = HSMP      // Oscillator Selection bits (HS oscillator (medium power 4-16 MHz))
#pragma config PLLCFG = ON      // 4X PLL Enable (Oscillator multiplied by 4)
#pragma config WDTEN = OFF      // Watchdog Timer Enable bits (Watch dog timer is always disabled. SWDTEN

volatile unsigned int TRISD     __at(0xf95);
volatile unsigned int TRISC     __at(0xf94);

volatile unsigned int LATD      __at(0xf8c);
volatile unsigned int PORTC     __at(0xf82);

int main(void) {

    TRISD &= ~(1 << 2);     // nastaveni RD2 jako výstup
    TRISC |= 0b1;           // nastavení RC0 jako vstup

    while(1){

        if (PORTC & 0b1){               //  kontrola stisknutí BTN1
            LATD ^= (1 << 2);           //  prevrácení LED1 pomocí XOR
        }
        for(long i=1; i<100000; i++);   //  cekání...
    }
    return 0;                           // nikdy se neprovede
}
```

Použití SFR v C

- Standardně se používá xc.h
- Makra SFR jsou v něm již hotová přesně pro náš MCU

```
// REV GPIO
#pragma config FOSC = HSMP      // Oscillator Selection bits (HS oscillator (medium power 4-16 MHz))
#pragma config PLLCFG = ON      // 4X PLL Enable (Oscillator multiplied by 4)
#pragma config WDTEN = OFF      // Watchdog Timer Enable bits (Watch dog timer is always disabled. SWDTEN

#include <xc.h>

#define LED1 LATDbits.LATD2
#define BTN1 PORTCbits.RC0

int main(void) {

    TRISDbits.TRISD2 = 0;
    TRISCbits.TRISC0 = 1;

    while(1){

        if (PORTCbits.RC0){             //  kontrola stisknuti BTN1
            LATDbits.LATD2 ^= 1;        //  prevráceni LED1 pomoci XOR
        }
        for(long i=1; i<100000; i++);   //  cekani...
    }
    return 0;                           // nikdy se neprovede
}
```

Práce s datasheetem

Práce s periferiemi vyžaduje manipulaci s SFR (special function registers).
V Datasheetu MCU nalezneme význam a popis nastavení .

Přiklad nastavuje část registru s názvem IRCF na 111 která znamená 16MHz víz printscreen

OSCCON = (OSCCON & 0b10001111) | 0b01110000;

Masky:

&	|	^
10011110	10011110	10011110
00001111	00000001	00001111
00001110	10011111	10010001

2.3 Register Definitions: Oscillator Control

REGISTER 2-1: OSCCON: OSCILLATOR CONTROL REGISTER

R/W-0	R/W-0	R/W-1	R/W-1	R-q	R-0	R/W-0	R/W-0
IDLEN	IRCF<2:0>			OSTS(1)	HFIOFS	SCS<1:0>	
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	q = depends on condition
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

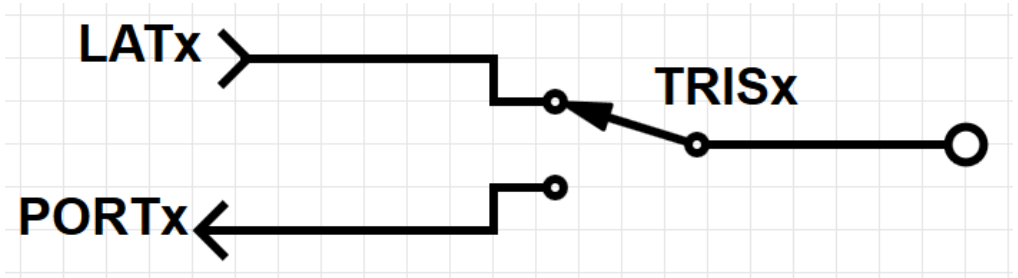
bit 7 **IDLEN:** Idle Enable bit
1 = Device enters Idle mode on SLEEP instruction
0 = Device enters Sleep mode on SLEEP instruction

bit 6-4 **IRCF<2:0>:** Internal RC Oscillator Frequency Select bits(2)
111 = HFINTOSC – (16 MHz)
110 = HFINTOSC/2 – (8 MHz)
101 = HFINTOSC/4 – (4 MHz)
100 = HFINTOSC/8 – (2 MHz)
011 = HFINTOSC/16 – (1 MHz)(3)

If INTSRC = 0 and MFIOSEL = 0:
010 = HFINTOSC/32 – (500 kHz)
001 = HFINTOSC/64 – (250 kHz)
000 = LFINTOSC – (31.25 kHz)

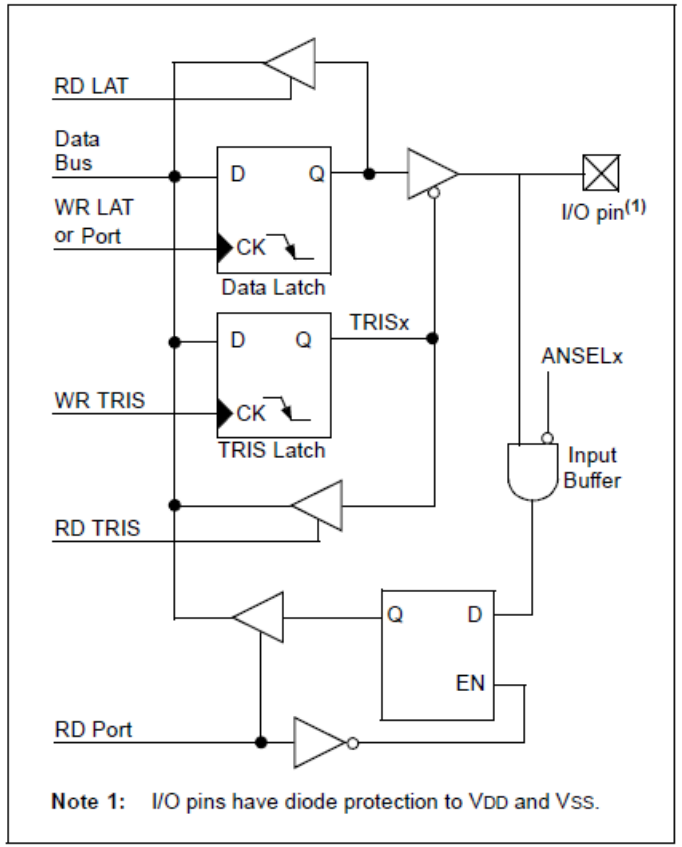
GPIO pin

- General purpose input/output, tedy obecný vstupně/výstupní pin.
- Slouží k základní interakci MCU s okolním světem.
- Na GPIO pin lze zapisovat 1, tedy napětí blízké napájecímu 3.3V, nebo 0 napětí blízké 0V.
- V dalším režimu lze pinem číst napětí, pokud je blízké 0V čte se jako 0, nebo blízké 3,3V jako 1.



Zjednodušení

FIGURE 10-1: GENERIC I/O PORT OPERATION

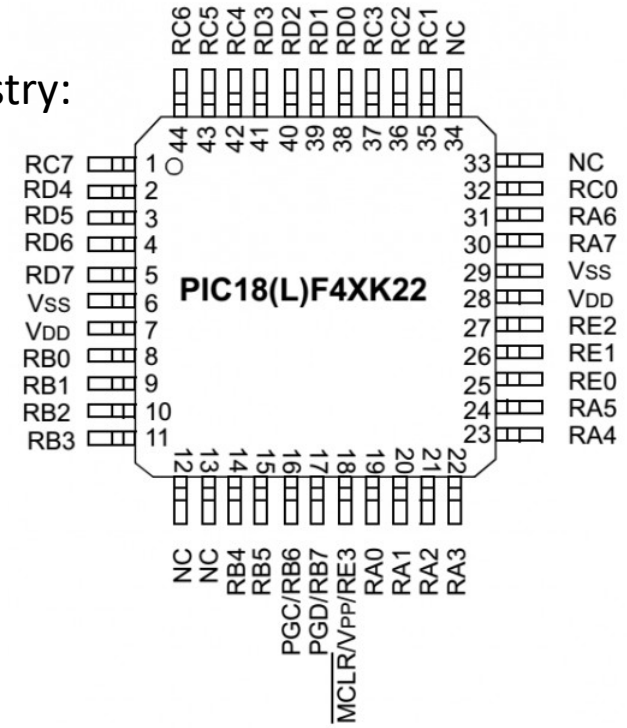


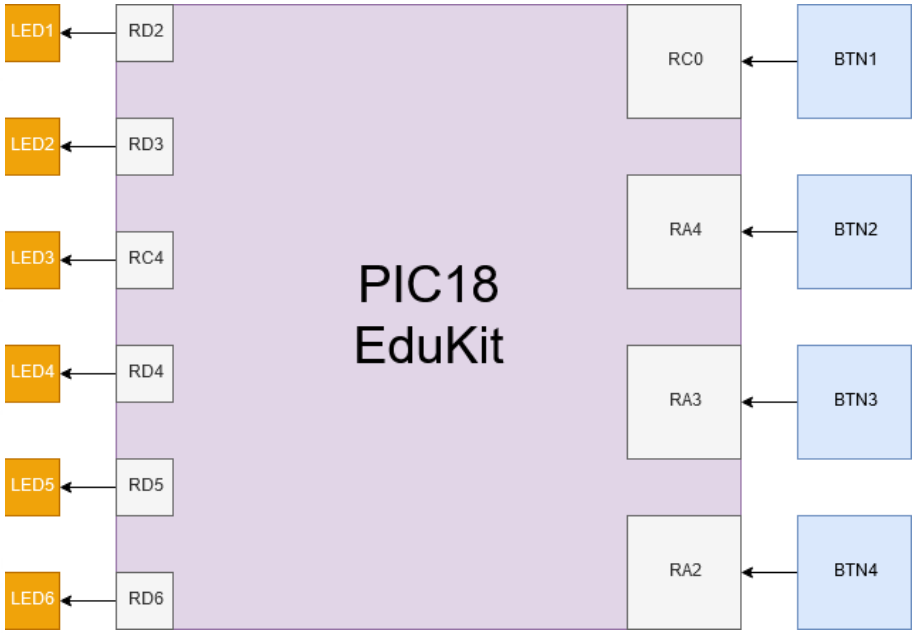
Note 1: I/O pins have diode protection to VDD and VSS.

GPIO

Pro práci s I/O piny budeme používat tyto registry:

1. TRISx
2. LATx
3. PORTx
4. ANSELx





TRISx

Lze interpretovat jako pomyslný přepínač a nastavuje zda bude pin vstup 1, a nebo výstup 0.

ANSELx

Nastavuje pin do stavu pro čtení ADC což zatím nechceme.

Všechny vstupy, které sdílejí ADC, nebo komparátor!!

PORTx

Pokud je pin nastaven jako vstup, z tohoto registru lze přečíst stav příslušného pinu.

LATx

Pokud je pin přepnut jako výstup, lze tímto registrem nastavovat logickou úroveň na pinu. Z tohoto registru lze číst aktuální nastavení i přepsat „nastavit“ požadovaný stav.

TRISx

Nastavuje zda bude pin vstup 1, nebo výstup 0.

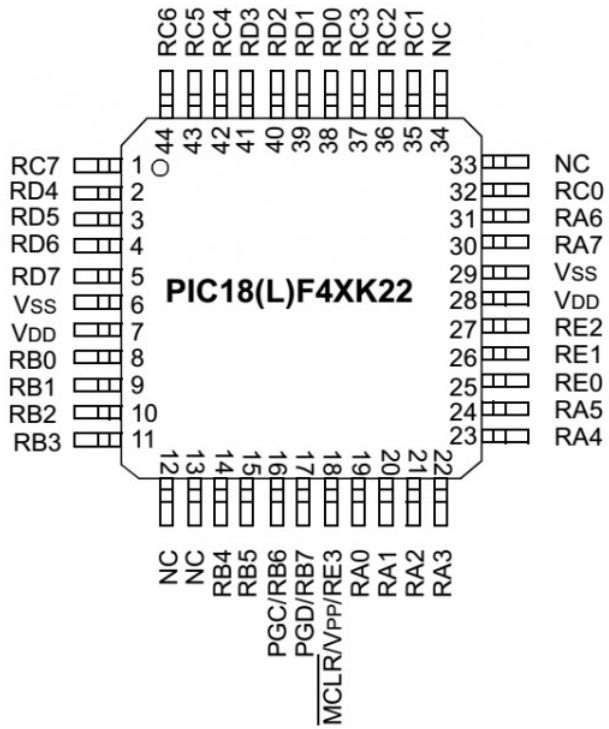
REGISTER 10-8: TRISx: PORTx TRI-STATE REGISTER(1)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TRISx7	TRISx6	TRISx5	TRISx4	TRISx3	TRISx2	TRISx1	TRISx0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-0 **TRISx<7:0>:** PORTx Tri-State Control bit
1 = PORTx pin configured as an input (tri-stated)
0 = PORTx pin configured as an output

Note 1: Register description for TRISA, TRISB, TRISC and TRISD.




```
TRISD = 0b00001111;      //nastaveni portu D pulka pinu vstup, zbytek vystup

TRISDbits.TRISD4 = 0;    //nastaveni pomoci jednotlivych bitu
TRISDbits.TRISD5 = 0;
TRISDbits.TRISD6 = 0;
```

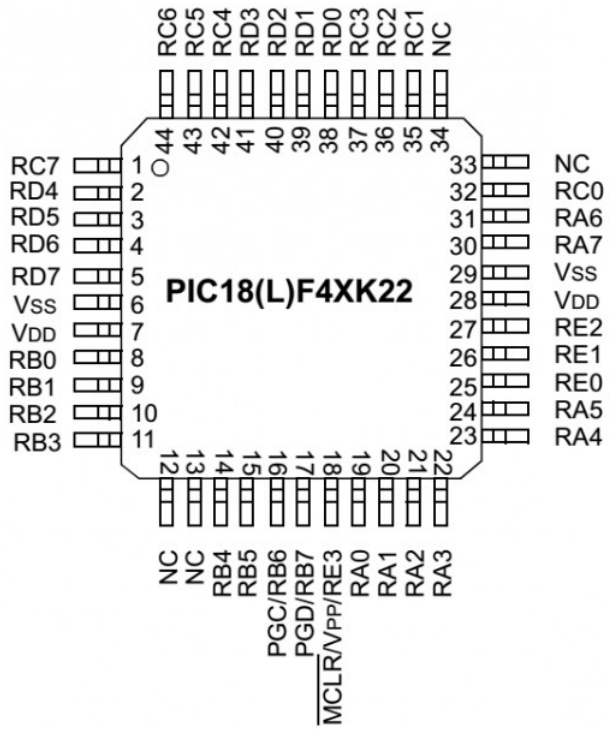
LATx

REGISTER 10-10: LATx: PORTx OUTPUT LATCH REGISTER(1)

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
LATx7	LATx6	LATx5	LATx4	LATx3	LATx2	LATx1	LATx0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-0 **LATx<7:0>**: PORTx Output Latch bit value(2)




```
LATDbits.LATD2 = 1;                 //zapis logicke 1 na pin
LATDbits.RD2 = 1;                   //totez alternativni nazev s nazvem pinu

LATD = 0xFF;                        //prepsani vsech RD pinu na 1

LATDbits.LATD2 = ~LATDbits.LATD2;   //prevraceni pinu
LATDbits.LATD2 ^= 1;                //xor je často rychlejsi
```

PORTx

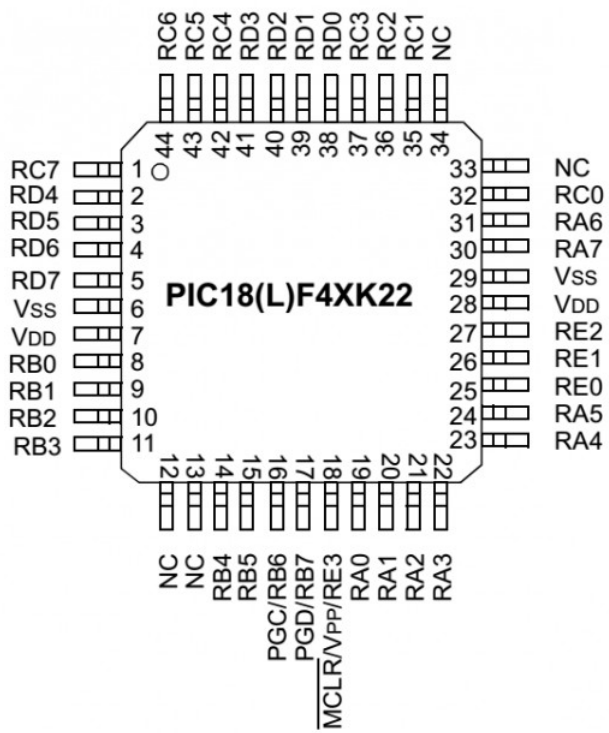
10.9 Register Definitions – Port Control

REGISTER 10-1: PORTX(1): PORTx REGISTER

R/W-u/x	R/W-u/x	R/W-u/x	R/W-u/x	R/W-u/x	R/W-u/x	R/W-u/x	R/W-u/x
Rx7	Rx6	Rx5	Rx4	Rx3	Rx2	Rx1	Rx0
bit 7							bit 0

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

bit 7-0 **Rx<7:0>**: PORTx I/O bit values(2)




```
if(PORTCbits.RC0 == 1){
    //magic happens here
}
```

Uživatelská makra

```
#define BTN1    PORTCbits.RC0
#define BTN2    PORTAbits.RA4
#define BTN3    PORTAbits.RA3
#define BTN4    PORTAbits.RA2

#define LED1    LATDbits.LATD2
#define LED2    LATDbits.LATD3
#define LED3    LATCbits.LATC4
#define LED4    LATDbits.LATD4
#define LED5    LATDbits.LATD5
#define LED6    LATDbits.LATD6
```

V kódu pak používám definovaná makra namísto krkolomného zápisu.

```
if(BTN1){
      LED1 = 1;
}
```

V makru lze definovat i cele části kódu

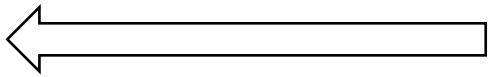
```
#define True     1
#define False    0
#define ledOn(led)      do{ led = 0;}while(0)
#define ledOff(led)     do{ led = 1;}while(0)
#define ledToggle(led)  do{ led = ~led;}while(0)
```

Na EduKitu se přečte zmáčknuté tlačítko jako logická 1.

Naopak LED diody svítí na logickou 0

Inicializace a nastavení GPIO

```
void driveLED(char in){
    in = ~in;
    LATD2 = in & 1;          //LED0
    LATD3 = in & 2 ? 1 : 0;  //LED1
    LATC4 = in & 4 ? 1 : 0;  //LED2
    LATD4 = in & 8 ? 1 : 0;  //LED3
    LATD5 = in & 16 ? 1 : 0; //LED4
    LATD6 = in & 32 ? 1 : 0; //LED5
}
```



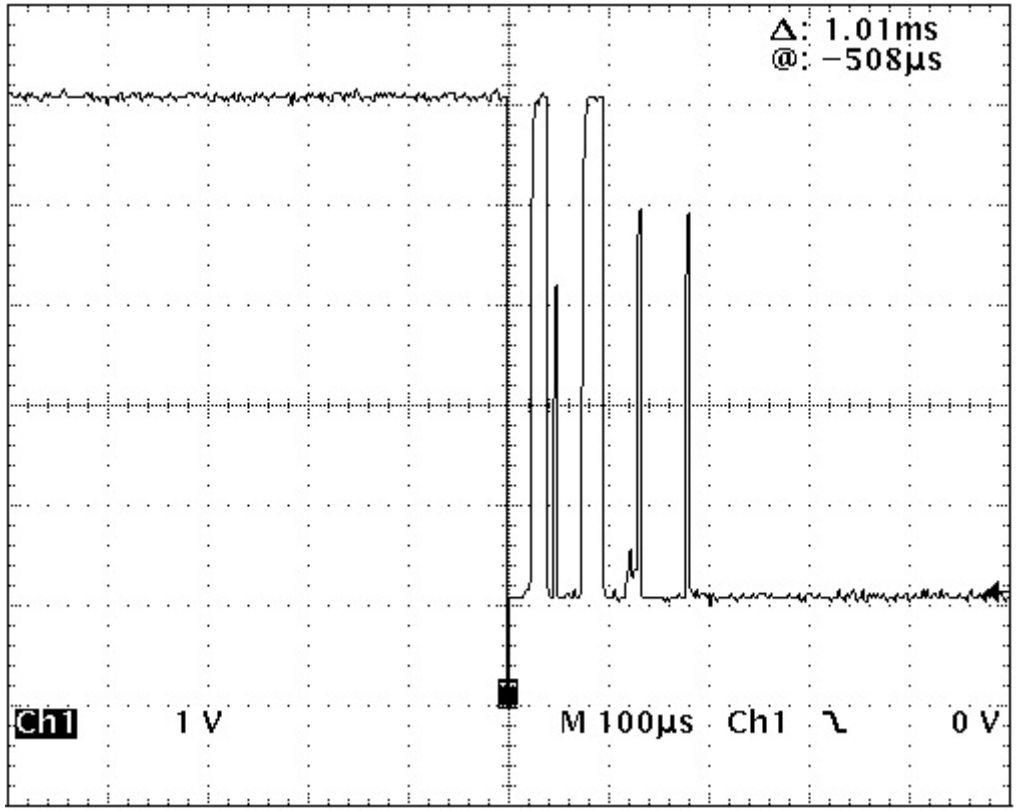
Na cvičení bude pracovat s funkcí obsluhující LED na kitu. Zápis probíhá pomocí proměnné typu char. Kolik a jaké led se rozsvítí po zápisu hodnoty 6dec?

```
void init(void){
    ANSELA = 0x00;
    ANSELC = 0x00;

    // set pins as outputs
    TRISDbits.TRISD2 = 0;
    TRISDbits.TRISD3 = 0;
    TRISCbits.TRISC4 = 0;
    TRISDbits.TRISD4 = 0;
    TRISDbits.TRISD5 = 0;
    TRISDbits.TRISD6 = 0;

    // set pins as inputs
    TRISAbits.TRISA4 = 1;
    TRISAbits.TRISA3 = 1;
    TRISAbits.TRISA2 = 1;
    TRISCbits.TRISC0 = 1;

    LED1 = 1;
    LED2 = 1;
    LED3 = 1;
    LED4 = 1;
    LED5 = 1;
    LED6 = 1;
}
```

Debouncing

- V praxi se může vyskytnout problém při stlačení tlačítka
- Ten se projevuje tak, že tlačítko se například vyhodnotí jako zmáčknuté vícekrát apod.
- Problém je třeba řešit jak vhodným HW tak nejlépe i v SW

- Tento jev trvá cca 5-10ms
- Nejednodušší (ne nejlepší) řešení je přečíst tlačítko, počkat a přečíst znovu



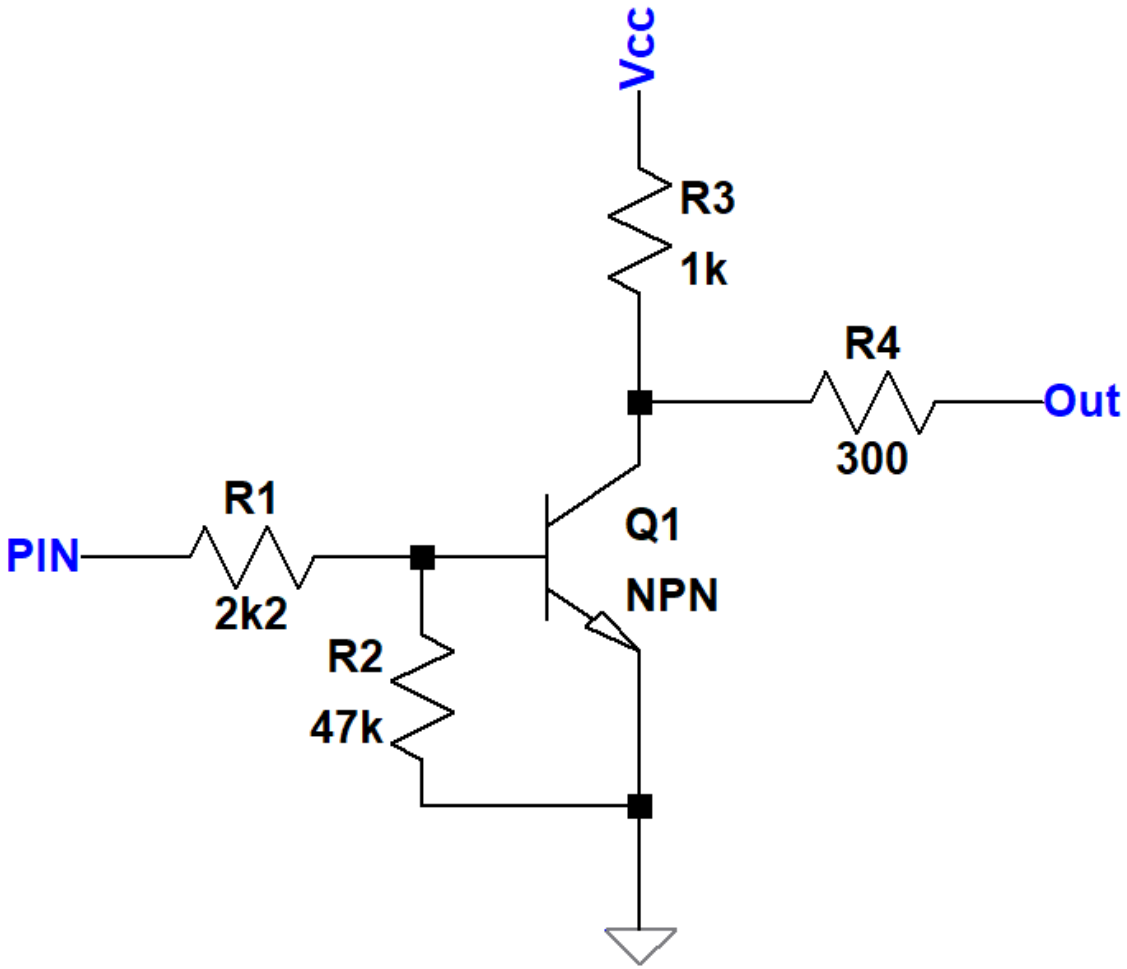
GPIO příklady

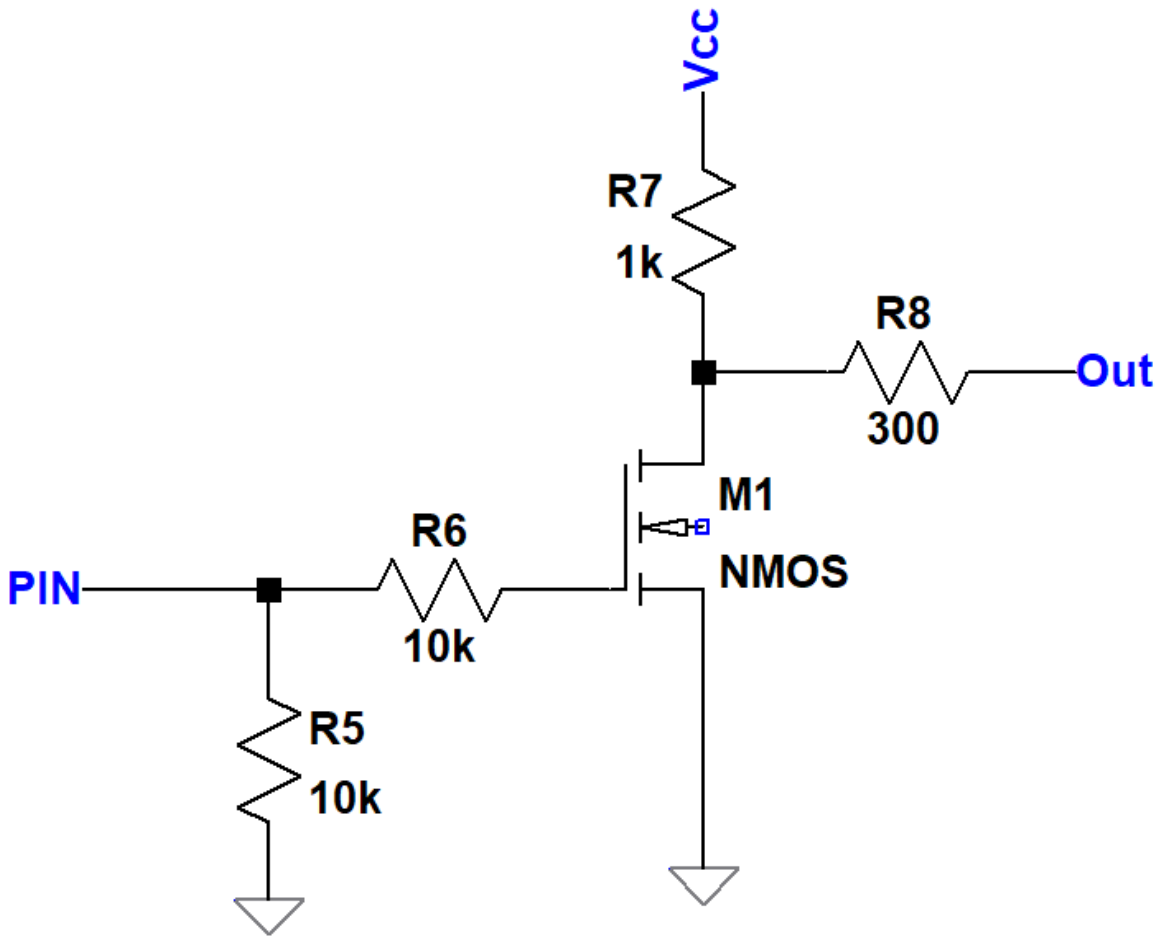
```
void main(void)
{
   init();
   unsigned char leds = 63;

   while(True){
         __delay_ms(1000);
         leds ^= 63;
         driveLED(leds);
      }
   }

void driveLED(char in){
      in = ~in;
      LATD2 = in & 1;              //LED0
      LATD3 = in & 2 ? 1 : 0;      //LED1
      LATC4 = in & 4 ? 1 : 0;      //LED2
      LATD4 = in & 8 ? 1 : 0;      //LED3
      LATD5 = in & 16 ? 1 : 0;     //LED4
      LATD6 = in & 32 ? 1 : 0;     //LED5
}
```

Přiložený kód převrací stav ledky po zmáčknutí příslušného tlačítka

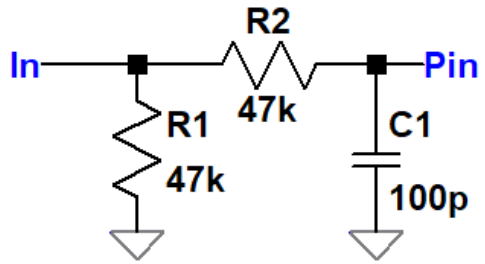
```
while(1){
      if(BTN1 | BTN2 | BTN3 | BTN4){
         __delay_ms(10);
         if(BTN1){
            ledToggle(LED1);
            while(BTN1);
         }
         else if(BTN2){
            ledToggle(LED2);
            while(BTN2);
         }
         else if(BTN3){
            ledToggle(LED3);
            while(BTN3);
         }
         else if(BTN4){
            ledToggle(LED4);
            while(BTN4);
         }
      }
   }
```

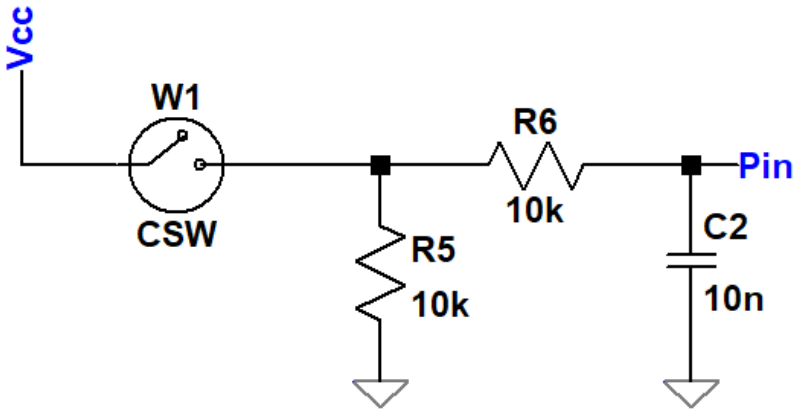
Hardware

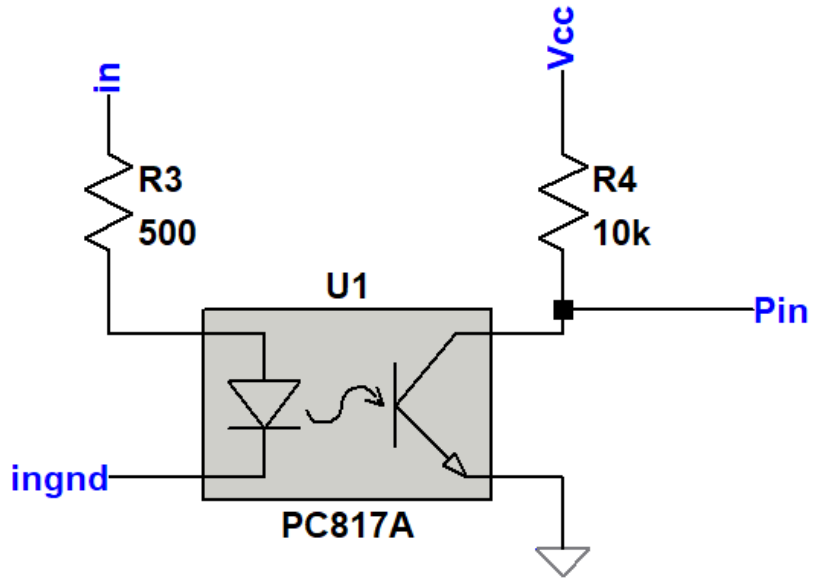

Vcc
R3
1k
R4
300
Out
R1
PIN
2k2
Q1
NPN
R2
47k

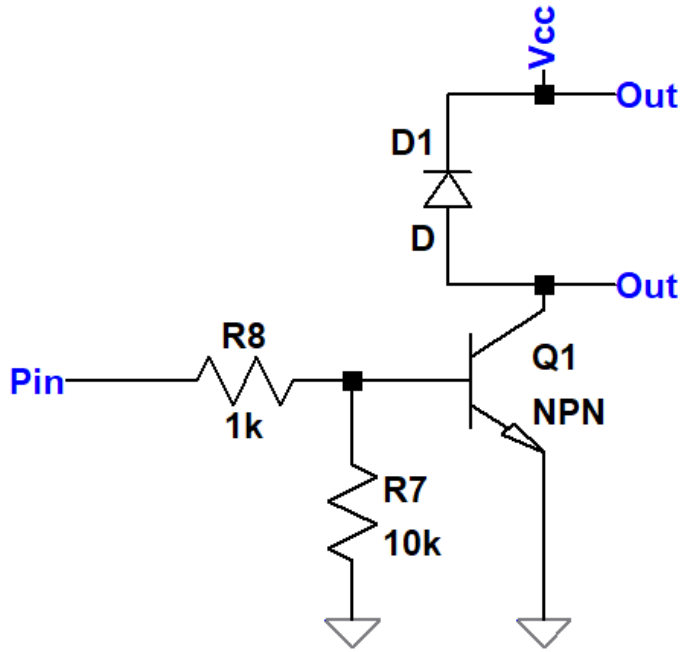

Vcc
R7
1k
R8
300
Out
M1
NMOS
R6
10k
PIN
R5
10k

Hardware


In
R2
47k
Pin
R1
47k
C1
100p


Vcc
W1
CSW
R6
10k
Pin
R5
10k
C2
10n


in
R3
500
Vcc
R4
10k
U1
Pin
ingnd
PC817A


Vcc
Out
D1
D
Out
R8
Pin
1k
Q1
NPN
R7
10k