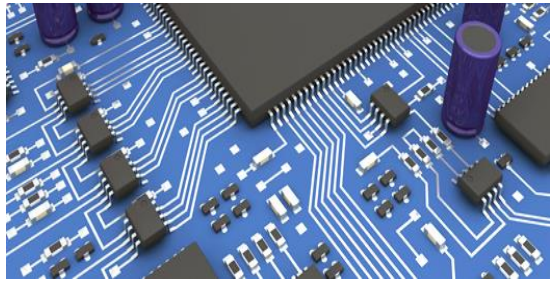
Aplikace Embedded systémů v Mechatronice



MPLAB® X IDE

Michal Bastl
A2/713a

Aplikace Embedded systémů v Mechatronice

Obsah přednášky:

- Opakování
- Datasheet
- GPIO piny
- TRISx/ANSELx registr
- LATx registr
- PORTx registr
- Ukázky použití
- Hardware poznámky

Opakování

K čemu slouží v C pointer/ukazatel?
Jak se ukazatel vytváří?
K čemu je vhodné předávat funkcím reference?
K čemu slouží klíčové slovo typedef?
Jak popíšete strukturu?
Co se stane, když k ukazateli přičtu jedna?

Práce s datasheetem

- 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

Ukázka práce s datasheetem

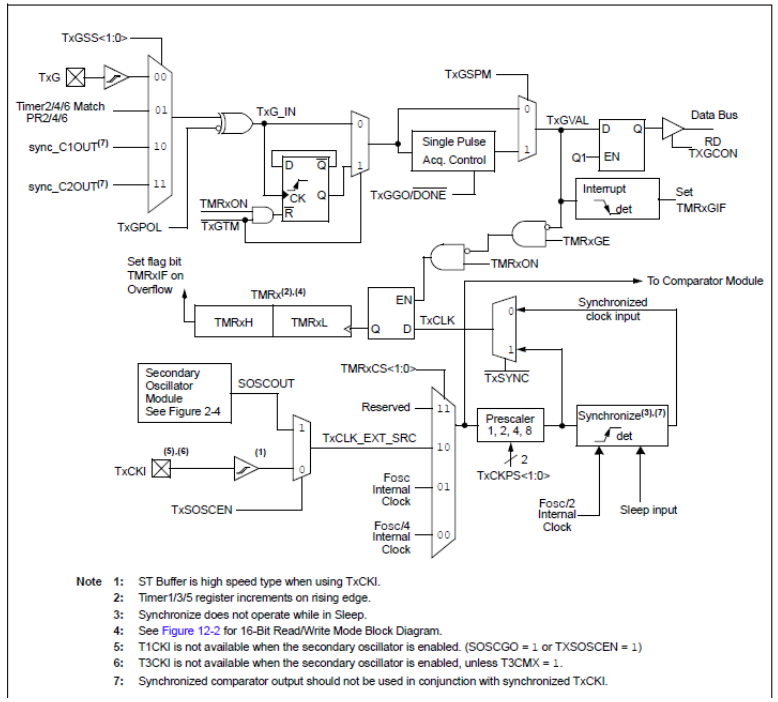
12.0 TIMER1/3/5 MODULE WITH GATE CONTROL

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.

Práce s datasheetem

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

Například nastavení interního oscilátoru z REV-basic.c

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

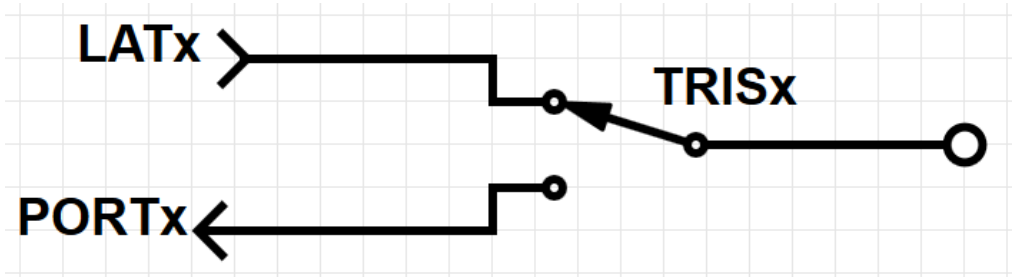
bit 7 **IDLEN:** Idle Enable bit
1 = Device enters Idle mode on SLEEP instruction
0 = Device enters Sleep mode on SLEEP instruction

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)

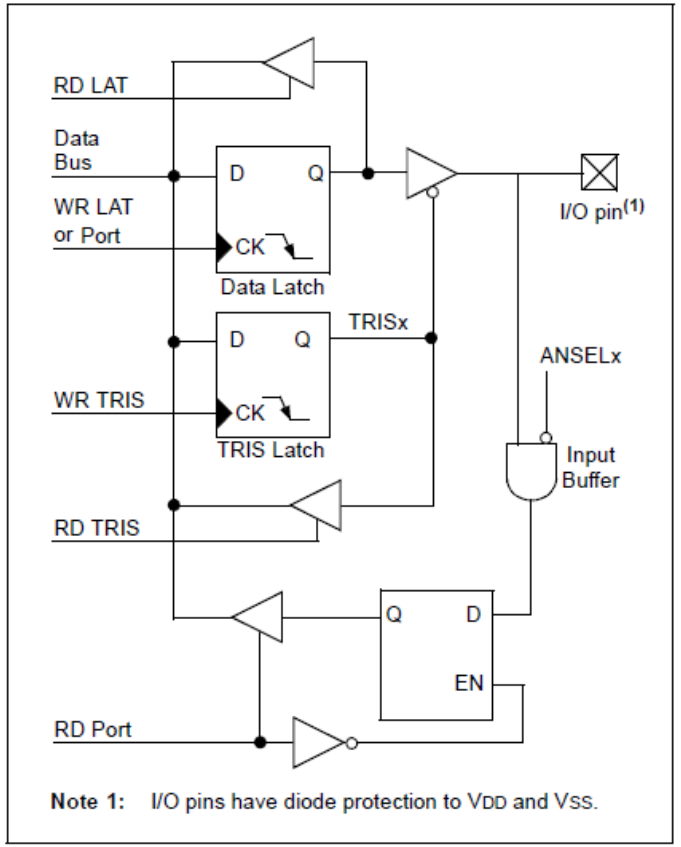
GPIO pin

- 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.

PORTx

Pokud je pin nastaven jako vstup, z tohoto registru lze přečíst stav příslušného pinu. Nemá tedy smysl do něj zapisovat!

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.

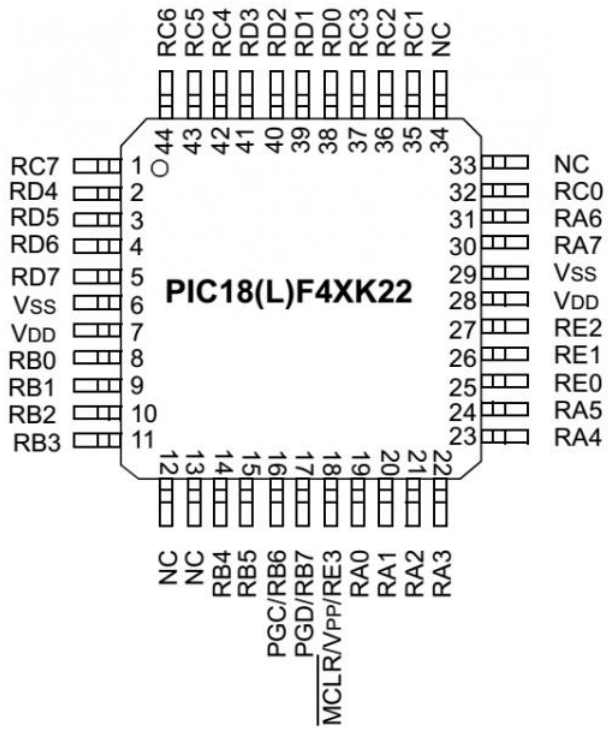
REGISTER 10-8: TRISx: PORTx TRI-STATE REGISTER(1)

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;
```

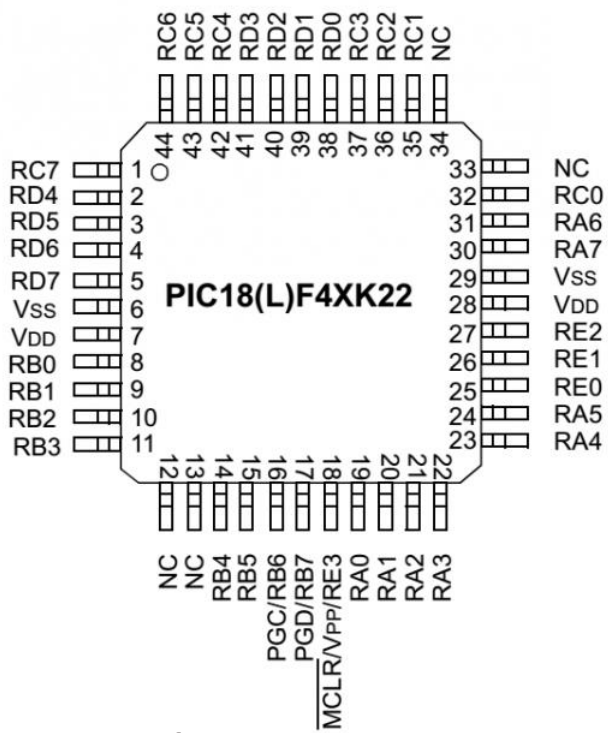
LATx

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
```

PORTx

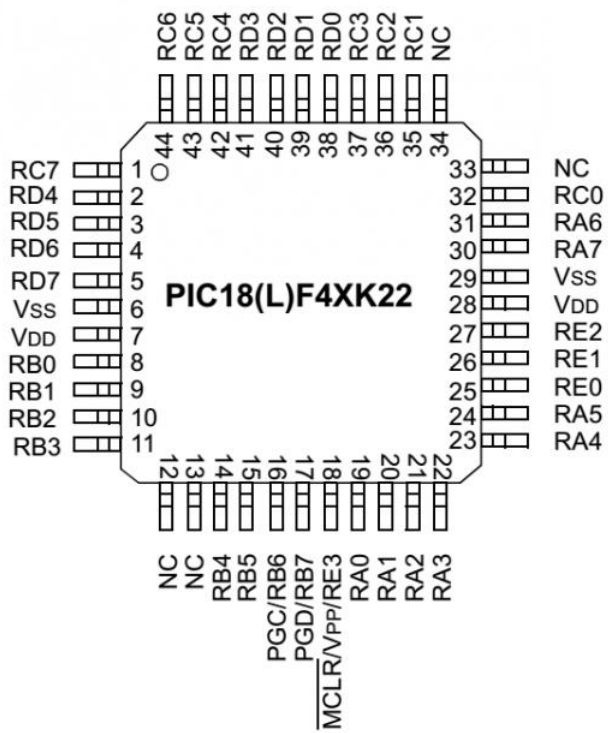
10.9 Register Definitions – Port Control

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 == 0){
        //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

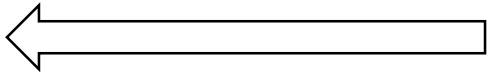
```
#define True     1
#define False    0
#define ledOn(led)      do{ led = False;}while(0)
#define ledOff(led)     do{ led = True;}while(0)
#define ledToggle(led)  do{ led = ~led;}while(0)
```

Na EduKitu se přečte zmáčknuté tlačítko jako logická 1.

Naopak LED diody svítí na logickou 0

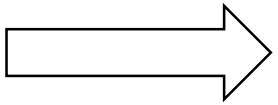
Inicializace a nastavení GPIO

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

asm(“nop”) je konstrukce umožňující zapsat assembler tedy v tomto případě instrukci procesoru nop, která trvá jeden cyklus a nedělá se nic.



```
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; asm("nop");
	LED2 = 1; asm("nop");
	LED3 = 1; asm("nop");
	LED4 = 1; asm("nop");
	LED5 = 1; asm("nop");
	LED6 = 1; asm("nop");
}
```

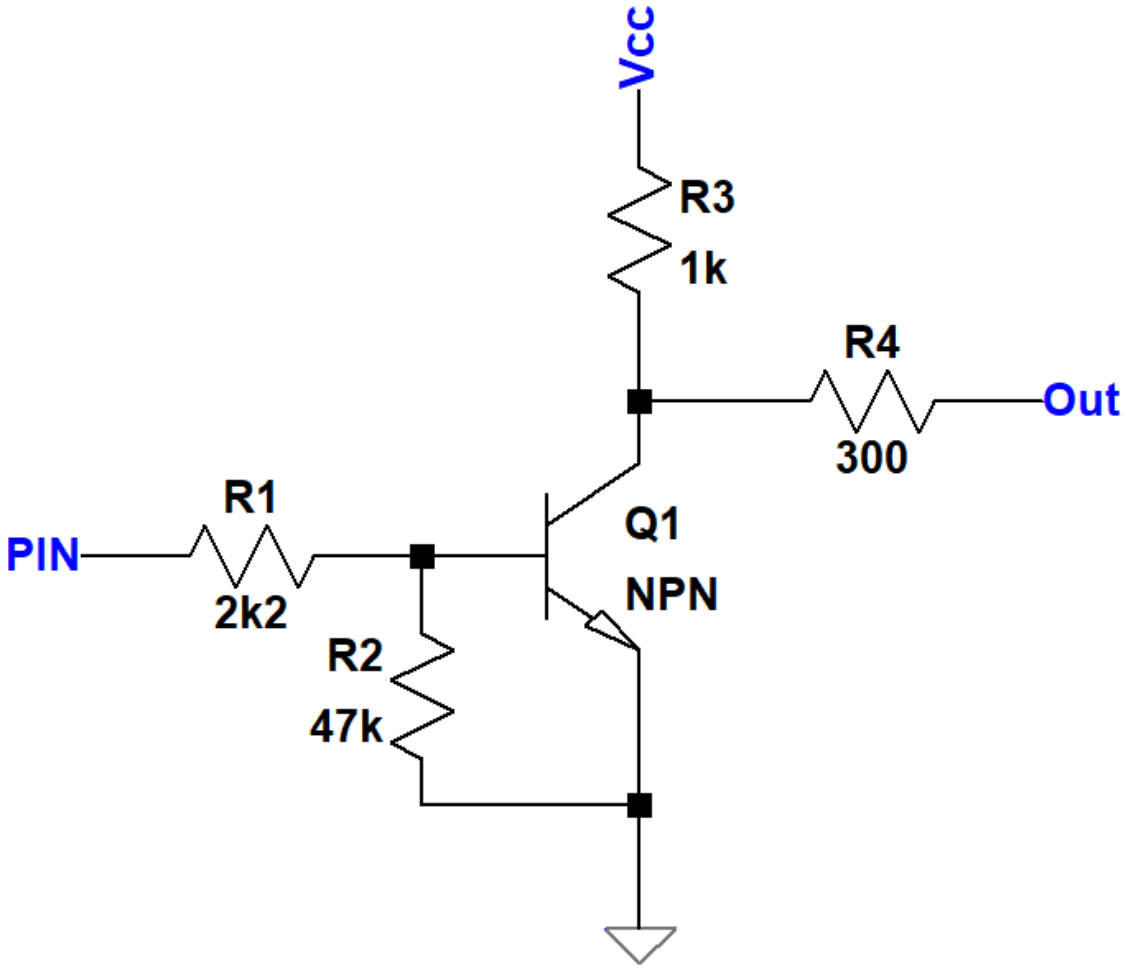
GPIO příklady

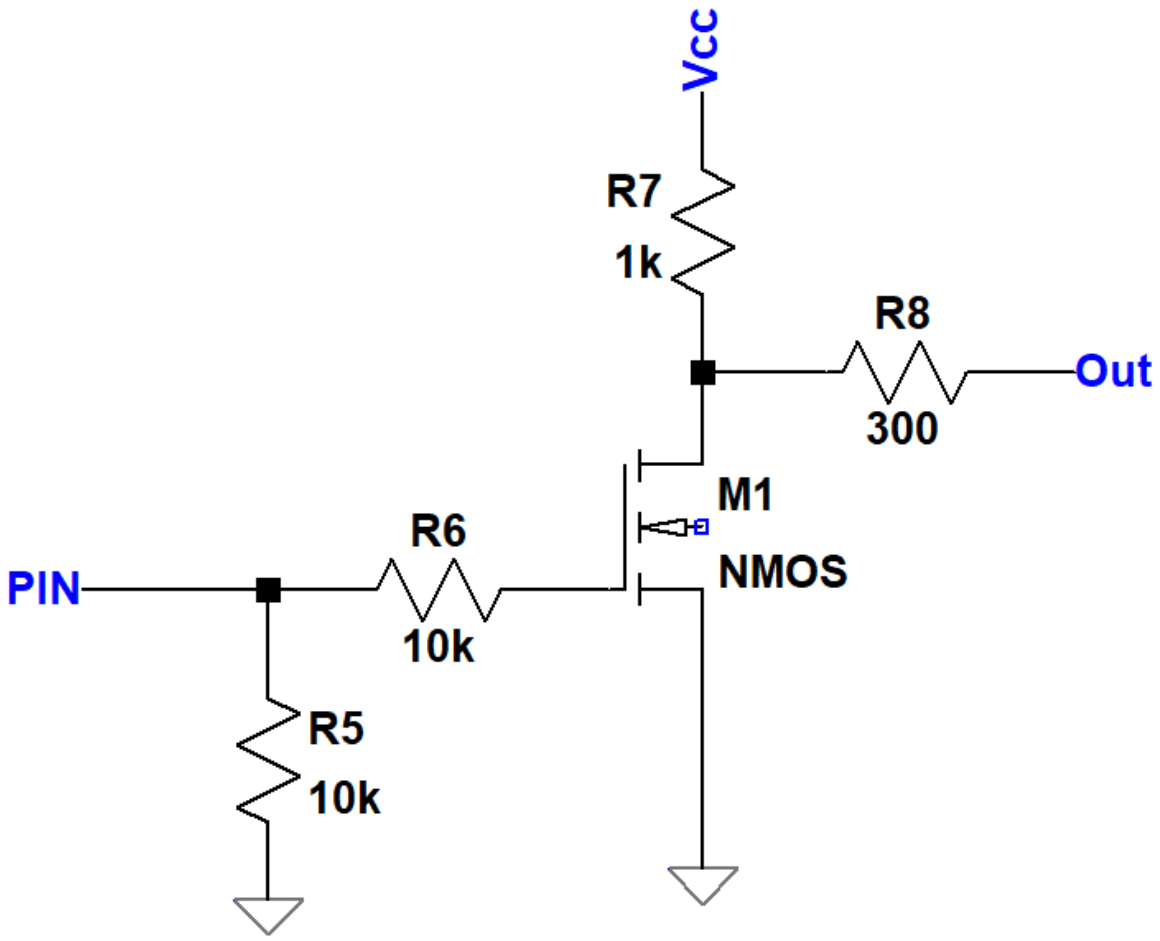
```
void main(void)
{
   init();
   unsigned char leds = 63;

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

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

Přiložený kód převrací stav ledky po zmáčknutí příslušného tlačítka

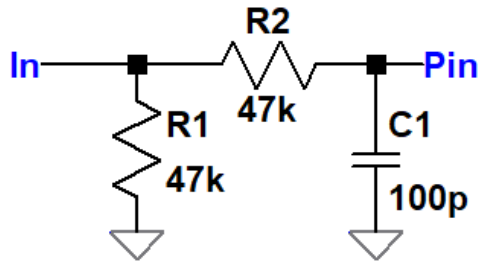
```
while(1){
    if(BTN1 | BTN2 | BTN3 | BTN4){
       __delay_ms(10);
       if(BTN1){
          ledToggle(LED1);
          while(BTN1);
       }
       else if(BTN2){
          ledToggle(LED2);
          while(BTN2);
       }
       else if(BTN3){
          ledToggle(LED3);
          while(BTN3);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

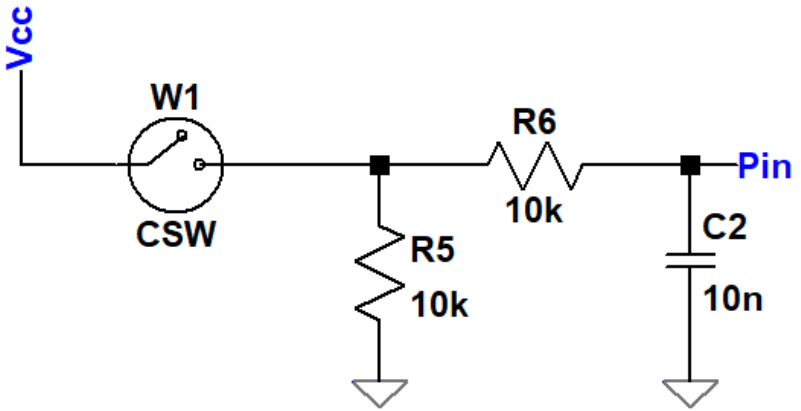
Hardware

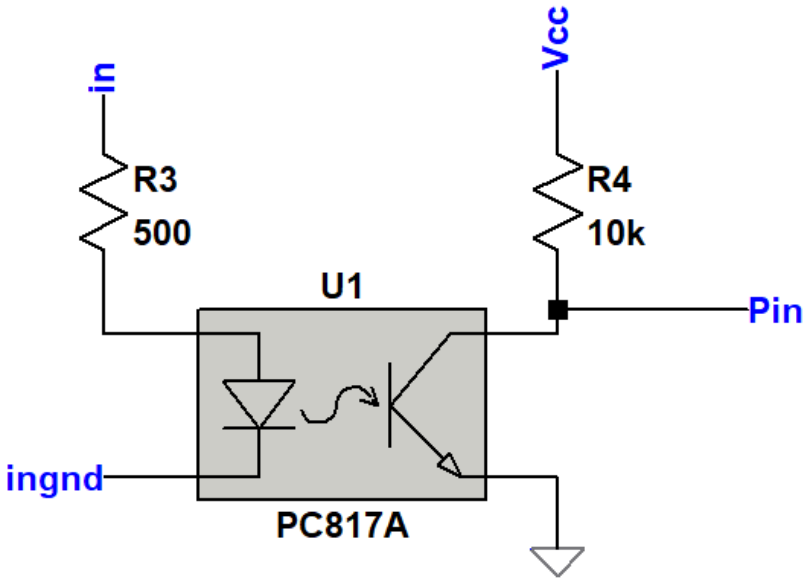

Vcc
R3
1k
R4
300
Out
R1
PIN
2k2
Q1
NPN
R2
47k

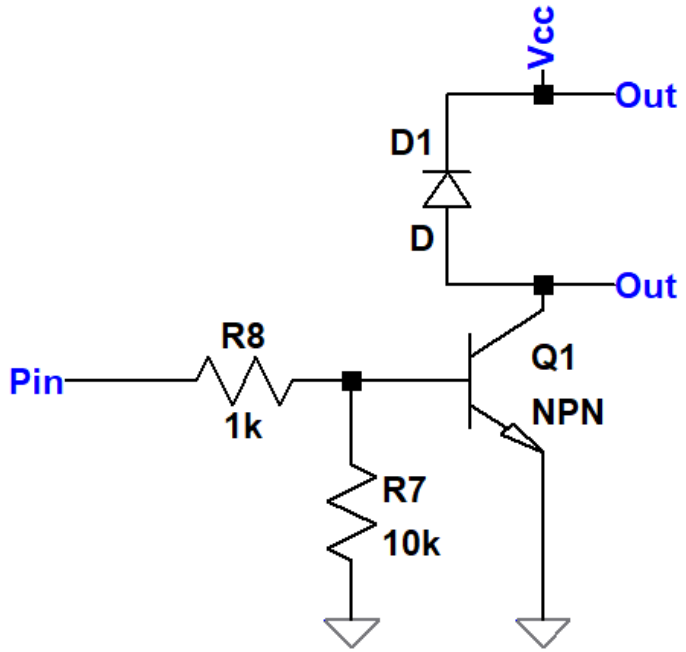

Vcc
R7
1k
R8
300
Out
M1
NMOS
R6
10k
PIN
R5
10k

Hardware


In
R2
47k
Pin
R1
47k
C1
100p


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


in
Vcc
R3
500
R4
10k
U1
Pin
ingnd
PC817A


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