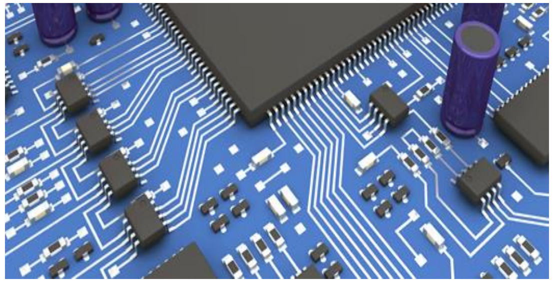
Aplikace Embedded systémů v Mechatronice



Michal Bastl
A2/713a

Aplikace Embedded systémů v Mechatronice

Obsah přednášky:

- Opakování
- Rekapitulace periferii
  - GPIO
  - TIMER
  - UART
- Analogově-digitální převod
- ADC periferie PIC18
- Nastavení ADC
- Ukázky použití
- Hardware poznámky

Opakování

Co víte o sběrnici UART?
Jak ji lze využít ke komunikaci s PC?
Co je EEPROM?
Co to znamená baudrate?

TRISx

Nastavuje zda bude pin vstup 1, nebo výstup 0.

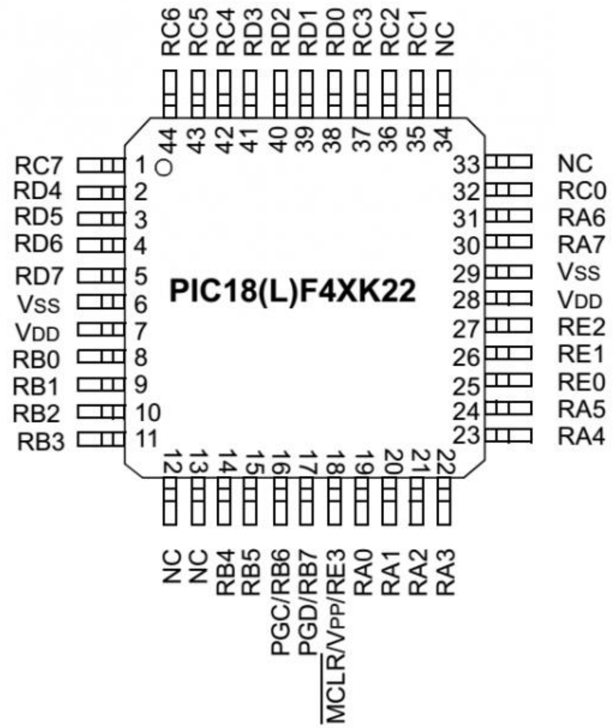
REGISTER 10-8: TRISx: PORTx TRI-STATE REGISTER[1]

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;
```

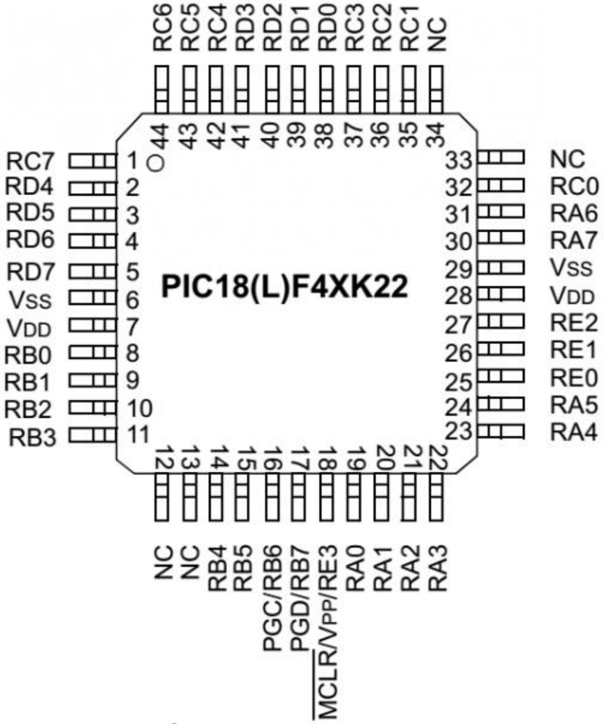
LATx

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

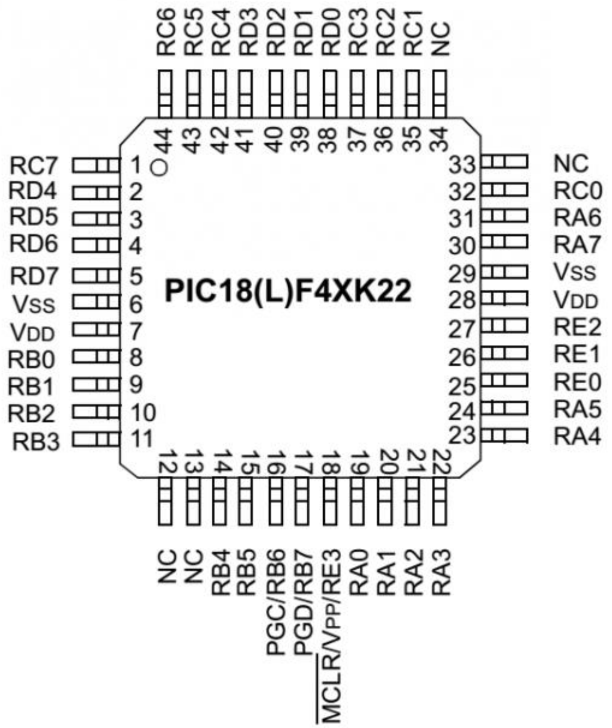
REGISTER 10-1: PORTX(1): PORTx REGISTER

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
}
```



Použití timeru

Nastavení timeru (inicializace periferie):

```
void init(void){

    // set pins as outputs
    TRISDbits.TRISD2 = 0;

    // Timer
    T1CONbits.TMR1CS = 0b00;   // TMR1 source (FOSC/4)
    T1CONbits.T1CKPS = 0b11;    // TMR1 prescaler (1:8)
    T1CONbits.TMR1ON = 1;        // TMR1 on
}
```

Použití v kódu:

```
void main(void)
{
    init();
    while(1){
        if(TMR1 >= 50000){
            LATDbits.LATD2 ^= 1;;
            TMR1 = 0;
        }
    }
}
```

Více zdrojů přerušení

```
T1CONbits.TMR1CS = 0b00;
T1CONbits.T1CKPS = 0b11;   // TMR1 prescaler
T1CONbits.TMR1ON = 1;      // TMR1 on

T5CONbits.TMR5CS = 0b00;
T5CONbits.T5CKPS = 0b11;   // TMR1 prescaler
T5CONbits.TMR5ON = 1;      // TMR1 on

/* init - interrupts */
PEIE = 1;          // global interrupt enable
GIE = 1;           // peripheral interrupt enable
TMR1IE = 1;        // enable TMR1 interrupt
TMR5IE = 1;        // enable TMR1 interrupt
```

```
void __interrupt() ISR(void){
  if(TMR1IE & TMR1IF){
      TMR1 = 0x8000;
      LED1 ^= 1;
      TMR1IF = 0;
  }

  if(TMR5IE & TMR5IF){
      TMR5 = 0;
      LED2 ^= 1;
      TMR5IF = 0;
  }
}
```

```
while(1){
      asm("NOP");
 }
```

Inicializace UART periferie

```
TRISCbits.TRISC6 = 0;   // uart TX as output
TRISCbits.TRISC7 = 1;   // uart RX as input

SPBRG = 25;     // (16_000_000 / (64 * 9600)) - 1

// final enable
RCSTAbits.SPEN = 1;      // enable UART peripheral
TXSTAbits.TXEN = 1;      // enable TX
RCSTAbits.CREN = 1;      // enable RX (aka Continuous receive)
```

Piny RX a TX je třeba v TRISx registru nastavit jako I/O

EXAMPLE 16-1: CALCULATING BAUD RATE ERROR

For a device with FOSC of 16 MHz, desired baud rate of 9600, Asynchronous mode, 8-bit BRG:

$$Desired\ Baud\ Rate = \frac{FOSC}{64([SPBRGHx:SPBRGx] + 1)}$$

Solving for SPBRGHx:SPBRGx:

$$X = \frac{\frac{FOSC}{Desired\ Baud\ Rate}}{64} - 1$$

$$= \frac{\frac{16000000}{9600}}{64} - 1$$

$$= [25.042] = 25$$

$$Calculated\ Baud\ Rate = \frac{16000000}{64(25 + 1)}$$

$$= 9615$$

$$Error = \frac{Calc.\ Baud\ Rate - Desired\ Baud\ Rate}{Desired\ Baud\ Rate}$$

$$= \frac{(9615 - 9600)}{9600} = 0.16\%$$

Uart je již „komplexnejší“ periferie.
Na PIC18 může fungovat i v synchronním módu
Omezíme se na základní nastavení!!

UART

Zápis dat na sběrnici:

```
while(1){
    while (TMR1 < DELAY);      // busy waiting for TMR1

    LATD2 = ~LATD2;
    TMR1 = 0;

    TXREG = '0' + i;          // char '0' + index
    i = (i == 9 ? 0 : i + 1);    // cycle 0...9
  }  // end of main loop
```

čtení dat a zápis dat na sběrnici:

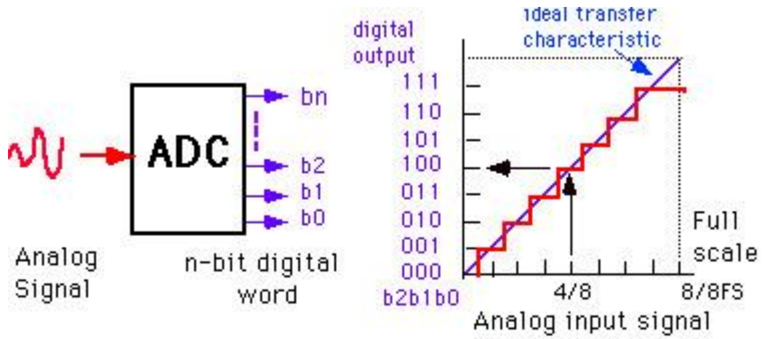
```
while(1){
    while(!PIR1bits.RCIF);  // waiting for data available flag
    LATB0 = ~LATB0;         // LED signal
    TXREG = RCREG;          // read byte and send it back
  }  // end of main loop
```

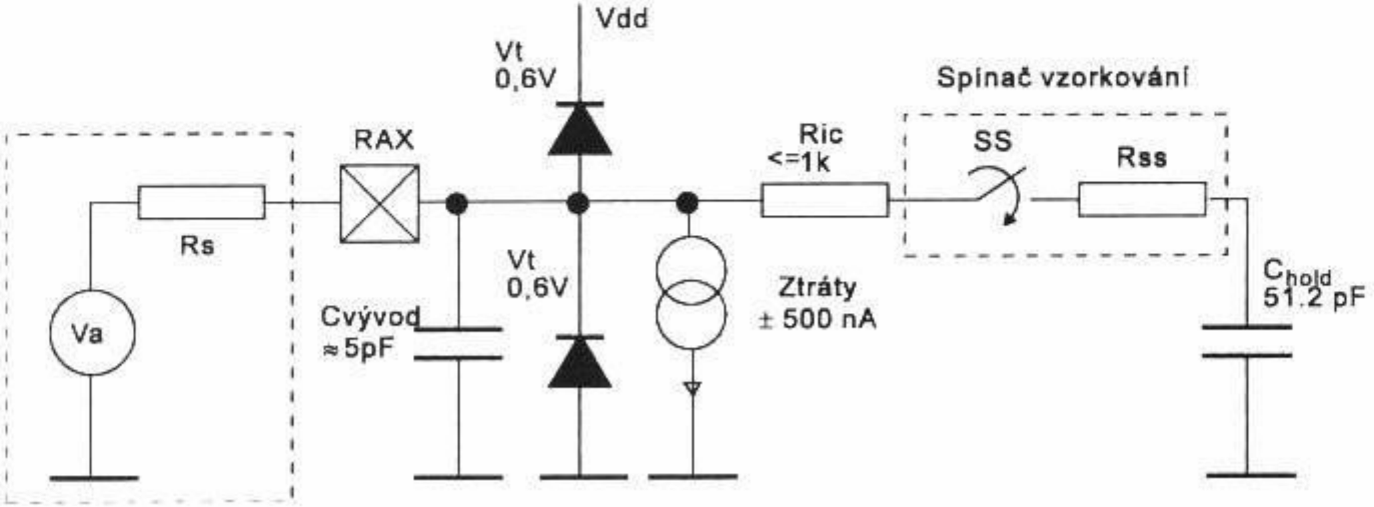
Pomocí přerušení:

```
RCIE = 1;        //RX interrupt enable
PEIE = 1;        // global interrupt enable
GIE = 1;         // peripheral interrupt enable

void __interrupt() handle(void){

    // check both interrupt enable & interrupt flag
    if (PIE1bits.RC1IE && PIR1bits.RCIF){
        TXREG = RCREG; // Transmit back one character
        LATD2 = ~LATD2;
        PIR1bits.RCIF = 0;
    }

    return;
}
```

Analogově digitální převod

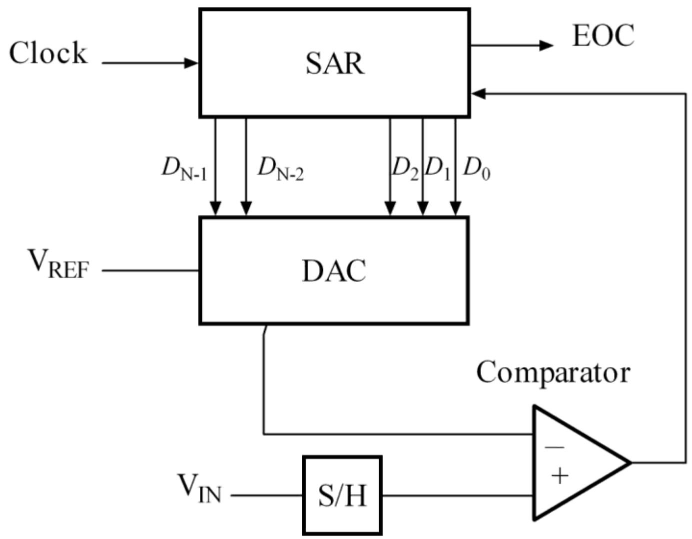
- Analogově digitální převodní převádí analogovou hodnotu napětí na její digitální interpretaci.
- Proto jedním ze základních parametrů je rozlišení převodníku.
- Jedná se o počet hladin na kterou dokáže převodník rozdělit interval referenčního napětí.
- ADC převodník na PIC 18 pracuje na principu postupné aproximace (Successive approximation ADC).



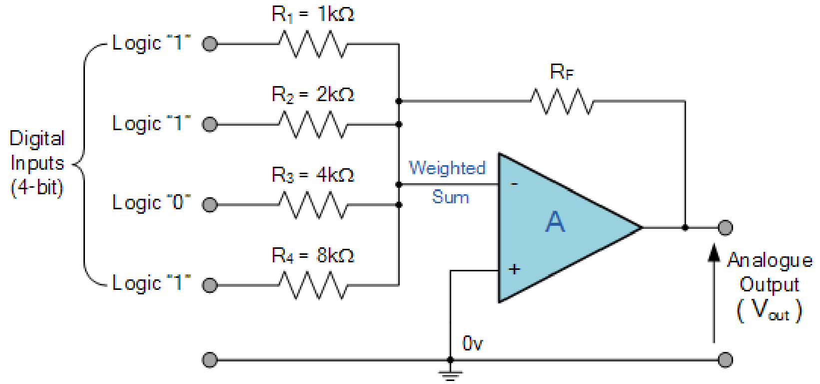


ADC převodník PIC

- ADC převodník MCU PIC18 se skládá z DAC převodníku, který slouží k nastavení požadované úrovně napětí
- Komparátorem je rozhodnuto, jestli napětí na AN pinu je větší nebo menší než nastavená hodnota
- Takto se projde všech 10 bitů
- Jedna se o analogii půlení intervalu



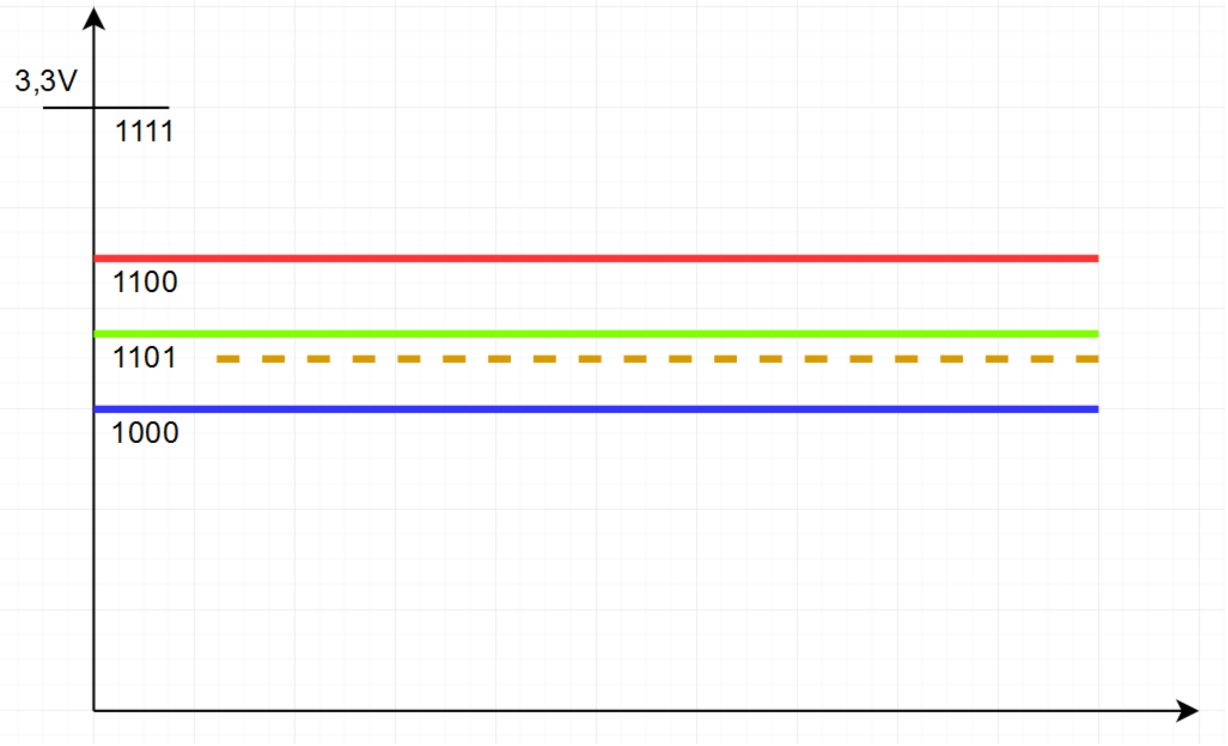
Zjednodušený DAC převodník:



$$U_{in} = ADC \cdot \frac{U_{ref}}{ADC_{res}}$$

ADC převodník PIC

- Převod probíhá od nejvyššího bitu ten vždy binárně rozdělí interval na polovinu.



ADC převodník PIC18

REGISTER 17-3: ADCON2: A/D CONTROL REGISTER 2

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	ACQT<2:0>			ADCS<2:0>		
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

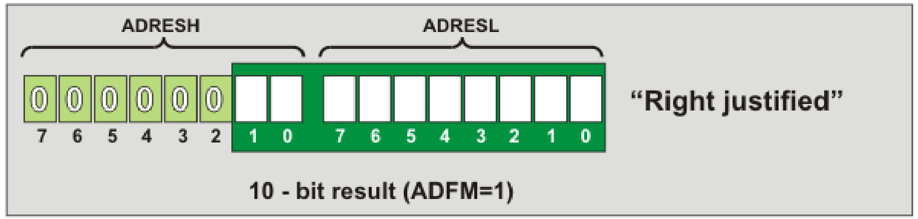
bit 7 **ADFM:** A/D Conversion Result Format Select bit
1 = Right justified
0 = Left justified

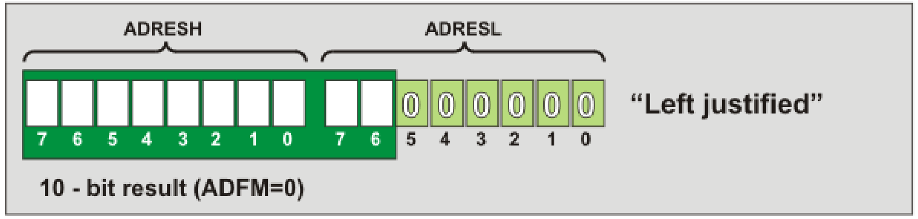
bit 6 **Unimplemented:** Read as ‘0’

bit 5-3 **ACQT<2:0>:** A/D Acquisition time select bits. Acquisition time is the duration that the A/D charge holding capacitor remains connected to A/D channel from the instant the GO/DONE bit is set until conversions begins.
000 = 0[(1)]
001 = 2 TAD
010 = 4 TAD
011 = 6 TAD
100 = 8 TAD
101 = 12 TAD
110 = 16 TAD
111 = 20 TAD

bit 2-0 **ADCS<2:0>:** A/D Conversion Clock Select bits
000 = FOSC/2
001 = FOSC/8
010 = FOSC/32
011 = FRC[(1)] (clock derived from a dedicated internal oscillator = 600 kHz nominal)
100 = FOSC/4
101 = FOSC/16
110 = FOSC/64
111 = FRC[(1)] (clock derived from a dedicated internal oscillator = 600 kHz nominal)

```
ANSELA = 0b00100000;        //AN4
ANSELE = 0b1;               //AN5

ADCON2bits.ADFM = 1;        //right justified
ADCON2bits.ADCS = 0b110;    //Fosc/64
ADCON2bits.ACQT = 0b110;    //16
ADCON0bits.ADON = 1;        //ADC turn on
```





ADC převodník PIC18

REGISTER 17-1: ADCON0: A/D CONTROL REGISTER 0

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	CHS<4:0>					GO/$\overline{\text{DONE}}$	ADON
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 **Unimplemented:** Read as '0'

bit 6-2 **CHS<4:0>: Analog Channel Select bits**
00000 = AN0
00001 = AN1
00010 = AN2
00011 = AN3
00100 = AN4
00101 = AN5(1)
00110 = AN6(1)
00111 = AN7(1)
01000 = AN8
01001 = AN9
01010 = AN10
01011 = AN11
01100 = AN12
01101 = AN13
01110 = AN14
01111 = AN15
10000 = AN16
10001 = AN17
10010 = AN18
10011 = AN19
10100 = AN20(1)
10101 = AN21(1)
10110 = AN22(1)
10111 = AN23(1)
11000 = AN24(1)
11001 = AN25(1)
11010 = AN26(1)
11011 = AN27(1)
11100 = Reserved
11101 = CTMU
11110 = DAC
11111 = FVR BUF2 (1.024V/2.048V/2.096V Volt Fixed Voltage Reference)(2)

bit 1 **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit
1 = A/D conversion cycle in progress. Setting this bit starts an A/D conversion cycle.
This bit is automatically cleared by hardware when the A/D conversion has completed.
0 = A/D conversion completed/not in progress

bit 0 **ADON:** ADC Enable bit
1 = ADC is enabled
0 = ADC is disabled and consumes no operating current

```
ANSELA = 0b00100000;        //AN4
ANSELE = 0b1;               //AN5

ADCON2bits.ADFM = 1;        //left justified
ADCON2bits.ADCS = 0b110;    //Fosc/64
ADCON2bits.ACQT = 0b110;    //16
ADCON0bits.ADON = 1;        //ADC turn on
```

```
ADCON0bits.CHS = 5;                 //select AN5
GODONE = 1;                         //start
while(GODONE);                      //wait until its done
pot1 = (ADRESH << 8) | ADRESL;      //combine registers
```

Hardware