

Convertorul analog-digital

a) Scopul lucrării

Familiarizarea cu convertorul analog-digital, parametri și funcționalități, precum și modul de lucru cu acesta.

b) Introducere

Un **convertor analog-digital (ADC)** este un dispozitiv electronic care transformă un semnal analogic (continuu) într-un semnal digital (discret).

Parametri importanți ai unui ADC

- **Rezoluție:** Numărul de biți ai ADC-ului, care determină precizia conversiei. De exemplu, un ADC pe 10 biți poate reprezenta 1024 de nivele discrete. Cele mai uzuale valori sunt 8, 10, 12 biți. La dsPIC-uri, rezoluția poate fi setată din regiștrii de control, astfel încât convertorul să lucreze pe 10 biți sau pe 12 biți.
- **Rata de eșantionare:** Cât de des se măsoară semnalul analogic. Trebuie să fie suficient de mare pentru a captura toate detaliile semnalului.
- **Liniaritate:** Cât de precis reproduce ADC-ului semnalul original.
- **Timp de conversie:** Timpul necesar pentru a finaliza conversia.

Modul de funcționare al unui ADC

1. **Eșantionare (Sampling):** Semnalul analogic este eșantionat la intervale regulate de timp. Eșantionarea se realizează prin circuite de eșantionare. Un singur circuit poate eșantiona într-un moment de timp doar un singur semnal analogic. Microcontrolerul dsPIC are 4 circuite de eșantionare la rezoluția de 10 biți și doar un singur circuit de eșantionare la rezoluția de 12 biți.

2. **Cuantizare (Quantization):** Fiecare eșantion este aproximat la cea mai apropiată valoare dintr-un set finit de nivele. Numărul de nivele depinde de rezoluția ADC-ului.
3. **Codare (Encoding):** Valoarea cuantizată este convertită într-un număr binar, care poate fi procesat de un sistem digital.

Pe microcontrolerul dsPIC, intrarea analogică, sau funcționarea de intrare analogică, este codificată cu *AN* și un *indice numeric*. În mod uzual, la activitățile din laborator se folosesc pinii RB2 și RB3 pentru aplicații cu conversiile AD.

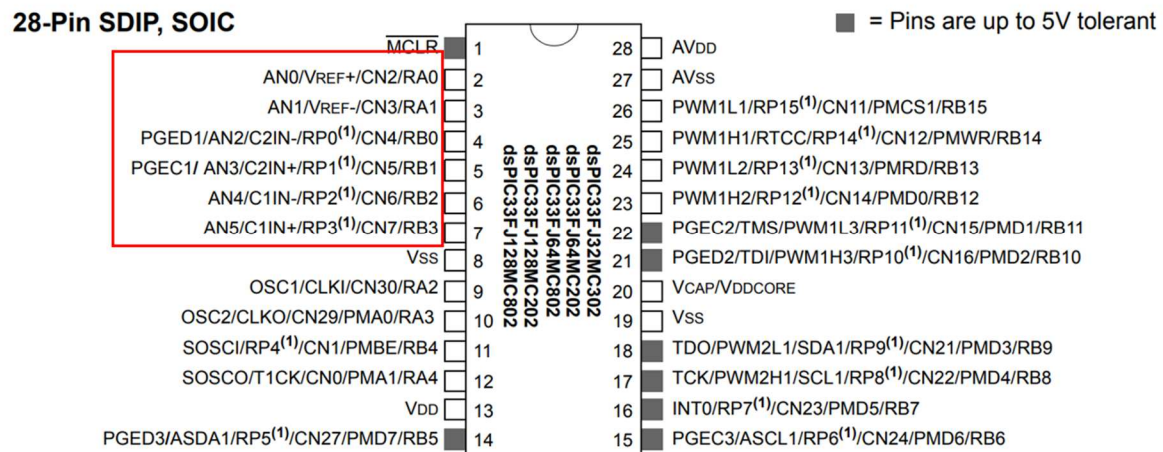


Figura 1. Capsula microcontrolerului dsPIC

c) Regiștrii de control ADC

1. AD1CON1: ADC1 CONTROL REGISTER 1

REGISTER 22-1: AD1CON1: ADC1 CONTROL REGISTER 1

R/W-0	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0
ADON	—	ADSIDL	ADDMABM	—	AD12B	FORM<1:0>	
bit 15						bit 8	

R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/C-0
SSRC<2:0>	—	SIMSAM	ASAM	SAMP	DONE		
bit 7						bit 0	

Legend:	HC = Cleared by hardware	HS = Set by hardware
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

Biții la care atragem atenția sunt următorii:

bit 15 ADON: *ADC Operating Mode bit* (este bitul prin care pornim lucrul cu converterul AD)

1 = ADC module is operating

0 = ADC is off

bit 10 AD12B: *10-bit or 12-bit Operation Mode bit* (bitul prin care setăm modul de lucru - pe 10 sau pe 12 biți)

1 = 12-bit, 1-channel ADC operation

0 = 10-bit, 4-channel ADC operation

bit 9-8 FORM: *Data Output Format bits* (bitul prin care setăm cum vrem să arate codul digital obținut în urma conversiei: întreg sau fracțional)

For 10-bit operation:

11 = Signed fractional (DOUT = sddd dddd dd00 0000, where s = .NOT.d)

10 = Fractional (DOUT = dddd dddd dd00 0000)

01 = Signed integer (DOUT = ssss sssd dddd dddd, where s = .NOT.d)

00 = Integer (DOUT = 0000 00dd dddd dddd)

For 12-bit operation:

11 = Signed fractional (DOUT = sddd dddd dddd 0000, where s = .NOT.d)

10 = Fractional (DOUT = dddd dddd dddd 0000)

01 = Signed Integer (DOUT = ssss sddd dddd dddd, where s = .NOT.d)

00 = Integer (DOUT = 0000 dddd dddd dddd)

Foarte important este să știm de unde să setăm cine ne va porni conversiile analog-digitale. Există posibilitatea să facem acest lucru din program sau să avem conversiile analog-digitale startate automat, dacă se poate, ori din exterior, ori din interior, de către un periferic al microcontrolerului. Putem selecta cu ajutorul bitului *SSRC* modalitatea prin care se va inițializa o conversie AD.

bit 7-5 SSRC: Sample Clock Source Select bits

111 = Internal counter ends sampling and starts conversion (auto-convert)

110 = Reserved

101 = Motor Control PWM2 interval ends sampling and starts conversion

100 = GP timer (Timer5 for ADC1) compare ends sampling and starts conversion

011 = Motor Control PWM1 interval ends sampling and starts conversion

010 = GP timer (Timer3 for ADC1) compare ends sampling and starts conversion

001 = Active transition on INT0 pin ends sampling and starts conversion

000 = Clearing sample bit ends sampling and starts conversion

Ultimul bit la care vom atrage atenția este bitul *ASAM*, care ajută la pornirea conversiei în mod automat. În mod uzual, în timpul lucrului de laborator îl vom seta pe 1 astfel încât, eșantionarea pentru următoarea conversie, eșantionarea tensiunii, să înceapă imediat când s-a terminat conversia anterioară.

bit 2 ASAM: ADC Sample Auto-Start bit

1 = Sampling begins immediately after last conversion. SAMP bit is auto-set.

0 = Sampling begins when SAMP bit is set

2. AD1CSSL: ADC1 INPUT SCAN SELECT REGISTER LOW

REGISTER 22-7: AD1CSSL: ADC1 INPUT SCAN SELECT REGISTER LOW^(1,2)

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0
—	—	—	—	—	—	—	CSS8
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CSS7	CSS6	CSS5	CSS4	CSS3	CSS2	CSS1	CSS0
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 15-9 Unimplemented: Read as '0'

bit 8-0 CSS: ADC Input Scan Selection bits

1 = Select ANx for input scan

0 = Skip ANx for input scan

CSSx = ANx, where x = 0 through 8.

Prin bitul CSS - Input Scan Select Register, indicăm de fapt de pe ce intrări analogice se vor realiza conversiile analog-digitale. Spre exemplu, pentru a seta convertorul AD pe pinul RB3, observăm de pe capsula microcontrolerului că pinului RB3 i se asociază intrarea analogică AN5, respectiv setăm CSS5 pe valoarea 1.

3. AD1PCFGL: ADC1 PORT CONFIGURATION REGISTER LOW

REGISTER 22-8: AD1PCFGL: ADC1 PORT CONFIGURATION REGISTER LOW^(1,2, 3)

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0
—	—	—	—	—	—	—	PCFG8
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PCFG7	PCFG6	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Legend:

R = Readable bit
-n = Value at POR

W = Writable bit
'1' = Bit is set

U = Unimplemented bit, read as '0'
'0' = Bit is cleared
x = Bit is unknown

bit 15-9 Unimplemented: Read as '0'

bit 8-0 PCFG: ADC Port Configuration Control bits

1 = Port pin in Digital mode, port read input enabled, ADC input multiplexor connected to AVSS

0 = Port pin in Analog mode, port read input disabled, ADC samples pin voltage

PCFGx = ANx, where x = 0 through 8.

Prin bitul *PCFG* indicăm ce intrări avem - analogice sau digitale. Spre exemplu, pentru a seta pinul RB3 ca intrare analogică, observăm de pe capsula microcontrolerului că pinului RB3 i se asociază intrarea analogică AN5, respectiv setăm *PCFG5* pe valoarea 0.

d) Inițiazare ADC pentru scanarea canalului AN4

```
void initAdc1(void)
{
    AD1CON1bits.AD12B = 1; // conversie AD pe 12 biti
    AD1CON1bits.FORM = 0; // rezultat conversie integer
    AD1CON1bits.SSRC = 2; // timerul 3 starteaza conversia
    AD1CON1bits.ASAM = 1; // incepe esantionarea noii valori imediat dupa
    // terminarea unei conversii
    AD1CON2bits.CSCNA = 1; // scaneaza intrarile pe CH0+ in timpul achizitiei A
    AD1CON2bits.CHPS = 0; // converteste doar CH0
    AD1CON2bits.SMPI = 0; // incrementeaza adresa DMA dupa terminarea fiecarei
    // conversii
    AD1CON3bits.ADRC = 0; // foloseste ceasul magistralei
    AD1CON3bits.ADCS = 63; // Timpul necesar unei conversii este de 22.4 us
    // Ceasul pentru conversia AD are formula  $T_{ad} = T_{cy} * (adcs + 1)$ 
    //  $T_{ad} = T_{cy} * (adcs + 1) = (1/40) * 64 = 1.6us$ 
    // Se seteaza intrarile analogice
    AD1CSSLbits.CSS4 = 1; // Selectam intrarea analogica AN4(RB2) pentru a fi
    // scanata
    // Scriem registrul de configurare al portului
    // Se va folosi doar registrul low al portului de configurare deoarece
    // dsPIC33fj128MC802
    // nu are implementati mai mult de 6 pini pentru ADC
    AD1PCFGL = 0xFFFF; // Setam toti pinii portului ADC1 pe modul digital,
    // si activeaza citirea la intrarea portului
    AD1PCFGLbits.PCFG4 = 0; // Setam pinul AN4(RB2) pe intrare analogica,
    // ADC verifica voltajele pe acel pin (achizitie AD)
    IFS0bits.AD1IF = 0; // Reseteaza flag-ul intreruperii convertorului AD
    IPC3bits.AD1IP = 6; // Seteaza prioritatea intreruperii convertorului AD
    IEC0bits.AD1IE = 1; // Permite intreruperea convertorului AD
```

```
AD1CON1bits.ADON = 1;}
```

e) Exemplu lucru cu conversie AD

main.c

```
#if defined(__dsPIC33F__)

#include "p33fxxxx.h"

#endif

#include "adcDrv1.h"

// Select Internal FRC at POR
_FOSCSEL(FNOSC_FRC);

// Enable Clock Switching and Configure
_FOSC(FCKSM_CSECMD & OSCIOFNC_OFF);

_FWDT(FWDTEN_OFF);          // Watchdog Timer Enabled/disabled by user software

void initPLL(void)
{
    // Configure PLL prescaler, PLL postscaler, PLL divisor
    PLLFBD = 41;              // M = 43 FRC
    //PLLFBFBD = 30;          // M = 32 XT
    CLKDIVbits.PLLPOST=0;     // N1 = 2
    CLKDIVbits.PLLPRE=0;      // N2 = 2

    // Initiate Clock Switch to Internal FRC with PLL (NOSC = 0b001)
    __builtin_write_OSCCONH(0x01); // FRC
    //__builtin_write_OSCCONH(0x03); // XT
    __builtin_write_OSCCONL(0x01);

    // Wait for Clock switch to occur
    while (OSCCONbits.COSC != 0b001); // FRC
    //while (OSCCONbits.COSC != 0b011); // XT

    // Wait for PLL to lock
    while(OSCCONbits.LOCK!=1) {};
```



```

    }

void main (void)
{
    // Disable Watch Dog Timer

    RCONbits.SWDTEN=0;

    // Peripheral Initialisation

    initPLL();

    initAdc1();           // Initialize ADC

    initTmr3();           // Initialise TIMER 3

    // Background Loop

    while (1)

    {

    }

}

adcDrv.c

#ifdef __dsPIC33F__
#include "p33fxxxx.h"
#endif

#include "adcDrv1.h"

#define SAMP_BUFF_SIZE      10           // Dimensiunea buffer-ului
in care se salveaza rezultatele conversiei

    // Timer-ul 3 este setat sa starteze conversia AD la fiecare 125 microsecunde
(8Khz Rate).

void initTmr3()
{

    TMR3 = 0;

    PR3 = 4999;

    T3CONbits.TON = 1; // Start Timer 3

}

int  ain4Buff[SAMP_BUFF_SIZE];

int  sampleCounter=0;

```

```

// rutina de tratare a intreruperii convertorului AD

void __attribute__((interrupt, no_auto_psv)) _ADC1Interrupt(void)
{
    ain4Buff[sampleCounter++]=ADC1BUF0;

    if(sampleCounter==SAMP_BUFF_SIZE)
        sampleCounter=0;

    IFS0bits.AD1IF = 0;          // Achita intreruperea convertorului AD
}

```

adcDrv.h

```

#ifndef __ADCDRV_H__
#define __ADCDRV_H__

// External Functions

extern void initAdc1(void);

extern void initTmr3();

extern void __attribute__((__interrupt__)) _ADC1Interrupt(void);

#endif

```