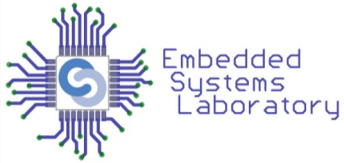


PROIECTAREA CU MICROPROCESOARE

Cursul 5
ADC

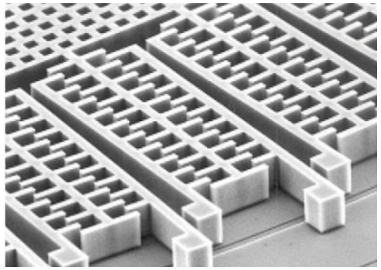
Facultatea de Automatică și Calculatoare
Politehnica București

Majoritatea semnalelor sunt analogice

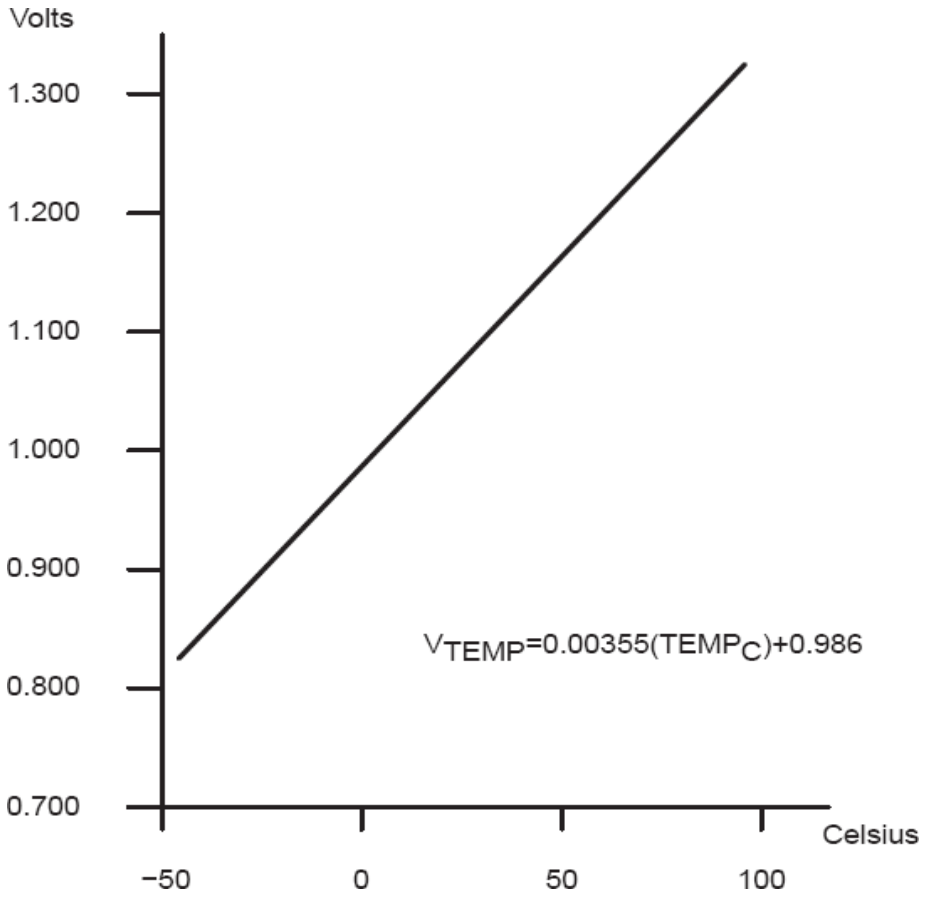
- Sunet, luminozitate, temperatură, presiune etc.
- Calea către un sistem digital
  - Sursă – tensiune variabilă – discretizare
- Traductor: transformă un tip de energie în altul
  - Electro-mecanic, optic, electric

Exemple

- Microfon/difuzor
- Termocuplu
- Accelerometru
- Senzor de lumină



Traductoarele transformă o formă de energie în alta

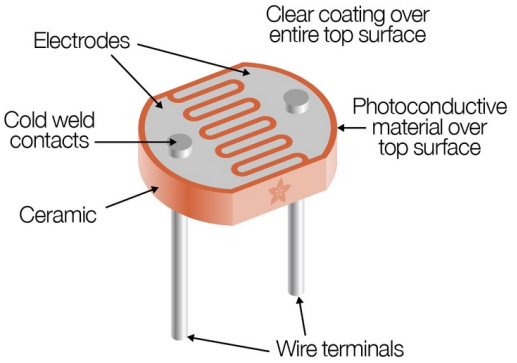


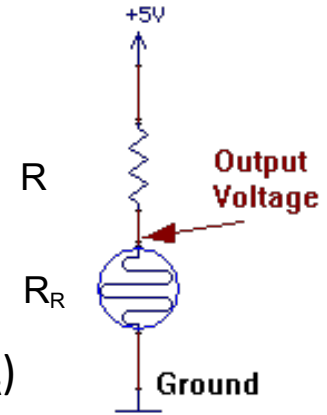
Exemplu: de la intensitate luminoasă la tensiune cu o celulă CdS

Alegem $R=R_R$ pentru o iluminare medie

CdS – ieftin de fabricat, consum redus

- $T_{RC} = (R+R_R)\ C_I$
- De obicei R_R = 50-200 kOhm
- C_I = 20 pF
- Deci, T_{RC} = 20-80 us
- f_{RC} = 10-50 kHz





$V_{signal} = (+5V)\ R_R/(R + R_R)$

Source: Forrest Brewer

Mulți alți senzori analogici

- Force.
  - Strain gauges - foil, conductive ink.
  - Conductive rubber.
  - Rheostatic fluids.
  - Piezorestive.
  - Piezoelectric films.
  - Capacitive force.
- Sound.
  - Microphones: current or charge.
  - Sonar: usually piezoelectric.
- Position.
  - Switches.
  - Shaft encoders.
  - Gyros.
- Atmospheric pressure.
- Acceleration
  - MEMS
  - Pendulum
- Monitoring
  - Battery energy
  - Motor velocity
  - Temperature
- Field
  - Antenna
  - Magnetic
    - Hall effect
    - Flux gate
- Location
  - Permittivity
  - Dielectric
- Conductivity
- Many, many more

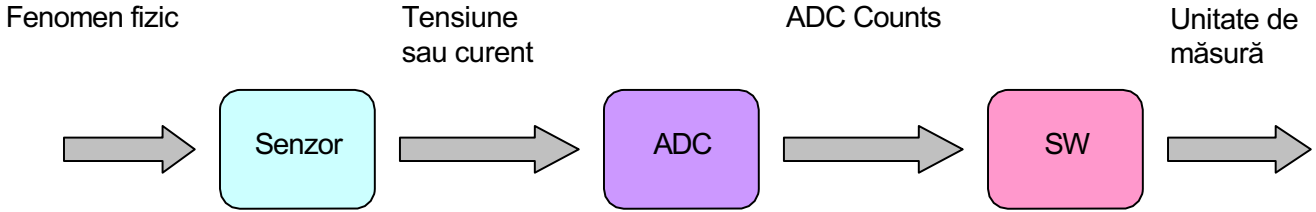
Puncte tari și slabe ale senzorilor

- De obicei senzorii produc informații relevante legate de mărimea măsurată
- În general, senzorii se comportă bine în condițiile de utilizare specificate în datasheet și prost în afara lor
- De ex. un senzor de temperatură ambientală poate măsura cu o precizie bună -20 .. 80 grade Celsius și deficitar în afara acestui interval. Un senzor de imagine se poate comporta prost în condiții de iluminare scăzută sau ceață

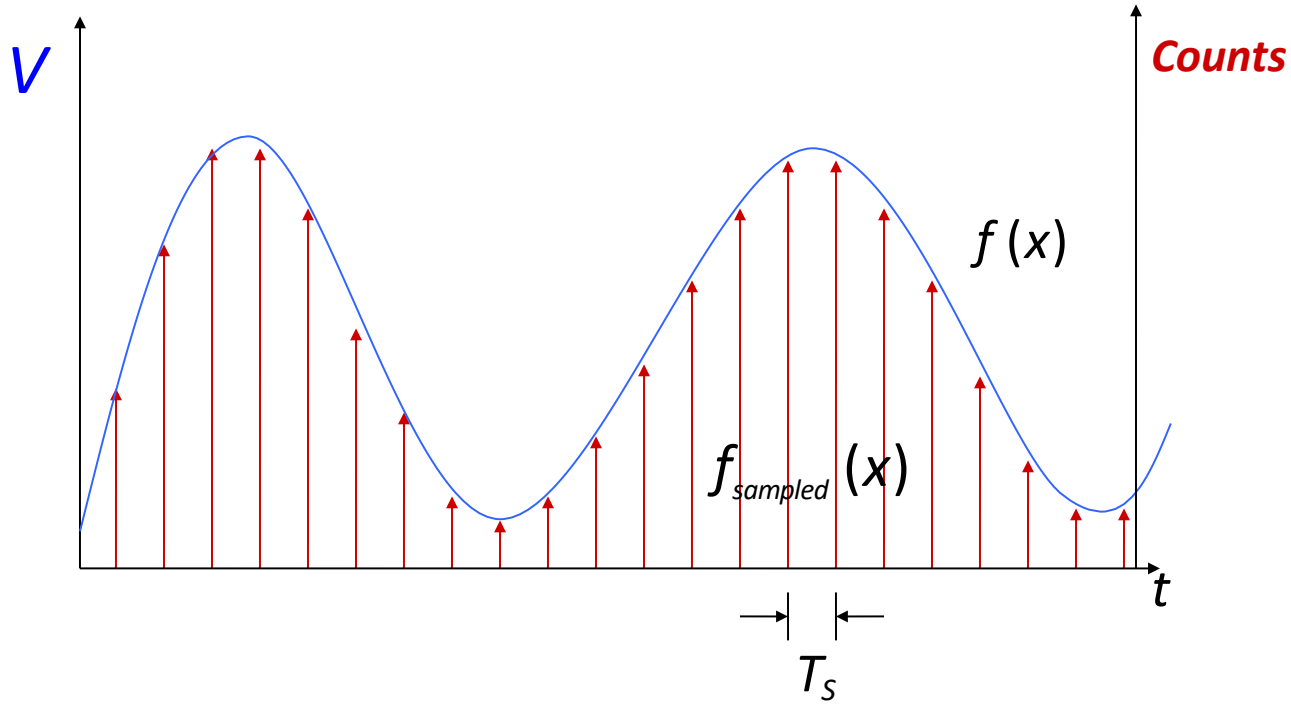
Analog to Digital

- Obiectivul unui sistem A/D

- Proces

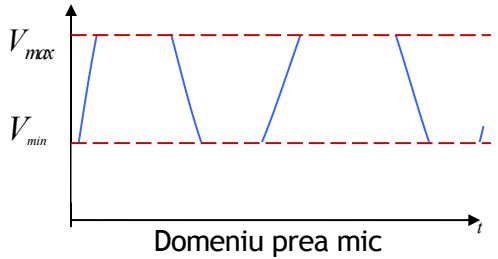


Reprezentarea digitală a unui semnal analogic

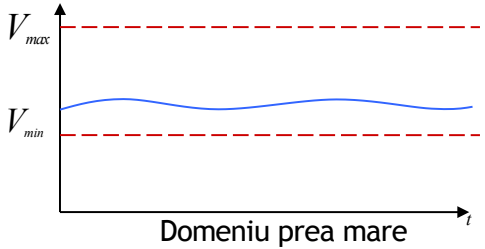


Selecția domeniului de valori

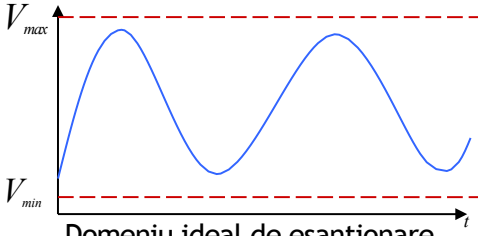
- Ce reprezintă valorile eșantionate?
- Întotdeauna vom eșantiona doar o (mică) parte din totalitatea semnalului



Domeniu prea mic



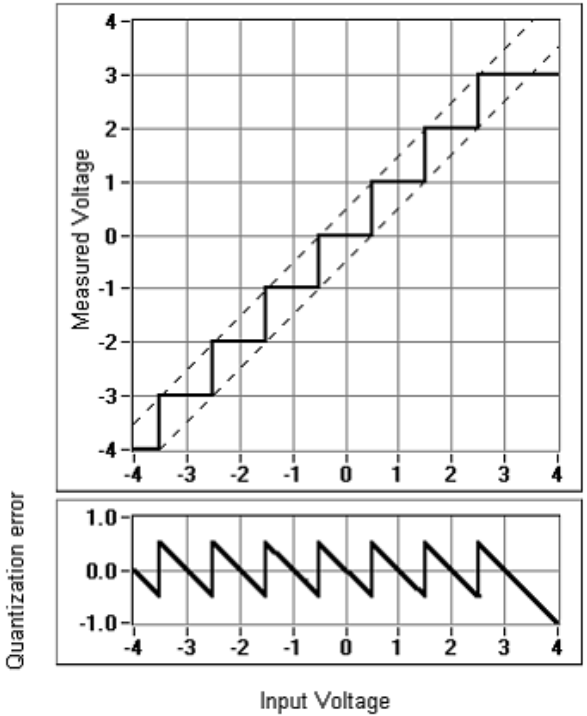
Domeniu prea mare



Domeniu ideal de eșantionare

Alegerea rezoluției corecte

- Rezoluție
  - Numărul valorilor discrete dintr-un interval de măsură
    - Ex.: 12-bit ADC – 4096 valori
    - Pas = Domeniu / 4096
- Eroare de cuantificare
  - Cât de departe este valoarea discretă de valoarea efectivă
  - ½ LSB = Domeniu / 8192
  - Cu cât este mai mare domeniul, cu atât mai mică e rezoluția

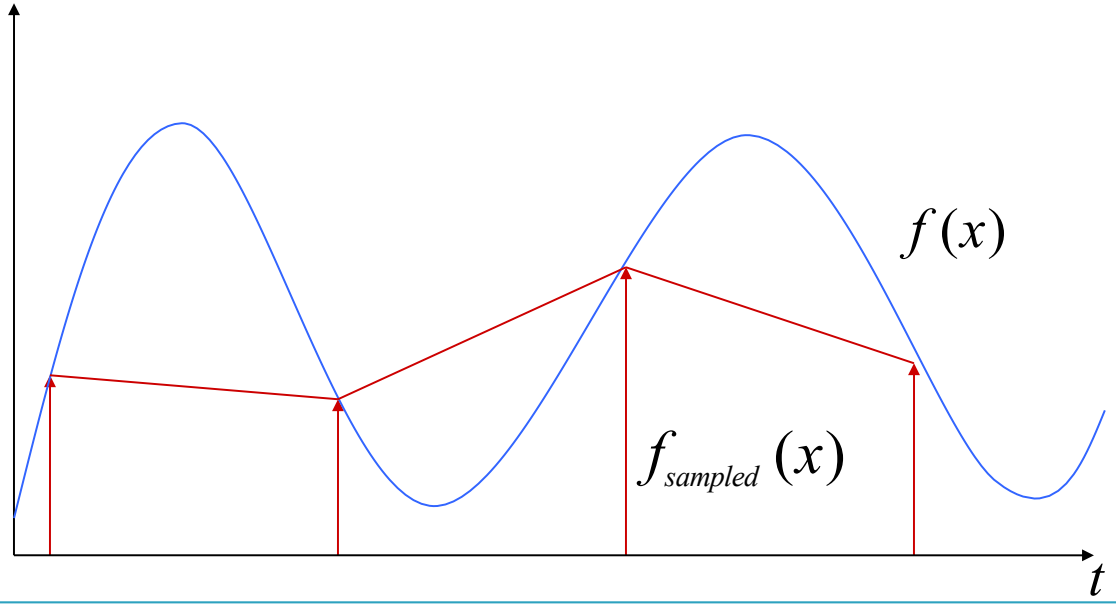


Alegerea rezoluției temporale

Prea scăzută: nu putem reconstitui semnalul original.

Prea ridicată: irosim putere de calcul, energie, resurse.

Teorema Nyquist-Shannon!



Teorema Shannon-Nyquist pentru eșantionare

Dacă un spectru de frecvențe continuu $f(t)$ nu conține frecvențe mai mari de f_{max}, atunci acesta poate fi complet reconstituit prin eșantioane discrete luate la frecvența:

$$f_{\text{eșantionare}} > 2 f_{\max}$$

Exemplu:

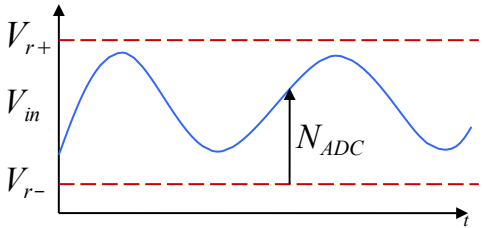
- Oamenii pot percepe semnale auditive între 20Hz și 20kHz
- CD audio: eșantionat la 44.1kHz

Dezavantaj: dacă semnalul conține zgomot de frecvență ridicată, folosirea unei rate de eșantionare mai mari de 2 x f_{max} poate să fie folositoare

- Permite filtrarea trece-jos, îmbunătățește acuratețea
- Poate fi uneori reparat prin filtrarea semnalului înainte de măsurare

Exemplu: conversia dintre valori analogice, valori ADC și unități de măsură

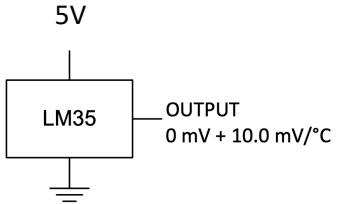
- Conversie: valori ADC – valori tensiune



Dacă avem un ADC cu rezoluție 10 biți (ATmega324p)

$$N_{ADC} = 1023 \times \frac{V_{in} - V_{r-}}{V_{r+} - V_{r-}} \qquad V_{in} = N_{ADC} \times \frac{V_{r+} - V_{r-}}{1023}$$

- Conversie: tensiune – unități de măsură



LM35 senzor analogic temperatură: $V_{OUT} = TEMP_{°C} \times 0.01V$

$$N_{ADC} = 1023 \times \frac{V_{OUT} - 0V}{5V - 0V} = TEMP_{°C} \times 0.01 \times \frac{1023}{5} = TEMP_{°C} \times 2.046$$

$$TEMP_{°C} = \frac{N_{ADC}}{2.046}$$

Câteva observații

- Câteva greșeli des întâlnite atunci când se fac conversii ADC

$$TEMP_{℃} = \frac{N_{ADC}}{2.046} \qquad V_{in} = N_{ADC} \times \frac{V_{r+} - V_{r-}}{1023}$$

```
volatile const int adccount;
float vtemp = adccount / 1023 * 5;
float tempc = adccount / 2.046;
```

Trebuie să cunoașteți regulile pentru asociativitate și conversie între tipurile de date din ANSI C!

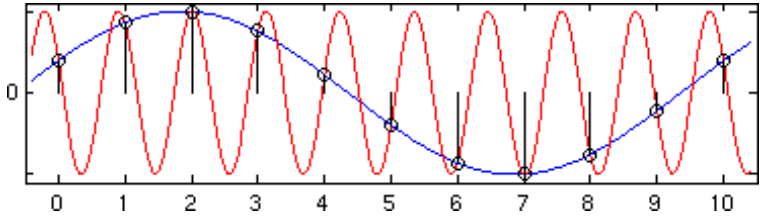
Exemplu:

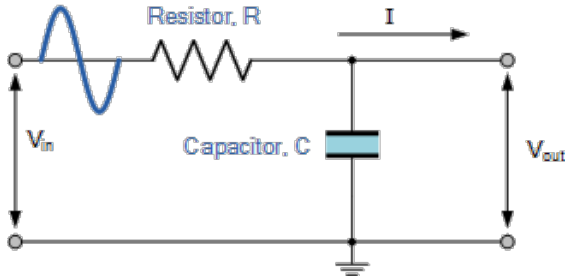
- Aritmetica în virgulă fixă
  - Cazurile de overflow și underflow pot da bătăi de cap
  - Pentru un dynamic range bun, poate fi folosită o scară logaritmică a reprezentării
- Aritmetica în virgulă mobilă
  - Deseori este emulată software (ceea ce este și cazul la ATmega324p)
  - Emularea este lentă și costisitoare ca energie pe procesoarele embedded

Folosirea filtrelor anti-aliasing pe intrările ADC

Aliasing

- Semnale de frecvență diferită sunt de nedistins atunci când sunt eșantionate
- Condiționarea intrării folosind un filtru trece-jos
  - Înlătură componentele de frecvență ridicată (a.k.a. filtru anti-aliasing)





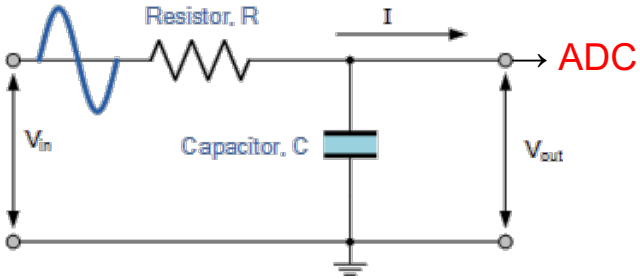
$$f_{tăiere} = \frac{1}{2\pi RC}$$

Exemplu

Filtru simplu trece-jos pentru un semnal analogic ce conține un semnal audio de la un microfon aplicat pe intrarea ADC

Eșantionarea trebuie să se facă la 40ksps (40000 eșantioane pe secundă)

Ce valori alegem pentru R și C pentru a pre-filtra orice semnal mai mare de 20kHz?



$$f_{tăiere} = 20kHz = \frac{1}{2\pi RC}$$

$$RC = \frac{1}{2\pi \times 20000} \cong 8 \times 10^{-6}$$

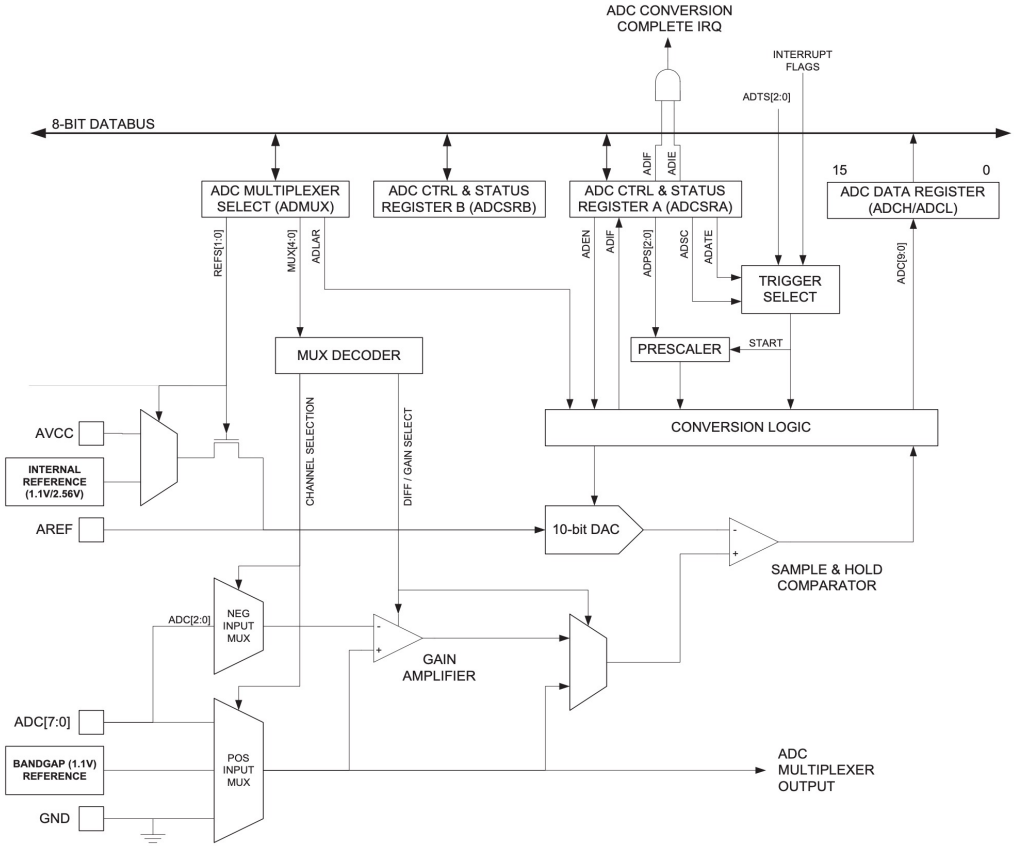
$$Dacă\ R = 100\Omega, atunci\ C = 8 \times 10^{-8}F = 80nF$$

Chiar trebuie să filtrez semnalul înainte de ADC?

- Da, de cele mai multe ori
- Multe ADC-uri au un filtru analogic pe intrare din construcție
- Aceste filtre au frecvența de tăiere un pic mai sus de ½ din rata **maximă** de eșantionare
  - E minunat atunci când folosiți rata maximă de eșantionare, nu e așa de util atunci când eșantionați la o frecvență mai mică
- Se pot folosi uneori defazări aleatorii ale eșantioanelor pentru a evita probleme de alising, dar depinde de tipul de ADC folosit

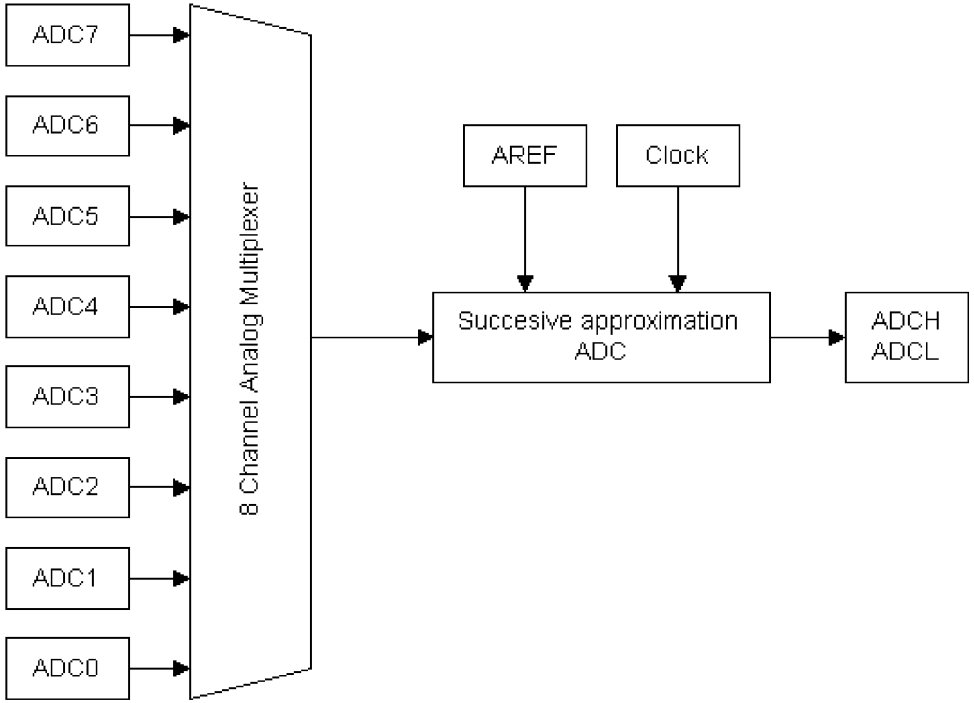
ATmega324p

- 10 biți rezoluție
  - 1024 valori
- Domeniu 0-5V
  - 4.88mV rezoluție
  - 2LSB precizie (9.76mV)



Funcționare

- 8 intrări multiplexate
  - Single ended sau diferențial cu câștig preconfigurabil de 1x, 10x sau 100x
- Referință de tensiune selectabilă
  - 2.56V și 1.1V (intern)
  - Vcc (3.3V) extern
- Mod de funcționare free running sau single conversion
  - Poate genera întrerupere la conversie
- Timp conversie 13-260μs
  - Conversie lentă = precizie crescută



Registre

ADMUX

- REFS – referința de tensiune
- ADLAR – rezultatul aliniat stânga sau dreapta
- MUX – selecția canalului de pe care se face conversia

ADCSRA

- ADEN – enable global ADC
- ADSC – scrieți 1 pentru pornirea unei conversii
- ADATE – scrieți 1 pentru auto triggering
- ADIF – se pune pe 1 când o conversie este finalizată
- ADIE – activarea întreruperii de conversie completă
- ADPS – prescaler pentru aproximațiile succesive ale rezultatului

ADCSRB

- ADTS – sursa trigger

ADC

- Aici se găsește rezultatul conversiei

ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
(0x7B)	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADCL and ADCH – The ADC Data Register

Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	

Exemplu cu polling

- Funcții simple de inițializare și citire ADC
  - Canal selectabil ca parametru
  - Citire cu busy waiting!

```c
void initADC(uint8_t channel)
{
  /*default to input 0 if bad channel*/
  if(channel > 7) channel = 0;

  /*Vcc as reference, channel as input*/
  ADMUX = (1 << REFS0) | channel;

  /* ADC enable */
  ADCSRA = (1 << ADEN);
}

uint16_t adcRead()
{
  ADCSRA |= (1 << ADSC); /* start conversion*/

  /* wait until conversion is complete*/
  loop_until_bit_is_set(ADCSRA, ADIF);
  return ADC;
}

int main()
{
  uint16_t adcValue;
  initUART();

  /* init ADc to read from channel 0 */
  initADC(0);

  while(1)
  {
    adcValue = adcRead();
    printf("ADC Value: %d\n", adcValue);
  }
  return 0;
}
```

Exemplu întreruperi

- Întreruperea este declanșată la terminarea unei conversii
  - Dar tot trebuie să declanșăm manual o conversie de fiecare dată!
- Observați folosirea variabilei globale *adcValue*

```
volatile int16_t adcValue = -1;

ISR(ADC_vect)
{
  adcValue = ADC;
}

void initADC(uint8_t channel)
{
  /*default to input 0 if bad channel*/
  if(channel > 7) channel = 0;

  /*Vcc as reference, channel as input*/
  ADMUX = (1 << REFS0) | channel;

  /* ADC enable & interrupt enable */
  ADCSRA = (1 << ADEN) | (1 << ADIE);
}

int main()
{
  uint16_t adcValue;
  initUART();

  /* init ADC to read from channel 0 */
  initADC(0);
  sei();

  while(1)
  {
    ADCSRA |= (1 << ADSC); /* trigger conversion*/

    if(val != -1)
      printf("ADC Value: %d\n", adcValue);

    _delay_ms(1000);
  }
  return 0;
}
```

ADC Auto-Triggering

- Permite declanșarea automată de conversii la apariția unui eveniment
- Surse posibile evenimente (biții ADTS din ADCSRB):
  - Free-running (conversii în rafală)
  - Comparator analogic
  - Întreruperea externă INT0
  - Timer0 Compare Match sau Overflow
  - Timer1 Compare Match, Overflow sau Capture

Exemplu auto-trigger

- Folosim Timer/Counter1 pentru a declanșa conversii succesive la fiecare 100ms
- Legăm sursa trigger ADC din ADCSRB la eveniment CTC Timer1
- După primele 100ms de rulare, sau la schimbarea canalului, cea mai nouă valoare a conversiei se va găsi întotdeauna în ADC

```c
void initADC(uint8_t channel)
{
  /*default to input 0 if bad channel*/
  if(channel > 7) channel = 0;

  /*Vcc as reference, channel as input*/
  ADMUX = (1 << REFS0) | channel;

  /* ADC enable, 128 prescaler and auto-trigger enable */
  ADCSRA = (1 << ADEN) | (5 << ADPS0) | (1 << ADATE);

  /* Trigger event: Timer1 compare channel B */
  ADCSRB = (5 << ADTS0);
}

void initTimer1()
{
  /* CTC mode, TOP at OCR1A, 1024 prescaler*/
  TCCR1B = (1 << WGM12) | (5 << CS0);

  /* set value for 10Hz (100ms) */
  OCR1B = 12000000UL/1024/10 - 1;
  /* ADC trigger is on OCR1B */
  OCR1A = OCR1B;
}

int main()
{
  initUART();

  /* init ADC to read from channel 0 */
  initADC(0);
  initTimer1();

  while(1)
  {
    /* after first 100ms, ADC is always updated with newest value*/
    printf("ADC Value: %d\n", ADC);

    _delay_ms(1000);
  }
  return 0;
}
```

Exemplu complex

- Ce facem dacă avem mai mulți senzori analogici conectați la uC?
  - Citirea lor ciclică presupune constrângeri suplimentare
- Putem să cerem o conversie de fiecare dată când avem nevoie de date de la senzori
  - Overhead pentru citire
- Sau putem să facem conversii ciclice și să citim la nevoie
  - Citire (aproape) instantanee, dar folosim resurse suplimentare (un timer)

Exemplu 1

- La fiecare 10ms se declanșează conversii pentru toate canalele ADC configurate prin *NRSENSORS*
- Vectorul *adc_values[]* va conține periodic valori noi pentru fiecare canal

```
#define NRSENSORS 5
volatile uint8_t channel = 0;
volatile uint8_t adc_values[NRSENSORS]

ISR(TIMER1_COMPA_vect) /* triggers a new conversion every 10ms */
{
  ADCSRA |= (1 << ADSC);
}

ISR(ADC_vect) /* reads progressively ADC channels until adc_values[] is full */
{
  adc_values[channel] = ADC;
  channel = (channel + 1) % NRSENSORS;
  ADMUX = (ADMUX & 0xE0) | channel;
  if(channel != 0)
    ADCSRA |= (1 << ADSC);
}

void initADC()
{
  ADMUX = 1 << REFS0;  /* AVCC reference */
  ADCSRA = (1 << ADEN) | (1 << ADIE) | (5 << ADPS0);
}

void initTimer1()
{
  TCCR1B = (1 << WGM12) | (5 << CS10);  /* CTC, TOP OCR1A, prescaler 1024 */
  TIMSK1 = 1 << OCIE1A;                 /* compare interrupt */

  OCR1A = 12000000UL/1024/10 - 1; /* 10Hz - 100ms */
}

int main()
{
  ...
}
```

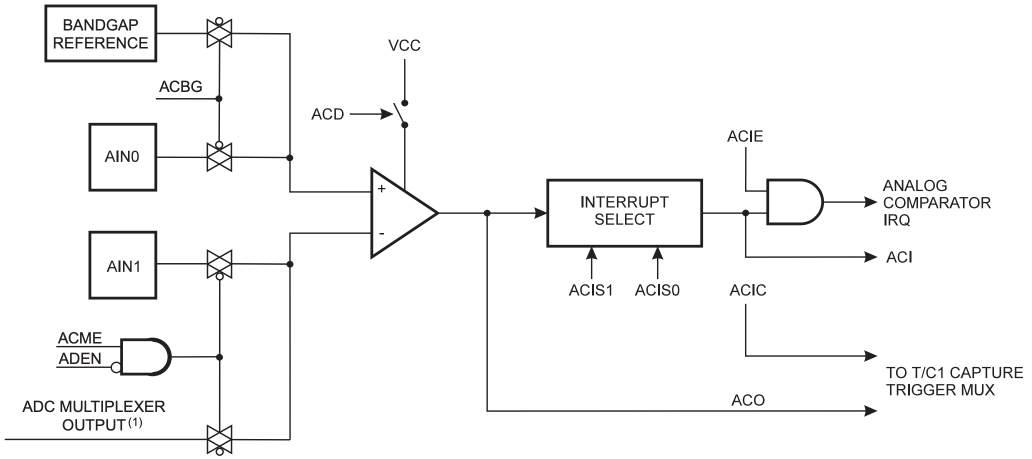
Exemplul 2

- Vectorul *adc_values[]* va conține periodic valori noi pentru fiecare canal
- De data aceasta, conversiile sunt pornite la Timer1 Output Compare, dar fără întrerupere

```
#define NRSENSORS 5
volatile uint8_t channel = 0;
volatile uint8_t adc_values[NRSENSORS]

ISR(ADC_vect) /* reads progressively ADC channels until adc_values[] is full */
{
  adc_values[channel] = ADC;
  channel = (channel + 1) % NRSENSORS;
  ADMUX = (ADMUX & 0xE0) | channel;
  if(channel != 0)
    ADCSRA |= (1 << ADSC);
}

void initADC()
{
  ADMUX = 1 << REFS0;  /* AVCC reference */
  ADCSRA = (1 << ADEN) | (1 << ADATE) | (1 << ADIE) | (5 << ADPS0);
  ADCSRB = 6 << ADTS0; /* trigger Timer1 overflow */
}

void initTimer1()
{
  TCCR1A = (3 << WGM11) ;  /* Fast PWM, TOP OCR1A, prescaler 1024 */
  TCCR1B = (3 << WGM12) | (5 << CS10);

  OCR1A = 12000000UL/1024/10 - 1; /* 10Hz - 100ms */
}

int main()
{
  ...
}
```

Comparatorul Analogic

- Periferic simplu, dar care poate fi foarte util în anumite situații
- Compară două tensiuni analogice
  - Sau o tensiune cu o valoare de referință
- Rezultatul comparației poate fi citit din cod sau poate să declanșeze un eveniment intern



Registre și biți

- ACME: Analog Comparator Multiplexer Enable – 0 logic leagă AIN1 la intrarea negativă a comparatorului
- ACD: Analog Comparator Disable
- ACBG: Analog Comparator Bandgap Select
- ACO: Analog Comparator Output
- ACI: Analog Comparator Interrupt Flag
- ACIE: Analog Comparator Interrupt Enable
- ACIC: Analog Comparator Input Capture Enable
- ACIS1:ACIS0: Analog Comparator Interrupt Mode Select

ACSR – Analog Comparator Control and Status Register

Bit	7	6	5	4	3	2	1	0	
0x30 (0x50)	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	0	0	

ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
(0x7B)	–	ACME	–	–	-	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Exemplu cod

- Comparator pozitiv
- Referință internă de 1.23V
- Când tensiunea de pe pinul PB3 depășește 1.23V, PD0 este setat la 1 logic

```
#include <avr\io.h>
#include <avr\interrupt.h>

#define AINpin PB3
#define LED PD0

void ACinit(){

    DDRB &= ~(1 << AINpin);  //as input
    PORTB &= ~(1 << AINpin); //no Pull-up
    DDRD |= (1 << LED);      //Led pin as output
    PORTD |= (1 << LED);     //Initally LED is OFF
    SFIOR |= (1 << ACME);    //enable multiplexer
    ADCSRA &= ~(1 << ADEN);  //make sure ADC is OFF
    //select ADC3 as negative AIN
    ADMUX |= (0 << MUX2) | (1 << MUX1) | (1 << MUX0);
    ACSR |=
    (0 << ACD)|     //Comparator ON
    (1 << ACBG)|    //Connect 1.23V reference to AIN0
    (1 << ACIE)|    //Comparator Interrupt enable
    (0 << ACIC)|    //input capture disabled
    (0 << ACIS1)|   //set interrupt on output toggle
    (0 << ACIS0);
    sei();//enable global interrupts
}
// Interrupt handler for ANA_COMP_vect
//
ISR(ANA_COMP_vect) {
  if(bit_is_clear(ACSR, ACO))
    PORTD &= ~(1 << LED);//LED is ON
    else
      PORTD |= (1 << LED);//LED is OFF
}

int main(void) {
     ACinit();
     while(1) { }    // Infinite loop; interrupts do the rest
return 0;
}
```

Acknowledgements

- These slides contain materials from Prabal Dutta, Mark Brehob and Thomas Schmid (UMich)