Rīgas Tehniska Universitāte

Datorvadības, automātikas un datortehnikas institūts

Datoru tīklu un sistēmas tehnoloģijas katedra

Mikroprocesoru tehnika
Laboratorijas darbs Nr. 4

Asist. R. Taranovs
Profesors V.Zagurskis
Students Vitālijs Hodiko
3.kurss 1.grupa

Uzdevums

1. Izmainīt piedāvāto 8 bitu ADC pārveidošanas kodu, lai būtu iespēja strādāt ADC 10 bitu režīmā.
2. Izmainīt mērāmo signālu uz sin().
3. Pierādīt koda pareizo darbību.

Teorētiskais apraksts

Analogs-Cipars pārveidotājs

ATmega 128 tiek izmantots 10 bitu secīga tuvinājuma ADC. ADC ir pievienots multipleksoram ar 8 kanāliem. Tas nozīmē, ka ar šādu ADC var mērīt 8 dažādu analogos signālus. Trūkums ir tāds, ka vienā laika momentā ir iespējams pārveidot tikai vienas ieejas analogo signāli. Praktiski tas nozīmē, kas vispirms ir jāizvēlas kanāls, kura analogo signālu vēlas mērīt, jāpārveido šī signāls un tad ir jāpārslēdz cita ACP ieeja ar multipleksoru. Secīga tuvinājuma

ADC satur Sample and Hold shēmu, kas nodrošina, ka ADC ieejas spriegums tiek turēts nemainīgā līmenī, kamēr notiek pārveidošana.

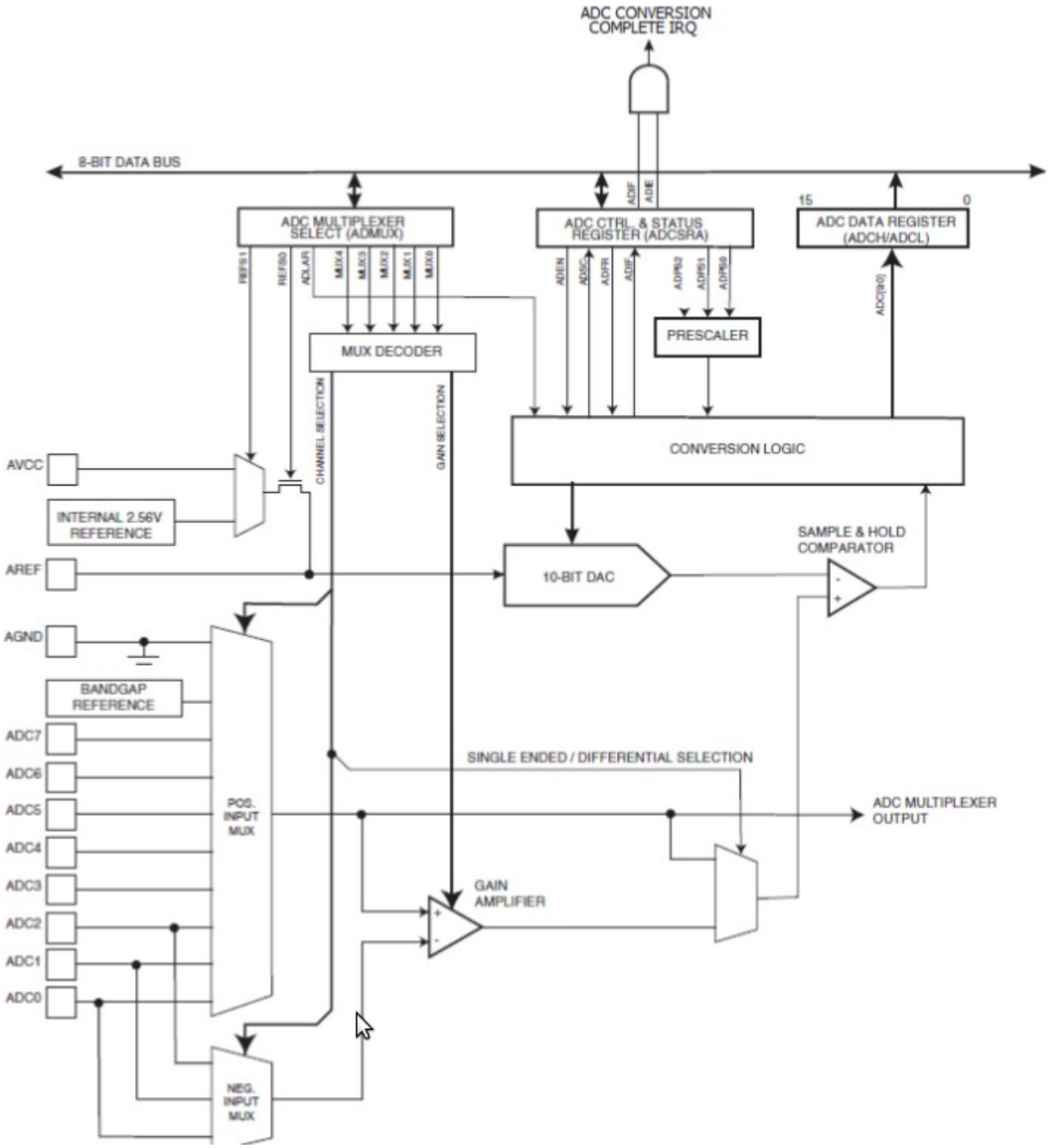
ADC ir iespējami divi barošanas avoti – viens ārējs, kuru pieslēdz pie AVCC, un viens iekšējs, kurš parasti ir 2.56V un tiek nodrošināts no shēmas. To, kuru izmantot ir iepriekš jāizvēlas.

Īsumā var uzskaitīt galvenās ATmega 128 ADC īpašības:

- 10 bitu izšķirtspēja
- 0.5 LSB integrālā nelinearitātes kļūda
- ±2 LSB absolūtā precizitāte
- 13-260 μs pārveidošanas laiks
- 8 multipleksēti vienas ieejas kanāli
- Var izvēlēties rezultāta izkārtošanu – izlīdzinājums pa kreisi vai labi
- 0-VCC ADC ieejas sprieguma diapazons
- Izvēles 2.56V ADC atbalsta spriegums
- Nepārtrauktas pārveidošanas vai vienreizējas pārveidošanas režīmi
- Pārtraukums uz ADC pārveidošanas pabeigšanu
- Miega režīma trokšņu slāpētājs

ADC darbības iestādīšanai izmanto ADCSRA (ADC statusa un vadības reģistrs) un ADMUX (ADC multipleksēšanas izvēles reģistrs. Lai ieslēgtu ADC jāuzstāda ADEN bits šajā reģistrā. Ja ADEN nav uzstādīt mikrokontrolleris nepatērē enerģiju, tāpēc ieejot miega režīmā to ir vēlams atslēgt.

Pārveidošanas rezultāts tiek glabāts ADC datu reģistros – ADCH un ADCL. Pēc noklusējuma rezultāts ir izlīdzināts pa labi, bet to var izlīdzināt arī pa kreisi. ADMUX reģistrā attiecīgi izmainot bitu ADLAR (Sk. Att. 21). Ja rezultāts ir izlīdzināts pa kreisi un nav nepieciešama vairāk kā 8 bitu precizitāte, tad var nolasīt tikai ADCH reģistru. Pretējā gadījumā vispirms jānolasa ADCL un tikai tad ADCH, lai nodrošinātu, ka rezultāts pieder vienai un tai pašai pārveidošanai. Nolasot ADCH reģistru tiek atjaunota ADC pieeja rezultāta reģistram un tas var ierakstīt jaunu rezultātu.

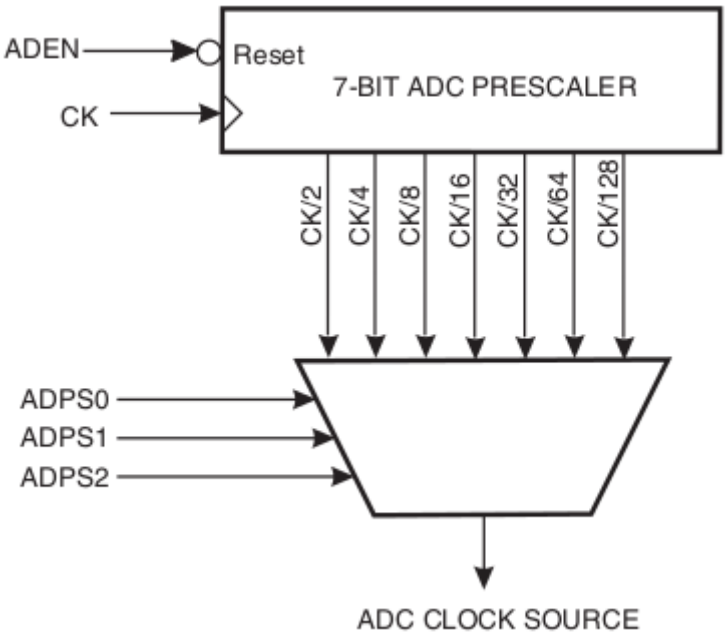


Attēls 1: ADC blokshēma

ADC ir savs pārtraukums, kurš tiek izsaukts tad, kad tiek pabeigta pārveidošana. Lai aprakstītu ADC pārtraukuma apstrādes funkciju:

```
ISR(ADC_vect) {

//Kods

}
```

Nepārtrauktas atjaunošanas režīmā ADC nepārtraukti ciparo signālu un atjauno datu reģistrus. To izvēlas uzstādot ADFR bitu ADCSRA reģistrā. Lai uzsākto pirmo pārveidošanu ir jāuzstāda ADSC bits šajā pašā reģistrā.



Attēls 2: ADC priekšdalītājs

Pēc noklusējuma, lai iegūtu maksimālo izšķirtspēju ieejas takts frekvencei jābūt no 50 kHz līdz 200 kHz. Ja nepieciešama izšķirtspēja ir mazāka par 10 bitiem, tad frekvence var būt lielāka par 200 kHz, lai iegūtu augstāku parauga ātrumu(sample rate).

Priekšdalītājs ģenerē pieņemamo ADC takts frekvenci no jebkuras CPU frekvences virs 100 kHz. To uzstāda ar ADPS bitiem ADCSRA reģistrā. To izmanto, lai palielinātu vai samazinātu rezultāta atšķirību no oriģinālā ieejas signāla. Lai pārveidotu signālu ir nepieciešami 13 ADC takts impulsi(pirmais pārveidojums aizņem 25 ADC taktis, lai inicializētu analogo shēmu).

Nosacījums	Sample & Hold (Cycles from Start of Conversion)	Pārveidošanas laiks(Cycles)
Pirmais pārveidojums	13.5	25
Normāls pārveidojums, single ended	1.5	13
Normāls pārveidojums, differential	1.5/2.5	13/14

Tabula 1: ADC pārveidojumu laiks

Priekšdalītājs sak skaitīt, kad ADC tiek ieslēgts iestādot ADEN bitu ADCSRA reģistrā un strādā kamēr tas ir iestādīts un nepatraukti atjaunojas ja ADEN ir zems. Pēc pārveidojuma rezultāts tiek ierakstīts ADC reģistros un ADIF ir uzstādīts(Vienas pārveides režīma ADSC ir notīrīts vienlaicīgi – lai uzsāktu jaunu pārveidošanu jāuzstāda ADSC un ar ADC augošo fronti pārveidojums sāksies).

Diferenciālā Pieauguma kanāli(Differential Gain Channels)

Diferenciālie pārveidojumi ir sinhronizēti ar iekšējo pulksteni CK_{ADC2} vienādu ar pusi no ADC pulksteņa. Sinhronizēšana notiek automātiski ar ADC interfeisu, tā ka sample&hold notiek konkrēta CK_{ADC2} frontē.

Pieauguma posms ir optimizēts uz 4 kHz joslas platumu. Augstākas frekvencēs var būt pakļautas nelineāram pastiprinājumam. Jāizmanto ārējo zemas frekvenču caurlaidības filtru(low-pass filter), ja ieejas signāls satur augstāku frekvenci nekā ir pieauguma posma joslas platums.

ADC pulksteņa frekvence ir neatkarīga no pieauguma posma caurlaides ierobežojuma. Piem., ADC takts periods var būt 6 us, ļaujot pārveidot kanālu ar 12 kSPS(Kilo Samples Per Second) neskatoties uz šī kanāla joslas platumu.

Kanāla maiņa vai atbalsta sprieguma izvēle(Changing Channel or Reference Selection)

MUXn un REFS1:0 biti ADMUX reģistrā ir vienreiz buferizēti caur pagaidu reģistru, kuram CPU ir nejauša piekļuve. Tas nodrošina, ka kanāli un atbalsta sprieguma izvēle notiek droša vieta pārveidošanas laikā. Tie tiek pastāvīgi atjaunināti kamēr pārveidošana sāksies. Kad tā sākas, tā bloķējas, lai nodrošinātu pietiekamu ADC izlases laiku. Tas atsākas pēdēja ADC taimera takti pirms pārveidošana beigsies(ADIF is set).(NB pārveidošanas sākas ADC taimera augoša frontē, tādēļ nav ieteicams rakstīt jauna kanāla vai atbalsta sprieguma izvēles vērtības ADMUX reģistrā pirms vienas ADC taimera takts pēc ADSC ir ierakstīts)

Jābūt īpaši uzmanīgam mainot diferenciālos kanālus. Kad tas ir izvēlēts, pieauguma posms var ilgt 125 us, lai stabilizētu jaunu vērtību. Tādējādi, pirms tā laika nevajadzētu uzsākt pārveidojums. Alternatīvi, var nereģistrēt pārveidošanas rezultātus šajā laikā perioda. Tādu pašu stabilizācijas laiku jāievēro pirmajām diferenciālām pārveidojumam pēc ADC pārraides izvēles(mainot REFS1:0 bitus ADMUX reģistrā). (Ja JTAG interfeiss ir ieslēgts, tas atcel ADC kanālu funkciju uz PORTF7:4)

ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
Def: 0x00	R/W							

Tabula 2: ADMUX(ADC Multiplexer Selection)reģistrs

REFS1:0: Atbalsta sprieguma izvēles biti(Reference Selection Bits)

Ja tiek mainīti pārveidojuma laikā, tā notiks pēc pārveidojuma beigām(ADIF ir uzstādīts). Iekšēja avota atsauces opcijas var neizmantot, ja ārējais ir pieslēgts pie AREF pina.

REFS1	REFS0	Atbalsta sprieguma izvēle
0	0	AREF, iekšējais V_{ref} ir atslēgts
0	1	AVCC ar ārējo kondensatoru uz AREF pina
1	0	Rezervēts
1	1	Iekšēja 2.56V avota atsauce ar ārējo kondensatoru uz AREF pina

Tabula 3: ADC Atbalsta sprieguma izvēle

ADLAR: Izklidzināšana pa labi(ADC Left Adjust Result)

1 – izlīdzināt pa labi, citādi pa kreisi. Maiņa notiek uzreiz, neskatoties uz tekošiem pārveidojumiem.

MUX4:0: Ieejas kanāla un pastiprinājuma izvēle (Analog Channel and Gain Selection Bits)

Ierakstot 2-0 bitā var izvēlēties kanālu, bet 4-3 var izvēlēties vai šo signālu ir nepieciešams pastiprināt. Ja tie ir mainīti pārveidošanas laikā, tie stājas spēka pēc tas(ADIF is set).

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0			
00001	ADC1			
00010	ADC2			
00011	ADC3	N/A		
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000(1)		ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010(1)		ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010	N/A	ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	1.23V (V_{BG})	N/A		
11111	0V (GND)			

Attēls 3: Ieejas kanāla un pastiprinājuma izvēle

ADC pārveidošanas rezultāts

Pēc pārveidošanas ADIF ir High un rezultātu var atrast ADC Rezultāta Reģistros ADCL, ADCH.

Vienreizējas pārveidošanas rezultāts ir: $ADC = \frac{V_{IN} * 1024}{V_{REF}}$, kur

V_{IN} - ir ieejas pina sprieguma avots; V_{REF} - atbalsta spriegums; 0x000 GND, 0x3FF izvēlētais atbalsta spriegums mīnus 1 LSB.

Ja tiek izmantoti diferenciālie kanāli: $ADC = \frac{(V_{POS} - V_{NEG}) * GAIN * 512}{V_{REF}}$, kur

VPOS – pozitīvas ieejas spriegums; VNEG – negatīvas ieejas spriegums; GAIN – izvēlētais pastiprinājums; VREF izvēlētais atbalsta spriegums. Rezultāts tiek saglabāts papildkoda formā. No 0x200(-512d) līdz 0x1FF(+511d). NB ja vajag ātri pārbaudīt rezultātu polaritāti, pietiekami nolasīt MSB no rezultāta(ADC9 no ADCH – ja tas ir 1, tad negatīvs un otrādi).

Piemērs:

```
ADMUX = 0xED (ADC3 - ADC2, 10x gain, 2.56V reference, left
adjusted result)
ADC3 = 300 mV, ADC2 = 500 mV.
ADCR = 512 * 10 * (300 - 500) / 2560 = -400 = 0x270
ADCL būs nolasīts kā 0x00, un ADCH - 0x9C. Ierakstot 0 ADLAR
izlīdzina pa kreisi rezultātu: ADCL = 0x70, ADCH = 0x02.
```

ADCSRA	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Def: 0x00	R/W							

Tabula 4: ADC Kontroles un Statusa Reģistrs A(ADC Control and Status Register A)

ADEN – ADC atļaušana (ADC Enable)

Uzstādot šo bitu tiek iedarbināts ADC, nodzēšot bitu tas tiek izslēgts(ja izslēdz pārveidošanas laika – tā tiek anulēta)

ADSC – ADC pārveidošanas uzsākšana (ADC Start Conversion)

Vienas pārveidošanas režīmā pirms katras pārveidošanas uzsākšanas ir jāuzstāda šis

bits loģiskajā 1. Nepārtrauktas pārveidošanas režīmā to nepieciešams veikt tikai vienu

reizi. Pirmā pārveidošana inicializē ADC.

ADFR – nepārtrauktas pārveidošanas režīma izvēle (ADC Free Running Select)

Uzstādot šo bitu tiek ieslēgts nepārtrauktas pārveidošanas režīms(dzēšot - atslēdz). Šajā režīmā nepārtraukti tiks atjaunoti datu rezultātu reģistri.

ADIF – ADC pārtraukuma karodziņš (ADC Interrupt Flag)

Šis bits tiek automātiski uzstādīts tad, kad tiek pabeigta pārveidošana un tiek atjaunoti datu reģistri. Ja ir atļauts pārtraukums tad tiek izsaukta pārtraukuma apstrādes funkcija. Tas tiek nodzēsts, kad pārtraukuma apstrādes funkcija tiek pabeigta. Alternatīvi to var

nodzēst ierakstot loģisko 1 karodziņā. Uzmanies, darot read-modify-write operācijas ar ADCSRA, gaidīts pārtraukums var būt atslēgts. Tas arī attiecas uz SBI un CBI instrukcijām.

ADIE – ADC pārtraukuma atļaušana (ADC Interrupt Enable)

Uzstādot šo bitu tiek atļauta ADC pārtraukuma apstrādes funkcijas izpildīšana.

ADPS2:0 – ADC pirms dalītāja izvēles biti (ADC Prescaler Select Bits)

Ar šiem bitiem tiek uzstādīta ADC pirms dalītāja vērtība. Par pamatu tiek ņemta takts frekvence no XTAL un dalīta ar attiecīgo dalītāju.

ADPS2	ADPS1	ADPS0	Pirms dalītājs
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Tabula 5: Pirms dalītāja izvēles biti

ADLAR	0	ADCH	–	–	–	–	–	ADC9	ADC8
		ADCL	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1
	1	ADCH	ADC1	ADC2	ADC3	ADC4	ADC5	ADC6	ADC7
		ADCL	ADC8	ADC9	–	–	–	–	–

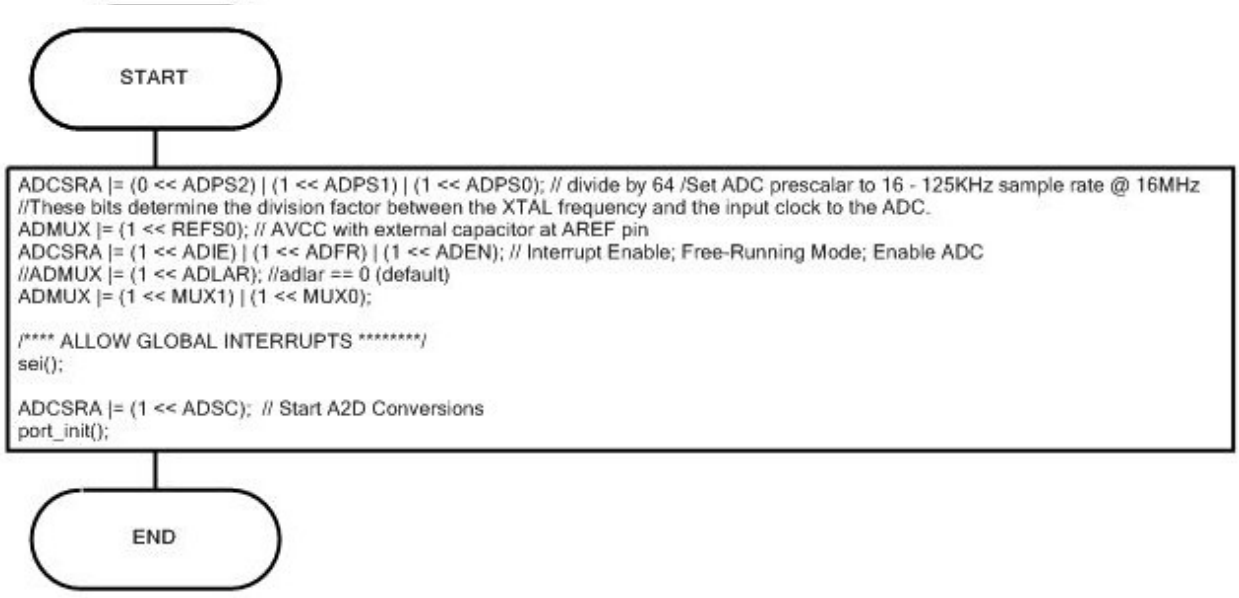
Tabula 6: ADC Datu Registrs – ADCL un ADCH

Diferenciālajā režīma rezultāts ir papildkodā. Kad ir nolasīts ADCL, ADC Datu Reģistrs nebūs atjaunots kamēr ADCH nebūs nolasīts. Tāpēc, ir vajadzīga 8 bitu precizitāte ir pietiekami nolasīt ADCH.

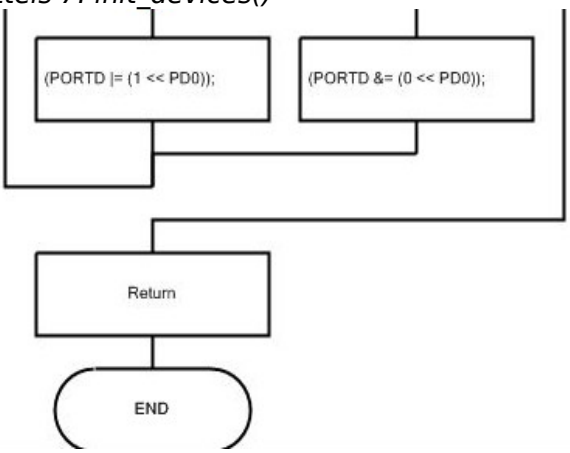
Programmas kods

```
/***** GLOBAL CONST DEFINE ****************************************/
#define F_CPU 14745600UL     //Hz - cycles per second;
/***** BIBLIOTEKU IEKLAUSANA **************************************/
#include <avr/io.h>
#include <avr/interrupt.h>
/***** FUNCTION PROTOTYPES ****************************************/
void port_init(void);
void init_devices(void);
/***** GLOBAL VARS ************************************************/
unsigned long ten_bit_value;
//unsigned int  LA;
/***** INTERRUPTS *************************************************/
ISR(ADC_vect){
      //ten_bit_value = ADLAR == 1 ? (ADCL >> 6) | (ADCH << 2) : ADCL | (ADCH
<< 8);
      ten_bit_value = ADC; //easy way
      ADCSRA |= (1 << ADSC);  // Start A2D Conversions
}
/***** MAIN *******************************************************/
int main(void){
      init_devices();
      while(1){
            //iededzinam LED ja ir izmantots vairaak par 8 bitiem
            ten_bit_value > 255 ? (PORTD |= (1 << PD0)) : (PORTD &= (0 <<
PD0));
      }
      return 1;
}
/***** PORTU INICIALIZACIJA ***************************************/
void port_init(void){
      DDRD = 0xFF; //visas porta D l?nijas uz IZvadi
      PORTD = ~(0x00); //porta D izejas l?niju l?me?i uz 0
}
/***** REZIMU[sakuma vertiibu] UZSTADISANA ************************/
void init_devices(void){
      //ADC SETUP
      ADCSRA |= (0 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // divide by 64
/Set ADC prescalar to 16 - 125KHz sample rate @ 16MHz
      //These bits determine the division factor between the XTAL frequency
and the input clock to the ADC.
      ADMUX |= (1 << REFS0); // AVCC with external capacitor at AREF pin
      ADCSRA |= (1 << ADIE) | (1 << ADFR) | (1 << ADEN); // Interrupt Enable;
Free-Running Mode; Enable ADC
      //ADMUX |= (1 << ADLAR); //adlar == 0 (default)
      ADMUX |= (1 << MUX1) | (1 << MUX0);

      /**** ALLOW GLOBAL INTERRUPTS ********/
      sei();

      ADCSRA |= (1 << ADSC);  // Start A2D Conversions
      port_init();
}
```

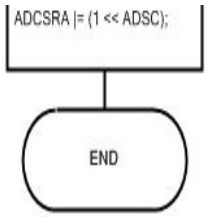
Blokshēma



Attēls 7: init_devices()



Attēls 4: Main()



Attēls 5: ISR(ADC_vect)

Secinājumi

Dotais darbs pārsvara bija teorētisks, jo nekādas fiziskas saskarsnes ar izstrādes plati nebija, kas padarīja darbu no vienas puses vieglāku, un tajā paša brīdi ne tik aizraujošo, bet no otras puses sarežģītāku, jo virtuālajā vide, jeb izmantojot Proteus 7 shēmas simulācijai ir iespējamas neatkarīgas lietojumprogrammatūras kļūdas. Piemēram, man mājās ar pirms-dalītāju zemāku par 8 shēma strādāja nepareizi vai nestrādātāja vispār.

Pārveidot ADC no 8 bitu uz 10 bitu režīmu arī nesagādāja lielas problēmas, neieskaitot iepriekšējas kļūdas ietekmi uz procesu. Pēc šā darba izpildīšanas lielu pārsteigumu sagādāja tas, ka pārveidošanas rezultātu var nolasīt pa tiešo no ADC reģistra, kas nebija minēts ne datu lapa, ne kur citur.

Darbojoties papildus nodarbības ar ADC pārveidotāju pēc lielām pūlēm sanāca pārveidot analogas vērtības no distances sensora ieejam uz digitālam. Fiziska saskarsne ar ierīcēm ir sarežģītāka par teoriju tik un tā. Taču vislielākās grūtības sagādāja ne datu nolasīšana, bet paralēla USART izmantošana nolasīto datu sūtīšanai. Kas savukārt bija apgrūtināts ar iepriekšēja koda izmantošanu. Sakuma ideja bija, ka tas atvieglos darbu, bet sanāca pretēji – labāk lieku reizi uzrakstīt no jauna nekā atkļūdot esošo.

Beigu beigas vissarežģītākais sanāca izprast Naikvista kritērija darbības principu, kaut arī tas man nesanāca. Citējot: “Mērāma signāla frekvencei jābūt 2x mazākai par pārveidošanas frekvenci”, ir viegli saprast, ka pat frekvence būs augstāka tā nebūs detektēta, jeb tas būs datu izkropļojums.