Rīgas Tehniska Universitāte

Datorvadības, automātikas un datortehnikas institūts

Datoru tīklu un sistēmas tehnoloģijas katedra

Mikroprocesoru tehnika
Laboratorijas darbs Nr.2

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

Uzdevums

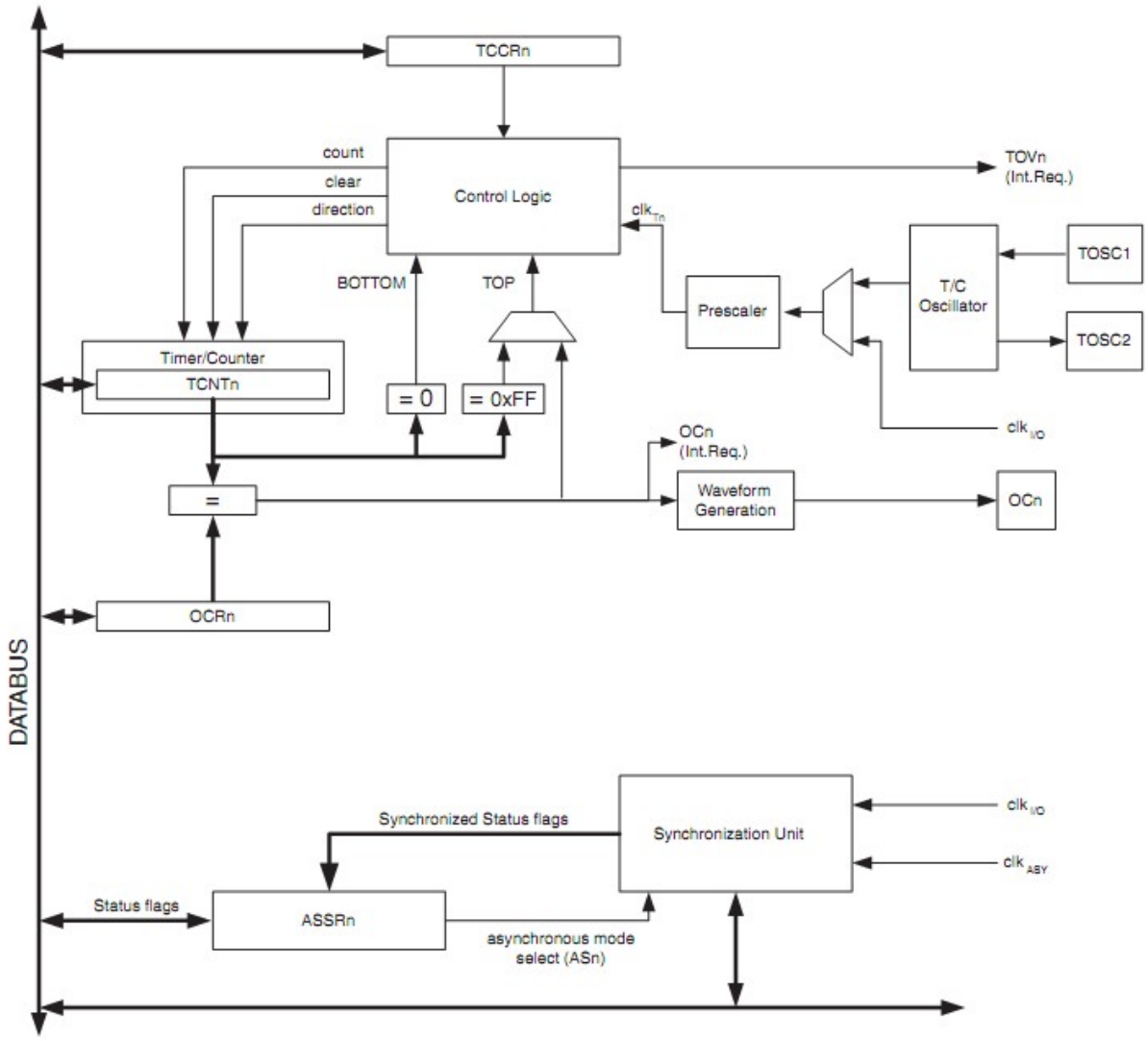
Modificēt pirmā laboratorijas darba programmas pirmkodu tā lai pauze starp gaismas diožu pārslēgšanām būtu vienāda vienai sekundei, laika mērīšanai izmantojot ATmega128 Taimeri/skaitītāju 0.

ATmega128 timer/counter

8-bit Timer/Counter0 ar PWM un Asynchronous Operation

Timer/Counter0 - vienkanāla, 8-bit Timer/Counter[TCNT] modulis ar šādam īpašībām:

- Vienkanāla skaitītājs
- Taimera notīrīšanās pēc salīdzināšanas sakrīšanas (Auto Reload)
- Fāzē korekta impulsa modulators (Pulse Width Modulator)
- Frekvences ģenerators
- 10-bit takts priekš dalītājs
- Pārpildīšanas un salīdzināšanas pārtraukumu avoti (Timer OVerflow 0 and Output Compare Flag 0)
- Atļauj taktēšanu no ārēja 32 kHz kristāla neatkarīgi no I/O takts



Attēls 1 8-bit Timer/Counter vienkāršota bloku diagramma

SREG	I	T	H	S	V	N	Z	C
Def: 0x00	R/W							

Tabula 1: Status Register

Tiek atjaunināts pēc visām ALU operācijām. Vairākos gadījumos tas novērš nepieciešamību izmantot īpaši veidotas salīdzināšanas operācijas, veidojot daudz ātrāku un kompaktāku kodu. Tas nav automātiski saglabāts ieejot pārtraukumā un atjaunināts izejot no tā(to jādara programmatiski).

- I – bit (Global Interrupt Enable) Atļauj globālus pārtraukumus sei(), cli() komandas programma.
- T – bit (Bit Copy Storage) Bitu kopēšanas instrukcijas BLD(Bit LoaD – kopē T bitu reģistrā Reģistru failā) un BST(Bit Store – kopē bitu no reģistra Reģistru failā uz T) izmanto to kā avotu vai mērķi operētam bitam.
- H – bit (Half Carry Flag) dažas aritmētiskas operācijās norāda uz pus pārnesi. Ir noderīgs BCD(Binary Coded Decimal) aritmētikā.
- S – bit (Sign bit = N ⊕V)
- V – bit (Two’s Complement Overflow Flag) papildkods
- N – bit (Negative Flag) norāda uz negatīvo rezultātu ALU operācijas
- Z – bit (Zero Flag) norāda uz 0 rezultātu ALU operācijas
- C – bit (Carry Flag) norāda uz pārnesi ALU operācijas

TCNT0 un OCR0 ir 8 bitu reģistri.

TCNT0	TCNT0[7:0]
Def: 0x00	R/W

Tabula 2: Taimera/Skaitītaja reģistrs

OCR0	OCR0[7:0]
Def: 0x00	R/W

Tabula 3: Izejas salīdzināšanas reģistrs

Visi pārtraukumu pieprasījuma signāli ir redzami Timer Interrupt Flag Registr'ā (TIFR) un tos var aktivizēt Timer Interrupt Register'ā (TIMSK).

TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
Def: 0x00	R/W							

Tabula 4: Taimera pārtraukumu maskēšanas reģistrs

TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
Def: 0x00	R/W							

Tabula 5: Taimera pārtraukumu karodziņu reģistrs

TCNT var būt taktēts no iekšēja oscilatora ar priekš-dalītāju vai asinhroni no TOSC1/2 piniem.

Ir daudz efektīvs ceļš kā uzzināt, ka ir pagājusi viena sekunde, nekā izsaukt pārtraukumu katru 30.5uS kā ir ieteikts apraksta. Vajag izmantot iebūvētas mikrokontrolleri īpašības, lai minimizētu nepieciešamos resursus.

Piemēram, pievienojot kristālu pie TOSC1/2 piniem un liekot TIMER0 izmantot to, nevis iekšējo, to var dalīt, lai saņemtu dažādas frekvences. Ar priekš-dalītāju 128, viņš pārtrauksies tieši pēc vienas sekundes[32.768kHz / 128 = 1Hz overflow]. Šāda veida var ieekonomēt daudz enerģijas, ja jūs to izmantojiet kā Reāla laika pulkstenis, ka arī tas prasīs daudz mazāk apstrādes laika, jo tas strādās uz 1Hz nevis 32.768KHz, jeb 1 reizi sekunde, nevis 32768.

Šo metodi pat var uzlabot, tajos mikrokontrolleros, kur TIMER2 var strādāt asinhrona režīma, kurš ļauj virtuāli izslēgt visu čipu, kamēr kristāls strādās uz piniem, saglabājot daudz vairāk enerģijas uz baterijām bāzētiem projektiem. Un tas viegli atmodinās čipu, kad uzstādītājs nosacījums izpildīsies.

Asinhronais režīms ir kontrolēts no Asinhrona Statusa reģistra(ASSR).

Kad TCNT0 strādā asinhrona režīma daži apsvērumi jāņem vēra:

- Pārslēdzoties no asinhrona režīma uz sinhrono TCNT0, OCR0 un TCCR0 var būt bojāti. Droša procedūra ir:
    - atslēgt TCNT0 pārtraukumus notīrot TOIE0, OCIE0
    - izvēlies taktēšanas avotu uzstādot AS0 vajadzības gadījumā
    - ieraksti jaunas vērtības TCNT0, OCR0 un TCCR0
    - lai pārslēgtos asinhrona režīma: gaidi pēc TCN0UB, OCR0UB un TCR0UB
    - notīri TCNT0 pārtraukumu karogus
    - atļauj pārtraukums ja vajag

- Ģenerators ir optimizēts strādāt ar 32.768 kHz kristālu. Pieslēdzot ārēju kristālu pie TOSC1 pina var izraisīt TCNT0 nepareizu darbību. CPU kristāla frekvencei jābūt vairāk nekā 4x lielākai par ģeneratora frekvenci.

- Rakstot kādā no TCNT0, OCR0 vai TCCR0 reģistriem vērtība tiek saglabāta pagaidu reģistrā un fiksēta pēc divām pozitīvam malām uz TOSC1. Kamēr pagaidu reģistrs nav nodevis vērtības galamērķi nedrīkst rakstīt jaunas vērtības. Katrām no trim reģistriem ir savs pagaidu reģistrs un nekas netraucē, piemēram, rakstot uz TCNT0 uzreiz rakstīt uz OCR0. Lai atklātu ka vērtība tika nodota tiek izmantots ASSR reģistrs.

- Ieejot Power-save vai Extended Stanby režīmā pēc vērtības ierakstīšanas kādā no TCNT0, OCR0 vai TCCR0 reģistriem jāuzgaida kamēr tas atjaunināts, ja TCNT0 izmantot lai atmodinātu ierīci. Citādi MCU ieies miega režīma pirms izmaiņām. Tās ir īpaši svarīgi ja OC0 pārtraukums tiek izmantots, lai atmodinātu ierīci, jo rakstot OCR0 vai TCNT0 reģistra izejas salīdzināšanas funkcija tiek atslēgta. Ja rakstīšanas cikls nav pabeigts, kad MCU ieiet miega režīma pirms OCR0UB atgriež nulli, tad salīdzināšanas atbilstības pārtraukums nekād nebūs izsaukts un MCU neatmodīsies.

- Ja TCNT0 tiek izmantos lai atmodinātu ierīci no Power-save vai Extended Stanby režīmā daži apsvērumi jāņem vēra, jā vajadzēs atgriezties viena no šiem režīmiem:

    - pārtraukumu loģikai vajag viena TOSC1 takts, lai re startētos. Ja laiks starp atmošanos un atkārtotas ienākšanas miega režīmā ir mazāks, pārtraukums netiks izsaukts un ierīce nespēs pamosties. Šādu algoritmu var izmantot, lai pārliecinātos ka tas ir pagājis:

        - ieraksti TCNT0, OCR0 vai TCCR0 reģistrā

        - uzgaidi kamēr atbilstošs UB karogs ASSR reģistrā atgriezīs 0

        - ieej Power-save vai Extended Stanby režīmā

- Asinhronajā režīma 32.768kHz TCNT0 ģenerators vienmēr strādā izņemot Power-down un Stanby režīmus. Pēc Power-up Reset vai atmošanās no Power-down vai Stanby režīma jāapzinās to faktu, ka ģeneratoram vajag sekundi lai stabilizētos. Ir rekomendēt uzgaidīt vizmās vienu sekundi pirms izmantot TCNT0. Visu TCNT0 reģistru saturu var uzskatīt par zaudētu sakara ar nestabilo kristāla signālu pēc starta, vienalga izmanto iekšējo ģeneratoru vai ārējo kristālu uz TOSC1 pina.

- Atmošanās apraksts no Power-save vai Extended Stanby režīmā, kad skaitītais ir taktēts asinhroni:

    - Kad pārtraukuma nosacījums ir izpildīts, atmošanās process sākas tai paša taimera takti, tas ir, taimeris ir priekšā vismaz par 1 pirms procesors var nolasīt skaitītāja vērtību. Pēc atmošanās procesors ir apturēts uz 4 taktīm, tas izpilda pārtraukumu rutīnu(interrupt routine), un atsāk izpildi no komandas kas seko pēc SLEEP

- Nolasot TCNT0 reģistru īsi pēc atmošanās no Power-save var dot nepareizu rezultātu. Tā ka TCNT0 ir asinhroni taktēts no TOSC kristāla, TCNT0 nolasīšanu var īstenot izmantojot reģistru sinhronizētu ar iekšējo I/O pulksteņa domēnu. Sinhronizācija notiek katru TOSC1 augošo fronti. Pamostoties no Power-save režīma un $clk_{I/O}$ atkal kļūst aktīvs, TCNT0 būs

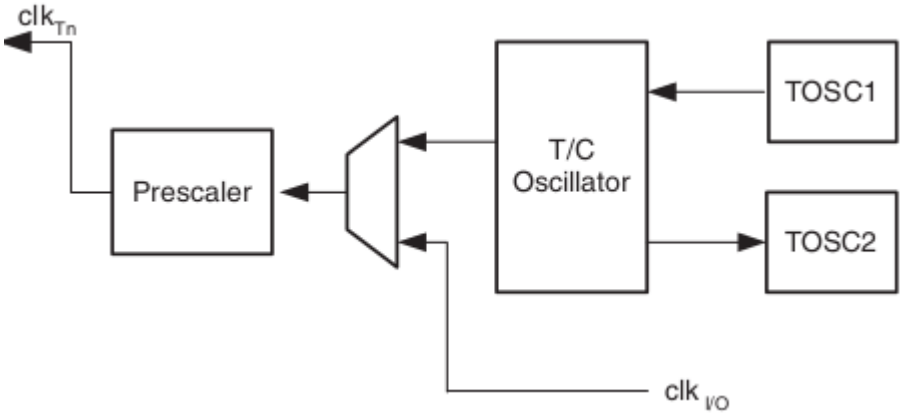
nolasīta iepriekšēja vērtība(pirms ieejas miega režīmā) līdz nākamajai TOSC1 augošai frontei. TOSC kristāla posms pēc pamošanās no Power-save režīma ir ļoti neprognozējams, jo tas ir atkarīgs no pamošanās laika(wake-up time). Ieteicama TCNT0 nolasīšanas procedūra ir:

- ierakstī jebkuru vertību uz OCR0 vai TCCR0
- gaidi attiecīga UB karoga notīrīšanos
- nolasi TCNT0

- Asinhronajā režīmā pārtraukuma karogu sinhronizācija asinhronajām taimerim aizņem 3 CPU taktis + 1 taimera takts. Tādēļ taimeris ir palielināts par 1 pirms procesors var nolasīt taimera vērtību izraisot pārtraukuma karoga iestatījumu. Izejas salīdzināšanas pins (OC0 pin) ir izmainīts taimera taktī un nav sinhronizēts ar procesora kristālu.

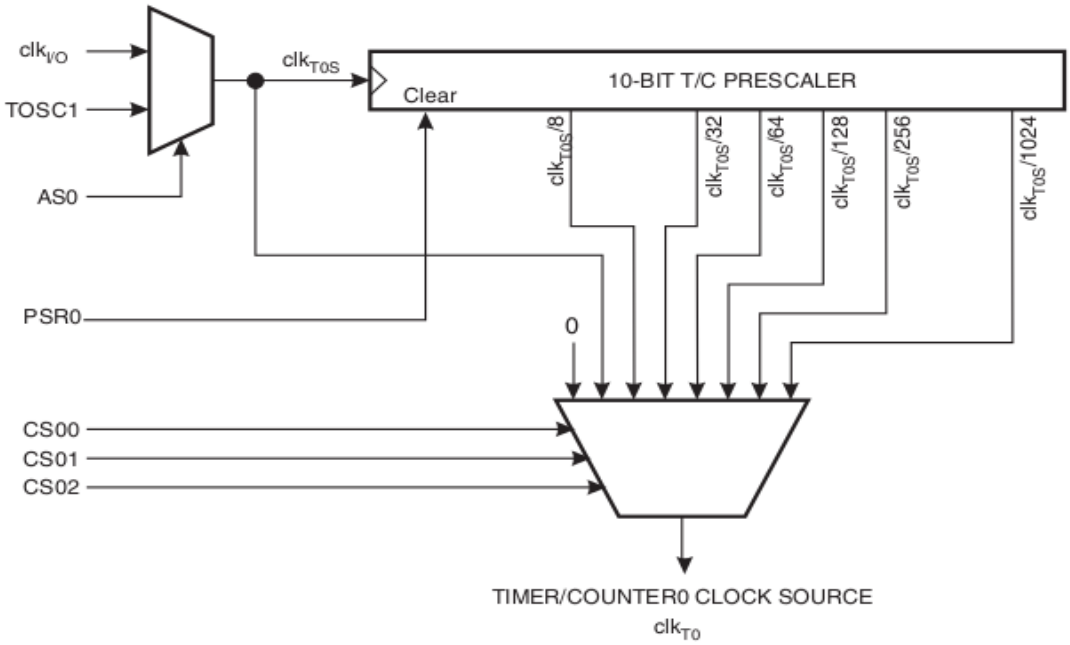
ASSR	-	-	-	-	AS0	TCN0UB	OCR0UB	TCR0UB
Def: 0x00	R				R/W	R		

Tabula 6: Asinhrona Statusa Reģistrs



Attēls 2: Kristāla izvēles loģiskais bloks(Clock select logic block)

Kristāla izvēles loģiskais bloks kontrolē kuru kristālu izmantos TCNT, lai palielinātu/samazinātu to vērtību. Kad tas nav izvēlēts TCNT ir atslēgts. Izejā no bloka tiek nosaukta - taimera pulkstenis(clkTn).



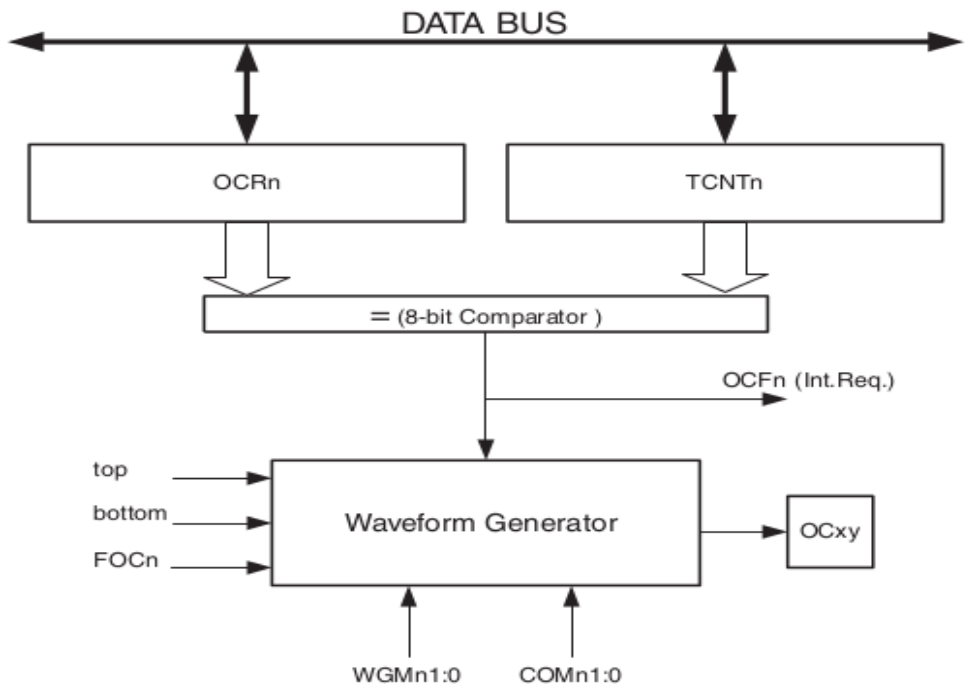
Attēls 3: TCNT priekšdalītājs

clk(T0) pēc noklusējuma ir pievienots pie iekšēja sistēmas kristāla clk(I/O). Iestādot AS0(TOSC2:1 tiek atvienoti no PORTC. Pēc tam kristāls var būt pievienots starp TOSC2:1 piniem, lai kalpotu kā TCNT0 taktu avots) ASSR reģistrā tas tiek asinhroni taktēts no TOSC1 pina. Tas ļauj izmantot TCNT0 ka Reāla Laika Skaitītāju(Real Time Counter). Iespējamas priekš-dalītāja vērtības ir: 8, 32, 64, 128, 256, 1024. Iestādot PSR0(Prescaler Reset TCNT0, netiek notīrīts jā TMS[Test Mode Select] ir ierakstīts) bitu Speciālajā Funkciju I/O Reģistrā (SFIOR) TCNT0 priekšdalītājs būs re-startēts. Atkarība no režīma, skaitītais ir notīrīts, palielināts vai samazināts katru takti.

SFIOR	TSM	-	-	-	ACME	PUD	PSR0	PSR321
Def: 0x00	R/W	R			R/W			

CS02	CS01	CS00	Apraksts
0	0	0	Nav taktēšanas avota(TCNT apturēts)
0	0	1	clkT0S/(nav priekš-dalīšanas)
0	1	0	clkT0S/8(No priekšdalītāja)
0	1	1	clkT0S/16(No priekšdalītāja)
1	0	0	clkT0S/32(No priekšdalītāja)
1	0	1	clkT0S/64(No priekšdalītāja)
1	1	0	clkT0S/128(No priekšdalītāja)
1	1	1	clkT0S/1024(No priekšdalītāja)

Tabula 7: Clock Select bit



Attēls 4: Izejas Salīdzināšanās Bloks(O/P Compare Unit)

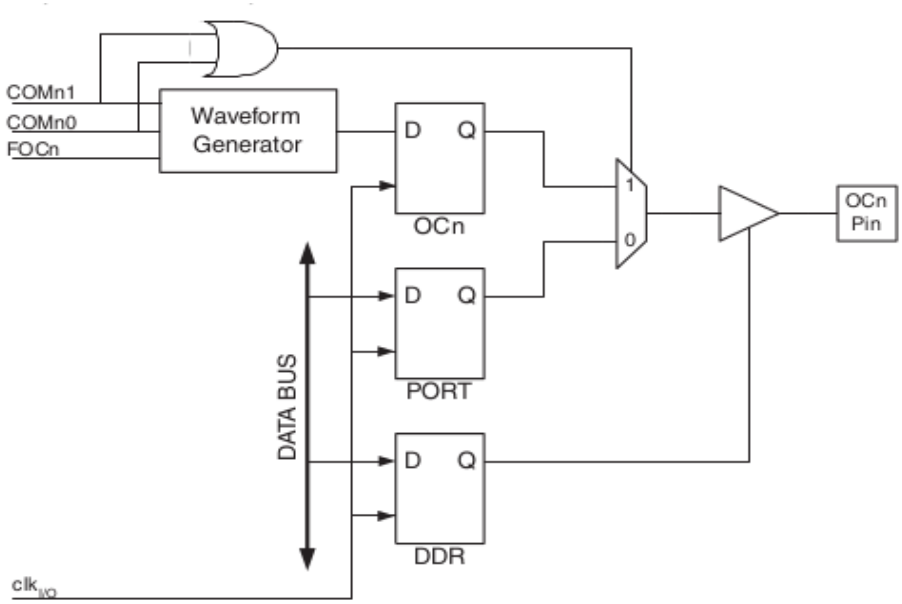
OCRn katru takti tiek salīdzināts ar TCNTn vērtību. Rezultāts var būt izmantots, lai uzģenerētu PWM vai mainīgas frekvences vilni uz Izejas Salīdzināšanās Pinu(OCxy, kur x – ir skaitītāja numurs, y – kanāla burts, ja ir vairāki). Salīdzināšanas sakritība uzstādīs Salīdzināšanas karogu (OCFn nākamajā takti būs notīrīts, jā OCIEn ir uzstādīts, tad pēc pārtraukuma. Alternatīvi to var uzstādīt programmatiski), ko var izmantot, lai pieprasītu Izejas Salīdzināšanas Pārtraukumu(O/p Compare Int.Req). Jā OCxy ir pievienots pie pina COM01:0 bitu funkcijas ir atkarīgas no WGM01:0 bitu uzstādījuma.

Režīms	WGM01	WGM00	TCNT režīms	TOP	OCRn update at	TOVn flag set on
0	0	0	Normal	0xFF	Tūlītējs	MAX
1	0	1	Fāze korekta viļņa ģen.	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Tūlītējs	MAX
3	1	1	Ātrais PWM	0xFF	Bottom	MAX

sRežīm	0	1	2	3
TCNT0	BOTTOM → TOP Tajā paša takti, kad TCNT0 kļūst 0 TOV0 karogs būs uzstādīts(var būt izmantots ka 9 bits. Notīrās tikai pārtraukumā, ja tas ieslēgts vai programmatiski). Jaunu TCNT0 vērtību var ierakstīt jebkād.	BOTTOM → TOP→ BOTTOM Atkārtoti skaita. Skaitītais tiek palielināts kamēr sasniegs MAX. Kad sasniedz to maina skaitīšanas virzienu. Tas bus vienāds MAX vienu takti. TOV0 karogs ir uzstādīts katru reizi sasniedzot BOTTOM	Skaitītais ir notīrīts kad sakrīt ar OCR0. Tajā paša takti, kad TCNT0 kļūst 0 TOV0 karogs būs uzstādīts.	BOTTOM → TOP Tiek palielināts kamēr sasniedz MAX vērtību un būs notīrīts nākamajā takti. TOV0 tiek uzstādīts tajā paša takti, kad TCNT0 sasniedz MAX.
OCR0	Var būt izmantots, lai ģenerētu pārtraukumus pēc noteikta laika. Nav ieteicams izmantot viļņu ģenerēšanai, jo tas aizņems daudz CPU laika.	Ja uzstāda uz BOTTOM, izejā būs const zema. Uzstādot uz MAX rezultāta iegūsim izeja const augstu signālu ne invertēta režīmā. There are two cases that give a transition without Compare Match: OCR0 changes its value from MAX, like in Figure 40. When the OCR0 value is MAX the OCn pin value is the same as the result of a down-counting Compare Match. To ensure symmetry around BOTTOM	Definē TCNT0 TOP vērtību	Ja uzstāda uz BOTTOM, izejā būs šaura smaile (narrow spike) katrā MAX+1 taimera taktī. Uzstādot uz MAX rezultāta iegūsim izeja const augstu(H) vai zemu(L) signālu (atkarīgs no COM01:0 bitiem).

		the OCn value at MAX must correspond to the result of an up- counting Compare Match. The timer starts counting from a higher value than the one in OCR0, and for that reason misses the Compare Match and hence the OCn change that would have happened on the way up.		
COM01:0		OC0 vērtība būs redzama tikai ja tas ir uzstādīts uz o/p. PWM vilnis tiek ģenerēts notīrot/uzstādot OC0 reģistru salīdzināšanas atbilstībās laikā starp OCR0 un TCNT0 kad skaitītais inc, un uzstādot/notīrot OC0 reģistru salīdzināšanas atbilstībās laikā starp OCR0 un TCNT0 kad skaitītais dec.	Lai ģenerētu vilni uz katru salīdzināšanas atbilstību var uzstādīt uz 1 (Toggle mode). OC0 vērtība būs redzama tikai ja tas ir uzstādīts uz o/p.	OC0 vērtība būs redzama tikai ja tas ir uzstādīts uz o/p. PWM vilnis tiek ģenerēts uzstādot/notīrot OC0 reģistru salīdzināšanas atbilstībās laikā starp OCR0 un TCNT0, un notīrot/uzstādot OC0 reģistru taimera taktī, TCNT0 vērtība tiek notīrīta. Frekvences izeju (ar 50% darbības ciklu) var panākt uzstādot COM01:0 uz 1(OC0 pārslēgsies uz katru sakritību)
		Divu slīpumu operācija (dual-slope). Augstas izšķirtspējas PWM viļņa ģenerācija. Ir mazāka maksimāla frekvence nekā Ātra PWM režīmā. PWM izšķirtspēja ir fiksēta uz 8 bitiem Simetriskas īpašības dēļ šīs režīms ir piemērots motoru kontroles sistēmas. $f_{OCnPCPWM} = \frac{f_{clk_{I/O}}}{N * 510}$	Atvieglo ārējo notikumu skaitīšanu. Pārtraukums var būt ģenerēts katru reizi kad TCNT0 vērtība sasniedz TOP vērtību, izmantojot OCF0 karogu. Pārtraukumā var mainīt TOP vērtību. (NOTE: Var izlaist salīdzināšanas atbilstību). $f_{OCn} = \frac{f_{clk_{I/O}}}{2 * N(1+OCI}$	Augstas frekvences viļņa ģenerācijas iespēja. Viena slīpuma darbība (single-slope). Tās dēļ frekvence var būt divreiz lielāka par fāzes korekta PWM frekvenci, kura izmanto divu slīpumu operāciju (dual-slope). Augstas frekvences dēļ tā ir labi piemērota barošanas regulēšanai, labošanai (rectification), un DAC sistēmas. Ļauj samazināt sistēmas izmaksas, jo augsta frekvencē var izmantot mazas ārējas

			, kur N = (1, 8, 32, 64, 128, 256, or 1024) .	komponentes(spoles, kondensatori). Dubulta buferizācija ir ieslēgta. $f_{OCnPWM} = \frac{f_{clk_{I/O}}}{N * 256}$



Attēls 5: Salīdzināšanās Sakritības Izejas Bloks(Compare Match O/P Unit). Ir parādīti tikai tie reģistri, kuri ir ietekmēti ar COM1:0 bitiem – OCn(iekšējais Ocn reģistrs, nevis Ocn ports), PORT un Datu Virziena Reģistrs(DDR)

Izejas Salīdzināšanās O/P Režīma(COM1:0) bitiem ir divas funkcijas. Viļņu ģenerators izmanto tos, lai definētu OCn stāvokļi nākamajā Salīdzināšanās Sakritībā(Compare Match). Galvena I/O funkcija tiek atcelta ar OCn no viļņa ģeneratora, ja kāds no COM1:0 bitiem ir uzstādīts. OCn pina virziens ir kontrolējams arī no DDR reģistra(DDR_OCn). Tam jābūt uzstādītam kā O/P pirms OCn vērtība būs redzama uz pina. Ne PWM režīmos izejas sakritība var būt ģenerēta programmatiski iestādot FOCn(Tas neģenerē pārtraukumu, un nenotīrīs TCNTn CTC režīma izmantojot OCRn kā TOP. Vienmēr nolasās ka 0) bitu Taimera Kontroles Reģistrā(TCCRn).

TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Def: 0x00	W	R/W						

Tabula 8: TCNT Kontroles Reģistrs

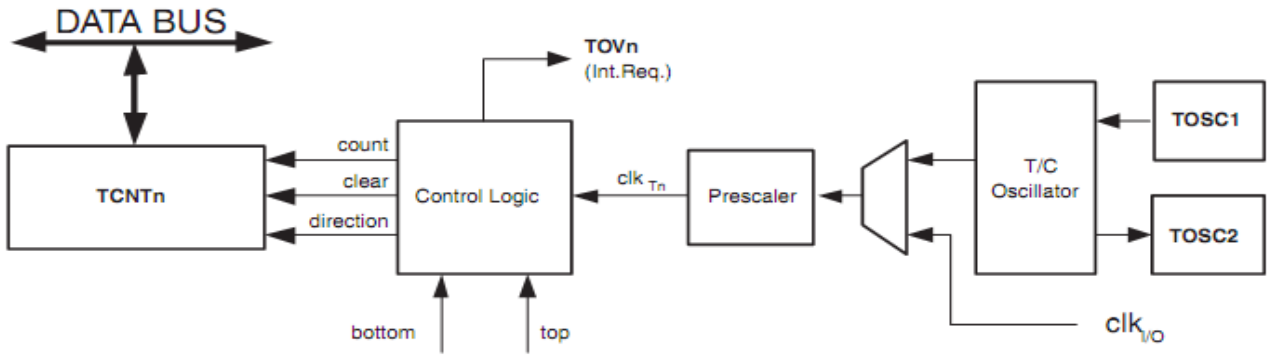
COM01	0	0	1	1
COM00	0	1	0	1
Apraksts	Nē PWM (Normal vai CTC[Clear Timer on Compare match])			
	OC0 atslēgts	Pārslēgt OCn uz Salīdzināšanas Sakritību	Notīrīt OCn uz Salīdzināšanas Sakritību	Uzstādīt OCn uz Salīdzināšanas Sakritību
	Ātrais PWM			
	OC0 atslēgts	Rezervēts	Notīrīt OCn uz Salīdzināšanas Sakritību, uzstāda OCn at BOTTOM (non inverting)	Uzstādīt OCn uz Salīdzināšanas Sakritību, notīra OCn at BOTTOM (inverting)
	Fāze korekta viļņa ģen.			
	OC0 atslēgts	Rezervēts	Notīrīt OCn uz Salīdzināšanas Sakritību, kad skaita uz augšu. Uzstādīt OCn skaitot uz lēju uz Salīdzināšanas Sakritību (non inverting)	Uzstādīt OCn uz Salīdzināšanas Sakritību, kad skaita uz augšu. Notīrīt OCn skaitot uz lēju uz Salīdzināšanas Sakritību (inverting)

Tabula 9: Compare O/P Mode

COM01:0 biti kontrolē vai PWM izējā ir invertēts vai nē. Ne PWM režīmos tas kontrolē vai izeju uzstāda, notīra vai pārslēdz uz sakritību.

Dubulta buferizācija sinhronizē Izejas Salīdzināšanas Reģistra(OCR0 ir double buffered tikai kādā Fāzē korekta impulsa modulācijas režīmā (PWM)) izmaiņās skaitīšanas secība TOP vai BOTTOM laikā. Tas pasarga no nepāra garuma(odd-length), ne simetriska PWM impulsa, padarot izeju bez kļūdainu.

BOTTOM	0x00
MAX	0xFF (decimal 255)
TOP	TCNT sasniedz TOP, kad tas kļūst vienāds lielākai vērtībai skaitīšanas secībā. Tā var būt fiksēta(MAX) vai vertība, kas glabājas OCR0 reģistra. Tas ir atkarīgs no režīma.



Attēls 6: Skaitītāja Bloks(bi-directional counter unit)

count – palielina/samazina TCNT0 par 1.

direction – izvēlas starp palielināšanu/samazināšanu

clear - notīra TCNT0 (uzstāda visus bitus uz nullēm)

clkT0 - Timer/Counter kristāls.

Top – signalizē ka TCNT0 sasniedza MAX vērtību

bottom - signalizē ka TCNT0 sasniedza BOTTOM vērtību

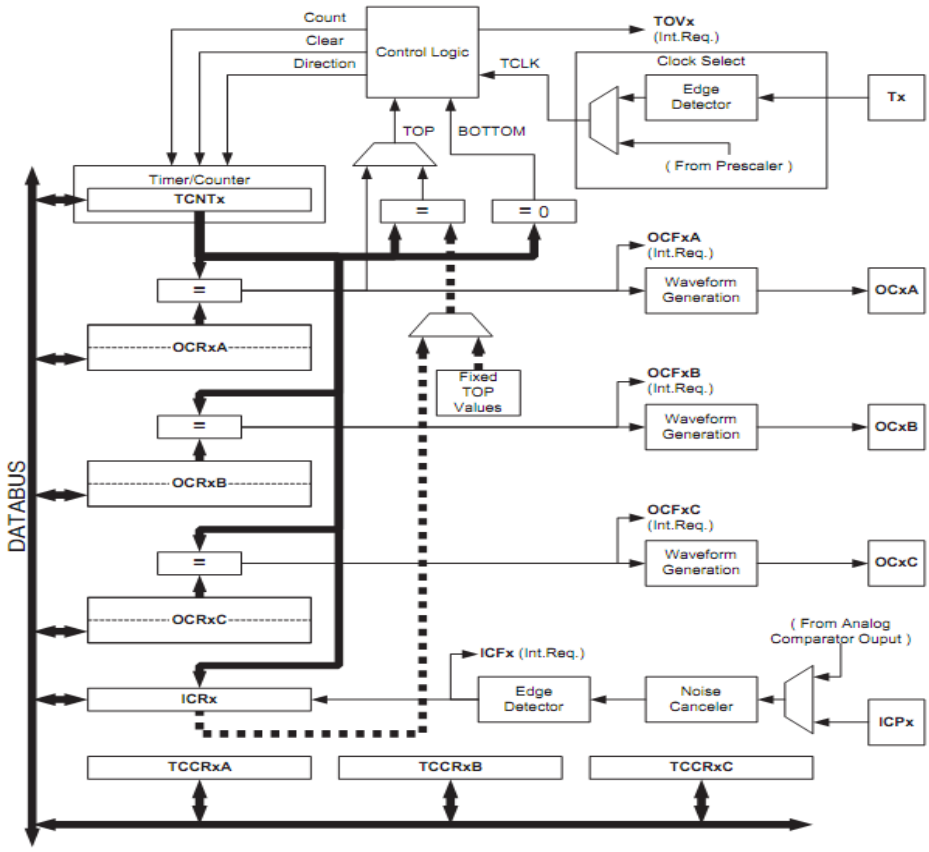
TCNT0 vērtība ir sasniedzama ar CPU pat ja clkT0 nav norādīts. CPU ir prioritāte par visam notīrīšanas/skaitīšanas operācijām. Visas CPU rakstīšanas operācijas uz TCNT0 reģistru bloķē jebkādu salīdzināšanas sakritību nākamajā taktī, pat ja taimeris būs apturēts. Tas ļauj inicializēt OCRn ar tādu pašu vērtību kā TCNTn neizsaucot pārtraukumu, kad TCNT ir ieslēgts(Risks izlaist sakritības pārbaudi).

16-bit Timer/Counter (Timer/Counter1 and Timer/Counter3)

16-bit Timer/Counter modulis ļauj precīzi kontrolēt programmas izpildes laiku (event management), viļņu ģenerēšana, un sinhronizācijas signāla mērīšanā. Galvenas īpašības:

- 16-bit dizains (i.e.,ļauj 16-bitu PWM)
- Trīs neatkarīgi izejas salīdzināšanas vienības
- Dubulta buferizētas izejas salīdzināšanas reģistri
- Viena ievades tveršanas vienība
- Trokšņu filtrs uz ievades
- Taimera notīrīšanās pēc salīdzināšanas sakrišanas(Auto Reload)
- Fāzē korekta impulsa modulators (PWM)
- Mainīgs PWM periods
- Frekvences ģenerators
- Ārējais notikumu skaitītājs

- 10 neatkarīgu pārtraukumu avotu (TOV1, OCF1A, OCF1B, OCF1C, I[nput]C[apture]F[lag]1, TOV3, OCF3A, OCF3B, OCF3C, un ICF3)

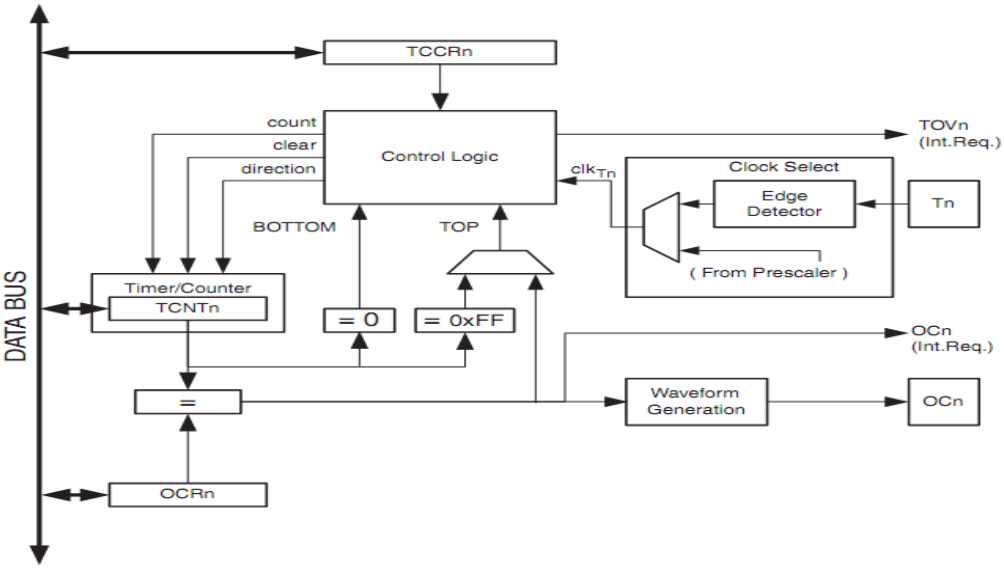


Attēls 7 16-bit Timer/Counter Block Diagram

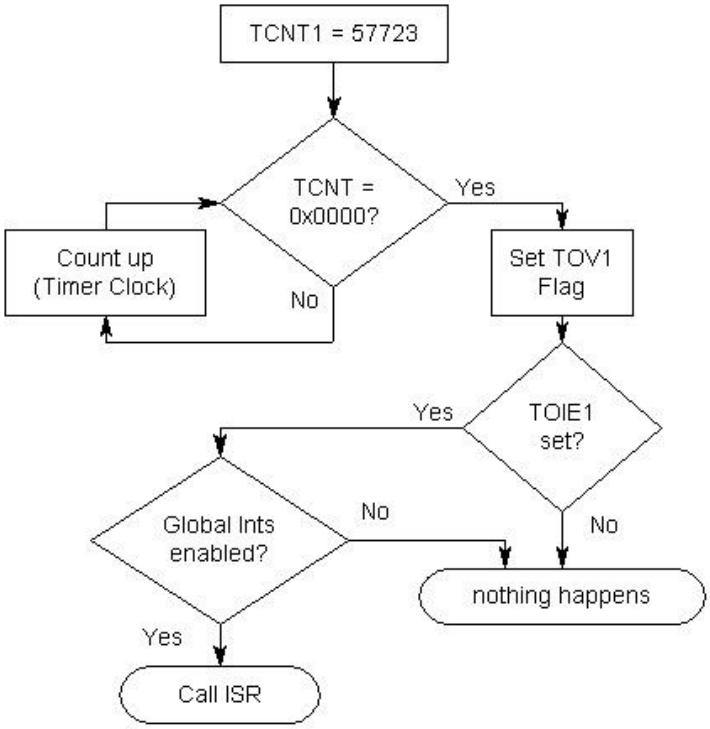
8-bit Timer/Counter2 ar PWM

Atšķiras no 8-bit Timer/Counter ar šādu īpašību:

- Ārējais notikumu skaitītājs



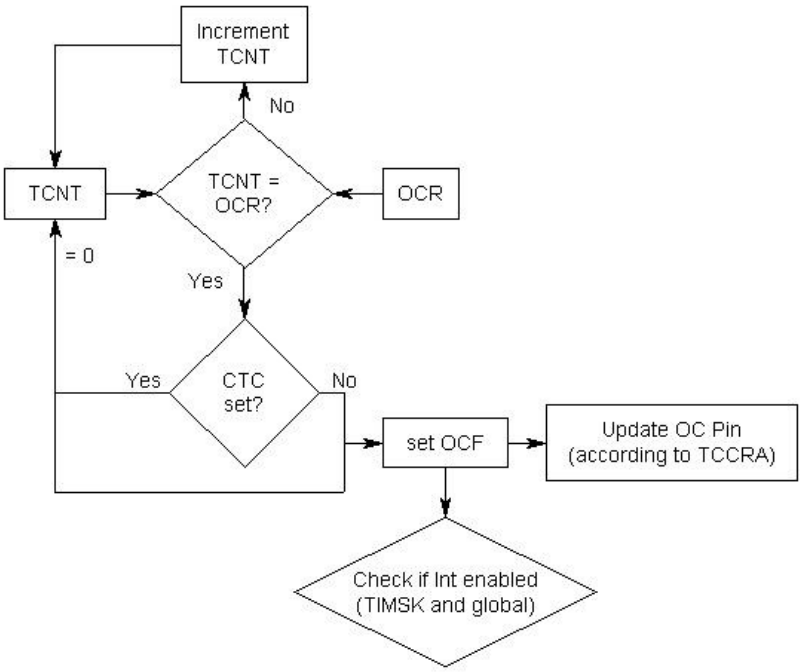
Attēls 8 8-Bit Timer/Counter Block Diagram



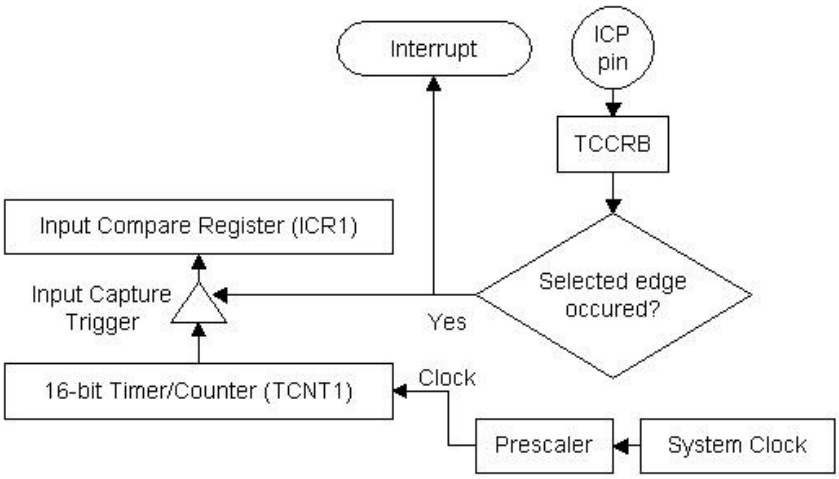
Attēls 9 Normal Mode: timer ovf int

Soļi, lai to ieslēgtu:

- uzstādīt priekš dalītāju uz 1024 (set bits CS12 un CS10 TCCR1B reģistrā)
- ieraksti 57723 TCNT1
- atļauj TOIE1 TIMSK reģistrā
- uzstādīt I-bit



Attēls 10 Compare Mode: timer ovf int

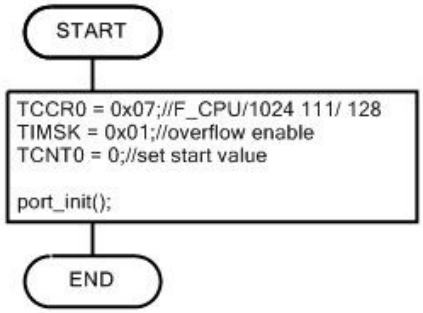


Attēls 11 Input Capture Mode: timer ovf int

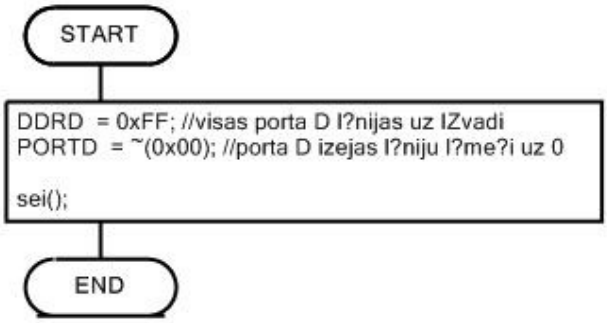
Programmas kods

```
/*********************************************************************/
#define F_CPU 14745600UL        //cycles per second; repeats F_CPU times per second
/********* Standarta C un specialo AVR bibliot?ku iek?au?ana**********/
#include <avr/io.h>
#include <avr/interrupt.h>
/******************** Portu inicializ?cijas funkcija *****************/
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

        sei();
}
/******************** Rezimu[sakuma vertiibu] uzstadisana ************/
void init_devices(void){
        //timer0 setup
        TCCR0 = 0x05;  //F_CPU/1024 111/ 128
        TIMSK = 0x01;  //overflow enable
        TCNT0 = 0;       //set start value

        port_init();
}
/*****FUNCTION PROTOTYPES*********************************************/
unsigned char _rotl(const unsigned char value, unsigned char shift) ;
/******************** Globalie mainigie*******************************/
volatile unsigned long timer = 0;//no copy's in reg,
                                 //no code opt - as compilator doesnt know-interrupt can change it
/******************** Partraukumi ************************************/
ISR(TIMER0_OVF_vect) {
        timer += 255 * 128;
}
/***************************** Main funkcija ************************/
int main (void){
        init_devices();

        //timer variable
        unsigned char port_data = 0x80;
        while (1){
                if(timer > F_CPU){
                        port_data = _rotl(port_data,1);
                        PORTD = ~port_data;
                        timer = 0;
                }
        }
        return 1;
}
/*********************************************************************/
/***  CIRCULAR SHIFT ****/
unsigned char _rotl(const unsigned char value, unsigned char shift) {
   if ((shift &= sizeof(value)*8 - 1) == 0) return value;
   return (value << shift) | (value >> (sizeof(value)*8 - shift));
   //return (value >> shift) | (value << (sizeof(value)*8 - shift)); //right
}
```

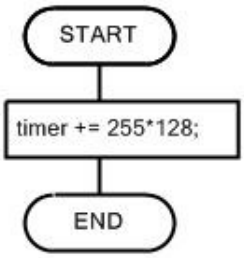
Blokshēma



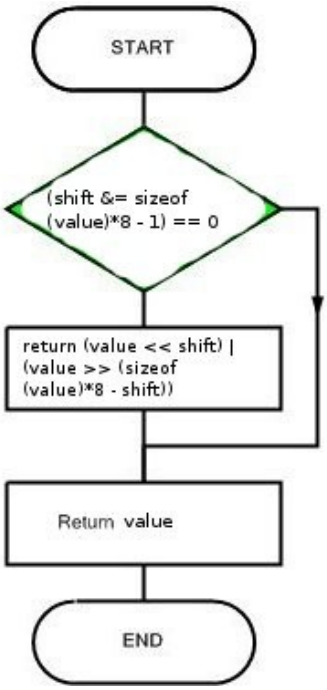
Attēls 12: init_devices()



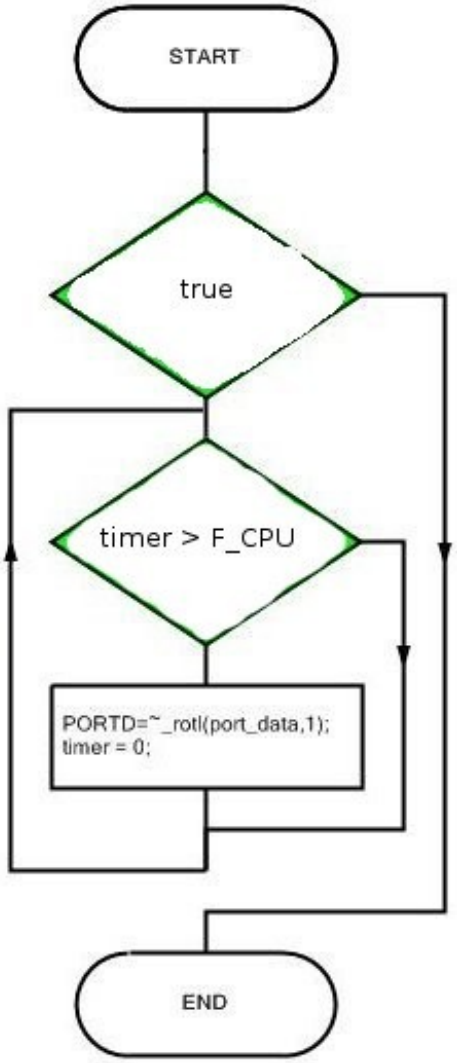
Attēls 13: port_init()



Attēls 14: Timer
Overflow pārtraukums



Attēls 15: _rotl(value, shift)



Attēls 16: main()

Secinājumi

Dotajā darba iepazīstinājos ar pārtraukumu būtību, ka arī iemācījos izmantot Taimeri/Skaitītāju0 un to pārtraukumu. Tika izstrādāta programma izlabojot pirmā laboratorijas darba kodu.

Pirmajā darba, lai kontrolētu diodu pārslēgšanos pēc noteikta laika es izmantoju _delay_ iebūvētu funkciju, kurai ir ierobežota funkcionalitāte. Piem.: _delay_us()(makro, kas izmanto _delay_loop_1() - funkcija, kas izmanto 8-bitu skaitītāju – max 256 iterācijas, kur katra iterācija aizņem trīs CPU taktis) maksimāla aizture ir 768 us/F_CPU Mhz, mūsu gadījuma 11.325 ms. Vai _delay _ms()(izmanto _delay_loop_2 - funkcija ar 16 bitu skaitītāju(aizņēm 4 CPU taktis)), kur maksimāla aizture ir 262.1 ms/F_CPU Mhz, mūsu gadījuma 3.865 s. Šīs funkcijas nav drošas ja optimizācijas nav ieslēgtas, kas rezultē koda neefektivitāte un tādējādi aiztures ilguma ne prognozējamība. Šī iemesla dēļ tos ir ieteicams izmantot nelielam aizturam. Vēl jo vairāk šīs funkcijas izpildes laikā 'paralēli' nekā cita veikt nevar.

Ka jau ir gaidāms šajā darba vajadzēja operēt ar skaitļiem, lai izskaitļotu taimera priekš-dalītāja vērtību vajadzīgam laikam. Piemēram, no paša sakuma es izmantoju volatile(nozīme ka mainīgai var būt mainīts ārpus main cikla, piemēram pārtraukumā. Neoptimizē acīmredzamu rezultātu) unsigned long mainīgo kas ir 4 baitu garš(4,294,967,296), lai varētu salīdzināt to ar iekšēja pulksteņa frekvenci kas ietilpst šajā diapazona. Tagad es zinu ka varēja rīkoties dažādi: izmantot 16 bitu skaitītāju un pa tiešo to salīdzināt ar vērtību, kas atbilstu vienai sekundei, vai izmantojot priekš-dalītāju samazināt frekvenci līdz iespējamam minimumam(ar priekšdalītāju 1024 frekvence ir 14,4 Khz, jeb 14400 taimera taktis uz sekundi(neieskaitot 0)), kas ļautu izmantot mazākus mainīgos. Maksimāla iespējama aizture sanāks izmantojot maksimāli lielu mainīgo (uint64_t, kas ir 8 baitu gara ar maksimālu vērtību 18 446 744 073 709 551 615) ar maksimāli lielu priekš-dalītāju(1024): (18446744073709551615 - 10815) dalot ar 14400 sanāk 1.28102389e15 sekundes paies kamēr mainīgais būs lielāks par (18446744073709551615 - 10815) = 40 593 197.5 Juliana gadu(Julian year) =2 136 532.18 Metonyc ciklu = 1.28102389e22 shake = 0.132348916 manvantaru(“Brahmas diena”), jeb relatīvi ļoti daudz. Programmatiski tik liela aizture(un pat vēl lielāka) ir iespējama(ja mani aprēķini ir pareizi), bet fiziski ierīcē nenodzīvos tik ilgu laiku...

Protams, mainīgo vajag kaut ka palielināt un mēs to darām uz 1 taimera ciklu(ir 128 CPU cikli) vienu reizi. Tā ka viena sekunde ir (F_CPU/128) = 115 200 taimera cikli, tad mums jāpalielina mainīga vērtību par 1 salīdzinot ar (F_CPU/128), vai ar 128 salīdzinot ar F_CPU. Taču mana programma, nezinot ko daru, es to palielināju par 255*128, kas sanāk ir 0,004 sekundes, proporcionāli to attēlojot sanāk:

1 sek – 115 200 ticks
x sek – 452 ticks (= F_CPU / (255 * N(= 128) * 1Hz)),
jeb x = 452 / 115 200 (sek)

Pārtraukumi ir noderīga lieta, kas ļauj veikt paralēli dažādas lietas, piemēram: pabaudīt temperatūru uz čipa un ja tā pārsniedz max apturēt visas darbības. Pārtraukumi reaģē uz noteiktiem notikumiem, dotajā gadījuma uz skaitītajā pārpildīšanos.