Taimeris

Taimera modulis, kas konkrēti runā par skaitītāja moduli, ir ļoti svarīga daļa katram mikrokontrollerim, un vairums kontrolleru piedāvā vienu vai vairākus taimerus ar 8 un/vai 16 bitu izšķirtspēju. Taimerus izmanto dažādu veidu uzdevumos, sākot no vienkāršiem posmu kavējumu mērījumiem līdz viļņu paaudzēm. Galvenā taimera izmantošana ir kā skaitītājs, bet parasti tie arī ļauj lietotājam laikspiedolot (Ciparparaksts, kas fiksē dokumenta saturu laikā, lai viegli varētu atklāt jebkuru pēc tam izdarītu dokumenta izmaiņu) ārējos notikumus, lai ierosinātu pārtraukumus pēc noteiktiem takts cikliem, un pat rada mehāniskās kontroles impulsa platuma pazeminātas frekvences signālus.

2.6.1 Skaitītājs

Katrs taimeris būtībā ir skaitītājs, kas ir vai nu papildināts vai samazināts katrā takts ciklā. Virziens (augšup- vai lejup-skaitītājs) ir vai nu fiksēts vai konfigurējams. Konkrētās vērtības skaits var tikt nolasīts caur skaita reģistru un var tikt iestatīts kā noteikta vērtība lietotājam. Skaita vērtība n taimera izšķirtspējai ir robežās [0, 2^n-1]. Jāuzmanās, kad taimera garums pārsniedz kontrollera vārda garumu, piemēram, kad lietojam 16-bitu taimeri 8-bitu kontrollerī. Šādā gadījumā, piekļūt 16-bitu skaita vērtībai nepieciešams izdarīt divos ceļos, kas var radīt neatbilstošas vērtības. 0x00FF vērtības taimeris pārslēgsies uz 0x0100 nākamajā ciklā. Ja augstais baits tiks nolasīts kā pirmais, bet tikai nākamajā reizē zemais baits, tiks iegūts 0x0000. Ja tiks darīts apgrieztā secībā, iegūs 0x01FF, kas jebkurā gadījumā nav nekas labāks. Lai novērstu šādu problēmu, ATmega16 kontrolleris izmanto bufera reģistru, lai saglabātu taimera augsto baitu. Tātad, ja programma nolasa skaitītāja reģistrā zemo baitu, augstais baits vienlaikus tiks ievietots buferī un nākamajā ciklā varēs to nolasīt. Tāpat, lai ierakstītu jaunu skaitītāja vērtību, augsto baitu nepieciešams rakstīt pirmo (un ievietotam kontrollera buferī), un tiklīdz būs ierakstīts zemais baits, gan augstais, gan zemais baits ierakstīsies skaitītāja reģistrā vienā piegājienā.

Taimeris var kopumā ierosināt pārtraukumu, ja skaita vērtība pārpildās. To var izmantot, lai īstenotu vienkāršus periodiskus signālus, nosakot skaita vērtību attiecīgajā sākotnējā vērtībā, un tad gaidīt pārpildi. Tomēr šī metode nedod precīzu periodu, jo pēc pārpildes, taimerim jāiestata programmas sākotnējā vērtība. Līdz ar to, laiku no pārpildes līdz sākuma vērtībai nepieciešams ielādēt, vai nu taimerim izvērtējot un iekļaujot sākuma vērtībā, vai arī periods būs ilgāks nekā nepieciešams.

Lai izvairītos no šī trūkuma, daži taimeri piedāvā moduļu režīmu, kas automātiski pārlādē starta vērtības, kad taimerī notikusi pārpilde. Cita metode, lai iegūtu precīzu periodu, ir izmantot izvades, ko mēs īsi aprakstīsim, lai salīdzinātu iezīmes.

Lai arī taimeris parasti taktējas ar to pašu avotu, ko izmanto mikrokontrolleris, ne vienmēr tā ir nepieciešamība. Mikrokontrolleris var atļaut vienu vai vairākus atsevišķus avotus, lai liktu strādāt taimerim neatkarīgi.

Sistēmas taktētājs (iekšējais taktētājs)

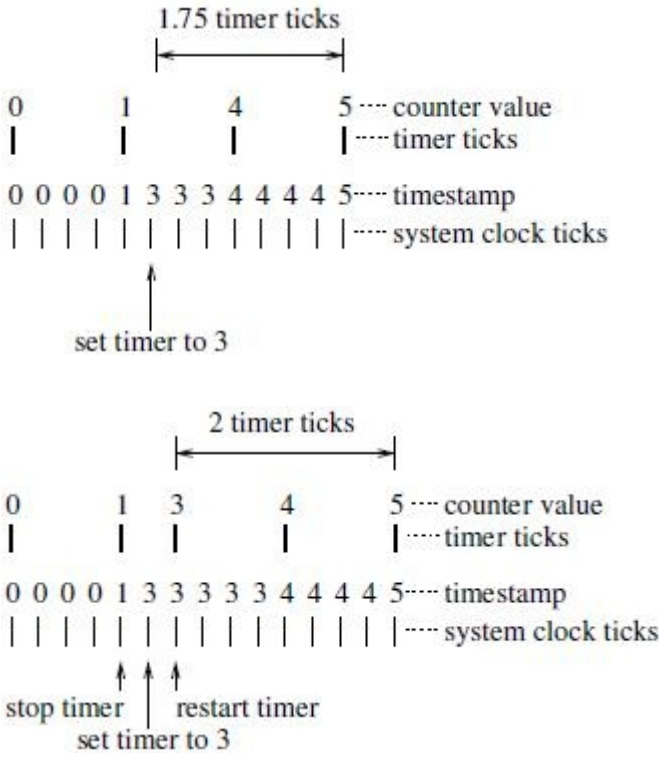
Šajā režīmā taimeris pieaug ar katru sistēmas takti. Tas ir noklusējuma režīms. Ņemiet vērā, ka termins „iekšējs" attiecas tikai uz to faktu, ka tas ir taktēšanas avots visiem kontrollera lietojumiem. Oscilators, kas atbild par to, var būt arī ārējs.

Prescalers

Šis režīms arī izmanto sistēmas takti, bet izfiltrētu cauri prescaleram. Prescaleris pamatā ir kā dažāda garuma skaitītājs (8 vai 10 biti ir to tipiskākās vērtības), kas palielinās ar katru sistēmas takti. Tomēr pats taimeris taktējas ar vienu no prescalera bitiem. Piemēram, ja prescalers pieaug ar katru sistēmas taktētāja fronti, tad tā vismazāk nozīmīgais bits (lsb) dubultos sistēmas takts periodu. Tātad, ja taimeris tiek taktēts ar lsb, tas ietvers samazināto vērtību divniekam un taimeris darbosies tikai ar sistēmas takts puses frekvenci. Blakus lsb esošais bits atkal sadala frekvenci uz pusēm un tā tālāk. Taimera modulis sagādā režīma bitus, kas atļauj lietotājam izvēlēties prescale vērtības (8, 64, 256, ...).

Ir svarīgi saprast, ka, lai arī prescaleris ir lietderīgs, paplašinot taimeru klāstu, tas atsaucas uz zemas kvalitātes taimeru granularitātes izmaksām. Piemēram, ja izmantojat 8-bitu taimeri 1MHz, tā diapazons būs 255 µs un granularitāte 1 µs. Tas pats taimeris ar 1024 prescaleru iegūs aptuveni 260 ms intervālu, bet tā granularitāte būs tikai kāda 1 ms. Tātad prescalēts taimeris spēj izmērīt ilgākus laika brīžus, bet mērījumu kļūda ir lielāka. Ņemot to pašu, prescaled taimeris atļauj programmai uzgaidīt ilgāku apjoma laiku, bet augstāka prescale vērtība nodrošinās taimeram nogaidīt precisely, patvaļīgam periodam. Kā rezultātā, tas parasti ir saprātīgi, kad izmērītos garumus izmanto vismazākajām prescaler vērtībām, kas atbilst vajadzīgajām darbībām, lai iegūtu labāko granularitāti no pieejamām izvēlēm.

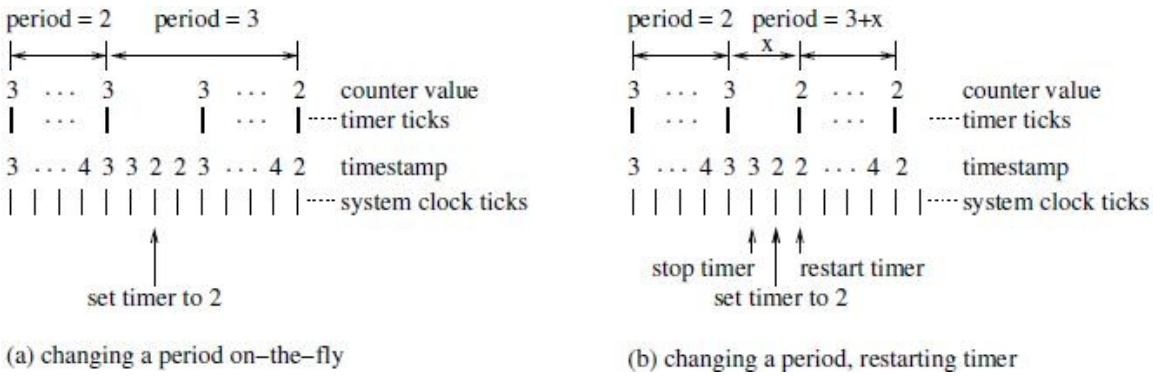
Jāapzinās, kad mēs izmantojam precsaleru un nepieciešams izmainīt taimera vērtību, ir jāizlemj vai mainīt frontes vērtību, vai nē. Kā jau paskaidrojām 2.6.1 sadaļā, ierakstot jaunu vērtību skaitītāja reģistrā strādājošā taimerī var būt problemātiski. Bet pat neapskatot atomiskas piekļuves jautājumu, prescaleri rada citu apdraudējumu: Kad izmantojat prescaleru, tiek samazināts sistēmas takts frekvence taimeram. Ar P prescaleru, tikai katra P-tā takts sistēmas taktētājā izraisa taimera takti. Tomēr darbība joprojām darbojas ar sistēmas taktētāja frekvenci, tādēļ kods, izmainot taimera vērtību, visdrīzāk nesakritīs ar taimera takti, bet būs kaut kur starp divām taimera taktīm. Piemēram, pieņemsim, ka vēlaties iestatīt taimera vērtību uz T. Jūs to darāt laikā k*P+x, kur $x \in (0,P)$ ir nobīdīts no k-tā taimera takts. Taimeris palielinās vērtību uz T+1 laikā (k+1)*P, tātad pirmā takts ilgs x<P sistēmas taktis.



(a) izmainot taimera vērtību darbības laikā (b) izmainot taimera vērtību, taimeri restartējot

2.26 attēls: Taimera vērtības izmainīšana darbības laikā (a) un (b) taimeri apstādinot un restartējot.

Šāda rīcība var būt divējāda, gan kā lāsts, gan kā svētība. 2.26 attēlā vēlamies nogaidīt 2 taktis (no 3 līdz 5). Scenārijā (a) izmainām taimera vērtību darbības laikā, bet (b) situācijā apstādinām taimeri, izmainām vērtību un restartējam taimeri. Protams, (b) ir labāks risinājums šim uzdevumam, jo veido pirmā taimera takti sakrītoši ar sistēmas takti, kurā taimeris sācies.



(a) changing a period on–the–fly

(b) changing a period, restarting timer

2.27 attēls: Taimera perioda izmainīšana moduļa režīmā (a) darbības laikā un (b)- taimeri apstādinot un restartējot.

2.27 attēlā izmantojam taimeru moduļa režīmā, lai ģenerētu periodu. Taimeris ir iestatīts uz 2 periodiem (no 3 līdz 4, to iekļaujot), vēlamies izmainīt periodu uz 3 (no 2 līdz 4). Atkal (a) izmaina taimera vērtību darbības laikā, bet (b) sākumā apstādina taimeri. Šajā gadījumā labāks risinājums ir (a), jo taimeris paliek strādājošs un netiek pazaudēts laiks. (c) versijā vairākas sistēma taktis tiek pazaudētas, apstādinot taimeri, līdz ar to pirmais pārtraukums no jaunā perioda būs garāks.

Ārējais impulss (pulsa akumulators)

Šajā gadījumā taimeris iegūs taktis no ārējā signāla, kas ir savienots ar noteiktu ievades pinu kontrollerī. Taimeris palielina savu skaitītāja vērtību ikreiz, kad, piemēram, tiek novērota kāpjoša fronte ievades pinā. Tā kā ārējie signāli tiek iztverti tieši tāpat kā pārējie ievades signāli, laiks starp frontēm jābūt lielākam par sistēmas takts ciklu.

Ārējie kristāli (asinhronais režīms)

Šeit taimeris taktēts ar ārējo kvarcu, kas savienots ar diviem kontrollera ievades piniem. Šis režīms parasti paredzēts priekš 32.768 kHz takts kristāla, ko var izmantot, lai īstenotu reālā-laika takti (RTC). Skaitītājs tiek palielināts, saskaņā ar ārējo signālu un darbojas asinhroni atšķirībā no visa kontrollera.

2.6.2 Ievades uztveršana

Ievades itveršanas funkcija tiek izmantota, lai laikspiedotu (parasti ārējus) notikumus, kuri var atkal paaugstināt un/vai pazemināt frontes vai līmeņus. Ikreiz, kad notiek notikums, taimeris automātiski nokopē savu pašreizējo skaitītāja vērtību ievades uztveršanas reģistrā, kur to var nolasīt ar programmas palīdzību. Tas arī iestata ievades uztveršanas karogu un var izraisīt pārtraukumu, lai darītu zināmu programmai, ka ir ievades uztveršana ir notikusi. Mikrokontrolleri var piedāvāt vienu vai vairākus pinus ar ievades uztveršanas funkciju.

Ievades uztveršanas īpatnība var arī būt saistīta ar iekšējo notikumu avotiem. Piemēram, ATmega16 var izraisīt ievades uztveršanu no tās analogā komparatora (*Ierīce, kas spēj salīdzināt divus lielumus A un B. Salīdzināšanas rezultātā komparators izstrādā izejas signālus, kuru vērtības atbilst vienai no trim situācijām: A mazāks par B, A vienāds ar B, A lielāks par B.)* moduļa, ļaujot darbībai laikspiedot izmaiņas komparatora izvadē.

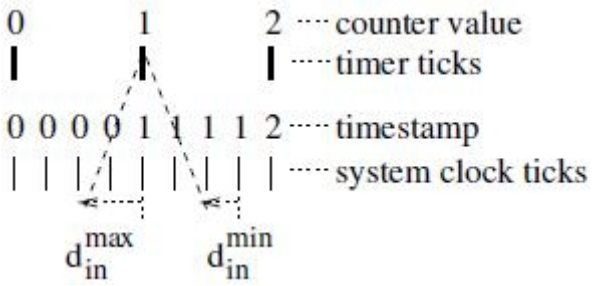
Ņemiet vērā, ka atļaujot pinam ievades uztveršanu, ne vienmēr tam jābūt uzstādītam ievadē. To joprojām var izdarīt ar programmas palīdzību. Patiesībā, ATmega16 atļauj izmantot pinu kā

izvade un pārtrauks ievades uztveršanu, ja programma ģenerēs atbilstošu notikuma stāvokli. Piemēram, tas var tikt izmantots, lai noteiktu aizkavi starp izvades notikumu un sistēmas reakciju.

Tā kā ievades uztveršanas īpatnība tiek izmantota laikspiedolojot notikumus, laikspiedolam jābūt cik vien iespējams precīzam. Kā jau tika paskaidrots 2.6.1 sadaļā, taimerim ir noteikta granularitāte, kas tiek ietekmēta ar prescaleru, un tas ietekmē laikspiedola precizitāti, kā rezultātā

$$t_{ev} - t_{cap} \in (-d_{in}^{\max}, P - 1 - d_{in}^{\min}] \qquad (2.7)$$

takts cikli, kur t_{ev} ir (reālais-)laiks, kas atbilsts laikspiedola notikumam ar ko tas bija laikspiedolots. $d_{in}^{\max}$,attiecīgi arī $d_{in}^{\min}$ ir sliktākais gadījums un labākais ievades kavējumā (sk. 2.3.1 attēlu), un P ir prescaler vērtība. 2.28 attēls ilustrē formulu.



2.28 attēls: Minimālā un maksimālā vērtība prescaler vērtībai, kad P=4.

Attēlā pieņemts, ka notikums laikspiedolots ar 1, tādēļ to jāatzīst vienā no sistēmas takts cikliem, kur laikspiedols bijis 1. Agrāks notikums var tikt laikspiedolots ar 1 $d_{in}^{\max}$ takts ciklā pirms pirmā šāda veida sistēmas takts cikla. Vēlākie notikumu var rasties un joprojām laikspiedoloti ar 1 $d_{in}^{\min}$ ciklos pirms pēdējais sistēmas takts cikls laikspiedolots ar 1. Tieši tādēļ formula ir šāda.

Rezultātā, sliktākajā gadījumā kļūda (kas iegūta takts ciklos), kas mēro periodu starp diviem notikumiem, ir

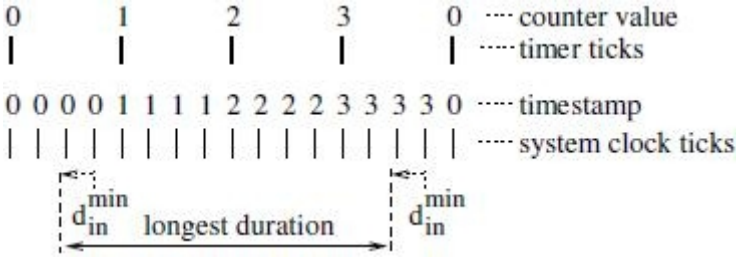
$$d_{in}^{\max} + P - 1 - d_{in}^{\min} . \qquad (2.8)$$

Protams, vajag paturēt prescaler vērtību cik vien mazu iespējams, lai iegūtu visprecīzāko rezultātu.

Visbeidzot, ir interesanti zināt, cik liels var kļūt laiks starp diviem secīgiem notikumiem pirms mēs iegūstam pārpildi. Lielākais šāda veida intervāls noteikti neradīs pārpildi ilgstošā

$$(2^r - 1) * P \qquad (2.9)$$

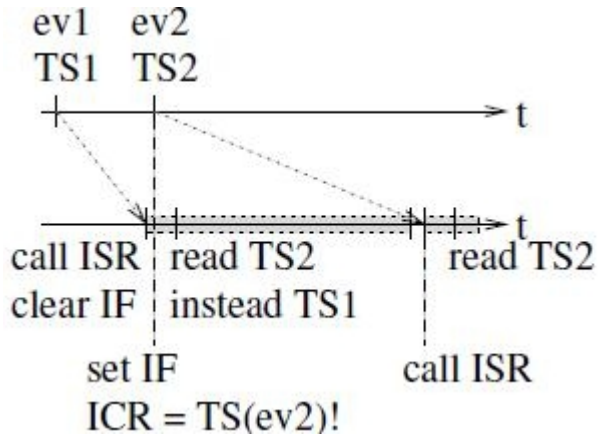
takts ciklā, kur r ir taimera izšķirtspēja. Skatīt 2.29 attēlu.



2.29 attēls: Maksimālais laiks starp diviem notikumiem, kurus var izmērīt bez pārpildes. Prescaler vērtība P=4, taimera izšķirtspēja r=2.

Neatkarīgi no šīm formulām, ir vēl viena interesanta lieta par ievades uztveršanu, ko nepieciešams apzināt, un tam ir sakars ar to kā un kad ievades uztvērēja reģistrs lasa. Kā jau tika paskaidrots, laikspiedols tiek saglabāts ievades uztveršanas reģistrā, kad notikums tiek sastapts, un visticamāk ISR tiks izsaukts, lai nolasītu šo reģistru. Lai nu kā, kāpumi pārtraukumos aizņem vairākus ciklus, tādēļ var gadīties, ka cits uztvertais ievades notikums arī tiks paņemts šajā periodā. Šis otrais notikums atkal rada pašreizējo laikspiedolu, lai tiktu saglabāts uztvertās ievades reģistrā, efektīvi pārrakstot veco vērtību. ISR, kas tika izsaukts pirmajā notikumā, pēc tam nolasīs laikspiedolu otrajam notikumam.

Tā, protams, vēl nav īsti problēma, jo tiek pazaudēts tikai pirmais notikums un reaģēts uz otro. Reālā problēma sākas ar mikrokontrolleru, kas notīra IF bitu automātiski pirms ISR izpildes. Šajā gadījumā otrais notikums var rasties pēc tam, kad ievades uztvērējs noskaidro IF pirms ISR sācis lasīt ievades uztveršanas reģistru. Otrais notikums iestatīs atkal IF un pārrakstīs ievades uztveršanas reģistru, līdz ar to ISR nolasīs otrā notikuma laikspiedolu. Tomēr, kamēr otrais notikums iestatīs IF pa jaunam, tiklīdz ISR pabeidzis savu darbību, tas tiks izsaukts vēlreiz, šoreiz, lai kalpotu otram notikumam, un nolasīs to pašu laikspiedolu, ko pirms tam. Tātad rezultātā tiek reaģēts uz abiem notikumiem, bet kļūdaini tiek piešķirts otrā notikuma laikspiedols abiem notikumiem, skatīt 2.30 attēlu.



2.30 attēls: Abi ISR izsaukumi nolasīs otrā notikuma laikspiedolu.

Ja tā var notikt, tad neko daudz nevar izdarīt, lai risinātu šo problēmu, izņemot to, ka nolasīt ievades uztveršanas reģistru tik ātri cik vien tas iespējams. Var arī pārbaudīt vai secīgi laikspiedoli gadījumā nav vienādi, un ja tā, tad izmest pirmo. Ja tiek izmantots kontrolleris, kas atļauj (vai pat pieprasa) uzstādīt atpakaļ IF, uzstādiet to atpakaļ pēc uztveršanas reģistra nolasīšanas. Bet domājams, ka tas joprojām radīs iespēju pazaudēt kādu notikumu. Labākā aizsardzība ir pārliecināties, ka šādi notikumi nenotiek pārāk tuvu viens otram. Tātad minimālais intervāls starp diviem notikumiem vajadzētu būt garākam par pārtraukuma latentumu (Laika intervāls datorā starp datu pieprasījuma brīdi un brīdi, kad tiek uzsākta datu pārsūtīšana.) plus vēl laiks, kamēr tiek nolasīts uztveršanas reģistrs.

Galvenokārt kā ārējie pārtraukumi, ievades uztveršana cieš no trokšņainiem ievades signāliem. Tādējādi daudzi kontrolleri piedāvā trokšņu slāpētājus, kas parasti ieviesti kā jau tika norādīts 2.3.1 sadaļā (vairāki paraugi tiek paņemti un salīdzināti).

2.6.3 Izvades salīdzinātājs

Izvades uztveršanas iezīme dublicē ievades uztvērumu. Lai vēlāk laikspiedols tiktu saglabāts, kad kaut kas interesants notiek ievades līnijā. Ar izvades uztveršanu, kaut kas notiek izvades līnijā, kad ir sasniegts kāds laika moments. Lai īstenotu šo iezīmi, taimeris piedāvā izvades uztveršanas reģistrā, kur var ievadīt laiku, kurā izvades uztveršanai jānotiek. Ikreiz, kad skaitītāja vērtība sasniedz šo uztveršanas vērtību, tiek iedarbināts izvades notikums. To var automātiski iestatīt vai notīrīt izvades līnijā, vai arī pārslēgt tā stāvokli. Tas var arī neko nedarīt un vienkārši palielināt iekšējo pārtraukumu.

Izvades salīdzinātājs visbiežāk ir ar atjaunošanas iespēju, kas automātiski atjaunina pārtraukumu, kad salīdzinājuma vērtība ir sasniegta. Tas ļauj izveidot periodisku pārtraukumu (vai izvades signālu) ar minimālo mēģinājumu.

2.6.4 Impulsa platuma modulācija

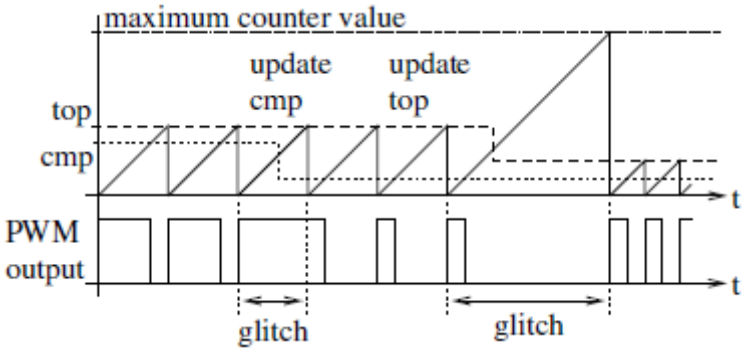
Impulsa platuma modulācijas (PWM) režīms ir speciāls gadījums izvades salīdzinājumā. Tajā taimeris ģenerē periodiskus digitālās izvades signālus ar konfigurējamu pēdējo laiku un periodu. Divi reģistri no galvenā interfeisa uz PWM, vienu periodam (arī sauktu par dežūrējošo ciklu) un vienu priekš pēdējo laiku (vai zemo-laiku). Daži taimeri atļauj lietotājam tikai konfigurēt high-time, un vai nu izmanto visu taimeri, uzskatot to kā periodu, vai arī piedāvā ierobežotu izvēli

iespējamam periodam. Papildus šiem reģistriem, taimera moduļi piedāvā bitus, kas pieejami PWM un iespējams lai kontrolētu režīmu.

PWM signāli ir noderīgi daudzām lietām. Neatkarīgi no to lietojumiem vienkāršā d/a konvertētājā, tie var tikt izmantoti, piemēram, lai īstenotu ABS automašīnās, lai padarītu vājākas gaismas diodes vai ciparu displejus, vai motora kontroles (distancēs, motora soļos, ātruma kontrolē dc motoros).

Iekšējā PWM realizācija patiesībā ir diezgan vienkārša un izmanto tikai skaitītāju un divus salīdzinātājus. Ir iespējamas divas implementēšanas (īstenojumi), viena, izmantojot up-skaitītājs (vai down-skaitītājs) un viens up-down skaitītāja izmantošanai. Turpmākajos skaidrojumos pieņemsim, ka lietotājs nosaka signāla pēdējo laiku, kuru sauksim par salīdzinājuma vērtību, un ka periods ir dots augšējai vērtībai.

Up-skaitītāja versijā, skatīt 2.31 attēlu, izvade ir iestatīta kā augsta, kad skaitītājs sasniedz nulli, un tā ir iestatīta kā zema, kad skaitītājs sasniedz salīdzinājuma vērtību. Tiklīdz augšējā vērtība ir sasniegta, skaitītājs atjaunojas uz nulli. Šīs metodes priekšrocība ir resursu efektivitāte. Tomēr, ja skaitītāju un augšējās vērtības var atjaunināt ar vajadzīgo ciklu, jūs varat radīt pēkšņus bojājumus PWM signālā, kas ir spēkā pagaidu cikliem. Piemēram, ja tiks iestatīta augšējā vērtība zemāka nekā pašreizējā skaitītāja vērtība, tad taimeris skaitīs vienlaikus cauri visam apgabalam pirms tiks pārslēgts uz vajadzīgo ciklu.

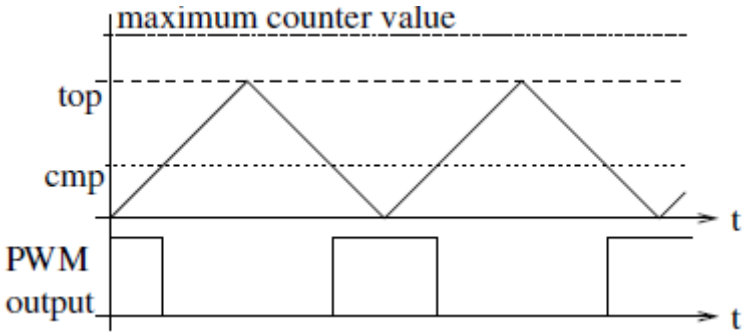


2.31 attēls: PWM signāls ģenerēts, izmantojot up-skaitītāju un rezultē unbuffered atjauninājumus.

Kontrollerus, kurus izmanto šādai metodei, līdz ar to var pārņemt tikai jaunās augšējās vērtības un tās salīdzināt, kad skaitītājs ir sasniedzis nulli. Ja tiks iestatīta skaitītāja vērtība pārsniedzot augšējo vērtību, izvadē būs pastāvīgi augsts līmenis.

Up-down-skaitītāja versijā, kuru varat redzēt 2.32 attēlā, skaitītājs sākumā pieskaita no nulles līdz augšējai vērtībai un pēc tam pārslēdz virzienu, un nonāk atpakaļ lejā pie nulles. Skaitītājs sākas iestatot izvadi kā augstu un sāk skaitīt no nulles. Ikreiz, kad salīdzinājuma vērtība tiek sasniegta upcount, izvade iestata sevi kā zemu. Kad salīdzinājuma vērtība ir atkal atrasta downcount, izvade iestatās atpakaļ kā augsta. Kā varat redzēt, tas summējas simetriski skaistā signālā ar periodu, kuru var dubultot, ja vien tas ir tīrs up-skaitītājs. Atkal salīdzinājuma

asinhronie atjauninājumi vai augšējā vērtība var izveidot pēkšņu bojājumu, līdz ar to kontrollerim jāsaglabā vērtības, kamēr nulle vēl nav sasniegta.



2.32 attēls: PWM signāls ģenerējas ar up-down-skaitītāja palīdzību

Abās versijās sasniedzamais periods ir noteikts ar taimera izšķirtspēju. Ja high laiks ir iestatīts uz nulli vai uz (vai augstāk) augšējās vērtības, tas parasti rezultējas pastāvīgi zemā vai augstā signālā.