AGH
Akademia Górniczo-
Hutnicza w Krakowie
Katedra Elektroniki WIET

Technika Microprocesorowa

Laboratorium 6

Timery i liczniki

Auhor: Paweł Russek
Tłumaczenie: Ernest Jamro
http://www.fpga.agh.edu.pl/tm
wer. 27.05.15

1. Wstęp

1.1. Cel ćwiczenia

Celem tego laboratorium jest zapoznanie się z timerami (czasomierzami) i licznikami znajdującymi się w mikrokontrolerach AVR. Zaprezentowane zostaną liczniki i timery ośmiobitowe i szesnastobitowe oraz związane z nimi rejestry sterujące. Zostaną podane przykłady programowania preskalera (wstępnego dzielnika częstotliwości), opóźnienia czasowego oraz generatora fali prostokątnej. Po zakończeniu laboratorium student powinien umieć użyć timera w celu uzyskania odpowiedniego opóźnienia czasowego lub generacji fali prostokątnej o żądanej częstotliwości na wyjściu mikrokontrolera.

1.2. Wymagania wstępne

Zaliczenie ćwiczeń laboratoryjnych 1, 2, 3, 4 oraz 5.

2. Nowe instrukcje asemblera

2.1. Instrukcja: ustaw bity w rejestrze (set bits in register - sbr)

Instrukcja *sbr* ustawia określone bity w rejestrze *Rd.* Wybrane bity określają jedynki w podanej masce *K*. Operacja jest równoznaczna wykonaniu operacji logicznej LUB (OR) na indywidualnych bitach rejestru *Rd* i 8-bitowej stałej *K*. Wynik operacji jest zapisywany do rejestru *Rd*.

sbr Rd, K ;$16 \le d \le 31$, $0 \le K \le 255$

Przykład:

```
sbr r16, 3      ; Set bits 0 and 1 in r16
sbr r17, $F0    ; Set 4 Most Significant Bits (MSBs) in r17
```

2.2. Instrukcja: zeruj bity w rejestrze (clear bits in register - cbr)

Instrukcja *cbr* zeruje określone bity w rejestrze *Rd.* Wybrane bity określają jedynki w podanej masce *K*. Operacja jest równoważna operacji logicznej I (AND) na bitach rejestru *Rd* i zanegowanych bitach 8-bitowej stałej *K*. Wynik operacji jest zapisywany do rejestru *Rd*.

cbr Rd, K ;$16 \le d \le 31$, $0 \le K \le 255$

Przykład:

```
cbr r16, $F0    ; Clear upper nibble of r16
cbr r18, 1      ; Clear bit 0 in r18
```

3. Timery i liczniki

Timery (czasomierze) i liczniki są bardzo często wykorzystywane w rzeczywistych aplikacjach. Wiele programów wymaga zliczania zdarzeń, generacji fali prostokątnej o określonej częstotliwości i współczynniku wypełnienia, pomiaru różnicy czasu pomiędzy zdarzeniami lub generacji opóźnienia. W konsekwencji w mikrokontrolerach znajdują się dedykowane liczniki i timery. Warto podkreślić, że elementy te pracują niezależne od procesora, czyli procesor może wykonywać niezależny kod programu w czasie kiedy, np. następuje zliczanie impulsów na pinie wejściowym. Pomiar czasu i liczby impulsów realizowany jest nie w kodzie programu procesora, ale poprzez dedykowane układy liczników (podobne liczniki były projektowane na przedmiocie Technika Cyfrowa). Dzięki temu szybkość i dokładność pracy samego licznika jest większa. Zliczanie programowe nie może być tak samo dokładne bo mikrokontroler często musi równolegle wykonywać wiele niezależnych zadań. Interakcja pomiędzy procesorem i timerem / licznikiem odbywa się poprzez odczyt/zapis odpowiednich rejestrów kontrolnych i statusowych.

W przypadku kiedy chcemy zliczać zdarzenia (zmiany stanu z 0 na 1 lub 1 na 0), należy podłączyć źródło zdarzeń do odpowiedniego wyprowadzenia mikrokontrolera, które pełni funkcję wejścia zegara licznika. W przypadku kiedy chcemy wygenerować odpowiednie opóźnienie, należy wejście zegara licznika podłączyć do źródła o znanej częstotliwości (np. oscylatora). W takim wypadku zmierzony czas jest równy iloczynowi stanu licznika oraz znanego okresu zegara oscylatora.

Każdy timer / licznik posiada flagę przepełnienia, która jest ustawiana w przypadku przekroczenia zakresu licznika. Datego metodą generacji programowalnego opóźnienia jest wstępne załadowanie licznika odpowiednią wartością początkową i czekanie na wystawienie flagi przepełnienia. Na przykład dla zegara o częstotliwości 1MHz i ośmiobitowego licznika, aby otrzymać opóźnienie 3 μs należy wpisać do licznika stan początkowy 0xFD i czekać aż zostanie ustawiona flaga przepełnienia, co wymaga trzech taktów zegara.

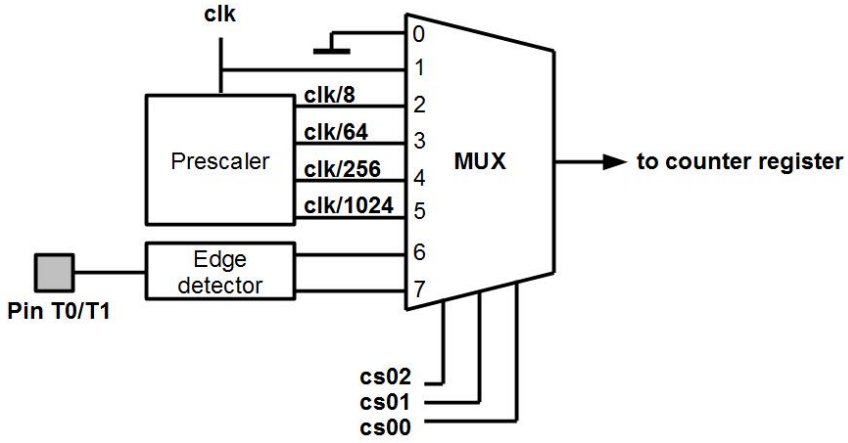
Niektóre mikrokontrolery AVR posiadają nawet do 6 timerów / liczników. ARMega32 posiada ich trzy: *Timer0*, *Timer1* oraz *Timer2*. *Timer0* i *Timer2* są ośmiobitowe, natomiast *Timer1* jest 16-bitowy. W tym laboratorium używane będą tylko *Timer0* i *Timer1*, niemniej użycie innych timerów AVR jest bardzo podobne. Nazwy rejestrów liczników (*counter*) w ATMega32 są następujące: ***tcnt0***, ***tcnt1*** oraz ***tcnt2,*** ogólnie ***tcntN,*** gdzie *N* – numer licznika / timera, liczba od 0 do 2.

3.1. Źródła sygnału zegarowego dla liczników

Każdy licznik / timer potrzebuje sygnału zegarowego, który może być albo sygnałem zewnętrznym albo wewnętrznym mikrokontrolera. Jeżeli używamy sygnału wewnętrznego, to jego częstotliwość może być odpowiednio zredukowana dzięki modułowi preskalera (dzielnika). Schemat połączeń przy wyborze sygnału zegarowego dla *Timer0* i *Timer2* jest przedstawiony na rysunku 1.

Wyprowadzenie *T0* i *T1* stanowią zewnętrzne źródła dla *Timer0* i *Timer1*. Wyprowadzenie *T0* jest współdzielone z bitem 0 portu B, podobnie wyprowadzenie *T1* to bit 1 portu B. Dla *Timer2* sygnałem zewnętrznym zegara jest wyprowadzenie dodatkowego oscylatora kwarcowego. Bity *csN0*, *csN1*, *csN2* (zob. rys. 1) to bity rejestru kontrolnego licznika *timerN* o nazwie *tccrN* (zob. rozdział 6 z opisem rejestrów). Wpisanie do rejestru *tccrN* wartości 0 powoduje wyłączenie licznika *timerN.* Wpisanie wartości 1 do 5 powoduje wybranie wewnętrznego sygnału zegarowego o różnej częstotliwości (od 1 MHz do ~1kHz). Wpisanie

wartości 6 lub 7 powoduje wybór zewnętrznego sygnału zegarowego, 6 – reakcja na zbocze narastające, 7– opadające.



Rys. 1. Schemat wyboru zegara wejściowego

Przykład:

```
; *** Select clk÷256 as a clock source for Timer0 ***
; First solution
    .include"m32def.inc"          ;contains definitions of tccr0, and cs00, cs01, cs02
                                  ;tccr0=0x33, cs00=0, cs01=1, and cs02=2
    in r16, tccr0                 ;read the content of tccr0 register
    cbr r16, (1<<cs00) | (1<<cs01)  ;clear bit cs00 and bit cs01
    sbr r16, (1<<cs02)            ;set bit cs02 to one
    out tccr0, r16                ;update tccr0


; Alternative solution that does not preserve tccr0 previous content
    .include"m32def.inc"
    ldi r16, (1<<cs02)            ;r16=0b00000100
    out tccr0, r16                ;set tccr0
```

3.2. Flaga przepełnienia timera (timer overflow - tovN)

Flaga przepełnienia timera (*tovN*) jest częścią rejestru ***tifr*** (*timer interrupt flag register* - zob. rozdział 6 z opisem rejestrów). W normalnym trybie pracy timera flaga ta jest ustawiana podczas przekroczenia wartości maksymalnej licznika (po przejściu ze stanu 0xFF do stanu 0 dla *timer0*). Flaga *tovN* może być monitorowana przez program w celu odpowiedniego uruchomienia zadań zależnych czasowo. Flaga *tovN* zachowuje się więc jak dodatkowy bit przeniesienia, jest ona tylko ustawiana. Zerowanie flagi *tovN* odbywa się przez (programowy) zapis wartości 1 do tej flagi.

Przykład

```
loop:                   ; wait for timer0 to overflow
    in r16, tifr        ; load tifr in register 16
```

```
sbrs r16, tov0          ; skip next instruction if bit (zero) in register
; (r16) is set
rjmp loop               ; jump to loop if no Timer0 overflow occurred
; Event Service Code starts here
ldi r16, (1<<tov0)
out tifr, r16           ; clear tov0
…                       ;continue the program …
```

Uwaga

W asemblerze flagi są zdefiniowane jako wartości z numerami bitów odpowiedniego rejestru. Aby ustawić bit *N* w rejestrze *R* należy użyć następującej składni: *ldi R, (1<<N) ; przesuń 0b00000001 o N bitów w lewo i załaduj do rejestru R* gdzie '<<' operacja przesunięcia logicznego w lewo Możliwe jest łączenie znaków '<<' i operacji bitowej OR (znak '|') w celu ustawienia więcej niż jednego bitu, np. *ldi R, (1<<N) | (1<<M) ; ustaw bit N i M w rejestrze R*

Ćwiczenie 6.1

Napisz program który wykrywa piąte wystąpienie zdarzenia na wyprowadzeniu *T0*.

1. Użyj flagi przepełnienia *tov0* i rejestru *tcnt0* w celu wstępnego załadowania licznika.
2. Używaj przycisków w celu generacji zdarzeń.
3. Zmień stan diody LED po wystąpieniu piątego zdarzenia.
4. Uruchom program na ZL3AVR.
5. Zapisz program jako 'timer0_fifth.asm'

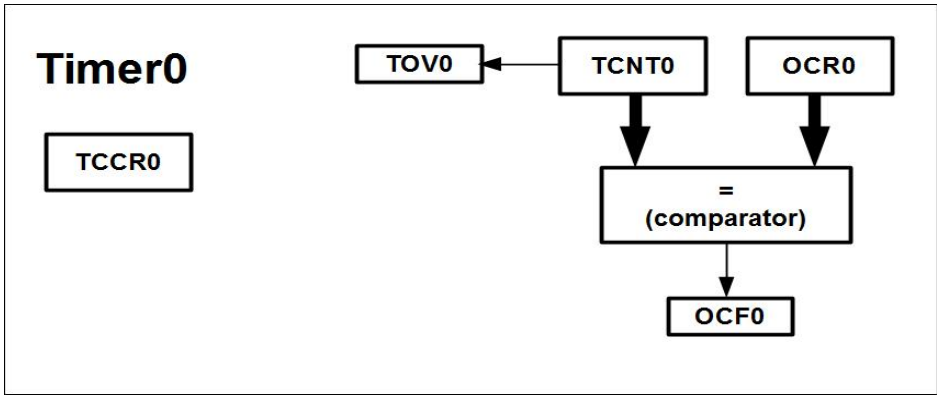
3.3. Rejestr porównania wyjścia (o*utput compare register - ocrN)*

Każdy timer posiada rejestr *ocr*N (Output Compare Register), którego zawartość jest porównywana z zawartością licznika *tcnt*N. W przypadku kiedy te dwa rejestry są równe ustawiana jest flaga *ocf*N (Output Compare Flag – zob. Rys. 2), która wchodzi w skład rejestru *tifr.* Flaga ta może być wyzerowana poprzez program, dzięki wykonaniu operacji zapisu wartości 1 do bitu tej flagi.

3.4. Flaga zerowania timera (clear timer on compare match ctcN)

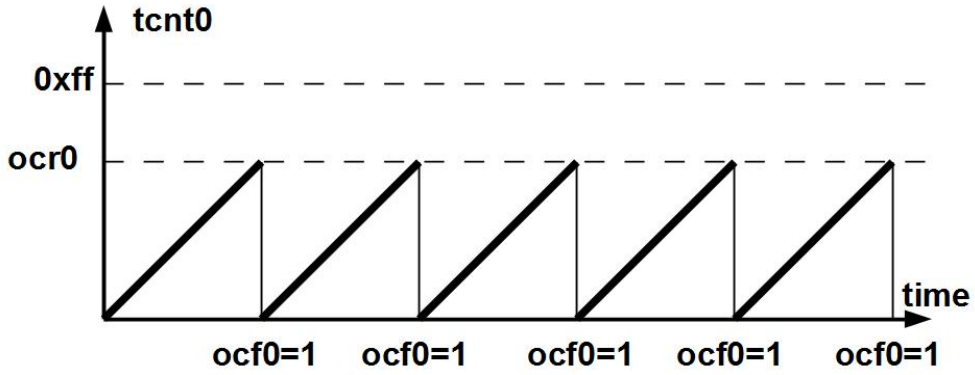
Razem z rejestrem *Output Compare Register* (OCR) można użyć trybu pracy porównywania z zerowaniem licznika (*Clear Timer on Compare - CTC*). W tym trybie licznik jest zerowany w momencie kiedy wartość licznika (*tcntN*) jest równa wartości rejestru *ocrN* (zob. Rys. 3). Czyli rejestr *ocrN* wyznacza maksymalną wartość licznika, w konsekwencji licznik staje się licznikiem modulo (*ocrN* +1). Tryb ten pozwala na lepszą kontrolę pracy licznika ponieważ

programowe zerowanie licznika w przypadku ustawienia flagi *ocrN* może być zbyt czasochłonne i generować niezamierzone opóźniania.



Rys. 2. Schemat generacji flagi OCF0

Aby ustawić timer w tryb pracy z zerowaniem (CTC) należy ustawić flagę *ctcN* w rejestrze *tccrN*.



Rys. 3. Diagram czasowy licznika w trybie z zerowaniem (CTC)

Przykład:

```
;Delay execution by 32*1024*clock cycles
; Clear timer on compare match / Timer Clock =
; system clock / 1024
ldi r16, (1<<ctc0)|(1<<cs02)|(1<<cs00)
out tccr0, r16          ; timer clock = system clock/1024
ldi r16, 1<<ocf0
out tifr, r16           ; clear ocf0
ldi r16, 32
out ocr0, r16           ; set output compare value to 32
ldi r16,0
out tcnt0, r16          ;clear timer0
```

```
again: in r16, tifr        ;delay loop
       sbrs r16, 1<<ocf0   ;wait for ocf0 flag
       rjmp again          ;delay if time not elapsed
       ....                ;do something after time delay
```

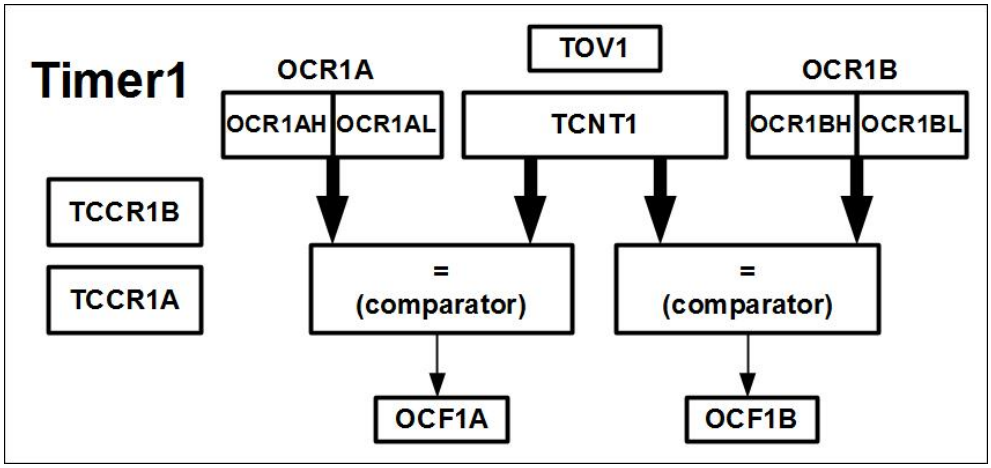
Ćwiczenie 6.2

Przepisz przykład 6.1 tak aby użyć *Timer0* i tryb z zerowaniem CTC w celu liczenia zdarzeń. Zachowaj program jako "timer0_ctc.asm".

4. Timer1

Timer1 jest 16-bitowy, jednak jego 16-bitowe rejestry są zapisywane i odczytywane bajtowo (procesor jest 8-bitowy). W konsekwencji rejestr *tcnt1* jest podzielony na dwa rejestry *tcnt1l* i *tcnt2h. Timer1* posiada również dwa rejestry kontrolne *tccr1a* i *tccr1b.* Flaga *tov1* (*timer overflow*) jest ustawiana w momencie kiedy następuje przepełnienie licznika.

Timer1 posiada dwa niezależne rejestry porównywania wyjścia (*Output Compare Register*): *ocr1a* i *ocr1b.* W konsekwencji *Timer1* ma dwie flagi ocf (*Output Compare Flags*): *ocf1a* i *ocf1b.* Ponieważ *Timer1* jest 16-bitowy również rejestry ocr1a i ocr1b są 16-bitowe, czyli każdy z nich jest podzielony na dwa rejestry ośmiobitowe: (o*cr1ah* i o*cr1al*) oraz (*ocr1bh* i o*cr1bl*), co przedstawia rysunek 4.



Rys. 4. Rejestry *Timer1*

Timer1 jest 16-bitowym licznikiem synchronicznym, co oznacza, że jego zegarem wejściowym jest zegar systemowy, wyjście preskalera lub też zegar zewnętrzny zsynchronizowany względem zegara systemowego (przez specjalną logikę detekcji zbocza zegara zewnętrznego). W celu uzyskania stabilnej pracy systemu, w którym *Timer1* jest 16-bitowy a odczyt / zapis rejestrów jest 8-bitowy konieczny jest równoczesny odczyt / zapis 16-bitowy, co wymaga użycia dodatkowego rejestru tymczasowego (*temp*). W konsekwencji odczyt / zapis rejestrów wymaga odpowiedniej sekwencji. Podczas odczytu *tcnt1* należy najpierw odczytać młodszy bajt (a potem starszy). Podczas zapisu należy najpierw zapisać starszy bajt.

Przykład:

```
;read operation of a 16-bit register can look like this:
```

```
in r16, tcnt1l
in r17, tcnt1h
;access the registers in the opposite order:
out tcnt1h, r17
out  tcnt1l, r16
```

Bity *cs12*, *cs11*, i *cs10,* które wybierają źródło sygnału zegarowego znajdują się w rejestrze *tccr1b*. Bit *ctc1* (*Clear Timer on Compare*) znajduje się również w rejestrze *tccr1b.* W trybie z zerowaniem (CTC), rejestr *ocr1a* jest używany do zerowania rejestru *tcnt1*.

Przykład:

```
;prepare timer for delay measure of  1777*1024 system clock cycles
;setting Timer1 in CTC mode, timer clock=system clock÷1024
ldi r16, (1<<ctc1)|(1<<cs12)|(1<<cs10)
out tccr1b, r16
; set output compare value to 1777
ldi r16, high(1777)
out ocr1ah, r16
ldi r16, low(1777)
out ocr1al, r16
ldi r16, 0x00
out tcnt1h, r16          ;clear tcnt1
out  tcnt1l, r16
again: in r16, tifr      ;test ocf1a  flag
sbrs r16, 1<<ocf1a       ;test   flag
rjmp again
;start delayed procedure
ldi r16,(1<<ocf1a)       ;clear  ocf1a flag
out tifr, r16
…                        ;do something here
```

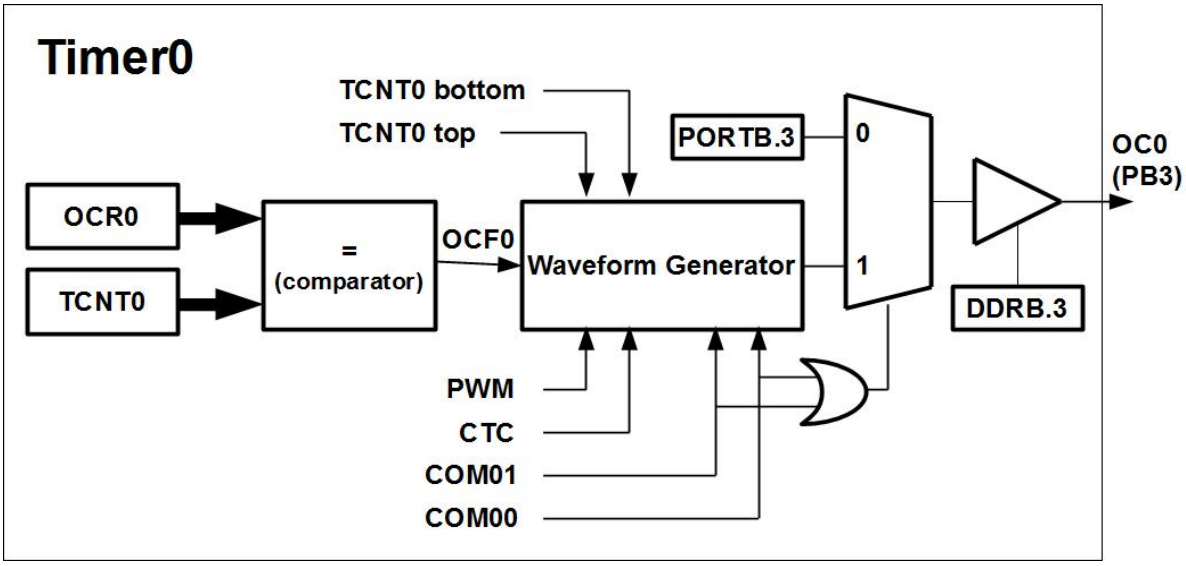
Ćwiczenie 6.3

Zakładając częstotliwość zegara systemowego 1 MHz, napisz program który przy użyciu *Timer1* zapala i gasi diodę LED z częstotliwością 1 Hz. Zapisz program jako “timer1_1sec.asm”.

5. Generator fali prostokątnej

W każdym timerze AVR znajduje się generator fali prostokątnej, którego wyjście jest podawane na wyprowadzenie *ocN*. Bity: *pwm, ctc, csN1, csN0* określają tryb pracy generatora. Generator może odpowiednio reagować w momencie kiedy stan licznika (rejestr *tcntN*) osiąga

swoją maksymalną wartość (*top*), minimalną wartość (*bottom*) lub równa się wartości rejestru OCR (*Output Compare Register*), co przedstawia rysunek 5.



Rys. 5. Schemat blokowy generatora fali prostokątnej *Timer0*

W ATMega32, dla *Timer0* bit *oc0* jest współdzielony z wyprowadzeniem 3 portu B. Bity *com01 oraz com00* znajdują się w rejestrze *tccr0*. Funkcje bitów *com01* oraz *com00* są podane w tabeli 1.

Tab.1. Funkcje bitów *com01* oraz *com00*

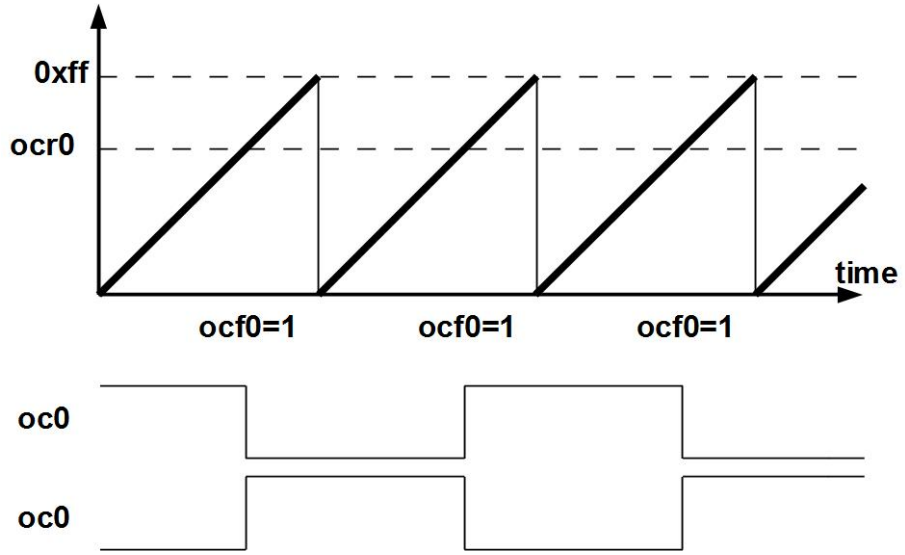
		Non-PWM mode	***Fast PWM mode***	***Phase correct PWM mode***
com 01	***com 00***	***Description pwm=0***	***Description pwm=1, ctc=0***	***Description pwm=1, ctc=1***
0	0	Normal port B operation. *Oc0* is disconnected.		
0	1	Toggle *oc0* on compare match	Reserved	
1	0	Clear *oc0* on compare match	Clear *oc0* on compare match, set *oc0* at bottom,(non-inverting mode)	Clear *oc0* on compare match when up-counting. Set *oc0* on compare match when downcounting.
1	1	Set *oc0* on compare match	Set *oc0* on compare match, clear *oc0* at bottom, (inverting mode)	Set *oc0* on compare match when up-counting. Clear *oc0* on compare match when downcounting.

5.1. Tryb Non-PWM

Aby wygenerować falę prostokątną o wypełnieniu 50% możemy ustawić timer w tryb pracy Non-PWM (*com01:00*=01). Wyprowadzenie *oc0* będzie na przemian kasowane i ustawiane (*toggle*) w momencie kiedy stan licznika równa się wartości rejestru OCR (*Output Compare Register*).

a) Bez zerowania (non-clear on Compare - ctc=0)

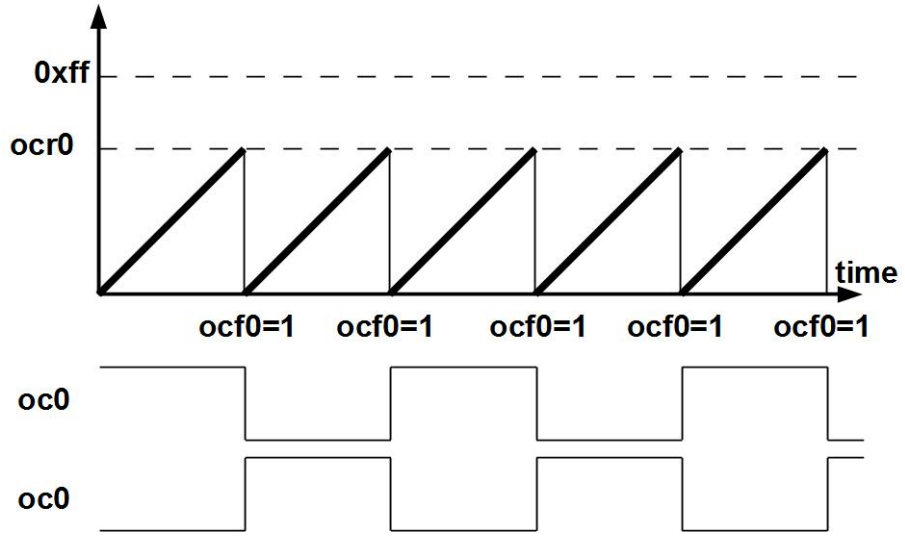
W przypadku kiedy bit *ctc* jest równy zero licznik nie ulega wyzerowaniu, licznik jest kasowany dopiero w momencie kiedy nastąpi jego przepełnienie, czyli dojdzie on do swojej maksymalnej wartości równej 0xFF. Przebieg wyjścia *oc0* został przedstawiony na rysunku 6.



Rys. 6. Przebiegi dla trybu bez zerowania, *oc0* – z inwersją lub bez inwersji

b) Tryb z zerowaniem (clear on compare match - ctc=1)

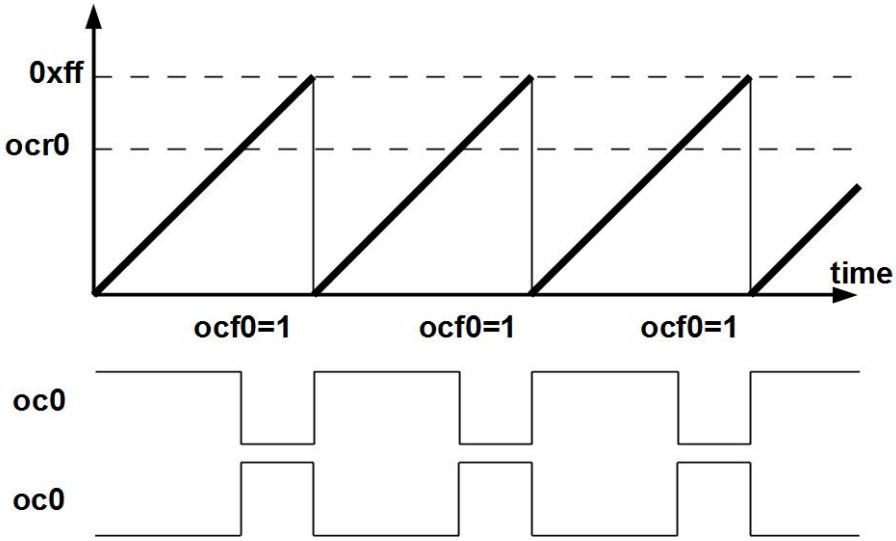
Dla bitu *ctc* równego 1, następuje zerowanie licznika w przypadku kiedy stan licznika równa się wartości rejestru *OCR* (*Output Compare Register*). W tym trybie częstotliwość generowanej fali ulega zmianie, zgodnie z ustawieniem rejestru OCR. Dla wyższych wartości rejestru OCR okres generowanej fali ulega wydłużeniu (częstotliwość ulega zmniejszeniu) i vice versa (zob. rysunek 7).



Rys. 7. Przebiegi dla trybu z zerowaniem, *oc0* – z inwersją lub bez inwersji

5.2. Tryb szybki PWM (Fast PWM)

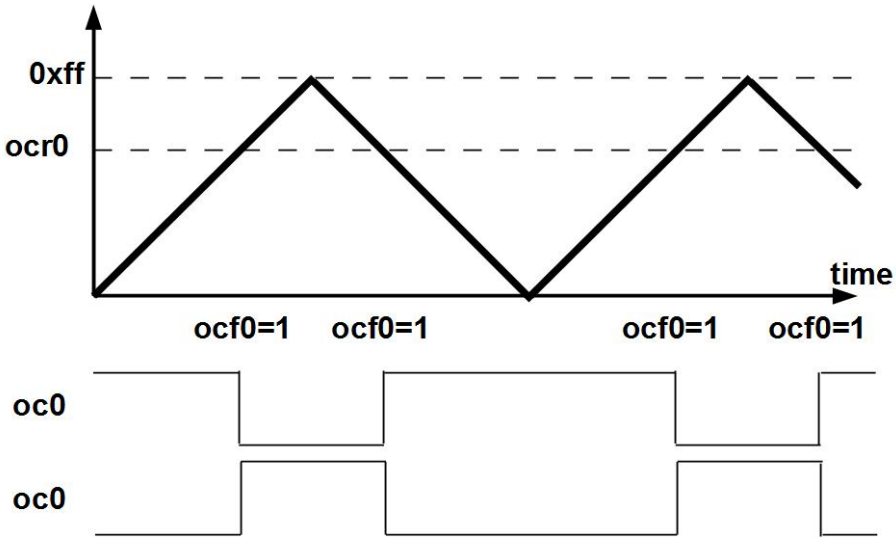
Aby ustawić tryb pracy PWM (Pulse Width Modulation) należy ustawić bit *pwm* w rejestrze *tccr0.* W przypadku kiedy bit *ctc* w rejestrze *tccr0* jest wyzerowany, timer pracuje w tzw. trybie szybki PWM. W trybie tym licznik pracuje bez zerowania a stan wyjścia *oc0* zależy od wyniku porównania wartości licznika z wartością rejestru *OCR*. W trybie bez inwersji (*com01:00*= 10) wyprowadzenie *oc0* jest równe '1' jeśli stan licznika jest większy niż stan rejestru *OCR* (w przeciwnym wypadu jest równe '0'). W trybie z inwersją (*com01:00*=11) stan wyprowadzenia *oc0* jest równy '0' jeśli stan licznika jest większy niż stan rejestru *OCR*. W konsekwencji, w trybie bez inwersji współczynnik wypełnienia wynosi: (*ocr*+1)/256 (jest zawsze większy od 0). W trybie z inwersją współczynnik wypełnienia jest zawsze mniejszy od 1 i wynosi: (255-*ocr)/*256. Przedstawia to rysunek 8.



Rys. 8. Tryb szybki PWM (bez lub z inversją)

5.3. Tryb PWM z korekcją fazy (Phase correct PWM)

Aby ustawić tryb pracy PWM należy ustawić bit *pwm* w rejestrze *tccr0.* W przypadku kiedy bit *ctc* w rejestrze *tccr0* jest ustawiony, timer pracuje w trybie PWM z korekcją fazy. W trybie tym licznik *tcnt0* zlicza w górę i w dół. W trybie szybki PWM faza fali prostokątnej różni się dla różnego współczynnika wypełnienia. W trybie z korekcją fazy, faza fali prostokątnej pozostaje bez zmian.



Rys. 9. Tryb PWM z korekcją fazy

6. Podsumowanie

W celu podsumowania podano funkcje poszczególnych bitów różnych rejestrów. Jest wiele dodatkowych opcji pracy timera, które nie zostały opisane w niniejszej instrukcji, dlatego dociekliwym czytelnikom polecamy przeczytanie instrukcji ATMega32.

6.1. TIFR

Timer interrupt flag register (tifr) (read/write)

7	6	5	4	3	2	1	0
ocf2	tov2	icf1	ocf1a	ocf1b	tov1	ocf0	tov0

tov0 D0 Timer0 overflow flag.

ocf0 D1 Timer0 output compare match flag.

tov1 D2 Timer1 overflow flag.

ocf1a D3 Timer1 output compare A match flag.

ocf1b D4 Timer1 output compare B match flag.

icf D5 Input capture flag.

tov2 D6 Timer2 overflow flag.

ocf2 D7 Timer2 output compare match flag.

6.2. TCCR0

Timer/counter 0 control register (tccr0)

7	6	5	4	3	2	1	0
	pwm	com02	com00	ctc0	cs02	cs01	cs00

cs00 *D0* *clock source control bit 1 for timer 0*

cs01 *D1* *clock source control bit 2 for timer 0*

cs02 *D2* *clock source control bit 3 for timer 0*

ctc0 *D4* *clear timer on compare match bit for timer 0. In PWM mode (pwm=1) it switches between fast PWM and phase correct PWM modes.*

com00 *D5* *OcO pin operating mode bit 1*

com01 *D6* *OcO pin operating mode bit 1*

pwm *D7* *set PWM mode for waveform generator*

6.3. TCCR1b

Timer/counter 1 control register B (tccr1b)

7	6	5	4	3	2	1	0
				ctc1	cs12	cs11	cs10

cs10 *D0* *clock source control bit 1 for timer 1*

cs11 *D1* *clock source control bit 2 for timer 1*

cs12 *D2* *clock source control bit 3 for timer 1*

ctc1 *D4* *clear timer on compare match bit for timer 1*

Ćwiczenie 6.4

Napisz program, który umożliwia ustawianie intensywności świecenia diody LED. Użyj *timer0* i trybu PWM. Zmiana poziomu jasności diody LED odbywa się poprzez rejestr *ocr0.* Użyj dwóch przycisków w celu zmiany intensywności świecenia (jaśniej / ciemniej). Zapisz program jako “led_dimmer.asm”.