Timery w AVR: nastawy i opis funkcjonalny

Celem tego artykułu jest opis funkcjonowania oraz sposobów konfiguracji i wykorzystania układów tzw. timerów w mikrokontrolerach z rodziny AVR. Rozpoczynając od ogólnego opisu, poprzez przykładowe programy, postaram się wytłumaczyć jak wykorzystać wbudowany w strukturę mikrokontrolera AVR timer dla własnych potrzeb. W przykładach programów posługiwałem się mikrokontrolerem AT90S8535.

Wprowadzenie

Trudno jest znaleźć polski odpowiednik słowa „timer". Większość konstruktorów, którzy mieli już do czynienia z mikrokontrolerami doskonale wiedzą jaki to jest rodzaj urządzenia. Pozwólcie więc, że będę się tym słowem posługiwał bez poszukiwania odpowiednika w naszym, ojczystym języku.
Timer to prosty układ liczący, najczęściej o rozdzielczości 8 lub 16 bitów. Niech nie zwiedzie nas jednak prostota jego budowy – z każdym timerem związany jest bowiem szereg różnych zmiennych (najczęściej są to bity rejestru kontrolnego) wpływających na to w jaki sposób będzie on pracował. Często istnieją więc możliwości nastaw kierunku zliczania (w górę lub w dół) oraz wyboru źródła impulsów zegarowych – czy to z otoczenia mikrokontrolera, czy też z wewnętrznego generatora zegarowego lub dołączonego rezonatora kwarcowego (AVR). Programista – elektronik najczęściej używa timera bądź to do zliczania impulsów, bądź to do pomiaru czasu ich trwania albo też do budowy tak zwanego generatora PWM. Będzie o tym mowa w dalszej części artykułu.
Najczęściej jeśli timer wykorzystywany jest do pomiaru czasu trwania impulsu, to jako wzorzec wykorzystuje się wewnętrzny generator zegarowy lub wzorcowy, zewnętrzny sygnał odniesienia. Prowadzi to nas do wniosku, że czas trwania impulsu może być mierzony z dokładnością do czasu trwania impulsów wzorcowych. Stanowią one swego rodzaju jednostkę pomiarową. Najważniejszą jednak cechą timera jest ta, że może on funkcjonować niezależnie od reszty procesów obsługiwanych przez jednostkę centralną mikrokontrolera (abstrahując od konfiguracji bitów kontrolujących pracę timera, która musi być wykonana przez CPU).
Struktury współczesnych mikrokontrolerów wyposażane są w 2 lub 3 układy timerów. Generalnie rodzina AVR (AT90- i ATMega) posiadają dwa timery 8-bitowe i jeden 16-bitowy. W większości zastosowań lepszy jest timer 16-bitowy, jednak dla wielu aplikacji rozdzielczość 8-bitowa jest wystarczająca. Jest ona też lepiej dopasowana do architektury rdzenia (który jest 8-bitowy) i przez to umożliwia znacznie szybsze wykonywanie operacji arytmetycznych czy porównań ze stałymi, czy zmiennymi używanymi przez daną aplikację.
Ze względu na swoją elastyczność, timery mikrokontrolerów AVR mogą być wykorzystywane dla różnych celów. Dalsza część tekstu ma na celu przybliżenie tych zastosowań oraz wytłumaczenie w jaki sposób niezależne układy funkcjonalne komunikują się z CPU mikrokontrolera oraz jak mogą być przezeń wykorzystane.

Sygnalizacja zdarzeń

CPU mikrokontrolera AVR może monitorować do 3 zdarzeń powodowanych przez każdy z timerów. Zdarzenia te sygnalizowane są przez ustawienie odpowiednich bitów statusu (tak zwanych flag) w rejestrze TIMSK (Timer Interrupt Mask). Tak więc kontrola stanu timera sprowadza się do testowania przez CPU mikrokontrolera maksymalnie 3 bitów sygnalizujących stan timera. Bitami tymi są:

- **Timer Overflow (przepełnienie timera)**
  Ustawienie tego bitu informuje, że timer osiągnął wartość maksymalną i zostanie wyzerowany w następnym cyklu zegarowym. Jak wcześniej wspomniałem, AVR wyposażony jest w dwa timery 8-bitowe oraz jeden 16-bitowy. W praktyce oznacza to dwa timery mogące liczyć do wartości 0xFF oraz jeden liczący do 0xFFFF. Przepełnienie sygnalizowane jest przy pomocy bitu noszącego nazwę Timer Overflow Flag (TOVx) w rejestrze TIFR (Timer Interrupt Flag Register).
- **Compare Match (spełniony warunek porównania)**
  W przypadku, gdy nie jest konieczne monitorowanie stanu flagi przepełnienia, może być używane przerwanie typu COMPARE MATCH wywoływane, gdy wartość zapamiętana w rejestrze OCRx (Output Compare Register) zgadza się ze zliczoną przez timer. Wskazanie przez timer wartości identycznej z zapisaną w rejestrze OCRx powoduje ustawienie właściwego bitu OCFx (Output Compare Flag) w rejestrze TIFR. Timer może być również skonfigurowany w taki sposób, aby jednocześnie z ustawieniem flagi OCFx wartość rejestru liczącego timera była zerowana. Istnieje również możliwość wyboru takiego trybu pracy, dzięki któremu automatycznie, w momencie spełnienia warunku porównania, odpowiednim wyprowadzeniom mikrokontrolera może zostać przypisany stan niski, wysoki lub zanegowany. Funkcja ta jest bardzo użyteczna podczas budowy generatorów sygnału prostokątnego o różnej częstotliwości. Oferując szeroki zakres generowanych częstotliwości umożliwia na przykład budowę prostych przetworników cyfrowo – analogowych, jakkolwiek do tego zastosowania bardziej właściwym wydaje się wykorzystanie trybu generatora o modulowanej szerokości impulsu (PWM).

- **Input Capture (przechwycenie wartości)**
  Mikrokontrolery AVR posiadają wejście nazywane Input Capture (IC). Zmiana stanu na tym wejściu powoduje, że aktualna wartość timera jest odczytywana i zapamiętywana w rejestrze ICRx (Input Capture Register). Jednocześnie ustawiana jest flaga ICFx (Input Capture Flag) w rejestrze TIFR. Funkcja ta najczęściej wykorzystywana jest do pomiaru czasu trwania impulsu.

Każdy z wyżej wymienionych bitów może wywoływać odpowiedni wektor przerwania. Przerwaniami oraz ich obsługą zajmiemy się w dalszej części artykułu.

Kontrola stanu timera

Są trzy podstawowe metody kontrolowania zdarzeń generowanych przez timer a tym samym powodowania reakcji mikrokontrolera w zależności od stanu timera:
1.Kontrolowanie stanów bitów statusu (flag) w czasie pracy programu poprzez ich testowanie metodą odpytywania (z ang. pooling – odpytywanie) i podejmowanie akcji odpowiedniej dla danej ich kombinacji.
2.Odpowiednie ustawienie rejestru kontrolującego przerwania a następnie automatyczne przerywanie pracy programu głównego i wykonywanie programów obsług przerwań.
3.Sprzętowa i całkowicie automatyczna zmiana stanu odpowiedniego wyprowadzenia mikrokontrolera.
Kontrola statusu flag korzysta z faktu, że wewnętrzne układy mikrokontrolera ustawiają określone bity powodujące przejście do procedury obsługi przerwania o ile ta nie została zabroniona. Oczywiście warunkiem korzystania z tej metody jest wyłączenie obsługi przerwania, bo inaczej odpytywanie nie miałoby sensu. Kontrola stanu bitów flag, jakkolwiek chyba najłatwiejsza do wykonania, jest jednocześnie mało efektywną bo zajmuje czas mikrokontrolera. Należy również liczyć się z pewnym opóźnieniem przy podejmowaniu akcji, ponieważ CPU zanim zacznie kontrolować stan flag, może być zaangażowane w realizację zupełnie innej części kodu związanej z obsługą całkowicie innych funkcji mikrokontrolera. Poniższy fragment programu w języku asemblera ilustruje użycie tej metody wykorzystanej do kontroli Timera 0. Linie te powinny być umieszczone w pętli głównej wykonywanego programu a stan flag musi być kontrolowany tak często, jako tylko jest to możliwe.

```
loop:                           ;główna pętla programu
        .........
        in      r16,TIFR        ;załadowanie rejestru TIFR do r16
        sbrs    r16,TOV0        ;omiń następną instrukcję, jeśli bit 0 w r16 jest ustawiony
        rjmp    loop            ;wykonaj skok do początku pętli głównej programu jeśli bit przepełnienia
                                ;Timera 0 nie był ustawiony
event:
        .........               ;tu rozpoczyna się obsługa zdarzenia „przepełnienie Timera 0”
```

Najlepszą moim zdaniem metodą kontroli stanu timera jest wykorzystanie systemu przerwań. Jak wcześniej wspomniałem, określone zdarzenia związane ze stanem timera powodują ustawianie flag w rejestrze TIMSK. Powodem ustawienia flagi może być przepełnienie rejestru liczącego, spełnienie warunku porównania czy też zakończenie działania przez funkcję pomiaru czasu trwania impulsu związaną z wejściem ICP. Tyle gwoli przypomnienia. O ile wykonywanie funkcji obsług przerwań jest dozwolone, CPU mikrokontrolera przerywa wykonywanie bieżącego programu lub wychodzi ze stanu uśpienia i wykonuje skok pod ściśle określony adres związany z danym powodem przerwania. Jednocześnie zapamiętany zostaje stan licznika rozkazów tak, że możliwe jest jego odtworzenie w momencie powrotu do programu głównego.
Jest to metoda bardzo efektywna – oszczędza czas mikrokontrolera angażując CPU tylko wówczas, gdy to jest naprawdę potrzebne, chociaż nastręcza pewne trudności przy implementacji. Program główny jest bowiem przerywany w momencie, który trudno przewidzieć i to programista musi zadbać o to, aby przy wejściu do procedury obsługi przerwania i po jej opuszczeniu program nadal wykonywany był normalnie. Odpowiednie przerwania załączane są przez nastawy bitów w rejestrze TIMSK (Timer Interrupt Mask). Poniższy przykład w języku asemblera ilustruje w jaki sposób włączyć procedurę obsługi przerwania na skutek przepełnienia Timera 2.

```
        ldi     r16,1<<OCIE2
        out     TIMSK,r16       ;zezwolenie na przerwanie Output Compare Timera 2
        sei                     ;zezwolenie na przyjmowanie przerwań
```

Tryby pracy, w które wyposażono Timer 1 i Timer 2 umożliwiają również nastawy akcji wykonywanych w sposób sprzętowy, bez konieczności wykonywania żadnego podprogramu. Odpowiednie wyprowadzenie mikrokontrolera może zostać skonfigurowane w taki sposób, aby było ustawiane, zerowane bądź też negowane w momencie spełnienia warunku porównania. W stosunku do dwóch poprzednich rozwiązań ten tryb nie angażuje w żaden sposób CPU mikrokontrolera. Poniższy przykład ilustruje ten sposób

konfiguracji z wykorzystaniem Timera 2. Poziom logiczny wyprowadzenia OC2 jest negowany w momencie spełnienia warunku porównania (gdy licznik Timera 2 osiągnie wartość dziesiętną 32). Zawiera on też sposób ustawienia wartości porównywanej. Konfiguracja timera jest dokonywana przy pomocy ustawienia bitów COMx0 i COMx1 w rejestrze TCCRx – w przypadku użycia Timera 2 są to bity COM20 i COM21 w rejestrze TCCR2.

```
ldi     r16,(1<<COM20)|(1<<CS20)
out     TCCR2,r16       ;OC2 negowany po spełnieniu warunku compare/match
                        ;zegar = zegar systemowy
ldi     r16,32
out     OCR2,r16        ;ustawienie porównywanej wartości na 32
```

Należy jednak pamiętać o tym, że wybór trybu pracy timera nie wpływa na ustawienie kierunku linii portu właściwej dla OC2. Aby zezwolić na ustawianie wartości wyprowadzenia OC2, odpowiedni bit konfiguracji kierunku bitu portu musi być ustawiony w taki sposób aby wyprowadzenie to pracowało jako wyjściowe.

Opcje nastaw zegara.

Generator zegarowy AVR zawiera preskaler podłączony do multipleksera. Preskaler to dzielnik częstotliwości zegara. Został on zaimplementowany jako licznik z kilkoma wyjściami o różnych stopniach podziału. W przypadku AT90S8535 jest to 10-bitowy licznik używany do wytworzenia czterech (w przypadku Timera 2 sześciu) różnych częstotliwości taktujących timery, wynikających z podziału częstotliwości generatora zegarowego. Multiplekser używany jest do wyboru która z czterech (sześciu) częstotliwości używana jest jako podstawa czasu timera. Alternatywnie multiplekser może być użyty do ominięcia preskalera oraz konfiguracji zewnętrznego wyprowadzenia jako wejściowego dla timera. Timery 0 i 1 są timerami synchronicznymi i używają zegara systemowego CPU jako źródła sygnału zegarowego. Asynchroniczny Timer 2 wymaga własnego preskalera co czyni go niezależnym od zegara systemowego. Na rysunku 1 pokazano połączenia pomiędzy preskalerem i multiplekserem. Danych na temat konkretnej konfiguracji dla danego mikrokontrolera AVR należy szukać w jego karcie katalogowej. Tabela 1 zawiera listę możliwych nastaw preskalera. I tu również należy odwołać się do danych zawartych w konkretnej karcie katalogowej, gdzie prawdopodobnie będą one opisane dokładniej i powiązane z konkretnym modelem mikrokontrolera.

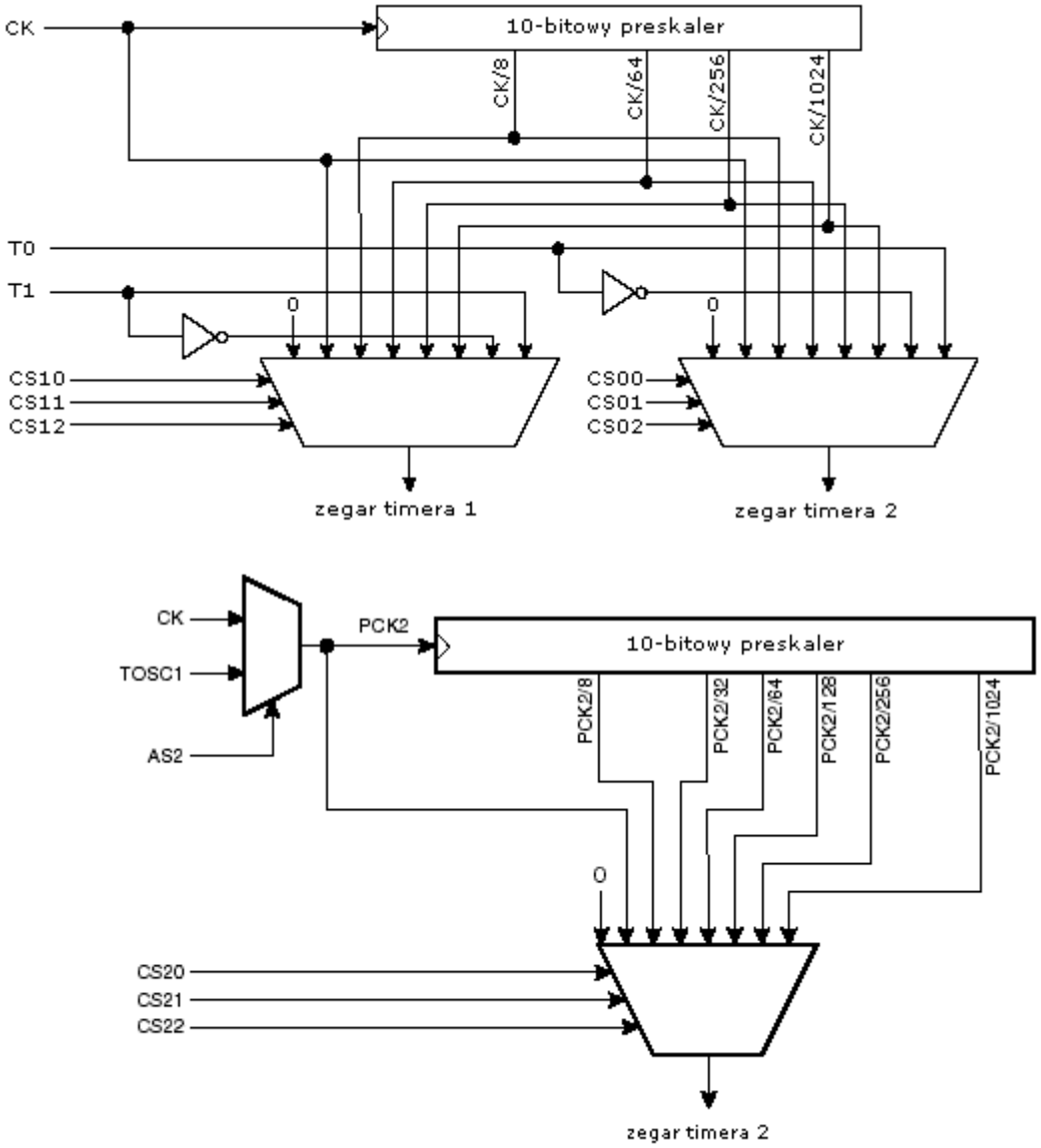
Taktowanie przez zegar systemowy.

Zegar systemowy używany jest jako wejściowy dla preskalera również wówczas, gdy częstotliwość taktowania CPU została wybrana jako jedna z otrzymywanych z preskalera. Timer pracuje więc synchronicznie z zegarem systemowym. Wszystkie trzy timery AT90S8535 oraz timery większości innych mikrokontrolerów AVR pracują w ten sposób. Nie są wymagane żadne dodatkowe układy zewnętrzne. Zaletą takiego rozwiązania jest fakt, że dzięki bardzo wysokiej częstotliwości zegara systemowego (o wiele wyższej niż tej, która taktuje CPU) operacje przeprowadzane przez mikrokontroler mogą być mierzone z o wiele większą dokładnością.
Częstotliwość przepełnienia timera jest dobrym wskaźnikiem rozmiaru ramki czasowej, którą jest w stanie pokryć timer. Wyrażenie 1 ukazuje powiązanie pomiędzy częstotliwością przepełnienia timera TOV_{CK}, maksymalną wartością, którą może być wpisana do timera MaxVal ,częstotliwością zegara systemowego f_{CK} i współczynnikiem podziału preskalera PVal.

$$TOV_{CK} = \frac{f_{CK}}{MaxVal} = \frac{(P_{CKx} / PVal)}{MaxVal} = \frac{P_{CKx}}{(PVal \bullet MaxVal)}$$

Dla przykładu jeśli CPU taktowane jest częstotliwością 3,69MHz i timer ma rozdzielczość 8 bitów (MaxVal = 256) wartość preskalera 64 spowoduje, że timer taktowany częstotliwością T_{CK} równą 3,69MHz / 64 wygeneruje ok. 225 sygnałów przepełnienia w czasie 1 sekundy.

$$TOV_{CK} = \frac{f_{CK}}{MaxVal} = \frac{(3.69\ \text{MHz} / 64)}{256} = \sim 225$$



Uwagi:

1. Preskaler pracuje nieprzerwanie – również podczas wprowadzania nastaw timerów. W przypadkach gdy wymagane jest bardzo dokładne odmierzanie czasu, należy samemu zadbać o to, aby timer został zatrzymany i preskaler zaczął podział od wartości 0. W mikrokontrolerach nie przeprowadzających zerowania preskalera może ono zostać przeprowadzone przez detekcję przepełnienia preskalera przez aplikację oraz inicjalizację rejestru TCNTx po tym zdarzeniu.
2. W nowszych mikrokontrolerach posiadających preskaler dzielony pomiędzy kilka timerów, przeprowadzenie sekwencji reset w taki sam sposób wpływa na wszystkie podłączone urządzenia, inicjując je i przeprowadzając odliczanie od wartości 0.

Rysunek 1. Poglądowy schemat połączeń pomiędzy preskalerem i multiplekserem dla Timerów 0 i 1 oraz 2.

TCCRx			Synchroniczny Timer 0 i Timer 1 $P_{CK0,1}$ = zegar syst.	Synchroniczny/Asynchroniczny Timer 2 P_{CK2} = zegar syst./zegar zewn.
Bit 2	Bit 1	Bit 0		
0	0	0	0 (Timer 0/1 zatrzymany)	0 (Timer 2 zatrzymany)
0	0	1	P_{CK} (zegar systemowy)	P_{CK2} (zegar systemowy lub asynchroniczny)
0	1	0	$P_{CK}/8$	$P_{CK2}/8$
0	1	1	$P_{CK}/64$	$P_{CK2}/32$
1	0	0	$P_{CK}/256$	$P_{CK2}/64$
1	0	1	$P_{CK}/1024$	$P_{CK2}/128$
1	1	0	P_{CK} opadające zbocze na wypr.Tx	$P_{CK2}/256$
1	1	1	P_{CK} narastające zbocze na wypr.Tx	$P_{CK2}/1024$

Tabela 1. Nastawy bitów preskalera

Wygenerowanie liczby 225 przepełnień timera w czasie 1 sekundy oznacza konieczność wysłania sygnału przepełnienia co 4,4ms. Maksymalna wartość preskalera powoduje przepełnienie co 71ms, minimalna co 69 µs. Wymagania aplikacji determinują częstotliwość przepełnień timera. Bazując na nich oraz na znanej częstotliwości taktowania timera łącznie z jego rozdzielczością, nastawa preskalera może być wyliczona

$$PVal = \frac{P_{CKx}}{(TOV \bullet MaxVal)}$$

przy pomocy następującego wyrażenia:

Implementacja w języku asemblera może wyglądać tak, jak poniższy przykład programu. Ustawia on wartość preskalera przy pomocy TCCR0 na podział częstotliwości zegara przez 1024.

```
ldi     r16,(1<<CS02)|(1<<CS00)
out     TCCR0,r16       ;zegar taktujący timer = zegar systemowy / 1024
```

Taktowanie przez zegar asynchroniczny.

W odróżnieniu od innych timerów, które nie obsługują tej opcji Timer 2 AT90S8535 może być taktowany przez zewnętrzne źródło sygnału. W tym celu kwarc lub rezonator ceramiczny podłącza się do wyprowadzeń TOSC1 i TOSC2. Oscylator jest zoptymalizowany dla kwarcu tzw.zegarkowego o częstotliwości 32768Hz. Ta częstotliwość jest bardzo dobra zwłaszcza dla implementacji zegara czasu rzeczywistego. Główną zaletą tego rozwiązania jest uniezależnienie od zegara systemowego. Umożliwia ono również CPU pracę z dużą częstotliwością przetwarzania, niekoniecznie dobraną pod kątem pomiaru czasu, podczas gdy timer pracuje z częstotliwością dla nich optymalną. Dodatkowo tryb oszczędzania energii ma opcję umożliwiającą wprowadzenie części układów mikrokontrolera w tryb uśpienia podczas gdy asynchroniczny timer ciągle pracuje. Tutaj jedna uwaga: częstotliwość zewnętrznego oscylatora jest różna dla różnych typów mikrokontrolerów. Jej dolna granica zawiera się w przedziale od 0Hz do 256kHz a górną wyznacza częstotliwość zegara systemowego: powinna być ona mniejsza lub równa niż F_{CK} / 4. Praca z timerem taktowanym asynchronicznie wymaga pewnych dodatkowych rozważań. Ponieważ Timer 2 taktowany jest asynchronicznie w stosunku do zegara systemowego, zdarzenia generowane przez Timer muszą być synchronizowane przez CPU. Z tej cechy wynika wymaganie aby częstotliwość taktowania timera była co najmniej czterokrotnie mniejsza niż częstotliwość zegara systemowego. Z drugiej strony możliwe są konflikty pomiędzy synchroniczymi i asynchronicznymi żądaniami obsługi (np.przerwania). Jak CPU radzi sobie z takimi sytuacjami? Obsługa zdarzeń jest przeprowadzana przez rejestry tymczasowe. Bity statusu sygnalizują kiedy przeprowadzane jest uzupełnianie zawartości rejestrów. Dokładny opis rejestrów ASSR (Asynchronous Status Register) można znaleźć w karcie katalogowej. Częstotliwość z jaką ustawiany jest bit przepełnienia można obliczyć identycznie jak w poprzednim przypadku z tym, że do równania musi zostać wstawiona częstotliwość zewnętrznego źródła sygnału. Nastawy preskalera Timera 2 zostały podane w tabeli 1, częstotliowość taktowania preskalera Timera 2 jest funkcją bitu AS2 w rejestrze ASSR. Jeśli ten bit jest wyzerowany, timer pracuje w trybie synchronicznym z częstotliwością zegara systemowego jako wejściową. Jeśli ten bit jest ustawiony, asynchroniczny sygnał zegarowy z wyprowadzeń TOSC1 i TOSC2 jest używany jako sygnał wejściowy preskalera. Fragment programu w języku asemblera ustawia preskaler Timera 2 na maksymalną wartość podziału (1024)

```
ldi r16, (1<<CS22)|(1<<CS21)|(1<<CS20)
out TCCR2,r16  ;zegar timera 2 = zegar systemowy / 1024
```

Taktowanie przy pomocy zewnętrznego generatora

Timer 0 i Timer 1 mogą być taktowane z zewnętrznego generatora sygnału zegarowego. Tryb ten zapewnia obsługę szeregu różnych źródeł jako generatorów sygnału zegarowego. Jest to taktowanie synchroniczne co oznacza, że CPU wykrywa stan wyprowadzenia i jeśli wykryta została zmiana zewnętrznego sygnału, to przeprowadza odpowiednią akcję synchronicznie z zegarem systemowym. Każde opadające zbocze zegara systemowego powoduje pobranie próbki zewnętrznego sygnału. CPU potrzebuje co najmniej 2 cykli aby wykryć zmianę zewnętrznego sygnału. Ogranicza to maksymalną częstotliwość sygnału zewnętrznego do F_{CK} / 2. W zależności o konfiguracji, opadające lub narastające zbocze sygnału na wyprowadzeniu T0 / T1 może oznaczać zmianę sygnału zegarowego. Wybór zbocza dokonywany jest przy pomocy bitów CSO0..1 znajdujących się w rejestrze TCCRx (patrz opis w tabeli 1). Poniższy fragment kodu w języku asemblera pokazuje w jaki sposób ustawić Timer 0 aby pracował z zewnętrznym źródłem sygnału reagując na każde jego narastające zbocze

```
ldi     r16,(1<<CS02)|(1<<CS01)|(1<<CS00)
out     TCCR0,r16       ;zegar timera = zewnętrzne wyprowadzenie T0, narastające
                        ;zbocze sygnału
```

Stosując ten tryb pracy należy upewnić się, że nastawy kierunku bitu dokonane w rejestrze DDRB (Data Direction Register, Port B) są właściwe. Wybór trybu pracy timera nie powoduje zmian nastaw bitów portu. Po sygnale reset wyprowadzenia portu B są ustawiane domyślnie jako wejścia sygnałów.

Jak zatrzymać Timer?

Zatrzymanie timera jest bardzo proste: zapis wartości 0 do preskalera (rejestr TCCRx) zatrzymuje odpowiedni timer. Należy jednak pamiętać, że preskaler w dalszym ciągu pracuje. Kod w języku asemblera zatrzymujący pracę Timera 0 może wyglądać jak niżej:

```
clr     r16
out     TCCR0,r16       ;zapis wartości 0 do TCCR0 zatrzymuje Timer 0
```

Jeśli zależy nam na zachowaniu wartości resjestru TCCR0 w związku z innymi nastawami, zapis nastaw bitów CSO0..1 kosztuje dodatkowe linie programu i może wyglądać jak niżej:

```
in      r16,TCCR0       ;odczyt aktualnej wartości rejestru TCCR0
andi    r16,~((1<<CS02)|(1<<CS01)|(1<<CS00))
out     TCCR0,r16       ;Zapis 0 do bitów CS02, CS01, and CS00 w TCCR0
                        ;zatrzymuje Timer 0
```

Nastawy trybów pracy timerów.

Ta część tekstu koncentruje się na sposobach wykonywania nastaw trybów pracy timerów. Należy jednak pamiętać, że podany niżej przykłady dotyczą mikrokontrolera AT90S8535 i dla innych mikrokontrolerów mogą być konieczne zmiany. Jak wcześniej wspomniałem, moim zdaniem używanie przerwań to jedna z najbardziej efektywnych metod obsługi zdarzeń generowanych przez timery: większość z przykładów programowania będzie zawierać obsługę przerwań.
Niezależnie od różnych rozszerzeń oferowanych przez trzy timery, mają one pewne cechy wspólne. Każdy z timerów musi być uruchomiony przez wybór źródła sygnału zegarowego i jeśli używane są przerwania, to również muszą zostać dokonane związane z nimi nastawy. Jedną z zasad obowiązujących przy tworzeniu procedur obsługi przerwań jest ta, że jeśli te same rejestry używane są w programie głównym co i w procedurze obsługi przerwania, to muszą one zostać podczas obsługi przerwania zapamiętane a następnie odtworzone przy powrocie do programu głównego. Jeśli nie wszystkie 32 rejestry (AT90S8535) muszą być używane, dobrze jest użyć odrębnych dla programu głównego i dla procedury obsługi przerwania. Bardzo ważnym jest aby pamiętać, że rejestr statusy SREG (Status Register) nie jest automatycznie zapamiętywany przez procedurę obsługi przerwania i należy również zatroszczyć się o jego zawartość. Tak jest w przypadku programów napisanych w języku asemblera. W tych napisanych w językach wysokiego poziomu, takich jak Bascom czy C, kompilator automatycznie zapamiętuje zawartość SREG przy wejściu do procedury obsługi przerwania i odtwarza ją przy powrocie. O resztę rejestrów należy zatroszczyć się „ręcznie". W przypadku programów napisanych w języku asemblera można posiłkować się instrukcjami PUSH i POP jednak należy pamiętać o tym, że niektóre z modeli mikrokontrolerów AVR nie posiadają tych rozkazów na swojej liście wykonywanych poleceń.

8-bitowy Timer 0.

Timer 0 jest timerem synchronicznym, co oznacza że jest taktowany przez zegar systemowy, zegar systemowy o częstotliwości zmniejszonej przez preskaler lub przez sygnał zewnętrzny ale zawsze synchronicznie z zegarem systemowym używanym przez CPU.

Przykład – procedura obsługi przerwania na skutek przepełnienia Timera 0
Przykład pokazuje w jaki sposób Timer 0 może być używany do wywoływania procedury obsługi przerwań. Każde wywołanie zmienia stan portów wyjściowych portu B. Jeśli do wyprowadzeń portu B zostaną podłączone diody LED, to będą one migotać z częstotliwością, którą można wyznaczyć przy pomocy wcześniej poznanej formuły.

```
;podprogram inicjujący tryb pracy mikrokontrolera
init_Ex1:
        ldi     r16,(1<<CS02)|(1<<CS00)
        out     TCCR0,r16       ;zegar Timera 0 = zegar systemowy / 1024
        ldi     r16,1<<TOV0
        out     TIFR,r16        ;kasowanie bitu TOV0 / kasowanie bieżącego przerwania
        ldi     r16,1<<TOIE0
        out     TIMSK,r16       ;załączenie Timera 0, zezwolenie na generowanie przerwań
        ser     r16
        out DDRB,r16   ;ustawienie portu B jako wyjściowego
        ret
```

W następnym kroku zaimplementujemy procedurę obsługi przerwania. Będzie ona wywoływana po każdym przepełnieniu Timera 0. Jej przeznaczeniem jest zmiana stanu bitów portu B.

```
;procedura obsługi przerwania Timera 0
ISR_TOV0:
        push    r16
        in      r16,SREG        ;zapamiętanie rejestru statusu oraz r16
        push    r16
        in      r16,PORTB       ;czytaj stan portu B
        com     r16             ;zaneguj bity rejestru r16
        out     PORTB,r16       ;zapisz r16 do portu B
        pop     r16
        out     SREG,r16        ;odtworzenie rejestru statusu i r16
        pop     r16
        reti
```

16-bitowy Timer 1.

Podobnie jak Timer 0, Timer 1 pracuje synchronicznie. Dla upewnienia się, że wykonywany jest jednoczesny zapis i odczyt 16-bitowego rejestru timera, do przeprowadzenia tych operacji używany jest rejestr tymczasowy Temp. Czyni to niezbędnym dostęp do tego rejestru w specyficzny sposób. Metoda jest opisana dokładnie w nocie aplikacyjnej firmy Atmel „AVR072: Accessing 16-bit I/O Registers”. Bardzo dużym skrótem rozważań na ten temat jest właściwy dla AVR sposób dostępu do rejestrów 16-bitowych przedstawiony w tabeli 2. Dociekliwych zachęcam do lektury, tu zajmiemy się wyłącznie przykładami programów użytkowych.

Rodzaj przeprowadzanej operacji	W pierwszej kolejności	W drugiej kolejności
Odczyt	Odczyt młodszego bajtu (LSB)	Odczyt starszego bajtu (MSB)
Zapis	Zapis starszego bajtu (MSB)	Zapis młodszego bajtu (LSB)

Przykłady użycia:

- odczyt:
  ```
  in      r16,TCNT1L
  in      r17,TCNT1H
  ```
- zapis:
  ```
  out TCNT1H,r17
  out TCNT1L,r16
  ```

Tabela 2. Właściwy sposób dostępu do rejestrów 16-bitowych

Obsługa przerwania Timera 1 pochodzącego od wejścia ICP (Capture Input).

Przykład ten pokaże prostą metodę użycia zdarzenia generowanego na skutek zmiany stanu wejścia ICP oraz obsługi jego przerwania. Wyprowadzenie bitu 6 portu D używane jest jako wejście dla funkcji pomiaru sygnału zewnętrznego i nosi nazwę ICP. Funkcja pomiaru związana z tym wejściem funkcjonuje w taki sposób, że Timer może zmierzyć czas pomiędzy dwoma następującymi po sobie opadającymi lub narastającymi zboczami sygnału podanego na wejście ICP. W prezentowanym przykładzie 8 bardziej znaczących bitów Timera 1 zostanie zapisanych do portu B. Jeśli tak, jak w przykładzie powyżej, do wyprowadzeń portu B podłączymy diody LED, uzyskamy prostą funkcję wskazującą czas trwania impulsu. Bit 6 portu D (wejście ICP) może być podłączony do generatora fali prostokątnej lub po prostu do przycisku. W prezentowanym przykładzie, dla rezonatora kwarcowego około 4MHz, maksymalny mierzony czas zbliżony jest do 1 sekundy.

```
;podprogram inicjalizacji trybu pracy mikrokontrolera
init_Ex2:
        ldi     r16,(1<<CS11)|(1<<CS10)
        out     TCCR1B,r16      ;zegar Timera 1 = zegar systemowy / 64
        ldi     r16,1<<ICF1
        out     TIFR,r16        ;kasowanie bitu ICF1/kasownie obsługi trwającego przerwania
        ldi     r16,1<<TICIE1
        out     TIMSK,r16       ;zezwolenie na obsługę przerwań od ICP
        ser     r16             ;ustawienie bitów w r16
        out     DDRB,r16        ;załączenie trybu pracy portu B jako wyjściowego
        cbi     DDRD,PD6        ;załączenie PD6/ICP jako wejście
        ret
```

Następnie wykonamy procedurę obsługi przerwania. Jej zadaniem jest po pierwsze wyprowadzenie starszego bajtu licznika Timera 1 przez port PB oraz przygotowanie timera do następnego pomiaru.

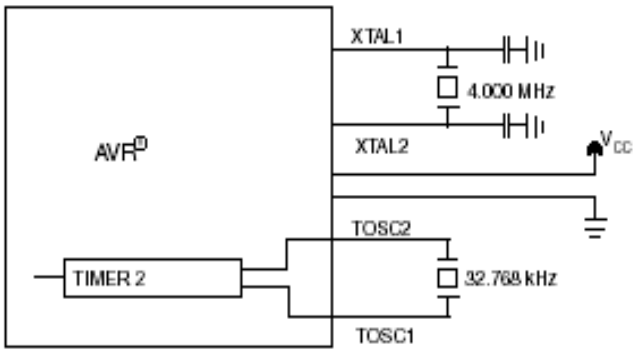
```
TIM1_CAPT:
        push    r16
        in      r16,SREG        ;zapamiętanie wartości rejestru statusu i r16
        push    r16
        in      r16,ICR1L       ;odczyt młodszego bajtu ICR
                                ;tu można zapamiętać młodszy bajt w zmiennej
        in      r16,ICR1H       ;odczyt starszego bajtu ICR
        com     r16             ;negowanie odczytanych bitów ze względu na diody LED
                                ;jeśli LEDy nie są podłączone nie jest konieczne
        out     PORTB,r16       ;zapis ICR1H to portu B
        clr     r16
        out     TCNT1H,r16      ;zapis rejestru Temp
        out     TCNT1L,r16      ;a teraz jednoczesny zapis 16-bitów rejestru TCNT1 równoważne
                                ;z zerowaniem TCNT1
        pop     r16
        out     SREG,r16        ;odtworzenie rejestru statusu i r16
        pop     r16
        reti
```

Inwersja wprowadzona przy pomocy rozkazu *com r16* jest konieczne ze względu na sposób podłączenia diod LED: anodą do pozytywnego napięcia zasilania, katodą do wyprowadzenia portu. W efekcie dioda świeci się, gdy wyprowadzenie portu znajduje się w stanie niskim. Jest to stan odwrotny niż spodziewany intuicyjnie – człowiek oczekuje, że „jedynka" logiczna oznacza zaświeconą diodę. Powyższa implementacja ma jedną poważną wadę: nie jest wskazywane przekroczenie zakresu pomiarowego.

Asynchroniczny Timer 2. Wywołanie przerwania na skutek porównania zawartości licznika Timera 2 z wartością zadaną.

Timer 2 może pracować w trybie synchronicznym tak, jak Timer 0 i Timer 1. Dodatkowo został wyposażony w tryb asynchroniczny opisywany już wcześniej. Przykład ten pozkauje w jaki sposób używać funkcji porównywania wartości timera z wartością zadaną. Timer zostanie skonfigurowany w taki sposób, że warunek porównania będzie spełniony co sekundę. Ta właściwość może być wykorzystana np.do budowy zegara. W prezentowanym przykładzie wykorzystamy jednak, podobnie jak poprzednio, diodę

LED podłączoną do portu B, która będzie migotać z częstotliwością 0,5Hz. Ten przykład programu wymaga podłączenia rezonatora zegarkowego 32,768kHz do wyprowadzeń TOSC1 (PC6) i TOSC2 (PC7).



Wartość nastaw wpisywana do rejestrów może być wyliczona za pomocą podanego wcześniej równania. Zamiast wartości MaxVal wpisywanej do Timera 2 musi zostać użyta wartość OCR2. Częstotliwość zegara preskalera (P_{CK}) w tym przypadku ma wartość podłączonego z zewnątrz rezonatora kwarcowego, bit TOV musi być ustawiany z częstotliwością 1Hz. Korzystając z powyższych danych wyznaczymy wartość wpisywaną do rejestru „capture / compare".

$$1 = TOV_{CK} = \frac{f_{OSCCK}}{\mathrm{PVal} \bullet \mathrm{OCR2}} = \frac{32.768\ \mathrm{kHz}}{\mathrm{PVal} \bullet \mathrm{OCR2}}$$

Wybrana wartość preskalera 1024 oraz wartość 32 wpisywana do rejestru OCR2 umożliwia uzyskanie częstotliwości 1Hz. Teraz program, tradycyjnie zaczniemy od podprogramu nastaw timerów:

```
init_Ex3:
        ldi     r16,1<<AS2
        out     ASSR,r16            ;zezwolenie trybu asynchronicznego Timera 2
                                    ;kasowanie timera po spełnieniu warunku
                                    ;zegar timera = zegar systemowy / 1024
        ldi     r16,(1<<CTC2)|(1<<CS22)|(1<<CS21)|(1<<CS20)
        out     TCCR2,r16
        ldi     r16,1<<OCF2
        out     TIFR,r16            ;kasowanie flagi OCF2 trwającego przerwania
        ldi     r16,1<<OCIE2
        out     TIMSK,r16           ;zezwolenie na wywołanie przerwania po spełnieniu
        ldi     r16,32              ;warunku porównania
        out     OCR2,r16            ;ustawienie wartości porównywanej na 32
        ser     r16
        out     DDRB,r16            ;ustawienie portu D jako wyjściowego
loop:
        sbic    ASSR, OCR2UB ;oczekiwanie na ustalenie wartości rejestrów
        rjmp    loop
        ret
```

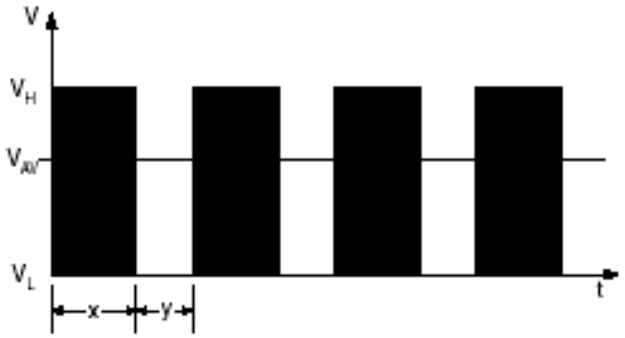
W następnym kroku podprogram obsługi przerwania. Jego zadaniem jest negowanie stanu portu B po każdym wywołaniu.

```
ISR_OCIE2:
        push    r16
        in      r16,SREG            ;przechowanie wartości r16 i rejestru statusu
        push    r16
        in      r16,PORTB           ;odczyt stanu portu B
        com     r16                 ;negacja bitów rejestru r16
        out     PORTB,r16           ;zapis wartości r16 do portu B
        pop     r16
```

```
out     SREG,r16        ;odtworzenie stanu r16 i rejestru statusu
pop     r16
reti
```

Podstawy PWM.

PWM jest skrótem od pochodzącej z języka angielskiego nazwy Pulse Width Modulation (modulacja szerokości impulsu). Jest to specjalny tryb pracy, w którym mogą pracować Timer 1 i Timer 2. W tym trybie timer pracuje jako licznik w górę lub w dół. Oznacza to, że timer liczy w górę od 0 do wartości maksymalnej a następnie w dół, z powrotem do wartości 0. Cechą generatora PWM jest to, że wypełnienie impulsów może być zmieniane. Jeśli PWM jest skonfigurowane w taki sposób, że zmienia się stan wyprowadzenia OCx (Output Compare), wówczas sygnał oglądany przy pomocy oscyloskopu na tym wyprowadzeniu, może wyglądać jak na rysunku 2.



V_H napięcie wyjściowe stanu wysokiego
V_L napięcie wyjściowe stany niskiego
V_{AV} uśrednione napięcie wyjściowe
x czas trwania stanu wysokiego
y czas trwania stanu niskiego

Rysunek 2. Sygnał wyjściowy generatora PWM.

Filtr dolnoprzepustowy dołączony do wyjścia generatora PWM o parametrach dobranych do właściwości generatora umożliwi otrzymanie napięcia stałego na wyjściu, zmieniającego się w zależności od wypełnienia doprowadzonego przebiegu, zamiast fali prostokątnej. Równanie pokazuje w jaki sposób można wyliczyć jego wartość:

$$V_{AV} = \frac{(V_H \bullet x + V_L \bullet y)}{(x + y)}$$

Jeśli w miejsce *x* i *y* podstawimy odpowiednie wartości wyznaczające czas trwania impulsów otrzymywanych przy pomocy naszego generatora PWM

$$x = OCRx \bullet 2$$
$$y = (MaxVal - OCRx) \bullet 2$$

otrzymamy następującą zależność umożliwiającą wyznaczenie wartości napięcia wyjściowego:

$$V_{AV} = \frac{(V_H \bullet OCRx + V_L \bullet (MaxVal - OCRx))}{MaxVal}$$

Jak wynika z powyższej lektury, możliwa jest budowa prostych przetworników cyfrowo – analogowych tylko z wykorzystaniem generatora PWM i prostego układu filtru.

Timer 2 jako 8-bitowy generator PWM

Ten przykład pokazuje w jaki sposób należy skonfigurować Timer 2 aby mógł on pracować jako generator PWM o rozdzielczości 8 bitów. Nasz generator wytwarzał będzie falę prostokątną o napięciu niskim zbliżonym do GND i wysokim zbliżonym do VCC. Do obserwacji wytworzonej fali ponownie użyjemy diody LED podłączonej do wyprowadzenia OC2 (PD7). W tym przykładzie rolę filtra „uśredniającego" wskazania diody będzie spełniało nasze oko, toteż efekt pracy generatora będzie można zaobserwować jako zmianę jasności świecenia diody. Wypełnienie sygnału wyjściowego PWM można zmieniać się od 1/8 do 7/8 (wartość OCR2 = 0xE0). W tym przykładzie wyprowadzany sygnał będzie zanegowany ze względu na sposób podłączenia diody LED.

```
init_Ex4:
                        ; 8 bit PWM (Fck/510)
        ldi     r16,(1<<PWM2)|(1<<COM21)|(1<<CS20)
        out     TCCR2,r16
        ldi     r16,0xE0
        out     OCR2,r16        ;ustawienie wartości porównywanej, od której zależy wypełnienie
                                ;impulsów wyjściowych
        ldi     r16,0x8F
        out     DDRD,r16        ;ustawienie trybu PD7/OC2 jako portu wyjściowego
        ret
```

Jacek Bogusz
jacek.bogusz@easy-soft.tsnet.pl