POLITECHNIKA CZĘSTOCHOWSKA, **WYDZIAŁ INŻYNIERII MECHANICZNEJ I INFORMATYKI**
KATEDRA INŻYNIERII KOMPUTEROWEJ

Laboratorium Systemów Wbudowanych

Ćwiczenie nr 3:

TEMAT: Przetwornik ADC w mikrokontrolerze AT91SAM7X256

1. Wprowadzenie

Przetwarzanie ADC odbywa się wg metody SAR (Successive Approximation Register) z dokładnością 8 lub 10 bitową. Zakres przetwarzania ustala wejście ADVREF. W przypadku płytki ewaluacyjnej wejście to jest podłączone do napięcia zasilającego 3,3 V, więc uzyskany zakres przetwarzania to: 0-3,3 V. Wynik przetwarzania zapisywany jest w dwóch rejestrach: wspólnym dla wszystkich kanałów (ADC_LDCR) i specyficznym dla wybranego kanału (ADC_CDR0-7).
Uruchomienie przetwarzania można uzyskać:

- programowo
- zewnętrznie (zbocze narastające na nóżce ADTRG)
- wewnętrznie przy pomocy TIMER-a

Przetwornik umożliwia przejście do trybu uśpienia (Sleep Mode) i wykorzystanie przetwarzania sekwencyjnego, co redukuje pobór mocy lub zaangażowanie procesora.
Ponadto, ADC umożliwia ustawienie czasu wybudzenia (Startup Time) i stałej czasowej dla układu próbkująco-pamiętającego (Sample & Hold Time).
W mikrokontrolerach SAM7X wejścia ADO-4 przetwornika multipleksowane są wyprowadzeniami kontrolera PIOB: PB27-PB30. Żeby użyć przetwornika ADC na tych wyprowadzeniach, kontroler PIO dla nich powinien być uaktywniony (za pomocą PIO_PER, PIO Enable Register), a wyprowadzenia powinny być skonfigurowane jako wysokoimpedancyjne wejścia. Bezwzględnie do pomiaru należy wyłączyć tryb wyjściowy (za pomocą PIO_ODR, Output Disable Register), trzeba także wyłączyć wewnętrzne podciągnięcie na tych wyprowadzeniach (za pomocą rejestrów PIO_PPUDR, Pullup Disable Register).
Wyprowadzenia AD4, AD5, AD6 i AD7 mikrokontrolera SAM7X są wyprowadzone osobno, bez przeszkadzającego kontrolera PIO. W ich przypadku nie jest wymagana żadna dodatkowa konfiguracja kontrolera PIO, aby móc bez przeszkód mierzyć napięcia na nich.

2. Konfiguracja przetwornika

Zestaw rejestrów związanych z przetwornikiem ADC przedstawia rys. 1. Konfigurację przetwornika ADC wykonujemy poprzez zapis do pól bitowych rejestru ADC_MR.
Znaczenie pól w tym rejestrze jest następujące:
TRGEN – wybór pomiędzy sprzętowym a programowym uruchomieniem przetwornika (0- tylko programowe)
TRGSEL – wybór rodzaju sprzętowego uruchomieniem przetwornika:

0 0 0 – kanałem 0 TIMER-a
0 0 1 – kanałem 1 TIMER-a
0 1 0 – kanałem 2 TIMER-a
1 1 0 – zewnętrznym sygnałem

Offset	Register	Name	Access	Reset
0x00	Control Register	ADC_CR	Write-only	–
0x04	Mode Register	ADC_MR	Read-write	0x00000000
0x08	Reserved	–	–	–
0x0C	Reserved	–	–	–
0x10	Channel Enable Register	ADC_CHER	Write-only	–
0x14	Channel Disable Register	ADC_CHDR	Write-only	–
0x18	Channel Status Register	ADC_CHSR	Read-only	0x00000000
0x1C	Status Register	ADC_SR	Read-only	0x000C0000
0x20	Last Converted Data Register	ADC_LCDR	Read-only	0x00000000
0x24	Interrupt Enable Register	ADC_IER	Write-only	–
0x28	Interrupt Disable Register	ADC_IDR	Write-only	–
0x2C	Interrupt Mask Register	ADC_IMR	Read-only	0x00000000
0x30	Channel Data Register 0	ADC_CDR0	Read-only	0x00000000
0x34	Channel Data Register 1	ADC_CDR1	Read-only	0x00000000
...	...	...	...	...
0x4C	Channel Data Register 7	ADC_CDR7	Read-only	0x00000000
0x50 - 0xFC	Reserved	–	–	–

Rys. 1. Rejestry przetwornika ADC

31	30	29	28	27	26	25	24
–	–	–	–	SHTIM			
23	22	21	20	19	18	17	16
–	STARTUP						
15	14	13	12	11	10	9	8
PRESCAL							
7	6	5	4	3	2	1	0
–	–	SLEEP	LOWRES	TRGSEL			TRGEN

Rys. 2. Zawartość rejestru ADC_MR

LOWRES – wybór dokładności przetwarzania (0 – 10 bitowa, 1 – 8 bitowa)
SLEEP – tryb uśpienia (0 – normalny)
PRESCAL – częstotliwość taktowania ADC (ADCClock = MCK / ((PRESCAL+1) * 2))
STARTUP – czas wybudzania ADC (Startup Time = (STARTUP+1) * 8 / ADCClock)
SHTIM – stała czasowa dla S&H (Sample & Hold Time = SHTIM+1/ADCClock)

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	–	–	START	SWRST

Rys. 3. Zawartość rejestru ADC_CR

Znaczenie pól:
START – 1 uruchamia konwersję
SWRST – 1 resetuje ADC

Podstawowe wykorzystanie przetwornika wymaga znajomości tylko niektórych poniżej wymienionych bitów z pozostałych rejestrów sterujących.
CH0-7 (ADC_CHER) – wybór kanału (ów) do przetwarzania
EOC0-7 (ADC_SR) – 1 oznacza, że kanał jest włączony do przetwarzania i konwersja jest zakończona
DRDY (ADC_SR) – 1 oznacza, że co najmniej jedna konwersja została zakończona, a wynik jest dostępny w ADC_LCDR
Poza tym wynik konwersji jest dostępny w rejestrze, którego numer odpowiada wybranemu kanałowi (ADC_CDR0-7), a wynik ostatniej konwersji dodatkowo dostępny jest w ADC_LCDR.

3. Przykładowy program

3.1 Najprostszy sposób odczytu ADC

W pliku main.c przykładowego projektu (P3) w pętli głównej programu do zmiennej adcResult pobierana jest wartość zmierzonego napięcia w jednostkach ADC. Odczyt ADC odbywa się za pomocą funkcji adcGet przyjmującej jako argument numer kanału, na którym chcemy zmierzyć napięcie. Następnie zmierzone napięcie jest wyświetlane na wyświetlaczu LCD. Odświeżanie wyświetlanej wartości napięcia odbywa się kilka razy na sekundę.
Funkcja adcGet jako argument pobiera numer kanału przetwornika ADC (channel), na którym przeprowadzona będzie konwersja, a zwraca odczytaną z tego kanału wartość. Jej deklaracja w programie przykładowym jest następująca:

```
uintl6_t adcGet(uint8_t channel);
```

Przed użyciem modułu ADC nie trzeba włączać odpowiadającego mu sygnału taktującego. Zegar ten włącza się automatycznie przy rozpoczęciu konwersji. Gdy używa się kanałów 4-7 przetwornika, nie ma konieczności inicjalizacji kontrolera PIO tak, aby nie przeszkadzał przy pomiarze napięcia. Z powyższych względów, funkcja inicjalizująca ADC staje się niepotrzebna, a minimalną inicjalizację przeprowadzimy na początku funkcji adcGet.
Zaczniemy od wstępnego skonfigurowania ADC za pomocą rejestru ADC_MR.

```
//maski, które wpiszemy do niektórych pól bitowych ADC_MR
const uint32_t SHTIM_MASK = 0x04 << 24; //sample and hold
const uint32_t STARTUP_MASK = 0x0C<< 16; //czas startup
const uint32_t PRESCAL_MASK = 0x4 << 8; //konfig. prescalera

//konfiguracja rejestru ADC_MR:
AT91C_BASE_ADC→ADC_MR =
SHTIM_MASK | STARTUP_MASK | PRESCAL_MASK; //przygotowane maski
```

Powyższy fragment kodu wpisze do rejestru ADC_MR (Mode Register) trzy pola bitowe, ustawiając tym samym podstawowe zależności czasowe potrzebne do działania ADC. Pozostałe pola bitowe rejestru będą wyzerowane (oznacza to ich typowe wartości). Widok wszystkich pól rejestru ADC_MR znajduje się na rysunku 2.
Poniżej opisano, co zostało skonfigurowane.

- Do pola bitowego PRESCAL wpisaliśmy wartość 4. Oznacza to, że główny sygnał zegarowy MCK taktujący mikrokontroler (około 48 MHz) zostanie podzielony przez 10. Zatem częstotliwość taktowania przetwornika ADC wyniesie około 4,8 MHz.

- Pole bitowe STARTUP wypełnione zostanie wartością 12 (0x0C). Oznacza to czas wybudzania przetwornika ADC (przy takim ustawieniu PRESCAL jak wyżej) wynoszący około 21,6 µs.
- Czas działania układu sample and hold zdefiniowany jest w polu bitowym SHTIM. Czas ten jest związany z rezystancją wyjściową układu podającego napięcie na wejście przetwornika ADC. Tutaj ustawiliśmy czas sample and hold odpowiedni dla rezystancji rzędu pojedynczych kiloomów lub mniejszej.

Powyższa konfiguracja timingu modułu ADC będzie odpowiednia przy:

- typowej częstotliwości taktowania mikrokontrolera (około 48 MHz),
- wspomnianej rezystancji wyjściowej układu podającego napięcie na wejście ADC wynoszącej mniej niż 10 kΩ,
- maksymalnej możliwej do uzyskania rozdzielczości ADC: 10 bitów.

Po skonfigurowaniu timingu, wybieramy kanał, na którym chcemy zmierzyć napięcie. Wyboru kanału dokonujemy poprzez wpis 1 do bitu o numerze odpowiadającym wybranemu kanałowi w rejestrze ADC_CHER (Channel Enable Register).

```
//wartość channel przekazaliśmy w parametrze
AT91C_BASE_ADC→ADC_CHER = l<<channel;
```

Po tej konfiguracji , moduł ADC jest gotowy do rozpoczęcia pomiaru napięcia na wybranym kanale. Rozpoczęcie przetwarzania sygnału analogowego odbywa się poprzez wpisanie 1 do bitu ADC_START w rejestrze ADC_CR (Control Register).

```
AT91C_BASE_ADC→ADC_CR = AT91C_ADC_START;
```

Konwersja sygnału analogowego na cyfrowy nie odbywa się od razu, lecz zajmuje pewien czas. Najłatwiej rozpoznać, że konwersja zakończyła się, sprawdzając, czy ustawiony został bit DRDY (Data Ready) w rejestrze statusowym ADC_SR (Status Register).

```
while( ! (AT91C_BASE_ADC→ADC_SR & AT91C_ADC_DRDY) );
```

Po zakończonej konwersji można odczytać jej wynik z rejestru ADC_LCDR (Last Data Converted). W funkcji adcGet kopiujemy ten wynik do zmiennej retval, a wartość tej zmiennej zostanie przekazana jako efekt działania funkcji .

```
uintl6_t retval = AT91C_BASE_ADC→ADC_LCDR;
```

Interpretacja wyniku uzyskanego z przetwornika ADC wbudowanego w mikrokontroler nie jest skomplikowana. Funkcja adcGet zwróci wartość typu uintl6_t. 10 najmłodszych bitów tej wartości będzie stanowiło liczbę reprezentującą zmierzone napięcie. Jeśli napięcie podane na mierzony kanał ADC będzie wynosiło 0 V, funkcja zwróci wartość 0. Gdy natomiast na wejście przetwornika podamy napięcie równe napięciu VREF (u nas 3,3 V), to wynik pomiaru będzie liczba 1023 (0x03FF).
Przetwornikiem ADC wbudowanym w SAM7 można mierzyć napięcia także z rozdzielczością 8 bitów. Wtedy napięcie wejściowe zostanie podzielone na 256 przedziałów, a z rejestru „wynikowego" ADC_LCDR odczytamy bezpośrednio wartość 8-bitową. Gdybyśmy w funkcji adcGet chcieli włączyć tryb 8-bitowych pomiarów, to na początku funkcji adcGet przy konfiguracji przetwornika należałoby w rejestrze ADC_MR (Mode Register) ustawić bit LOWRES (jego maska znajduje się w makrodefinicji

AT91C_ADC_LOWRES). W takim przypadku wpis do rejestru ADC_MR wyglądałby na przykład tak:

```
AT91C_BASE_ADC→ADC_MR =
SHTIM_MASK | STARTUP_MASK | PRESCAL_MASK | //przygotowane maski
AT91C_ADC_LOWRES; //bit ustawiający niższą rozdzielczość
```

3.2 Konfiguracja timingu ADC

W tabeli dotyczącej maksymalne częstotliwości taktowania ADC (ADCClock) przy rozdzielczości 8- i 10-bitowej odczytujemy:

- 5 MHz dla rozdzielczości 10-bitowej i
- 8 MHz dla rozdzielczości 8-bitowej .

Znajomość tych częstotliwości posłuży nam do wyznaczenia podzielnika głównego zegara MCK, który taktuje wszystkie układy peryferyjne mikrokontrolera. Zegar MCK należy podzielić tak, aby częstotliwość ADCClock nie przekroczyła częstotliwości maksymalnej dla danej rozdzielczości. Podzielnik ustawimy w polu PRESCAL w rejestrze ADC_MR. Zależność pomiędzy częstotliwością ADCClock a MCK jest następująca:

$$\text{ADCClock} = \text{MCK} /((\text{PRESCAL}+1)\ 2)$$

gdzie:

- PRESCAL to wartość, którą wpiszemy do pola PRESCAL w ADC_MR,
- ADCClock to częstotliwość taktowania modułu ADC w [Hz],
- MCK to częstotliwość taktowania mikrokontrolera w [Hz] (najczęściej około 48 MHz).

Przykładowo, przy rozdzielczości 10-bitowej, dla MCK=48 MHz, częstotliwość taktowania ADC (ADCClock) równą 4,8 MHz uzyskamy dla podzielnika PRESCAL=4.
Po wybraniu ADCClock należy zająć się czasem wybudzania przetwornika. Biorąc pod uwagę, że:

$$\text{Startup Time} = (\text{STARTUP}+1) * 8 / \text{ADCClock}$$

uzyskujemy Startup Time=21,6 µs dla STARTUP=12, co stanowi wartość wystarczającą (wymagana >=20µs).
Ostatnim parametrem jest SHTIM, czyli pole definiujące czas działania układu S&H. Czas ten jest zależny od impedancji wyjściowej źródła sygnału analogowego. Im wyższa, tym dłuższy czas S&H należy ustawić. Wzór opisujący powyższą zależność jest następujący:

$$\text{Sample \& Hold Time} = (Z_{outMax} +4529)/7{,}69$$

Przykładowo, dla Z_{outMax}=10 kΩ uzyskujemy S&HT=1,9 µs. Podstawiając tę wartość do wzoru:

$$\text{Sample \& Hold Time} = \text{SHTIM}+1/\text{ADCClock}$$

uzyskujemy poszukiwaną wartość dla pola SHTIM=9

3.3. Przetwarzanie sygnału analogowego poprzez próbkowanie

Do próbkowania sygnału będziemy potrzebować dwóch narzędzi: przetwornika ADC i układu timera. Przetwornik ADC będzie mierzył chwilową wartość napięcia, a timer będzie odmierzał czas pomiędzy pomiarami (próbkami sygnału). Problem doboru częstotliwości

próbkowania wykracza poza zakres tego ćwiczenia, więc dla uproszczenie można przyjąć, że częstotliwość próbkowania powinna być co najmniej dwa razy większa od częstotliwości przetwarzanego sygnału.
Można sobie wyobrazić następującą realizację tego zadania. Rozpoczęcie pomiarów następuje w funkcji inicjalizującej próbkowanie wywoływanej z main. Funkcja przyjmuje jako argument numer kanału ADC, na którym mają być przeprowadzone pomiary. Wstępnie częstotliwość próbkowania ustalana jest wewnątrz tej funkcji.
Natomiast docelowa częstotliwość próbkowania jest ustawiana w main przez wywołanie funkcji odpowiedzialnej za realizację tego zadania z argumentem informującym, ile próbek na sekundę sobie życzymy.

3.3.1. Konfiguracja ADC

Tym razem konfiguracja będzie polegała na wprawieniu w ruch przetwornika za pomocą jednej funkcji inicjalizującej. Po zakończeniu działania tej funkcji przetwornik będzie mierzył napięcie i po każdym zakończonym pomiarze zgłaszał w naszym programie przerwanie.
Zacznijmy od przygotowania przydatnych makrodefinicji . Będziemy się dość często odnosić do rejestrów przetwornika, więc dla uproszczenia warto zapisać jego adres bazowy w krótszej formie:

```
#define ADC AT91C_BASE_ADC
```

Dodatkowo zdefiniujmy także, jakiej rozdzielczości przetwornika będziemy używali w programie:

```
#define ADC_RESOLUTION_10_BIT 1
```

Jeśli makrodefinicja ADC_RESOLUTION_10_BIT będzie miała wartość niezerową, to będzie oznaczało, że zbieramy dane z rozdzielczością 10 bitów. Natomiast jeśli ADC_RESOLUTION_10_BIT będzie miała wartość 0, znaczy to, że zbieramy próbki 8-bitowe. Dzięki tej makrodefinicji będziemy mogli w innych miejscach programu sprawdzić, czy operujemy na danych 8-, czy 10-bitowych.
Deklaracja funkcji rozpoczynającej próbkowanie jest następująca:

```
void adcSamplingInit(uint8_t channel)
```

Funkcja pobiera jako parametr liczbę channel reprezentującą kanał ADC, na którym chcemy okresowo mierzyć napięcie (czyli który chcemy próbkować). Poprawne wartości parametru channel to 0-7.
Pierwszą czynnością wykonywaną w funkcji adcSamplingInit jest ustawienie trybu pracy (w tym timingu) modułu ADC. Dokonujemy tego zależnie od wybranej rozdzielczości.

```
#if ADC_RESOLUTION_10_BIT==1 //dla rozdzielczości 10 bitów...
ADC→ADC_MR =
AT91C_ADC_TRGEN_EN | //włączenie sprzętowego wyzwalania
AT91C_ADC_TRGSEL_TIOA0 | //źródłem wyzwalania będzie timer T0
(0x4<<8) | //ustawienie PRESCALERA
(0x0C<<16) | //ustawienie pola STARTDP
(0x9<<24); //pole konfiguracji układu SSH
#else //dla rozdzielczości 8 bitów...
ADC→ADC_MR =
AT91C_ADC_TRGEN_EN | //włączenie sprzętowego wyzwalania
AT91C_ADC_TRGSEL_TIOA0 | //źródłem wyzwalania będzie timer T0
```

```
AT91C_ADC_LOWRES_8_BIT | //bit wymuszający rozdzielczość 8-bitową
(0x2<<8) | //ustawienie PRESCALERA
(0xl4<<16) | //ustawienie pola STARTDP
(0xB<<24); //pole konfiguracji układu SSH
#endif
```

W przypadku wyboru 8-bitowej rozdzielczości uaktywniony jest bit LOWRES w rejestrze ADC_MR. Nowością tutaj jest uaktywnianie bitu TRGEN za pomocą makrodefinicji AT91C_ADC_TRGEN_EN oraz wpisywanie wartości AT91C_ADC_TRGSEL_TIOA0 do pola bitowego TRGSEL. Bit TRGEN włącza wyzwalanie (rozpoczynanie konwersji) za pomocą układów peryferyjnych mikrokontrolera. Wpisując wartość do pola TRGSEL, wybieramy, który układ ma wyzwalać przetwornik. W poprzednim przykładzie rozpoczynaliśmy przetwarzanie ADC, wpisując 1 do bitu START w rejestrze ADC_CR (Control Register). W omawianym przykładzie o rozpoczęciu konwersji będzie decydował kanał 0 układu Timer Counter (TC). Jak wspomniałem wcześniej, TC będzie odpowiedzialny za próbkowanie z zadaną przez nas częstotliwością. W omawianej teraz funkcji inicjalizującej wywołujemy jedynie funkcję rozpoczynającą TC na potrzeby ADC, o której będzie mowa w następnym podpunkcie.
Następną czynnością w funkcji inicjalizującej ADC jest włączenie potrzebnego kanału. W tym celu najpierw odłączamy wszystkie kanały, a później wybieramy ten, który nas interesuje. Informacje, który kanał należy uaktywnić, otrzymaliśmy jako argument channel.
Włączanie i wyłączanie kanałów odbywa się w identyczny sposób jak poprzednio.

```
//zabezpieczenie przed niepożądanymi wartościami w parametrze
channel = channel & 0x07;
//wyłączamy każdy z 8 kanałów ADC
ADC→ADC_CHDR = 0xFF;
//włączamy zadany kanał
ADC→ADC_CHER = l<<channel;
```

Na koniec funkcji inicjalizującej skonfigurujemy i włączamy przerwania od modułu ADC. Uczynimy to bardzo podobnie, jak w przypadku innych układów peryferyjnych, używając funkcji AT91F_AIC_ConfigureIt i AT91F_AIC_EnableIt. Chcemy, aby przerwanie było generowane za każdym razem, gdy zakończy się konwersja, więc jako bit generujący przerwanie wybierzemy DRDY (Data Ready). Przerwania od poszczególnych bitów ADC włączamy przez ustawienie bitu odpowiadającego danemu zdarzeniu w rejestrze ADC_IER (Interrupt Enable Register).

```
//definiujemy, jaki priorytet będzie miało przerwanie od ADC
const uint8_t ADC_INTERRUPT_PRIORITY = 2;
//najpierw przygotowujemy kontroler AIC na przyjęcie przerwania
AT91F_AIC_ConfigureIt (AT91C_BASE_AIC, //adres bazowy AIC
AT91C_ID_ADC, //numer identyfikacyjny modułu ADC
ADC_INTERRUPT_PRIORITY, //zdefiniowany wcześniej priorytet
AT91C_AIC_SRCTYPE_INT_HIGH_LEVEL, //typ wyzwalania
adcIrqHandler //wskaźnik do funkcji obsługi przerwania
) ;
//włączamy przerwanie od modułu ADC
AT91F_AIC_EnableIt (AT91C_BASE_AIC, AT91C_ID_ADC);
//włączamy generowanie przerwania od bitu DRDY
```

```
ADC→ADC_IER = AT91C_ADC_DRDY;
```

3.3.2. Inicjalizacja układu Timer Counter dla ADC

Sprzętowy moduł Timer Counter (TC) w SAM7 składa się z trzech identycznie działających kanałów (channels) mających osobne adresy bazowe: AT91 C_BASE_TC0, ...TC1 i ...TC2. Do wyzwalania konwersji ADC zastosujemy kanał 0 o adresie bazowym AT91C_BASE_TC0.
Kanały timera mają także specyficzne dla siebie numery identyfikacyjne: AT91C_ID_TC0, ...TC1 i ...TC2. Naszemu timerowi przypisany jest numer AT91C_ID_TC0. Dla ułatwienia pisania kodu, na początku pliku adc.c znajdują się makrodefinicje adresu bazowego i identyfikatora wybranego timera:

```
#define TIMER AT91C_BASE_TC0
#define TIMER_ID AT91C_ID_TC0
```

Inicjalizację timera rozpoczniemy od włączenia jego sygnału zegarowego w module PMC (Power Management Controller).

```
AT91C_BASE_PMC→PMC_PCER = 1<<TIMER_ID;
```

Następnie, dla pewności sprowadzimy krytyczne rejestry timera do znanego stanu.

```
//wyłączamy zegar timera
TIMER→TC_CCR = AT91C_TC_CLKDIS;
//wyłączamy wszystkie przerwania od timera
TIMER→TC_IDR = 0xFFFFFFFF ;

//odczyt rejestru statusowego timera
//spowoduje wyzerowanie ewentualnych aktywnych flag
uint32_t dummy = TIMER→TC_SR;
//aby kompilator nie ostrzegał...
dummy = dummy;
```

Kolejną czynnością jest konfiguracja trybu pracy timera.
Konfigurację przeprowadzamy, uzupełniając bity i pola bitowe rejestru TC_CMR (Capture Mode Register).

```
TIMER->TC_CMR =
AT91C_TC_WAVE | //tryb "waveform mode"
SELECTED_TC_CLKS | //wybór podzielnika zegara
AT91C_TC_ACPA_SET | //co się dzieje w połowie odliczania
AT91C_TC_ACPC_CLEAR | //...i co na końcu odliczania
AT91C_TC_WAVESEL_UP_AUT0; //tryb automatyczny z przeładowaniem
```

Komentarza wymagać może w pierwszej kolejności makrodefinicja SELECTED_TC_CLKS Została ona dopisana na początku pliku adc.c wraz innymi makrodefinicjami służącymi do wyboru sygnału zegarowego dla układu TC (wybór sygnału zegarowego oznacza także wybór podzielnika tego sygnału). Wyboru sygnału zegarowego dokonujemy przez uzupełnienie pola CLKS w rejestrze TC_CMR. W przypadku projektu ADC_Timer wybrane źródło zegara to

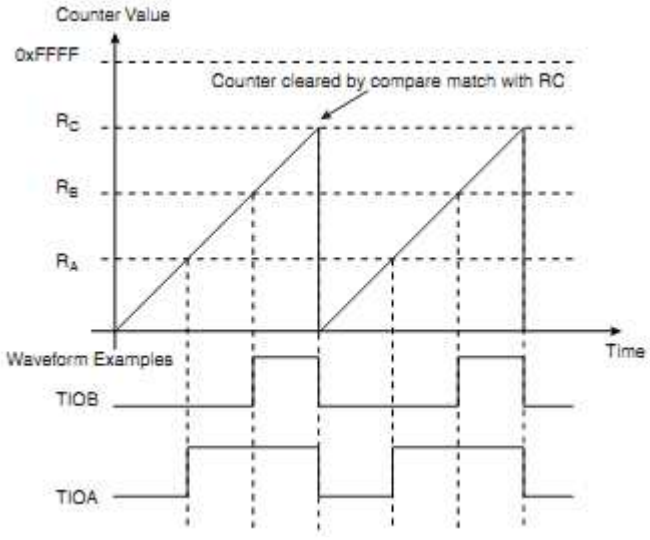
zegar główny MCK (o częstotliwości około 48 MHz) podzielony przez 128. Zestaw makrodefinicji dla pola CLKS jest następujący:

```
#define TC_CLKS_MCK2 0x0
#define TC_CLKS_MCK8 0x1
#define TC_CLKS_MCK32 0x2
#define TC_CLKS_MCK128 0x3
#define TC_CLKS_MCK1024 0x4

//w tej makrodefinicji dokonujemy wyboru źródła zegara
#define SELECTED_TC_CLKS TC_CLKS_MCK128

//a do tej wpisujemy wartość wybranego podzielnika
#define TC_PRESCALER 128
```

Ostatnia makrodefinicja (TC_PRESCALER) przyda się do konfiguracji częstotliwości przeładowań timera (która będzie także częstotliwością próbkowania).
Timer Counter przy naszych ustawieniach działa na zasadzie zwiększania wartości licznika TC_CV (Counter Value). Wartość licznika jest zwiększana z każdym taktem wybranego przez nas zegara. Podczas konfiguracji rejestru TC_CMR zaprogramowaliśmy ten licznik tak, aby reagował na dwa punkty przystankowe: na wartości w rejestrach TC_RA i TC_RC. Jeśli wartość w liczniku TC_CV zrówna się z zawartością rejestru TC_RA, nastąpi ustawienie sygnału TIOA wyzwalającego konwersję ADC.
Wspomniane ustawienie TIOA zaprogramowaliśmy, ustawiając pole bitowe ACPA makrodefinicją AT91C_TC_ACPA_SET. Natomiast gdy zawartość licznika TC_CV w timerze osiągnie wartość ustawioną w rejestrze TC_RC, nastąpi wyzerowanie sygnału TIOA wyzwalającego przetwornik ADC. Tę opcję włączyliśmy, ustawiając pole bitowe ACPC makrodefinicją AT91C_TC_ACPC_CLEAR. Oprócz zerowania sygnału TIOA, zrównanie się licznika TC_CV z zawartością TC_RC powoduje wyzerowanie TC_CV i - co za tym idzie - rozpoczęcie cyklu odliczania od nowa. Cała sytuacja jest przedstawiona na rysunku 4. W rejestrze TC_RA znajduje się wartość o połowę mniejsza niż w TC_RC, zatem sygnał TIOA ma kształt prostokątny o wypełnieniu 50% (rysunek przedstawia przebiegi dla innych proporcji TC_RA do TC_RC). Omawiany tutaj tryb pracy timera wybraliśmy makrodefinicją AT91C_TC_WAVESEL_ UP_AUTO wpisaną do pola WAVESEL rejestru TC_CMR.



Rys.4 Zależności czasowe układu TC (TIOA uruchamia ADC)

Opisany wyżej sygnał TIOA będzie wyzwalał konwersję w przetworniku ADC. Pamiętamy zapewne, że w konfiguracji ADC wybraliśmy do wyzwalania ten właśnie sygnał TIOA za pomocą makrodefinicji AT91C_ADC_TRGSEL_TIOA0 wpisanej do rejestru ADC_MR.
Częstotliwość próbkowania ustawimy przez wpisanie odpowiednich wartości do rejestrów TC_RA i TC_RC. Wiemy, że cały okres timera wybierzemy w rejestrze TC_RC, a do TC_RA wpiszemy połowę tego okresu. Z punktu widzenia wysoko poziomowej obsługi programu najwygodniej byłoby, gdybyśmy mogli wpisywać częstotliwość próbkowania od razu w hercach, czyli - w przypadku ADC - w samplach na sekundę. Do tego potrzebne jest wykonanie prostych przeliczeń zadanej częstotliwości na wartość, którą wpiszemy do TC_RA i TC_RC.
Jeśli założymy, że w zmiennej sps (od samples per second) znajduje się wartość częstotliwości próbkowania w [Sa/s], to do rejestru TC_RC i TC_RA wpiszemy:

```
TIMER→TC_RC = MCK / TC_PRESCALER / sps; //cały okres
TIMER→TC_RA = MCK / TC_PRESCALER / sps / 2; //pół okresu
```

Od teraz timer ustawiony będzie tak, aby generował sygnał wyzwalania dla ADC tyle razy na sekundę, ile wpisaliśmy do zmiennej sps. Po ustawieniu częstotliwości próbkowania, pozostało jeszcze włączyć timer (bitem CLKEN i wprawić maszynerię w ruch wyzwalając timer (bitem SWTRG). Obie czynności wykonamy przez wpisy do rejestru TC_CCR (Channel Control Register) timera.

```
//włączenie sygnału zegarowego wybranego kanału timera
TIMER→TC_CCR = AT91C_TC_CLKEN;
//rozpoczęcie działania timera
TIMER→TC_CCR = AT91C_TC_SWTRG;
```

W mikrokontrolerach SAM7 moduł TC jest rozbudowanym układem. Konfigurując go do pracy jako trigger (wyzwalanie) dla przetwornika ADC, wykorzystaliśmy zaledwie ułamek jego możliwości .

3.3.3. Funkcja obsługi przerwania

Dysponujemy już skonfigurowanym przetwornikiem ADC i timerem wyzwalającym przetwornik. Sam moduł ADC i kontroler przerwań zaprogramowane zostały tak, aby z każdą zakończoną konwersją ADC pojawiało się przerwanie. Podczas inicjalizacji ADC, jako funkcję obsługi przerwania (ISR) wskazaliśmy funkcję adcIrqHandler znajdującą się w module adc.c. Zawartość tej funkcji może wyglądać następująco:

```
void adcIrqHandler(void)
{
LED_R_TOGGLE;
uintl6_t adcResult;
adcResult = ADC→ADC_LCDR;
…….
}
```

Jedynym nieodzownym elementem tej funkcji jest linijka adcResult = ADC→ADC_LCDR; w której dokonujemy odczytu rejestru ADC_LCDR przechowującego wynik konwersji. Odczyt rejestru ADC_LCDR zeruje flagę DRDY w rejestrze statusowym, a właśnie ta flaga generuje przerwanie (ustawiana jest po zakończeniu konwersji). Odczytana wartość napięcia jest umieszczana w zmiennej adcResult. To, co zrobimy z wynikiem, należy do nas. W

projekcie przykładowym ADC_Timer wynik konwersji jest wyświetlany na wyświetlaczu LCD.

4. Program ćwiczenia

4.1. Uruchomić środowisko Eclipse, utworzyć nowy projekt a następnie zaimportować przykładowy projekt.
4.2. Przeanalizować pliki źródłowe projektu.
4.3. Skompilować przykładowy program i przesłać kod wynikowy do pamięci flash mikrokontrolera poprzez interfejs JTAG. Zaobserwować działanie programu.
4.4. Zrealizować przetwarzanie sygnału analogowego poprzez próbkowanie (p. 3.3)
4.5. Zrealizować pomiar temperatury przy pomocy termistora (na płytce). Do skalowania termometru przyjąć założenie, że charakterystyka napięciowa termistora w funkcji temperatury jest liniowa. Do skalowania wykorzystać podane dwa punkty pracy:
u = 1,405 V dla T = 20ºC
u = 1,710 V dla T = 30ºC