Politechnika Wrocławska

Podstawy Techniki Mikroprocesorowej
Laboratorium

Ćwiczenie 2
Przetwornik analogowo/cyfrowy (ADC)

Program ćwiczenia:

- obsługa przerwań,
- obsługa konwertera A/C.

Zagadnienia do przygotowania:

- jak do ćwiczenia 1,
- rejestry i obsługa konwertera A/C w ATmega8535,
- działanie konwertera i wynik konwersji.

Literatura:

[1] Dokumentacja mikrokontrolera ATmega8535, www.atmel.com.
[2] Opis instrukcji mikrokontrolerów AVR: AVR Instruction Set Manual, www.atmel.com.
[3] Wykład.
[4] Mikrokontrolery AVR ATmega w praktyce, R. Baranowski, BTC 2005.

Zawartość instrukcji

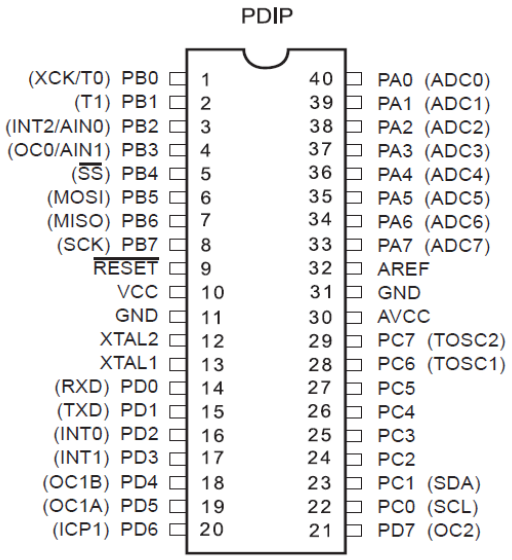
UWAGA!
Do poprawnej obsługi urządzeń niezbędne jest korzystanie z dokumentacji mikrokontrolera.

Instrukcja zawiera pewne uproszczenia w stosunku do rzeczywistych możliwości mikrokontrolera.

1. Wprowadzenie – podstawowe informacje o module ADC w ATmega8535

Przetwarzanie analogowo-cyfrowe (A/C) jest operacją bardzo często wykonywaną przez układy elektroniczne. Przetwarzanie elektrycznych wielkości analogowych (najczęściej napięcia) na postać cyfrową pozwala na konstruowanie cyfrowych układów pomiarowych i kontrolnych. Zastosowanie przetwornika ADC (ang. *Analog to Digital Converter*) wbudowanego w strukturę mikrokontrolera umożliwia m.in. budowanie tanich urządzeń pomiarowo-sterujących (w tym mikrosystemów) znajdujących zastosowania w wielu gałęziach przemysłu (motoryzacja, medycyna, mobilna aparatura kontrolna). Mikrokontroler ATmega8535 posiada wbudowany 10-bitowy przetwornik ADC z sukcesywną aproksymacją i czasem konwersji w zakresie 65÷260 µs. Sygnał wejściowy tego przetwornika doprowadzany jest za pomocą 8-kanałowego multipleksera (przełącznika) analogowego. Całkowity błąd przetwarzania przetwornika wynosi ± 2 LSB (ang. *Least Significant Bit*). Napięcie odniesienia może być doprowadzone z zewnątrz za pomocą wyprowadzenia AREF, jednak moduł przetwornika wyposażony jest także w wewnętrzne źródło napięcia odniesienia o wartości 2,56 V. Jako napięcia odniesienia można również użyć napięcia zasilania modułu przetwornika ADC (AVCC). Modułowi ADC przyporządkowane są następujące wyprowadzenia (rys. 1):

- AVCC – wejście napięcia zasilającego,
- AREF – wejście zewnętrznego napięcia odniesienia,
- ADC0.. ADC7 – wejścia sygnałów analogowych, dla których potencjałem odniesienia jest masa układu (potencjał wyprowadzenia GND),
- GND – wejście potencjału odniesienia mikrokontrolera (masa).



Rys. 1. ATmega8535 – rozmieszczenie i oznaczenia wyprowadzeń [1].

Aby konwersja sygnału analogowego na cyfrowy była możliwa, konieczne jest doprowadzenie do przetwornika dwóch napięć odniesienia. Dolnym napięciem odniesienia w mikrokontrolerach AVR jest zawsze potencjał masy układu (GND), czyli 0 V. Jako napięcie górne należy wskazać jedno z trzech wspomnianych wcześniej źródeł:

- wewnętrzne źródło napięciowe (2,56 V),
- wejście AREF,
- wejście AVCC.

Wykorzystywana na zajęciach makieta dydaktyczna została zaprojektowana w taki sposób, aby można było wykorzystać wejście AVCC o potencjale 5 V względem GND.

2. Sterowanie modułem ADC w mikrokontrolerze ATmega8535

Sterowanie modułem ADC obejmuje konfigurację rejestru **ADMUX** (ang. *ADC MUltipleXer selection register*) do zarządzania analogową częścią przetwornika (tab. 1) oraz rejestru **ADCSRA** (ang. *ADC Control and Status Register A*) do zarządzania cyfrową częścią przetwornika (tab 2). Wynik konwersji odczytywany jest z rejestru **ADC** podzielonego na rejestry **ADCH** i **ADCL**.

2.1. Rejestr ADMUX

Tab. 1. Organizacja rejestru ADMUX.

Bit	7	6	5	4	3	2	1	0
ADMUX	**REFS1**	**REFS0**	**ADLAR**	**MUX4**	**MUX3**	**MUX2**	**MUX1**	**MUX0**
Zapis/Odczyt	Z/O	Z/O	Z/O	Z/O	Z/O	Z/O	Z/O	Z/O
Wartość domyślna	0	0	0	0	0	0	0	0

Podczas inicjalizacji modułu ADC w rejestrze ADMUX należy wybrać kanał wejściowy ADCn konwertowanego sygnału (bity **MUX4...MUX0**, rys. 2) oraz źródło górnego napięcia odniesienia (bity **REFS1** i **REFS0**, rys. 3).

MUX4..0	Single Ended Input	Pos Differential Input	Neg Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			

Rys. 2. Kanały wejściowe konwertera ADC [1].

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Rys. 3. Napięcia odniesienia dla konwersji (AREF albo AVCC albo 2,56 V) [1].

Należy przy tym pamiętać o odpowiedniej konfiguracji rejestrów portu A, który współdzieli wyprowadzenia mikrokontrolera z modułem ADC. Linie portu A wykorzystywane do konwersji analogowo-cyfrowej powinny pracować w trybie wejściowym bez włączonego PULL-UP.

Bit **ADLAR** (ang. *ADC Left Adjust Result*) odpowiada za sposób zapisu 10-bitowego wyniku konwersji w rejestrach ADCH i ADCL. Dla domyślnej wartości ADLAR = 0 wynik zostanie wpisany do rejestrów „naturalnie" z wyrównaniem do prawej strony, tj. 2 najstarsze bity wyniku w ADCH, a 8 młodszych w ADCL. Gdy ADLAR = 1, wynik zostanie zapisany z wyrównaniem do lewej strony (8 najstarszych bitów wyniku w ADCH, 2 najmłodsze w ADCL) – zilustrowano to na rys. 4.

2.2. Rejestr ADC

Rejestr ADC, przechowujący 10-bitowy wynik konwersji, fizycznie składa się z dwóch 8-bitowych rejestrów: ADCH (*High*) i ADCL (*Low*). Łączna pojemność tych rejestrów to 16-bitów, ale wykorzystywanych jest tylko 10. Jak wspomniano wcześniej, sposób zapisu wyniku konwersji ustawiany jest za pomocą bitu ADLAR w rejestrze ADMUX, co ilustruje rysunek 4.

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Rys. 4. Sposób zapisu wyniku konwersji w zależności od ustawienia bitu ADLAR.

W parze rejestrów ADCH/ADCL zostaje umieszczona reprezentacja binarna wyniku konwersji sygnału analogowego na cyfrowy. Wynik ten zależy od poziomu napięcia zmierzonego na aktywnym kanale wejściowym ADCn (V_{in}) oraz od wybranego górnego napięcia odniesienia (V_{ref}):

$$wynik\_ADC = \frac{V_{in} \cdot 1024}{V_{ref}}$$

W celu zapewnienia poprawnej konwersji, rejestr ADCH jest podwójnie buforowany. Uzyskanie prawidłowego wyniku wymaga więc zachowania właściwej kolejności odczytu rejestrów: najpierw ADCL, potem ADCH. Odwrotna kolejność odczytu spowoduje uzyskanie wyniku złożonego z dwóch różnych momentów konwersji. Możliwy jest też odczyt tylko rejestru ADCH (przydatny, gdy wystarczy 8-bitowa rozdzielczość wyniku), zaś odczyt wyłącznie ADCL spowoduje zablokowanie konwersji.

ŹLE	**DOBRZE**
`in R21,ADCH`	`in R20,ADCL`
`in R20,ADCL`	`in R21,ADCH`

2.3. Rejestr ADCSRA

Tab. 2. Organizacja rejestru ADCSRA.

Bit	7	6	5	4	3	2	1	0
ADCSRA	**ADEN**	**ADSC**	**ADATE**	**ADIF**	**ADIE**	**ADPS2**	**ADPS1**	**ADPS0**
Zapis/Odczyt	Z/O	Z/O	Z/O	Z/O	Z/O	Z/O	Z/O	Z/O
Wartość domyślna	0	0	0	0	0	0	0	0

Mikrokontroler może być taktowany sygnałem o dowolnej częstotliwości z zakresu 0÷16 MHz, natomiast konwerter ADC jedynie z zakresu 50÷200 kHz (wyższe częstotliwości obniżają rozdzielczość wyniku). W celu ustawienia odpowiedniej częstotliwości taktującej moduł ADC wykorzystuje się wbudowany dzielnik częstotliwości (preskaler), który jest konfigurowany za pomocą bitów **ADPS2..0** (tab. 2, rys. 5). Na przykład, gdy taktowanie MCU wynosi 8 MHz, należy wybrać preskaler 64 lub 128. Należy zauważyć, że im mniejszy preskaler, tym szybciej zakończy się konwersja (np. dla 8 MHz i preskalera 64 będzie to $13 \cdot \frac{1}{8\ MHz\ /\ 64} = 104\ \mu s$, „13" ponieważ konwersja trwa 13 taktów zegara). Mikrokontrolery na makietach dydaktycznych taktowane są sygnałem o częstotliwości 1 MHz.

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Rys. 5. Dostępne dzielniki częstotliwości taktującej konwerter ADC.

Bit **ADIE** (ang. *ADC Interrupt Enable*) w rejestrze ADCSRA jest odpowiedzialny za aktywację przerwania po zakończeniu konwersji, tzn. gdy konwersja zakończy się, wówczas sygnalizowane jest

to ustawieniem flagi **ADIF**. Jeżeli wcześniej ustawiono wspomniany bit ADIE oraz aktywowano przerwania globalnie (np. SEI) – mikroprocesor rozpocznie obsługę przerwania (czyli przejdzie do linii nr 0x0E w kodzie programu – zgodnie z tab. 19 na s. 45 z dokumentacji ATmega8535).

Za pomocą bitu **ADATE** (ang. *ADC Auto Trigger Enable/ Free Running mode*) wybierany jest tryb pracy przetwornika – manualny (‘0’) bądź automatyczny (‘1’). W trybie manualnym każdą kolejną konwersję należy aktywować ręcznie (zob. niżej). W trybie automatycznym konwersja jest aktywowana seryjnie, tzn. kolejna rozpoczyna się automatycznie po zakończeniu poprzedniej. Konwersję automatyczną można przerwać wyłączając moduł konwertera lub resetując mikrokontroler.

Moduł konwertera ADC jest domyślnie wyłączony, a w celu jego włączenia należy ustawić bit **ADEN**. Rozpoczęcie konwersji wymaga ustawienia bitu **ADSC**, przy czym wcześniej (lub równocześnie) należy oczywiście włączyć konwerter.

Po zakończeniu konwersji wynik można sprawdzić odczytując rejestry ADCL i ADCH (lub tylko ADCH, gdy wystarczy 8-bitowa rozdzielczość wyniku), np. `in R16, ADCH`. Zakończenie konwersji można sprawdzić na dwa sposoby:
- obserwując flagę ADIF (‘1’ oznacza zakończoną konwersję; zeruje się automatycznie po odczycie wyniku z rejestru ADC);
- obserwując bit ADSC (‘0’ oznacza zakończoną konwersję) – ten sposób działa tylko dla konwersji manualnej.

3. Przykładowe zadania

1) Wykorzystując przetwornik A/C napisać program pozwalający na sterowanie częstotliwością migania wybranej diody LED (kopiowanie wyniku konwersji do rejestrów pętli opóźniającej). Za źródło odniesienia należy przyjąć napięcie zasilania mikroprocesora (AVCC). Jako sygnał wejściowy należy wykorzystać napięcie z potencjometru na makiecie.

2) Za pomocą przetwornika A/C opracować prosty woltomierz pracujący w zakresie 0÷5 V. Wynik powinien zostać wyświetlany na diodach LED w postaci naturalnego kodu binarnego. Za źródło odniesienia należy przyjąć napięcie zasilania mikroprocesora (AVCC). Jako sygnał wejściowy należy wykorzystać napięcie z potencjometru na makiecie.

3) Za pomocą przetwornika A/C opracować prosty woltomierz pracujący w zakresie 0÷5 V. Wynik powinien zostać wyświetlany na diodach LED w postaci linijki. Każda kolejna zaświecona dioda to wzrost mierzonego napięcia o 1,0 V. Za źródło odniesienia należy przyjąć napięcie zasilania

mikroprocesora (AVCC). Jako sygnał wejściowy należy wykorzystać napięcie z potencjometru na makiecie.

4) Za pomocą przetwornika A/C opracować prosty woltomierz pracujący w zakresie 0÷5 V. Wynik powinien być prezentowany na dwóch wyświetlaczach 7-segmentowych (jedności oraz części dziesiętne wolta) z dokładnością do 0,1 V. Można posłużyć się przykładem z **dodatku A**. Za źródło odniesienia należy przyjąć napięcie zasilania mikroprocesora (AVCC). Jako sygnał wejściowy należy wykorzystać napięcie z potencjometru na makiecie.

Dodatek A – rozkład liczby na setki, dziesiątki i jedności

ZADANIE: wyświetlić liczbę dziesiętną o wartości 217 na trzech wyświetlaczach 7-segmentowych.

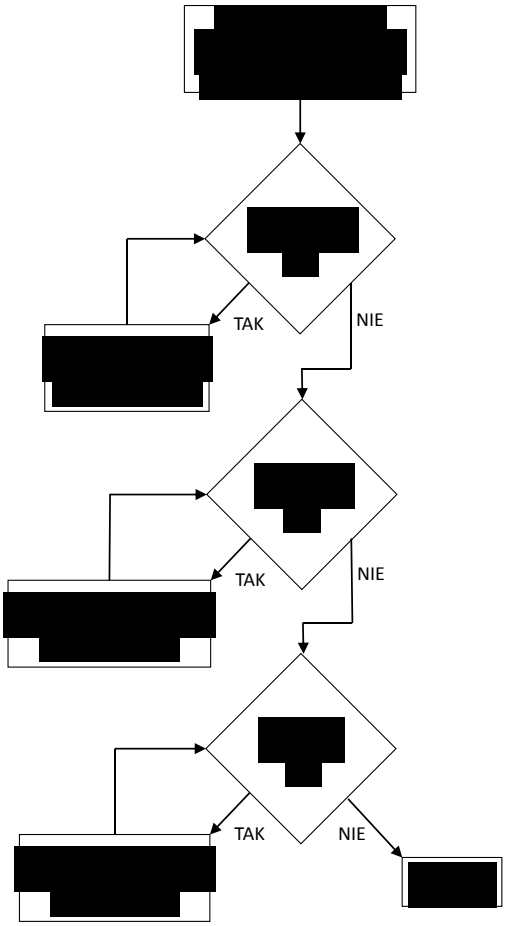
Na pierwszy rzut oka zadanie wydaje się trywialne. Na kolejne wyświetlacze 7-segmentowe A, B i C wysyłamy liczbę setek (A), liczbę dziesiątek (B) oraz liczbę jedności (C). Problem polega na tym, że człowiek patrząc na liczbę 217 od razu widzi: „2 setki, 7 jedności, itd", zaś mikrokontroler pracuje na liczbach binarnych i nie potrafi dokonać takiego podziału, o ile nie dostarczy się mu odpowiedniego algorytmu. Zresztą wystarczy odwrócić problem, żeby go lepiej uzmysłowić: 0b11011001 – ile jest tu pełnych setek, dziesiątek, jedności? Jak widać odpowiedź nie jest oczywista.

Programistyczny podział liczby binarnej na setki, dziesiątki i jedności nie jest skomplikowany. Algorytm takiego podziału może być następujący (patrz rys. A1):

a) utwórz 3 zmienne: *LICZBA_SETEK, LICZBA_DZIESIATEK, LICZBA_JEDNOSCI* (początkowo wszystkie są wyzerowane);
b) porównaj analizowaną liczbę (nazwijmy ją BIN) z liczbą 100;
jeśli jest większa/równa idź do (b2), jeśli mniejsza do (c);
b2) zmniejsz BIN o 100 oraz inkrementuj *LICZBA_SETEK;*
wróć do (b);
c) porównaj BIN z liczbą 10;
jeśli jest większa/równa idź do (c2), jeśli mniejsza do (d);
c2) zmniejsz BIN o 10 oraz inkrementuj *LICZBA_ DZIESIATEK;*
wróć do (c);
d) analogicznie.

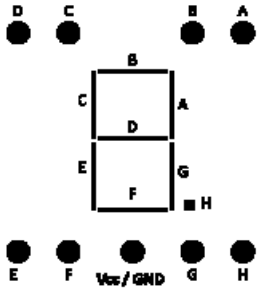
Rezultat:

LICZBA_SETEK = 2,
LICZBA_DZIESIATEK = 1,
LICZBA_JEDNOSCI = 7.



Rys. A1. Algorytm podziału liczby na setki, dziesiątki i jedności.

Mając w osobnych zmiennych (rejestrach roboczych) wartości setek, dziesiątek i jedności należy teraz „nauczyć" procesor, że np. gdy *LICZBA_SETEK* = 4 to na wyświetlaczu A musi pojawić się cyfra „4". W tym wypadku oznacza to wysłanie wartości „1" do segmentów a, c, d, g (rys. A2) tego wyświetlacza. Tutaj wystarczy prosta instrukcja warunkowa, która sprawdzi czy w zmiennej *LICZBA_SETEK* jest wartość „4". Jeśli tak, należy załączyć odpowiednie segmenty. Jeśli nie – przejść do sprawdzania kolejnych warunków (będzie ich 10, dla każdej z cyfr).



Rys. A2. Schemat wyświetlacza siedmiosegmentowego

Kod przykładowego programu:

```
.include "m8535def.inc"
.def SETKI = R20 //od teraz alternatywną nazwą R20 będzie "SETKI"
.def DZIES = R21 //można ich używać zamiennie
.def JEDN = R22
.def BIN = R16


//INICJALIZACJA
ldi SETKI,0 //początkowe wartości (zerujemy)
ldi DZIES,0
ldi JEDN,0
ldi BIN,235 //liczba, której podziału dokonujemy - dowolna, np. 235


//TUTAJ PODZIELIMY LICZBĘ NA SETKI, DZIESIĄTKI I JEDNOŚCI
rozklad:
cpi BIN,100     //porównaj liczbę BIN (235) z liczbą 100
                //(instrukcja zapisuje rezultat porównania na flagach w SREG)
brcc licz_100   //jeśli BIN >= 100 skocz do etykiety "licz_100"
cpi BIN,10
brcc licz_10    //jeśli BIN >= 10 skocz do etykiety "licz_10"
cpi BIN,1
brcc licz_1     //jeśli BIN >= 1 skocz do etykiety "licz_1"

//TUTAJ DECYZJA CO MA SIĘ POJAWIĆ NA WYSWIETLACZU "SETKI"
wyswietlacze:
cpi SETKI,0     //porównaj SETKI z wartością "0" (wynik na flagach)
breq setki_0    //jeśli są równe skocz do etykiety setki_0
cpi SETKI,1     //porównaj SETKI z "1"
breq setki_1    //jeśli równe - skok do setki_1
cpi SETKI,2     //porównaj SETKI z "2"
breq setki_2    //jeśli .....  itd - dla wszystkich cyfr
//...

//KONIEC PROGRAMU GŁÓWNEGO - PĘTLA ODDZIELAJĄCA OD PODPROGRAMÓW
j:
rjmp j

//PODZIAŁ NA SETKI, DZIESIATKI, JEDNOŚCI - PODPROGRAMY
licz_100:
    inc SETKI       //zwiększ liczbę zliczonych setek o 1
    subi BIN,100    //pomniejsz liczbę BIN o 100
    rjmp rozklad    //wróć i sprawdź czy (BIN-100) nadal jest ? 100
licz_10:
    //proszę przemyśleć i napisać analogicznie do "licz_100"
licz_1:
    //proszę przemyśleć i napisać analogicznie do "licz_100"

//WYŚWIETLANE ZNAKI - PODPROGRAMY
setki_0:
ldi r17,0b11101110  //ustawiamy cyfrę "0"
out portc,r17       //wyświetlamy (wyświetlacz podłączony do PORTC)
rjmp gdzies         //powrót w odpowiednie miejsce - przemyśleć gdzie
setki_1:
ldi r17,0b10000010  // ustawiamy cyfrę "1"
out portc,r17       //wyświetlamy
rjmp gdzies
setki_2:
ldi r17,0b11011100  // ustawiamy cyfrę "2"

// itd - dla wszystkich 10-u cyfr
//program realizuje wyświetlanie SETEK (wyświetlacz A)
//należy dodać jeszcze obsługę wyświetlaczy "DZIESIĄTKI" (B) oraz "JEDNOŚCI" (C)
```