KoNaR

KOŁO NAUKOWE ROBOTYKÓW

Sterowanie multipleksowe 4-cyfrowego wyświetlacza siedmiosegmentowego w oparciu o system przerwań mikrokontrolera ATmega16 w języku Asembler

Robert Budziński

Wrocław, 11. maja 2009

Spis treści

1. Sterowanie multipleksowe... 3
2. System przerwań 4
2.1. Idea działania przerwań 4
2.2. Uaktywnienie i blokowanie przerwań 4
2.3. Wektory przerwań 4
3. Obsługa wyświetlacza 5
3.1. Deklaracja stałych, zmiennych i nazw symbolicznych 5
3.2. Wypełnienie wektorów przerwań 5
3.3. Inicjalizacja wyświetlacza 6
3.4. Obsługa wyświetlacza 7
3.5. Wyświetlanie cyfr 8
4. Uwagi końcowe 9
5. Bibliografia 9

1. Sterowanie multipleksowe

W wielu układach do prezentowania wartości liczbowych stosuje się siedmiosegmentowe wyświetlacze LED. W większości przypadków jeden wyświetlacz to za mało, natomiast dołączenie kolejnego wyświetlacza sterowanego statycznie ciągnie za sobą konieczność wykorzystania następnych 8 linii mikrokontrolera. Aby zmniejszyć liczbę użytych wyprowadzeń mikrokontrolera przy sterowaniu wieloma wyświetlaczami LED należy zastosować sterowanie multipleksowe. Taki sposób sterowania minimalizuje liczbę potrzebnych do sterowania linii do 8 + liczba wyświetlaczy.

W sterowaniu multipleksowym wszystkie katody poszczególnych segmentów wyświetlaczy są połączone ze sobą. Anody wyświetlaczy są dołączone do plusa zasilania przez tranzystory p-n-p (np. BC558), których włączeniem steruje mikrokontroler. W ten sposób każda grupa segmentów posiada pojedynczą linię zasilającą. Przy multipleksowej obsłudze czterech wyświetlaczy siedmiosegmentowych wykorzystuje się jedynie 12 linii mikrokontrolera. Stosując sterowanie statyczne należałoby użyć 32 linie.

Jeśli grupy segmentów będą obsługiwane cyklicznie z dostatecznie dużą częstotliwością, to bezwładność oka ludzkiego spowoduje, że widz będzie miał wrażenie statyczności wyświetlanego obrazu.

Przy sterowaniu multipleksowym wyświetlaczy LED, dla zapewnienia odpowiedniej jasności świecenia, rezystory ograniczające prąd płynący przez segmenty powinny mieć wartości w zakresie od 47-100 Ω. Wynika to z impulsowego zasilania segmentów LED.

2. System przerwań

2.1. Idea działania przerwań

Programy sterujące mikrokontrolerami rzadko mają postać jednolitej listy instrukcji. Odpowiednie podprogramy muszą być wykonywane selektywnie i w kolejności zależnej od stanów modułów wewnętrznych lub sygnałów zewnętrznych. Umożliwia to system przerwań, który w zależności od zadanego warunku przenosi wykonywanie zadania w wybrane miejsce. Jeśli pewien układ zgłasza żądanie przerwania, mikrokontroler przerywa aktualnie wykonywane operacje i wykonuje podprogram przyporządkowany danemu źródłu przerwania, a następnie wraca do wcześniej wykonywanej czynności.

2.2. Uaktywnienie i blokowanie przerwań

Mikrokontroler domyślnie blokuje obsługę przerwań. Aby przerwania zostały obsłużone należy włączyć zezwolenie globalne. Do zarządzania zezwoleniami globalnymi służą instrukcje:

`sei` - zezwolenie na przerwanie

`cli` - zablokowanie przerwań

2.3. Wektory przerwań

Mikrokontrolery AVR wyposażono w wiele modułów będących źródłem przerwań, co wymaga ustalenia kolejności ich obsługi. Kolejność ta jest ustalona odgórnie przez wektor przerwań, czyli adres w pamięci programu, w którym umieszcza się zwykle instrukcję skoku do właściwego podprogramu. Położenie wektorów przerwań w pamięci programu dla mikrokontrolera ATmega16 przedstawia poniższa tabela:

Adres wektora	Źródło przerwania	Powód żądania
0x000	RESET	Inicjalizacja
0x002	INT0	Zewnętrzny
0x004	INT1	Zewnętrzny
0x006	TIMER2	Zgodne porównanie
0x008	TIMER1	Przepełnienie
0x00A	TIMER1	Przechwycenie
0x00C	TIMER1	Zgodne porównanie A
0x00E	TIMER1	Zgodne porównanie B
0x010	TIMER1	Przepełnienie
0x012	TIMER0	Przepełnienie
0x014	SPI	Koniec transmisji
0x016	USART	Poprawnie odebrany bajt
0x018	USART	Pusty bufor nadawczy
0x01A	USART	Koniec transmisji wychodzącej
0x01C	ADC	Konwersja skończona
0x01E	EEPROM	Gotowy
0x020	ANA_COMP	Zmiana stanu wyjścia
0x022	TWI	Każda aktywność modułu
0x024	INT2	Zewnętrzny
0x026	TIMER0	Zewnętrzny
0x028	SPM	Koniec zapisu pamięci flash

3. Obsługa wyświetlacza

3.1. Deklaracja stałych, zmiennych i nazw symbolicznych

Na początku programu zadeklarujemy wszystkie wykorzystywane stałe i zmienne oraz wprowadzimy nazwy symboliczne rejestrów związanych z portem, do którego dołączono wyświetlacz. Dzięki tej operacji program stanie się bardziej uniwersalny, a ewentualne modyfikacje w projekcie ograniczą się jedynie do kilku zmian.

```
.INCLUDE "m16def.inc"      ; SEGMENT DEKLARACJI
.EQU SYS_FREQ = 1          ; częstotliwość pracy w MHz
.EQU K_W7S_SEG = DDRA      ; rejestr kierunku portu segmentów
.EQU O_W7S_SEG = PORTA     ; rejestr wyjściowy portu segmentów
.EQU K_W7S_KOL = DDRD      ; rejestr kierunku portu kolumn
.EQU O_W7S_KOL = PORTD     ; rejestr wyjściowy portu kolumn
.EQU W7S_KOL0 = 7          ; nr linii dla kolumny 1. (cyfra najmłodsza)
.EQU W7S_KOL1 = 6          ; nr linii dla kolumny 2.
.EQU W7S_KOL2 = 5          ; nr linii dla kolumny 3.
.EQU W7S_KOL3 = 4          ; nr linii dla kolumny 4. (cyfra najstarsza)
.DSEG                      ; SEGMENT DANYCH
.ORG 0x0060                ; początek pamięci SRAM
W7s_kolumna: .BYTE 1       ; zmienna numeru aktywnej kolumny
W7s_cyfry: .BYTE 4         ; tablica 4. zmiennych przechowujących
                           ; wartości aktualnie wyświetlanych cyfr
```

3.2. Wypełnienie wektorów przerwań

Ponieważ wyświetlacz multipleksowy będzie korzystał z mechanizmu przerwań, konieczne jest wypełnienie odpowiednich wektorów przerwań. Podprogram obsługujący wyświetlacz powinien być uruchamiany przy zdarzeniu zgodnego porównania zawartości licznika 0 z rejestrem OCR0. Jeśli licznik ten uruchomimy w trybie CTC, to uzyskamy możliwość programowego sterowania częstotliwością multipleksowania.

Oprócz obsługi wyświetlacza wykorzystamy również przerwanie inicjalizacyjne. Jest to przerwanie o najwyższym priorytecie (ulokowane w adresie 0x000), a ponadto nie jest objęte blokadą. Wywoła ono podprogram inicjalizujący zawartość pamięci SRAM. Wypełnienie wektorów przerwań w początkowej części pamięci programu wygląda następująco:

```
.CSEG                      ; segment programu
.ORG 0                     ; wektor zerowania (adres 0x000)
   jmp Reset               ; skok do procedury inicjalizującej
.ORG OC0addr               ; wektor przerwania OC0 (adres 0x026)
   jmp Multipleksuj        ; skok do procedury obsługi wyświetlacza
```

Po włączeniu zasilania w pamięci SRAM znajdują się zwykle wartości przypadkowe, które mogą zakłócić pracę programu, dlatego też w programie inicjalizującym umieszczamy pętlę zerującą zawartość wszystkich zmiennych ulokowanych w SRAM. W naszym programie wykorzystujemy 5 pierwszych komórek SRAM, zatem pętlę zerującą wykonujemy pięciokrotnie, rozpoczynając od adresu 0x0060.

```
Reset:                     ; podprogram inicjalizujący
   ldi R17, high(RAMEND)
   ldi R16, low(RAMEND)
   out SPH, R17
   out SPL, R16            ; inicjalizacja wskaźnika stosu
   clr R16
   ldi R17, 5              ; R17 określa liczbę zajętych przez zmienne komórek
```

```
   ldi XH, high(0x0060)    ; pamięci SRAM (licznik poniższej pętli)
   ldi XL, low(0x0060)     ; X wskazuje na początek pamięci SRAM
Reset_1:                   ; pętla zerująca zawartość wszystkich używanych
   st X+, R16              ; przez zmienne komórek pamięci SRAM
   dec R17
   brne Reset_1
   rcall Ini_w7s           ; inicjalizacja wyświetlacza
   sei                     ; odblokowanie przerwań
```

3.3. Inicjalizacja wyświetlacza

Przygotowanie mikrokontrolera do pracy z wyświetlaczem polega na konfiguracji portów wejścia-wyjścia, licznika 0 oraz uaktywnienia generacji przerwań wywoływanych zdarzeniem zgodnego porównania. Licznik będzie okresowo przypominał o konieczności obsługi wyświetlacza.
Licznik 0 będzie działał w trybie CTC – jego zawartość będzie zerowana po zrównaniu z wartością rejestru OCR0. Źródłem impulsów zliczanych jest wartość zegara systemowego podzielona przez 1024. Zdarzenie zgodnego porównania wywoła przerwanie OC0. Rejestr OCR0 ładowany jest wartością, która określa częstotliwość generacji przerwania. Poprzez nastawę wartości rejestru OCR0 można manipulować okresem multipleksowania.

Zakładamy, że mikrokontroler pracuje z częstotliwością 1MHz. Przy podziale przez 1024 licznik będzie zliczał impulsy z częstotliwością 1000000/1024≈976,6 Hz. Aby migotanie nie było dostrzegalne, częstotliwość odświeżania pojedynczej cyfry powinna być większa lub równa 50Hz, a zatem 200Hz dla całego wyświetlacza 4-cyfrowego. Dzieląc wartość częstotliwości taktującą licznik 0 przez żądaną wartość odświeżania otrzymujemy wartość, którą należy wpisać do rejestru OCR0 – 976,6/200≈5. Na koniec liczbę tę mnożymy przez prędkość zegara taktującego mikrokontroler a określoną w sekcji deklaracji stałych jako SYS_FREQ, dzięki czemu zmiana rezonatora taktującego mikrokontroler będzie wymagała jedynie aktualizacji wartości SYS_FREQ. Jeśli migotanie jest dostrzegalne, do rejestru OCR0 można wpisać mniejszą wartość.

```
Ini_w7s:
   push R16
   ldi R16, (1<<WGM01)|(1<<CS02)|(1<<CS00)
   out TCCR0, R16          ; licznik 0. - tryb CTC, f=fosc/1024
   in R16, TIMSK
   sbr R16, 1<<OCIE0
   out TIMSK, R16          ; przerwanie OC0 uaktywnione
   ldi R16, SYS_FREQ*5
   out OCR0, R16           ; częstotliwość odświeżania pojedynczej
                           ; cyfry równa ok. 49 Hz
                           ; (przepełnianie z częstotliwością 195,3 Hz)
   ldi R16, 0xFF
   out K_W7S_SEG, R16      ; port segmentów w trybie wyjściowym
   in R16, O_W7S_KOL
   ori R16, (1<<W7S_KOL0)|(1<<W7S_KOL1)|(1<<W7S_KOL2)|(1<<W7S_KOL3)
   out O_W7S_KOL, R16      ; wyjścia kolumn w stanie wysokim (nieaktywne)
   in R16, K_W7S_KOL
   ori R16, (1<<W7S_KOL0)|(1<<W7S_KOL1)|(1<<W7S_KOL2)|(1<<W7S_KOL3)
   out K_W7S_KOL, R16      ; linie kolumn w trybie wyjściowym
   pop R16
   ret
```

3.4. Obsługa wyświetlacza

Parametrem wejściowym podprogramu obsługującego wyświetlacz siedmiosegmentowy jest tablica wartości cyfr zadeklarowana na początku programu jako *W7s_cyfry* w pamięci SRAM.

Podprogram sprawdza i inkrementuje numer wyświetlanej cyfry zapisany w zmiennej *W7s_kolumna* – jeśli cykl zakończono, ładuje wartością zerową. Wartość cyfry, która ma zostać wyświetlona na wyświetlaczu pobierana jest z odpowiedniej komórki tablicy *W7s_cyfry*.

```
Multipleksuj:
   push R16
   push R17
   push ZL
   push ZH
   in R16, SREG
   push R16
   lds R16, W7s_kolumna    ; odczytaj nr aktualnie wyświetlanej kolumny
   inc R16                 ; zwiększ nr aktualnie wyświetlanej kolumny
   cpi R16, 4
   brne Mul_w7s_0          ; jeśli zakończono cykl wyświetlania, to
   ldi R16, 0              ; wyzeruj nr aktualnie wyświetlanej kolumny
Mul_w7s_0:
   sts W7s_kolumna, R16    ; zapamiętaj nr aktualnie wyświetlanej kolumny
   ldi ZH, high(W7s_cyfry)
   ldi ZL, low(W7s_cyfry) ; Z wskazuje na tablicę wyświetlanych cyfr
   ldi R17, 0
   add ZL, R16
   adc ZH, R17             ; do Z dodawany jest nr aktualnie wyświetlanej
                           ;   kolumny
   ld R16, Z               ; R16 przyjmuje wartość cyfry, która powinna
                           ;   się znaleźć na aktualnie wyświetlanej pozycji
   ldi ZH, high(W7s_wzory_cyfr<<1)      ; Z wskazuje na tablicę 7-segmentowych
   ldi ZL, low(W7s_wzory_cyfr<<1)       ;   wzorów cyfr
   ldi R17, 0
   add ZL, R16             ; do Z dodawana jest wartość cyfry, której wzór
   adc ZH, R17             ; ma zostać odczytany
   lpm R16, Z              ; R16 przyjmuje wartość 7-segmentowego wzoru
                           ; cyfry, która ma zostać wyświetlona
   ldi R17, 0x00
   out K_W7S_SEG, R17      ; wyłącz chwilowo bufory wyjściowe linii segm.
                           ; (wygaszanie wyświetlacza na czas zmiany)
   out O_W7S_SEG, R16      ; wyświetl cyfrę na wyświetlaczu 7-segmentowym
   ldi ZH, high(W7s_kolejnosc<<1)       ; Z wskazuje na tablicę kolejności
   ldi ZL, low(W7s_kolejnosc<<1)        ; cyfr (wymuszeń na liniach kolumn)
   lds R16, W7s_kolumna    ; odczytaj nr aktualnie wyświetlanej kolumny
   ldi R17, 0
   add ZL, R16             ; do Z dodawany jest nr kolumny, która ma
   adc ZH, R17             ; zostać uaktywniona
   lpm R17, Z              ; R17 przyjmuje wartość nowego wymuszenia
                           ; na liniach kolumnowych
   in R16, O_W7S_KOL
   ori R16, (1<<W7S_KOL0)|(1<<W7S_KOL1)|(1<<W7S_KOL2)|(1<<W7S_KOL3)
   and R16, R17
   out O_W7S_KOL, R16      ; zmiana aktywnej kolumny (wymuszenie niskiego
                           ;   stanu na nowej linii kolumnowej)
   ldi R17, 0xFF
   out K_W7S_SEG, R17      ; włącz bufory wyjściowe na liniach segmentów
   pop R16
   out SREG, R16
   pop ZH
   pop ZL
   pop R17
   pop R16
   reti
```

Zamiana cyfry na stan linii segmentowych powodujących wyświetlenie odpowiedniego kształtu dokonuje się w oparciu o gotowy wzorzec zawarty w tablicy *W7s_wzory_cyfr*. Ponieważ każda z cyfr zapisywana jest w jednym bajcie, można zdefiniować dodatkowo znaki specjalne. W zamieszczonym niżej kodzie zdefiniowano litery ABCDEF oraz znaki „–" i „_".
Kolejność zapalania poszczególnych pozycji wyświetlacza również określona jest za pomocą tablicy wzorów – *W7s_kolejnosc*.
Kolejność wzorów wyznaczana jest rosnącymi wartościami odpowiadających im cyfr.
Kolejność segmentów w programie określono jako: kgfedcba

```
W7s_wzory_cyfr:
   .DB ~0b00111111, ~0b00000110  ; 0, 1
   .DB ~0b01011011, ~0b01001111  ; 2, 3
   .DB ~0b01100110, ~0b01101101  ; 4, 5
   .DB ~0b01111101, ~0b00000111  ; 6, 7
   .DB ~0b01111111, ~0b01101111  ; 8, 9
   .DB ~0b01110111, ~0b01111100  ; A, B
   .DB ~0b00111001, ~0b01011110  ; C, D
   .DB ~0b01111001, ~0b01110001  ; E, F
   .DB ~0b01000000, ~0b00001000  ; -, _

W7s_kolejnosc:  ; tablica wymuszeń dla portu kolumn
  .DB ~(1<<W7S_KOL0), ~(1<<W7S_KOL1), ~(1<<W7S_KOL2), ~(1<<W7S_KOL3)
```

3.5. Wyświetlanie cyfr

Wyświetlenie cyfry na danej pozycji wyświetlacza siedmiosegmentowego nastąpi automatycznie po załadowaniu odpowiedniej komórki pamięci tablicy *W7s_cyfry* zawartością określonego rejestru roboczego. Do zapisu wartości w pamięci danych wykorzystuje się komendę **`sts`**.
Przykładowo poniższy kod umieszczony w dowolnym miejscu programu głównego (main) spowoduje zapalenie na wyświetlaczu wartości -397.

```
push R17
ldi R17, 7
sts (W7s_cyfry+0), R17
ldi R17, 9
sts (W7s_cyfry+1), R17
ldi R17, 3
sts (W7s_cyfry+2), R17
ldi R17, 16
sts (W7s_cyfry+3), R17
pop R17
```

4. Uwagi końcowe

Niniejsze opracowanie przygotowano jako pomoc dydaktyczną w ramach kursu „Podstawy techniki cyfrowej i mikroprocesorowej”.

Kody źródłowe umieszczone w niniejszym opracowaniu stanowią kompletny program. Główną procedurę (main) należy umieścić po procedurze *Reset1*.
Program został uruchomiony i przetestowany za pomocą środowiska AVR Studio w wersji 4.16 Build 628, płytki edukacyjnej ARE0025 (are.net.pl) oraz programatora AVR Dragon (w trybie ISP). Nie zauważono żadnych nieprawidłowości w funkcjonowaniu programu.

5. Bibliografia

Opracowanie przygotowano na podstawie kodów źródłowych dostępnych na stronie Wydawnictwa BTC (http://www.btc.pl/pliki/atmwp.zip).

- Rafał Baranowski, *Mikrokontrolery AVR ATmega w praktyce*
- Marcin Wiązania, *Programowanie mikrokontrolerów AVR w języku BASCOM*
- Katedra Optoelektroniki i Systemów Elektronicznych, *Dokumentacja mikrokontrolera Atmega16 firmy Atmel*