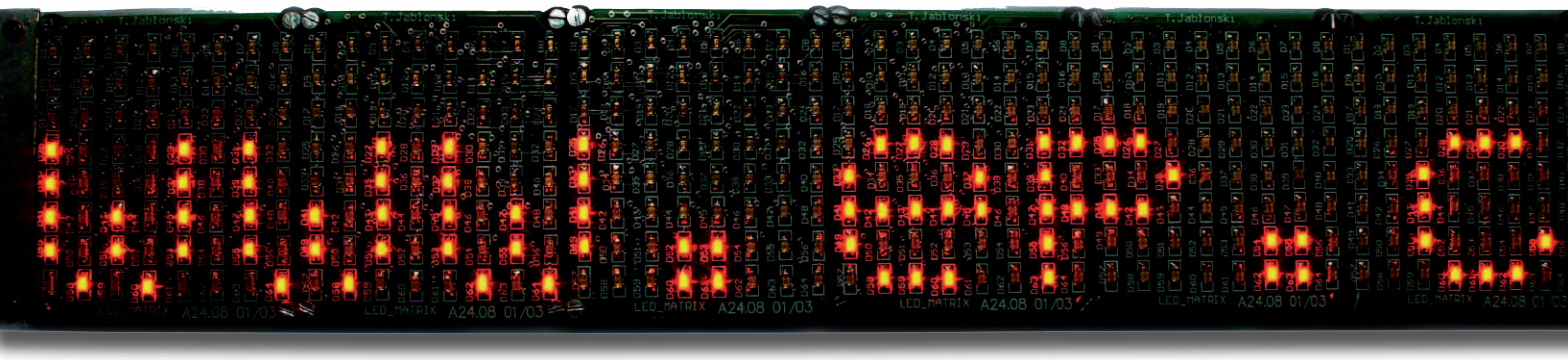


Graficzny, modułowy wyświetlacz LED

AVT 5226

Doskonały projekt wyświetlacza graficznego LED o konstrukcji modułowej. Moduły można łączyć w łańcuchy jak klocki, dostosowując wyświetlacz do własnych potrzeb. Idealne urządzenie do reklam świetlnych, publicznych stacji pogodowych, budowy tablic informacyjnych.

Rekomendacje: *Projekt zadowoli profesjonalistów i amatorów*

AVT-5226 w ofercie AVT:
AVT-5226A – płytka drukowana
AVT-5226B – płytka drukowana + elementy

Podstawowe informacje:
- konstrukcja modułowa,
- moduły łączone ze sobą „bokami",
- liczba diod: matryca 8×8 diod (pikseli),
- sterowanie przez interfejs I²C,
- zasilanie 5 V/0,64 A (opis w tekście),
- sterownik: mikrokontroler ATmega8

Dodatkowe materiały na CD i FTP:
ftp://ep.com.pl, user: 13835, pass: 4j4sfv4t
- wzory płytek PCB
- listingi
- karty katalogowe i noty aplikacyjne elementów oznaczonych na Wykazie elementów kolorem czerwonym

W interfejsach komunikacji z użytkownikiem królują wyświetlacze LCD. Trudno się temu dziwić, ponieważ mają praktycznie same zalety: niska cena, mały pobór prądu, wersje monochromatyczne i kolorowe, i stosunkowo proste sterowanie. Oprócz bardzo znanych wyświetlaczy alfanumerycznych ze sterownikiem HD44780 dużą popularność zdobywają wyświetlacze graficzne LCD. Mimo niezaprzeczalnych zalet, są jednak zastosowania, w których użycie technologii LCD jest albo zbyt drogie, albo wręcz niemożliwe. Zwykle, gdy wyświetlana informacja musi być bardzo dobrze widoczna z dużej odległości, stosowane są inne rozwiązania. Na przykład w panelach tablic synoptycznych lub układów sterowania procesami technologicznymi do wyświetlania cyfrowych wartości są używane klasyczne, 7-segmentowe wyświetlacze LED. Mają duży kontrast, a jasność ich świecenia nie zależy od zewnętrznych warunków oświetlenia. Aby zapewnić taki sam komfort odczytu wyświetlanej wartości w technologii LCD, trzeba by stosować kosztowne, duże panele graficzne, z bardzo dobrym podświetleniem.

Oprócz 7-segmentowych wyświetlaczy LED coraz chętniej stosowane są też matrycowe wyświetlacze LED o organizacji 8×8 lub 8×6 punktów i różnych wymiarach. Mają one wszystkie zalety 7-segmentowych wyświetlaczy LED, a oferują dużo większe możliwości. Stosując takie elementy z zewnętrznym sterownikiem, można wyświetlać czytelne napisy i ikony semigraficzne.

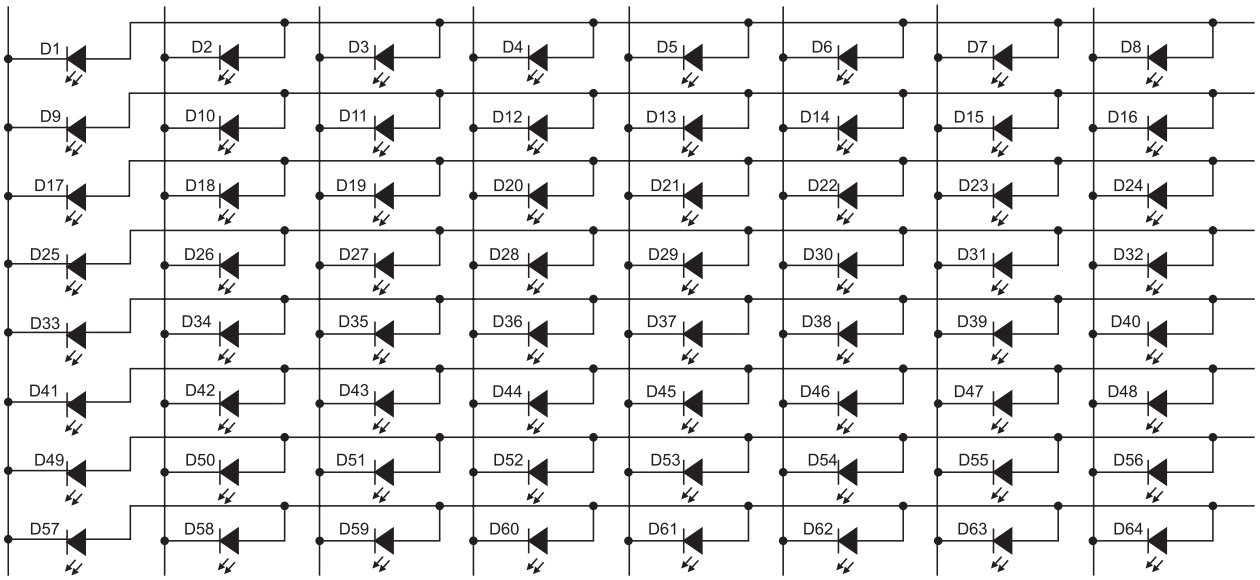
Większość z nas na pewno spotkała się z LED-owymi panelami informacyjnymi lub reklamowymi umieszczanymi w witrynach sklepów czy na przykład na dworcach. Nic nie stoi na przeszkodzie, by tam, gdzie jest to konieczne, stosować takie wyświetlacze we własnych konstrukcjach. Gotowe moduły matrycowe są drogie i raczej nie będą się nadawały. Zaprezentowany wyświetlacz jest tak pomyślany, by można było elastycznie, zależnie od potrzeb, dobierać liczbę segmentów, a sposób sterowania jest stosunkowo prosty.

Wstęp

Przed zaprojektowaniem wyświetlacza LED trzeba się zastanowić nad rodzajem sterowania. Do wyboru jest sterowanie statyczne i sterowanie dynamiczne. Każdy z nich ma swoje zalety i wady.

Sterowanie statyczne polega na sterowaniu każdej diody LED oddzielnie. Jeżeli dioda ma być zapalona, to podaje się na nią napięcie przez rezystor szeregowy ograniczający prąd. Zależnie od oczekiwanej jasności, prąd może być większy lub mniejszy, ale nie może przekraczać maksymalnego, ciągłego prądu dopuszczalnego dla diody wybranego typu. Dla zwykłych diod LED małej mocy ten prąd ma wartość 30 mA, a prąd pracy 20 mA. Dzięki rozwojowi technologii współczesne diody LED świecą już dość jasno przy prądzie ok. 5 mA, jednak z moich doświadczeń wynika, że wystarczający jest prąd o natężeniu około 10 mA, ale jeżeli dioda ma bardzo jasno świecić, to trzeba, by prąd miał wartość ok. 20 mA.

Podstawową wadą sterowania statycznego jest konieczność użycia dużej liczby linii sterujących i duży, sumaryczny prąd pobierany przez układ wyświetlacza. Załóżmy, że chcemy zbudować sterowany statycznie moduł 8×8 diod LED. Do sterowania 64 diod potrzeba będzie 64 linii sterujących, maksymalny prąd pobierany 64×10 mA=640 mA, a dla prądu 20 mA – 64×20 mA=1280 mA. Trudno sobie wyobrazić, aby taki wyświetlacz był sterowany bezpośrednio z linii portów mikrokontrolera sterownika. Do sterowania statycznego produkowane są specjalne układy scalone zawierające wielobitowy, równoległy rejestr zapisywany szeregowo i wzmacniacze prądu. Oczywiście, w trakcie pracy moduł nie będzie pobierał maksymalnego prądu, bo taki prąd będzie płynął, kiedy wszystkie diody są zapalone. Mimo to, średnio pobierany prąd nawet dla jednego modułu 8×8 punktów jest dość duży. Zaletą wyświetlania statycznego jest mały poziom generowanych zakłóceń, brak efektu migotania i mimo konieczności zapisywania układów driverów – dość łatwe sterowanie.



Rys. 1. Połączenie 64 diod w układ matrycowy 8×8

Sterowany statycznie moduł wyświetlacza LED wykonany z użyciem driverów produkowanych przez firmę STM widziałem na szkoleniowym spotkaniu STM TechDay i wyświetlane na nim informacje prezentowały się znakomicie.

Wyświetlanie dynamiczne wymaga połączenia diod LED w układ matrycowy – **rys. 1**.

Przy takim połączeniu otrzymujemy 8 kolumn i 8 wierszy. Do ich sterowania potrzeba tylko dwóch 8-bitowych portów ze wzmacniaczami prądowymi. Uproszczenie układu jest okupione bardziej skomplikowanym sterowaniem. Na początku sekwencji sterowania na linię pierwszej kolumny K1 wystawiany jest stan niski. Każda z diod, której katoda jest połączona do linii kolumny K1, a anoda ma połączone napięcie zasilania przez rezystor szeregowy, będzie się świecić. Inaczej mówiąc, na linię kolumny K1 wystawiamy stan niski, a na 8-bitowy rejestr wiersza stany wysokie dla tych diod, które mają w kolumnie K1 się zaświecić.

W następnym cyklu sterowania zdejmujemy stan niski z kolumny K1, a wystawiamy stan niski na kolumnę K2. Jednocześnie do rejestru wiersza wpisujemy stany wysokie dla tych diod, które mają się zaświecić w kolumnie K2. Sekwencja jest powtarzana cyklicznie dla wszystkich linii kolumn od K1 do K8. Jeżeli cykle sterowania będą się powtarzać z wystarczająco dużą częstotliwością, to bezwładność oka spowoduje, że na siatkówce powstanie obraz, tak jakby wybrane diody dla każdej kolumny były jednocześnie zaświecone dla całego modułu 8×8 diod.

Każda z diod kolumny jest zaświecana przez 1/8 okresu całkowitego cyklu sterowania wyświetlaczem. Z tego powodu średni prąd pobierany przez tak pracujący wyświetlacz jest dużo mniejszy niż przy wyświetlaniu statycznym.

Wykaz elementów

Rezystory: (SMD, 1206)
R9...R16: 150...330 Ω (opis w tekście)
R1...R8: 10 kΩ
Kondensatory
C2, C4, C5: 1 µF/25 V (tantalowy, SMD)
C1, C3, C6, C8: 100 nF (ceramiczny 1206)
Półprzewodniki
U1: TPIC6C596 (SMD)
U2: ATmega8 (SMD)
Tranzystory T1...T8: BSP171
D1...D64: diody LED czerwone (SMD 1206)

Zasada działania

Po zastanowieniu wybrałem wyświetlanie dynamiczne. Statyczne wyświetlanie wymagałoby zastosowania specjalizowanych driverów, a ja postanowiłem zbudować układ z w miarę popularnych i dostępnych elementów. Ten argument oraz zdecydowanie mniejszy pobór prądu przeważyły.

Następnym krokiem po wybraniu rodzaju sterowania było określenie koncepcji całego wyświetlacza. Założyłem, że wyświetlacz będzie miał wysokość 8 punktów (diod), a długość będzie wielokrotnością 8 punktów (diod). Naturalną konsekwencją takiego wyboru było podzielenie wyświetlacza na moduły o organizacji 8×8 punktów. Każdy z modułów ma własny sterownik odpowiedzialny za dynamiczne wyświetlanie i odbieranie wyświetlanej informacji ze sterownika głównego. Do komunikacji sterownika głównego ze sterownikami modułów zastosowałem interfejs I²C. Sterownik główny jest układem master, a moduły układami slave. Zastosowanie magistrali I²C pozwoliło na ograniczenie sterowania do 2 linii: SDA i SCL.

Schemat modułu pokazano na **rys. 2**. Jako sterownik zastosowano popularny i tani mikrokontroler ATmega8 w wersji SMD. Mikrokontroler ma wystarczającą liczbę linii portów i co ważne, sprzętowy interfejs I²C (TWI), który może pracować również w trybie slave.

Linie kolumn są podłączone do wyjść układu TPIC6C596. Jest to 8-bitowy rejestr równoległy ze wzmacniaczami prądu. Budowa i działanie tego układu zostanie dokładniej opisana przy okazji omawiania programowych procedur zapisywania danych do rejestru. Wzmacniaczami prądowymi TPIC6C696 są tranzystory DMOS z kanałem N o maksymalnym ciągłym prądzie wyjściowym 100 mA. Tranzystory pracują w konfiguracji otwartego drenu (OD).

Jako wzmacniacze linii wierszy są wykorzystane tranzystory MOS z kanałem P typu BSP171. Według danych katalogowych rezystancja kanału w stanie przewodzenia nie jest większa niż 0,36 Ω, a maksymalny prąd drenu jest równy 1,7 A. Wszystkie źródła tranzystorów są połączone do zasilania +5 V. Jeżeli na bramkę zostanie podany stan niski, to tranzystor wchodzi w stan nasycenia, podając napięcie zasilania na linie wiersza. W każdej z linii wiersza szeregowo z drenem tranzystora sterującego jest włączony rezystor ograniczający prąd o wartości 150...220 Ω. Wartość rezystora jest zależna od zastosowanych diod LED i żądanej jasności świecenia. Trzeba pamiętać, że układ dynamicznego sterowania może się w sytuacjach awaryjnych zatrzymać (zawiesić). Wtedy rezystor ogranicza prąd, tak by jego wartość nie zniszczyła diody. Napięcie na czerwonej diodzie LED w czasie przewodzenia wynosi ok. 1,8 V. Dla rezystora 150 Ω prąd diody w trakcie statycznego wyświetlania wynosi (5 V–1,8 V)/150 Ω =21,3 mA. Przy wyświetlaniu dynamicznym wartość tego rezystora jest wystarczająca. Jeżeli sekwencja wyświetlania zatrzyma się na dłuższy czas, to prąd jest ustawiony na granicy wartości uznawanej za prąd pracy diody. Dla bezpieczeństwa można zastosować rezystory o rezystancji 200 Ω. Wtedy prąd statyczny będzie miał bezpieczną wartość 16 mA, ale wyświetlacz będzie świecił mniej intensywnie. Ja w mo-

Na CD: karty katalogowe i noty aplikacyjne elementów oznaczonych na wykazie elementów kolorem czerwonym

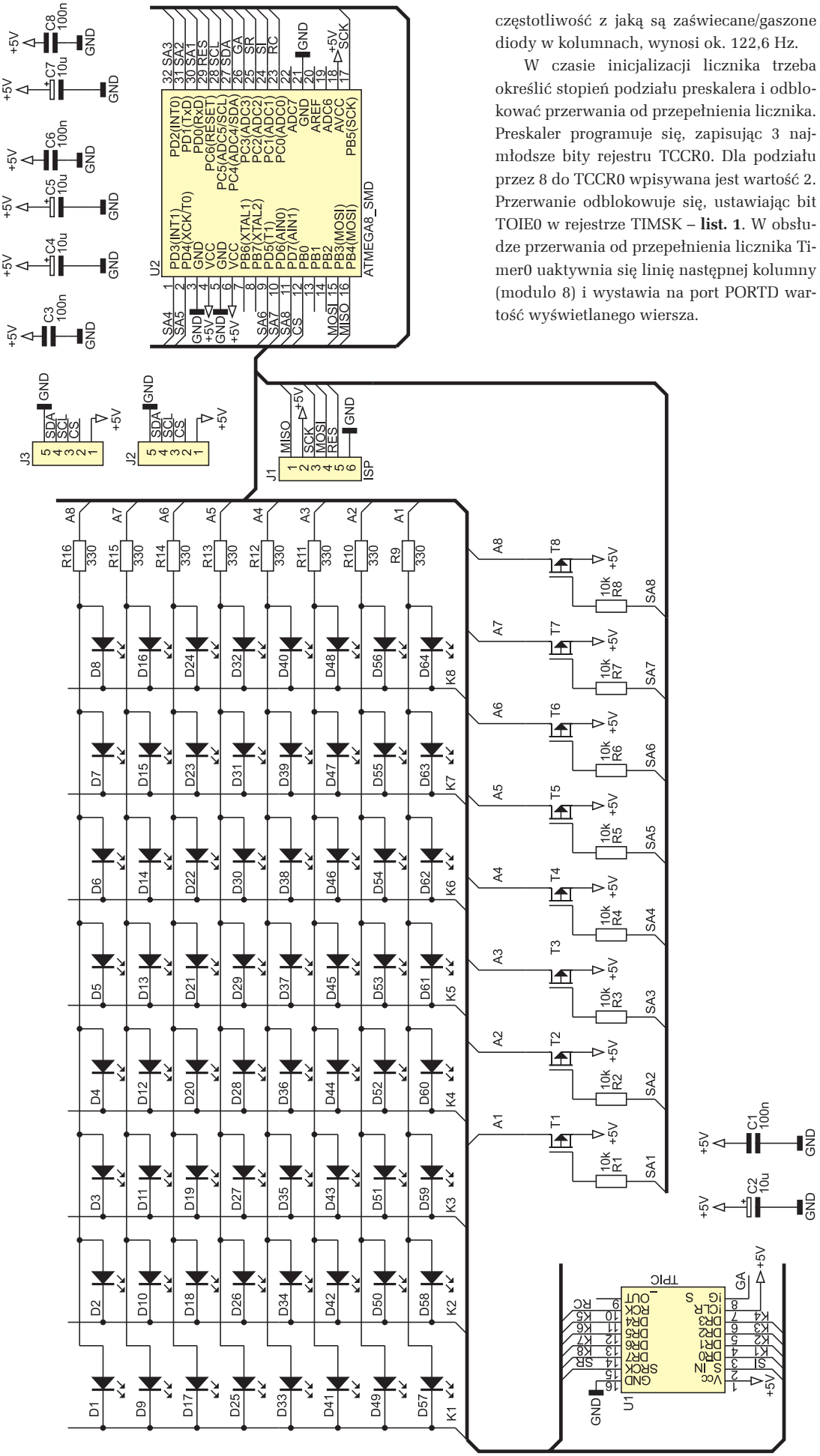
delowym egzemplarzu zastosowałem rezystory o wartości 150 Ω.

Układ sterownika jest wyposażony w złącze J1 do programowania mikrokontrolera w systemie i 2 złącza do podłączenia sygnałów interfejsu I²C i zasilania modułu.

Sterowanie

Dynamiczne wyświetlanie najłatwiej zrealizować, używając licznika, który generuje przerwanie przy przepełnieniu. Można wtedy w prosty sposób kontrolować częstotliwość uaktywniania kolejnych linii kolumn. Mikrokontroler ATmega8 ma wbudowane 3 liczniki: Timer0, Timer1 i Timer2. W programie sterującym modułem użyto 8-bitowego licznika Timer0. Licznik ten zlicza sygnał o częstotliwości taktowania mikrokontrolera z możliwością wstępnego podziału tej częstotliwości przez preskaler. Mikrokontroler jest taktowany wewnętrznym oscylatorem RC o częstotliwości 2 MHz. Jeżeli podzielmy tę częstotliwość przez 8, to Timer0 będzie zliczał sygnał o częstotliwości 250 kHz. Licznik zlicza w górę i przepełni się po odliczeniu 255 impulsów. Wpisanie do niego wartości początkowej równej 0x05 spowoduje, że licznik przepełni się po odliczeniu 250 impulsów. Jeżeli po każdym przepełnieniu będziemy wpisywać wartość początkową 5, to przepełnienia będą zgłaszane z częstotliwością 250 kHz/250=1 kHz. Można zrezygnować z inicjowania licznika i przerwania będą zgłaszane z częstotliwością 250 kHz/255=980,4 kHz. Ponieważ każda z linii kolumn jest sterowana co 8 cykli wyświetlania, częstotliwość z jaką są zaświecane/gaszone diody w kolumnach, wynosi ok. 122,6 Hz.

W czasie inicjalizacji licznika trzeba określić stopień podziału preskalera i odblokować przerwania od przepełnienia licznika. Preskaler programuje się, zapisując 3 najmłodsze bity rejestru TCCR0. Dla podziału przez 8 do TCCR0 wpisywana jest wartość 2. Przerwanie odblokowuje się, ustawiając bit TOIE0 w rejestrze TIMSK – **list. 1**. W obsłudze przerwania od przepełnienia licznika Timer0 uaktywnia się linię następnej kolumny (modulo 8) i wystawia na port PORTD wartość wyświetlanego wiersza.

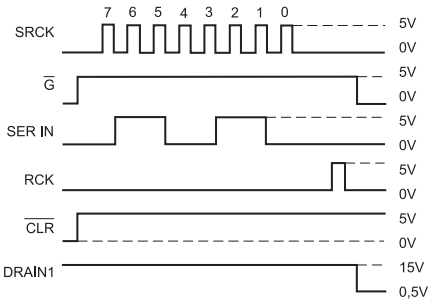


Rys. 2. Schemat modułu wyświetlacza

List. 1. Konfiguracja licznika Timer0
```
TCCR0=2;//bez preskalera
TIMSK|=(1<<TOIE0);//odblokowanie przerwania od przepełnienia  TMR0
```

List. 2. Zapis danych do rejestru TPIC6C696
```
/*************************************************************
Definicja linii sterujących
*************************************************************/
#define WR_0 //linia zapisu di rej, równoległego - RCK
#define DATA_1//linia danych SER IN
#define CLK_2 //linia zegarowa SRCK
#define GA_3 //linia bramkowania !G
#define WR_0  (PORTC&=~(1<<WR));
#define WR_1  (PORTC|=(1<<WR));
#define DATA_0  (PORTC&=~(1<<DATA));
#define DATA_1  (PORTC|=(1<<DATA));
#define CLK_0  (PORTC&=~(1<<CLK));
#define CLK_1  (PORTC|=(1<<CLK));
#define GA_0  (PORTC&=~(1<<GA));
#define GA_1  (PORTC|=(1<<GA));
/*************************************************************
Procedura zapisania 8-bitowej danej z argumentu data do rejestru
*************************************************************/
void SendTpic(unsigned char data){
  char i;
        GA_1      //!G=1
        WR_0      //RCK=0
        CLK_1     //SRCK=1
      for(i=0;i<8;i++){
       if((data&0x80)==0x80)
        {DATA_1}
         else
         {DATA_0}
         data<<=1;
         CLK_0             //zapisanie opadającym zboczem
         CLK_1
      }
      WR_1//wpis do rejestru równoległego
      WR_0
      GA_0        //!G=0

}
```



Rys. 4. Przebiegi czasowe zapisywania rejestrów układu TPIC6C696

Linia kolumny jest uaktywniana (linia jest aktywna, kiedy ma potencjał masy) przez wpisanie jedynki na odpowiadającej tej linii pozycji rejestru układu TPIC6C696.

Zapis rejestrów danych układu TPIC6C696 jest taktowany sygnałem zegarowym SRCK. Dane do wpisania są ustalane na wejściu SER IN w czasie wysokiego stanu na linii zegarowej SRCK, a wpisywane poczynając od najstarszego bitu (MSB) do najmłodszego (LSB) w takt opadającego zbocza SRCK. Po 8 taktach zapisany jest cały 8-bitowy rejestr C1. Teraz dane z rejestru C1 trzeba przepisać do rejestru równoległego C2 opadającym zboczem na wejściu RCK. Buforowanie wyjścia zapobiega niekontrolowanym zmianom na wyjściach rejestru w czasie szeregowego wpisywania danych do C1. W czasie wpisywania danych na wejściu G powinien być stan wysoki. Wtedy na wyjściach bramek AND sterujących tranzystory wyjściowe są stany niskie i tranzystory są zatkane. Stan niski na wejściu !CLR zeruje wszystkie wyjścia przerzutników. Przebiegi czasowe na liniach sterujących wpisem do rejestru TPIC6C696 pokazano na **rys. 4**. Na **list. 2** widzimy procedurę zapisu danych do rejestru z definicją linii sterujących.

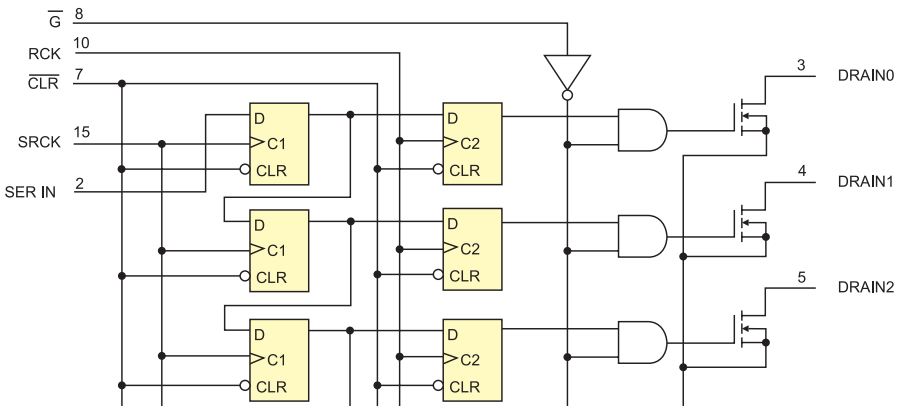
Kompletną procedurę obsługi przerwania od przepełnienia Timer0 zamieszczono na **list. 3**.

W trakcie prac na programem wyświetlania pojawił się problem. W opisywanej już zasadzie wyświetlania dynamicznego powiedzieliśmy, że w każdym cyklu przerwania trzeba wystawić napięcie zasilania na anody diod, które mają być zaświecone w aktywnej kolumnie (sterowanie wierszami) i uaktywnić kolumnę przez podanie na linie potencjału masy. Jednak po uaktywnieniu napięć w wierszu, jeszcze przez chwilę potrzebną na uaktywnienie nowej kolumny, jest aktywna poprzednia kolumna. To powodowało powstawanie prześwitów na wyświetlaczu. Żeby temu zapobiec, trzeba było na początku procedury ustawić wszystkie linie kolumn jako nieaktywne przez wpisanie do TPIC6C696 samych zer. Potem można ustalić stany na linii wierszy i na koniec uaktywnić kolejną kolumnę. Po tej modyfikacji wyświetlanie przebiegało prawidłowo.

Licznik aktywnej kolumny znajduje się w zmiennej *col*. Ta zmienna adresuje tablicę *TabCol*, w której są wpisane 8-bitowe dane do rejestru układu TPIC6C696, tak by w jego rejestrach była wpisana 1 na pozycji odpowiadającej numerowi zapisanemu w zmiennej *col*.

8-elementowa tablica *Disp* musi zawierać 8 bajtów określających wyświetlaną informację. Każdy bajt odpowiada pionowemu paskowi o wysokości 8 punktów (diod LED). Najbardziej znaczący bit (MSB) odpowiada najwyżej położonemu punktowi w pasku. Zależność pomiędzy zawartością tablicy *Disp* a wyświetlaną informacją pokazano na **rys. 5**.

Wygodnie jest stworzyć taką tablicę przy założeniu, że dioda jest zapalona, kiedy bit jest ustawiony. W rzeczywistości tranzystor

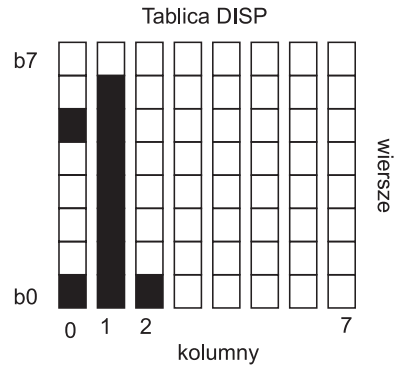


Rys. 3. Fragment schematu blokowego układu TPIC6C696

List. 3. Procedura obsługi przerwania od przepełnienia Timer0
```
const char TabCol[8]={1,2,4,8,0x10,0x20,0x40,0x80};
#pragma vector=TIMER0_OVF_vect
__interrupt void TmrOvl(void){

  SendTpic(0);                        //wyłączenie wszystkich kolumn
  PORTD=~(Disp[col]);      //wystawienie stanów na liniach wiersza
  SendTpic(TabCol[col]);  //uaktywnienie kolejnej kolumny
  ++col;                   //przygotowanie numeru kolumny dla następnego
przerwania
  col&=7;
  if(tmsek==1){            //fragment używany do odliczania opóźnień
  --msek;
  if(msek==0)
    tmsek=0;}

}
```



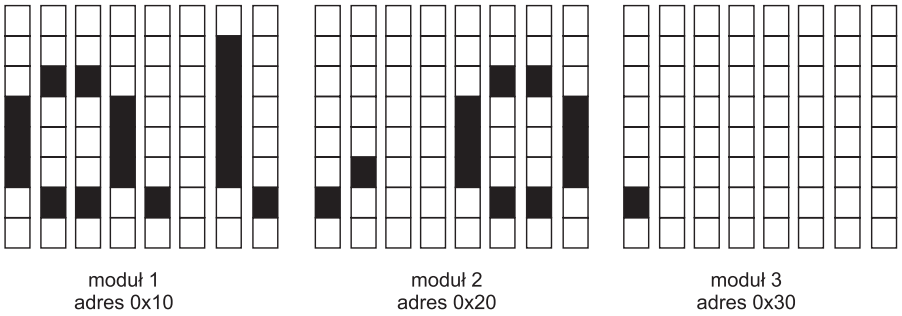
Rys. 5. Zależność pomiędzy zawartością tablicy Disp a wyświetlaną informacją

List. 4. Pętla odczytu danych przesyłanych po I²C i ich zapisu do tablicy Disp

```
while(1){

    if ( ! TWI_Transceiver_Busy() ) {
      if ( TWI_statusReg.lastTransOK )//czy ostatnia operacja się powiodła
      { __watchdog_reset();
        if ( TWI_statusReg.RxDataInBuf )//czy ostatnia operacja była odbiorem
danych
       {TWI_Get_Data_From_Transceiver(I2Cbuf, 8);  //odbiór 8 bajtów
            for(i=0;i<8;i++)
               Disp[i]=I2Cbuf[i];
           if ( TWI_Transceiver_Busy() )
          TWI_Start_Transceiver();}
       }

      }//end if ( ! TWI_Transceiver_Busy() )
```



Rys. 6. Idea wyświetlania znaków alfanumerycznych

z kanałem P zaczyna przewodzić, kiedy na bramce jest stan niski. Dlatego w procedurze przerwania elementy z tablicy przed wysłaniem na port PORTD są negowane.

W praktyce, przy tak napisanym programie, wystarczy zapisać tablicę *Disp*, a o resztę troszczy się procedura przerwania. Bajty zapisywane do tablicy *Disp* są przesyłane do modułu magistralą I²C. Moduł wyświetlacza jest elementem slave i ma swój unikalny adres pozwalający go jednoznacznie zidentyfikować. Mikrokontroler ATmega8 ma wbudowany interfejs sprzętowy TWI mogący pracować w trybie I²C slave. Atmel udostępnia na swoich stronach internetowych notę aplikacyjną *„AVR311. Using TWI module as I²C slave”*. Nota zawiera opis i kompletne źródła. Do własnego projektu trzeba dołączyć pliki TWI_slave.c i TWI_slave.h. Na **list. 4** pokazano pętlę nieskończoną, której zadaniem jest odebranie sekwencji start i adresu slave. Adres I²C modułu jest ustalany po wywołaniu funkcji *TWI_Slave_Initialise(unsigned char adr_slave)*. Jeżeli adres przesyłany przez układ master jest adresem modułu, to mikrokontroler w obsłudze przerwania odbiera 8 bajtów i wpisuje je do bufora *I2Cbuf*, a następnie zawartość *I2CBuf* jest przepisywana do tablicy *Disp*.

Z punktu widzenia zewnętrznego sterownika, moduł jest widziany jako układ I²C slave, do którego trzeba wpisać 8 kolejnych bajtów, które są potem przepisywane do tablicy *Disp*. Do jednej magistrali można podłączyć w prosty sposób wiele modułów, a sterowanie nimi jest bardzo proste. Ma to duże znaczenie praktyczne, bo do zbudowania wyświetlacza należy użyć kilku modułów. Na list. 4 pokazano procedurę użytą do zapisywania danych z bufora *BufPom* modułów wyświetlacza przez sterownik zbudowany w oparciu o mikrokontroler PIC18F2580.

Wyświetlacz złożony z kilku modułów będzie miał wysokość 8 punktów i szerokość równą liczbie modułów pomnożoną przez 8. Taki wyświetlacz jest idealny do wyświetlania tekstu. Pierwotnie założyłem, że każdy moduł będzie miał swoją pamięć generatora znaków alfanumerycznych i żeby wyświetlić znak wystarczy wysłać do modułu kod ASCII. Jednak znaki alfanumeryczne na matrycy 8×8 nie wyglądają dobrze. Zazwyczaj znak jest definiowany w matrycy 6×8 punktów. Sam znak ma rozmiar 5×8 punktów, a szósta kolumna jest dodawana jako przerwa pomiędzy znakami. Początkowo chciałem zrobić moduły 6×8 pikseli, ale po przemyśleniu pozostałem przy koncepcji 8×8 i zrezygnowania z generatora znaków w module. Można przecież traktować moduł jako obszar wyświetlacza „graficznego”, a funkcje generatora znaków i zapisywanie tablic *Disp* poszczególnych modułów powierzyć zewnętrznemu sterownikowi. Wyświetlacz jest traktowany przez zewnętrzny sterownik jako obszar punktów o rozmiarze 8xn*8, gdzie n to liczba zastosowanych modułów. W modelowym rozwiązaniu zastosowałem 6 modułów i otrzymałem wyświetlacz o rozmiarze 8×48 punktów. Zewnętrzny sterownik ma za zadanie utworzenie bufora o rozmiarze 48 bajtów i zapisanie go wartościami na podstawie tablicy generatora znaków. Potem sterownik logicznie dzieli bufor na 6 części, każdej części przydziela adres slave modułu i wysyła dane do każdego modułu magistralą I²C. Nie jest to zadanie zbyt skomplikowane i daje dużą elastyczność wyświetlania informacji. Można na przykład prosto zrealizować skrolowanie dłuższych napisów. Na **rys. 6** pokazana jest idea takiego rozwiązania. Napis z 3 znaków 6×8 punktów jest wyświetlany na 3 modułach. Na trzecim module można wyświetlić jeszcze jeden znak.

Na modelowym wyświetlaczu składającym się z 6 modułów można wyświetlić w ten sposób 8 znaków o rozmiarze 6×8 punktów.

Sterownikiem zewnętrznym może być dowolny sterownik mikroprocesorowy. Może to być na przykład sterownik zegara z kalendarzem, termometru, stacji pogodowej, tablicy reklamowej itp. Do testowania użyłem modułu ZL5PICz z mikrokontrolerem PIC18F458 produkowanego przez firmę KAMAMI.

Żeby wyświetlać napisy, trzeba zdefiniować tablice z generatorem znaków. Definicja jednego znaku to 6 kolejnych bajtów: 5 bajtów dla znaku i jeden zerowy dla przerwy pomiędzy znakami. Kiedyś zdefiniowałem tablicę generatora znaków dla wyświetlacza od telefonu Nokia 3310. Tablica jest tak zbudowana, że kod ASCII znaku pomnożony przez 6 określa początek definicji znaku w tablicy. Generator znaków można wyko-

List. 5. Zapisanie 8 bajtów do modułu wyświetlacza o adresie zawartym w argumencie *adr*.

```
//zapis 8 bajtów do wyświetlacza
void WriteLed(char addr){
char i=0;
         i2c_start();
         i2c_write(addr);//adres slave R/W=0
         i2c_write(BufPom[i++]);
         i2c_write(BufPom[i++]);
         i2c_write(BufPom[i++]);
         i2c_write(BufPom[i++]);
         i2c_write(BufPom[i++]);
         i2c_write(BufPom[i++]);
         i2c_write(BufPom[i++]);
         i2c_write(BufPom[i++]);
         i2c_stop();
}
```

List. 6. Zamiana bitów miejscami

```
//konwersja SWAP
unsigned char ConvData(unsigned char data){
char i,zn;
  zn=0;
  for(i=0;i<8;i++){
    if((data&0x80)==0x80)
      zn|=0x80;
    data<<=1;
    zn>>=1;}
   return(zn);
}
```

rzystać w naszym wyświetlaczu, ale wzorce znaków zostały zdefiniowane odwrotnie, to znaczy najwyższy punkt pionowej linijki odpowiada najmłodszemu bitowi. W naszym wyświetlaczu jest tak, że najwyższy punkt odpowiada najstarszemu bajtowi (rys. 5). Żeby nie powtarzać żmudnego procesu definiowania znaków, w każdym bajcie pobranym z tablicy generatora bity są zamieniane miejscami. Wykonuje to procedura pokazana na **list. 6**.

Procedura DispChar (**list. 7**) na podstawie kodu znaku code umieszczonego w argumencie wylicza początek definicja znaku w tablicy generatora rom_gen. Potem pobieranych jest 6 kolejnych bajtów z tablicy generatora, bajty są poddane konwersji procedurą ConvData i zapisane w buforze BufPom.

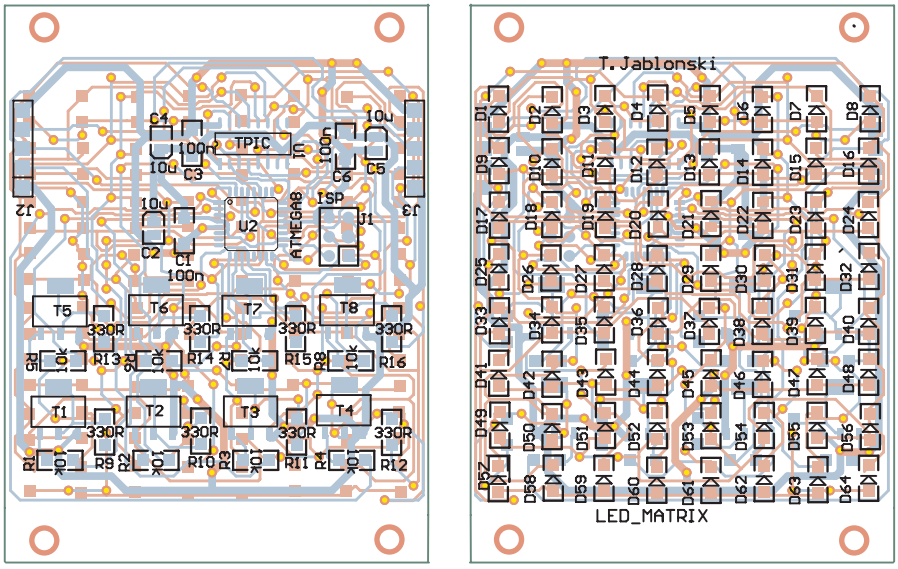
Jeżeli potrafimy zapisywać 6 kolejnych bajtów wzorca w buforze na podstawie kodu ASCII znaku, to możemy teraz napisać procedurę, która zapełni wzorami znaków cały bufor (pamięć wyświetlacza) o rozmiarze 48 bajtów dla 8 znaków 6×8 punktów. Procedura DispLed najpierw zeruje pamięć wyświetlacza, a potem pobiera kolejne kody ASCII z ciągu znaków argumentu procedury. Na podstawie kodu ASCII jest pobieranych kolejno 6 bajtów z tablicy generatora znaków i bajty te są zapisywane do bufora pamięci wyświetlacza BufString. Procedura automatycznie wykrywa koniec ciągu znaków i kończy swoje działanie. Nie ma tu zabezpieczeń i trzeba pamiętać, by napis nie był dłuższy niż rozmiar bufora podzielony przez 6.

Wywołanie procedury DispLed zapełnia pamięć BufString i trzeba teraz jej zawartość przesłać do kolejnych modułów wyświetlacza. Zakładamy, że adresy kolejnych modułów od lewej do prawej mają wartości 0x10, 0x20, 0x30, 0x40, 0x50 i 0x60. Procedura SendDisp pobiera kolejno 8 bajtów z bufora pamięci BufString – **list. 8** – i wysyła do kolejnych modułów. Argument ST określa, od której pozycji pamięci BufString zacząć wyświetlanie. Ma to znaczenie, kiedy napis jest dłuższy od 8 znaków i konieczne jest jego skrolowanie.

Montaż i uruchomienie

Moduł wyświetlacza zmontowano na dwustronnej płytce drukowanej (**rys. 7**). Wszystkie elementy są w wersji do montażu powierzchniowego. W czasie projektowania płytki przyjąłem założenie, że diody LED będą umieszczone na górnej warstwie, a wszystkie elementy sterownika modułu na dolnej warstwie. Montaż nie jest trudny, ale wymaga dokładności i cierpliwości. Dla 6 modułów trzeba prawidłowo przylutować aż 384 diody LED.

Płytkę zaprojektowano tak, że po złożeniu kilku modułów „na styk" odstępy między kolumnami sąsiednich płytek są takie same, jak



Rys. 7. Płytka modułu wyświetlacza

List. 7. Zapisywanie pamięci wyświetlacza wzorcami znaków

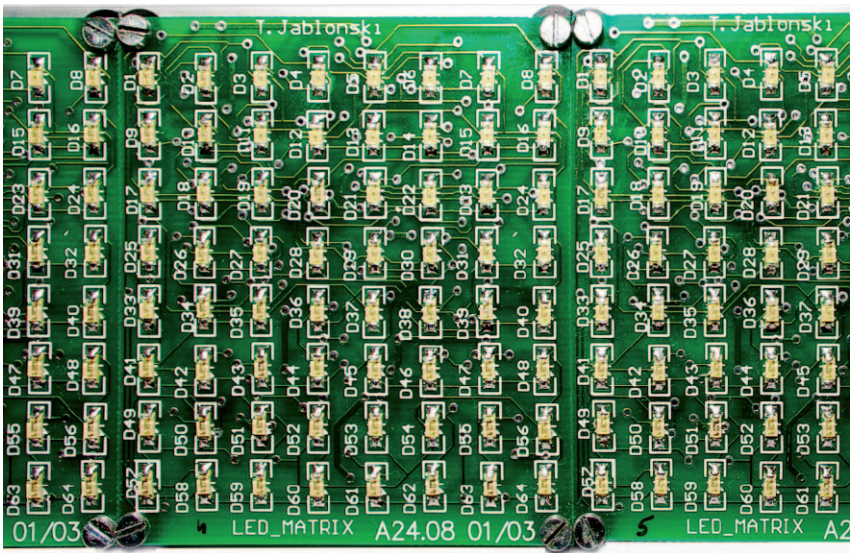
```
void DispChar(char code){
int code_point;
        code_point=((int)code*6);//wyliczenie  początku definicji znaku
w tablicy
        BufPom[0]=ConvData(rom_gen[code_point++]);
        BufPom[1]=ConvData(rom_gen[code_point++]);
        BufPom[2]=ConvData(rom_gen[code_point++]);
        BufPom[3]=ConvData(rom_gen[code_point++]);
        BufPom[4]=ConvData(rom_gen[code_point++]);
        BufPom[5]=ConvData(rom_gen[code_point]);
}
```

List. 8. Pobranie wzorca znaków z tablicy generatora

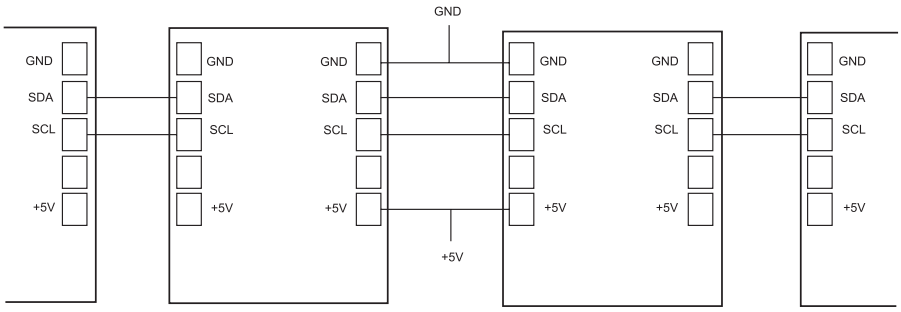
```
void DispLed(const char *s){
char i,j;
        j=0;
        for(i=0;i<MAXBUF;i++)//czyszczenie bufora
        BufString[i]=0;
        while(*s){
        DispChar(*s++);
        for(i=0;i<6;i++)
        BufString[j++]=BufPom[i];
        }
}
```

odstępy pomiędzy kolumnami na płytkach. W ten sposób powstaje jedna płaszczyzna z kolumnami w równych odstępach (**fot. 8**).

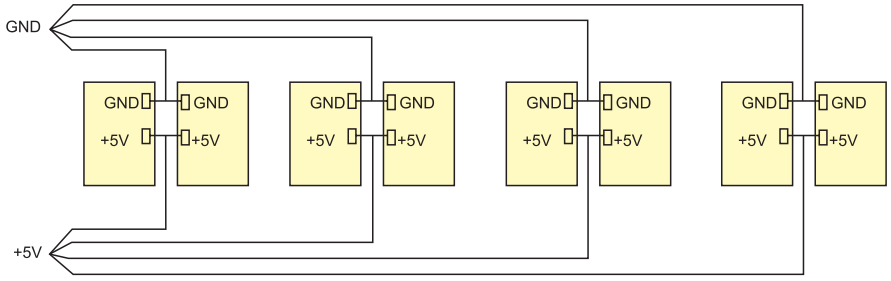
W modelowym rozwiązaniu 6 modułów zamocowano do dwu ceowników aluminiowych wkrętami M2,5 z plastikowymi tulejkami dystansowymi. Na krótszej krawędzi tak powstałego wyświetlacza przykręcono 9-pinowe męskie złącze DSUB. Tym złączem są doprowadzone do wyświetlacza: napięcie zasilające +5 V, masa oraz sygnały interfejsu SDA i SCL (**fot. 9**).



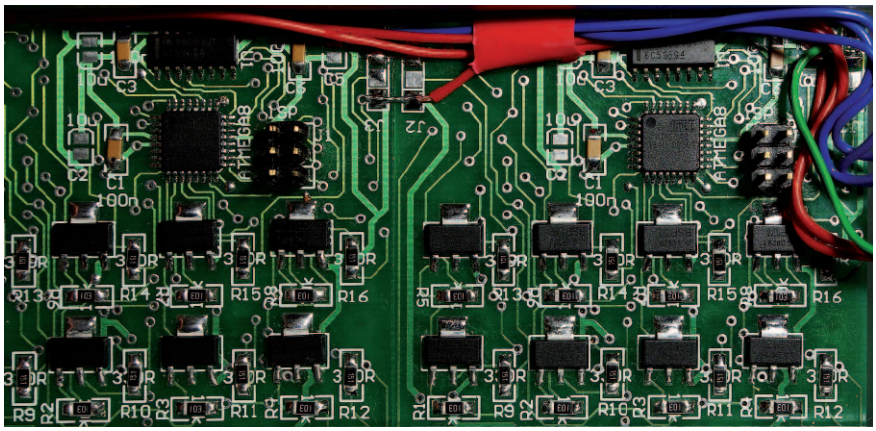
Fot. 8. Mechaniczne połączenie modułów



Rys. 9. Elektryczne połączenie modułów



Rys. 10. Podłączenie zasilania



Fot. 11. Moduł wyświetlacza – widok od spodu

Na krawędziach płytek modułów od strony elementów sterownika umieszczono punkty lutownicze przeznaczone do podłączenia zasilania i sygnałów interfejsu (**rys. 10**). Wyprowadzenia z obu krawędzi płytek są ze sobą połączone. Po zmontowaniu płytek na styk wystarczy tylko połączyć krótkimi przewodami sąsiadujące pola. Tak połączone są sygnały SDA i SCL. W przypadku zasilania modułów sprawa wygląda trochę inaczej.

Dynamiczne wyświetlanie powoduje impulsowy pobór prądu ze źródła zasilania. Gdybyśmy połączyli szeregowo zasilanie modułów, to przez ścieżki tego, do którego podłączone zostaną przewody zasilające od strony zasilacza, będzie płynęła suma prądów pobieranych przez wszystkie moduły i może się okazać, że napięcie zasilające mikrokontroler tego i najbliżej niego położonych modułów jest zbyt zakłócone. Dlatego w modelowym wyświetlaczu napięcia zasilające są podłączane izolowanymi przewodami, równolegle do pary modułów. Przewody zasilające z każdych takich par są podłączone do pinów zasilających złącza DSUB 9 (**rys. 11**).

Po zmontowaniu wszystkich modułów i mechanicznym ich połączeniu w jedną płaszczyznę wyświetlacza trzeba zaprogramować każdy z modułów. Bity konfiguracyjne mikrokontrolera ATmega8 pokazano na **rys. 12**.

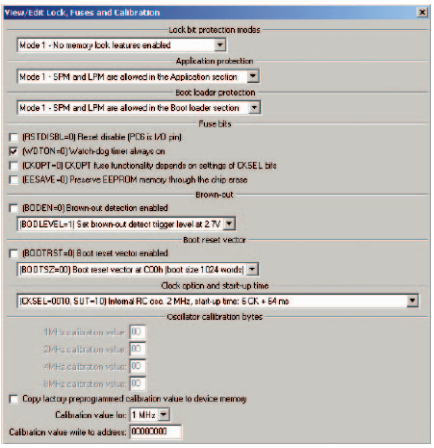
List. 9. Wysłanie zawartości bufora *BudString* do 6 modułów wyświetlacza

```
/*********************************************
zapisanie bufora do 6 segmentów wyświetlacza
*********************************************/

void SendDisp(char st){
char i,k;
char pom[8];
         for(i=0;i<6;i++)
          {
           for(k=0;k<8;k++)
            {BufPom[k]=BufString[st++];
             if(st==MAXBUF)
                  st=0;
                  }

       WriteLed(i*0x10+0x10);

         }
}
```



Rys. 12. Bity konfiguracyjne mikrokontrolera ATmega8

Moduły ze względu na brak miejsca na płytce nie mają możliwości sprzętowego ustawiania adresów slave. Dlatego zostało przygotowanych 8 plików z adresami 0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70 i 0x80. Program jest tak napisany, że po włączeniu zasilania i przed wysłaniem danych przez I²C, moduł o adresie 0x10 wyświetla cyfrę 1, o adresie 0x20 cyfrę 2 itd. Pozwala to na łatwą identyfikację modułów i wyeliminowanie błędów na przykład zaprogramowania podwójnego adresu. Do programowania jest przeznaczone 6-stykowe złącze J1. Mimo że piny tego złącza nie są przeznaczone do montażu przewlekanego, można do nich przylutować dwurzędowe złącze goldpinów.

Po zaprogramowaniu wszystkich modułów i sprawdzeniu prawidłowego ich umieszczenia w wyświetlaczu można przystąpić do sprawdzania działania przesyłania danych do modułu przez użycie interfejsu I²C. Moduły nie mają rezystorów podciągających linii SDA i SCL – takie rezystory musi mieć sterownik główny.

Jak już wspomniałem, do testów został użyty moduł ZL5PIC z mikrokontrolerem PIC18F458. Program testowy umożliwia zapisanie trzech dowolnych komunikatów o długości 16 znaków każdy. Komunikaty są przesyłane do sterownika przez łącze szeregowe RS232 z programu Hyper Terminal. Komunikat po odebraniu zostaje wyświetlony na wyświetlaczu LCD i zapisany w sterowniku w nielotnej pamięci EEPROM. Po zaprogramowaniu komunikatów są one wybierane po naciśnięciu jednego z przycisków SW0... SW2. Ponieważ wyświetlana informacja jest dłuższa niż 8 znaków, zastosowałem wyświetlanie z przewijaniem. Napisy są przesuwane od prawej strony do lewej.

Procedury używane w programie testowym pokazano na listingach 3...9. Ich analiza i zawarte w artykule informacje pozwalają na napisanie procedur obsługi wyświetlacza we własnych aplikacjach.

Tomasz Jabłoński, EP
tomasz.jablonski@ep.com.pl