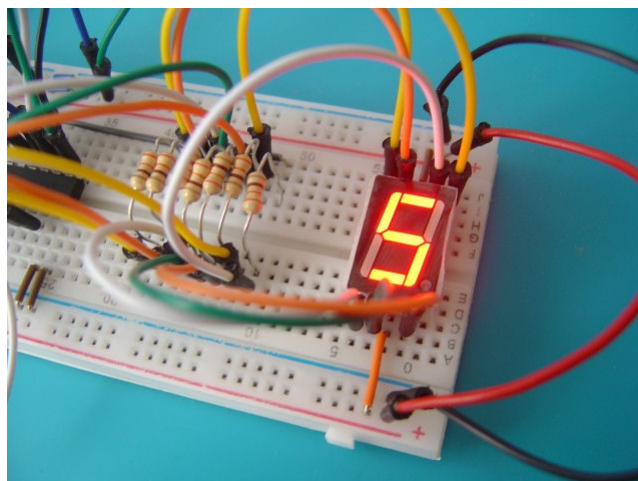


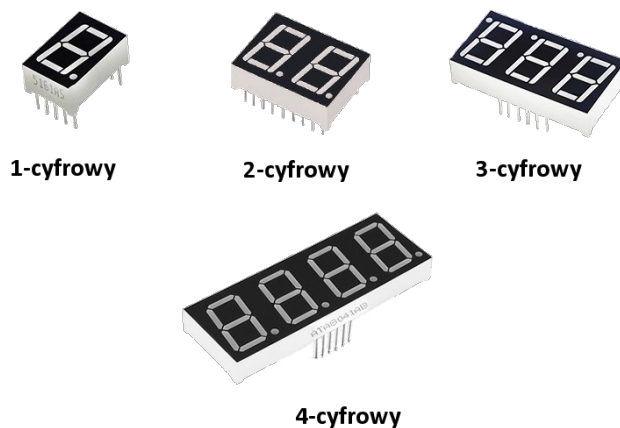
Arduino na miarę naszych możliwości potrzeb, cz. 4

wersja 12.05.2026



1. Wyświetlacze 7-segmentowe

Pozostając w szeroko rozumianej tematyce LED-ów czas na nieco bardziej ambitne zagadnienie. Jest nim obsługa 7-segmentowych wyświetlaczy LED¹. Czy ATtiny13 sprosta temu wyzwaniu?



Wyświetlacze 7-segmentowe są produkowane w rozmaitych odmianach różniących się barwą wyświetlanej cyfry (czerwona, niebieska, zielona) oraz rozmiarem (od około 1,42 cm do 16,51 cm). Poza tym dwa, trzy lub cztery wyświetlacze bywają łączone w zestawy umożliwiające wyświetlenie większej liczby cyfr.

Do potrzeb naszych eksperymentów wykorzystamy tylko wyświetlacze 1-cyfrowe oraz zaczniemy korzystać z dodatkowych układów scalonych. Potrzebować będziemy:

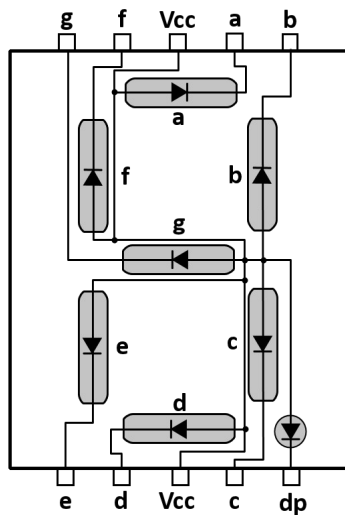
¹Nie licząc kropki (separator dziesiętny). Popularne wyświetlacze w kształcie cyfr z kropką mają w zasadzie 8 segmentów.

1 wyświetlacz 7-segmentowy ze wspólną anodą	około 3 PLN
3 wyświetlacze 7-segmentowy ze wspólną katodą	około 9 PLN
1 rejestr przesuwany 74HC595	od 2 PLN
Zestaw rezystorów 330 Ω	około 5 PLN
Rezystor 33 kΩ	około 1 PLN
Sterownik MAX7219	od 10 PLN

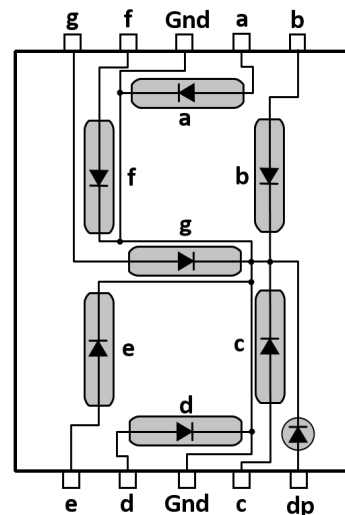
Cena jest orientacyjna, wg najtańszych ofert na Popularnym Portalu Aukcyjnym².

2. Zasada działania wyświetlacza

Pod względem technicznym wyświetlacze LED dzielą się na wyświetlacze ze wspólną anodą oraz wyświetlacze ze wspólną katodą. Każdy segment tworzący cyfrę jest osobną diodą (LED) przy czym dla ułatwienia sterowania jeden z biegunów wszystkich segmentów jest wspólny. W przypadku wyświetlaczy ze wspólną anodą wszystkie bieguny dodatnie diod są połączone z Vcc, natomiast w przypadku wyświetlaczy ze wspólną katodą bieguny ujemne diod połączone są z masą. Sterowanie segmentami odbywa się poprzez połączenie go z masą (wyświetlacze ze wspólną anodą) bądź przyłożenie napięcia (wyświetlacze ze wspólną katodą).



Wyświetlacz ze wspólną anodą



Wyświetlacz ze wspólną katodą

Przykładowym wyświetlaczem ze wspólną anodą jest FJ5161BH a ze wspólną katodą – 5611AH. Oba mają identyczne wymiary³ w zasadzie i różnią się tylko sposobem sterowania segmentami. Oba także nadają się do eksperymentu na płytce stykowej z uwagi na odpowiedni rozstaw nóżek.

Podobnie jak w przypadku każdego urządzenia składającego się z diod typu LED, należy pamiętać o podłączeniu rezystora ograniczającego prąd do każdego segmentu, gdyż jego pominięcie może skutkować uszkodzeniem diody. Warto skorzystać z noty katalogowej (ang. *datasheet*) producenta wyświetlacza, która powinna zawierać podstawowe parametry

²<http://www.allegro.pl/>

³ 19 × 12,6 × 8 mm.

diod (segmentów) oraz ich zachowanie w różnych warunkach. Wartość rezystancji można obliczyć korzystając ze wzoru $R = (V_{cc} - V_f) / I_f$, gdzie V_{cc} to napięcie zasilania, V_f – napięcie przewodzenia diody (zależne od typu i koloru diody), I_f – prąd przewodzenia (zwykle zależny od rozmiaru diody). W naszym przypadku napięcie zasilania wynosi 5 V, a napięcie i prąd przewodzenia mieszczą się w pewnych zakresach, jako wartości bezpieczne zarówno dla FJ5161BH jak i 5611AH przyjmując można odpowiednio $V_f = 1,7 - 1,8$ V i $I_f = 8 - 10$ mA⁴. Po podstawieniu do wzoru wartość rezystancji waha się od 320 do 412,5 Ω , a ponieważ nie można dobrać rezystora o dowolnej wartości lecz taki, który występuje w tzw. szeregu, to rezystor 330 Ω powinien być wystarczająco dobry. Jak wspomniano powyżej – trzeba przygotować tyle rezystorów ile segmentów (diod) będzie używanych, więc dla jednego wyświetlacza musi być ich 7 (lub 8, jeśli używany będzie wskaźnik separatora dziesiętnego). Nie można użyć jednego rezystora 330 Ω łącząc go równolegle z wszystkimi segmentami, gdyż wtedy jasność świecenia diod LED będzie zależna od ich ilości⁵.

3. Wyświetlacz ze wspólną anodą. Rejestr przesuwny.

Mając wiedzę teoretyczną możemy przystąpić do wykorzystania jej w praktyce. Na początek zajmijmy się wyświetlaczem ze wspólną anodą. Eksperyment będzie polegał na podłączeniu wyświetlacza do ATtiny13 i wysterowaniu go w taki sposób aby wyświetlał cyfry od 0 do 9 zmieniając wartość co pół sekundy.

Wydawałoby się, że możemy przystąpić do działania, ale pojawia się problem. ATtiny13 ma 8 pinów, z czego jeden to Vcc i jeden GND, więc zostaje 6 pinów I-O⁶ do wykorzystania. Nam aby wysterować wyświetlacz potrzeba 7 pinów (nie wykorzystujemy separatora dziesiętnego w wyświetlaczu). Brakuje więc jednego aby móc wyświetlić dowolną cyfrę. Czy jest na to jakaś rada?

Do rozszerzania liczby pinów mikrokontrolera służą np. rejestry przesuwne. Rejestr przesuwny to rodzaj bufora, przechowującego wprowadzone dane (bity, czyli stany wysokie lub niskie), które można następnie modyfikować (przesuwać) i odczytywać, ale w specyficzny sposób. Wyobraźmy sobie, że bufor ten jest 8-bitowym bajtem, do którego dane możemy wprowadzać szeregowo, bit po bicie aby móc ostatecznie odczytać jednocześnie (równolegle) wszystkie 8 bitów. Taki typ rejestru przesuwnego nazywa się *SIPO* (tj. z wejściem szeregowym i wyjściem równoległym)⁷ a co najważniejsze – w takim układzie do pracy potrzebujemy tylko 3 linie sterujące oraz zasilanie i masę zyskując 8 pinów wyjściowych⁸. Rejestrem przesuwным typu *SIPO* jest np. układ 74HC595.

Jak działa 74HC595? Dane należy szeregowo bit po bicie przekazywać na nóżkę 14 (SER), ale aby układ wiedział który stan jest właściwy towarzyszyć musi im sygnał zegarowy (SRCLK). Sygnał zegarowy jest nieaktywny w stanie niskim i aktywowany jest stanem wysokim. Jeśli chcemy przekazać logiczne 0 wówczas na wejście SER musimy podać stan

⁴ Wartości zalecane przez producentów są następujące: dla wyświetlacza 1,8 V i 10mA (5611AH) oraz 1,7 V i 8mA (FJ5161BH).

⁵ Teoretycznie można dobrać wartość rezystora w taki sposób aby w połączeniu równoległym na każdej diodzie LED dawał oporność 330 Ω .

⁶ Od ang. *Input-Output*, tj. wejścia-wyjścia.

⁷ Od ang. *Serial Input - Parallel Output*. Inne typy rejestrów przesuwnych to: *PIPO* - z wejściami i wyjściami równoległymi; *SISO* - z wyjściem i wejściem szeregowym; *PISO* - z wejściem równoległym i wyjściem szeregowym. Istnieją także rejestry przesuwne uniwersalne szeregowo-równoległe i równoległo-szeregowo.

⁸ Poświęcamy 3 linie zyskując 8, więc bilans to 5 dodatkowych linii.

niski a następnie aktywować SRCLK stanem wysokim. Ta operacja jednocześnie przesuwa bieżącą komórkę rejestru w prawo (tj od Q_A kolejno do Q_H , stąd rejestr przesuwany) i na pierwszym (Q_A) wyjściu przerzutnika pojawi się logiczne zero⁹. Jeśli następnie chcemy przekazać logiczne 1 to na wejściu SER ustawiamy stan wysoki oraz aktywujemy SRCLK. Wówczas logiczne zero z pierwszej operacji zostanie przesunięte na wyjście drugie (Q_B) a nasze logiczne 1 zostanie przekazane na wyjście pierwsze (Q_A). W ten sposób możemy przekazać 8 bitów szeregowo przesuując kolejne komórki i wypełniając bufor, jednak dane nie pojawią się od razu na pinach wyjściowych¹⁰. Aby je utrwalić należy użyć zatrzasku (RCLK) podając na nóżkę 12 stan wysoki. Układ posiada jeszcze ważne wejście – OE, które aktywuje możliwość korzystania z wyjść o ile na nóżce 13 jest stan niski. Zwykle łączymy go z masą aby wyjścia rejestru były aktywne na stałe. Z kolei pin 10 (SRCLR) służy do czyszczenia zawartości rejestru i aktywowany jest stanem niskim¹¹. Jeśli nie ma potrzeby czyszczenia zawartości rejestru podczas pracy można połączyć na stałe pin 10 z zasilaniem (stan wysoki).

74HC595		Pin	Symbol	Opis		
Q_B	1	16	V_{CC}	Wyjścia równoległe		
Q_C	2	15	Q_A			
Q_D	3	14	SER	9	Q_H'	Wyjście szeregowe (kolejny rejestr przesuwany)
Q_E	4	13	\overline{OE}	10	SRCLR	Reset (czyszczenie) rejestru
Q_F	5	12	RCLK	11	SRCLK	Sygnal zegarowy
Q_G	6	11	SRCLK	12	RCLK	Zatrask (<i>latch</i>)
Q_H	7	10	\overline{SRCLR}	13	OE	<i>Output Enable</i>
GND	8	9	Q_H'	14	SER	Wejście szeregowe
				16	V_{CC}	Zasilanie

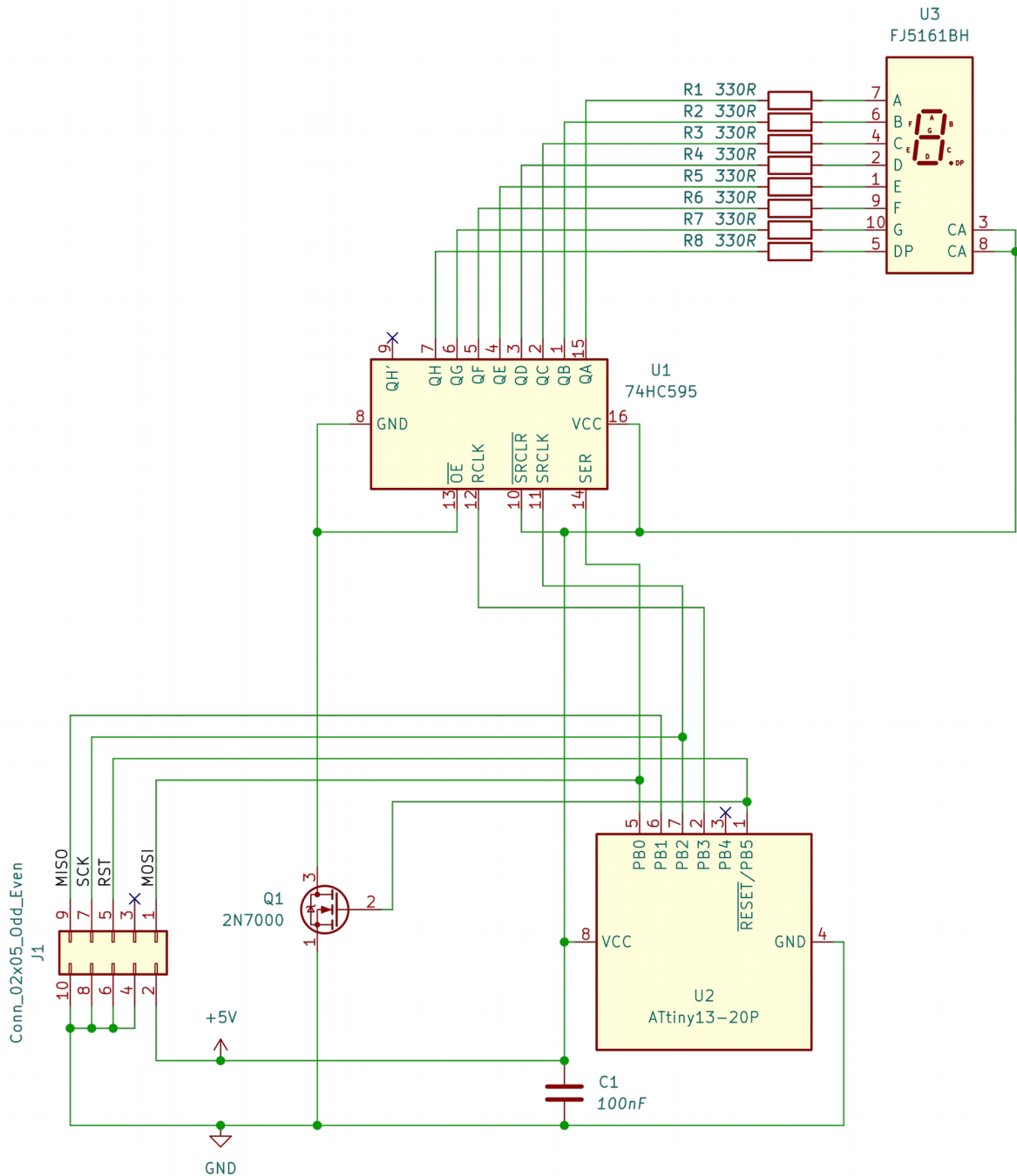
Połączmy zatem mikrokontroler z układem 74HC595 a ten z segmentami wyświetlacza. Możemy wykorzystać płytke stykową z większością elementów z poprzedniej części kursu, usuwając przycisk i diodę z rezystorem. Wejścia sterujące rejestrem możemy połączyć z dowolnymi dostępnymi pinami IO, my użyjemy PB0, PB2 i PB3 łącząc je kolejno z wejściami SER, SRCLK i RCLK¹². Następnie połączymy wyjścia równoległe rejestru z segmentami wyświetlacza, pamiętając o uwzględnieniu rezystorów 330 Ω w każdym połączeniu. Także tutaj kolejność połączeń jest dowolna, ale aby uniknąć komplikacji połączymy wyjścia Q_A - Q_H po kolei z segmentami A-DP wyświetlacza tak aby litery indeksowe wyjść równoległych i segmentów wyświetlacza były zgodne (segment DP traktujemy jako „H”). Ułatwi to nam znacząco programowanie mikrokontrolera. Schemat naszego układu może zatem wyglądać jak poniżej.

⁹ W praktyce układ 74HC595 zawiera w sobie dwie grupy przerzutników: pierwszą stanowiącą rejestr przesuwany, drugą stanowiącą wyjścia i umożliwiającą zapamiętanie ich stanu.

¹⁰ Będą tam wartości zbliżone do losowych, wynikające z obecności stanów przejściowych.

¹¹ Pozioma kreska nad symbolem oznacza, że dany sygnał aktywowany jest stanem niskim.

¹² Wybór pinów podyktowany jedynie wykorzystaniem konwencji nazewnictwa interfejsu SPI (por. opis SPI w rozdziale 4).



Budowa i mechanizm działania rejestru przesuwneego z 8 wyjściami niejako narzuca sposób programowania. Rejestr jest 8-bitowy (8 bitów danych szeregowo trafia na wyjścia równoległe), więc porcją naszych danych będzie bajt. Inaczej mówiąc każdą cyfrę widoczną na wyświetlaczu możemy zakodować w postaci 8 bitów, gdzie dany segment (bit) jest włączony lub wyłączony. W przypadku wyświetlacza jakiego używamy, tj. ze wspólną anodą aby segment został włączony musi na wejściu mieć stan niski (0), dla wyświetlacza ze wspólną katodą byłoby dokładnie odwrotnie, tj. wymagany byłby stan wysoki (1). W naszym programie segmenty A-H/DP będą odpowiadać potęgóm liczby 2 w kolejności rosnącej (od 0 do 7)¹³, co pozwoli na zakodowanie danej cyfry w postaci liczby w zakresie 0-255.

¹³ Oczywiście można zastosować kolejność malejącą (równie dobre rozwiązanie) albo dowolnie inną (wiązałoby się to z pewnymi komplikacjami w kodzie).

Cyfra	A $2^0=1$	B $2^1=2$	C $2^2=4$	D $2^3=8$	E $2^4=16$	F $2^5=32$	G $2^6=64$	H/DP $2^7=128$	Wsp. anoda (x=0; -=1)	Wsp. katoda (x=1; -=0)
0	x	x	x	x	x	x	-	-	192 (\$C0)	63 (\$3F)
1	-	x	x	-	-	-	-	-	249 (\$F9)	6 (\$06)
2	x	x	-	x	x	-	x	-	164 (\$A4)	91 (\$5B)
3	x	x	x	x	-	-	x	-	176 (\$B0)	79 (\$4F)
4	-	x	x	-	-	x	x	-	153 (\$99)	102 (\$66)
5	x	-	x	x	-	x	x	-	146 (\$92)	109 (\$6D)
6	x	-	x	x	x	x	x	-	130 (\$82)	125 (\$7D)
7	x	x	x	-	-	-	-	-	248 (\$F8)	7 (\$07)
8	x	x	x	x	x	x	x	-	128 (\$80)	127 (\$7F)
9	x	x	x	x	-	x	x	-	144 (\$90)	111 (\$6F)

Mając do dyspozycji powyższą tabelę możemy przystąpić do pisania kodu programu. W sekcji *uses* dodamy moduł *attiny13_basics* przygotowany w poprzednim odcinku kursu, aby skorzystać z procedury opóźniającej *DelayMs*. Następnie zdefiniujemy stałe PB0, PB2 i PB3 odpowiadające pinom portu PB mikrokontrolera. W sekcji *const* dodamy jeszcze dwie ważne tablice: *NumberMasks* – dziesięcioelementową tablicę (*array*) zakodowanych bajtów odpowiadających cyfram 0-9 oraz *PowersOfTwo* - ośmioelementową tablicę potęg liczby 2, ułatwiającą odkodowanie bajtów do postaci bitów.

Na początku programu zdefiniujemy piny portu PB jako wyjścia. Ponieważ zdanie polega na kolejnym wyświetlaniu cyfr i ciągłym powtarzaniu tej czynności, zastosujemy pętlę nieskończoną a w niej w kolejnej pętli iteracyjnej wykorzystamy zmienną *num* nadając jej wartości w zakresie od 0 do 9. W każdym przebiegu pętli iteracyjnej najpierw ustawimy stan niski na pinie PB3 (połączonym z RCLK rejestru) powodując zwolnienie zatrzaśniętych danych. Następnie w kolejnej, zagnieżdżonej pętli wykorzystamy zmienną *i* oraz tablicę *PowersOfTwo* do szeregowego przekazania bitów na pin PB0 (połączony z SER rejestru). Jest to kluczowa część programu. Zakodowaną wartość segmentów danej cyfry pobraną z tablicy *NumberMasks* odkodujemy sprawdzając czy kolejne bity są zapalone czy zgaszone. Ustalimy to używając operatora logicznego *and* (koniunkcji) z kolejnymi potęgami liczby 2. Kolejność potęg musi być malejąca ze względu na każdorazowe przesunięcia bieżącej komórki dokonywane przez rejestr. Bity zapalone należy przekazać włączając pin PB0 (SER) a zgaszone - wyłączając ten pin. Następnie poinformujemy rejestr, że wartość bieżącej komórki została już ustalona i trzeba ją przesunąć. Zrobimy to ustawiając stan wysoki i zaraz później stan niski na pinie PB2 (SRCLK). Po przekazaniu 8 bitów możemy zatrzasnąć dane ustawiając stan wysoki na pinie PB3 (RCLK). Dzięki temu na wyjściach rejestru pojawią się odpowiednie sygnały i zaświecą żądane segmenty wyświetlacza. Na koniec wykorzystamy procedurę *DelayMs* do programowego opóźnienia o pół sekundy tak aby zmiany cyfr na wyświetlaczu były czytelne¹⁴.

¹⁴ Wybrałem wartość 500 milisekund, która gwarantuje jeszcze poprawne działanie procedury *DelayMs*. Przy wyższych wartościach może dojść do „przekręcenia” licznika pętli opóźniającej i nieprawidłowego opóźnienia.

```

program display_7segca;

{$IFDEF attiny13}
  {$Fatal Invalid controller type, expected: attiny13}
{$ENDIF}

uses
  attiny13_basics;

const
  PB0 = %00000001; // pin PB0 = nóżka 5 mikrokontrolera -> SER
  PB2 = %00000100; // pin PB2 = nóżka 7 mikrokontrolera -> SRCLK (CLOCK)
  PB3 = %00001000; // pin PB3 = nóżka 2 mikrokontrolera -> RCLK (LATCH)

  //maski cyfr 0-9
  NumberMasks: array[0..9] of UInt8 = ($C0, $F9, $A4, $B0, $99, $92, $82,
  $F8, $80, $90);
  //potęgi liczby 2, kolejność malejąca ze względu na przesunięcia
  PowersOfTwo: array[0..7] of UInt8 = (128, 64, 32, 16, 8, 4, 2, 1);

var
  num, i: UInt8;
begin
  //rejestr kierunków
  DDRB := DDRB or PB0; // PB0 jako wyjście
  DDRB := DDRB or PB2; // PB2 jako wyjście
  DDRB := DDRB or PB3; // PB3 jako wyjście

  while true do // pętla nieskończona
  begin
    for num := 0 to 9 do
    begin
      PORTB := PORTB and not PB3; // stan niski na LATCH

      //szeregowa transmisja danych bit po bicie
      for i := 0 to 7 do
      begin
        if (NumberMasks[num] and PowersOfTwo[i]) = PowersOfTwo[i] then
          PORTB := PORTB or PB0 //stan wysoki na SER
        else
          PORTB := PORTB and not PB0; //stan niski na SER

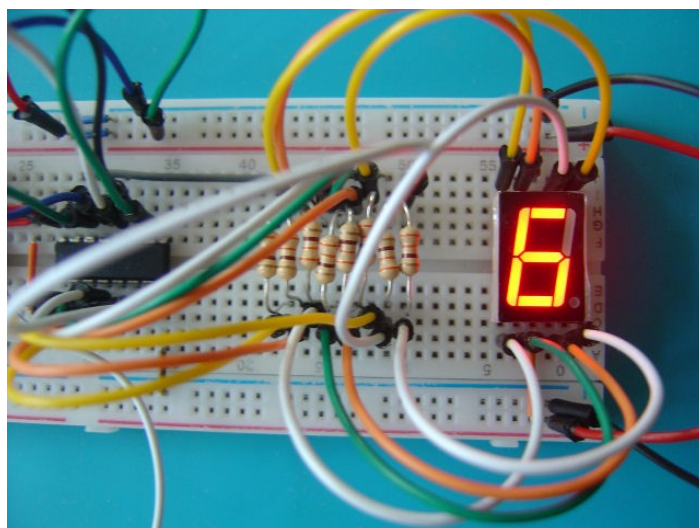
          PORTB := PORTB or PB2; // stan wysoki na CLOCK
          PORTB := PORTB and not PB2; // stan niski na CLOCK
        end;

        PORTB := PORTB or PB3; // stan wysoki na LATCH -
        zatrzaśnięcie danych

        DelayMs(500); // opóźnienie
      end;
    end;
  end.

```

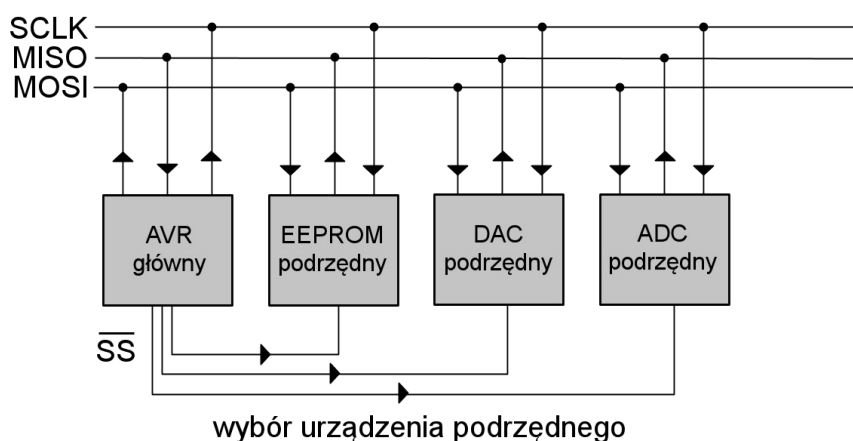
Po skompilowaniu powyższego kodu i przesłaniu go do pamięci flash mikrokontrolera układ powinien zadziałać zgodnie z oczekiwaniem. Pewną ułomnością estetyczną eksperymentu jest plątana przewodów na płytce stykowej, czego jednak trudno uniknąć.



4. Interfejs SPI.

Kontynuując tematykę transmisji szeregowej zastanówmy się czy możemy jej implementację uprościć bądź usprawnić. Istnieje sporo sposobów szeregowej wymiany danych między urządzeniami. Szeroko stosowanym standardem jest SPI, uznawany za względnie prosty w implementacji i szybki. SPI (ang. *Serial Peripheral Interface*, interfejs szeregowy urządzeń peryferyjnych) wykorzystywany jest do łączenia urządzenia głównego, którym zwykle jest mikrokontroler z urządzeniami peryferyjnymi, takimi jak przetworniki analogowo-cyfrowe, pamięci EEPROM, pamięci flash, karty SD i in.

Interfejs SPI wymaga obecności urządzenia głównego (ang. *master*, zwykle jest nim mikrokontroler) oraz minimum jednego urządzenia podrzędnego (ang. *slave*, zwykle jest nim jakieś urządzenie peryferyjne). Do komunikacji wykorzystuje cztery linie: MOSI, MISO, SCLK i SS. Linia MOSI (ang. *Master Output Slave Input*) służy do wysyłania danych z urządzenia głównego do podrzędnego. Linia MISO (ang. *Master Input Slave Output*) odpowiada za transmisje danych w kierunku odwrotnym – od urządzenia podrzędnego do głównego. Linia SCLK (ang. *Serial Clock*)¹⁵ jest sygnałem zegarowym (taktującym) pozwalającym na poprawny odbiór i wysyłanie danych między urządzeniami. Z kolei linia SS (ang. *Slave Select*)¹⁶ służy do wyboru aktywnego w danej chwili urządzenia peryferyjnego.

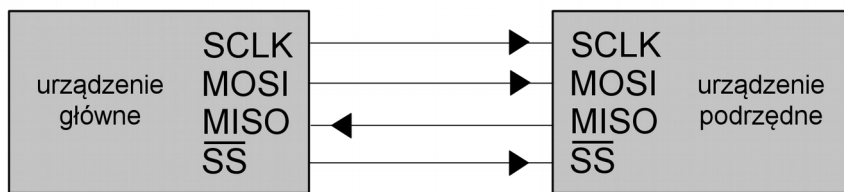


¹⁵Inny skrót to SCK.

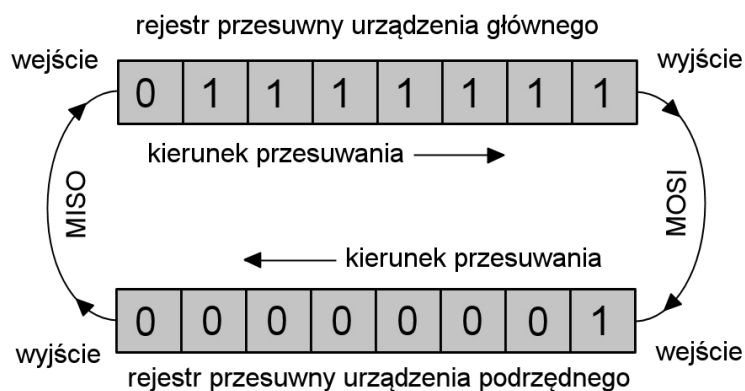
¹⁶Inna nazwa to CS (ang. *Chip Select*).

W połączeniach z wieloma urządzeniami stosuje się tzw. magistrale – zespół linii przenoszących dane do urządzeń dołączonych do magistrali. W standardzie SPI formę magistrali mają linie MOSI, MISO i SCLK i do nich podłączone są urządzenia główne i urządzenia podrzędne. Z kolei urządzenie główne musi posiadać tyle linii SS z iloma urządzeniami podrzędnymi jest połączone¹⁷.

Przyjrzyjmy się teraz kierunkom transmisji w liniach magistrali SPI oraz urządzeniach do niej podłączonych. Urządzenie główne wysyła dane szeregowo na linię MOSI, natomiast urządzenia podrzędne mogą tylko odbierać dane z tej linii. Natomiast urządzenia podrzędne wysyłają dane szeregowo na linię MISO, podczas gdy urządzenie główne może dane z tej linii tylko odbierać. Oznacza to, że na linii MOSI napięcia kontroluje urządzenie główne, natomiast na linii MISO dane urządzenie podrzędne. Dla wszystkich urządzeń istnieje wspólna linia zegarowa SCLK, której sygnał generuje urządzenie główne. Zwykle taktowanie odbywa się przy narastającym zboczu sygnału (tj. zmianie ze stanu niskiego na wysoki), ale istnieją tryby komunikacji SPI, gdzie taktowanie sygnalizuje zbocze opadające. Ponadto urządzenie główne decyduje z którym urządzeniem podrzędnym chce się aktualnie komunikować. Aktywowanie takiej komunikacji odbywa się przez zmianę stanu jego linii SS z wysokiego na niski a jej zakończenie przez powrót do stanu wysokiego.



W najprostszym układzie wymiana danych odbywa się między tylko dwoma urządzeniami: głównym i podrzędnym. Urządzenie główne aktywuje urządzenie podrzędne, wystawia sygnał zegarowy oraz wysyła dane na linię MOSI. Urządzenie podrzędne z kolei wysyła dane na linię MISO. Wysyłanie i odbieranie danych wykonywane jest jednocześnie, tzn. każde z urządzeń wysyła i odbiera dane¹⁸.



¹⁷ Ten fakt ogranicza liczbę urządzeń podrzędnymi do fizycznych możliwości (dostępnych pinów) urządzenia głównego.

¹⁸ Nie każde dane w komunikacji SPI mają znaczenie. Np. urządzenie podrzędne w oczekiwaniu na komendę (tj. jakiś ustalony ciąg bitów) wysyła w tym czasie dane, które należy zignorować. Zwykle takie dane składają się z samych „zer” (stanów niskich).

Urządzenia obsługujące sprzętowo protokół SPI mają dedykowany uniwersalny rejestr przesuwany ułatwiający komunikację, połączony z liniami MOSI i MISO. Wraz z sygnałem zegarowym bit wysyłany przez urządzenie główne pojawi się w ostatniej komórce rejestru urządzenia podrzędnego i jednocześnie do ostatniej komórki rejestru urządzenia głównego trafi bit wysyłany przez urządzenie podrzędne. Całość można sobie wyobrazić jako swoisty pierścień wymiany danych. Jeśli w urządzeniu głównym rejestr pierwotnie zawiera same jedynki a w urządzeniu podrzędnym same zera to po sygnale zegarowym dane przesuną się o jedną komórkę w rejestrach obu urządzeń tak, że na koniec szeregu bitów urządzenia podrzędnego trafi jedynka, a do rejestru urządzenia głównego – zero.

Wiele mikrokontrolerów AVR posiada sprzętową obsługę SPI w tym odpowiednie rejestry, jednak w przypadku ATtiny13 sprawa jest bardziej skomplikowana. Co prawda ATtiny13 ma dedykowane piny portu PB służące do szeregowej transmisji danych (MOSI – PB0, MISO – PB1 i SCLK – PB2, funkcję SS pełni pin RESET/PB5¹⁹), są one obsługiwane sprzętowo jedynie na potrzeby programowania pamięci flash, w trybie urządzenia podrzędnego, służąc komunikacji programatora z mikrokontrolerem. ATtiny13 nie posiada dostępnych do celów ogólnych rejestrów SPI (np. SPDR - rejestru danych). W tej sytuacji jedynym rozwiązaniem jest programowa obsługa komunikacji SPI.

Można by pomyśleć, że cała dygresja na temat interfejsu SPI nie miała sensu, ale ta wiedza przyda się nam niebawem. Istnieją bowiem układy sterujące wyświetlaczami 7-segmentowymi, korzystające z protokołu SPI.

5. Sterownik wyświetlaczy ze wspólną katodą.

Jednym z układów przeznaczonych do obsługi wyświetlaczy 7-segmentowych oraz matryc LED jest MAX7219. Jest to sterownik (ang. *driver*) kompatybilny z protokołem SPI. Podobnie jak w przypadku SPI, do pracy poza liniami zasilania i masy układ wymaga linii danych szeregowych (DIN), sygnału zegarowego (CLK). Pewną różnicą jest linia LOAD, która w przeciwieństwie do SS w standardzie SPI nie dostarcza sygnału początku i końca transmisji szeregowej lecz służy jako zatrząsk w wewnętrznym 16-bitowym rejestrze przesuwany wbudowanym w MAX7219.

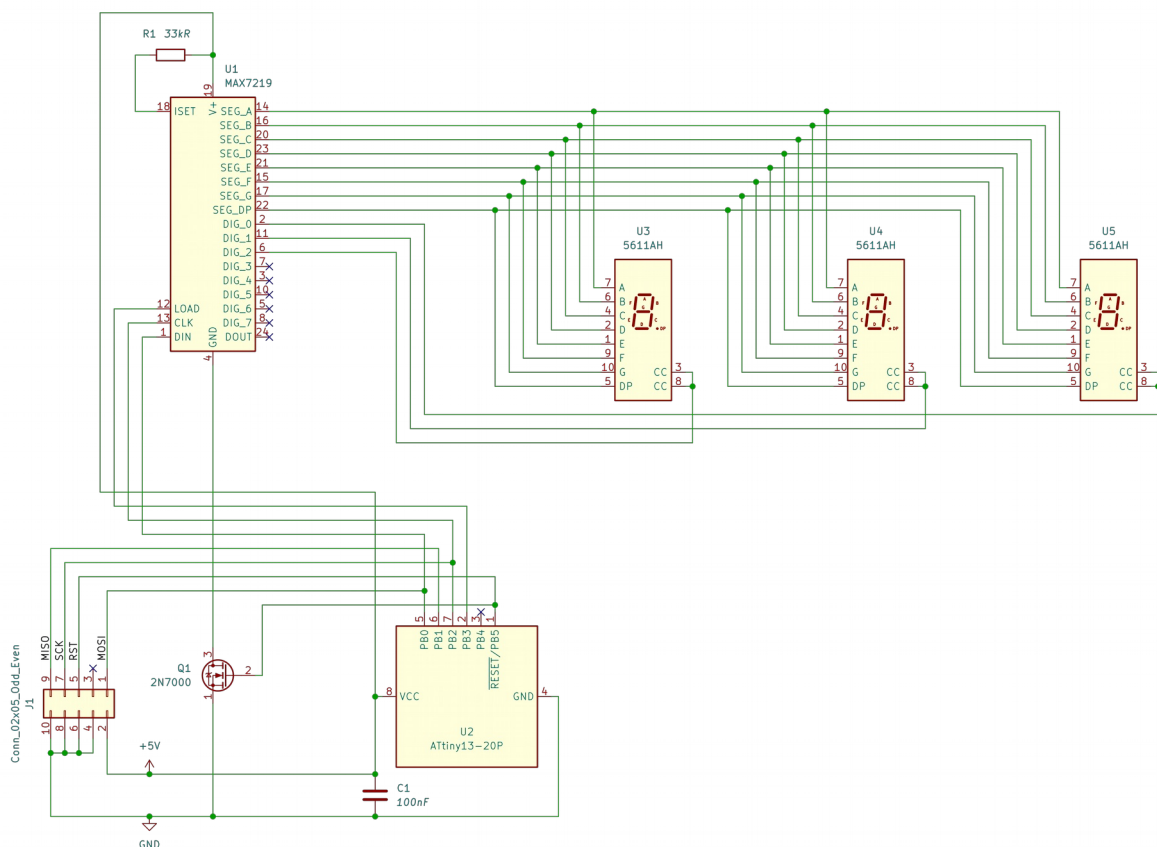
MAX7219 posiada 8 wyjściowych linii równoległych SEG A-G oraz SEG DP obsługujących odpowiednie segmenty wyświetlacza (bądź diody matrycy LED, w zależności od potrzeb) oraz 8 linii sterujących DIG. Oznacza to, że do układu można podłączyć maksymalnie 8 wyświetlaczy 7-segmentowych. Ponadto posiada wyjście rezystora ograniczającego (ISET) służącego do sterowania jasnością podłączonych wyświetlaczy oraz wyjście szeregowo DOUT umożliwiające podłączenie kolejnych układów MAX7219 w przypadku bardziej rozbudowanych projektów.

Arytmetyka podpowiada, że aby układ mógł obsługiwać osobno 8 wyświetlaczy składających się z 8 segmentów (licząc separator dziesiętny) powinien zarezerwować do tego celu 64 linie. MAX7219 działa jednak inaczej. Korzystając ze zjawiska bezwładności ludzkiego wzroku sekwencyjnie przełącza podłączone wyświetlacze wyświetlające daną cyfrę robiąc to na tyle szybko, że całość widzimy to jako stabilny obraz. Dlatego potrzebuje tylko po jednej linii włączającej i wyłączającej dany wyświetlacz (DIG) a linie obsługujące segmenty (SEG) są wspólne.

¹⁹ Z tego powodu podczas programowania mikrokontrolera pin RESET jest w stanie niskim.

MAX7219		Pin	Symbol	Opis
DIN	1	24	DOUT	Wyjście szeregowe
DIG 0	2	23	SEG D	Linie sterujące wyświetlaczami 0-7
DIG 4	3	22	SEG DP	
GND	4	21	SEG E	
DIG 6	5	20	SEG C	
DIG 2	6	19	V+	Masa
DIG 3	7	18	ISSET	Zatrząsk (<i>latch</i>)
DIG 7	8	17	SEG G	Sygnał zegarowy
GND	9	16	SEG B	Wyjścia równoległe segmentów wyświetlaczy
DIG 5	10	15	SEG F	
DIG 1	11	14	SEG A	Wyjście rezystora ograniczającego
LOAD	12	13	CLK	Zasilanie
		19	V+	Zasilanie
		24	DOUT	Wyjście szeregowe

W nowym eksperymencie wykorzystamy układ MAX7219 do sterowania trzema wyświetlaczami ze wspólną katodą. Katody wyświetlaczy podłączymy pod wyjścia DIG 0 - DIG 2 natomiast segmenty podłączymy do wspólnej linii SEG A - SEG DP. Wybór aktywnego wyświetlacza odbywać się będzie poprzez połączenie jego katody z linią masy natomiast sterowanie segmentami poprzez stany wysokie na wyjściach SEG.



Pewnego namysłu wymaga dobór rezystora ograniczającego prąd wyświetlaczy (ISET). Jego wartość zależy od natężenia prądu jaki może przepływać przez segmenty wyświetlacza. Wg noty katalogowej wyświetlacza 5611AH bezpieczną wartością będzie zakres między 10 a 20 mA. Dokumentacja MAX7219 z kolei określa prąd segmentu jako około 100 razy większy od prądu płynącego przez rezystor ISET. Więc przyjmując napięcie wejściowe 5 V, wartość rezystora powinna mieścić się między 25 (dla 20 mA) a 50 (dla 10 mA) kΩ²⁰. Wydaje się, że bezpiecznym rozwiązaniem będzie użycie rezystora 33 kΩ.

Przyjrzyjmy się teraz protokołowi komunikacji z MAX7219. Linię danych szeregowych DIN połączymy z pinem PB0 (MOSI), linię sygnału zegarowego CLK z pinem PB2 a linię LOAD z pinem PB3 mikrokontrolera²¹. Nie potrzebujemy linii transmisji od sterownika do mikrokontrolera (MISO), gdyż sterownik nie wysyła żadnych danych przez SPI. Jak wspomniano wcześniej MAX7219 ma wbudowany 16-bitowy rejestr danych (D0-D15), za pomocą którego można przesyłać polecenia sterownikowi. Posiada również 14 innych rejestrów służących do konfiguracji i sterowania wyświetlaczami.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
*	*	*	*	Adres				Dane							

Wartości bitów oznaczonych gwiazdką (*) nie mają znaczenia.

Budowa rejestru danych implikuje format danych wysyłanych do układu. Aby wysłać prawidłowo polecenia należy więc użyć dwóch bajtów. W pierwszym bajcie powinien znaleźć się adres rejestru konfiguracyjnego lub sterującego, w drugim dane przeznaczone dla tego rejestru, przy czym cztery bardziej znaczące bity adresu nie mają znaczenia i ich wartości mogą być dowolne²². Przyjrzyjmy się funkcjom rejestrów konfiguracyjnych i sterujących układu.

Rejestr	Adres					Wartość
	D15-D12	D11	D10	D9	D8	
No-Op (No-Operation)	0*	0	0	0	0	0 (\$00)
Digit0	0*	0	0	0	1	1 (\$01)
Digit1	0*	0	0	1	0	2 (\$02)
Digit2	0*	0	0	1	1	3 (\$03)
Digit3	0*	0	1	0	0	4 (\$04)
Digit4	0*	0	1	0	1	5 (\$05)
Digit5	0*	0	1	1	0	6 (\$06)
Digit6	0*	0	1	1	1	7 (\$07)
Digit7	0*	1	0	0	0	8 (\$08)
Decode Mode	0*	1	0	0	1	9 (\$09)

²⁰ Wartości obliczone wg wzoru $R_{ISET} \approx (100 \times V_{CC}) / I_{SEG}$, gdzie R_{ISET} to wymagana rezystancja rezystora ISET, V_{CC} – napięcie zasilania, I_{SEG} – dopuszczalny prąd segmentu.

²¹ Oczywiście można wybrać inne wolne piny ATtiny ale taki dobór ułatwi nam wykorzystanie części kodu z poprzedniego eksperymentu.

²² Wynika to z liczby obsługiwanych rejestrów (14), których zakodowane binarnie adresy zajmują jedynie 4 bity i pozostałe 4 bity są wolne.

Intensity	0*	1	0	1	0	10 (\$0A)
Scan Limit	0*	1	0	1	1	11 (\$0B)
Shutdown	0*	1	1	0	0	12 (\$0C)
Display Test	0*	1	1	1	1	15 (\$0F)

Bit y oznaczone gwiazdką (*) nie mają znaczenia, dla uproszczenia kalkulacji nadałem im tutaj wartość 0.

Rejestr *No-Op* (adres \$00) nie powoduje wykonania żadnej operacji, lecz jest wykorzystywany przy kaskadowym połączeniu wielu układów MAX7219. Wówczas pin DOUT służy do transmisji danych do wejścia DIN kolejnego układu. Ponieważ wykorzystujemy tylko jeden układ MAX7219 nie będziemy używać tego rejestru.

Rejestry *Digit0 - Digit7* (adresy \$01-\$08) odpowiadają za wybór wyświetlacza. Wówczas dane (D0-D7) wskazują na to które segmenty tego wyświetlacza mają być zapalone. Informacja o tym może być przekazana bezpośrednio w formie zakodowanego bajta, bądź jako liczby w formacie BCD²³. My użyjemy formatu wykorzystującego cały bajt wskazując bezpośrednio które segmenty mają zostać zapalone.

Rejestr *Decode Mode* (adres \$09) decyduje o sposobie kodowania segmentów wyświetlaczy. Jeśli wybrany zostaje ten rejestr to bity danych (D0-D7) określają czy liczby wyświetlane jako cyfry na danym wyświetlaczu mają być kodowane w formacie BCD (bit ustawiony, tj. 1) czy jako zakodowany bajt określający zapalone segmenty (bit nieustawiony, tj. 0).

Rejestr *Intensity* (adres \$0A) steruje intensywnością świecenia segmentów LED w zakresie od 0 do 15 (\$00-\$0F). Maksymalna dopuszczalna wartość liczbowa spowoduje, iż segmenty będą świecić z maksymalną intensywnością i będzie przez nie przepływał największy prąd, jednak nie większy od ustalonego przez wartość rezystora podłączonego do ISET.

Rejestr *Scan Limit* (adres \$0B) steruje liczbą podłączonych wyświetlaczy do układu, należy więc w bajcie danych podać jego wartość w zakresie 0-7. Pamiętaj, że w danym momencie tylko jeden wyświetlacz jest włączony, częstotliwość przełączania (multiplexowania) wyświetlaczy wynosi 800 Hz, jeśli podłączono ich osiem. Przy mniejszej liczbie wyświetlaczy częstotliwość wzrasta²⁴. Oznacza to także, iż wraz ze zmniejszaniem liczby podłączonych wyświetlaczy intensywność świecenia ich segmentów będzie wzrastać.

Rejestr *Shutdown* (adres \$0C) określa czy układ ma pracować w trybie normalnym (\$01 w rejestrze danych) czy w trybie wstrzymania (\$00 w rejestrze danych). W trybie wstrzymania wyświetlacze zostają zgaszone, wstrzymana zostaje praca wewnętrznego oscylatora układu, linie segmentów zostają podciągnięte do masy a linie sterujące zostają podciągnięte do plusa zasilania. W ten sposób układ zostaje wygaszony a pobór prądu zredukowany do około 150 μ A.

²³ BCD – ang. *binary-coded decimal*, zapis dziesiętny kodowany binarnie polegający na zakodowaniu kolejnych cyfr dziesiętnych danej liczby w systemie dwójkowym, przy użyciu czterech bitów (tj. połowy bajtu).

²⁴ Można ją obliczyć wg wzoru $8 * f_{osc} / N$, gdzie f_{osc} to częstotliwość typowa przy podłączonych 8 wyświetlaczach, N to liczba podłączonych wyświetlaczy.

Rejestr Display Test (adres \$0F) pozwala na włączenie trybu testu przez wpisanie do rejestru danych wartości \$01 i powoduje zapalenie wszystkich segmentów z największą intensywnością. Nie zmienia zawartości innych rejestrów.

Mając już wiedzę o sposobie konfiguracji sterownika MAX7219 możemy przystąpić do pisania kodu. Kluczową kwestią jest transmisja SPI, której ATtiny nie wspiera sprzętowo. Jak ją zaprogramować? Dobra wiadomość jest taka, że prawie to już zrobiliśmy, ponieważ software'owa obsługa SPI niewiele różni się od programowania rejestru przesuwanego. Obsługa linii danych i sygnału zegarowego będzie identyczna, pewna różnica wystąpi w wywołaniu „zatrasku”. Ponieważ rejestr przesuwany MAX7219 jest 16-bitowy, dane należy zatrzasnąć po wysłaniu drugiego bajtu.

Jest jeszcze jedna istotna różnica w porównaniu do kodu, który przygotowaliśmy dla rejestru przesuwanego. Chodzi o kodowanie włączonych segmentów do postaci bajtowej. MAX7219 wymaga aby wartość bitowa (potęga liczby 2) malała dla kolejnych segmentów licząc od A do G danej cyfry, czyli odwrotnie niż zrobiliśmy dla rejestru przesuwanego. Musimy więc przygotować nową tabelę kodowania cyfr.

Cyfra	H/DP $2^7=128$	A $2^6=64$	B $2^5=32$	C $2^4=16$	D $2^3=8$	E $2^2=4$	F $2^1=2$	G $2^0=1$	Wsp. katoda (x=1; -=0)
0	-	x	x	x	x	x	x	-	126 (\$7E)
1	-	-	x	x	-	-	-	-	48 (\$30)
2	-	x	x	-	x	x	-	x	109 (\$6D)
3	-	x	x	x	x	-	-	x	121 (\$79)
4	-	-	x	x	-	-	x	x	51 (\$33)
5	-	x	-	x	x	-	x	x	91 (\$5B)
6	-	x	-	x	x	x	x	x	95 (\$5F)
7	-	x	x	x	-	-	-	-	112 (\$70)
8	-	x	x	x	x	x	x	x	127 (\$7F)
9	-	x	x	x	x	-	x	x	123 (\$7B)

W naszym kodzie wykorzystamy definicję pinów przygotowaną dla rejestru przesuwanego, zdefiniujemy też kilka nowych stałych (REG_TEST, REG_SCAN_LIMIT, REG_DECODE, REG_SHUTDOWN, REG_INTENSITY) przydatnych do konfiguracji sterownika oraz zaktualizujemy tablicę *NumberMasks* o nowe wartości. Ponieważ dane szeregowo będziemy wysyłać wielokrotnie, wydzielimy do tego celu procedurę *Max7219_SendData*, w której wykorzystamy kod (z drobnymi modyfikacjami) przygotowany dla rejestru przesuwanego. Procedura ta będzie odpowiedzialna za wysyłanie dwóch bajtów: konfiguracyjnego (adres) oraz danych do MAX7219.

W głównej części programu pozostawimy definicje pinów portu PB. Następnie zajmiemy się konfiguracją sterownika. Aby mieć pewność, że wszystkie połączenia wykonaliśmy poprawnie wykonamy test naszego urządzenia, który powinien skutkować zaświeceniem wszystkich segmentów trzech wyświetlaczy przez około pół sekundy²⁵. W dalszej kolejności określimy liczbę używanych wyświetlaczy (3), tryb dekodowania cyfr

²⁵ Za wyjątkiem separatorów dziesiętnych, które pozostawiliśmy niepodłączone.

(0, tj. manualny), ustawimy maksymalną jasność świecenia segmentów oraz wyczyścimy dane podłączonych wyświetlaczy. Operacje te wykonamy w trybie wstrzymania pracy (*shutdown*) aby ewentualne przypadkowe wartości nie były wyświetlane w czasie konfiguracji naszego urządzenia.

Główny kod wstawimy do pętli nieskończonej i polegać będzie na wyświetlaniu kolejnych liczb z zakresu od 0 do 999 co około 100 milisekund na naszych wyświetlaczach. Aby określić prawidłową wartość jedności, dziesiątek i setek danej liczby skorzystamy ze zmiennej tymczasowej *tmp*, którą będziemy dzielić całkowicie (*div*) na dziesięć a resztę z dzielenia (*mod*) przypisywać do kolejnych cyfr. Warto zauważyć, że zmienna *tmp* jest typu *UInt16* aby zmieścić się w wymaganym zakresie liczbowym.

```

program display_7segcc;

{$IFDEF attiny13}
  {$Fatal Invalid controller type, expected: attiny13}
{$ENDIF}

uses
  attiny13_basics;

const
  PB0 = %00000001; // pin PB0 = nóżka 5 mikrokontrolera -> DIN
  PB2 = %00000100; // pin PB2 = nóżka 7 mikrokontrolera -> CLK
  PB3 = %00001000; // pin PB3 = nóżka 2 mikrokontrolera -> LOAD

  REG_TEST      = $0F;
  REG_SCAN_LIMIT = $0B;
  REG_DECODE     = $09;
  REG_SHUTDOWN  = $0C;
  REG_INTENSITY = $0A;

  INTENSITY_MAX = $0F;

  //maski cyfr 0-9
  NumberMasks: array[0..9] of UInt8 = ($7E, $30, $6D, $79, $33, $5B, $5F,
  $70, $7F, $7B);
  //potęgi liczby 2, kolejność malejącą ze względu na przesunięcia
  PowersOfTwo: array[0..7] of UInt8 = (128, 64, 32, 16, 8, 4, 2, 1);

procedure Max7219_SendData(const aCtrl, aData: UInt8);
var
  i: UInt8;
begin
  PORTB := PORTB and not PB3; // stan niski na LOAD

  //bajt kontrolny - szeregowo transmisja danych bit po bicie
  for i:=0 to 7 do
  begin
    if (aCtrl and PowersOfTwo[i]) = PowersOfTwo[i] then
      PORTB := PORTB or PB0 // stan wysoki na DIN
    else
      PORTB := PORTB and not PB0; // stan niski na DIN

    PORTB := PORTB and not PB2; // stan niski na CLK
    PORTB := PORTB or PB2; // stan wysoki na CLK
  end;

```

```

//bajt danych - szeregowo transmisja danych bit po bicie
for i:=0 to 7 do
begin
  if (aData and PowersOfTwo[i]) = PowersOfTwo[i] then
    PORTB := PORTB or PB0 // stan wysoki na DIN
  else
    PORTB := PORTB and not PB0; // stan niski na DIN

    PORTB := PORTB and not PB2; // stan niski na CLK
    PORTB := PORTB or PB2; // stan wysoki na CLK
end;

PORTB := PORTB or PB3; // stan wysoki na LOAD - zatrzaśnięcie
danych

DelayMs(10);
end;

var
  i, dig: UInt8;
  num, tmp: UInt16;
begin
  //rejestr kierunków
  DDRB := DDRB or PB0; // PB0 jako wyjście
  DDRB := DDRB or PB2; // PB2 jako wyjście
  DDRB := DDRB or PB3; // PB3 jako wyjście

  PORTB := PORTB or PB3; // stan wysoki na LOAD - zatrzaśnięcie
  danych

  //test wyświetlaczy
  Max7219_SendData(REG_SHUTDOWN, 1); // tryb normalny pracy
wyświetlacza
  Max7219_SendData(REG_TEST, 1); // włączenie testu
  DelayMs(500);
  Max7219_SendData(REG_TEST, 0); // wyłączenie testu
  Max7219_SendData(REG_SHUTDOWN, 0); // tryb wstrzymania pracy
wyświetlacza

  //konfiguracja wyświetlaczy
  Max7219_SendData(REG_SCAN_LIMIT, 2); // liczba wyświetlaczy: 3
  Max7219_SendData(REG_DECODE, 0); // tryb decode manualny
  Max7219_SendData(REG_INTENSITY, INTENSITY_MAX); // maksymalna jasność

  //czyszczenie danych
  for i:=0 to 2 do
    Max7219_SendData(i + 1, 0);

  Max7219_SendData(REG_SHUTDOWN, 1); // tryb normalny pracy
wyświetlacza
  DelayMs(500);

  while true do // pętla nieskończona
  begin
    for num := 0 to 999 do
    begin
      tmp := num;

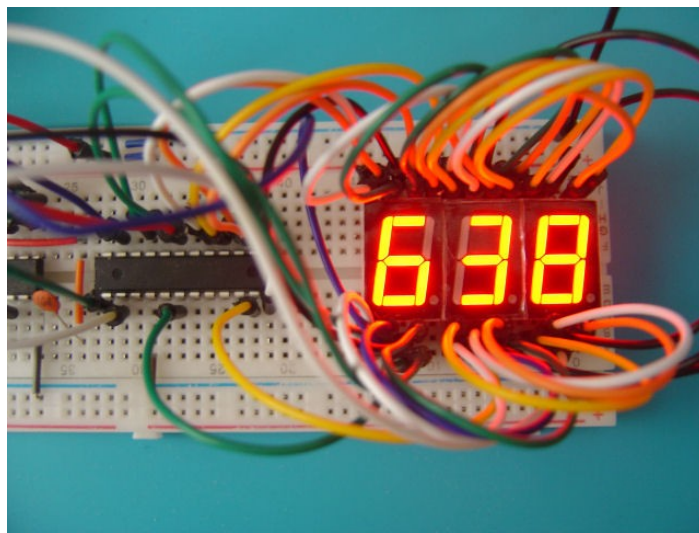
      for i := 0 to 2 do // iteracja jednościzdziesiątki-
setki
      begin

```

```
    dig := tmp mod 10;
    Max7219_SendData(i + 1, NumberMasks[dig]);
    tmp := tmp div 10;
end;

    DelayMs(100);
end;
end;
end.
```

Jeśli nigdzie nie popełniliśmy błędu to powyższy kod zadziała zgodnie z oczekiwaniem. Naturalnie, można go modyfikować według potrzeb, tworząc ciekawsze efekty wizualne, np. używając wspomnianych wcześniej matryc LED 8×8 . Widać jednak, że mały mikrokontroler ATTiny13, szczególnie w połączeniu z innymi układami scalonymi potrafi zdziałać całkiem sporo.



Trudnym do uniknięcia kosztem jest spora liczba przewodów, których musieliśmy użyć. Nie wpłynęło to pozytywnie na estetykę naszego projektu, jednak tworząc bardziej rozbudowane projekty musimy się z taką niedogodnością liczyć.

Życzę miłej zabawy!

Andrzej Karwowski

Nobody is perfect – znalazłeś/-aś błąd lub nieścisłość, masz pytanie bądź sugestię – napisz na adres ackarwow@gmail.com