Podstawy Elektroniki

Laboratorium

Ćwiczenie:

Sterowanie wyświetlaczem siedmiosegmentowym poprzez programowalny mikrokontroler z wykorzystaniem ośmiobitowego rejestru przesuwnego.

Wstęp.

Celem ćwiczenia jest wykorzystanie podstawowej wiedzy z elektroniki i programowania komputerów zdobytej w trakcie studiów do skonstruowania oraz przetestowania układu elektronicznego zawierającego mikrokontroler oraz wyświetlacz siedmiosegmentowy. Głównym narzędziem do realizacji tego celu jest zestaw złożony z mikrokontrolera Arduino UNO (**Zdjęcie 1**) oraz programu Arduino (**Zrzut ekranu 1**) służącego do jego zaprogramowania.

Ćwiczenie jest wykonywane w dwóch etapach.

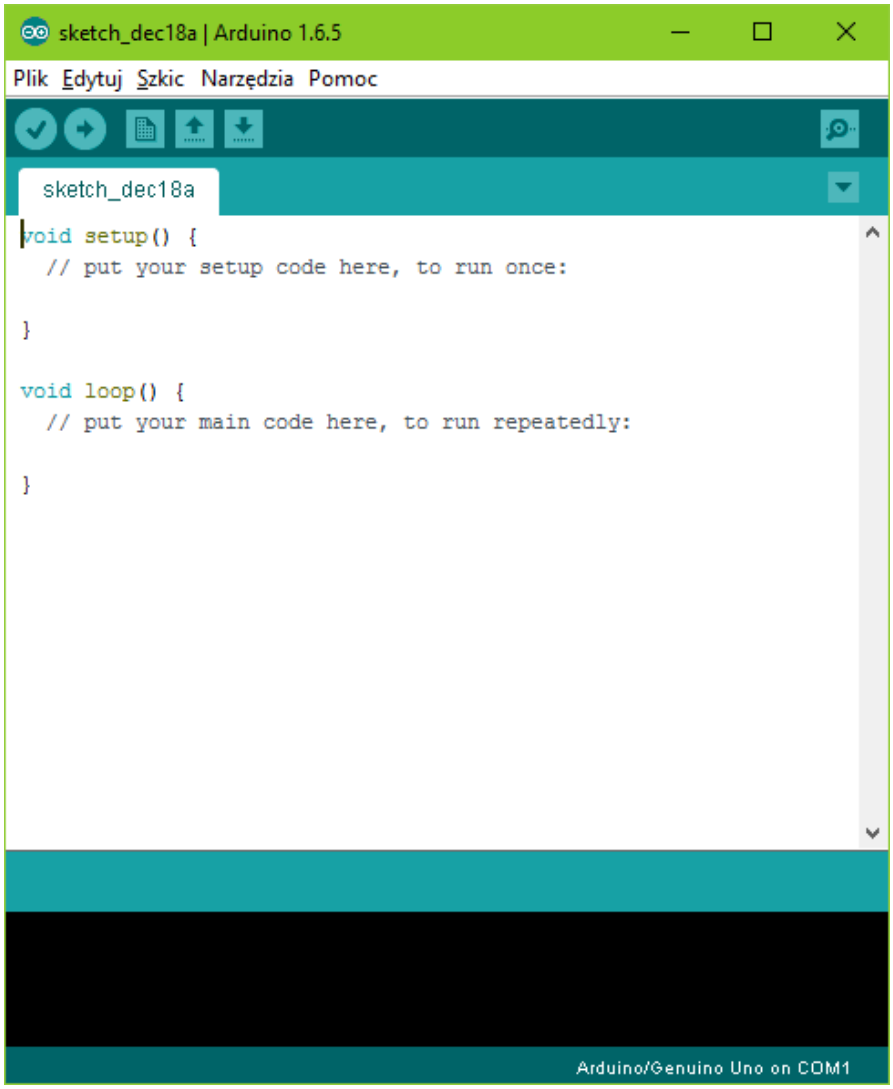
Pierwszy z nich polega na zbudowaniu układu zawierającego mikrokontroler, rejestr przesuwny i wyświetlacz siedmiosegmentowy z wykorzystaniem proponowanych schematów połączeń oraz na sprawdzeniu poprawności działania zestawionego systemu.

Drugi etap ćwiczenia polega na zaprogramowaniu zestawionego układu do wyświetlania funkcji stopera bądź minutnika na wyświetlaczu siedmiosegmentowym.

1. Opis narzędzia programistycznego Arduino Uno

Arduino jest platformą programistyczną dla systemów komputerowych przeznaczoną do programowania mikrokontrolerów. Nazwa ta jest również wykorzystywana dla mikrokontrolerów, które są w pełni kompatybilne z tym oprogramowaniem o tej samej nazwie. Arduino powstało w celu ułatwienia dostępu do tej dziedziny nauki wszelkim hobbystom. Mikrokontrolery, które możemy programować dzięki Arduino oraz tworzone przy jego pomocy układy są tanie i ogólnodostępne co przyczynia się do rozpowszechniania i czynienia tej dziedziny ogólnodostępnej. Platforma ta bazuje na języku C/C++, jednakże możliwe jest w niej wykorzystywanie też kilkunastu innych języków.

Zdjęcie 1 - moduł Arduino UNO



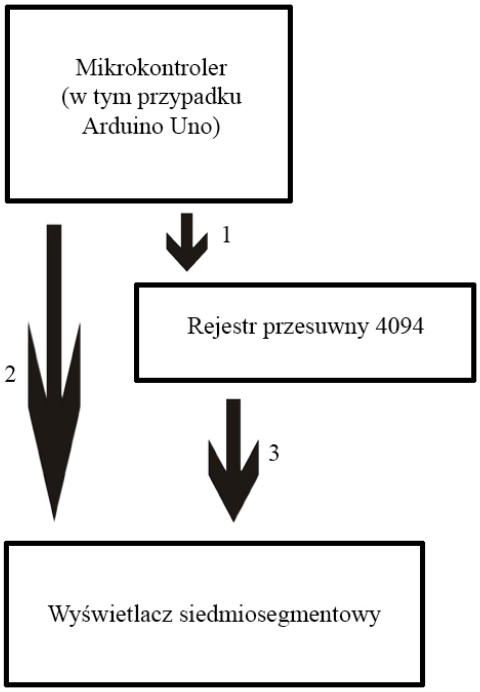
Zrzut ekranu 1 - program Arduino

Działanie mikrokontrolera Arduino opiera się na wykonywaniu instrukcji programu wgranego do jego pamięci. Zależnie od tego z czego składa się nasz układ program przetwarza dane wejściowe (np. z czujnika temperatur) i/lub steruje wyjściami naszego układu (np. wyświetlaczem LED).

Każdy program Arduino składa się z funkcji *setup()* oraz *loop()*, które odpowiadają za działanie programu. Funkcja *setup()* wykonywana jest tylko raz przy uruchomieniu układu, natomiast funkcja *loop()* wykonywana jest cały czas w trakcie działania układu. Można dodać swoje własne funkcje, jednakże układ samodzielnie nie będzie ich wykonywać (należy ich użyć w którymś miejscu w programie).

2. Opis działania układu do obsługi wyświetlacza:

Proponowany układ składa się z 3 elementów: mikrokontrolera (Arduino Uno), rejestru przesuwnego 4094 oraz wyświetlacza siedmiosegmentowego poczwórnego. Schemat ogólny jego działania prezentuje **Rys.1**:



Rys. 1 - Ogólna zasada działania układu do obsługi wyświetlacza 7 segmentowego

Zadaniem mikrokontrolera jest wysyłanie stanów logicznych wysokich i niskich w odpowiedni, zaprogramowany przez nas sposób do urządzeń z nim połączonych.

Rejestr przesuwny pełni tutaj rolę bufora, który umożliwia zmniejszanie liczby wyjść mikrokontrolera potrzebnych do wysterowania układu. Jego zastosowanie znacząco też redukuje kod źródłowy funkcji sterujących wyświetlaczem.

Wyświetlacz wykorzystywany w opisywanym tu układzie posiada wspólną anodę (+), dlatego podanie stanu niskiego (0) na dany segment powoduje jego zaświecenie, a stanu wysokiego (1) jego zgaśnięcie. Włączenie wyświetlacza odbywa się poprzez podanie mu stanu wysokiego (wysłanie do niego zasilania).

2.1 Rejestr przesuwny 4094

Rejestr jest to układ, który służy do przechowywania i odtwarzania informacji w postaci bitów. Rejestr składa się z ustalonej liczby przerzutników, z których każdy może przechowywać tylko jeden bit. Ilość przerzutników w rejestrze jest określana mianem długości rejestru. Rejestry dzielą się na cztery grupy ze względu na sposób wprowadzania i wyprowadzania z nich informacji:

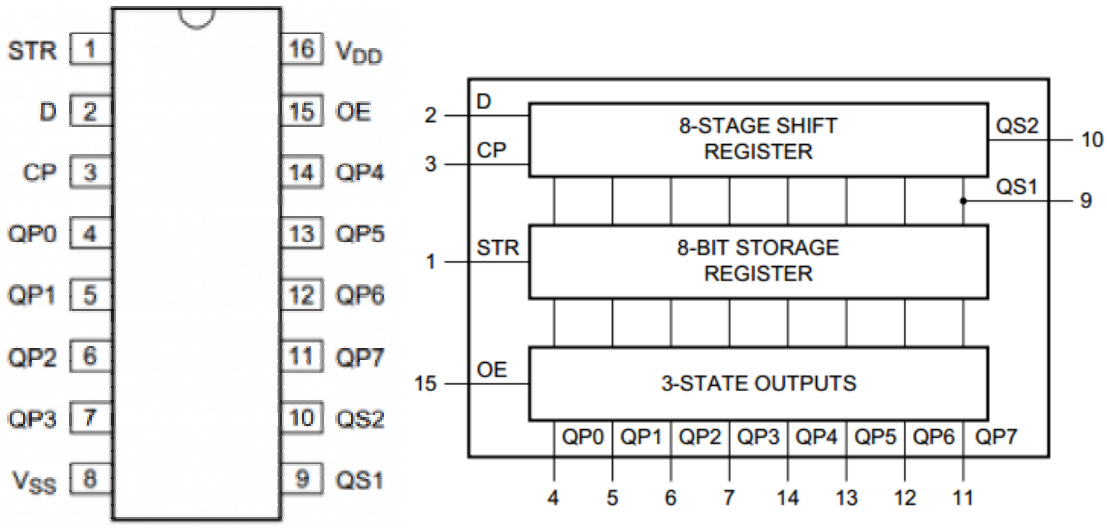
a) szeregowe - umożliwiające szeregowe wprowadzenie i wyprowadzenie danych (tzn. bit po bicie poprzez wyróżnione wejście szeregowe),

b) równoległe - umożliwiające równoległe wprowadzenie i wyprowadzenie informacji (jednocześnie do wszystkich wejść lub wyjść rejestru),

c) szeregowo-równoległe - umożliwiające szeregowe wprowadzenie i równoległe wyprowadzenie informacji,

d) równoległo–szeregowe - umożliwiające równolegle wprowadzenie i szeregowe wyprowadzenie informacji,

Rejestr, z którego korzystamy, zbudowany jest z ośmiostopniowego rejestru przesuwnego, ośmiobitowego rejestru magazynującego oraz wyjść trójstanowych. Rejestr przesuwny posiada wejście szeregowe (wejście **D** na **Rys. 2**) gdzie po wprowadzeniu każdego następnego bitu dana jest przesuwna o jedną pozycję wraz z taktowaniem wejścia zegarowego (wejście **CP** na **Rys. 2**). Posiada on też wyjście szeregowe (wyjście **QS2** na **Rys. 2**), które pozwala łączyć kilka rejestrów ze sobą, aby najstarsze bity były przesuwane do kolejnych rejestrów. Bity są przechowywane w rejestrze magazynującym, na który musimy podać stan wysoki (wejście **STR** na **Rys. 2**), aby zostały one podane na wyjścia **Q0-Q7** rejestru. Wyjścia te są zbudowane z bramek trójstanowych. Aby stany logiczne pojawiły się na wyjściach rejestru musimy do nich podać stan wysoki (wejście **OE** na **Rys. 2**).



Rys. 2 – Ośmiobitowy rejestr przesuwny 4094 i jego wewnętrzny schemat ideowo-koncepcyjny

2.2 Wyświetlacz siedmiosegmentowy

Wyświetlacz siedmiosegmentowy jak sama nazwa wskazuje składa się z siedmiu segmentów (ośmiu jeżeli posiada kropkę). Zależnie od technologii jego wykonania każdy segment może być np. oddzielną diodą LED. Działa on na zasadzie podawania stanów niskich lub wysokich do odpowiednich segmentów co skutkuje ich zaświeceniem lub zgaszeniem. To czy stan wysoki zaświeca dany segment czy gasi zależy od tego czy posiada on wspólną

katodę (wtedy stan wysoki zaświeca segment) czy wspólną anodę (wówczas stan wysoki gasi segment).

2.3 Realizacja sterowania wyświetlaczem

Sterowanie wyświetlaczem siedmiosegmentowym jest realizowane według układu z **Rys. 1** w następujący sposób:

1. Mikrokontroler wysyła sekwencję bitów do rejestru, które ustawiane są na jego wyjścia **Q0 – Q7** w odpowiedniej kolejności.

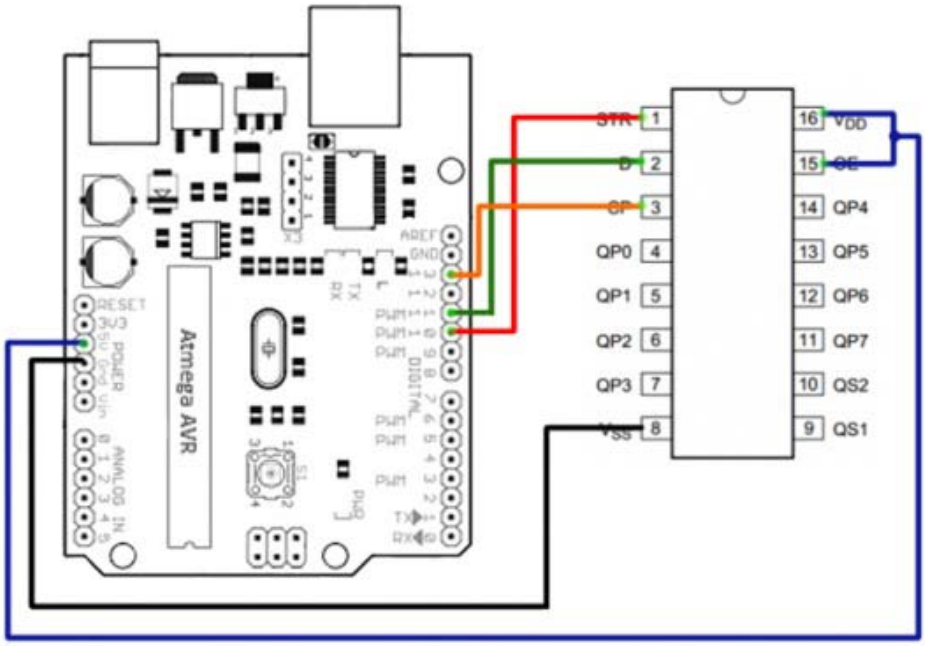
2. Mikrokontroler wysyła sygnał poprzez wybrane wyjście w celu włączenia odpowiedniego fragmentu wyświetlacza.

3. Rejestr po otrzymaniu stanu wysokiego na wejście ***Strobe*** wysyła odpowiednie stany wysokie lub niskie (zależnie od wysłanych bitów z punktu 1.) na wyjścia **Q0-Q7**, które trafiają do wyświetlacza i zapalają odpowiednie jego segmenty podłączone do tych wyjść.

3. Montaż układu:

Na wstępie łączymy rejestr przesuwny z modułem Arduino w następujący sposób:

Wejście rejestru Strobe (STR) łączymy z wejściem SS (10) na Arduino. Wejście Data (D) łączymy z MOSI (11) na Arduino, a wejście Clock (CP) z wejściem SCK (13). Do wejścia V_{SS} podpinamy masę GND. Napięcie 5V z Arduino wpinamy do wejścia V_{DD} oraz OE (Output Enable) w rejestrze. Powyższy schemat połączeń przedstawia **Rys. 4**:

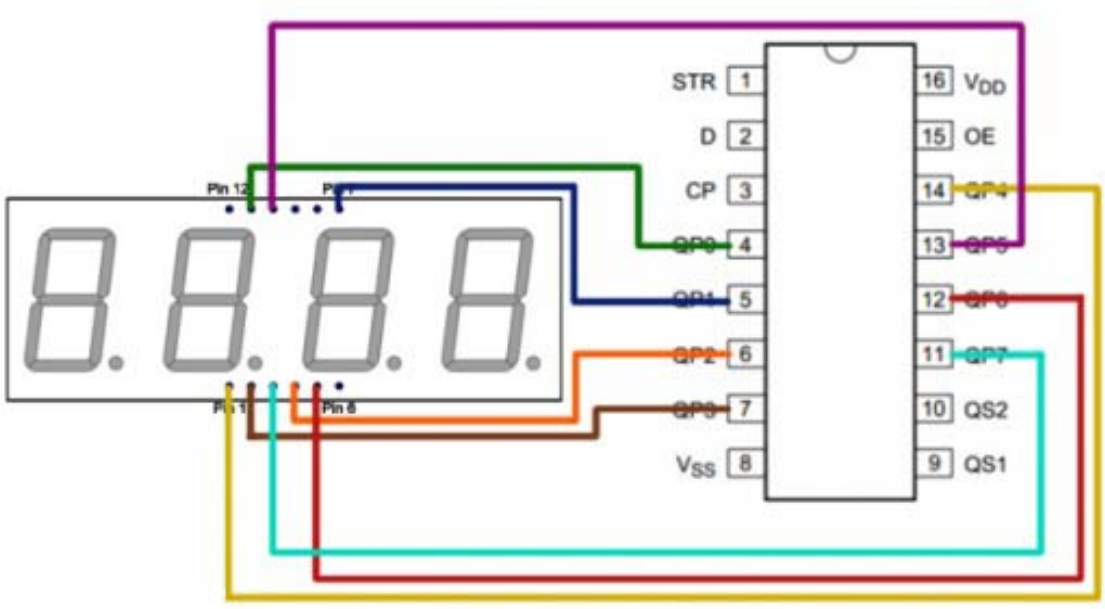


Rys. 4 – schemat połączenia rejestru przesuwnego z mikrokontrolerem

Wejścia QP0 – QP7 rejestru przesuwnego łączymy z wejściami wyświetlacza w kolejności: A, B, C, D, E, F, G, dp. Przedstawia to **Rys. 5**.

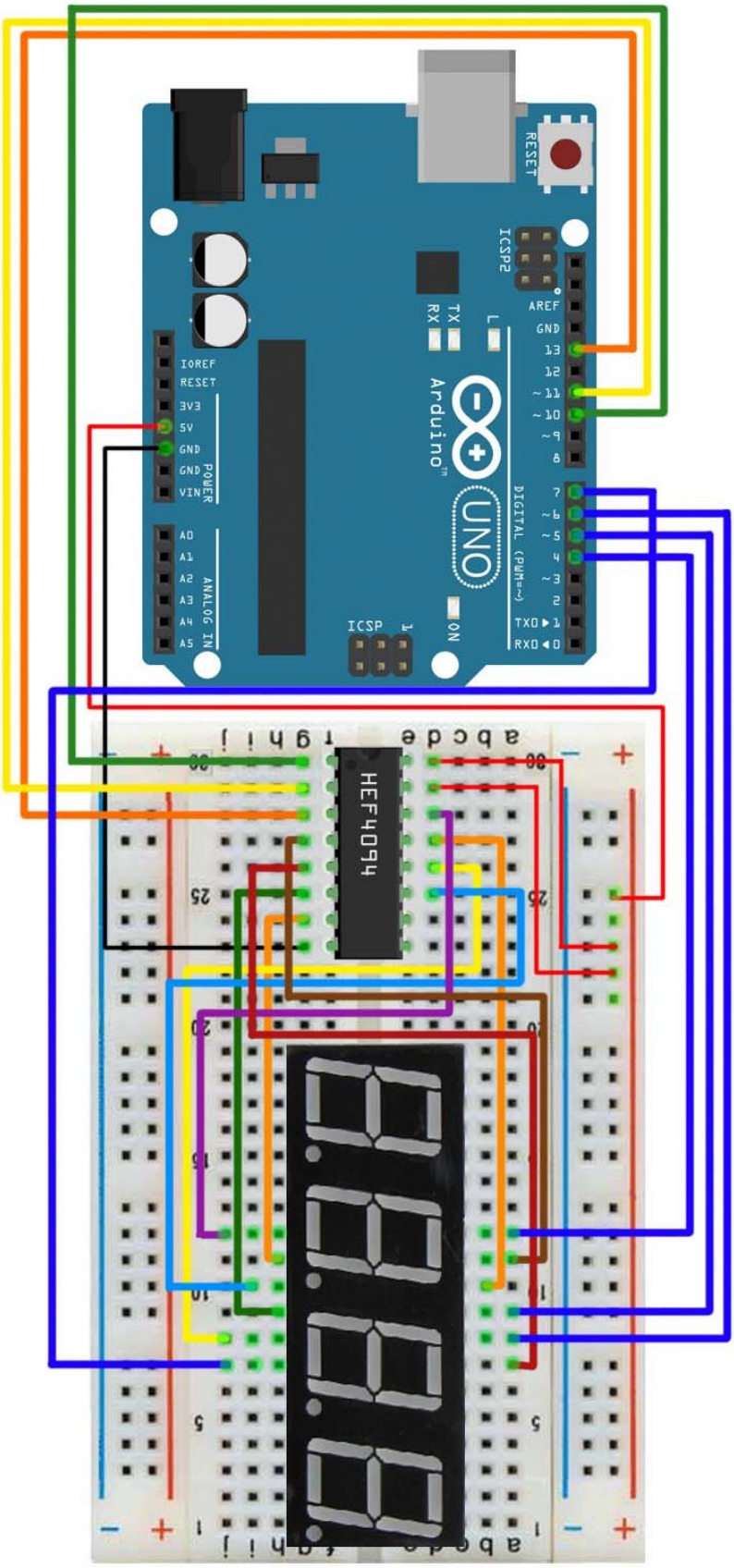
Pojedyncze segmenty odpowiadające danym pinom w wyświetlaczu siedmiosegmentowym:

- Pin 1 – E
- Pin 2 – D
- Pin 3 – dp
- Pin 4 – C
- Pin 5 – G
- Pin 6 - Wyświetlacz 4 (licząc od lewej strony)
- Pin 7 – B
- Pin 8 - Wyświetlacz 3
- Pin 9 – Wyświetlacz 2
- Pin 10 – F
- Pin 11 – A
- Pin 12 – Wyświetlacz 1



Rys. 5 – schemat połączenia rejestru przesuwnego z wyświetlaczem siedmiosegmentowym

Pozostałe wejścia wyświetlacza (piny 6, 8, 9 i 12) podłączamy do wolnych (obojętne których) wejść na Arduino (ważne abyśmy wiedzieli w jakiej kolejności to zrobiliśmy żeby potem wyświetlać na właściwym wyświetlaczu odpowiednie cyfry). Połączenie układu porównaj ze schematem znajdującym się na następnej stronie (**Rys. 6**).



Rys. 6 – widok połączonego układu

Wybranie wyjść 4, 5, 6, 7 na Arduino jest dowolne (w Twoim przypadku mogą to być którekolwiek z wolnych)

4. Programowanie układu

Przed wykonaniem tej części zadania upewnij się, że poprawnie zmontowałeś układ!

Bez wykorzystania rejestru każde wyświetlenie jednego segmentu wyświetlacza wymagałoby ośmiu linijek kodu oraz dodatkowych dwóch do zaświecenia, a następnie zgaszenia wyświetlacza. W związku z czym nasz program obsługujący wyświetlacz byłby bardzo długi, co miałoby negatywny wpływ na szybkość działania całego układu. Rejestr otrzymuje sekwencję bitów, które po kolei ustawia na swoich wyjściach, zaczynając od ostatniego bitu w kodzie. Dzięki temu zabiegowi nasz kod skraca się z 10 linijek do zaledwie 5-ciu: wysłanie sekwencji bitów do rejestru, wysłanie stanów wysokich i niskich na wyjścia **Q0-Q7** rejestru, uruchomienie danego wyświetlacza, wyłączenie wyjść **Q0-Q7** rejestru, wyłączenie danego wyświetlacza. Ilość zajętych wyjść mikrokontrolera w przypadku niekorzystanie z rejestru wynosi 12 (dla każdego pojedynczego pinu wyświetlacza potrzebujemy jedno wyjście). Korzystając z rejestru przesuwnego ośmiobitowego ilość ta zmniejsza się do 7 (4 do obsługi wyświetlaczy oraz 3 do obsługi rejestru).

Do wysyłania sekwencji bitów do rejestru przesuwnego będziemy korzystać z gotowej biblioteki SPI i jej funkcji. Dlatego nasz program musi zacząć się od jej dołączenia poprzez dyrektywę *#include <SPI.h>*.

Potrzebne nam też będą odpowiednie zmienne. Na początek wymagane są stałe, w których będziemy przechowywać numery wyjść na Arduino, do których są podłaczone cztery wyświetlacze oraz pin Strobe rejestru przesuwnego. Np.:

```
const int STRB = 10;
const int display1 = 4;
```

Prócz stałych potrzebowali będziemy też zmiennych, do których będziemy mogli przypisywać liczby do wyświetlenia. Są nam potrzebne minimum 3 zmienne, gdyż 2 z nich będziemy musieli wykorzystywać do przechowywania wartości upływającego czasu (np. sekund i minut), a jedna zmienna będzie nam potrzebna do przechowywania cyfry, którą mamy wyświetlić w danym momencie.

Niezbędne jest także utworzenie tablicy typu byte, do przechowywania gotowych ciągów 0 i 1, które będą odpowiadać odpowiednim liczbom na wyświetlaczu (zapaleniu odpowiednich segmentów wyświetlaczy). Postać pojedynczej zmiennej z tej tablicy wygląda następująco: Bxxxxxxxx, gdzie, poczynając od lewej strony, każdy kolejny ‘x’ to segment dp, g, f, e, d, c, b, a. W przypadku naszego wyświetlacza (wspólna anoda) stan wysoki (1) powoduje zgaszenie segmentu, a stan niski (0) umożliwia jego świecenie. Tak więc gdybyśmy chcieli wyświetlić cyfre np. 0 nasz kod wyglądałby tak: B11000000.

Tworząc tablice pamiętaj iż potrzebujesz 11 elementów (10 cyfr i osobno kropka), a jej elementy powinny być w kolejności ‘wyświetlanej cyfry’, aby odwołanie do tablicy poprzez indeks 0 skutkowało wyświetleniem cyfry 0.

Ostatnia wymagana zmienna jest nam potrzebna do obliczania upływającego czasu działania naszego programu. Jako iż czas ten będzie przechowywany w milisekundach musimy mieć możliwość przechowywania w naszej zmiennej bardzo dużych wartości, dlatego zmienna ta powinna być typu *unsigned long*. Aktualną wartość czasu będziemy obliczać za pomocą funkcji *millis()*.

Istnieje w programowaniu funkcja *delay(time)*, która wstrzymuje działanie programu na określony czas podany w milisekundach jako parametr funkcji. My jednakże nie możemy z niej korzystać. Dlaczego? Otóż jak można zauważyć na schemacie wyświetlacza (Schemat 3) mamy 8 pinów sterujących poszczególnymi segmentami oraz 4 piny sterujące zapalaniem odpowiedniego wyświetlacza. Kiedy chcemy wysłać cyfrę do konkretnego wyświetlacza musimy go włączyć, wtedy on odbierze stany ustawione na 8-miu pinach odpowiadających za segmenty, a w tym czasie pozostałe wyświetlacze muszą pozostać wyłączone, gdyż inaczej odbierałyby te same stany. W związku z czym nie jest możliwe wysyłanie (w przypadku tego konkretnego wyświetlacza) różnych cyfr do wszystkich wyświetlaczy jednocześnie. Dlatego aby uzyskać liczby wyświetlone na 4 wyświetlaczach musimy odpowiednio szybko (z dużą częstotliwością) zapalać konkretny wyświetlacz, wysyłać odpowiednie stany wysokie i niskie na piny obsługujące segmenty, następnie go zgasić i wykonać te same czynności dla pozostałych wyświetlaczy po czym wykonać te czynności ponownie. W związku z czym użycie funkcji *delay(time)* uniemożliwiałoby nam wykonywanie tych czynności, gdyż wstrzymuje ona działanie programu na dany czas. Dlatego ważne jest, aby czas obliczany był osobno, w trakcie działania pętli wyświetlającej.

Programowanie funkcji *setup()*:

Naszą pierwszą linią powinno być użycie wbudowanej funkcji z bilbioteki SPI, mianowicie: *SPI.begin();*. Funkcja ta przygotowuje nasze wyjścia SS, MOSI oraz SCK (podpięte kolejno do Strobe, Data i Clock rejestru) i ustawia je na OUTPUT.

W kolejnych liniach, korzystając z funkcji *pinMode(pin, mode)* musimy piny naszych 4 wyświetlaczy ustawić na OUTPUT, np. *pinMode(display1, OUTPUT)*.

Ostatnia linia kodu funkcji *setup()* powinna zawierać przypisanie początkowej wartości dla naszej zmiennej przechowującej czas działania programu za pomocą funkcji *millis()*. Funkcja *setup()* wykonuje się tylko raz, przy wgraniu programu dlatego to tutaj należy ustawiać wszelkie początkowe wartości.

Programowanie funkcji *loop()*:

W funkcji tej musimy zaprogramować wyświetlanie odpowiednich cyfr na właściwych wyświetlaczach oraz zliczanie upływającego czasu.

Przed wyświetleniem cyfry na wyświetlaczu musimy ją uzyskać za pomocą reszty z dzielenia bądź dzielenia całkowitego (w C oraz C++ odpowiednio % i /).

Kolejnym krokiem jest wysłanie do rejestru ciągu znaków do zapalenia segmentów dla odpowiedniej cyfry. Jeżeli tablica znaków została utworzona według wyższych zaleceń to podanie

cyfry jako indeks tablicy poskutkuje pobraniem właściwego ciągu znaków. Korzystamy tutaj z funkcji *SPI.transfer(buffer);*. Po wysłaniu ciągu znaków do rejestru należy na wejście Strobe rejestru podać stan wysoki za pomocą funkcji *digitalWrite(pin, mode)*, aby rejestr mógł wysłać odpowiednie stany na jego wyjścia Q0-Q7. Musimy także na nasz odpowiedni wyświetlacz podać stan wysoki, aby mógł on wyświetlić cyfrę. Ostatnim krokiem jest zarówno na wejście Strobe jak i nasz konkretny wyświetlacz podanie stanu niskiego. Dla wyświetlenia ostatniej cyfry (jednostka sekundy) będzie to wyglądać następująco:

```
1. digitToDisp4 = seconds % 10;
2. SPI.transfer(displayArray[digitToDisp4]);
3. digitalWrite(STRB, HIGH);
4. digitalWrite(display4, HIGH);
5. digitalWrite(STRB, LOW);
6. digitalWrite(display4, LOW);
```

1. Uzyskanie jednostki sekund poprzez resztę z dzielenia przez 10
2. Wysłanie odpowiedniego bufora do rejestru
3. Wysłanie sygnału z rejestru do wyświetlacza
4. Uruchomienie odpowiedniego wyświetlacza aby wyświetlić cyfrę
5. Wyłączanie wysyłania sygnału z rejestru
6. Wyłączenie wyświetlacza aby nie odbierał innych sygnałów

W taki sam sposób (lub podobny) należy zrobić wyświetlanie pozostałych cyfr. Aby oddzielić minuty od sekund na drugim wyświetlaczu należy zawsze zapalać kropkę, co też powinno się znaleźć w naszej funkcji.

Na koniec należy zająć się funkcją zliczania czasu. Funkcja millis() zwraca nam czas działania naszego urządzenia. Przy jego starcie przypisaliśmy go do zmiennej (dalej *startTime*). Sprawdzanie czy upłynęła sekunda jest dość proste:

aktualny czas – początkowy = czas, który upłynął.

Jeżeli program na działać jako stoper to wartości początkowe minut oraz sekund będą równe 0 i w trakcie działania będą one rosły. Gdyby natomiast miał działać jako minutnik to na początku należałoby przypisać odpowiednie wartości minutom oraz sekundom, a następnie zmniejszać je wraz z upływającym czasem.

Ciekawostka:

Jeżeli pozostało Ci jeszcze trochę czasu do końca laboratorium możesz teraz sprawdzić skutek użycia funkcji *delay(time)*, w programie. Nie musisz nic zmieniać. Dodaj tylko jako ostatnią linie kodu funkcję z parametrami odpowiednio:

- 1000 ms,
- 100 ms,
- 10 ms,

i zaobserwuj różnicę między tymi trzema parametrami oraz bez użycia funkcji *delay(time)*.