***Wojskowa Akademia Techniczna***

***im. Jarosława Dąbrowskiego***

Laboratorium z przedmiotu:

Interfejsy komputerów cyfrowych

Sprawozdanie z ćwiczenia laboratoryjnego nr 2:

**TRANSMISJA SZEREGOWA SYNCHRONICZNA**

Prowadzący:

mgr inż. Krzysztof Szajewski

**Wykonał:** Radosław Relidzyński

**Grupa:** WCY20IY4S1

**Data laboratoriów**: 11.04.2021 r.

Spis treści

[A. Treść zadania 2](#_Toc102832284)

[B. Schemat układu (Proteus) 3](#_Toc102832285)

[C. Program mikrokontrolera (Keil) 3](#_Toc102832286)

[D. Analiza i wnioski 6](#_Toc102832287)

[Cykl życia programu: 6](#_Toc102832288)

[Przykładowe działanie programu 7](#_Toc102832289)

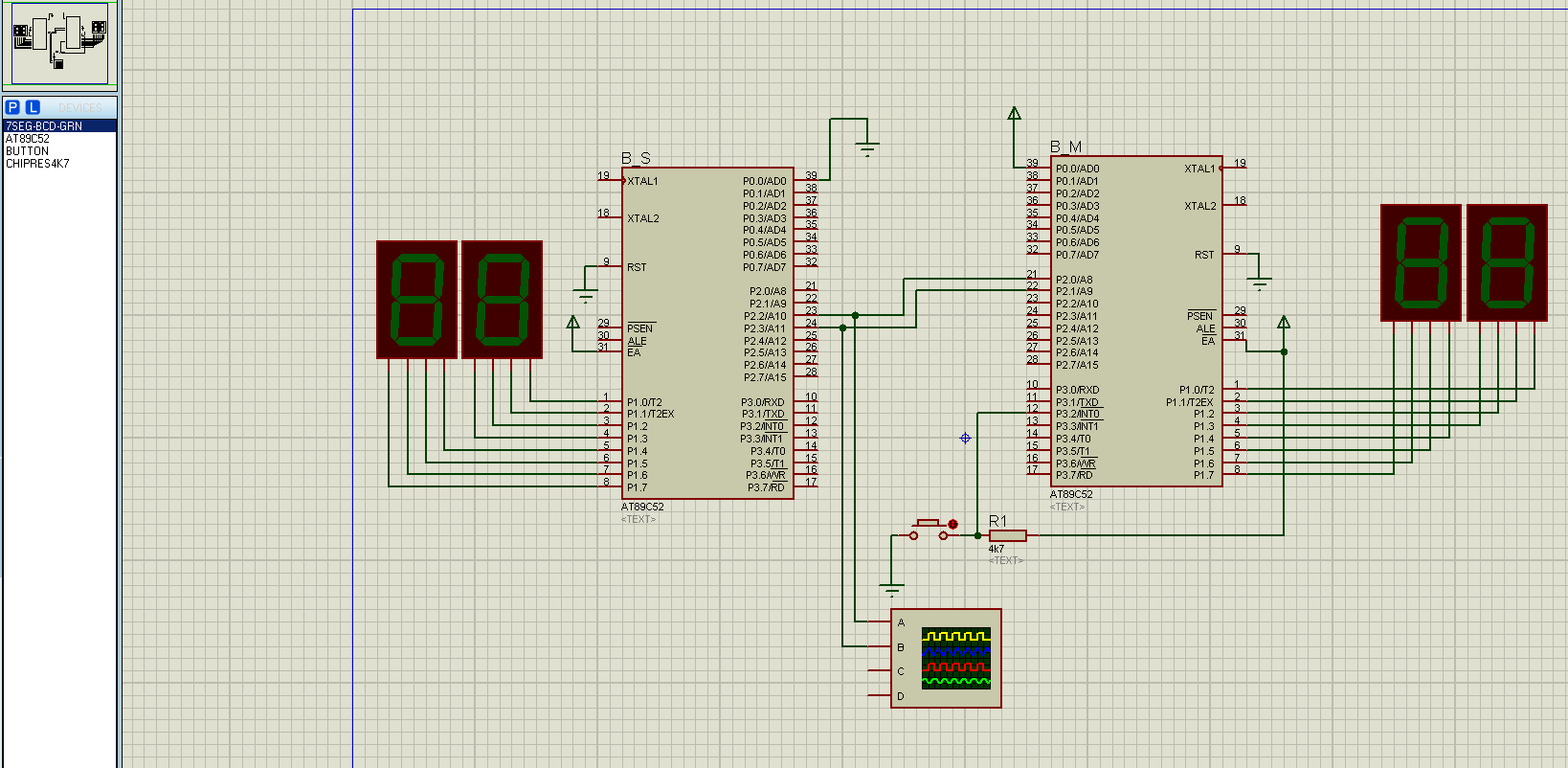
[Podsumowanie 10](#_Toc102832290)

# Treść zadania

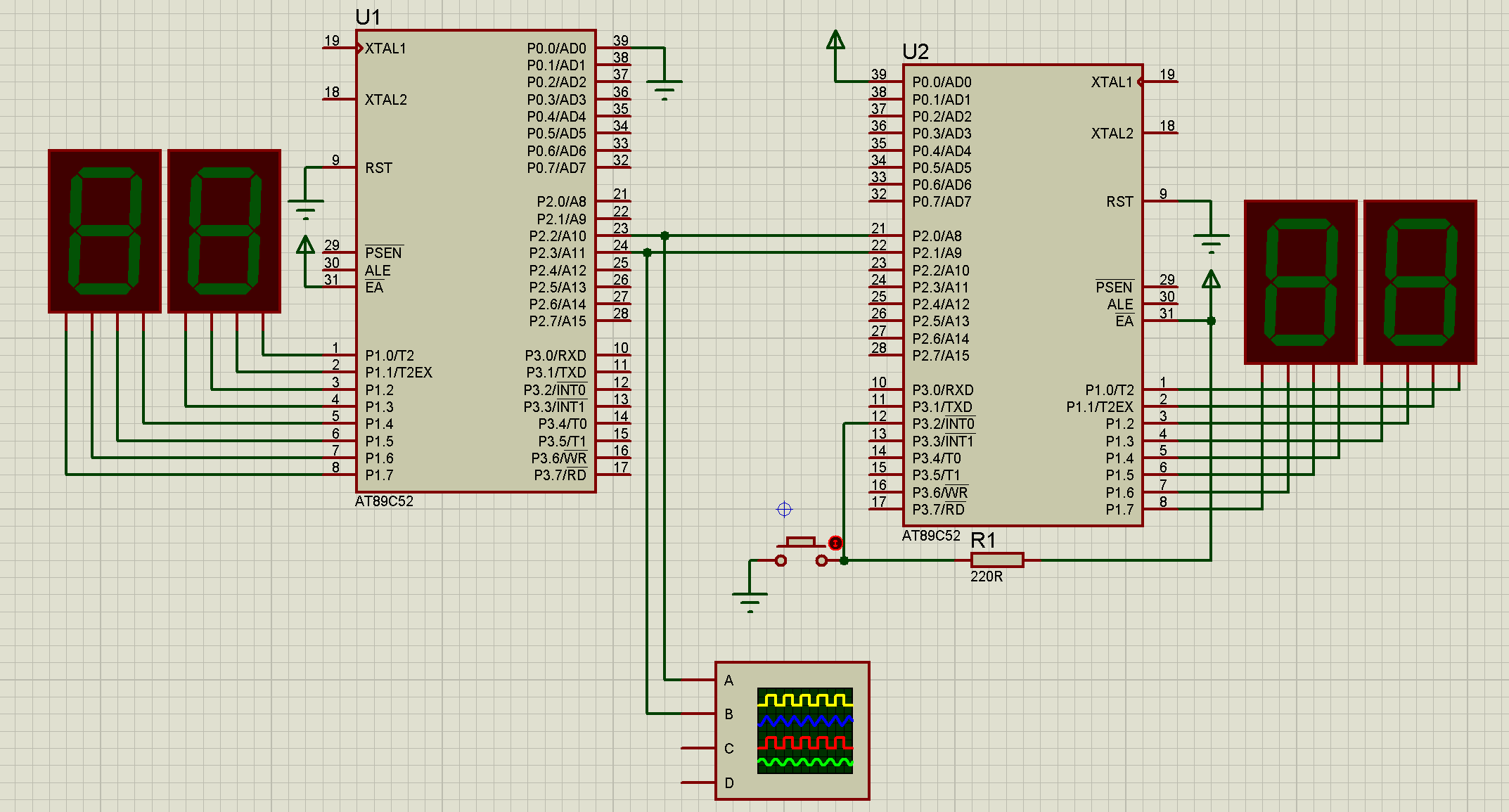
Zbudować w środowisku Proteus stanowisko realizujące transmisję szeregową synchroniczną. Stanowisko powinno zawierać:

* mikrokontroler AT89C52 - pełniący rolę nadajnika (MASTER) - sygnał "1" do pinu P0\_0
* mikrokontroler AT89C52 - pełniący rolę odbiornika (SLAVE) - sygnał "0" do pinu P0\_0
* po dwa siedmiosegmentowe wyświetlacze podlączone do pinów portu P1 nadajnika i odbiornika
* przycisk podłączony do pinu ~INT0 nadajnika
* linia SDA (serial data) pomiędzy pinami P2\_a nadajnika  i P2\_a+2 odbiornika
* linia SCL (serial clock) pomiędzy pinami P2\_b nadajnika  i P2\_b+2 odbiornika, gdzie a i b wskazuje prowadzący; jeśli nie są wskazane to a = 0, b = 1

Układ powinien wyglądać jak na rysunku:



# Schemat układu (Proteus)



# Program mikrokontrolera (Keil)

#include <REGX52.H>

 sbit clkM = P2^0; // Pierwszy bit mastera - zegar

 sbit dtaM = P2^1; // Drugi bit mastera - dane

 sbit clkS = P2^2; // Pierwszy bit slave'a - zegar

 sbit dtaS = P2^3; // Drugi bit slave'a - dane

 bit bM = 1; // Flaga sprawdzajaca czy wcisniety zostal przycisk

 int dt = 10; // Wartosc opoznienia do funkcji czekaj()

 int tt = 100; // Wartosc opoznienia do funkcji czekaj()

 volatile unsigned char liczbaM = 0x00;

 unsigned char liczbaS = 0x00;

 unsigned char bajt = 0x00;

 void czekaj(int i) // Wykonuje operacje wymuszajace wstrzymanie reszty programu

 {

    unsigned int k, l, m;

    for (l = 0; l < i; l++) // Wykonuje zestaw zlozonych czasowo operacji

    {

        k = 500;

        m = 1000;

        k = m \* l;

    }

 }

 void liczInt0() interrupt 0 // W przypadku przerwania

 {

    EX0 = 0; // Wylacza przerwanie

    liczbaM++; // Zwieksza liczbe do wyswietlenia

    if (liczbaM == 255)

        liczbaM = 0; // Zachowanie cyklicznosci, nastepna po maksymalnej mozliwej wartosci to 0

    EX0 = 1; // Wylacza przerwania

    bM = 0; // Zmiana flagi przycisku

 }

 void zapiszBajt(unsigned char bajt)

 {

    unsigned char liczbaBitow = 8;

    do

    {

        dtaM = bajt & 0x80; // Wartosc wyjsciowa na port P1

        czekaj(dt);

        czekaj(tt);

        clkM = 0; // Zmiana zegara na 0 (master)

        czekaj(tt);

        clkM = 1; // Zmiana zegara na 1 (master)

        czekaj(tt);

        bajt = (bajt << 1) + 1; // przesuniecie bajtu o 1

    } while(--liczbaBitow); // Wykonuje operacje po calym bajcie

    clkM = 1; // Zmiana zegara na 1 (master)

    dtaM = 1; // Dana wynosi 0

 }

 unsigned char czytajBajt() // Odczytaj wiadomosc z mastera

 {

    unsigned char liczbaBitow = 8;

    unsigned char wynik = 0;

    do

    {

        while(clkS == 1) // Dopoki zegar wynosi 1 (slave)

        {

            czekaj(dt);

        }

        wynik = wynik << 1; // przesuniecie wartosci binarnej o 1

        if(dtaS)

            wynik ++; // Jesli jest niezerowa dana zwieksz wynik

        while(clkS == 0)

        {

            czekaj(dt);

        }

    } while(--liczbaBitow); // Wykonuj po calym bajcie

    return wynik; // Po zakonczeniu petli mamy odkodowana cala dana

 }

 void initInt0() // Podane na odpowiednie wejscia poczatkowe wartosci

 {

    liczbaM = 0; // Poczatkowa wartosc wyjscia

    IT0 = 1; // INT0 aktywne zero

    EX0 = 1; // Wlaczenia INT0

    EA = 1; // Wlaczenie wszstkich przerwan

 }

 void main()

 {

    initInt0(); // Uzupelnij poczatkowymi wartosciami

    P1 = 0; // Na poczatku licznik wynosi 0

    clkM = 1; // Inicjalizacja zegara

    dtaM = 1; // Inicjalizacja transmisji (na razie bez danych)

    // Wykonuje sie albo master, albo slave

    // Master czeka na wcisniecie przycisku, zeby wyslac dana

    // Slave czeka na sygnal od mastera, zeby zaczac odczytywac dana

    while (P0\_0 == 1) //MASTER

    {

        while(bM) // Dopoki przycisk nie jest wcisniety czekaj

        {

            czekaj(dt);

        }

        P1 = liczbaM; // Zwieksz wartosc wyswietlana na masterze

        zapiszBajt(liczbaM); // Zapisz bajt i wyslij do slave'a

        bM = 1; // Wiadomosc wyslana, zmiana flagi, zeby czekac na ponowne wyslanie przycisku

    }

    while (P0\_0 == 0) //SLAVE

    {

        while(clkS == 1) // Dopoki nie ma sygnalu o transmicji czekaj

        {

            czekaj(dt);

        }

        liczbaS = czytajBajt(); // Odczytaj otrzymana wartosc

        P1 = liczbaS; // Wyswietl odczytana wartosc

    }

 }

# Analiza i wnioski

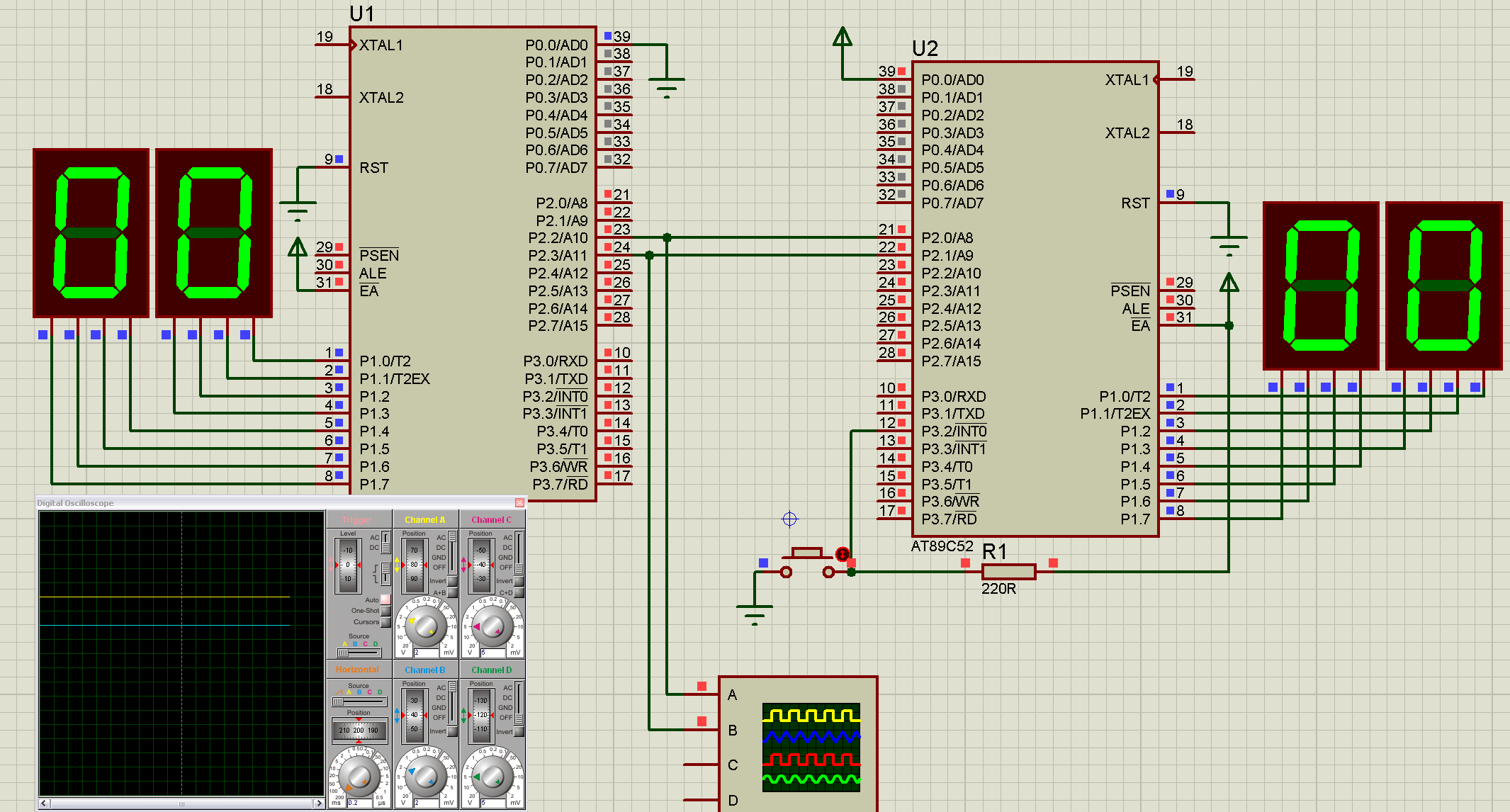
### Cykl życia programu:

1. I master, i slave czekają (nic się nie dzieje).
   1. Oscylator na obu łączach nic nie wysyła.
   2. Nie następują żadne zmiany na wyświetlaczach oraz na zmiennych.
   3. Master czeka na zmianę flagi „bm”, czyli na wciśnięcie przysicku.
2. Wciśnięty zostaje przycisk.
   1. Flaga „bm” się zmienia, pętla dla mastera się aktywuje.
3. Działanie mastera.
   1. Na wyświetlacz mastera zostanie wysłana następna wartość (na podstawie zmiennej ”liczbaM”, która przechowuje następną wartość).
   2. Nowa wartość zostaje zapisana i wysłana transmisją szeregową do slave’a (bit po bicie, z wejścia P2^1 mastera do P2^3 slave’a).
   3. Zegar mastera wysyła sygnał na zegar slave’a informujący o przesyłaniu danych (dzięki niemu slave będzie wiedział, że będzie miał dane do odczytania).
   4. Zmiana flagi „bm” na poprzednią wartość, koniec działania mastera.
4. Działanie slave’a.
   1. Slave odbiera sygnał od mastera o tym, że będzie on przesyłał dane.
   2. Pobiera dane bit po bicie z portu P2^3 i zapisuje je po kolei do zmiennej (z wykorzystaniem przesunięć binarnych).
   3. Odczytaną wartość wysyła na port P1, nowa wartość zostaje wyświetlona.
5. Koniec cyklu.
   1. Cykl działania się kończy w momencie, gdy master przestaje przesyłać sygnał do slave’a (zarówno o informacji o tranmisji, jak i samo przesyłanie danych).
   2. Na oscylatorze ponownie widać proste linie (brak jakichkolwiek zmian sygnału).

### Przykładowe działanie programu

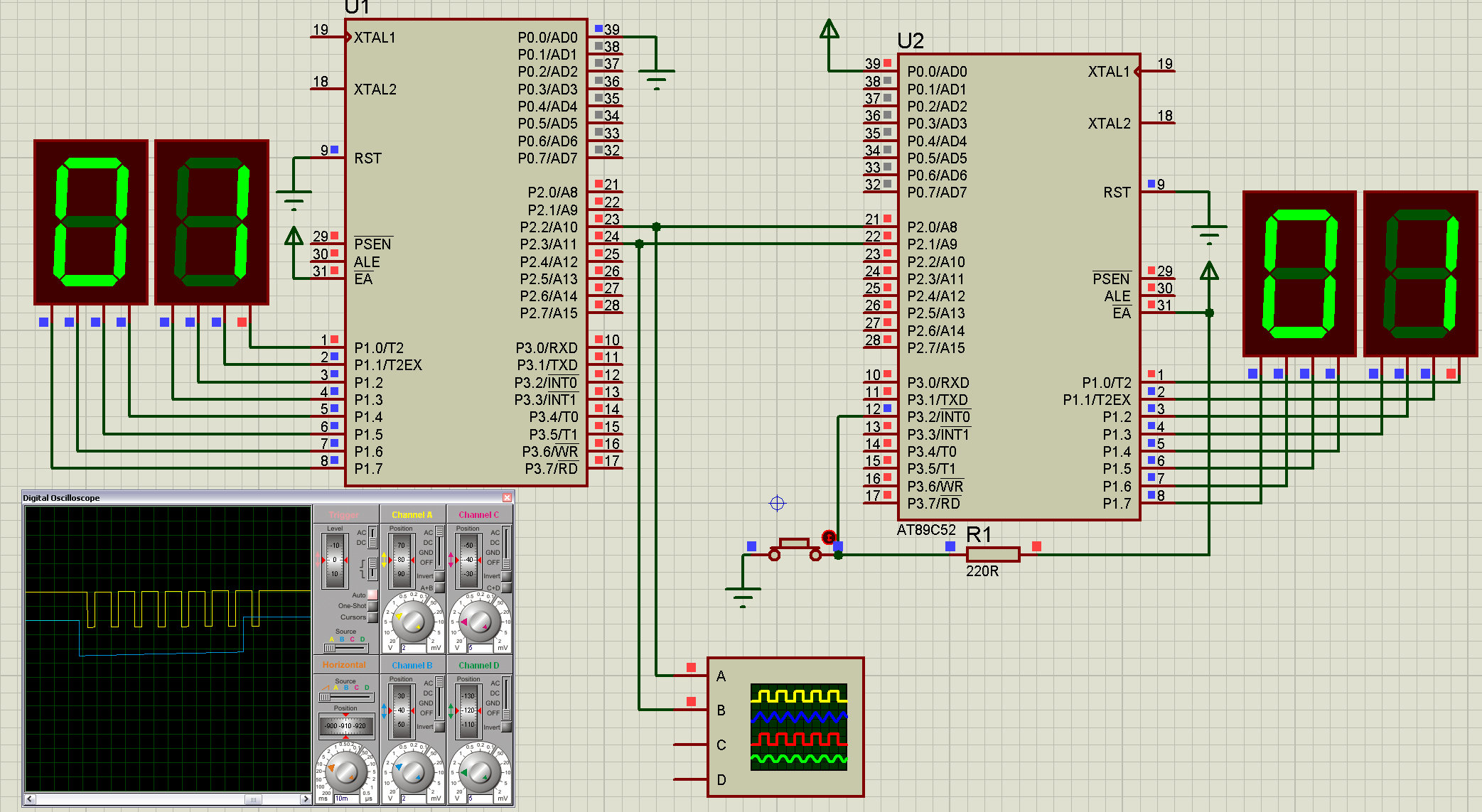
1. Start programu.

Wyzerowane wyświetlacze, brak jakichkolwiek zmian na oscylatorze.



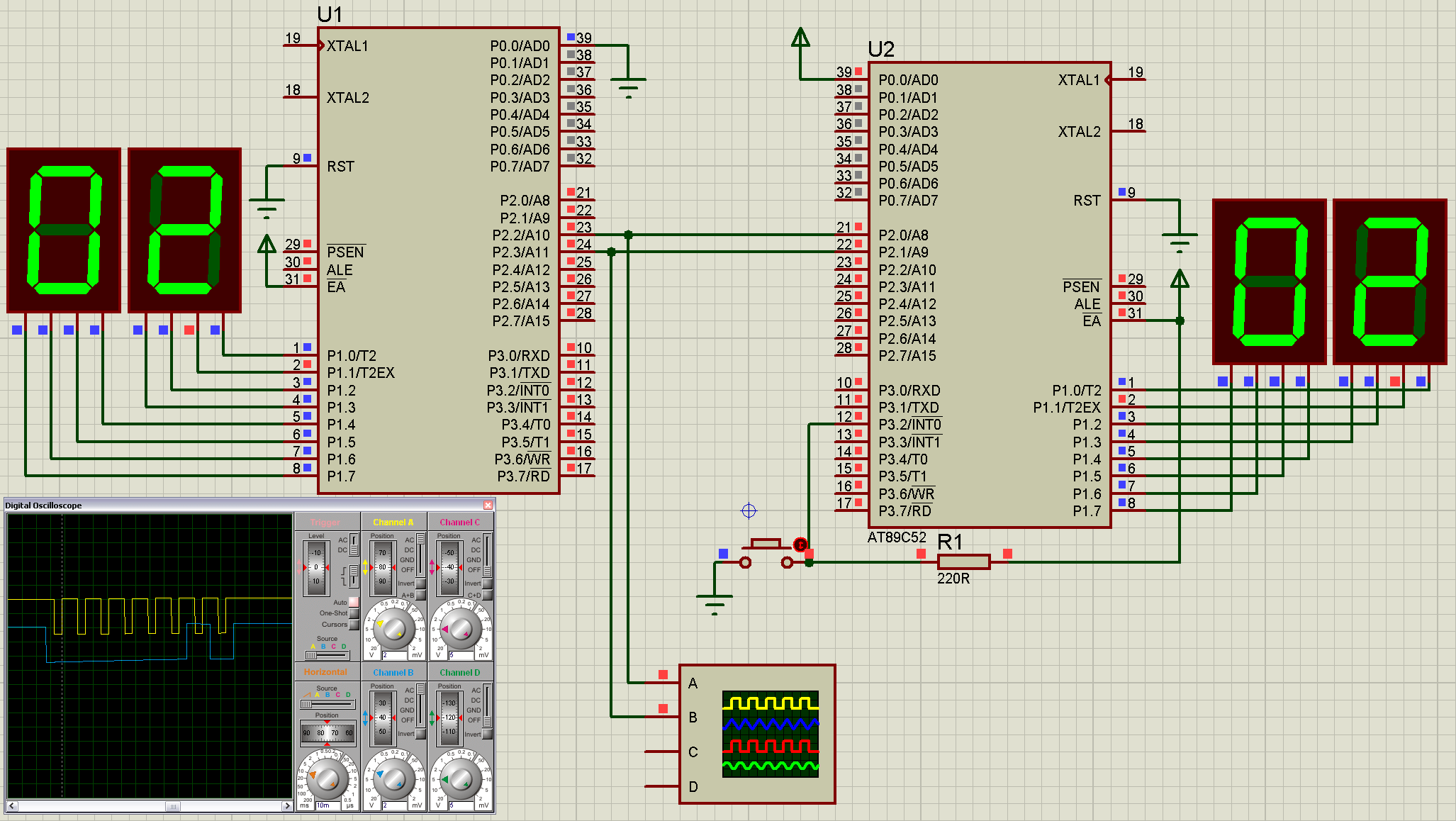
1. Pierwsze wciśnięcie

Jest sygnał nadawania, linią danych przesyłana jest wartość 0000 0001 (widać to po tym, że w ostatnim momencie wysyłania sygnału nadawania jest zmieniona wartość sygnału – jest to 1)



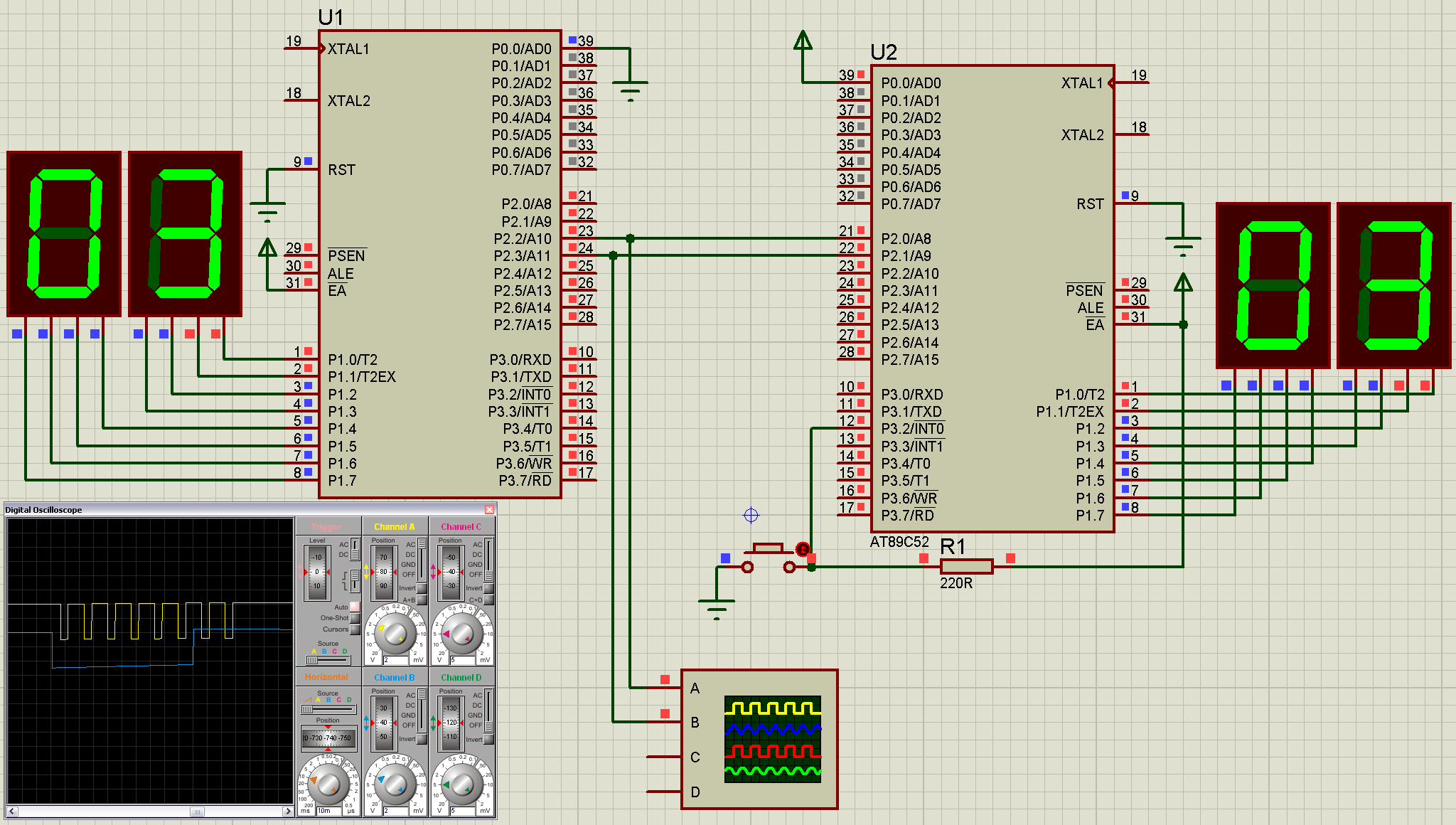
1. Drugie wciśnięcie

Sygnał 00000010 – czyli decymalnie 2



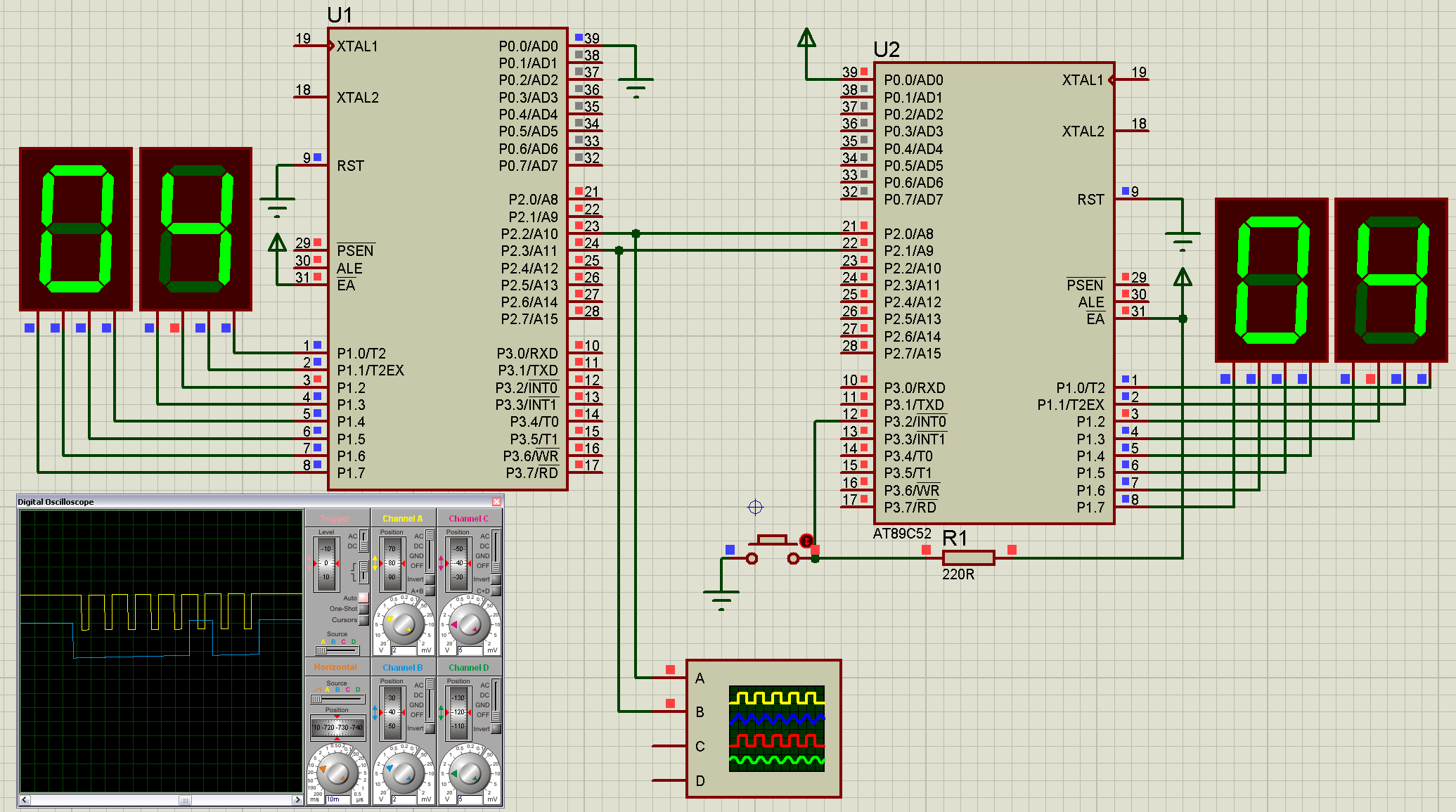
1. Trzecie wciśnięcie

Sygnał 00000011 – czyli suma dwóch poprzednich, wartość 3.



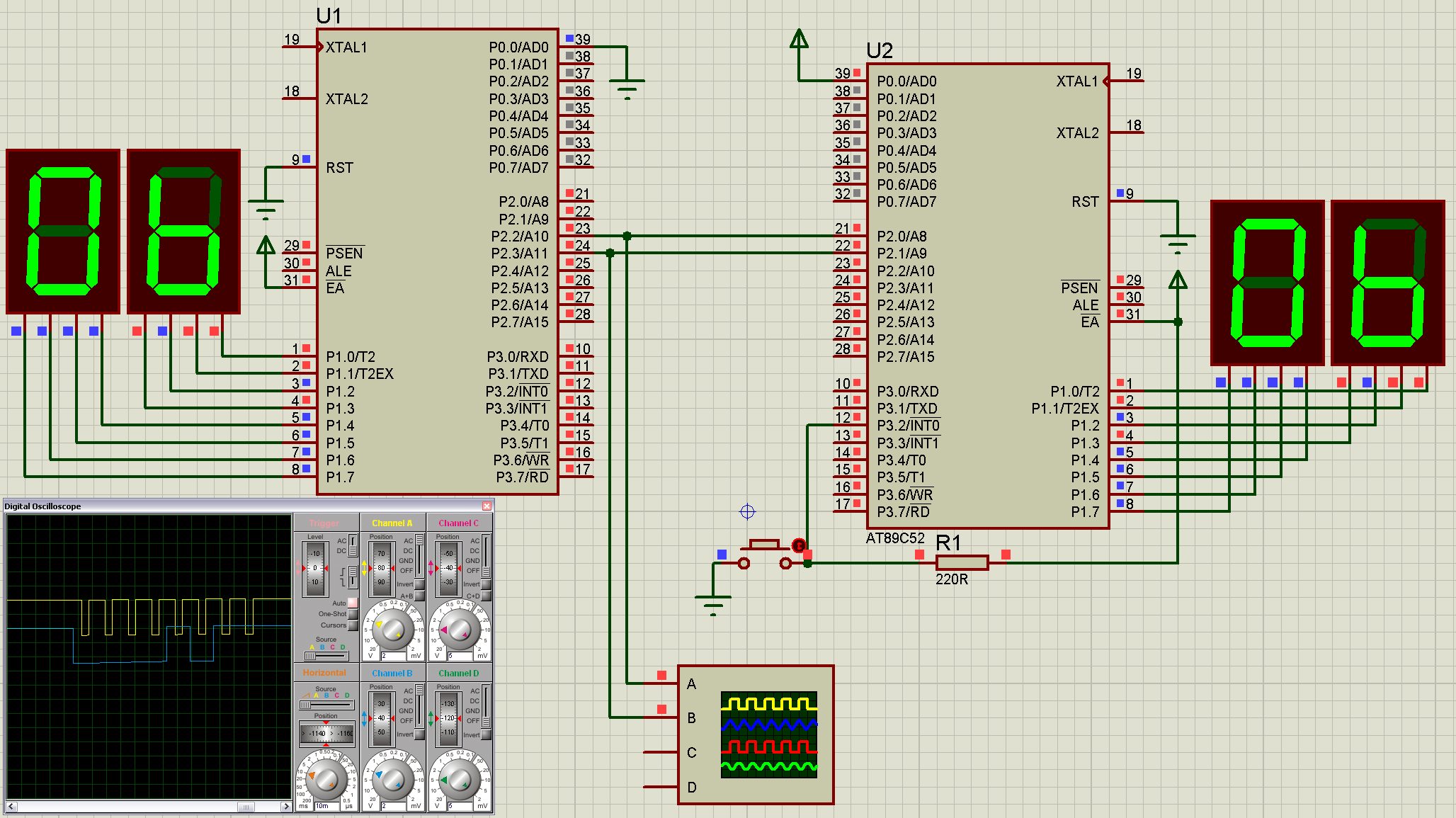
1. Czwarte wciśnięcie

Sygnał dodatni jest na 3 od końca bicie, czyli na wartości 23-1 co jest równe 4. To samo widać na obu wyświetlaczach.



1. Jedynaste wciśnięcie

Wartość 00001011 – kolejno 23 + 21 + 20 = 8 + 2 + 1 = 11. Heksadecymalnie jest to równe 0x0B, co widać na obu wyświetlaczach.



### Podsumowanie

W celu stworzenia skutecznej transmisji szeregowej należy zapewnić nie tylko połączenie do transportu danych, ale również połączenie do informowania o samym fakcie przesyłania. W tym celu w ramach powyższego układu należało stworzyć nie jeną, a dwie linie transmisyjne. W ramach nich master przesyłał do slave’a po pierwsze informacje o tym, że teraz przesyłana jest dana, po drugie, sama wartość danego bitu (zaczynając od najstarszego).

Gdybyśmy chcieli pominąć linię informującą o przesyłaniu danych, mogłyby zaistnieć 2 scenariusze:

Pierwszy: Slave nie wie, że dane są przesyłane i nic nie odczytuje.

Drugi: Slave myśli, że dane przesyłane są cały czas i stale odczytuje ciąg 1

Jak widać, oba te scenariusze są nieprawidłowe i należało je uniknąć.

Implementacja wspomnianego rozwiązania powoduje, że zaraz po odczytaniu danej przez mastera i wysłaniu jej do slave’a, mamy szeregową transmisję synchroniczną – potwierdzają to stale równe wartości na obu wyświetlaczach.