Systemy wbudowane

Lab2: Wytwarzanie i testowanie oprogramowania.

Bartnicki Mateusz, 59026, WCY20IX1N1

Data wykonania ćwiczenia: 15.05.2022 r.

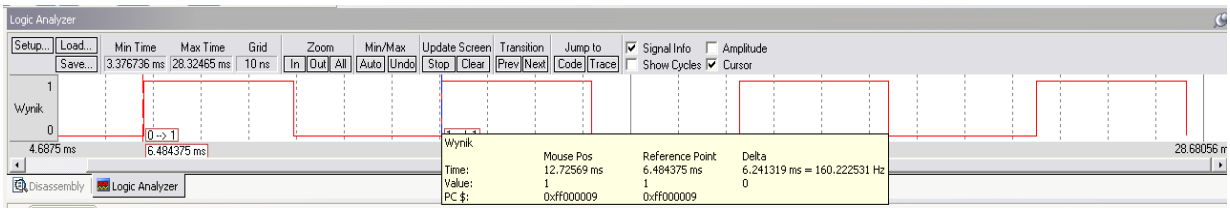
Wykonano zadania na ocenę dst, db i bdb.

Eksperyment w debugerze Keil, dotyczący pomiaru czasu trwania funkcji wprowadzającej opóźnienie:

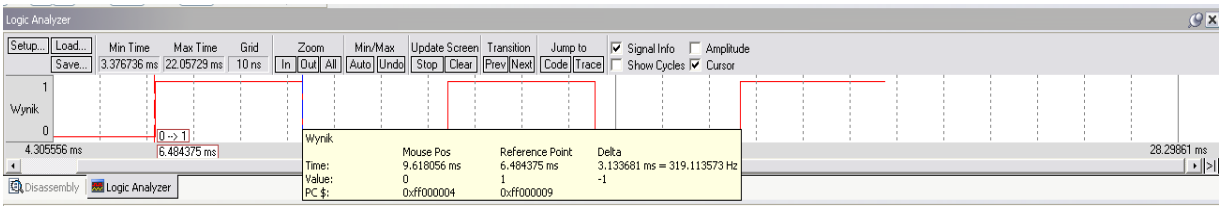
W pierwszej kolejności opracowano funkcję delay, która zgodnie z wytycznymi polegała na sprawdzeniu jak długo będzie wykonywać się „pusta" pętla for dla numer_w_dzienniku * 100 iteracji. Mam pierwszy numer w dzienniku, więc w moim przypadku pętla ta miała wykonać się 100 razy.

```
1  #include <REGX52.H>
2
3  void Delay(void);
4
5  void main (void)
6  {
7    unsigned char Wynik;
8    while(1)
9    {
10
11     Wynik = 0;
12     Delay();
13     Wynik = 1;
14     Delay();
15   }
16 }
17
18
19 void Delay(void)
20 {
21  unsigned char j;
22   unsigned char i;
23
24   for(i=0 ; i<10 ; i++)
25   {
26      for(j=0;j<10;j++)
27      {;}
28   }
29 }
```

W tym celu zadeklarowano zmienną pomocniczą o nazwie wynik, która zmieniała swoją wartość na przeciwną (z 0 na 1 lub 1 na 0) po każdym wywołaniu funkcji delay.

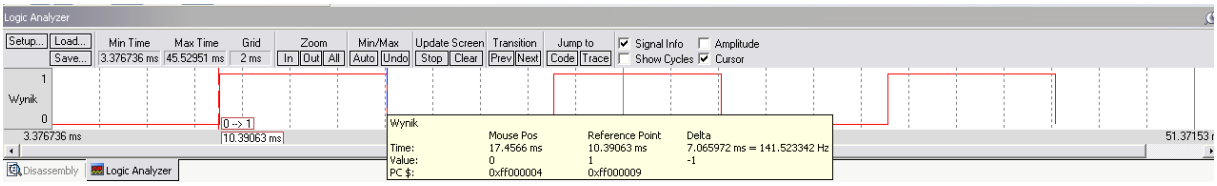


W pierwszej kolejności sprawdzono okres powtarzania impulsu, który wynosił 6,241319 ms (160,222531 Hz). Na tej podstawie oraz na podstawie opracowanej funkcji testowej można wywnioskować, że stan wysoki powinien utrzymywać się przez połowę okresu powtarzania impulsu, czyli ok. 3,1206595 ms.

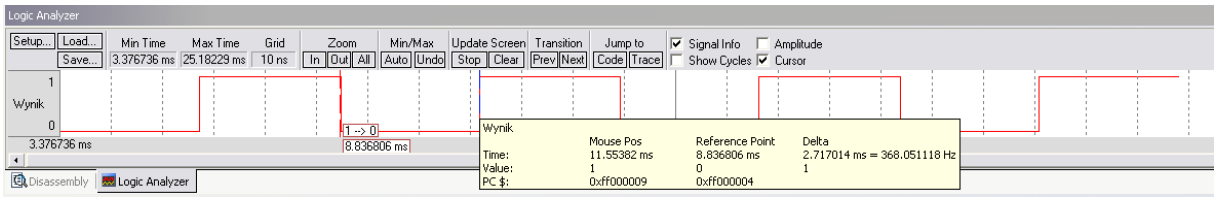


Następnie zmierzono czas trwania stanu wysokiego, który w debugerze Keil wynosiło 3,133681 ms (319,113573 Hz).

Dla przypadku, w którym i wynosiło 100, a j 1 czas trwania stanu wysokiego wynosił 7,065972 ms

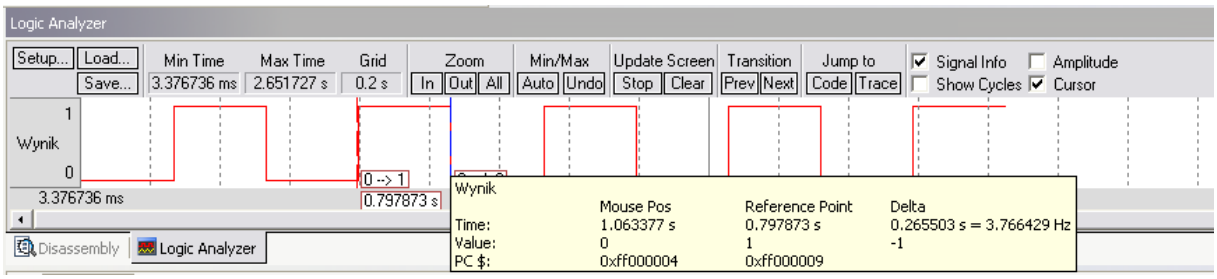


Dla odwrotnego przypadku, tj. i = 1, j = 100 czas ten wyniósł 2,717014 ms



Wyniki te można przyjąć za prawidłowe, rozbieżność wynika najprawdopodobniej z niedokładnego kliknięcia myszką celem wykonania pomiaru interesującego nas przedziału. Posiadając tą wiedzę można rozpocząć skalowanie timera.

Teoretycznie po podstawieniu wartości, dla 100i i 100j wartość opóźnienia powinna wynosić 0,28 s, jednakże w praktyce czas ten wynosi 0,26550s

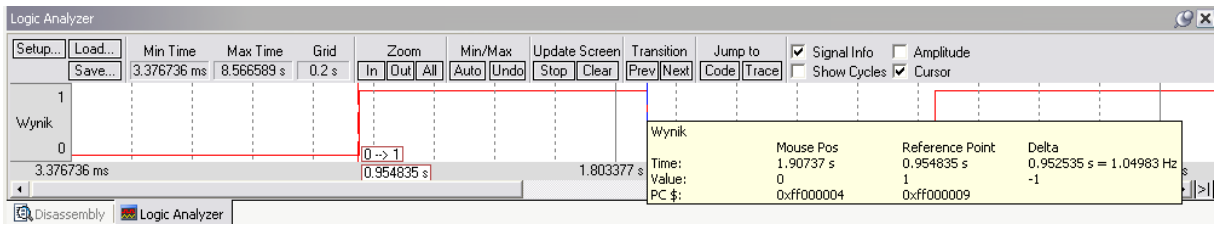


Dla potrzeb skalowania przyjęto i = 200, na której podstawie można obliczyć przybliżoną wartość j potrzebną do osiągnięcia czasu trwania równego 1 sekundzie.

$$1000ms = 200 * (0{,}07065972\ ms + x * 0{,}02717014\ ms)$$

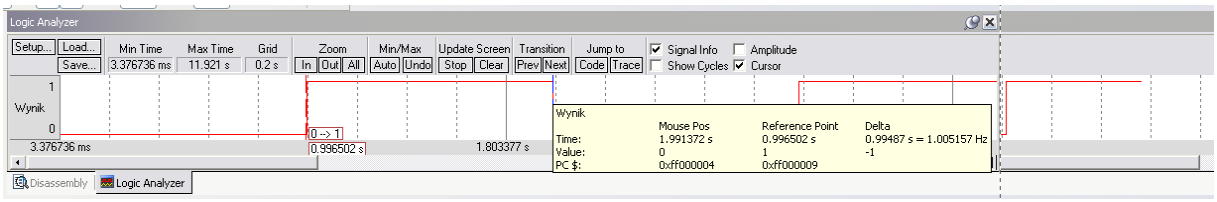
$$x \approx 181$$

Po podstawieniu wartości do programu otrzymujemy czas 0,952535s, który jest zadowalający, jednakże wartość tą możemy jeszcze bardziej „podciągnąć” bliżej 1.



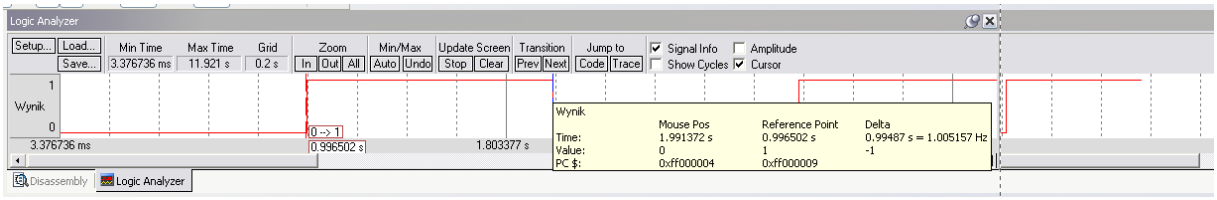
Wiemy, że czas wykonania 100 j wynosi 2,717014 ms, dla 200 j powinno to być 5,434028 ms. Brakująca nam różnica czasu do pełnej jedynki wynosi 47,465 ms. Dzieląc różnicę brakującego czasu przez czas wykonania 200 iteracji dla j otrzymujemy wynik wynoszący w przybliżeniu 8,734.

Po ręcznym zwiększeniu wartości j o 8 (zastosowano zaokrąglenie w dół) otrzymano następujący wynik:



Uzyskany wynik wyniósł 0,99487s, co jest wynikiem w pełni zadowalającym!

```
19 void Delay(void)
20 {
21   unsigned char j;
22   unsigned char i;
23
24    for(i=0 ; i<200 ; i++)
25    {
26       for(j=0;j<189;j++)
27       {;}
28    }
29 }
```

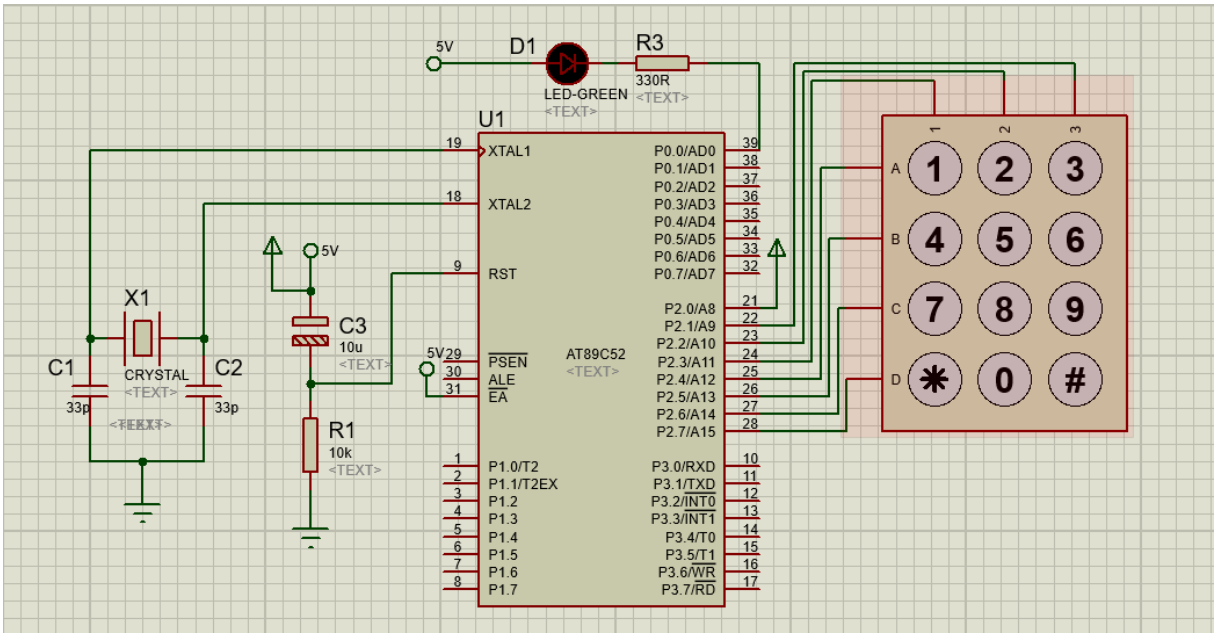


A) studenci o numerach nieparzystych skanują klawiaturę przez podawanie kolejno "wędrującego" zera na wiersze i sprawdzanie kolumn;

B) studenci o numerach parzystych skanują klawiaturę przez podanie kolejno "wędrującego" zera na kolumny i sprawdzanie wierszy.

Uwaga: numery wierszy nadajemy od góry: wiersze 1,2,3,4 to linie odpowiednio A, B, C, D. Numery kolumn od lewej, odpowiednio 1,2,3

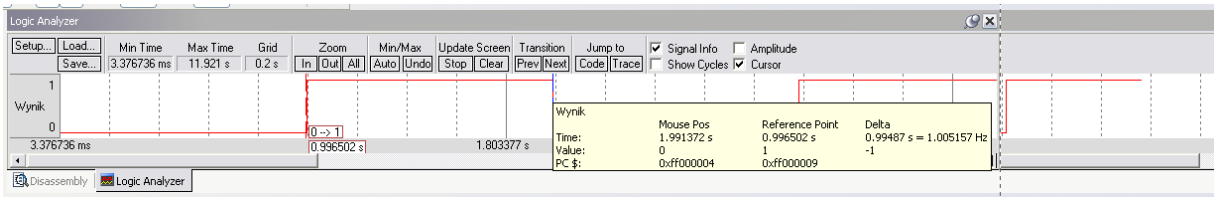
Na ocenę **dostatecznie** napisać dla schematu **Zad_5.pdsprj** program **lab2_1.c** w języku C, który w pętli wewnętrznej będzie kolejno obsługiwał klawiaturę w ten sposób, że po naciśnięciu dowolnego przycisku dioda LED rozbłyśnie jeden raz "przez 1 s" - czas przybliżony, uzyskany dzięki **przeprowadzonym osobiście i opisanym w sprawozdaniu pomiarom** z wykorzystaniem debugera Keil dla "Delay z podwójną pętlą for", początkowo dla **numer_w_dzienniku x 100** iteracji (patrz Profilowanie czasu wykonania fragmentu programu np. "pętli for" w środowisku Keil) a następnie **skalowaniu**. Po obliczeniach skalujących trzeba dodatkowo pokazać na zrzucie ekranu z debugera Keil, że uzyskano czas trwania opóźnienia około 1 sekundy. **Przytrzymanie naciśnietego klawisza nie powoduje powstawania kolejnych błysków**.



Zasadniczym problemem do rozwiązania było stworzenie odpowiedniej pętli wewnętrznej (funkcji), która wprowadzi opóźnienie w działaniu programu równe 1s oraz obsługa wejścia-wyjścia klawiatury wraz z przeciwdziałaniem powstawania wielu błysków podczas przytrzymania jednego z klawiszy.

Rozwiązanie dla pierwszego problemu zostało opisane w poprzedniej części sprawozdania. Aby uzyskać opóźnienie wynoszącą 1s zgodnie z narzuconymi

warunkami wykonania zadania należy stworzyć funkcję z dwoma pętlami for. Jedna z nich dokonuje iteracji zmiennej typu char do wartości 200, natomiast druga, zagnieżdżona w pierwszej pętla for iteruje zmienną typu char do wartości 189.



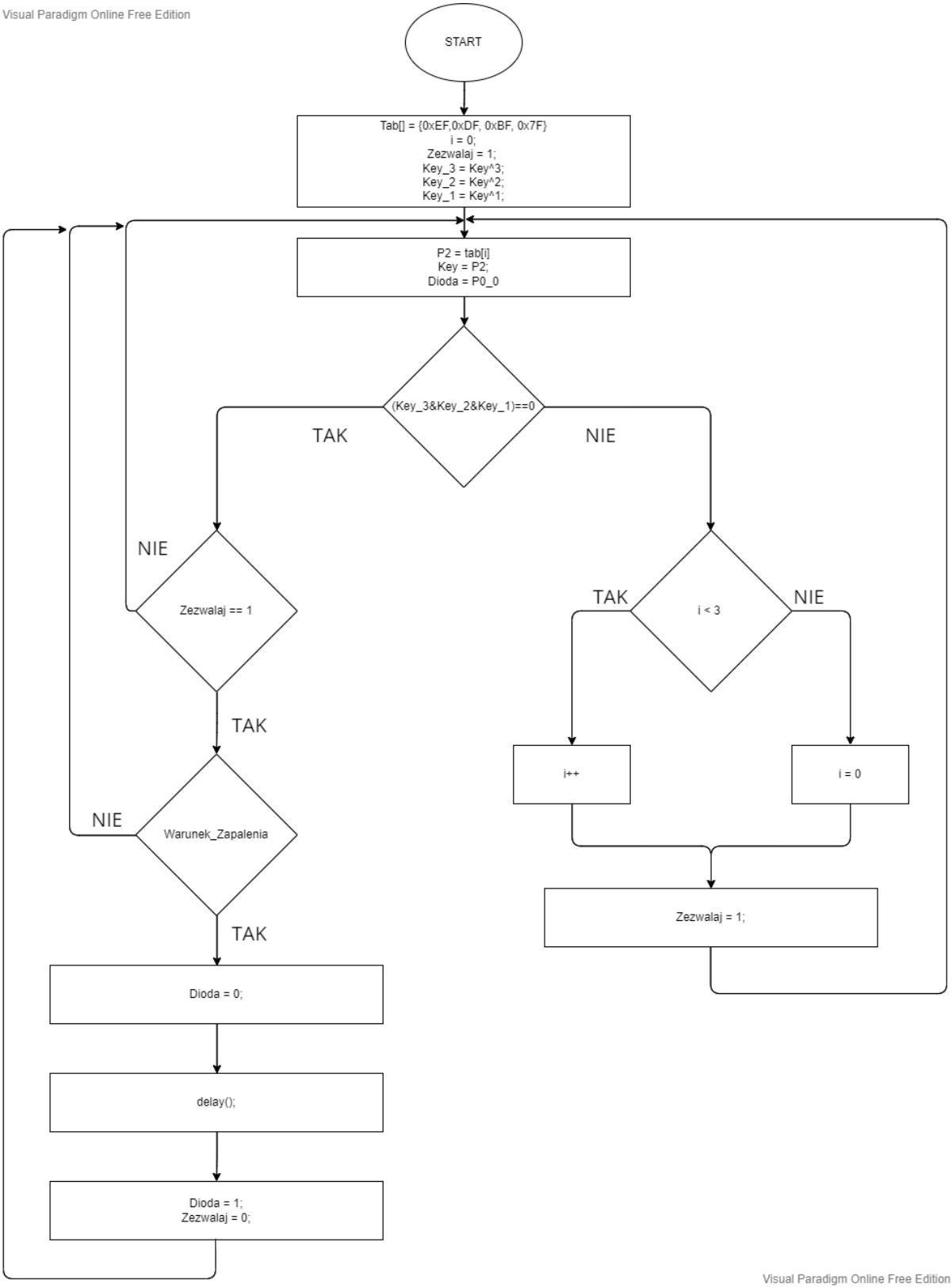
Uwaga: Zmienna Wynik była wykorzystywana w trakcie skalowania czasu trwania stanu wysokiego i nie występuje w praktycznej realizacji problemu;

Kolejnym problemem była odpowiednia obsługa systemu we-wy dla klawiatury. Zgodnie z założeniami (mam numer nieparzysty) skanowanie klawiatury następuje przez podawanie „wędrującego" zera po wierszach i sprawdzanie jego wystąpienia w kolumnach. Do sprawdzania wierszy wykorzystana jest 4 elementowa tablica przechowująca adresy portu P2, do których podłączony jest dany wiersz. Kolejno 0xEF,0xDF, 0xBF, 0x7F odpowiada 1, 2, 3 i 4 wierszowi. Do sprawdzania zmian w kolumnach wykorzystany jest warunek (Key_3&Key_2&Key_1)==0, gdzie dla każdego z kluczy obliczana jest wartość kolejno Key (zmienna key przyjmuje na zmianę jedną z 4 wartości zapisanych w powyższej tablicy) xor 3, 2, 1. W trakcie oczekiwanie układu na działanie użytkownika warunek ten nie jest spełniony, natomiast po naciśnięciu jednego z guzików następuje zapalenie diody. Przeciwdziałanie powstawania wielu błysków podczas przytrzymania jednego z klawiszy realizowane jest poprzez flagę o nazwie zezwalaj. Domyślnie jest ona ustawiona na 1, jednakże po wciśnięciu klawisza jej wartość zmieniana jest na 0 przez co nie następuje ponowne zapalenie diody. Flaga przyjmuje wartość 1 dopiero po puszczeniu przycisku. Zapalenie diody następuje, gdy spełniony jest następujący warunek:

Key == 0xE7 || Key == 0x7D || Key == 0xEB || Key == 0xED || Key == 0xD7 || Key == 0xDB || Key == 0xDD || Key == 0xB7 || Key == 0xBB || Key == 0xBD || Key == 0x77 || Key == 0x7B || Key == 0x7D

Zakodowane są w nim odpowiednio symbole 1, 2, 3, 4, 5, 6, 7, 8, 9, *, 0, # z klawiatury.

Schemat blokowy lab2_1.c opracowany za pomocą Visual Paradigm Online:

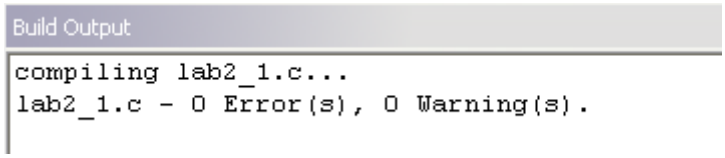


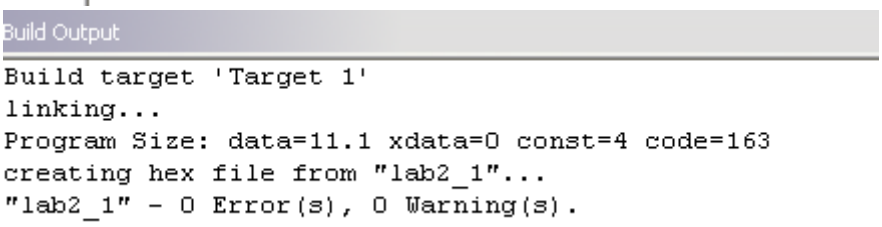
Treść programu:

lab2_1.c

```
#include <REGX52.H>
#define Dioda P0_0
#define numer 1
#define Warunek_Zapalenia Key == 0xE7 || Key == 0x7D || Key == 0xEB || Key == 0xED || Key == 0xD7 || Key == 0xDB || Key == 0xDD || Key == 0xB7 || Key == 0xBB || Key == 0xBD || Key == 0x77 || Key == 0x7B || Key == 0x7D

unsigned char code Tab[] = (0xEF,0xDF, 0xBF, 0x7F);
unsigned char bdata Key;
sbit Key_3 = Key^3;
sbit Key_2 = Key^2;
sbit Key_1 = Key^1;
void delay(void);

void main(void)
{
 unsigned char data i = 0;
 bit Zezwalaj = 1;     // flaga, zabezpieczajaca przed wielokrotnym wejsciem do procedury wykonawczej przy przytrzymaniu klawisza
 while(1)
 {
   P2 = Tab[i];
   Key = P2;
   if ((Key_3&Key_2&Key_1)==0)
   {
     if (Zezwalaj == 1)
     {
       if (Warunek_Zapalenia)
       {
         Dioda = 0;
         delay();
         Dioda = 1;
         Zezwalaj = 0;        // Zablokuj wejscie do procedur obslugi klawiszy
       }
     }
   }
   else
   {
     if (i<3)
     {
       i++;
     }
     else
     {
       i = 0;
     }
     Zezwalaj = 1;     // odblokowanie zezwalania na wejscie do procedur obslugi klawiszy
   }
 }
}

void delay(void)
{
 unsigned char j;
 unsigned char i;

 for(i=0 ; i<200 ; i++)
 {
   for(j=0;j<189;j++)
   {;}
 }
}
```

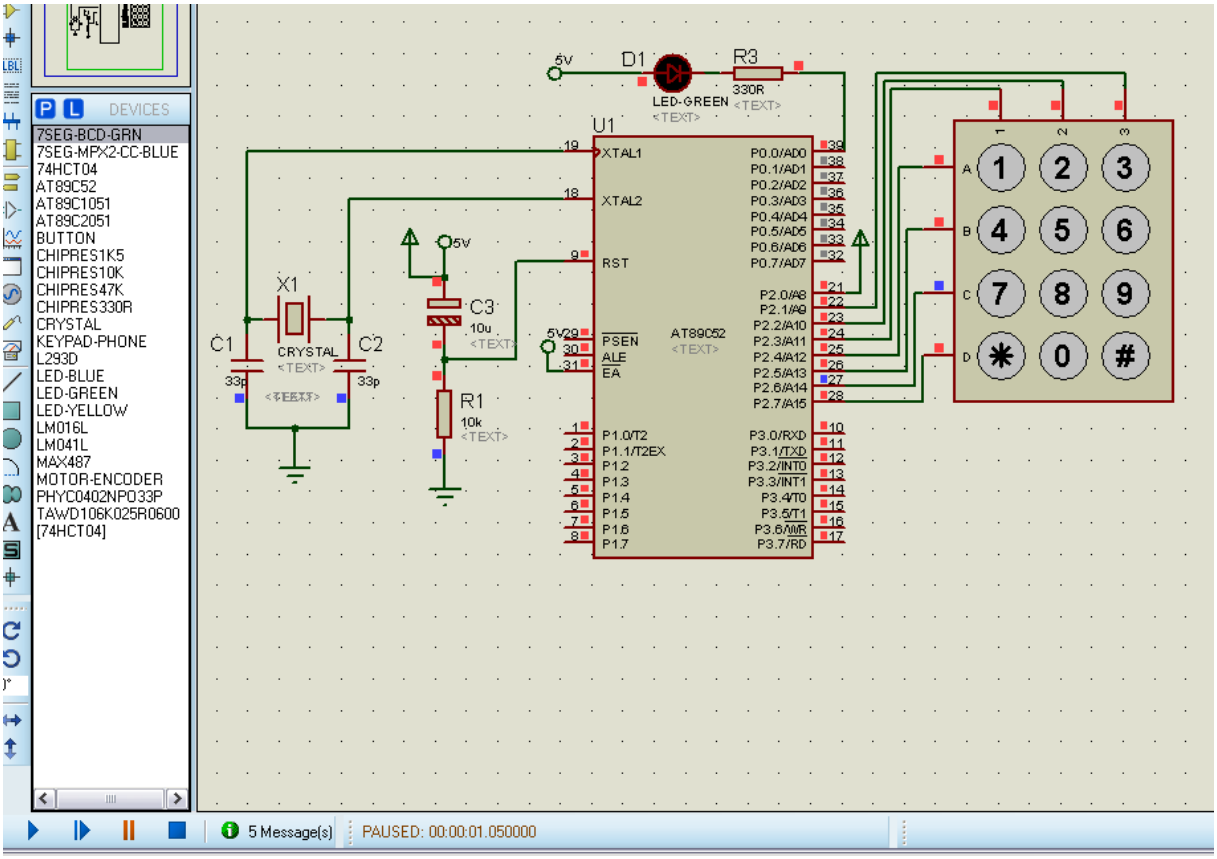
Zrzuty ekranu z kompilacji i linkowania pokazujące brak błędów oraz rozmiar kodu i danych:



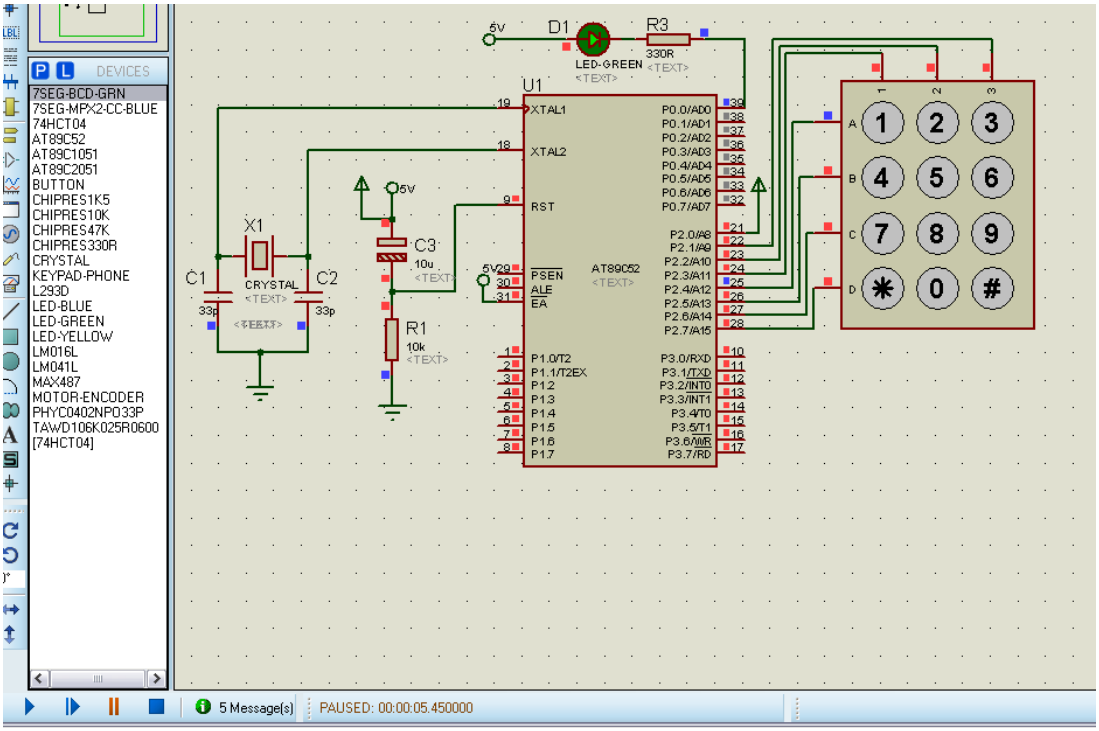


Zrzuty ekranu z symulatora Proteus:

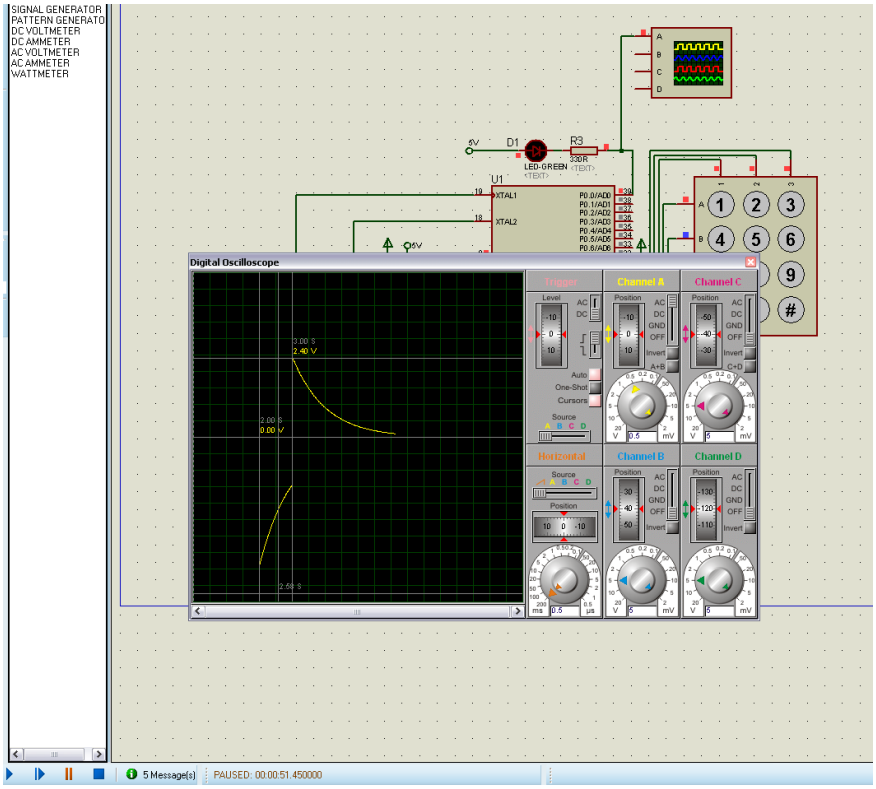
Domyślnie dioda D1 jest zgaszona.



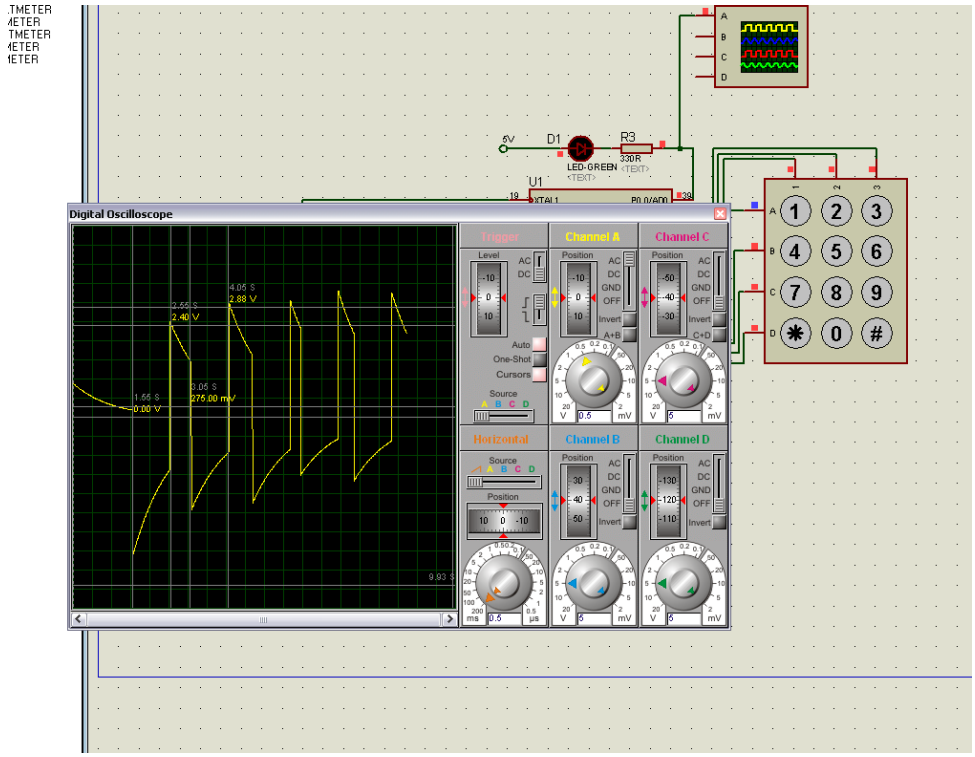
Naciśnięcie jednego z przycisków powoduje zapalenie się diody. Dioda świeci się przez około sekundę, a przytrzymanie przycisku nie powoduje migania diody.



Wbudowany oscyloskop w symulatorze Proteus pokazuje czas trwania zapalenia diody równo na 1 s, zgodnie z założeniami zadania.



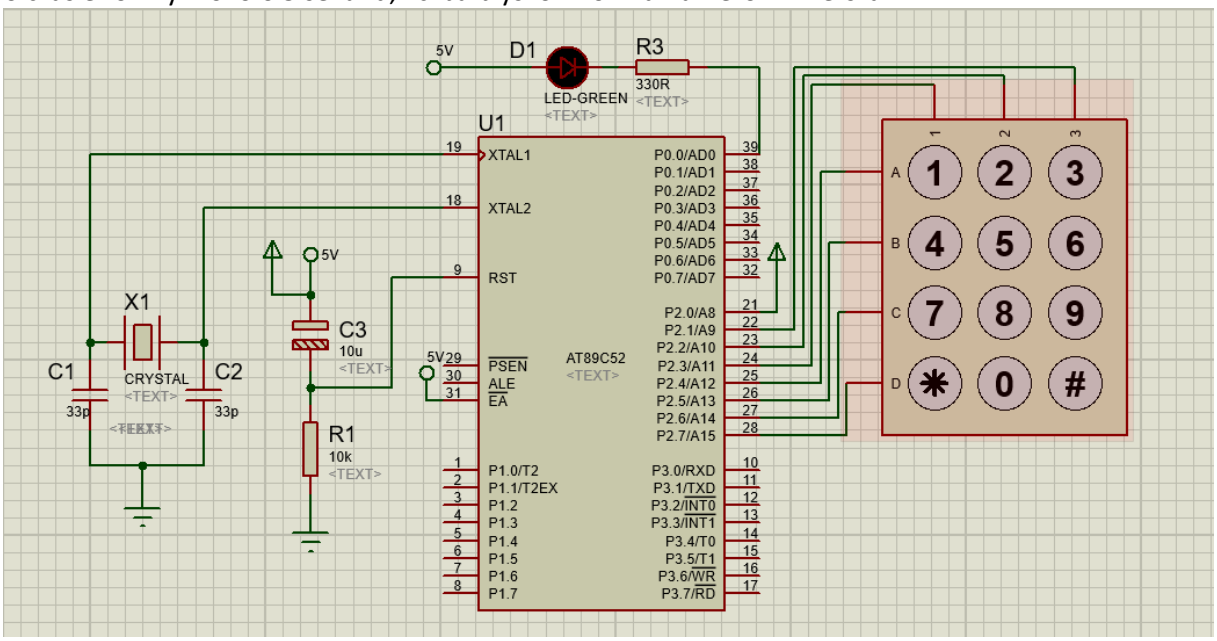
Wielokrotne naciskanie guzików podczas zapalenia diody nie powoduje przedłużenia czasu jej zapalenia, w każdym przypadku będzie się ona świeciła ok 1s. Zaświeci się ona drugi raz dopiero po tym, jak już zostanie obsłużone poprzednie zapalenie diody.



Na ocenę **dobrze** zrobić to, co na **dostatecznie, oraz ponadto** napisać dla Schematu **Zad_5.pdsprj** program **lab2_2.c** w języku C, który w pętli wewnętrznej będzie kolejno obsługiwał klawiaturę w ten sposób, że po naciśnięciu dowolnego przycisku mikrokontroler wywoła odpowiadającą mu sekwencję błysków diody LED (każdy trwający około 1s) tak, że:

a) dla studentów o numerach nieparzystych będzie to liczba błysków równa numerowi wiersza, przerwa o czasie równym około 3 sekundy, liczba błysków równa numerowi kolumny.

b) dla studentów o numerach parzystych będzie to liczba błysków równa numerowi kolumny, przerwa o czasie równym około 5 sekund, liczba błysków równa numerowi wiersza.



Zasadniczym problemem w realizacji zadania było odpowiednie zaprogramowanie systemu obsługi wejścia-wyjścia klawiatury, tak aby mikrokontroler wiedział, który znak w odpowiadającym mu wierszu i kolumnie został wciśnięty. Miało to spowodować zapalenie diody LED w ilości równej numerowi wiersza, a po 3s przerwie miało zapalić diodę LED w liczbie równej kolumnie.

W celu wprowadzenia 1s błysku wykorzystano taką samą funkcję wprowadzającą opóźnienie jak w pierwszym ćwiczeniu. Opóźnienie trwające 3s uzyskano poprzez wykorzystanie pętli for, w której funkcja opóźniająca wywoływana jest odpowiednią ilość razy zapewniając 3 sekundowe opóźnienie.

Zapobieganie dodatkowym błyskom diody spowodowane długotrwałym przyciśnięciem przycisku przez użytkownika realizowane jest w ten sam sposób jak zostało to opisane w pierwszym zadaniu. Wykorzystana do tego została flaga, która zmienia swoją wartość na przeciwną w momencie wciśnięcia przycisku i zmienia ona swoją wartość dopiero po puszczeniu tego przycisku.

Zasadniczy postawiony problem został rozwiązany poprzez wykorzystanie tablicy przechowującej następujące wartości:

unsigned char code Tab[] = {0xEF,0xDF, 0xBF, 0x7F};

Wartości te odpowiadają kolejno wierszom znaków na klawiaturze. Dzięki indeksowi z tej tablicy mamy wiedzę na temat tego ile razy dioda powinna błysnąć dla danego wiersza.

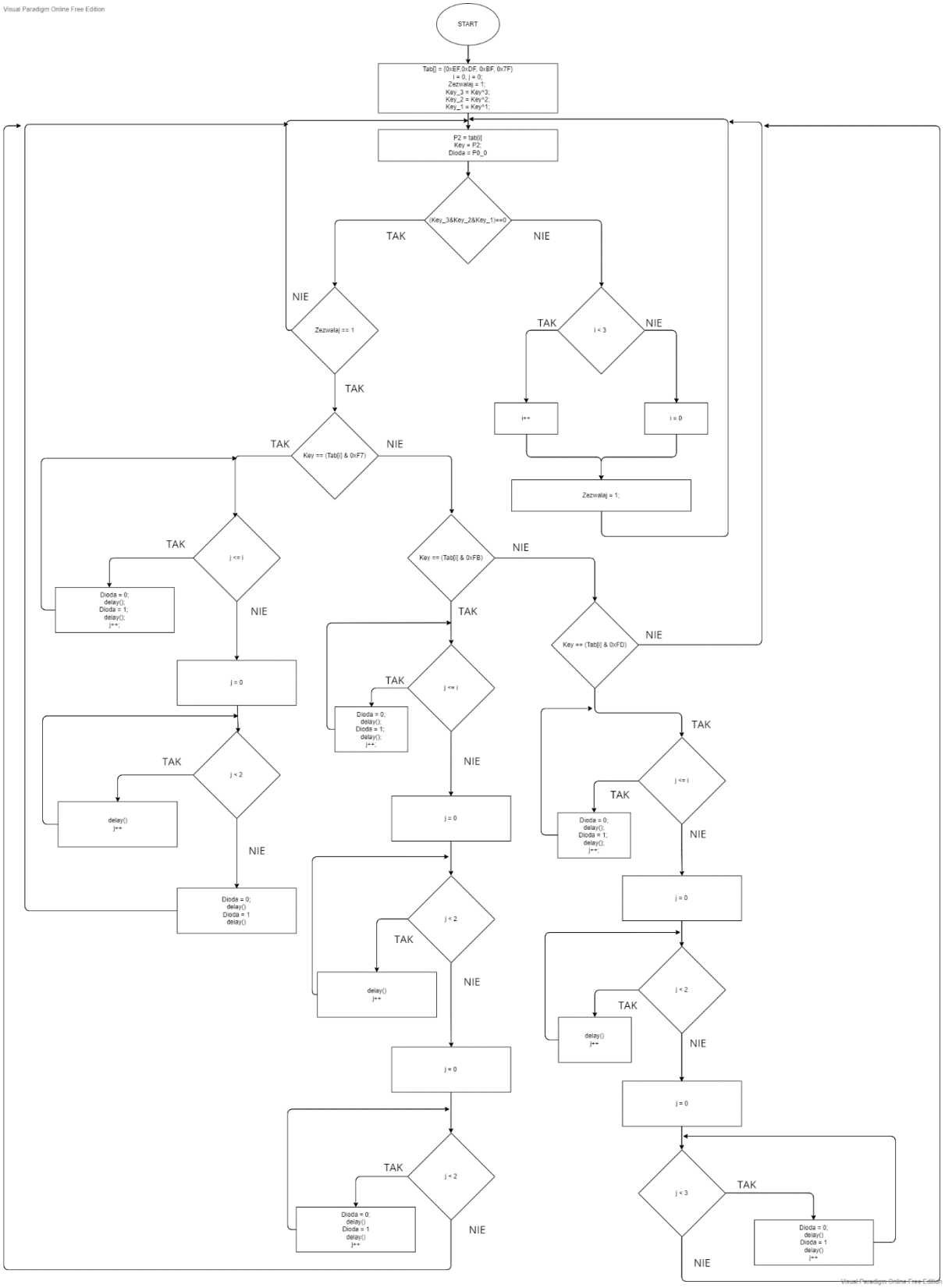
Sprawdzenie ile razy dioda ma błysnąć dla danej kolumny realizowane jest poprzez wykorzystanie działania logicznego AND &. Program zawiera instrukcje warunkową else if, która sprawdza kolejno następujące warunki:

Key == (Tab[i] & 0xF7), Key == (Tab[i] & 0xFB), Key == (Tab[i] & 0xFD).

Pierwsza wartość odpowiada pierwszej kolumnie, druga drugiej i trzecia trzeciej. Spełnienie któregoś z powyższych warunków daje nam informację w którym wierszu oraz w której kolumnie znajduje się dany symbol.

Zapalanie diod w odpowiedniej ilości realizowane jest w pierwszej kolejności przez pętle for, gdzie wartość indeksu i wyznacza nam ilość błysków do zrealizowania, następnie następuje 3 sekundowa przerwa po czym ilość błysków dla kolumn również jest realizowana przez wykorzystanie pętli for.

Schemat blokowy lab2_2.c opracowany za pomocą Visual Paradigm Online:

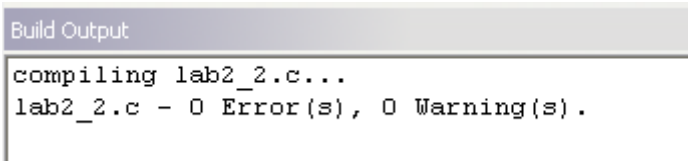


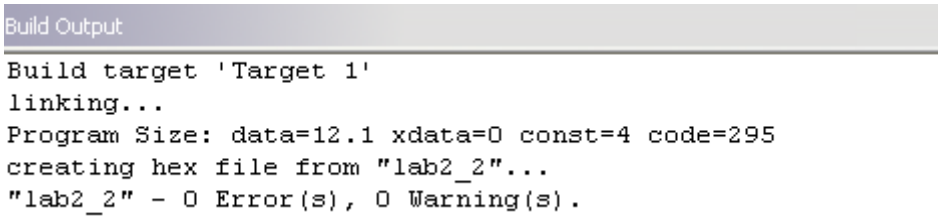
Treść programu:

lab2_2.c

```c
#include <REGX52.H>
#define Dioda P0_0
#define numer 1

unsigned char code Tab[] = {0xEF,0xDF, 0xBF, 0x7F};
unsigned char bdata Key;
sbit Key_3 = Key^3;
sbit Key_2 = Key^2;
sbit Key_1 = Key^1;
void delay(void);

void main(void)
{
  unsigned char data i = 0;
  unsigned char data j = 0;
  bit Zezwalaj = 1;     // flaga, zabezpieczajaca przed wielokrotnym wejsciem do procedury wykonawczej przy przytrzymaniu klawisza
  while(1)
  {
    P2 = Tab[i];
    Key = P2;
    if ((Key_3&Key_2&Key_1)==0)
    {
      if (Zezwalaj == 1)
      {
        Zezwalaj = 0;
        if (Key == (Tab[i] & 0xF7))
        {
          for(j = 0; j <= i ; j++)
          {
          Dioda = 0;
          delay();
          Dioda = 1;
          delay();
          }
          for(j = 0; j < 2 ; j++)
          {
            delay();
          }
          Dioda = 0;
          delay();
          Dioda = 1;
          delay();
        }
        else if (Key == (Tab[i] & 0xFB))
        {
          for(j = 0; j <= i ; j++)
          {
          Dioda = 0;
          delay();
          Dioda = 1;
          delay();
          }
          for(j = 0; j < 2 ; j++)
          {
            delay();
          }
          for(j = 0 ; j < 2 ; j++)
          {
          Dioda = 0;
          delay();
          Dioda = 1;
          delay();
          }
        }
        else if (Key == (Tab[i] & 0xFD))
        {
          for(j = 0; j <= i ; j++)
          {
          Dioda = 0;
          delay();
          Dioda = 1;
          delay();
          }
          for(j = 0; j < 2 ; j++)
          {
            delay();
          }
          for(j = 0 ; j < 3 ; j++)
          {
          Dioda = 0;
          delay();
          Dioda = 1;
          delay();
          }
        }
      }
    }
    else
    {
      if (i<3)
      {
        i++;
      }
      else
      {
        i = 0;
      }
      Zezwalaj = 1;     // odblokowanie zezwalania na wejscie do procedur obslugi klawiszy
    }
  }
}

void delay(void)
{
  unsigned char j;
  unsigned char i;

   for(i=0 ; i<200 ; i++)
   {
      for(j=0;j<189;j++)
      {;}
   }
}
```

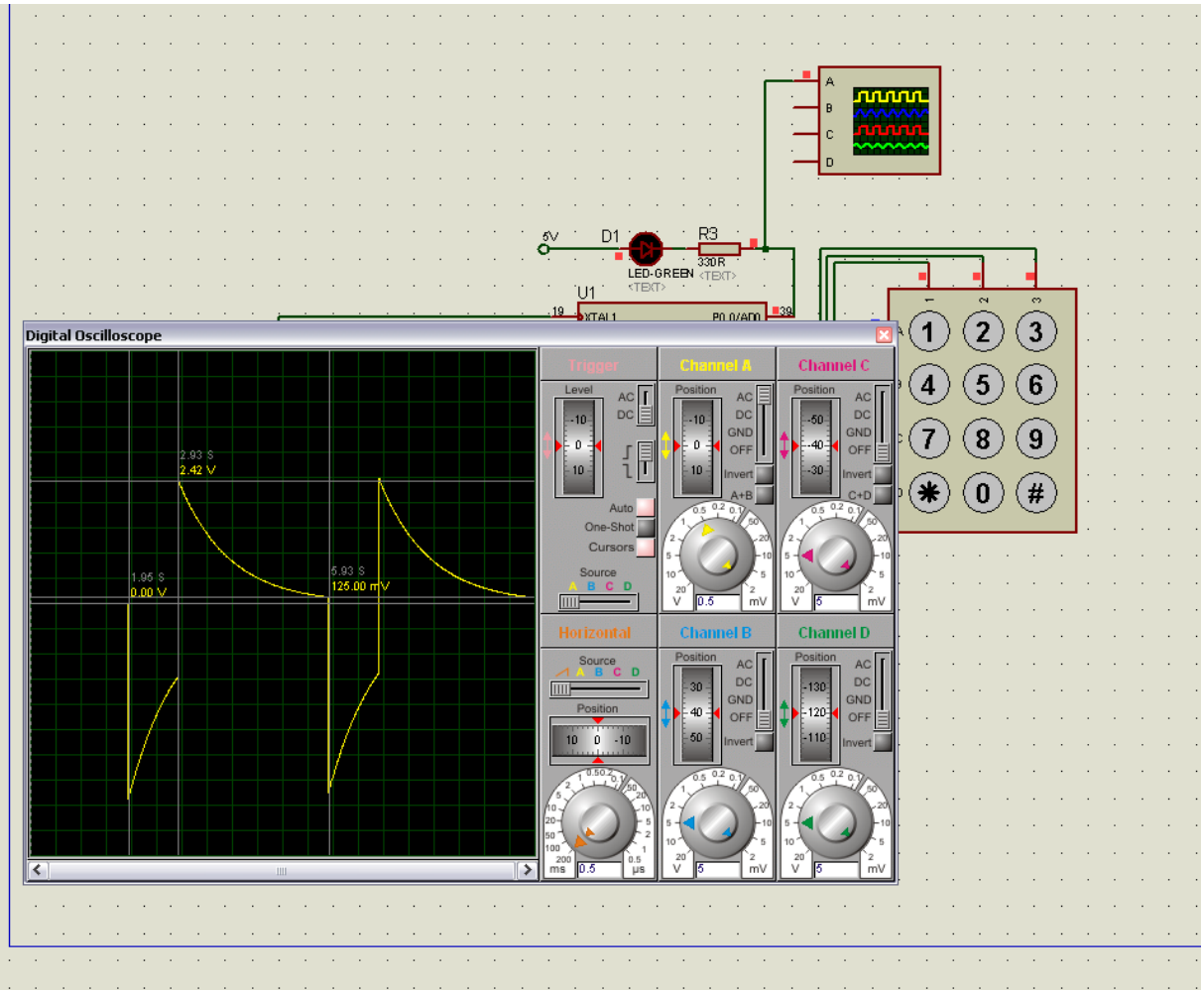
Zrzuty ekranu z kompilacji i linkowania pokazujące brak błędów oraz rozmiar kodu i danych:



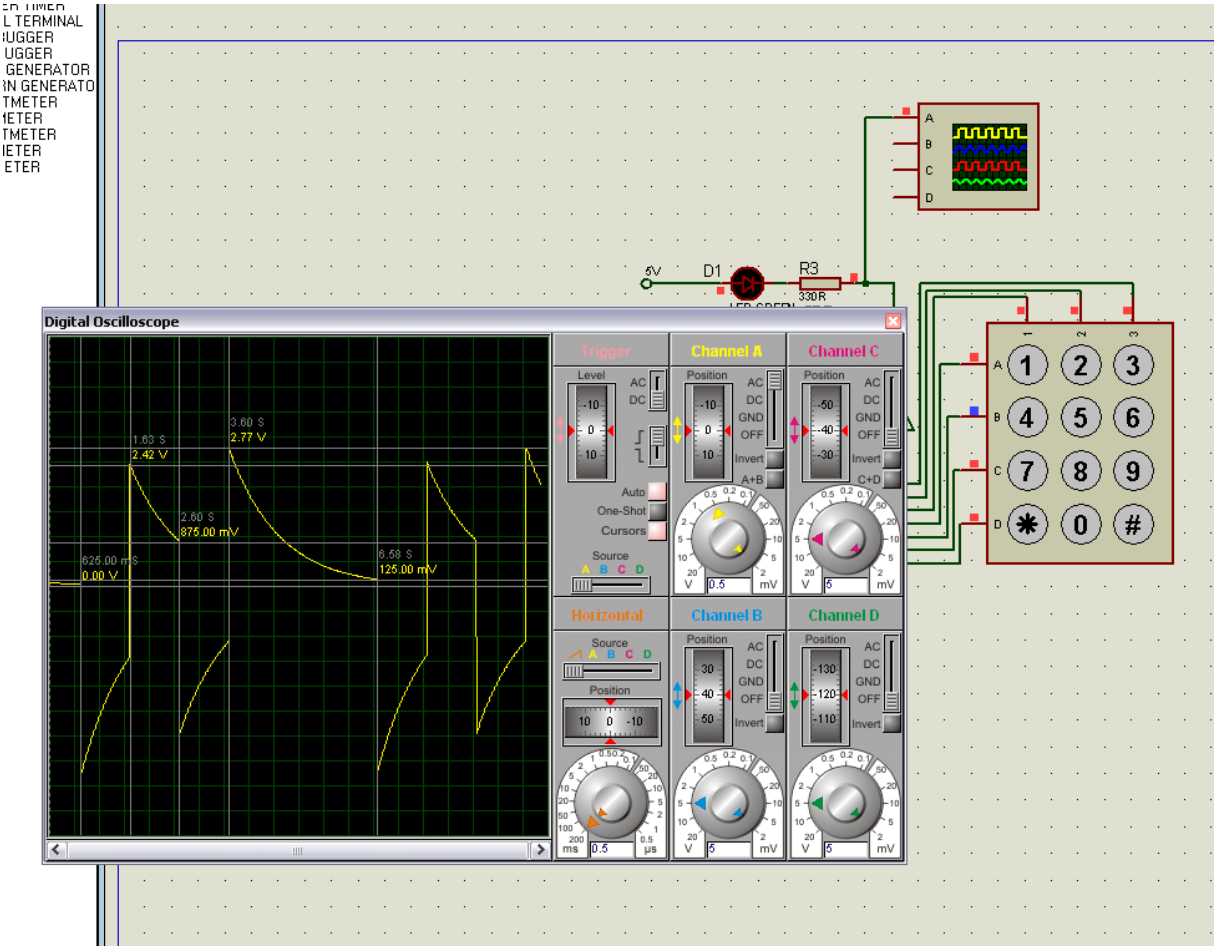


Zrzuty ekranu z symulatora Proteus:

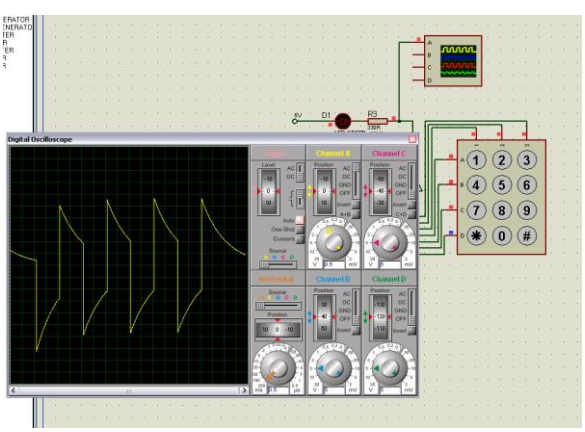
W celu sprawdzenia poprawności realizacji zadania w symulatorze Proteus, do obowiązującego schematu dodano oscyloskop, który podłączony został do pinu P0.0. Pozwala to na poprawne udokumentowanie zmian stanu diody. W pierwszej kolejności wciśnięto przycisk 1. Zgodnie z wytycznymi dioda powinna zapalić się raz na 1 sekundę, następnie zgasnąć na 3 s i ponownie zapalić się na sekundę. Poprawność realizacji potwierdza poniższy zrzut ekranu.

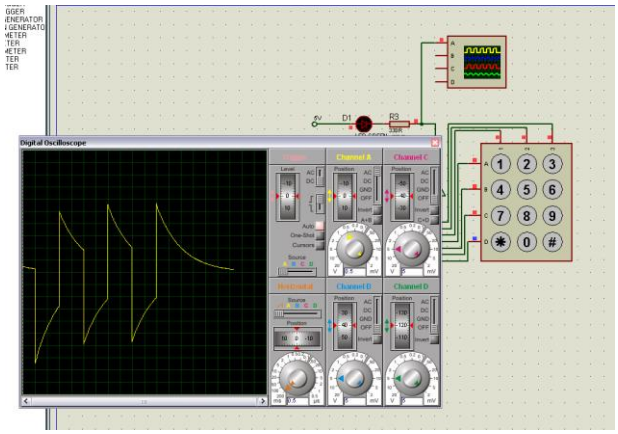


Następnie wciśnięto klawisz 5. Dioda powinna zapalić się 2 razy na 1 sekundę co 1 sekundę oraz następnie 2 razy na sekundę po 3 sekundowej przerwie. Odczytany przebieg z oscyloskopu potwierdza pomyślną realizację eksperymentu:



Na samym końcu wciśnięto klawisz #. Dioda najpierw zapaliła się 4 razy, a następnie 3. Potwierdza to poprawność działania programu.





Na ocenę **bardzo dobrze** zrobić to, co **na dobrze, oraz ponadto** napisać dla **schematu Zad_5.pdsprj** program **lab2_3.c** w języku C, który zrealizuje program sterujący zamka szyfrowego, pracującego w ten sposób, że wprowadzenie poprawnych 4 ostatnich cyfr numeru indeksu studentki/ studenta:

A) dla studentów o numerach nieparzystych - i znaku "krzyżyka" - spowoduje "otwarcie zamka" i zapalenie diody LED tyle razy, ile wynosi **ostatnia** cyfra numeru indeksu **(jeśli ta cyfra jest równa zero to ma mignąć 2 razy)**.

B) dla studentów o numerach parzystych - i znaku "gwiazdki" - spowoduje "otwarcie zamka" i zapalenie diody LED tyle razy, ile wynosi **przedostatnia**, cyfra numeru indeksu **(jeśli ta cyfra jest równa zero to ma mignąć 2 razy)**.

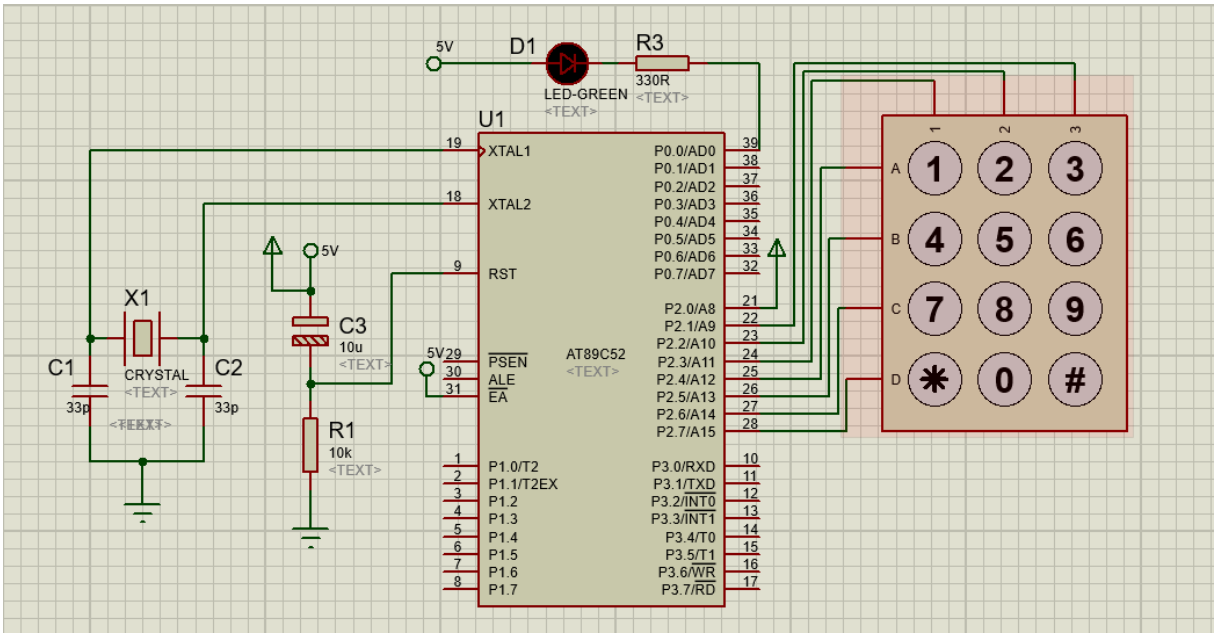
Uwaga: wprowadzenie poprawnych 4 ostatnich cyfr numeru indeksu studentki/ studenta, ale zakończenie ich **odwrotnym niż przydzielony znakiem** ("gwiazdka" dla nieparzystych/ "krzyżyk" dla parzystych) powoduje **sygnalizację błędu przez dwie potrójne serie błysków, rozdzielone pauzą (pauza trwa przez czas wybrany przez Autorkę/ Autora)**.

Wprowadzenie innych sekwencji ma być ignorowane. Każdy błysk i zgaszenie ma trwać po około 0.5s.

Program musi **bez resetowania** poprawnie obsłużyć następujący scenariusz zdarzeń:
a) wprowadzenie nieprawidłowego kodu (np. czterech zer), zatwierdzone poprawnym znakiem - brak sygnalizacji;
b) wprowadzenie poprawnego kodu, zatwierdzonego odwrotnym znakiem - sygnalizacja błędu;
c) wprowadzenie poprawnego kodu, zatwierdzonego właściwym znakiem - sygnalizacja "otwarcia zamka";
d) powtórne wprowadzenie nieprawidłowego kodu (np. czterech dziewiątek), zatwierdzonych odwrotnym znakiem - brak sygnalizacji;

e) powtórne wprowadzenie poprawnego kodu, zatwierdzonego właściwym znakiem - sygnalizacja "otwarcia zamka";

Jeżeli program zrealizuje scenariusz a - c, a potem "się zgubi" = ocena db+.



Zasadniczym problemem do zrealizowania była odpowiednia obsługa systemu wejścia-wyjścia klawiatury, tak aby mikrokontroler rozpoznawał wpisywane znaki, zapamiętał je oraz porównał z zapisanym w pamięci hasłem. Dodatkowo należało zmodyfikować funkcję obsługi opóźnienia programu.

W tym celu wprowadzono zmiany do funkcji sterującej trwaniem opóźnienia. Do tej funkcji przekazywana jest obecnie zmienna, która domyślnie ma wartość 0. W przypadku obsługi sygnalizacji błędu wartość ta jest zmieniana na 1. Zmiana wartości na 1 powoduje wydłużenie czasu trwania opóźnienia z 0.5s do 1s.

Zapobieganie dodatkowym błyskom diody spowodowane długotrwałym przyciśnięciem przycisku przez użytkownika realizowane jest w ten sam sposób jak w dwóch poprzednich zadaniach. Wykorzystana do tego została flaga, która zmienia swoją wartość na przeciwną w momencie wciśnięcia przycisku i zmienia ona swoją wartość dopiero po puszczeniu tego przycisku.

Rozwiązanie zasadniczego problemu polegało na zadeklarowaniu tablicy typu unsigned char, w której przechowywane jest hasło. W moim przypadku jest to:

Haslo[] = {0xBD, 0x7B, 0xEB, 0xDD}; // Haslo – 9026

Dodatkowo została też zadeklarowana druga, czteroelementowa tablica, której rolą jest zapamiętywanie wartości pojawiających się na pinie P2 w przypadku wciskania przycisków przez użytkownika. Na początku tablica ta zapamiętuje 4 dowolne symbole, które zostały naciśnięte przez użytkownika, po czym program czeka na podanie symbolu sterującego (podanie cyfry jako symbolu sterującego powoduje wyczyszczenie zawartości zapisanych do tablicy). Po podaniu symbolu sterującego (* lub #) następuje porównywanie zapisanych symboli z hasłem. W przypadku, gdy jeden z symboli nie jest taki sam jak odpowiadający mu symbol w haśle to dalsze sprawdzanie jest przerywane i zmienna blad przyjmuje wartość równą 1.

Dla * gdy zmienna blad ma wartość 1 – następuje zresetowanie wartości znajdujących się w tablicy przechowywującej symbole oraz zmiana wartości zmiennej j służącej do „wędrowania” po tablicy na 0. Dioda nic nie sygnalizuje.

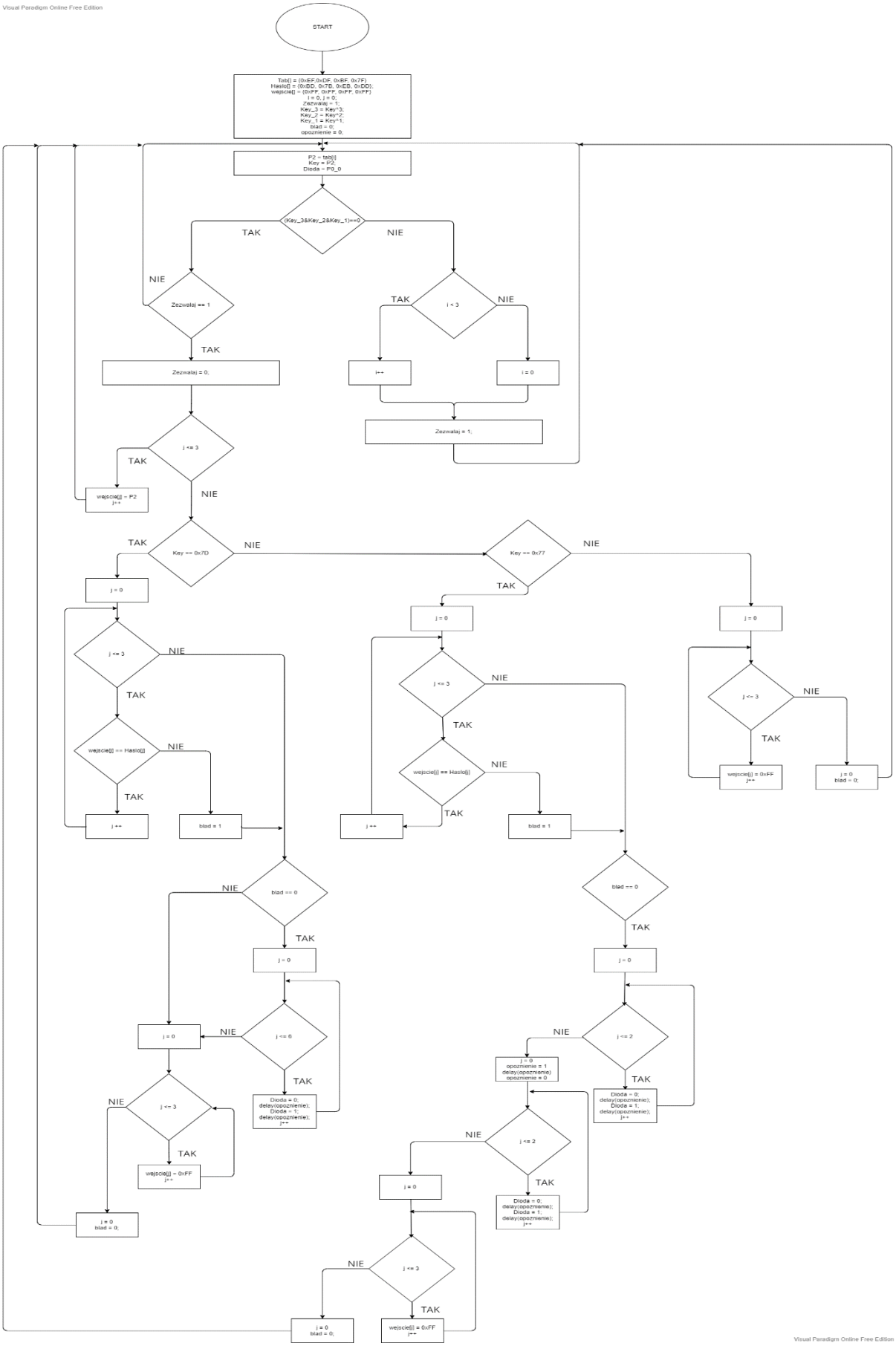
Dla * gdy zmienna blad ma wartość 0 – następuje zapalenie diody 3 razy (w odstępach po 0.5s), przerwa na 1,5s oraz ponowne zapalenie diody 3 razy. Potem następuje zresetowanie wartości przechowywanych w tablicy oraz zmiennej j.

Dla # gdy zmienna blad ma wartość 1 - następuje zresetowanie wartości przechowywanych w tablicy oraz zmiennej j. Dioda nic nie sygnalizuje.

Dla # gdy zmienna blad ma wartość 0 – następuje otwarcie zamka – zapalenie się diody 6 razy pod rząd w odstępach po 0.5s.

W każdym innych przypadku - następuje zresetowanie wartości przechowywanych w tablicy oraz zmiennej j. Dioda nic nie sygnalizuje.

Schemat blokowy lab3_2.c opracowany za pomocą Visual Paradigm Online:



Treść programu:

lab2_3.c

```
#include <REGX52.H>
#define Dioda P0_0
#define numer 1

unsigned char code Tab[] = {0xEF,0xDF, 0xBF, 0x7F};
unsigned char Haslo[] = {0xBD, 0x7B, 0xEB, 0xDD}; // Haslo - 9026
unsigned char wejscie[] = {0xFF, 0xFF, 0xFF, 0xFF};
bit opoznienie = 0;
unsigned char bdata Key;
sbit Key_3 = Key^3;
sbit Key_2 = Key^2;
sbit Key_1 = Key^1;
void delay(bit wolniej);
bit blad = 0;

void main(void)
{
  unsigned char data i = 0;
  unsigned char data j = 0;
  bit Zezwalaj = 1;     // flaga, zabezpieczajaca przed wielokrotnym wejsciem do procedury wykonawczej przy przytrzymaniu klawisza
  while(1)
  {
    P2 = Tab[i];
    Key = P2;
    if ((Key_3&Key_2&Key_1)==0)
    {
      if (Zezwalaj == 1)
      {
        Zezwalaj = 0;
        if(j <= 3)
        {
          wejscie[j] = P2;
          j++;
        }
        else
        {
          if(Key == 0x7D)
          {
            for(j = 0; j<=3 ; j++)
            {
              if(wejscie[j] == Haslo[j])
              {
                ;
              }
              else
              {
                blad = 1;
                break;
              }
            }
            if(blad == 0)
            {
              for(j = 0 ; j<6 ; j++)
              {
              Dioda = 0;
              delay(opoznienie);
              Dioda = 1;
              delay(opoznienie);
              }
            }

            for(j = 0; j<=3; j++)
            {
              wejscie[j] = 0xFF;
            }
            j = 0;
```

```
            blad = 0;
          }
          else if(Key == 0x77)
          {
            for(j = 0; j<=3 ; j++)
            {
              if(wejscie[j] == Haslo[j])
              {
                ;
              }
              else
              {
                blad = 1;
                break;
              }
            }
            if(blad == 0)
            {
              for(j = 0; j<=2; j++)
              {
                Dioda = 0;
                delay(opoznienie);
                Dioda = 1;
                delay(opoznienie);
              }
              opoznienie = 1;
              delay(opoznienie);
              opoznienie = 0;
              for(j = 0; j<=2; j++)
              {
                Dioda = 0;
                delay(opoznienie);
                Dioda = 1;
                delay(opoznienie);
              }

              for(j = 0; j<=3; j++)
              {
                wejscie[j] = 0xFF;
              }
            }
            j = 0;
            blad = 0;
          }
          else
          {
            for(j = 0; j<=3; j++)
            {
              wejscie[j] = 0xFF;
            }
          j = 0;
          blad = 0;
          }
        }
      }
    }
    else
    {
      if (i<3)
      {
        i++;
      }
      else
      {
        i = 0;
      }
```

```
        Zezwalaj = 1;     // odblokowanie zezwalania na wejscie do procedur obslugi klawiszy
      }
    }
}

void delay(bit wolniej)
{
  unsigned char data i;
  unsigned char data j;

  if(wolniej == 0)
  {
    for(i=0 ; i<200 ; i++)
    {
       for(j=0;j<94;j++)
       {;}
    }
  }
  else
  {
    for(i=0 ; i<200 ; i++)
    {
       for(j=0;j<189;j++)
       {;}
    }
  }
}
```

Zrzuty ekranu z kompilacji i linkowania pokazujące brak błędów oraz rozmiar kodu i danych:

```
Build Output
compiling lab2_3.c...
lab2_3.c - 0 Error(s), 0 Warning(s).
```

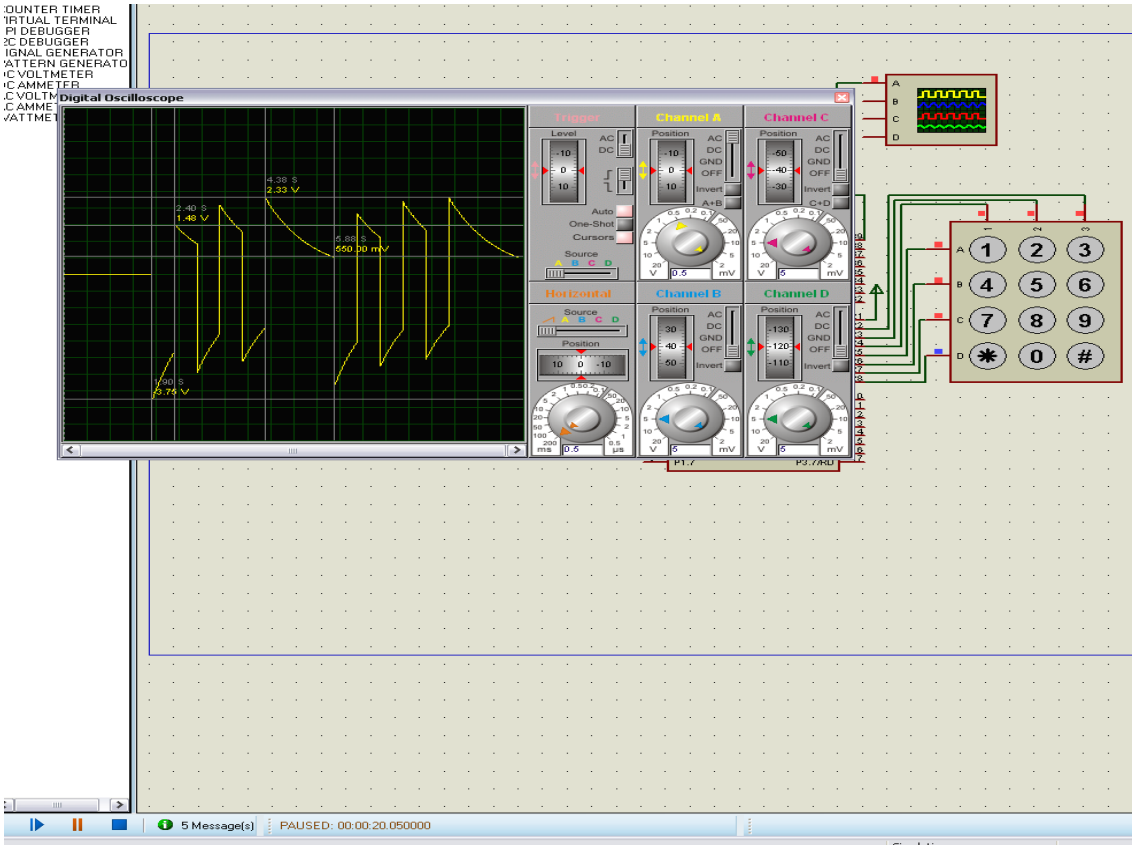
```
Build Output
Build target 'Target 1'
linking...
Program Size: data=20.4 xdata=0 const=4 code=472
creating hex file from "lab2_3"...
"lab2_3" - 0 Error(s), 0 Warning(s).
```

Zrzuty ekranu z symulatora Proteus:

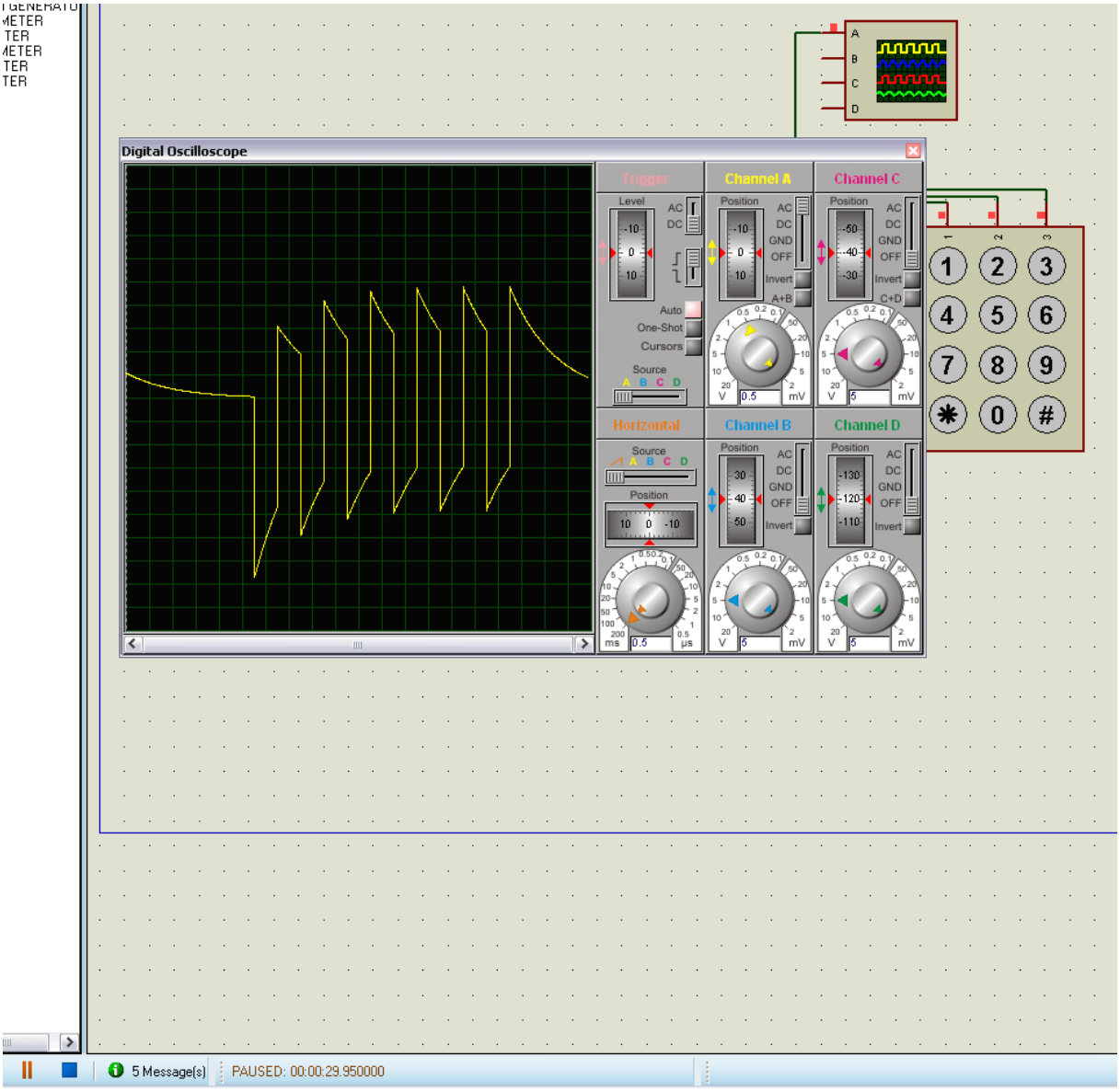
W celu sprawdzenia poprawności realizacji zadania w symulatorze Proteus, do obowiązującego schematu dodano oscyloskop, który podłączony został do pinu P0.0. Pozwala to na poprawne udokumentowanie zmian stanu diody. Przystąpiono do realizacji scenariusza:

a) wprowadzenie nieprawidłowego kodu (np. czterech zer), zatwierdzone poprawnym znakiem - brak sygnalizacji;

b) wprowadzenie poprawnego kodu, zatwierdzonego odwrotnym znakiem - sygnalizacja błędu;

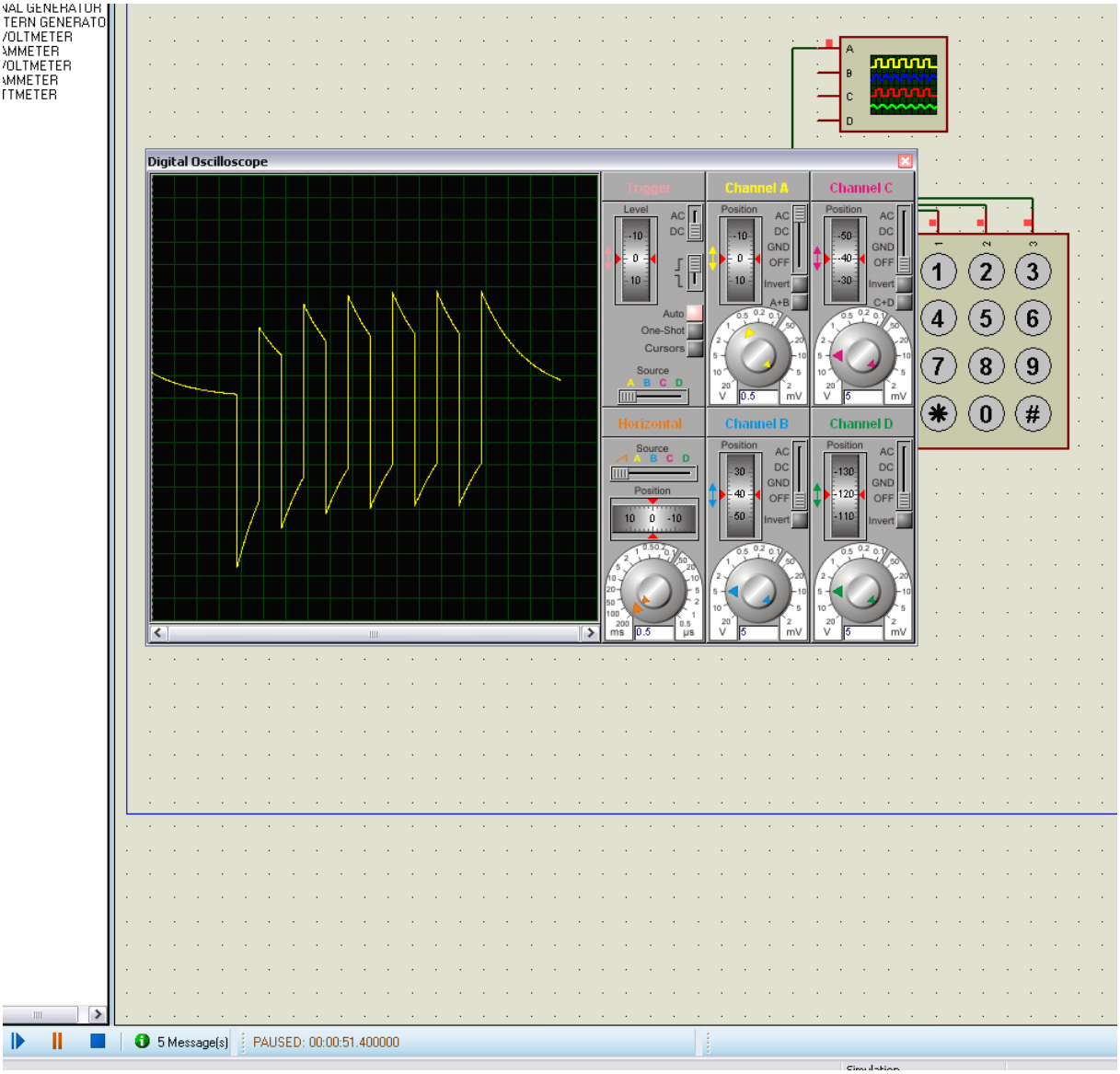


c) wprowadzenie poprawnego kodu, zatwierdzonego właściwym znakiem - sygnalizacja "otwarcia zamka";



d) powtórne wprowadzenie nieprawidłowego kodu (np. czterech dziewiątek), zatwierdzonych odwrotnym znakiem - brak sygnalizacji;

e) powtórne wprowadzenie poprawnego kodu, zatwierdzonego właściwym znakiem - sygnalizacja "otwarcia zamka";



Dzięki odpowiedniej obsłudze systemu wejścia – wyjścia oraz obsłudze wprowadzanych symboli przez mikrokontroler w trakcie całego scenariusza nie zaobserwowano niewłaściwego zachowania się układu.