Wojskowa Akademia Techniczna im. Jarosława Dąbrowskiego

Laboratorium z przedmiotu:

Systemy wbudowane

Sprawozdanie z ćwiczenia laboratoryjnego nr 3:

Obsługa systemu wejścia - wyjścia

Prowadzący:
mgr inż. Artur Miktus

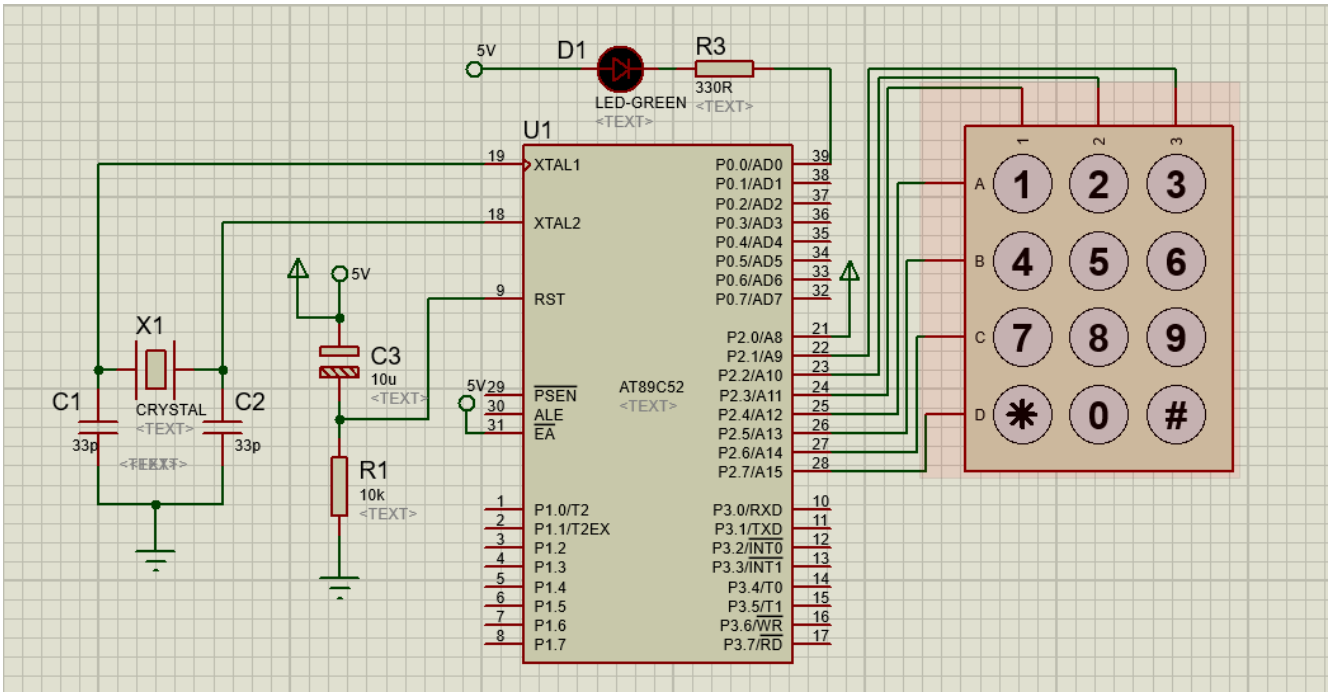
Wykonał: Radosław Relidzyński
Nr albumu: 76836
Grupa: WCY20IY4S1
Data laboratoriów: 20.05.2022 r.
Deklarowana ocena: 3, 4, 5

Spis treści

A. Treść zadania

Zadanie na laboratorium nr 3.

Dany jest schemat układu jak na rysunku:



Zadania laboratoryjne.

Jako zasadę na dzisiejszych zajęciach proszę przyjąć, że:

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.

---

1. Na ocenę **dostatecznie** napisać dla schematu **Zad_5.pdsprj** program **lab3_1.c** w języku C, który w pętli wewnętrznej będzie kolejno obsługiwał klawiaturę w ten sposób, że po naciśnięciu dowolnego przycisku dioda LED rozbłysnie jeden raz "przez 1 s" - czas przybliżony, uzyskany dzieki **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.
Uwaga - pomiar czasu trwania pętli i włączanie diody wykonywać w maszynie wirtualnej, ja też będę oceniał Państwa prace w maszynie wirtualnej (Keil uvision 4.60). Dotyczy również zadań na db i na bdb.

---

2. Na ocenę **dobrze** zrobić to, co na **dostatecznie, oraz ponadto** napisać dla schematu **Zad_5.pdsprj** program **lab3_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.

   Uwaga - patrz dopisek na dole strony o niepoprawnej obsłudze klawiatury.

---

3. Na ocenę **bardzo dobrze** zrobić to, co **na dobrze, oraz ponadto** napisać dla **schematu Zad_5.pdsprj** program **lab3_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.

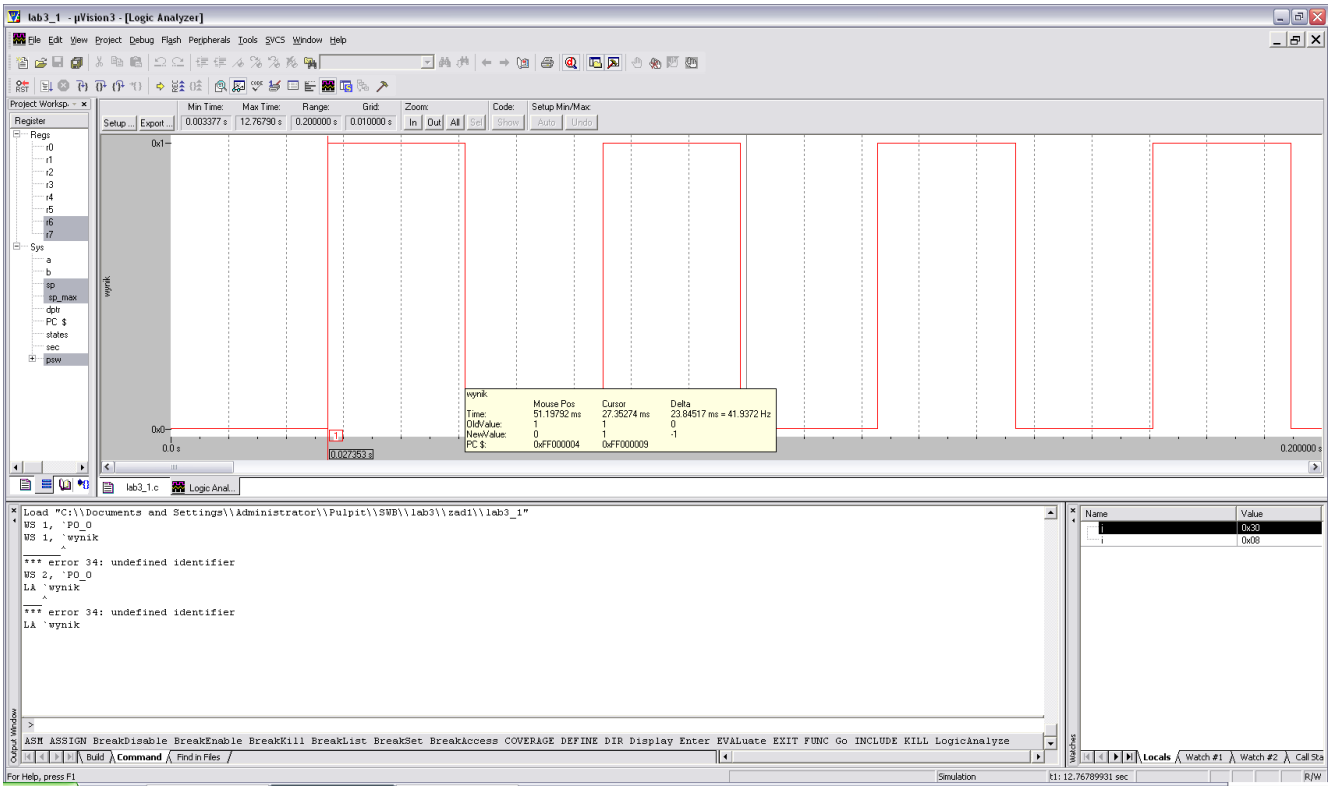
   Program musi **bez resetowania** poprawnie obsłużyć nastepujący scenariusz zdarzeń:
   a) wprowadzenie nieprawidłowego kodu (np. czterech zer), zatwierdzone poprawnym znakiem - brak sygnalizacji;
   b) wprowadzenie poprawnego kodu, zatwierdzonego odwrotnym znakiem - sygnalizacja błędu;
   c) wprowadzenie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

B. Opracowywanie funkcji Delay()

W ramach ćwiczenia potrzebne będzie utworzenie funkcji Delay(), która przez odpowiednie zagnieżdżenie pętli będzie wykonywać się przez około 1 sekundę. W tym celu stworzę minimalny program, który pozwoli mi badać i dostosowywać tą funkcję do zamierzonego efektu. Na początku w pętlach ustawię wartości 9 i 100. Funkcja wygląda następująco:

```
#include <REGX52.H>
void Delay(void);
void main (void)
{
    unsigned char wynik;
    while(1)
    {
        wynik=0;
        Delay();
        wynik=1;
        Delay();
    }
}
void Delay(void)
{
    unsigned char i, j;
    for(i=0;i<9;i++)
        for(j=0;j<100;j++) {;}
}
```

Czas działania tej funkcji jest pokazany poniżej:



Czas to niecałe 24 ms.

Aktualnie funkcja wykonuje 900 operacji (9 * 100), jak widać na powyższym zrzucie ekran tyle operacji zajmuje programowi ponad sekundę. W związku z tym przy pomocy prostego wzoru przeliczę, ile mniej więcej operacji będzie potrzebnych, aby zbliżyć się bardziej do równej sekundy.

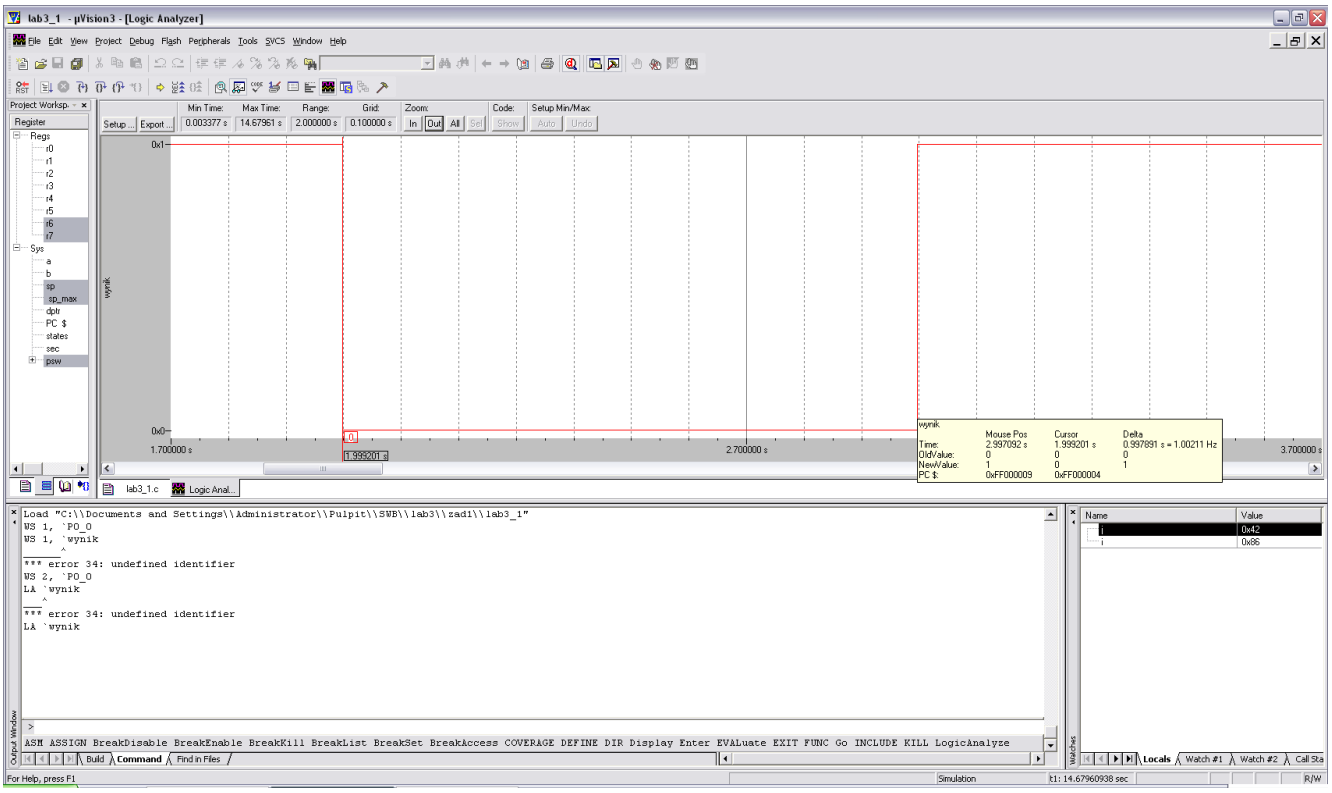
$$ilość\ operacji\ na\ sekundę = \frac{aktualna\ ilość\ operacji}{aktualny\ czas} = {}^{900}/_{0{,}02385517} \approx 37\,743$$

Rozpiszę ten wynik na mniejsze liczby: $37\,743 \approx 190 * 200$

Gotowy kod:

```
#include <REGX52.H>
void Delay(void);
void main (void)
{
    unsigned char wynik;
    while(1)
    {
        wynik=0;
        Delay();
        wynik=1;
        Delay();
    }
}
void Delay(void)
{
    unsigned char i, j, k;
    for(i=0;i<190;i++)
        for(j=0;j<200;j++) {;}
}
```

Dzięki temu udało się uzyskać czas 0,997891 sekundy.



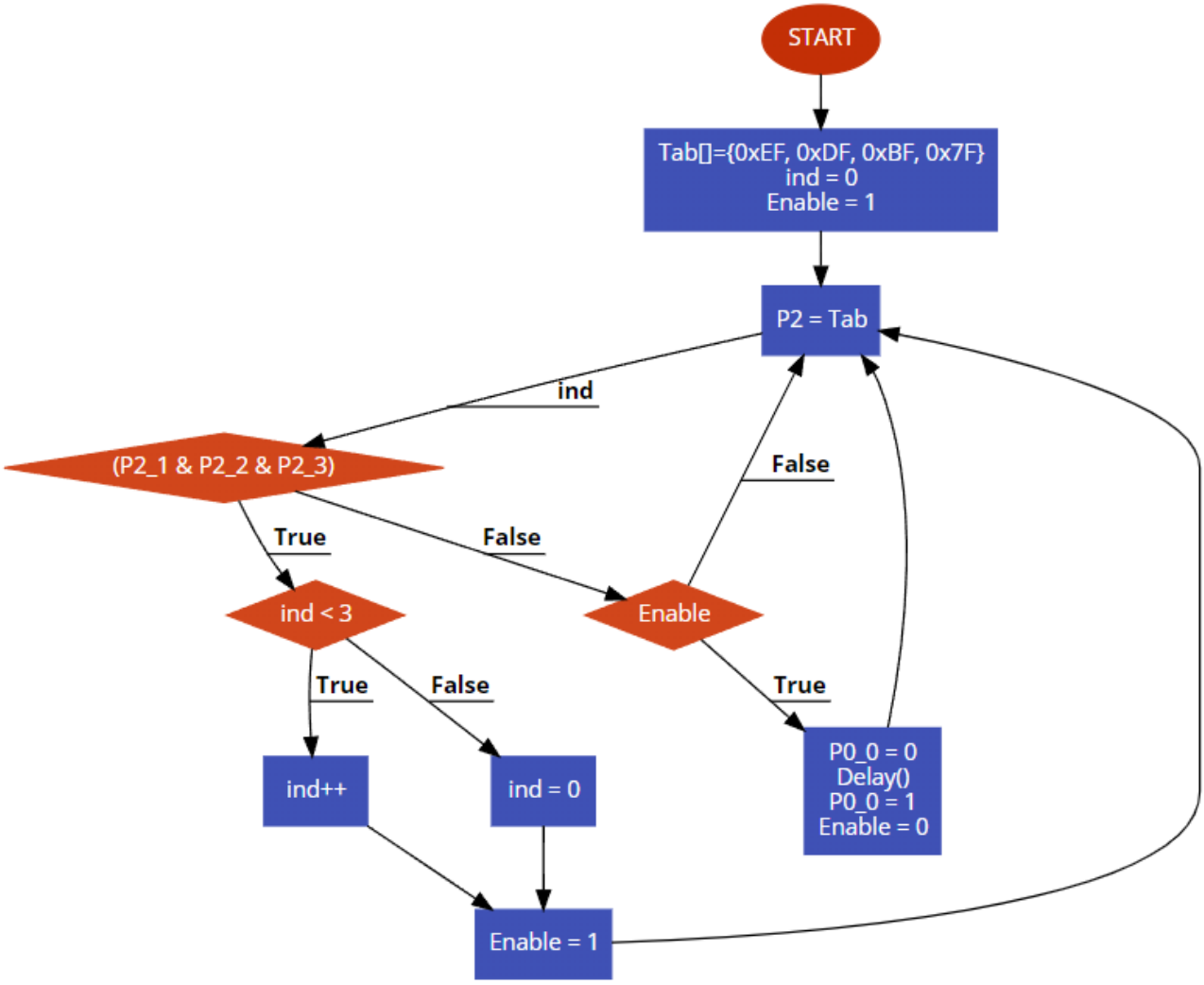
Stworzoną funkcję Delay() będę wykorzystywał w rozwiązaniach zadań laboratoryjnych.

C. Zadanie na ocenę dostateczną

Opis mojego rozwiązania

Do mojego rozwiązania wykorzystuję tablicę „tab” zawierającą wartości dla wędrującego zera (po wierszach). Kiedy żaden przycisk nie jest wciśnięty, program cyklicznie iteruje po tablicy i zmienia położenie zera. Gdy zostanie wciśnięty jakikolwiek przycisk, do diody zostanie wysłany sygnał 0, przez co się zapali. Po wykonaniu się funkcji Delay() działającej przez sekundę do diody wysłany będzie sygnał 1 gaszący ją. Po tym zacznie z powrotem działać pętla dla wędrującego zera. Sygnał zarządzany jest zmienną „Enable”

Schemat blokowy rozwiązania



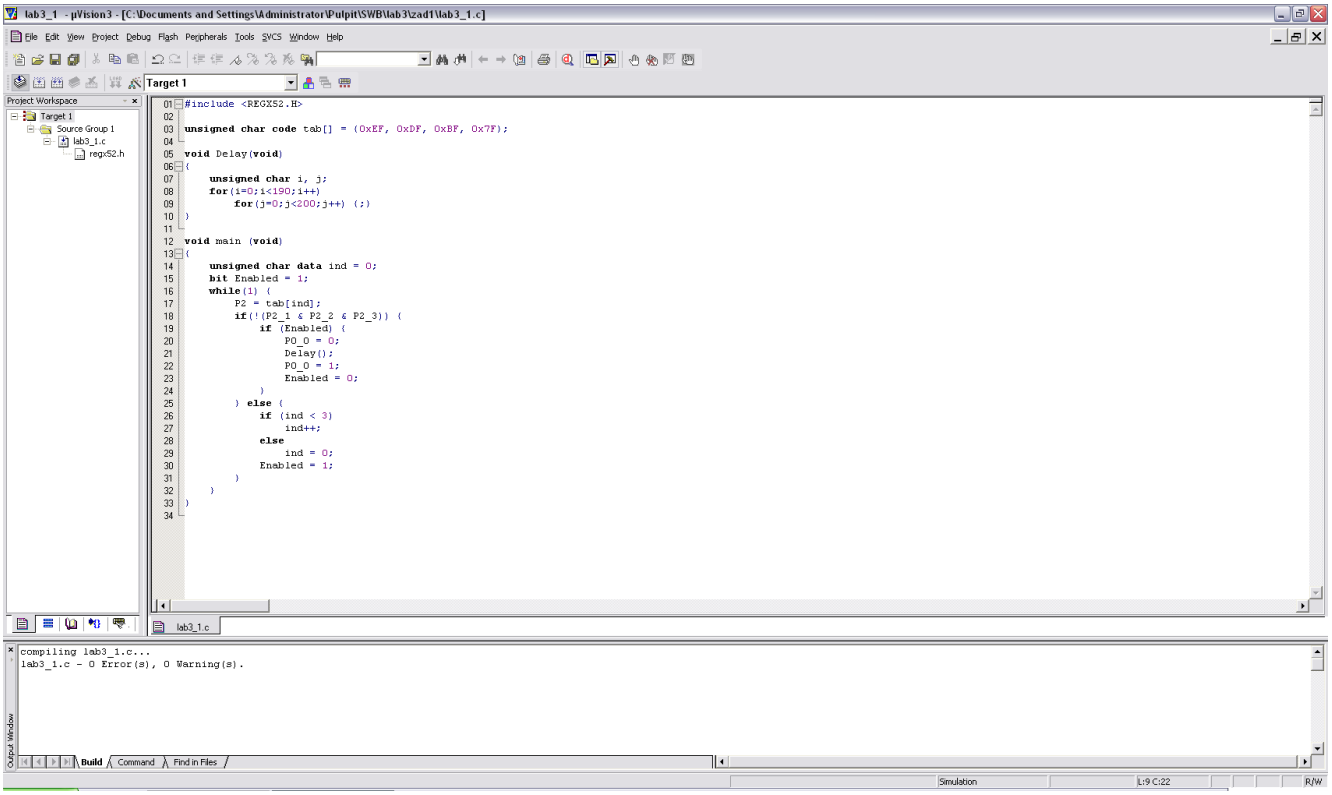
Listing programu

```
#include <REGX52.H>

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

void Delay(void)
{
    unsigned char i, j;
    for(i=0;i<190;i++)
        for(j=0;j<200;j++) {;}
}

void main (void)
{
    unsigned char data ind = 0;
```

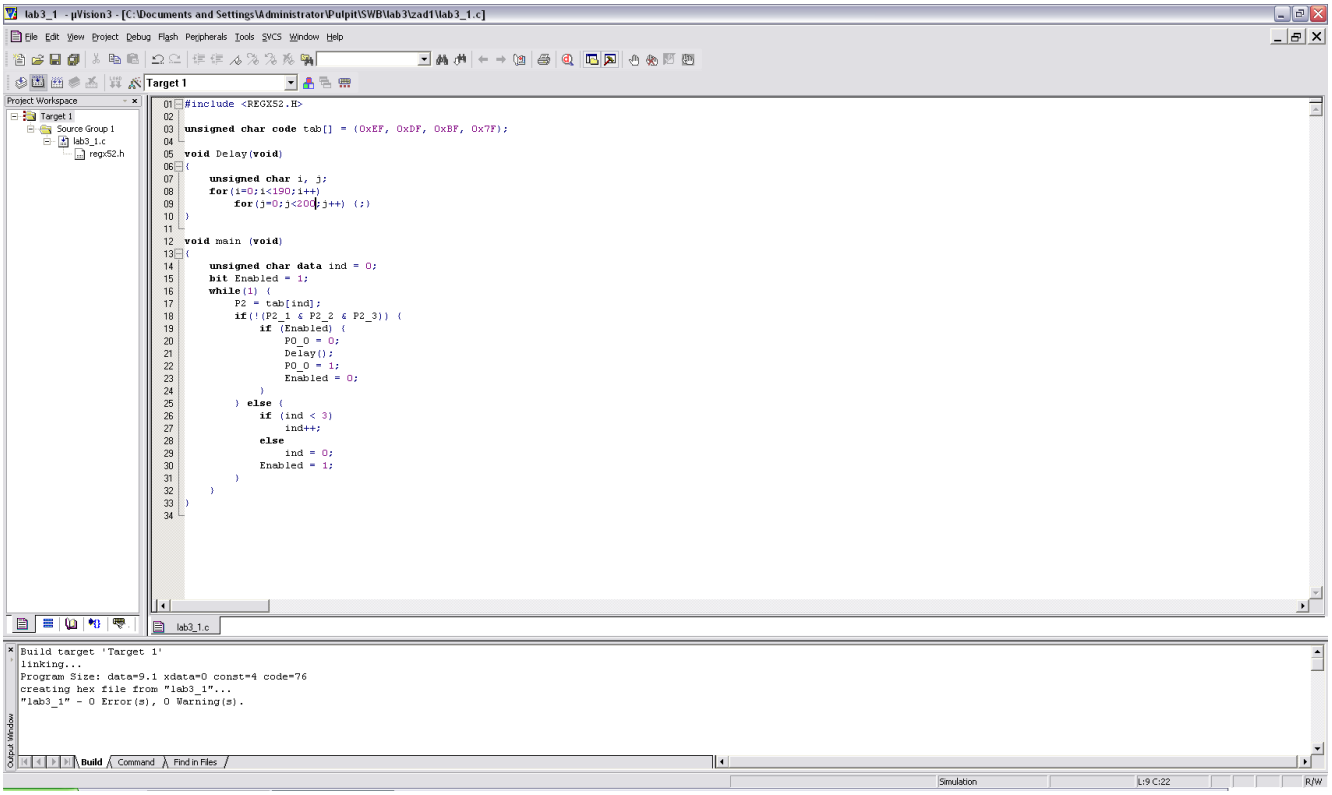
```
    bit Enabled = 1;
    while(1) {
        P2 = tab[ind];
        if(!(P2_1 & P2_2 & P2_3))
        {
            if (Enabled)
            {
                P0_0 = 0;
                Delay();
                P0_0 = 1;
                Enabled = 0;
            }
        } else {
            if (ind < 3)
                ind++;
            else
                ind = 0;
            Enabled = 1;
        }
    }
}
```

Sprawdzenie poprawności

Kompilowanie

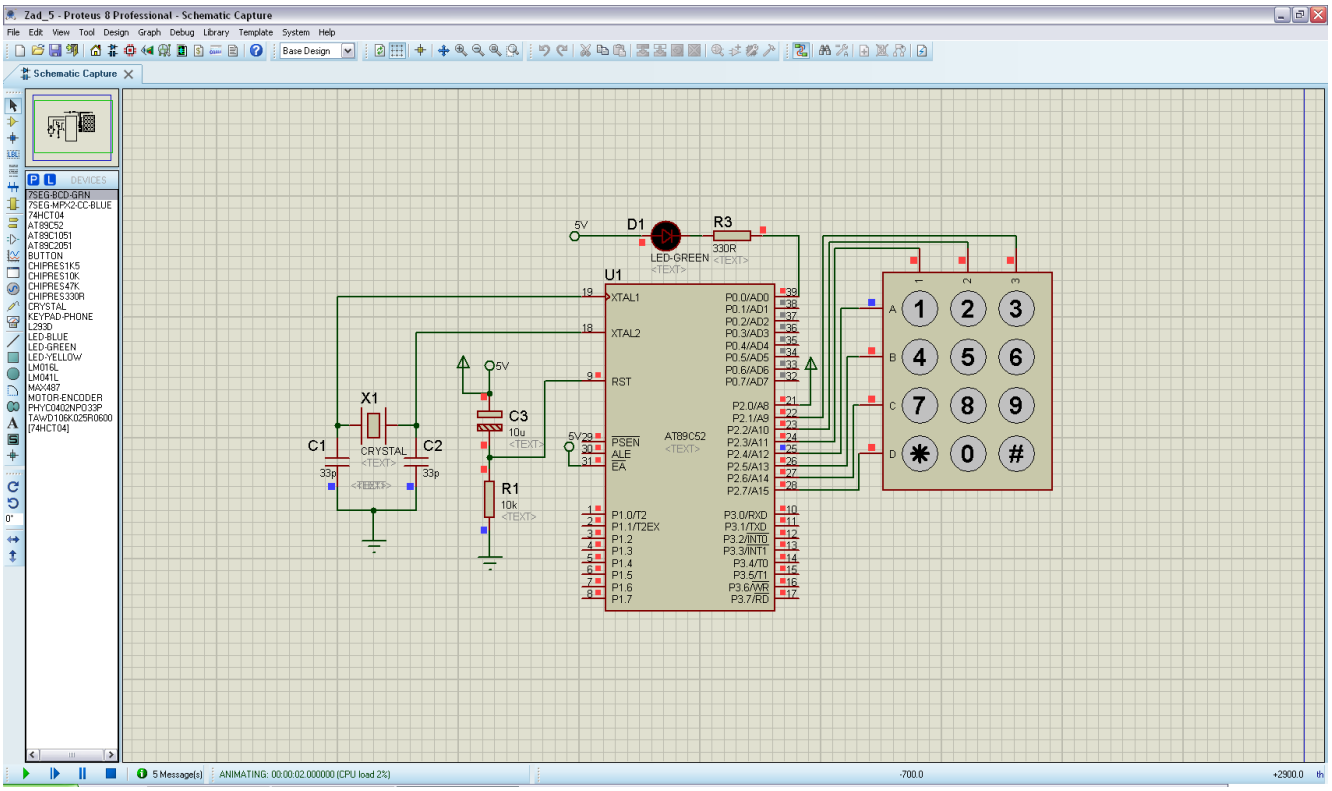


Linkowanie

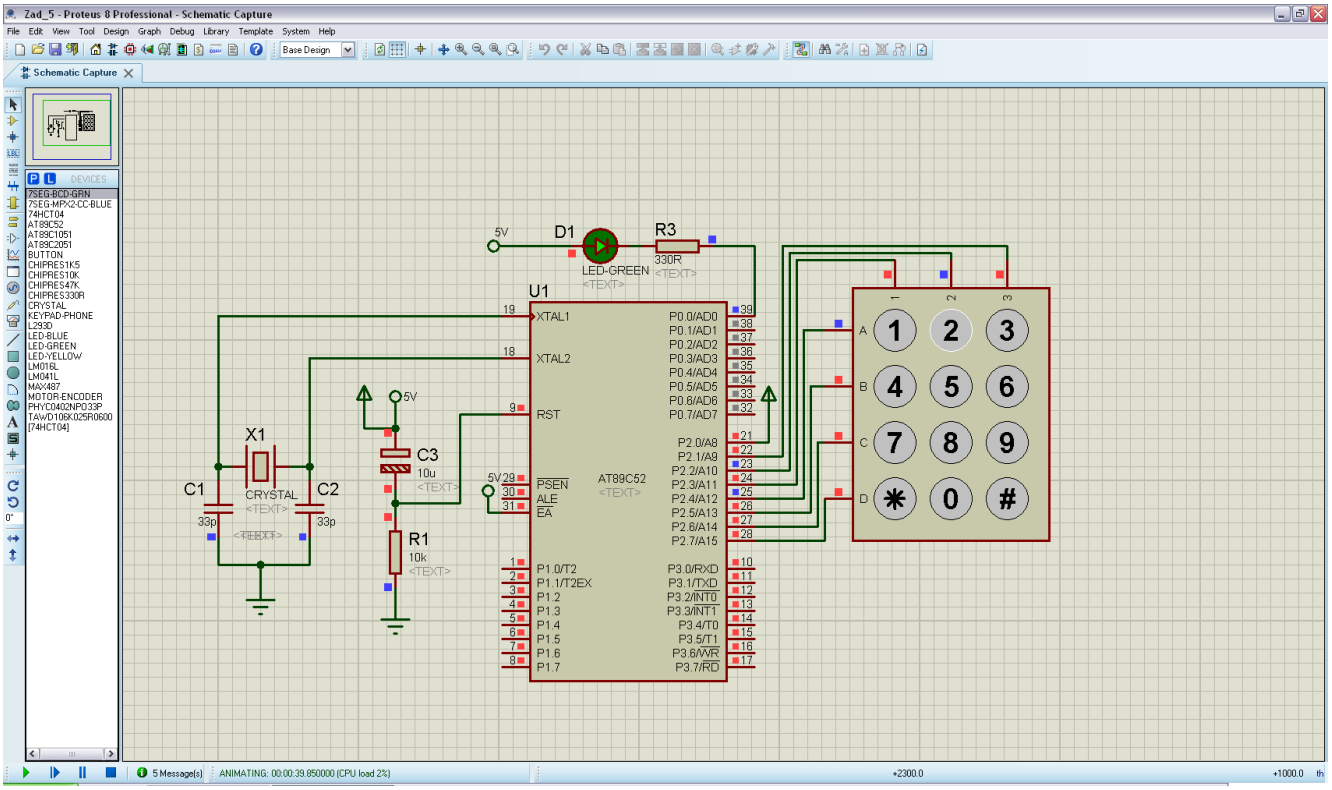


Prezentacja realizacji zadania przez program

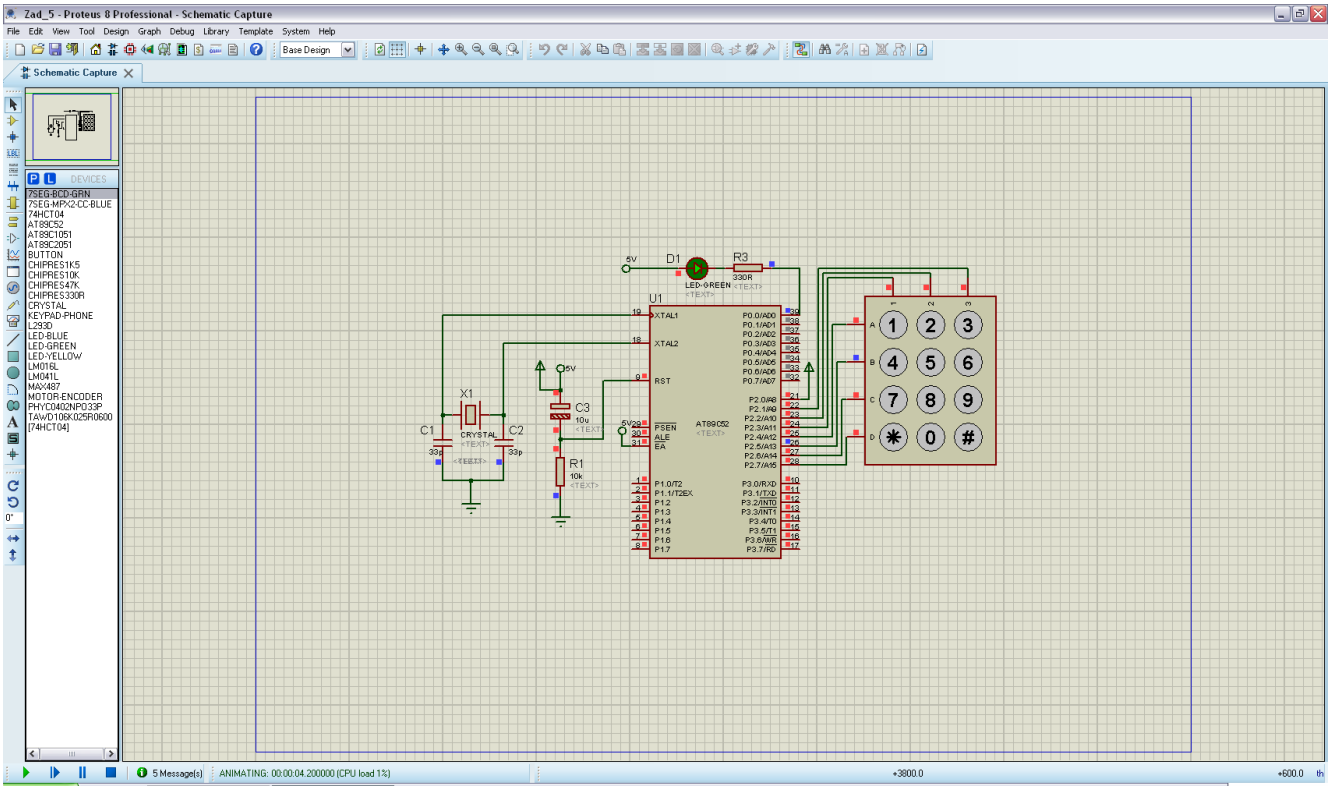
Stan początkowy, wędrujące zero



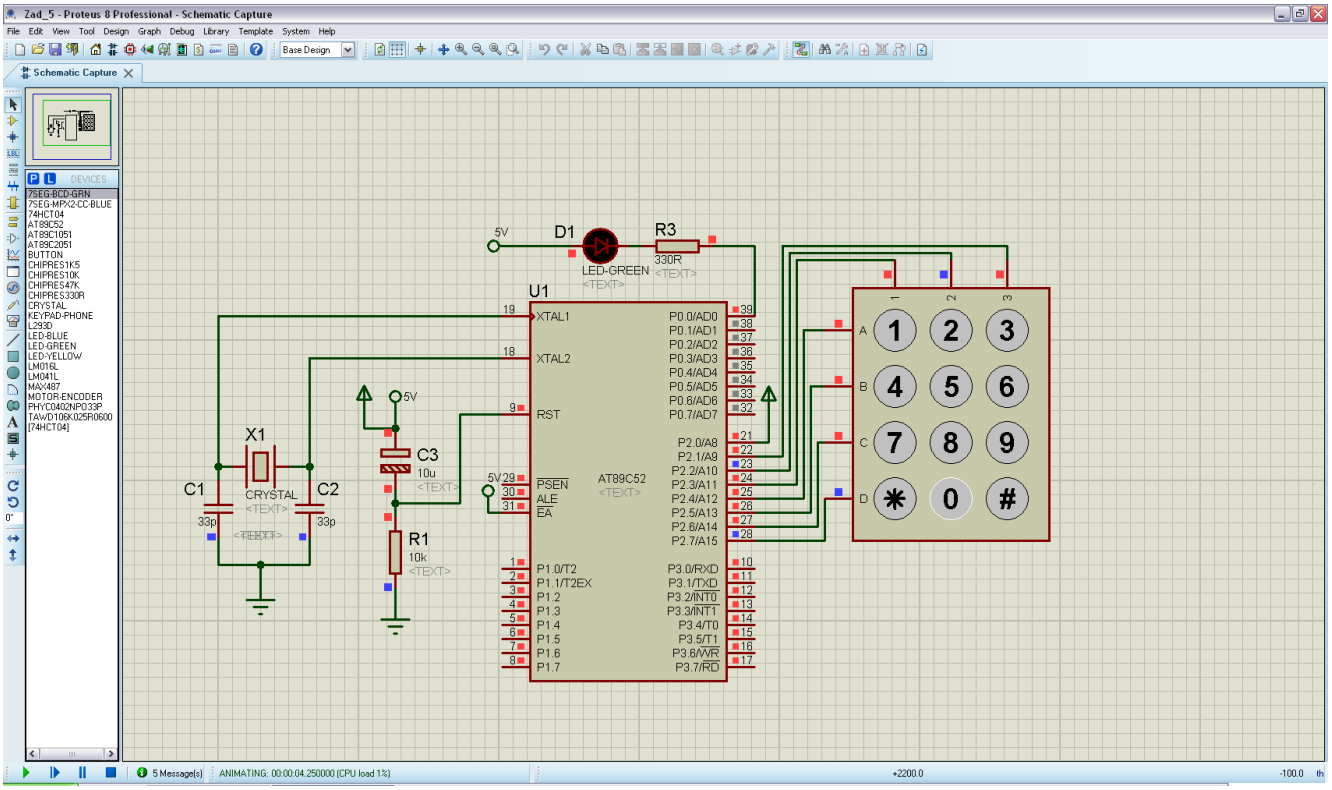
Wciśnięcie przycisku 2 – wynik w momencie wciśnięcia



Wciśnięty przycisku 6 – puszczenie przycisku



Przytrzymanie przycisku 0 po upływie czasu.

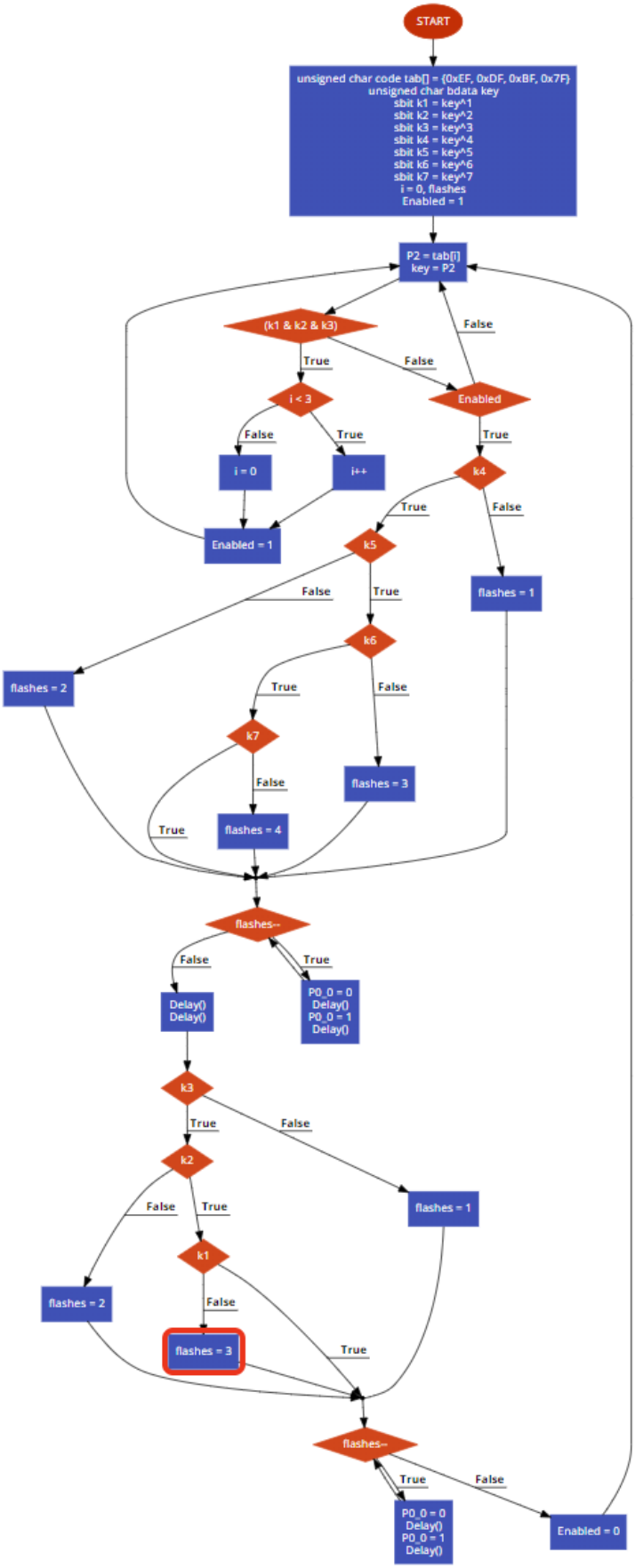


A. Zadanie na ocenę dobrą

Opis mojego rozwiązania

Bieżący problem rozwiązałem poprzez rozbudowanie programu z poprzedniego zadania. Nadal korzystam z iteracji po tablicy generującej wędrujące zero. W ramach wspomnianego rozbudowania dodałem zmienną „key” i zmienne wskazujące na jej bity. Następnie w momencie wciśnięcia przycisku najpierw pobieram numer wiersza do zmiennej „flashes”, potem tyle razy wykonuję błysk diody, następnie dodaję opóźnienia, żeby była przerwa 3 sekundy i wykonuję to samo dla numeru kolumny.

Schemat blokowy rozwiązania



Listing programu

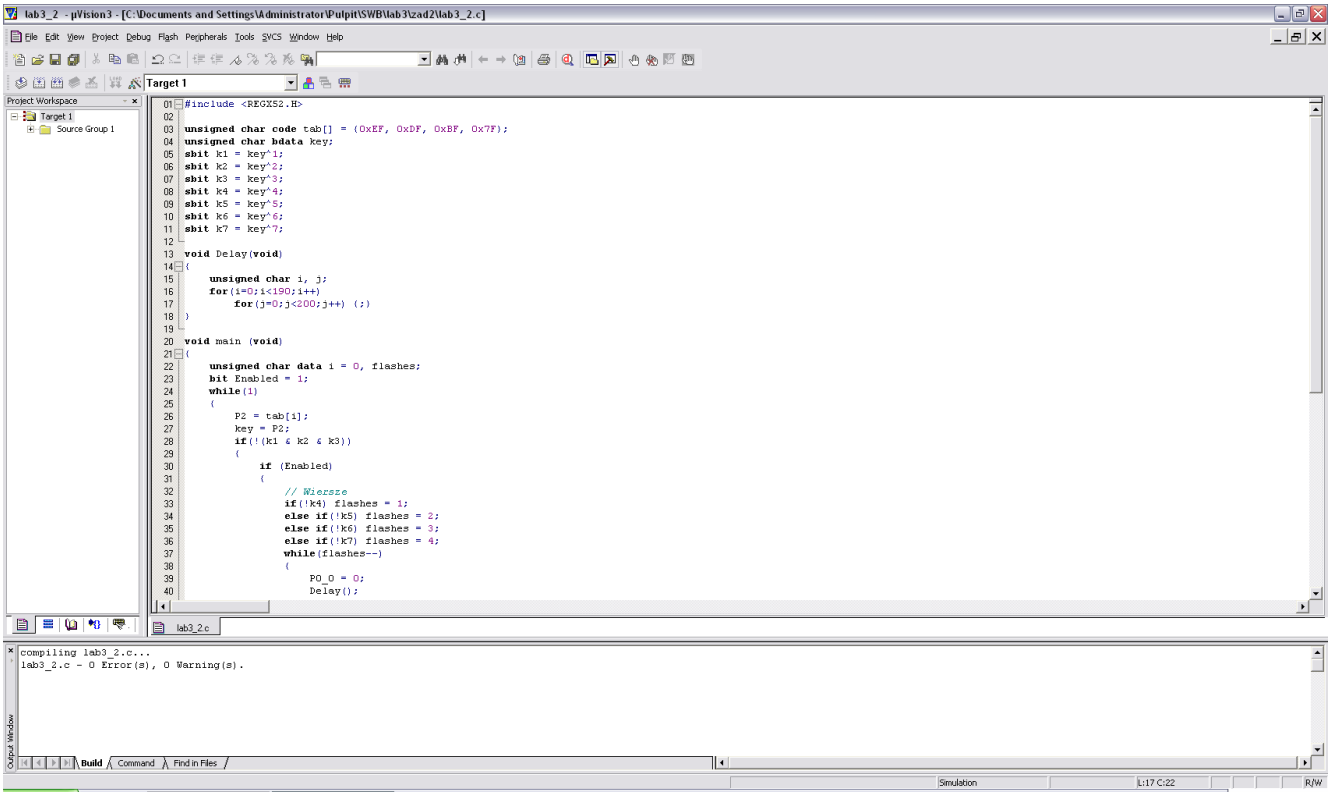
```
#include <REGX52.H>

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

```
unsigned char bdata key;
sbit k1 = key^1;
sbit k2 = key^2;
sbit k3 = key^3;
sbit k4 = key^4;
sbit k5 = key^5;
sbit k6 = key^6;
sbit k7 = key^7;

void Delay(void)
{
    unsigned char i, j;
    for(i=0;i<190;i++)
        for(j=0;j<200;j++) {;}
}

void main (void)
{
    unsigned char data i = 0, flashes;
    bit Enabled = 1;
    while(1)
    {
        P2 = tab[i];
        key = P2;
        if(!(k1 & k2 & k3))
        {
            if (Enabled)
            {
                // Wiersze
                if(!k4) flashes = 1;
                else if(!k5) flashes = 2;
                else if(!k6) flashes = 3;
                else if(!k7) flashes = 4;
                while(flashes--)
                {
                    P0_0 = 0;
                    Delay();
                    P0_0 = 1;
                    Delay();
                }
                Delay(); Delay();
                // Kolumny
                if(!k3) flashes = 1;
                else if(!k2)  flashes = 2;
                else if(!k1) flashes = 3;
                while(flashes--)
                {
                    P0_0 = 0;
                    Delay();
                    P0_0 = 1;
                    Delay();
                }
                Enabled = 0;
```

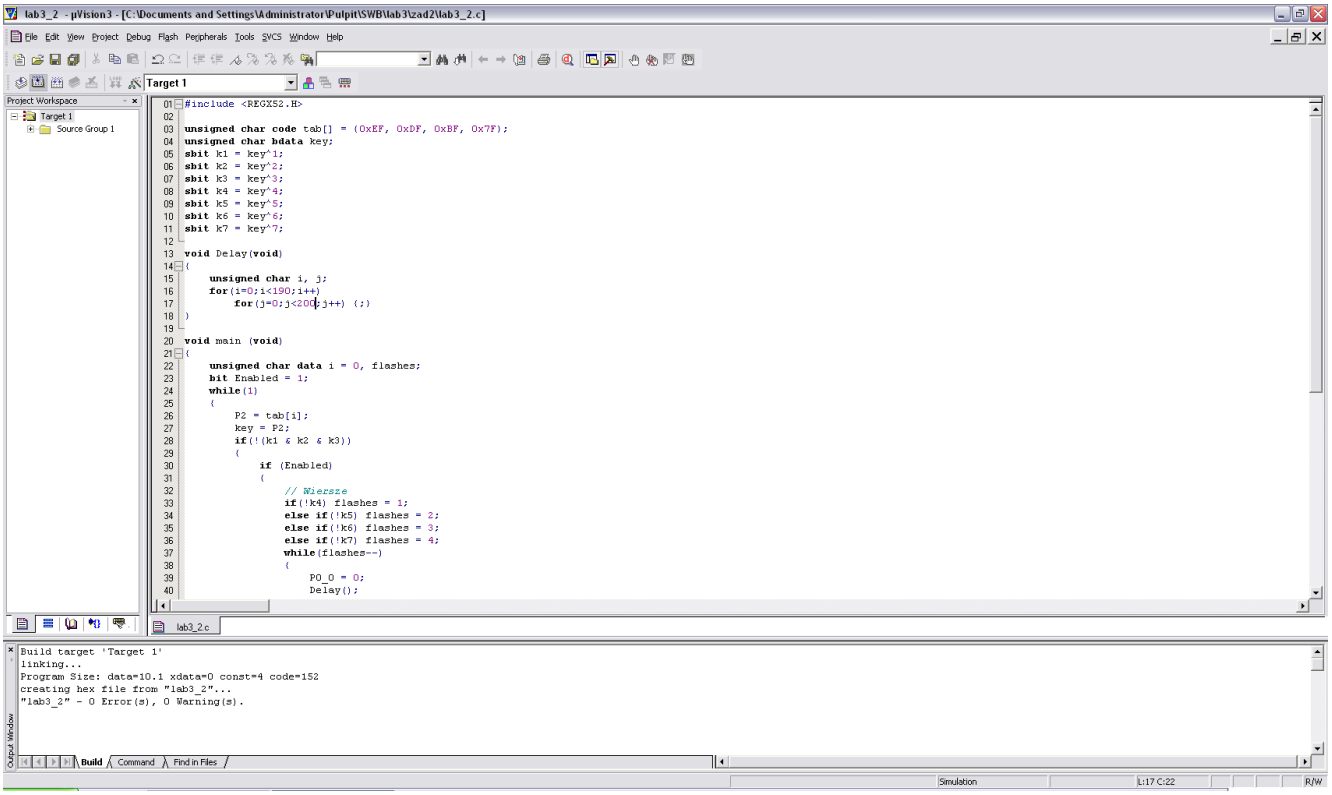
```
            }
        } else
        {
            if (i < 3)
                i++;
            else
                i = 0;
            Enabled = 1;
        }
    }
}
```

Sprawdzenie poprawności

Kompilowanie

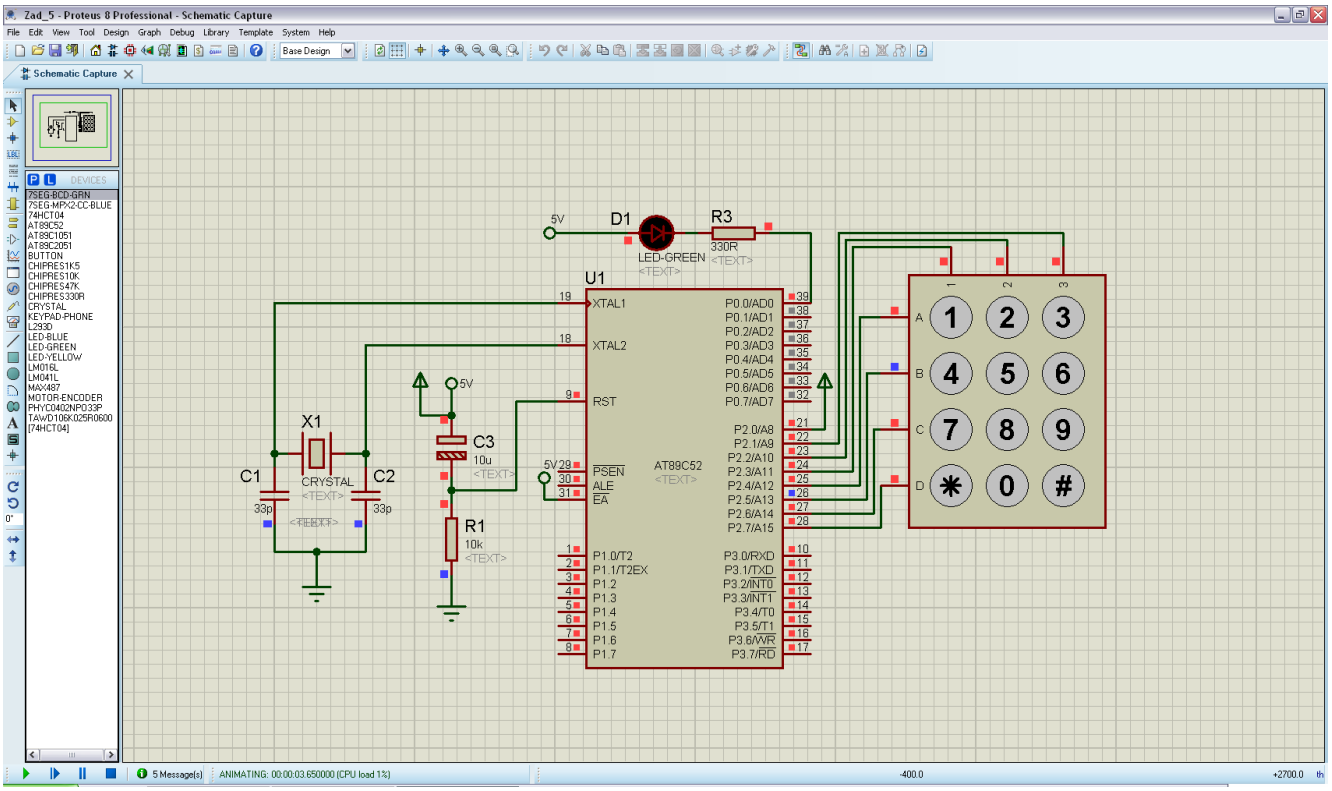


Linkowanie

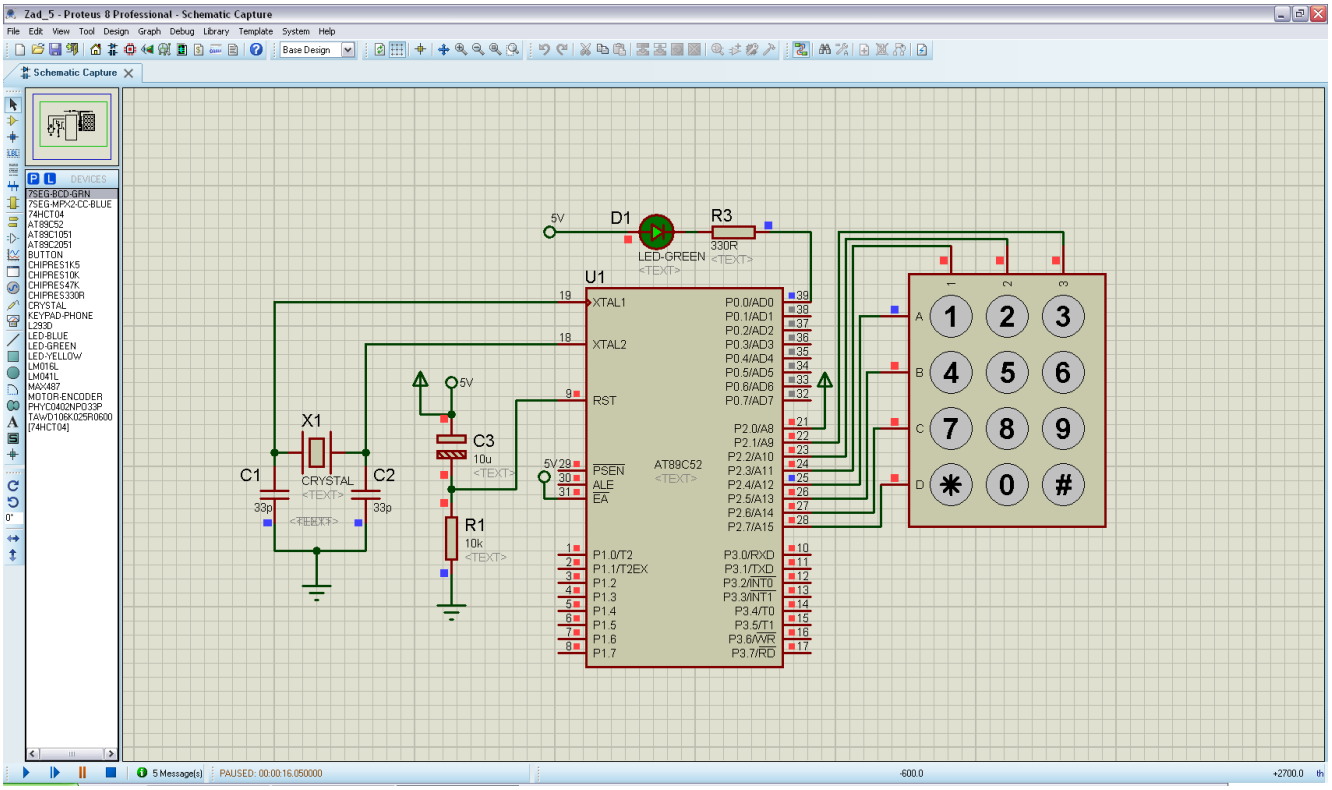


Prezentacja realizacji zadania przez program

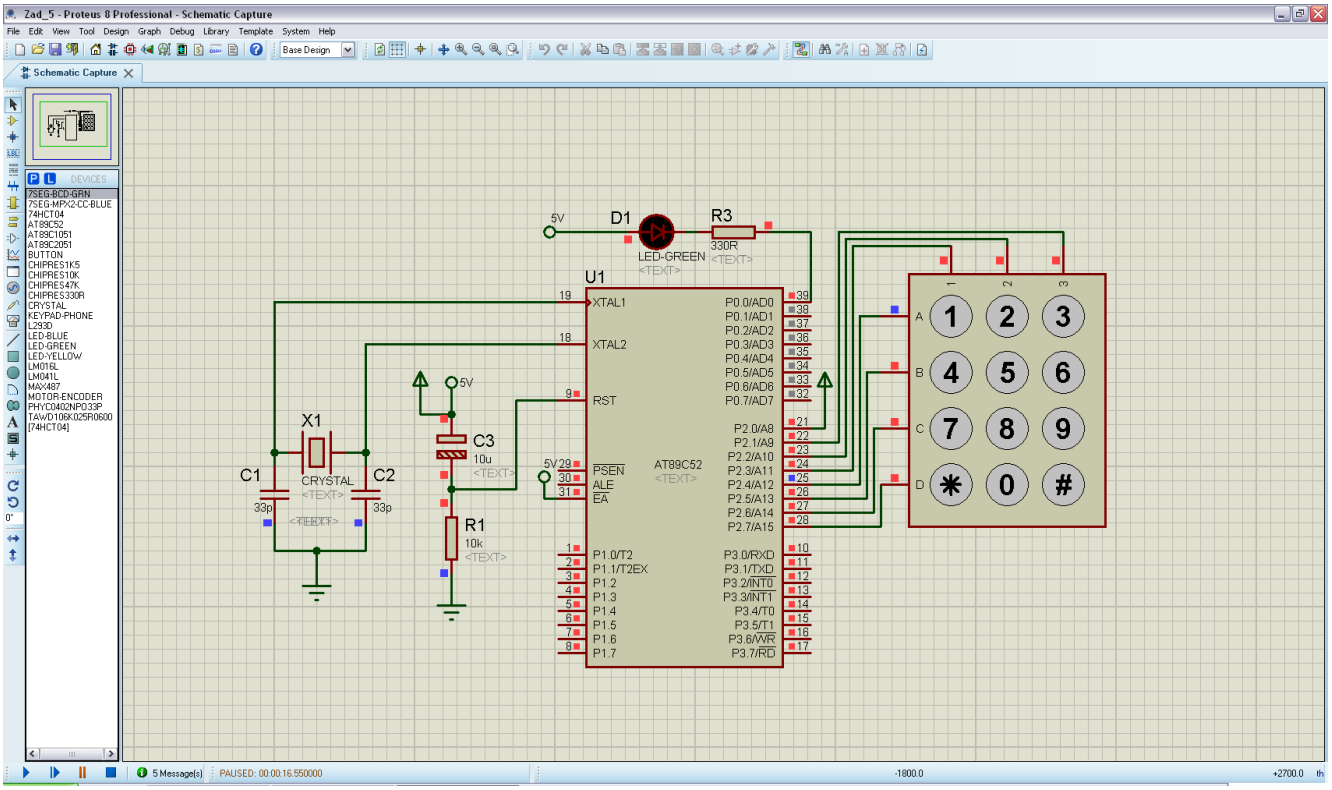
Stan początkowy, wędrujące zero



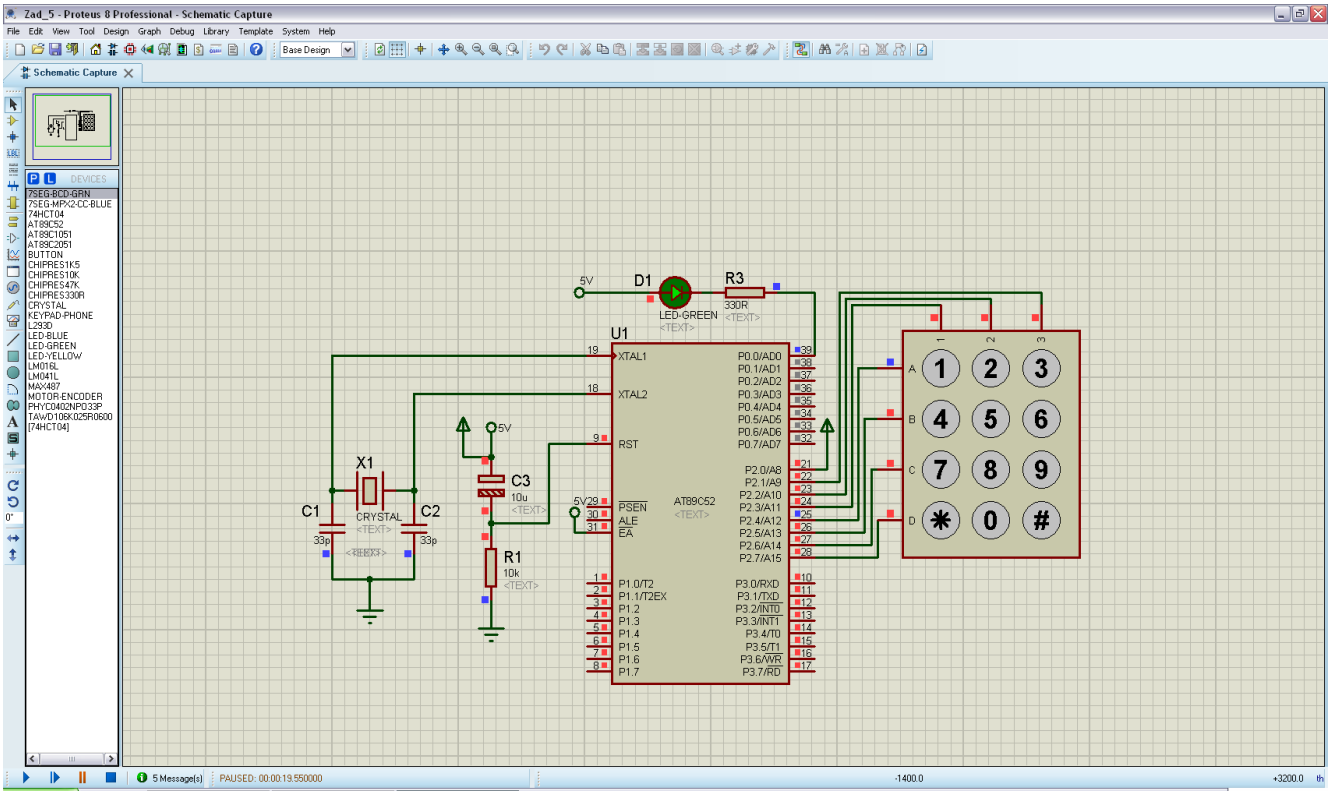
Wciśnięcie przycisku 1, dioda się zapala (1 błysk, bo pierwszy wiersz)



Po zgaśnięciu dioda się nie świeci przez 3 sekundy



Potem znowu się zapala (1 błysk, bo pierwsza kolumna)



A. Zadanie na ocenę bardzo dobrą

Opis mojego rozwiązania

W ramach tego zadania wykorzystałem mechanikę odczytywania znaków z poprzednich zadań. W ramach działania programu stworzyłem funkcję „checkCode()” wykorzystującą następujące zmienne:

1. „success” – bit z informacją, czy wprowadzony kod jest prawidłowy
2. correctCode[] – tablica wartości kolejnych poprawnych cyfr kodu.
3. „ccind” – indeks, zmienna do wskazywania odpowiedniego elementu tablicy „correctCode[]”.

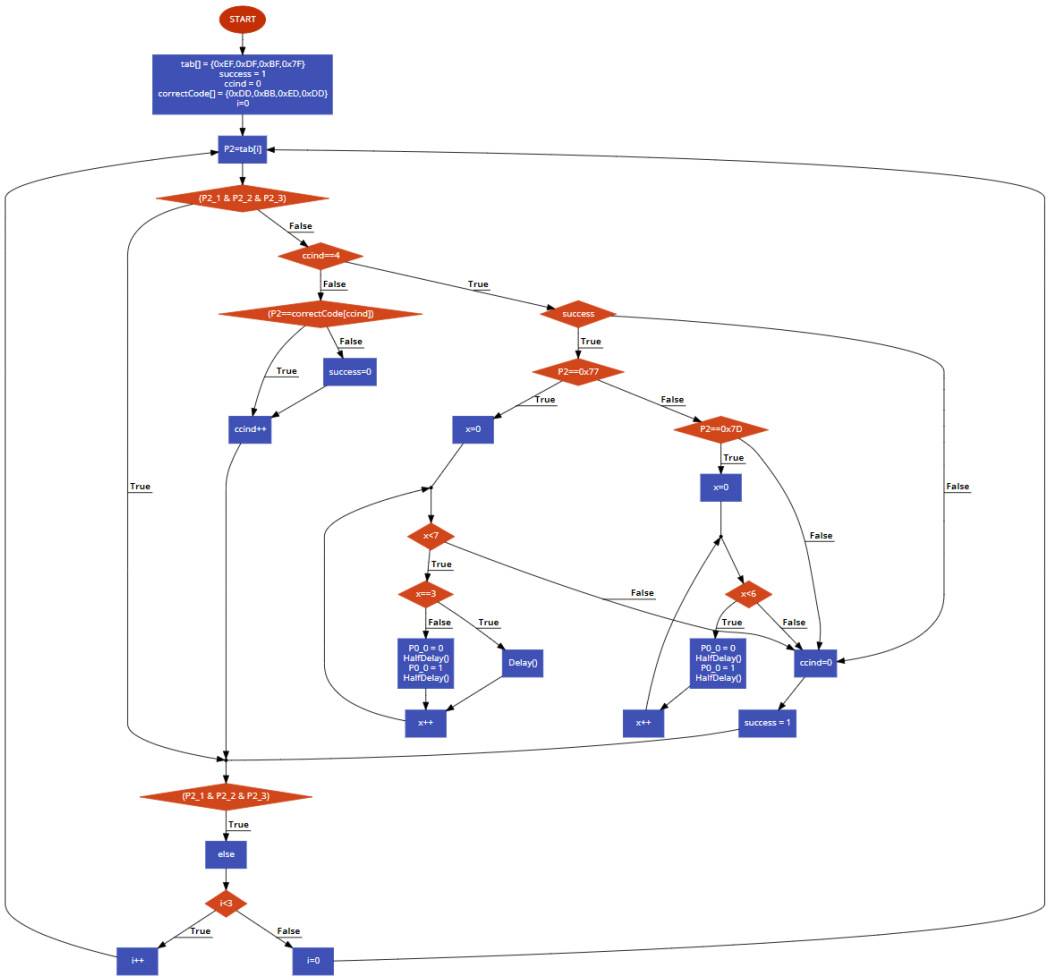
Początkowo wartość bitu success ustawiona jest na 1. Kiedy wprowadzane są z klawiatury kolejne cyfry odpowiedni warunek sprawdza, czy wprowadzona cyfra jest równa cyfrze w odpowiednim miejscu w tablicy. Jeśli się pokrywa, warunek nic nie robi. Natomiast jeśli wartości te będą różne, wtedy bit success zmienia wartość na 0.

W momencie, gdy naciśnięte zostaną 4 klawisze, wciśnięcie piątego przenosi nas do warunku sprawdzającego bit success, czyli czy wprowadzony kod jest prawidłowy. Jeśli jest nieprawidłowy, to nie robi nic. Jeśli jest prawidłowy to sprawdza wprowadzony piąty klawisz. Wtedy dzieje się jeden z trzech poniższych scenariuszy:

1. Jeśli nie jest ani „*”, ani „#”, nie robi nic.
2. Jeśli jest „*”, to wywołuje sygnał błędu (2 serie po 3 błyśnięcia 0,5s, z przerwą 1s).
3. Jeśli jest „#”, to wywołuje sygnał otwarcia zamka (6 błyśnięć po 0,5s).

Po tym program „sprząta” po wykonanej sekwencji, serując indeks ccind oraz ustawiając bit success na 1. Dzięki temu program może nieprzerwanie przyjmować kolejne sekwencje.

Schemat blokowy rozwiązania



Listing programu

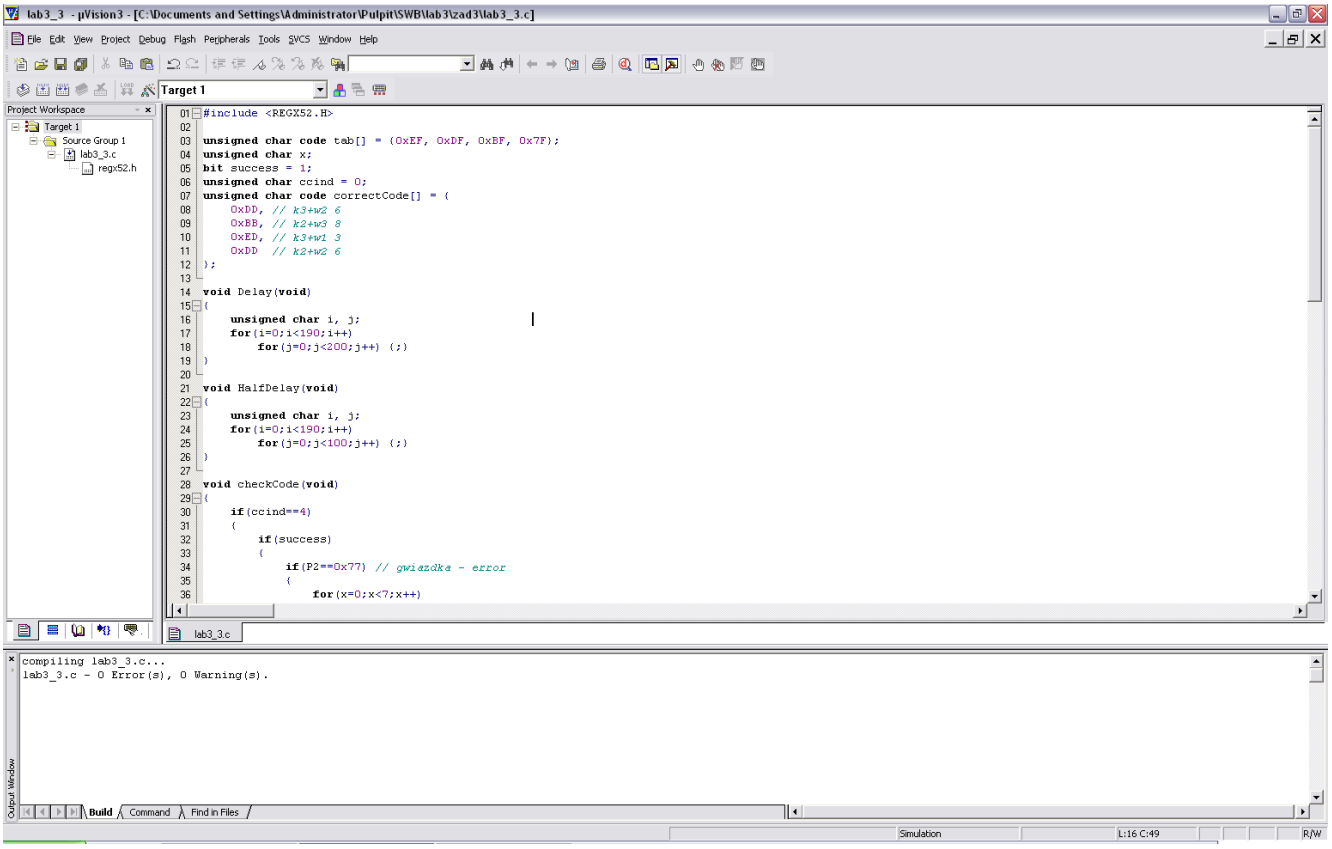
```
#include <REGX52.H>

unsigned char code tab[] = {0xEF, 0xDF, 0xBF, 0x7F};
unsigned char x;
bit success = 1;
unsigned char ccind = 0;
unsigned char code correctCode[] = {
      0xDD, // k3+w2 6
      0xBB, // k2+w3 8
      0xED, // k3+w1 3
      0xDD  // k2+w2 6
};

void Delay(void)
{
      unsigned char i, j;
      for(i=0;i<190;i++)
            for(j=0;j<200;j++) {;}
}

void HalfDelay(void)
{
      unsigned char i, j;
      for(i=0;i<190;i++)
```

```
            for(j=0;j<100;j++) {;}
}

void checkCode(void)
{
      if(ccind==4)
      {
            if(success)
            {
                  if(P2==0x77) // gwiazdka - error
                  {
                        for(x=0;x<7;x++)
                        {
                              if(x==3) //Pauza
                              {
                                    Delay();
                              }
                              else //Blysk
                              {
                                    P0_0 = 0;
                                    HalfDelay();
                                    P0_0 = 1;
                                    HalfDelay();
                              }
                        }
                  }
                  else if(P2==0x7D) // krzyzyk - otwarcie
                  {
                        for(x=0;x<6;x++)
                        {
                              P0_0 = 0;
                              HalfDelay();
                              P0_0 = 1;
                              HalfDelay();
                        }
                  }
            }
            ccind=0;
            success = 1;
      }
      else
      {
            if(!(P2==correctCode[ccind]))
            {
                  success=0;
            }
            ccind++;
      }
}

void main (void)
{
      unsigned char data i=0;
```

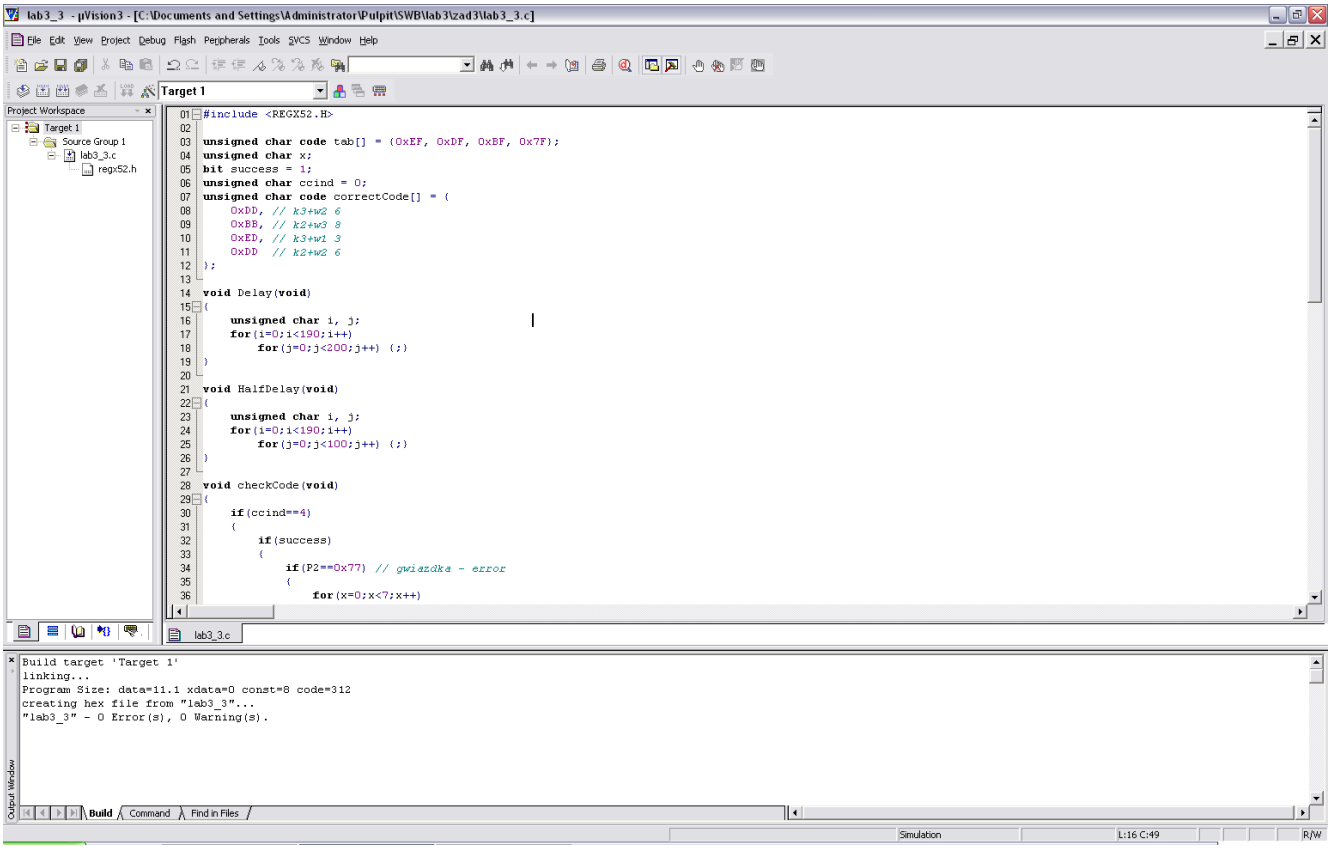
```
	while(1)
	{
		P2=tab[i];
		if(!(P2_1 & P2_2 & P2_3))
		{
			checkCode();
			while(!(P2_1 & P2_2 & P2_3));
		}
		else
		{
			if(i<3)
				i++;
			else
				i=0;
		}
	}
}
```

Sprawdzenie poprawności

Kompilowanie

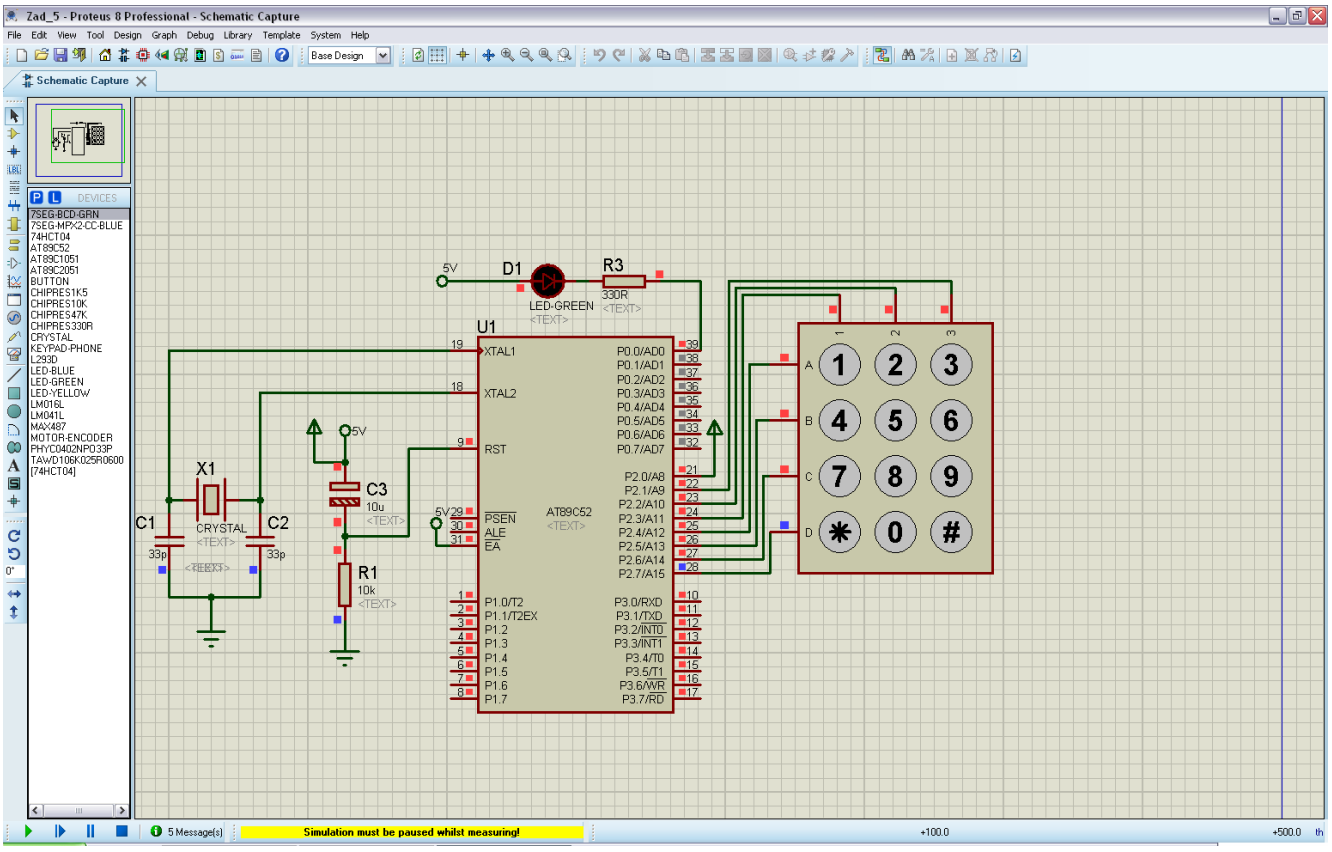


Linkowanie

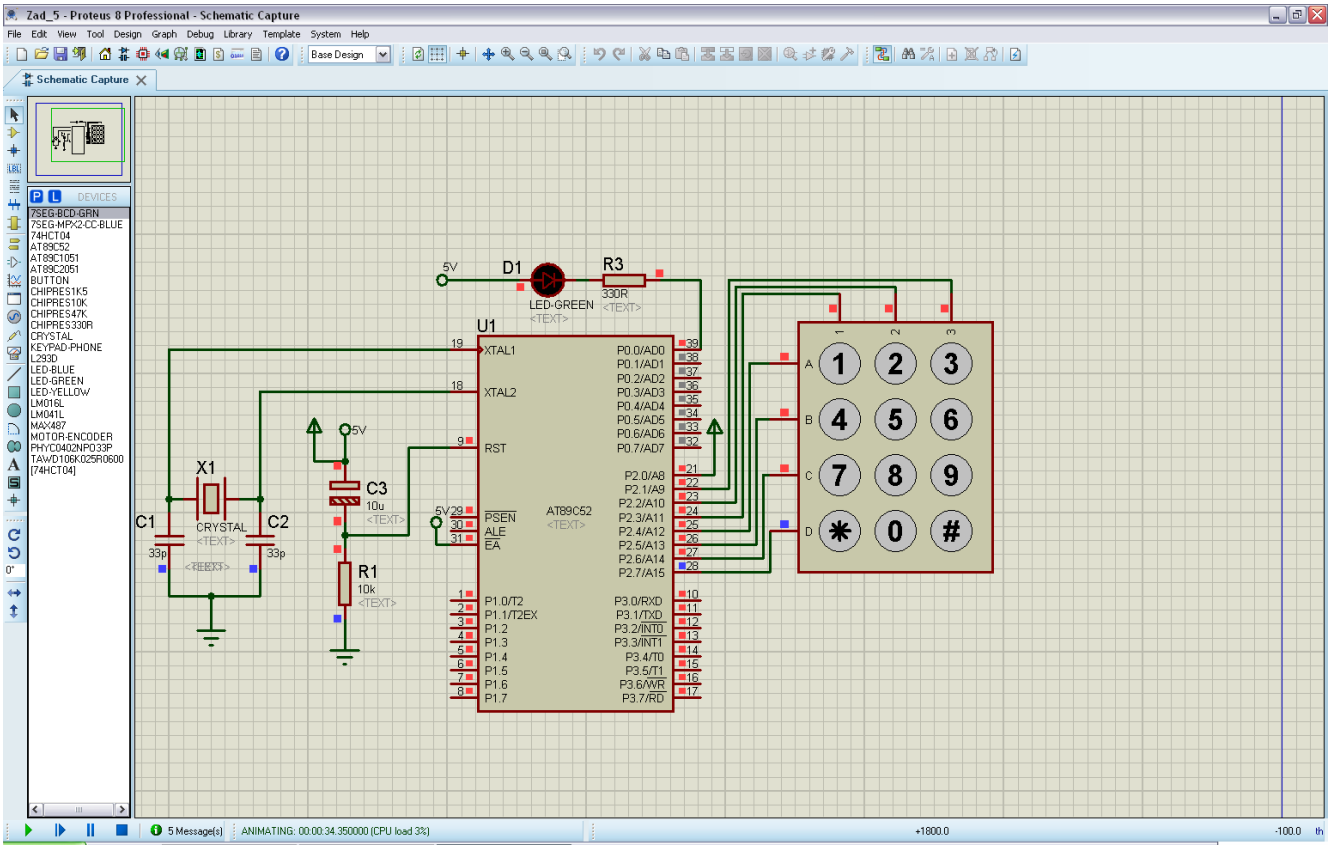


Prezentacja realizacji zadania przez program

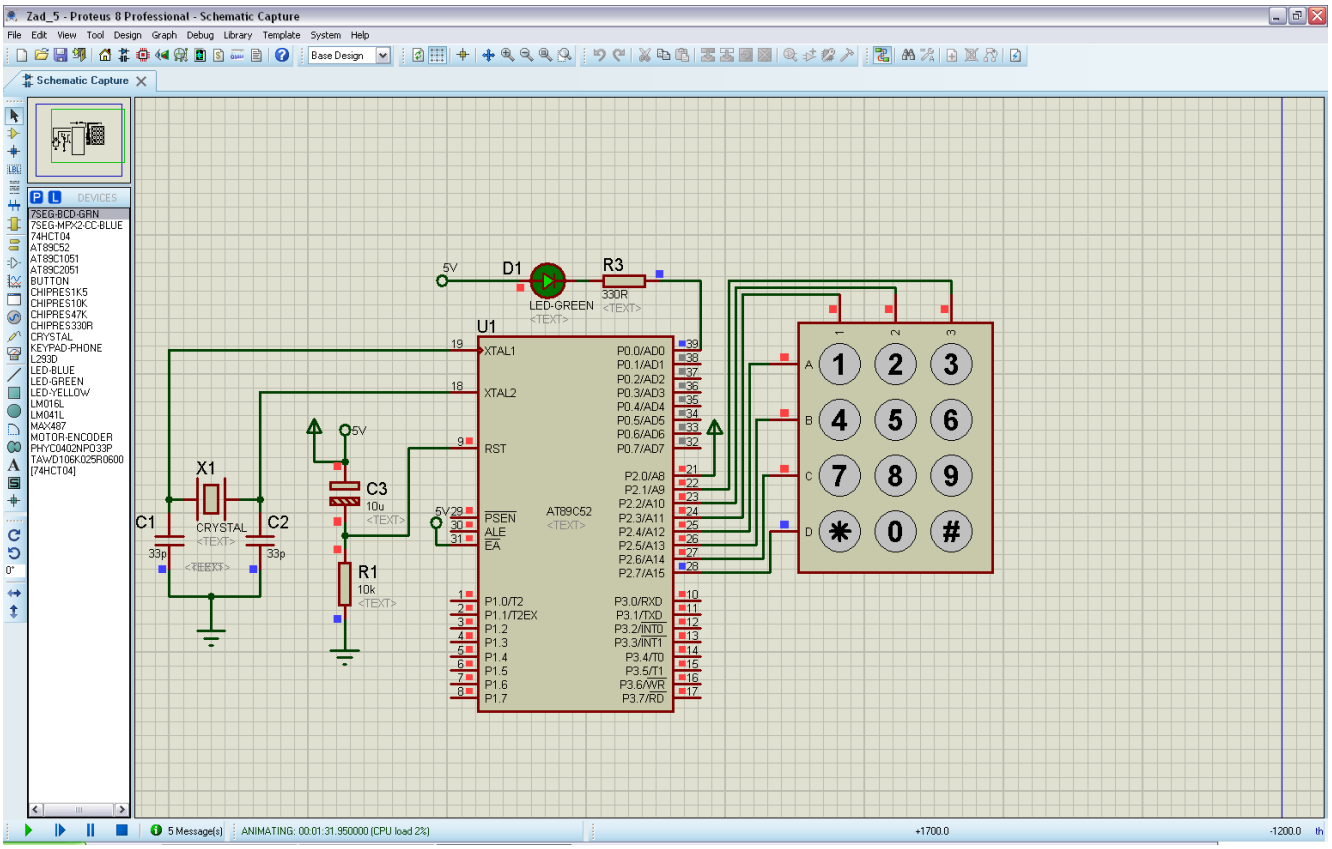
Stan początkowy, wędrujące zero



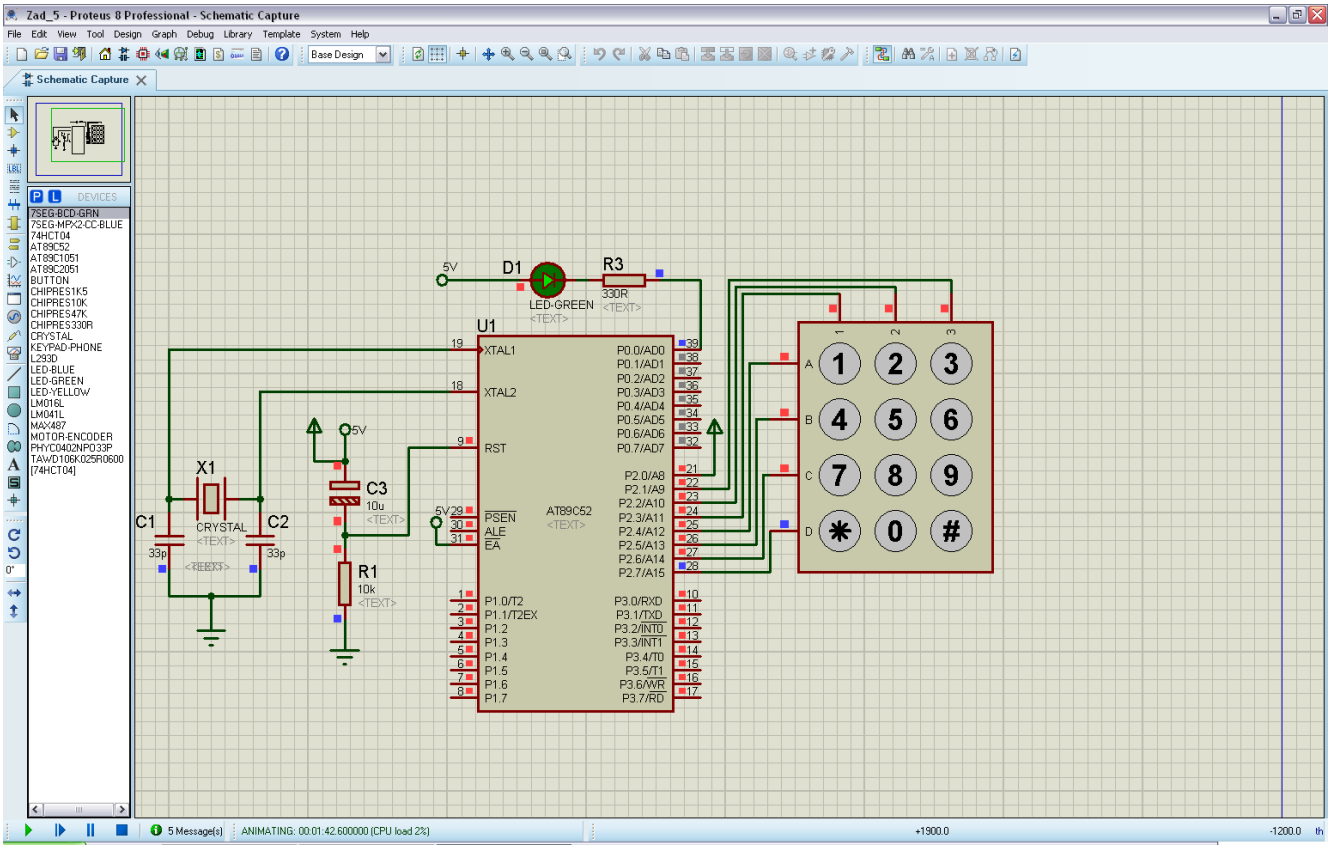
Wprowadzenie nieprawidłowego kodu oraz zatwierdzenie (brak zmian)



Wprowadzenie poprawnego kodu i zatwierdzenie * (błąd).



Wprowadzenie poprawnego kodu i zatwierdzenie # (otwarcie).


Zad_5 - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
DEVICES
7SEG-BCD-GRN
7SEG-MPX2-CC-BLUE
74HCT04
AT89C52
AT89C1051
AT89C2051
BUTTON
CHIPRES1K5
CHIPRES10K
CHIPRES47K
CHIPRES330R
CRYSTAL
KEYPAD-PHONE
L293D
LED-BLUE
LED-GREEN
LED-YELLOW
LM016L
LM041L
MAX487
MOTOR-ENCODER
PHYC0402NP033P
TAWD106K025R0600
[74HCT04]
5V
D1
LED-GREEN
R3
330R
U1
XTAL1
XTAL2
RST
PSEN
ALE
EA
AT89C52
P0.0/AD0
P0.1/AD1
P0.2/AD2
P0.3/AD3
P0.4/AD4
P0.5/AD5
P0.6/AD6
P0.7/AD7
P2.0/A8
P2.1/A9
P2.2/A10
P2.3/A11
P2.4/A12
P2.5/A13
P2.6/A14
P2.7/A15
P1.0/T2
P1.1/T2EX
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.6/WR
P3.7/RD
X1
CRYSTAL
C1
33p
C2
33p
C3
10u
R1
10k
A 1 2 3
B 4 5 6
C 7 8 9
D * 0 #
5 Message(s)
ANIMATING: 00:01:42.600000 (CPU load 2%)
+1900.0
-1200.0