

Systemy wbudowane

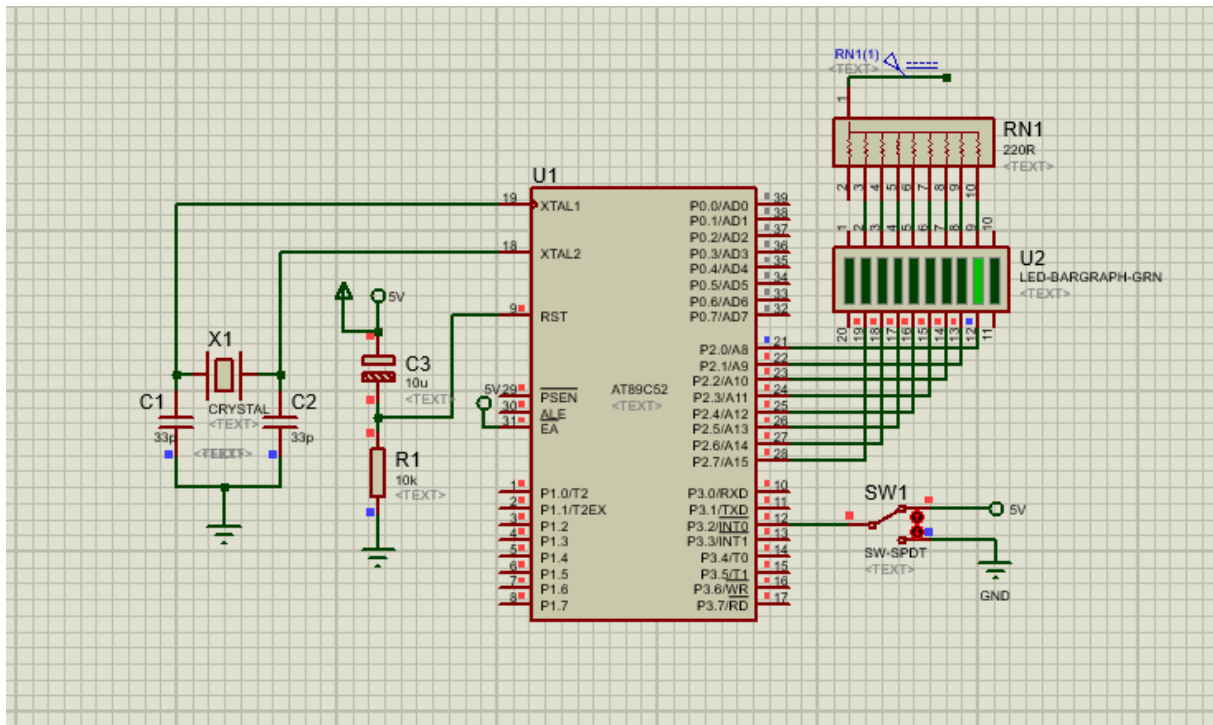
Lab1: Wytwarzanie i testowanie oprogramowania

Bartnicki Mateusz WCY20IX1N1

Data wykonania ćwiczenia: 24.04.2022 r.

Wykonano zadania na ocenę dst, db i bdb.

Zadanie na ocenę **dostateczną**: napisz program **prog1.c**, powodujący **cykliczne wędrujące zapalenie się pojedynczej diody** w grupie U2 od strony prawej (wyjście P2.0) do lewej (P2.7) z opóźnieniem, wygodnym do obserwacji (należy dobrać właściwy czas opóźnienia Delay()). Program nie reaguje na zmiany stanu P3.2.



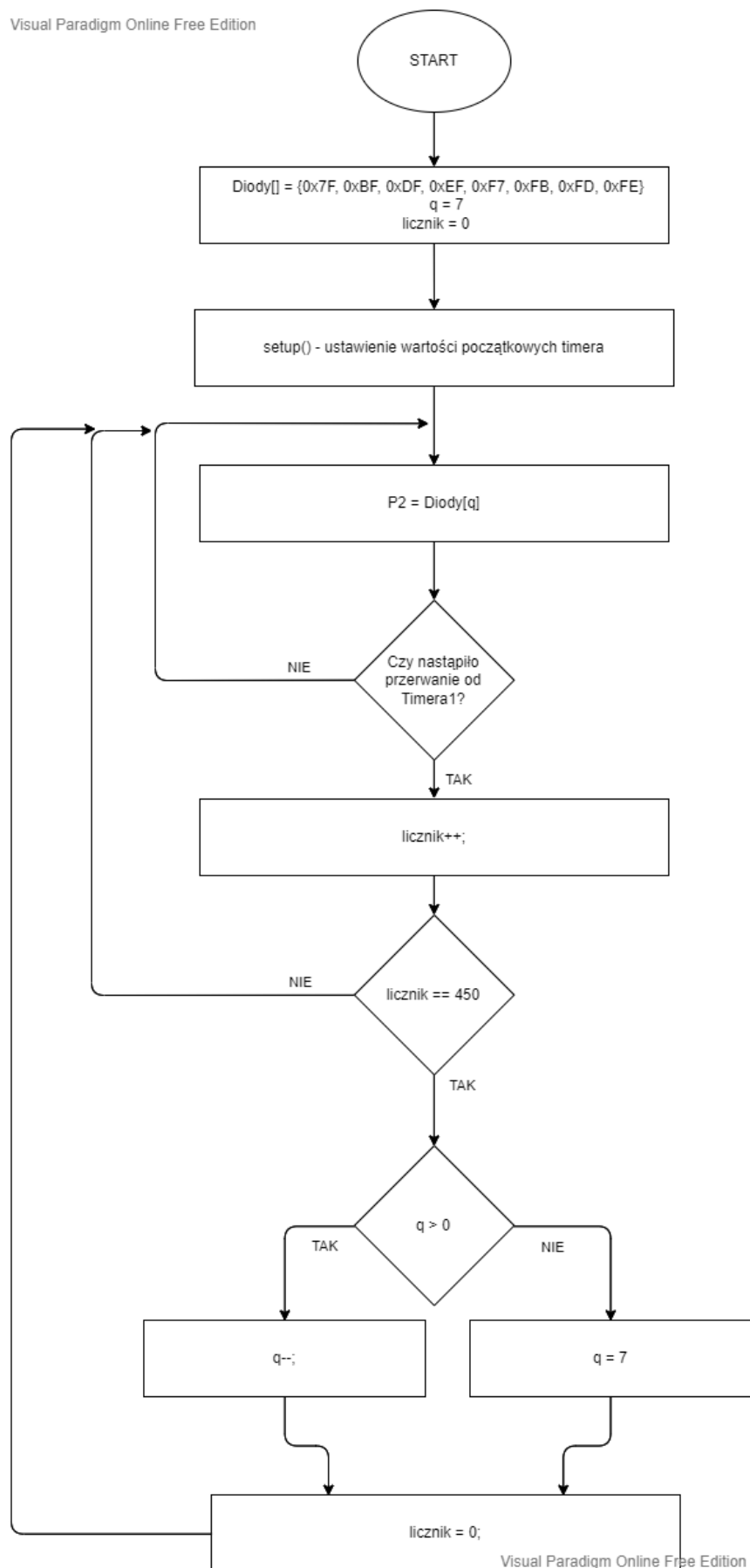
Zasadniczym problemem w realizacji zadania było odpowiednieysterowanie stanu portu P2 oraz zapewnienie odpowiedniego sterowania przerwaniami w taki sposób, aby wygodnie można było obserwować zapalenie się diod od strony prawej do strony lewej.

W celu poprawnej realizacji zadania przygotowano tablicę o nazwie Diody, która w każdym ze swoich elementów przechowuje takie ustawienia bitów portu P2, które pozwalają nam na zapalenie wybranej diody (licząc od lewej strony tj. dla Diody[0] port P2 przyjmuje wartość 0x7F, co na schemacie odpowiadać będzie zapaleniu diody podłączonej do P2.7).

Drugim elementem, który został wykorzystany przy odpowiedniej implementacji rozwiązania jest wykorzystania Timera1. Przeładowanie timera z trybem 8 bitowym z automatyczną obsługą przeładowania następuje co około 2,26ms. W związku z powyższym przyjęto oszacowanie, że zmiana zapalanej diody będzie następować co sekundę, czyli konieczne musi zadeklarowanie zmiennej, które będzie iterowana o wartość 1 w górę z każdym przeładowaniem timera aż do wartości 450 (licznik iterowany jest do wartości 450). Po osiągnięciu wartości 450 następuje zmiana wartości q, co powoduje zmianę zapalanej diody, licznik jest zerowany i cykl się powtarza.

Schemat blokowy prog1.c opracowany za pomocą Visual Paradigm Online:

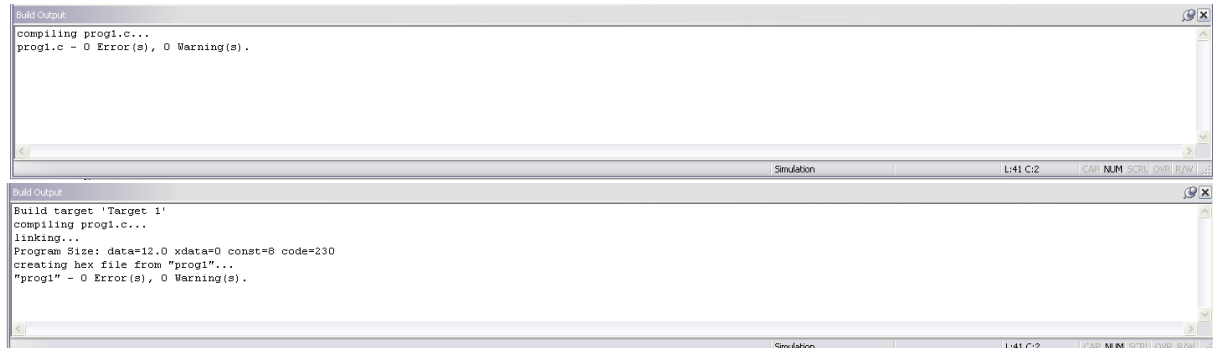
Visual Paradigm Online Free Edition



Treść programu:

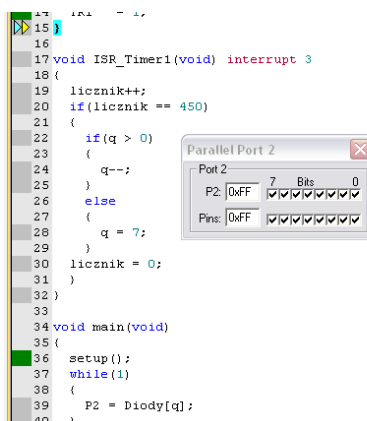
```
1 #include <REGX52.H>
2
3 unsigned char code Diody[] = {0x7F, 0xBF, 0xDF, 0xEF, 0xF7, 0xFB, 0xFD, 0xFE};
4 unsigned char q = 7;
5 unsigned int licznik = 0;
6
7 void setup(void)
8 {
9     TH1 = TL1 = 0x00;    // ustawienie wartosci początkowych timera
10    THOD = THOD & 0xDF;
11    THOD = THOD | 0x20;
12    ET1 = 1;              //włączenie pierwszego timera
13    EA = 1;
14    TR1 = 1;
15 }
16
17 void ISR_Timer1(void) interrupt 3
18 {
19     licznik++;
20     if(licznik == 450)
21     {
22         if(q > 0)
23         {
24             q--;
25         }
26         else
27         {
28             q = 7;
29         }
30     }
31     licznik = 0;
32 }
33
34 void main(void)
35 {
36     setup();
37     while(1)
38     {
39         P2 = Diody[q];
40     }
41 }
```

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

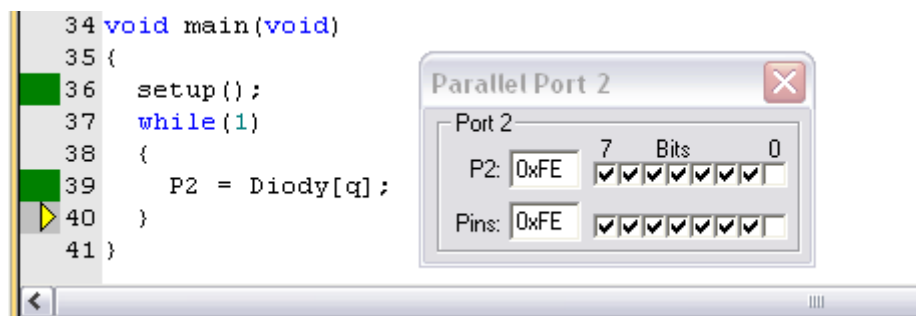


Zrzuty ekranu z debugera Keil:

Stan przed wykonaniem rozkazu P2 = Diody[q]

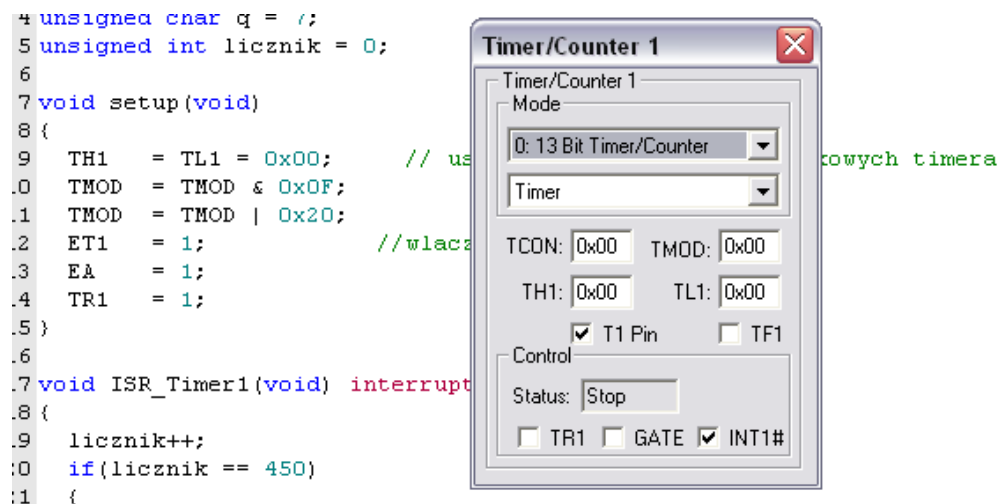


Oraz stan po wykonaniu rozkazu `P2 = Diody[q]`

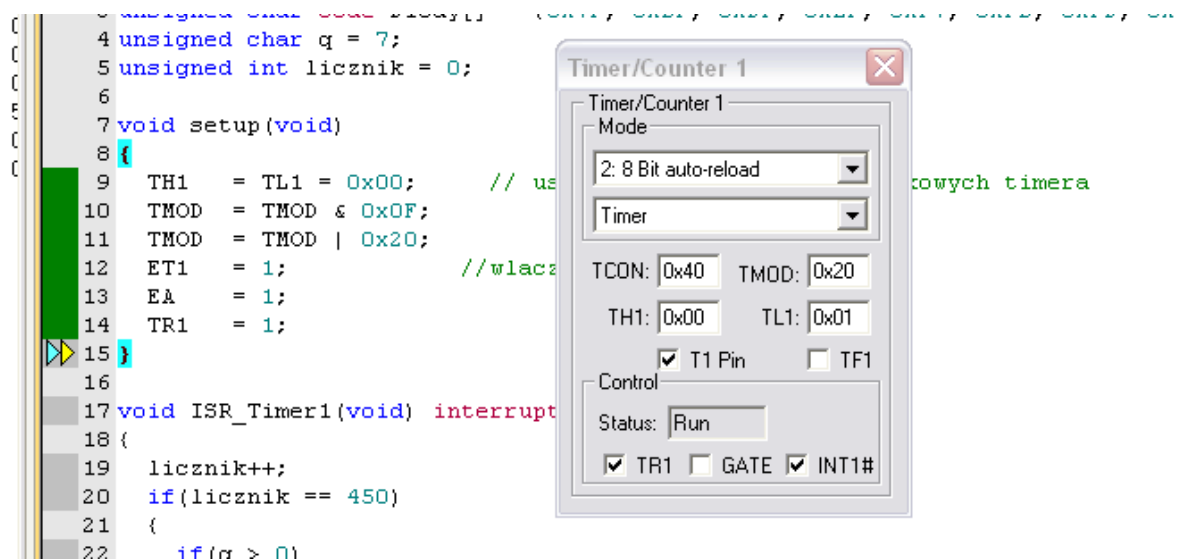


Znak ptaszka w danym polu oznacza, że wskazany w tym polu bit ma wartość 1, a puste pole oznacza wartość 0. Powyższe ustawienie bitów sugeruje nam, że zapaleniu ulegnie dioda podłączona do P2.0.

Ustawienia Timera 1 przed wykonaniem funkcji setup:



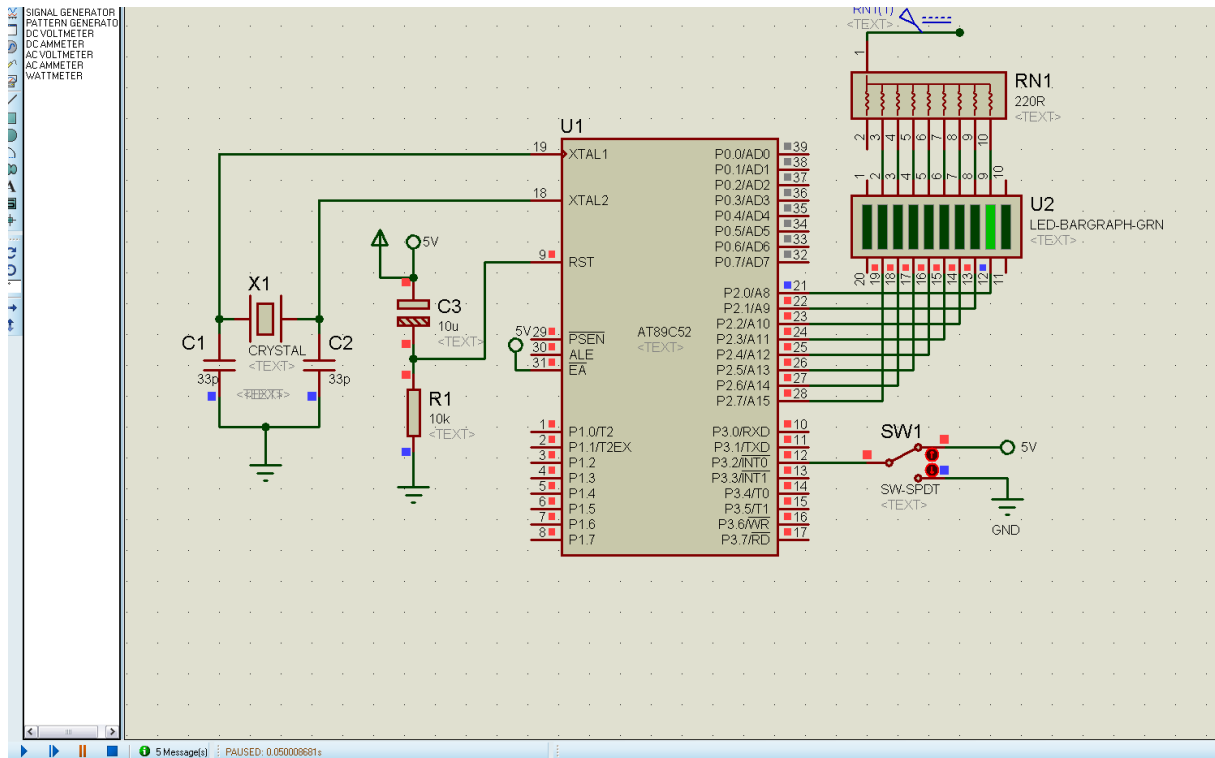
Ustawienia Timera 1 po wykonaniu funkcji setup:



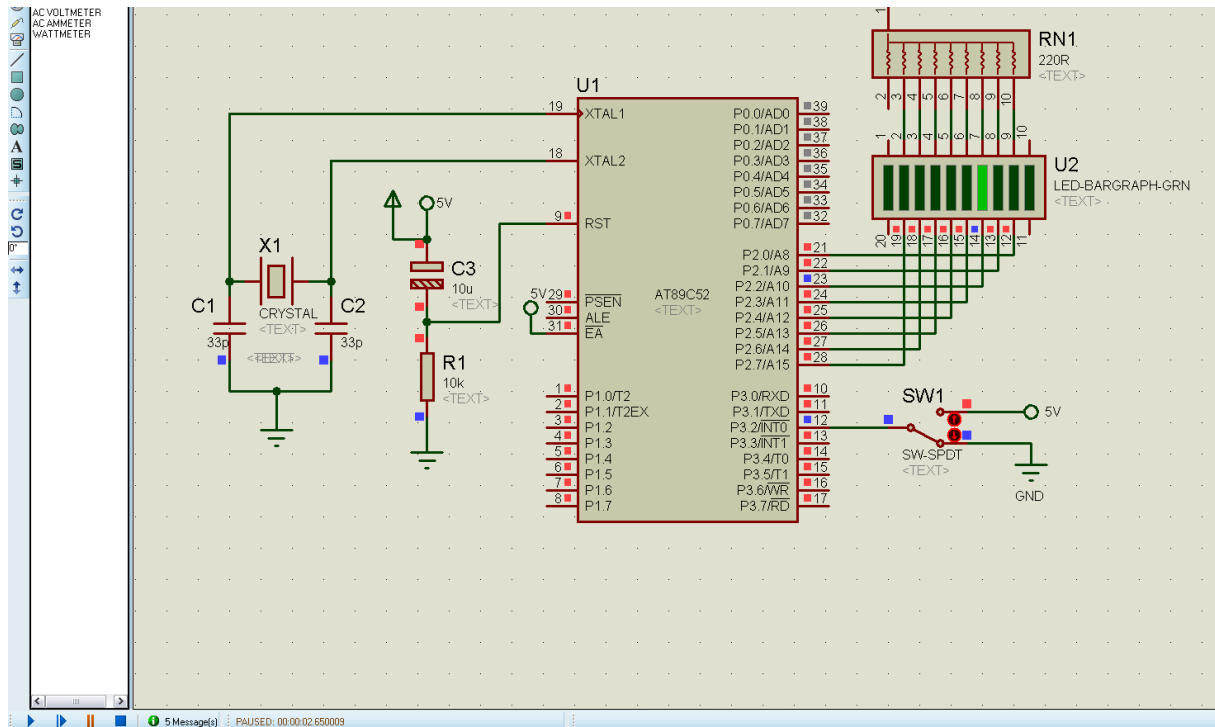
W funkcji następuje ustawienie timera jako 8 bitowy licznik z automatyczną obsługą przeładowania, jest on zerowany oraz na samym końcu włączany.

Zrzuty ekranu z symulatora Proteus:

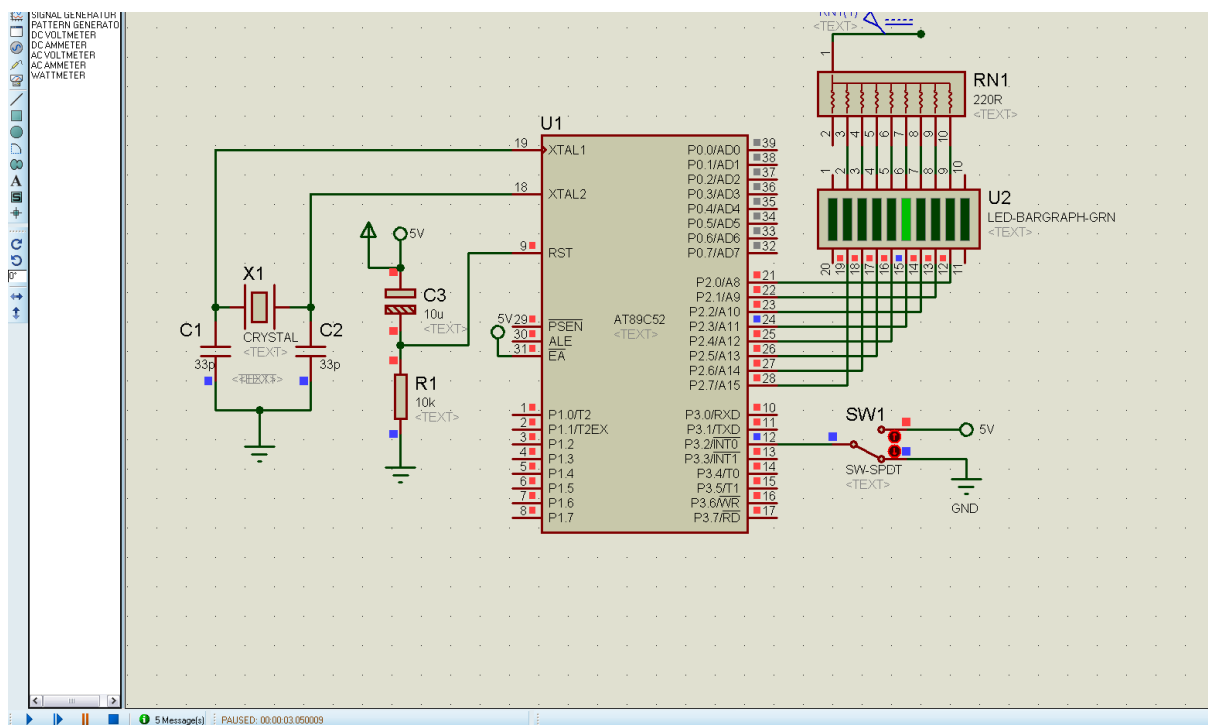
Po uruchomieniu programu zapalana jest dioda podłączona do P2.0



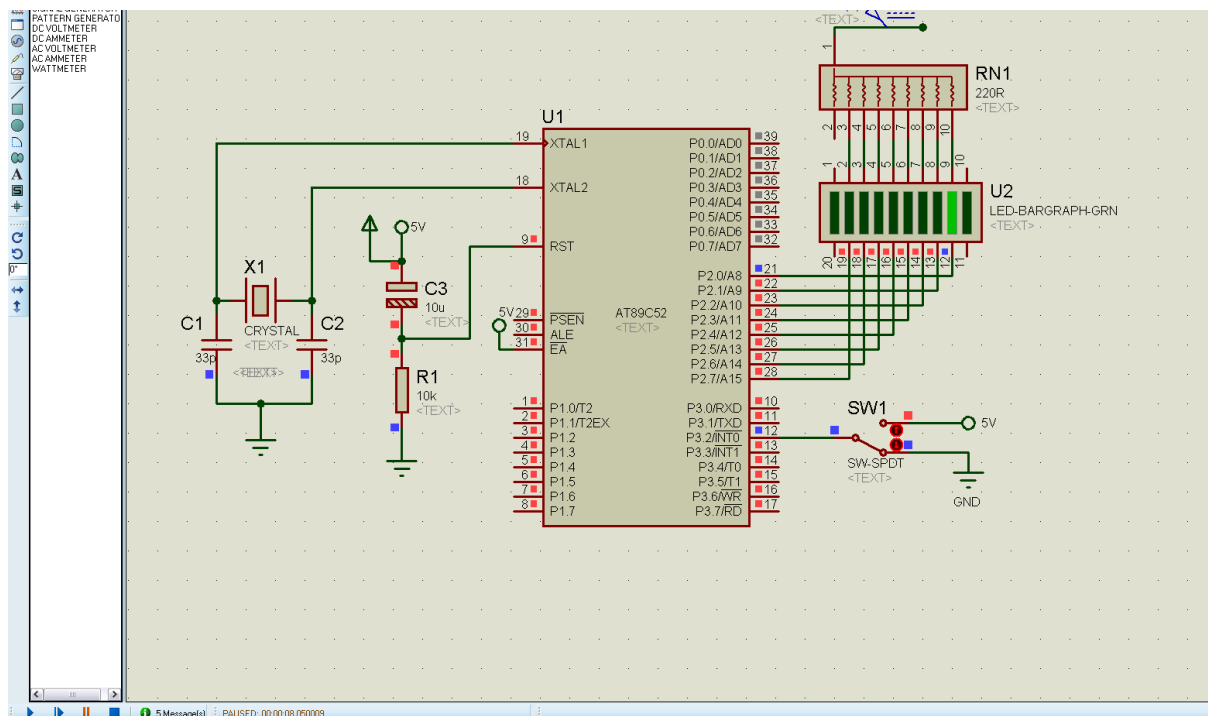
Przełączanie przycisku SW1 nie wpływa na zachowanie się układu:



Zgaszenie obecnie świecącej się diody oraz zapalenie kolejnej następuje po około 1 sekundzie:



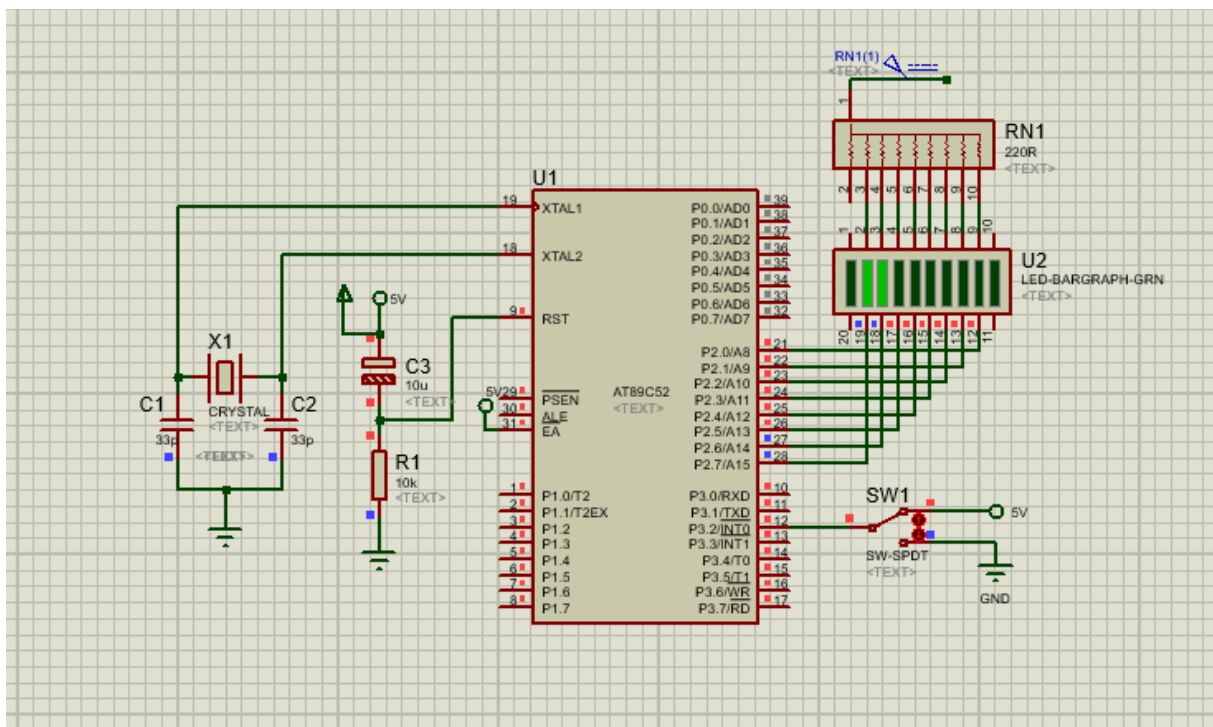
Przyjmując założenie, że każda dioda pali się ok. 1 sekundy, to cały cykl powtarza się co ok. 8 sekund, co zgodne jest z poniższym zrzutem ekranu:



Zadanie na ocenę **dobrą**: to co na ocenę dostateczną i ponadto napisz program **prog2.c**, powodujący cykliczne **wędrujące zapalanie się jednocześnie po 2 diody** w grupie U2 od strony lewej (wyjście P2.7) do prawej (wyjście P2.0), przy czym co druga zmiana diod ma się odbywać z podwójnym opóźnieniem (np. 1s, 2s, 1s, 2s itd.), a diody zapalają się z przeskokiem o 1 pozycję, czyli w 1 cyklu wykona się **7 kroków dla zapalania diod sąsiednich i 6 kroków dla zapalania diod przedzielonych jedną zgaszoną**. Program nie reaguje na zmiany stanu P3.2.

Studenci o numerach na liście grupy nieparzystych zapalają i przesuwają dwie sąsiednie diody [np. tak, jak na rysunku poniżej świecą kolejno diody pod numerami (2 3), (3 i 4) itd.] - **7 kroków**.

Studenci o numerach na liście grupy parzystych zapalają i przesuwają dwie diody, przedzielone jedną zgaszoną [np. na rysunku poniżej świecą kolejno diody pod numerami (2 i 4), (3 i 5) itd.] - **6 kroków**.



Zasadniczym problemem w realizacji zadania było odpowiednie wysterowanie stanu portu P2 (tak aby jednocześnie zapalały się dwie diody) oraz zapewnienie odpowiedniego sterowania przerwaniami w taki sposób, aby wygodnie można było obserwować zapalanie się diod od strony prawej do strony lewej. Dodatkowo należało zmodyfikować timer w taki sposób, aby co drugie przejście odbywało się z podwójnym opóźnieniem.

Jestem studentem o numerze nieparzystym dlatego też zapalam i przesuwam dwie sąsiednie diody (tak jak jest to pokazane na powyższym rysunku).

W celu poprawnej realizacji zadania przygotowano tablicę o nazwie Diody, która w każdym ze swoich elementów przechowuje takie ustawienia bitów portu P2, które pozwalają nam na zapalenie wybranej diody (licząc od lewej strony tj. dla Diody[0] port P2 przyjmuje wartość 0x7F, co na schemacie odpowiadać będzie zapaleniu diody podłączonej do P2.7).

Do poprawnego zapalania obu sąsiadujących ze sobą diod wykorzystano wartości zapisane w tablicy dioda oraz iloczyn logiczny. Np. dla 0x7F i 0xBF:

$$\begin{array}{r} 0111\ 1111 \\ \& \ 1011\ 1111 \\ \hline 0011\ 1111 \end{array}$$

Wynik wynosi 0x3F – co oznacza, że zapalone zostaną diody podłączone do P2.7 i P2.6.

Drugim elementem, który został wykorzystany przy odpowiedniej implementacji rozwiązania jest wykorzystania Timera1. Przeładowanie timera z trybem 8 bitowym z automatyczną obsługą przeładowania następuje co około 2,26ms. W związku z powyższym przyjęto oszacowanie, że zmiana zapalanej diody będzie następować co sekundę, czyli konieczne musi zadeklarowanie zmiennej, które będzie iterowana o wartość 1 w górę z każdym przeładowaniem timera aż do wartości obliczanej za pomocą równania:

$$\text{variant} * 450.$$

W związku z powyższym w celu zapewnienia odpowiedniego sterowania opóźnieniem w przesuwaniu diod wykorzystano dodatkową zmienną wariant, która na zmianę przyjmuje wartość 1 lub 2.

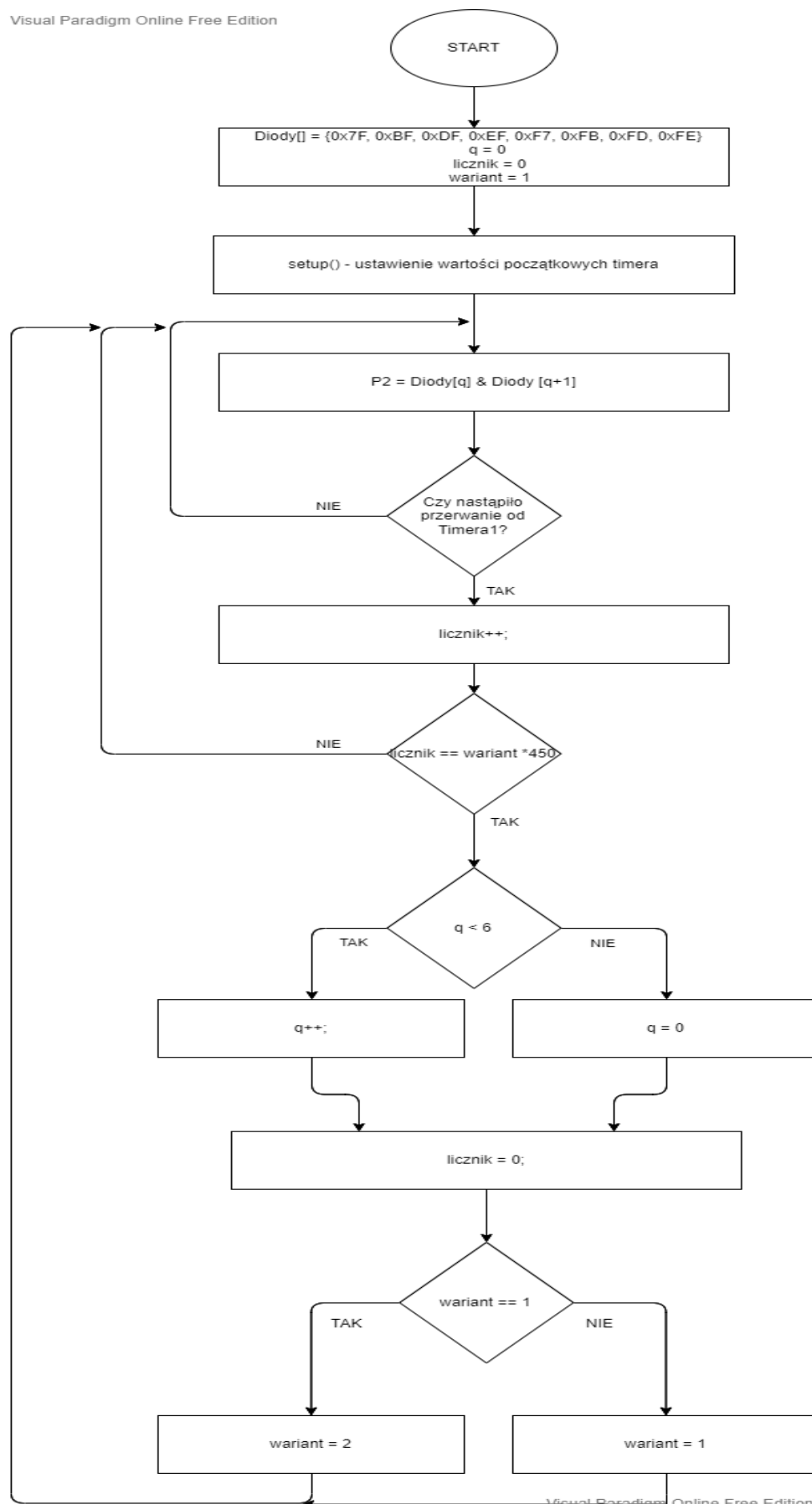
Po osiągnięciu wartości

$$\text{variant} * 450$$

następuje zmiana wartości (zwiększenie o 1) q, co powoduje zmianę zapalanych diod, licznik jest zerowany, wartość zmiennej wariant zmieniana jest na 1 lub 2 (w zależności od tego jaki wariant został wykorzystany w poprzednim cyklu – domyślnie program rozpoczyna się z wariantem 1) i cykl się powtarza.

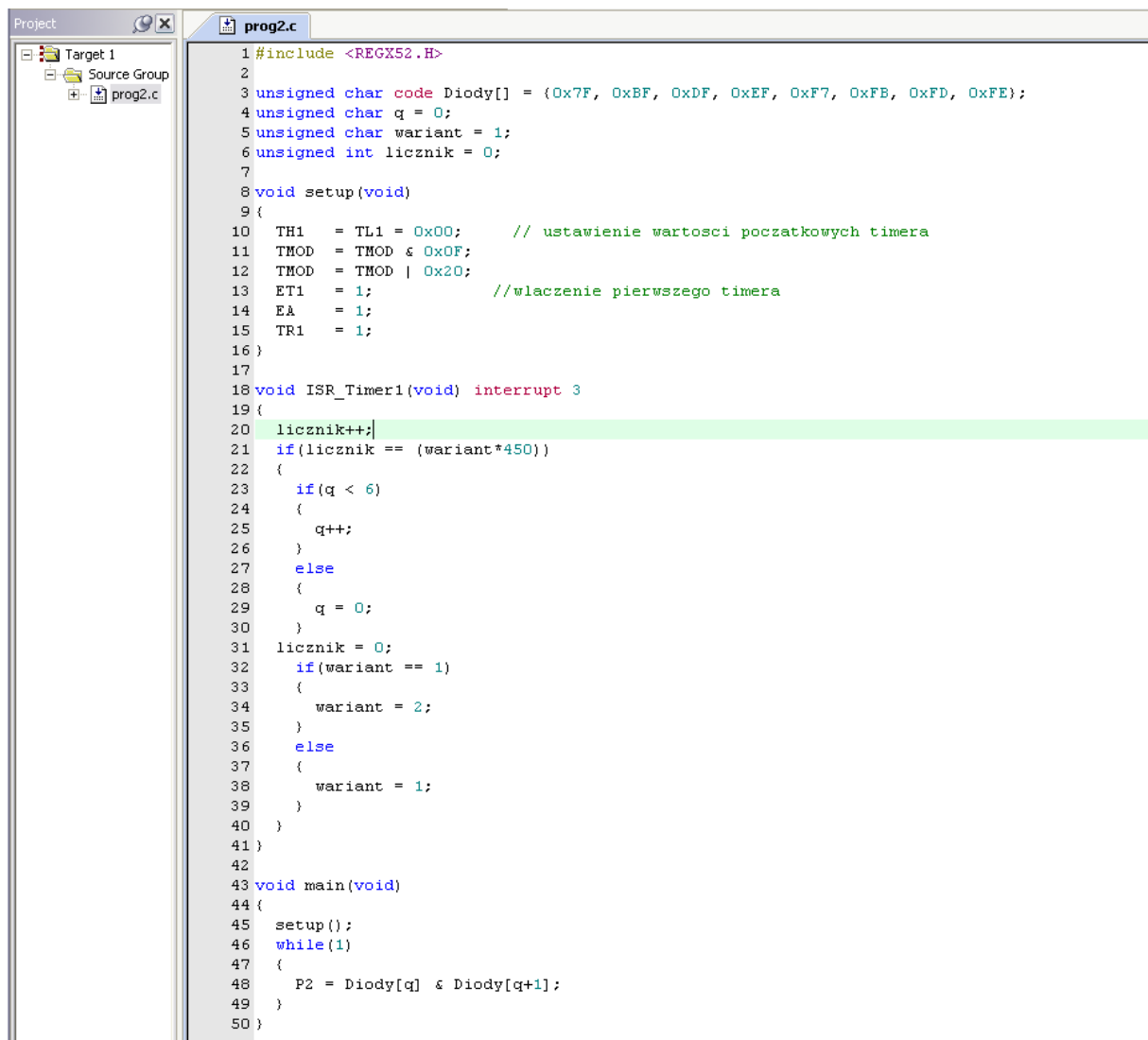
Schemat blokowy prog2.c opracowany za pomocą Visual Paradigm Online:

Visual Paradigm Online Free Edition



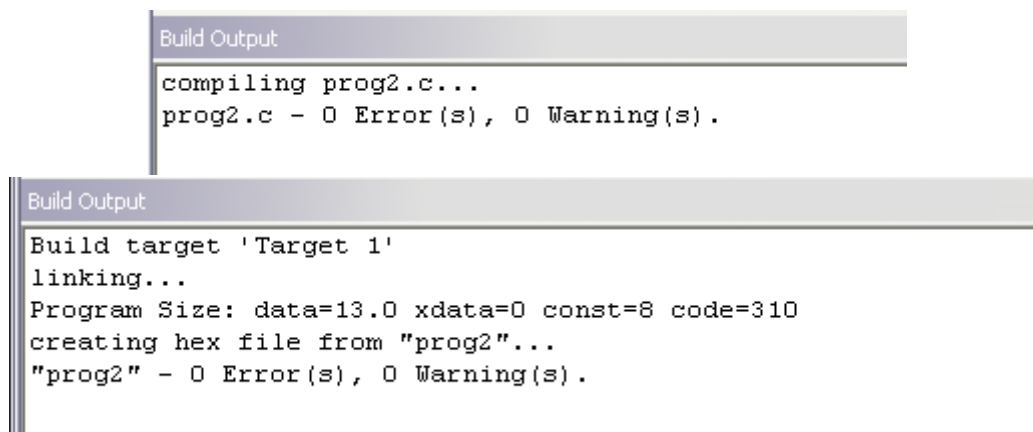
Visual Paradigm Online Free Edition

Treść programu:



```
1#include <REGX52.H>
2
3unsigned char code Diody[] = {0x7F, 0xBF, 0xDF, 0xEF, 0xF7, 0xFB, 0xFD, 0xFE};
4unsigned char q = 0;
5unsigned char wariant = 1;
6unsigned int licznik = 0;
7
8void setup(void)
9{
10    TH1 = TL1 = 0x00;    // ustawienie wartosci poczatkowych timera
11    TMOD = TMOD & 0x0F;
12    TMOD = TMOD | 0x20;
13    ET1 = 1;            //wlaczenie pierwszego timera
14    EA = 1;
15    TR1 = 1;
16}
17
18void ISR_Timer1(void) interrupt 3
19{
20    licznik++;
21    if(licznik == (wariant*450))
22    {
23        if(q < 6)
24        {
25            q++;
26        }
27        else
28        {
29            q = 0;
30        }
31        licznik = 0;
32        if(wariant == 1)
33        {
34            wariant = 2;
35        }
36        else
37        {
38            wariant = 1;
39        }
40    }
41}
42
43void main(void)
44{
45    setup();
46    while(1)
47    {
48        P2 = Diody[q] & Diody[q+1];
49    }
50}
```

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

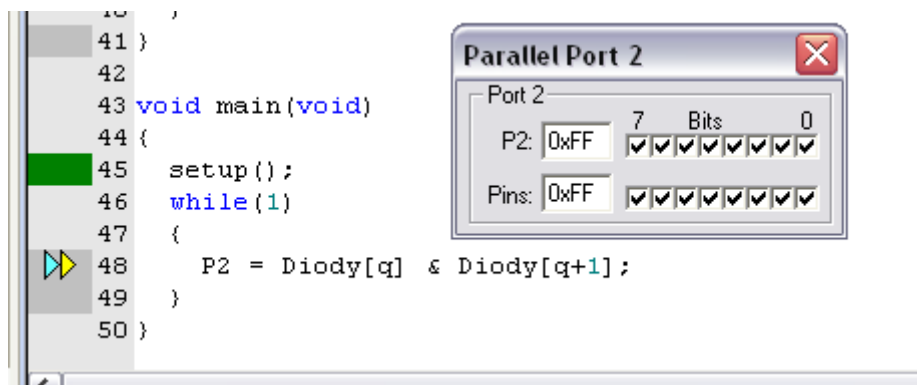


```
Build Output
compiling prog2.c...
prog2.c - 0 Error(s), 0 Warning(s).

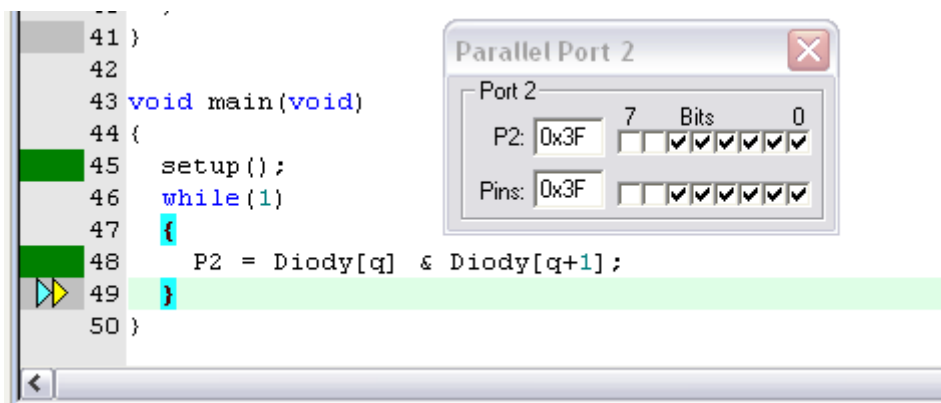
Build Output
Build target 'Target 1'
linking...
Program Size: data=13.0 xdata=0 const=8 code=310
creating hex file from "prog2"...
"prog2" - 0 Error(s), 0 Warning(s).
```

Zrzuty ekranu z debugera Keil:

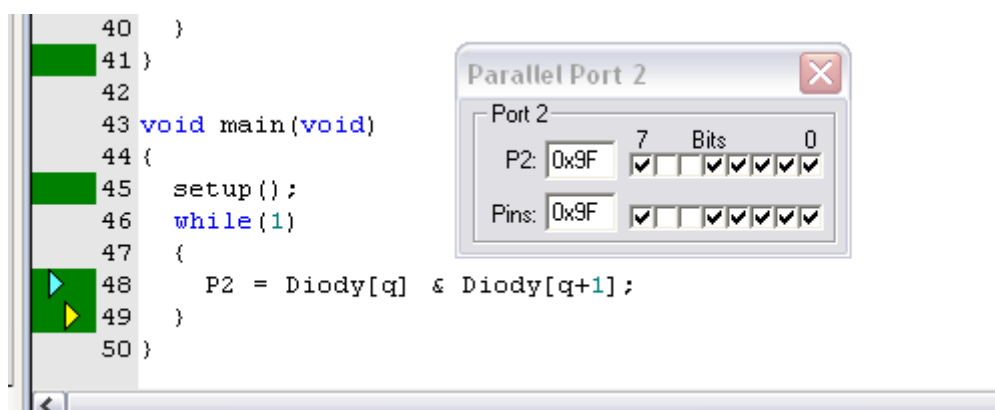
Zrzut ekranu na chwilę przed zmianą wartości na porcie P2:



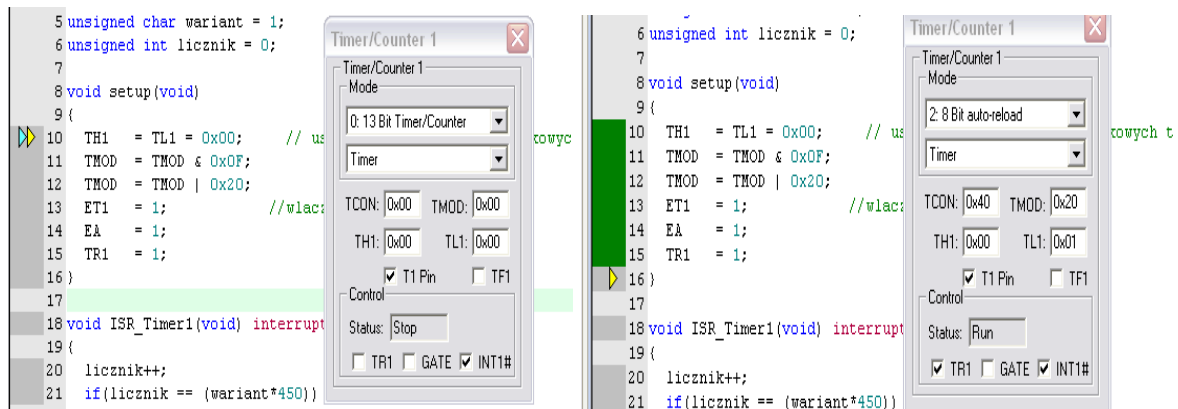
Wartości bitów ustawione na porcie P2 po wykonaniu operacji znajdującej się w 48 linijce. Powyższe ustawienie bitów sugeruje nam, że zapaleniu ulegną diody podłączona do P2.7 i P2.6. Wynik zgodny jest z przykładem przytoczmy w części opisującej rozwiązanie postawionego problemu.



Jak widać na poniższym zrzucie ekranu to następuje poprawne przejście zapalany diod po odpowiedniej ilości obsłużonych przerwań przez TIMER.



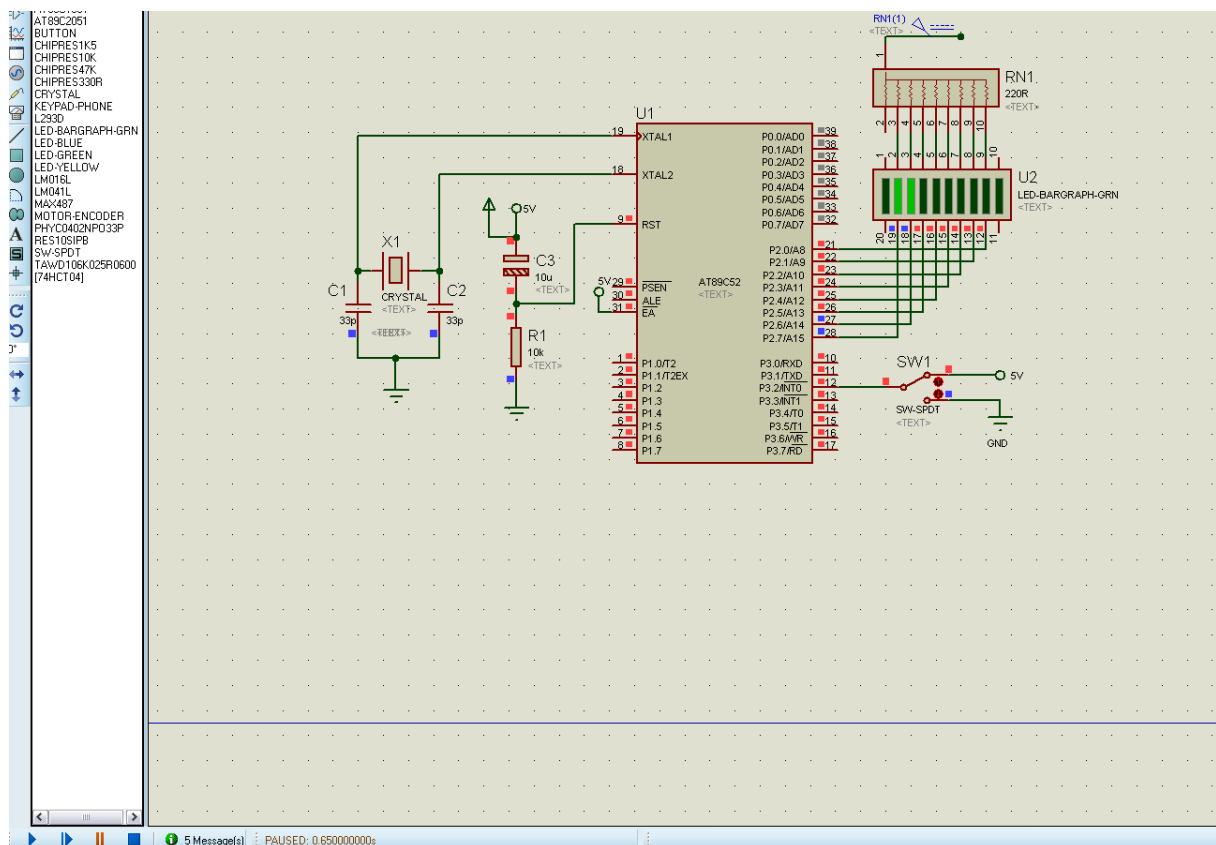
Tak samo jak dla powyżej przytoczonego programu prog1.c i w tym przypadku następuje odpowiednie ustawienie Timera1 na samym początku programu:



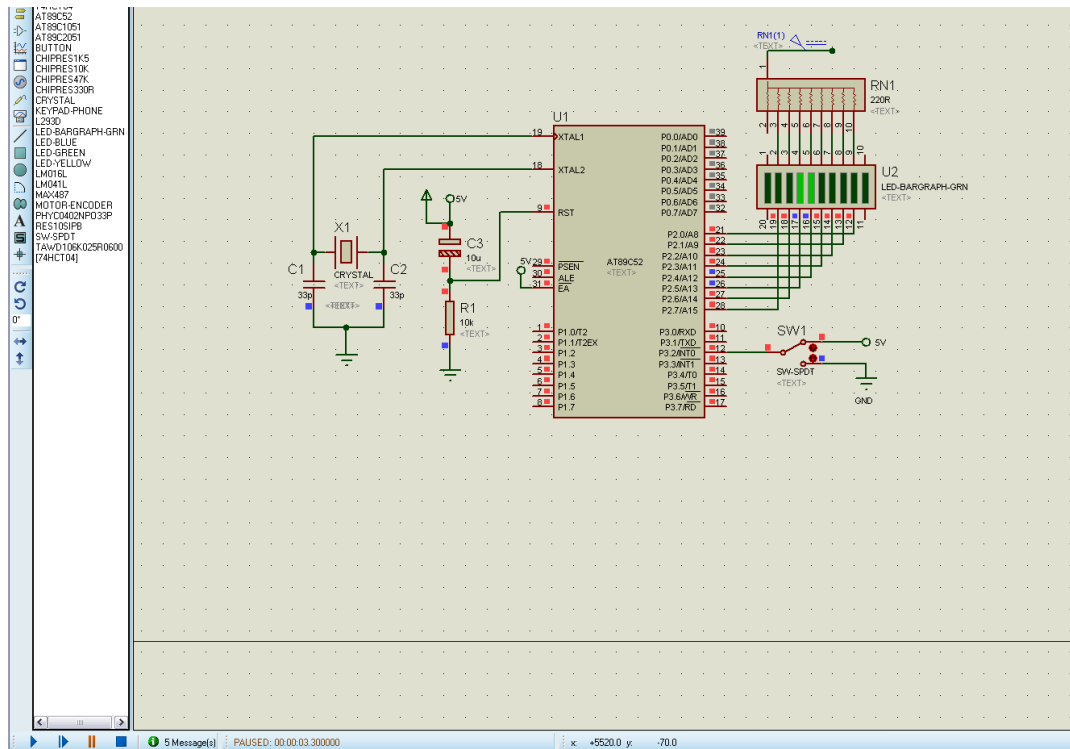
W funkcji następuje ustawienie timera jako 8 bitowy licznik z automatyczną obsługą przeładowania, jest on zerowany oraz na samym końcu włączany.

Zrzuty ekranu z symulatora Proteus:

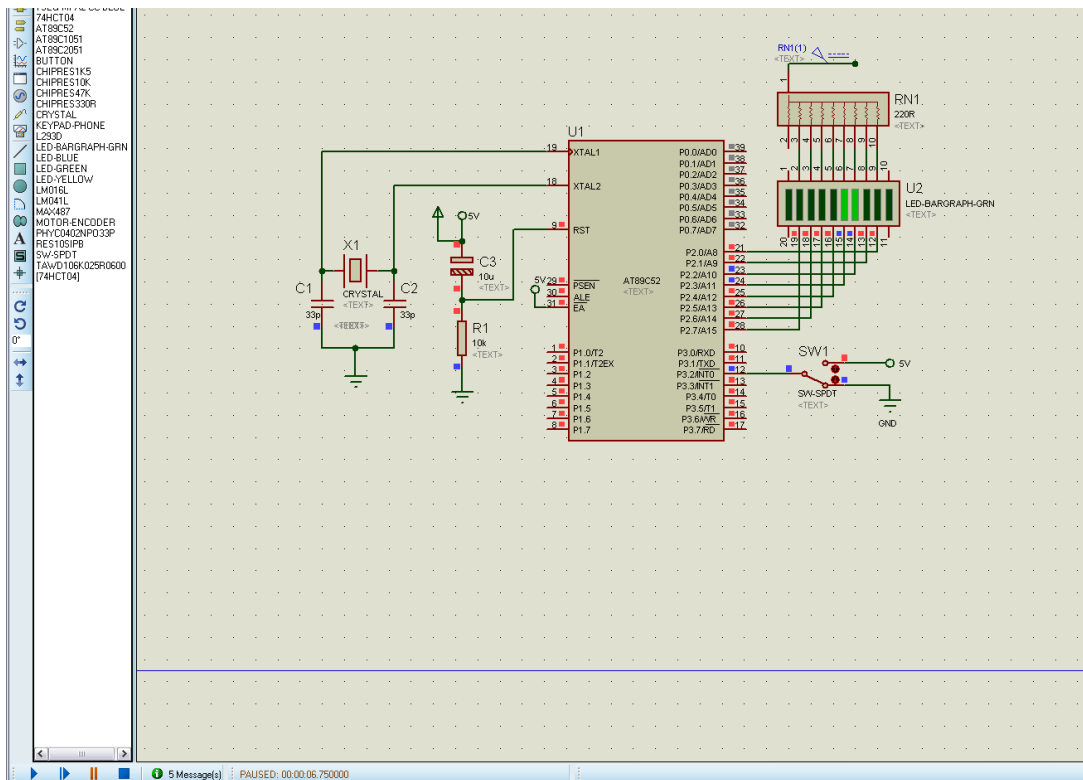
Na samym początku programu zapalana jest dioda podłączona do P2.7 i P2.6. Stan ten będzie utrzymywał się przez ok. 1 sekundę:



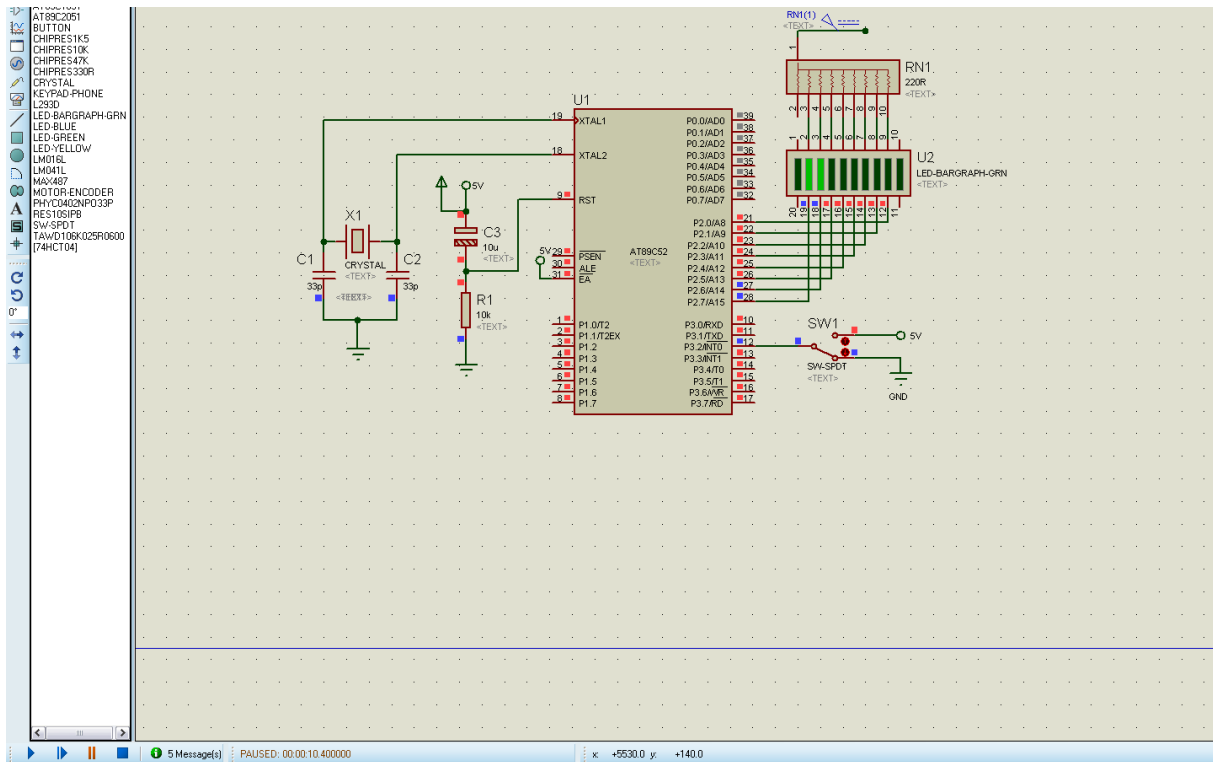
Następnie zgodnie z założeniami zadania dwie kolejne diody (P2.6 i P2.5) zapalane będą przez 2 sekundy. Potwierdza to fakt, że dopiero po ok 3. Sekundach działania programu zapalane są diody podłączone do P2.5 i P2.4.



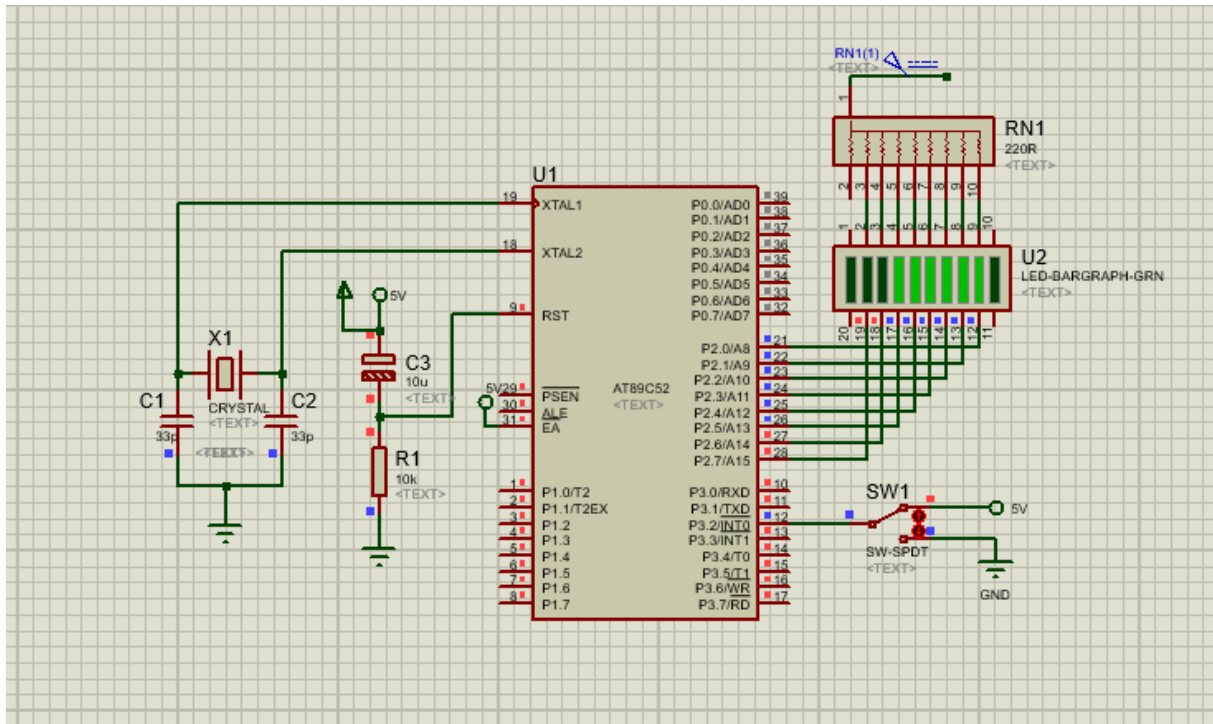
Zmiana ustawienia przełącznika SW1 nie wpływa na działanie wgranego programu:



Zapalenie diod zapętlilo się po ok. 10s. działania programu, co potwierdza, że przejście diod zachodziło na zmianę – raz co ok. 1s, a raz co ok. 2s.



Na ocenę **bardzo dobrą**: to co na ocenę dobrą i ponadto napisz program **prog3.c**, w którym cyklicznie **diody zapalają się pojedynczo, a już zapalone nie gasną** (coraz szerszy świecący pasek), w kolejności od początku prawej do lewej (Led 9 --> Led 2) **gdy P3.2 = 1** i od początku lewej do prawej (Led 2 --> Led 9) **gdy P3.2 = 0**. Po dojściu do końca paska LED wszystkie diody na chwilę (proszę wybrać takie opóźnienie, aby można było to wygodnie zaobserwować) gasną, a proces zapalania LED się powtarza. Sprawdzanie stanu przełącznika SW1, podłączonego do P3.2 ma się odbywać w każdej iteracji pętli tak, aby w trakcie symulacji można było zmieniać kierunek narastania świecącego paska przez zmianę stanu P3.2 - w takiej sytuacji dotychczas zapalone diody gasną, a pasek wydłuża się od początku drugiej strony.



Zasadniczym problemem w realizacji zadania było odpowiednie wysterowanie stanu na porcie P2 i P3 w taki sposób, aby w zależności od stanu portu P3 następowało zapalanie kolejnych po sobie diod od lewej lub od prawej. Dodatkowo należało opracować taką obsługę Timera, aby zapewniona była możliwość obserwacji zachowania układu.

Przełącznik SW1 jest podłączony do takiego portu P3, że teoretycznie można zastanowić się nad wykorzystaniem obsługi przerwań do sterowania zapalaniem diod. Jednakże dla tej implementacji układu jest to ślepy zaułek, ponieważ, gdy układ wykryje przerwanie na porcie P3.2 to będzie się ono wykonywało cały czas aż do momentu kolejnego przełączenia przycisku. Dlatego też w celu wykonania zadania wykorzystano fakt, że gdy SW1 podłączony jest do źródła napięcia to port P3 ma wartość 0xFF natomiast, gdy jest podłączony do ziemi to ma wartość 0xFB.

W celu poprawnej realizacji zadania przygotowano tablicę o nazwie Diody, która w każdym ze swoich elementów przechowuje takie ustawienia bitów portu P2, które pozwalają nam na zapalenie wybranej diody (licząc od lewej strony tj. dla Diody[0] port P2 przyjmuje wartość 0x7F, co na schemacie odpowiadać będzie zapaleniu diody

podłączonej do P2.7). Dodatkowo zastosowano zmienną char q , która wykorzystywana jest do kolejnego zapalania diod. Zmienna reset wraz z wartością w bitach portu P3 odpowiedzialne są za zresetowanie działania układu w przypadku naciśnięcia przycisku SW1. Zmienna zgaśnij odpowiada za zgaszenie wszystkich zapalonych diod po dojściu do końca paska LED.

Sterowanie włączaniem diod odbywa się za pomocą iloczynu logicznego wyrażonego wzorem:

$$P2 = P2 \ \& \ \text{Diody}[q];$$

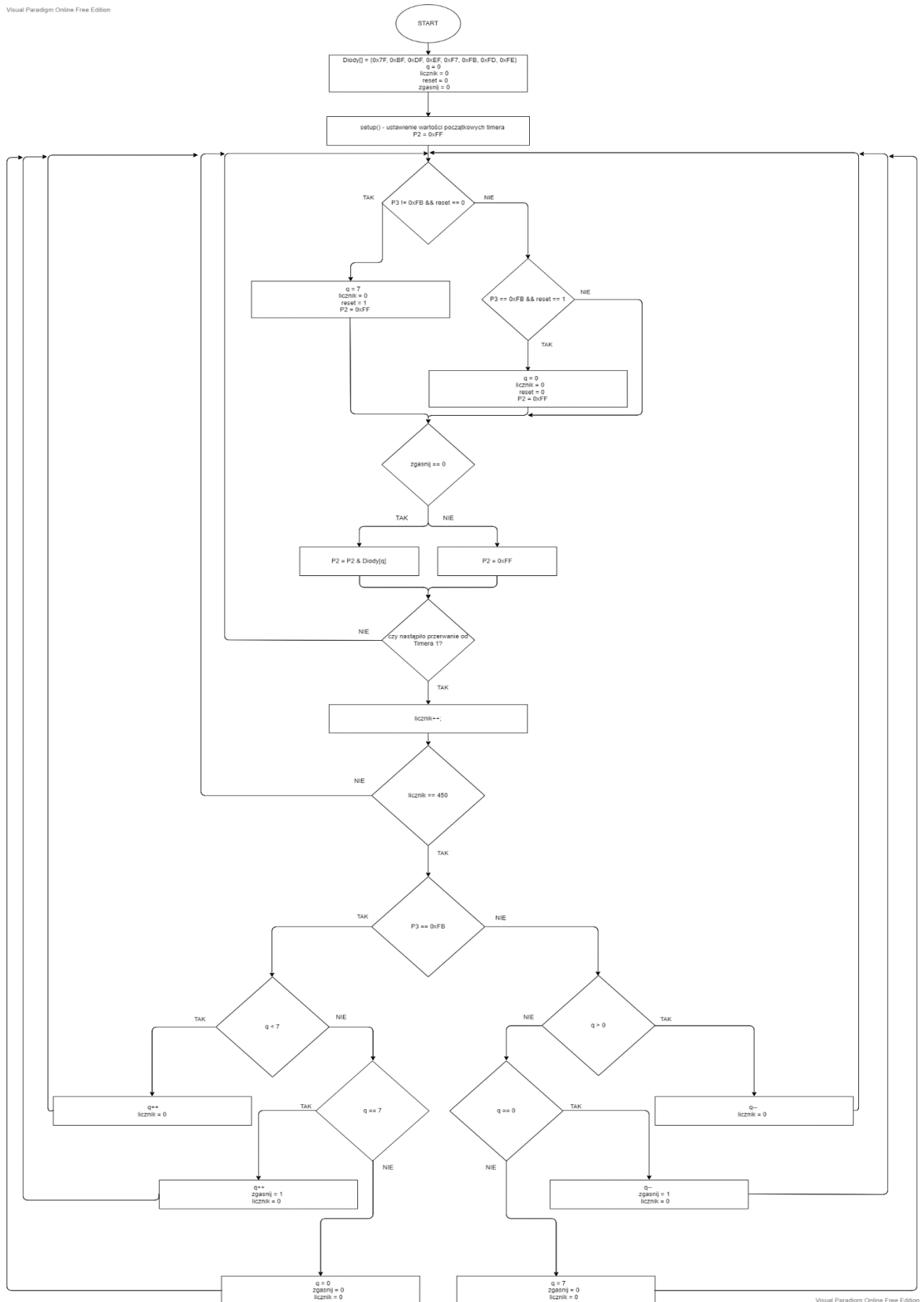
Do jego poprawnego działania niezbędne jest określenie wartości P2 na 0xFF na samym początku działania programu. Bez tego zabiegu nie jest możliwe poprawne funkcjonowanie układu.

Drugim elementem, który został wykorzystany przy odpowiedniej implementacji rozwiązania jest wykorzystanie Timera1. Przeładowanie timera z trybem 8 bitowym z automatyczną obsługą przeładowania następuje co około 2,26ms. W związku z powyższym przyjęto oszacowanie, że zmiana zapalanej diody będzie następować co sekundę, czyli konieczne musi być zadeklarowanie zmiennej, która będzie iterowana o wartość 1 w górę z każdym przeładowaniem timera aż do wartości 450 (licznik iterowany jest do wartości 450).

Po osiągnięciu wartości 450 następuje zmiana wartości q . W zależności od tego czy na porcie P3 odczytana została wartość 0xFB, lub jakakolwiek inna wartość to kolejno zmienna q jest zwiększana lub zmniejszana. Po osiągnięciu odpowiedniej wartości q w zależności od stanu portu P3 następuje chwilowe zgaszenie diod, po czym cykl się powtarza.

Schemat blokowy prog3.c opracowany za pomocą Visual Paradigm Online:

Visual Paradigm Online Free Edition



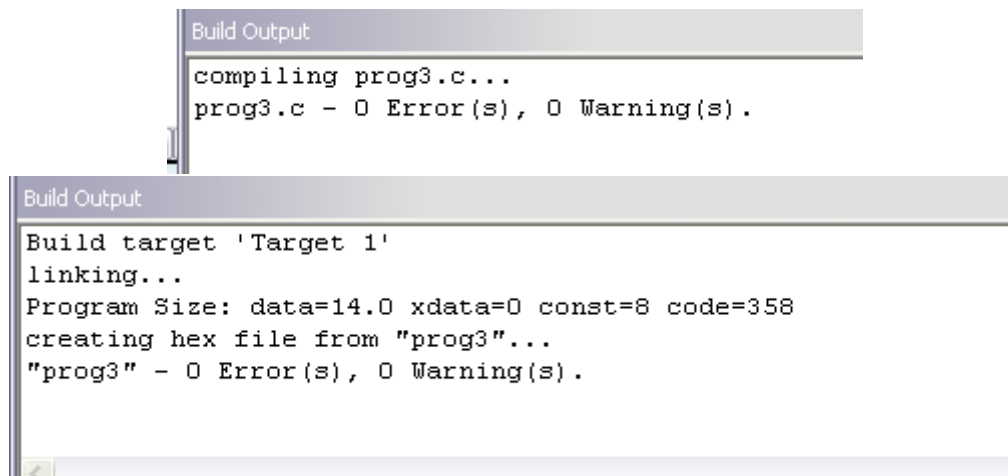
Visual Paradigm Online Free Edition

Treść programu:

```
Project
Target 1
Source Group
prog3.c

1#include <REG52.H>
2unsigned char code Diody[] = {0x7F, 0xBF, 0xDF, 0xEF, 0xF7, 0xFB, 0xFD, 0xFE};
3char q = 0;
4unsigned char zgasnij = 0;
5unsigned char reset = 0;
6unsigned int licznik = 0;
7void setup(void)
8{
9    TH1 = TL1 = 0x00;    // ustawienie wartosci poczatkowych timera
10    TMOD = TMOD & 0x0F;
11    TMOD = TMOD | 0x20;
12    ET1 = 1;            //wlaczenie pierwszego timera
13    EA = 1;
14    TR1 = 1;
15}
16void ISR_Timer1(void) interrupt 3
17{
18    licznik++;
19    if(licznik == 450)
20    {
21        if(P3 == 0xFB)
22        {
23            if(q < 7)
24            {
25                q++;
26            }
27            else if(q == 7)
28            {
29                zgasnij = 1;
30                q++;
31            }
32            else
33            {
34                q = 0;
35                zgasnij = 0;
36            }
37        }
38        else
39        {
40            if(q>0)
41            {
42                q--;
43            }
44            else if (q == 0)
45            {
46                zgasnij = 1;
47                q--;
48            }
49            else
50            {
51                q = 7;
52                zgasnij = 0;
53            }
54        }
55        licznik = 0;
56    }
57}
58void main(void)
59{
60    setup();
61    P2 = 0xFF;
62    while(1)
63    {
64        if(P3 != 0xFB && reset == 0 )
65        {
66            q = 7;
67            licznik = 0;
68            reset = 1;
69            P2 = 0xFF;
70        }
71        else if (P3 == 0xFB && reset == 1)
72        {
73            q = 0;
74            licznik = 0;
75            reset = 0;
76            P2 = 0xFF;
77        }
78        if(zgasnij == 0)
79        {
80            P2 = P2 & Diody[q];
81        }
82        else
83        {
84            P2 = 0xFF;
85        }
86    }
87}
```

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

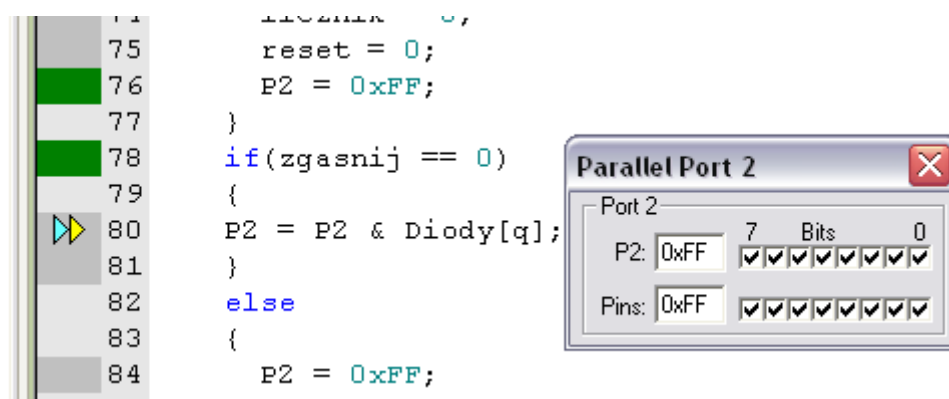


```
Build Output
compiling prog3.c...
prog3.c - 0 Error(s), 0 Warning(s).

Build Output
Build target 'Target 1'
linking...
Program Size: data=14.0 xdata=0 const=8 code=358
creating hex file from "prog3"...
"prog3" - 0 Error(s), 0 Warning(s).
```

Zrzuty ekranu z debugera Keil:

Stan portu P2 przed wykonaniem 80 linijki programu:



```
75     reset = 0;
76     P2 = 0xFF;
77 }
78 if(zgasnij == 0)
79 {
80     P2 = P2 & Diody[q];
81 }
82 else
83 {
84     P2 = 0xFF;
85 }
```

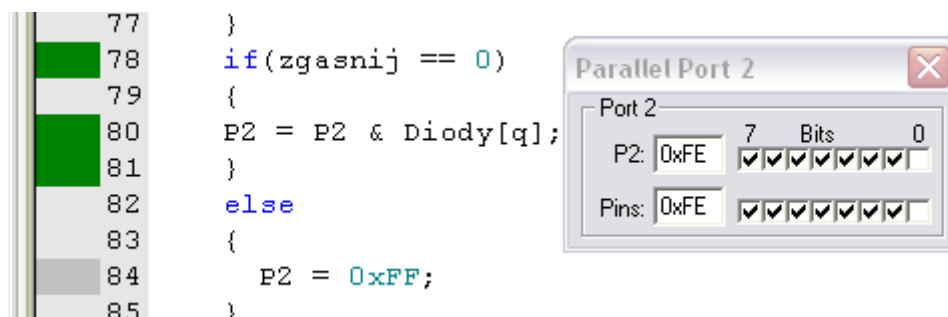
Parallel Port 2

Port 2

P2: 0xFF 7 Bits 0

Pins: 0xFF

Stan portu P2 po wykonaniu 80 linijki programu. Domyślnie w programie wartość P3 ustawiona jest na 0xFF. Odpowiada to sytuacji, w której diody będą zapalane od strony prawej do strony lewej. Potwierdza to poniższy zrzut ekranu:



```
77 }
78 if(zgasnij == 0)
79 {
80     P2 = P2 & Diody[q];
81 }
82 else
83 {
84     P2 = 0xFF;
85 }
```

Parallel Port 2

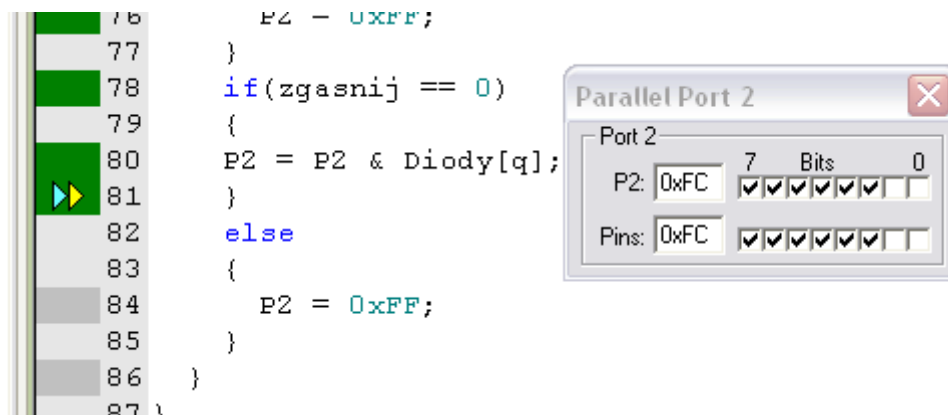
Port 2

P2: 0xFE 7 Bits 0

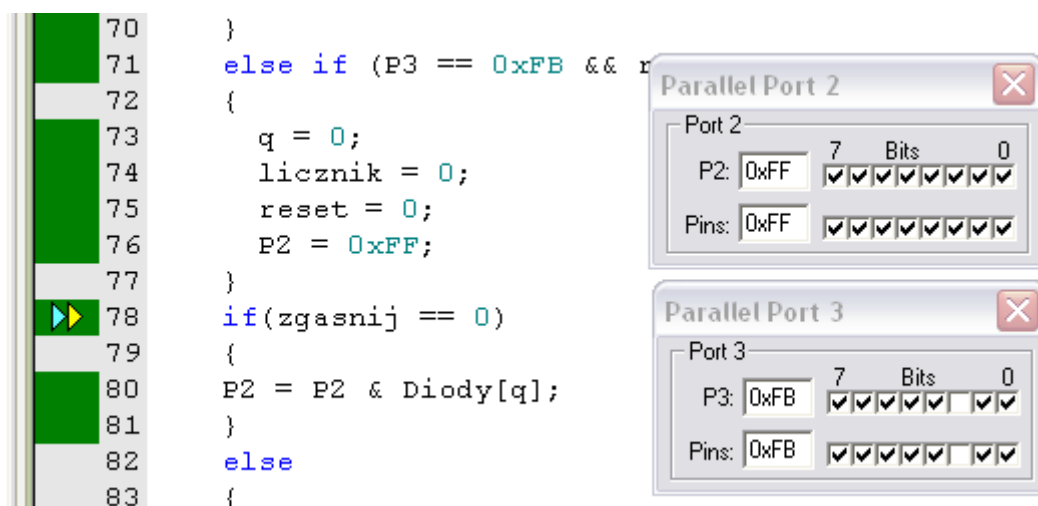
Pins: 0xFE

Po wymuszeniu przejścia do linijki 20 kodu (obsługa sytuacji, w której wartość licznik osiągnęła 450) możemy następnie zaobserwować kolejne wykonanie kodu

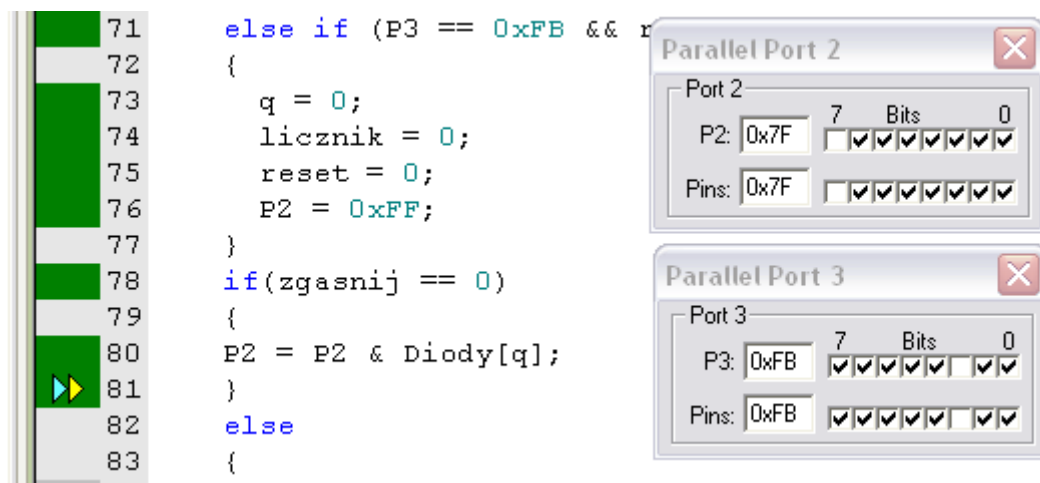
$P2 = P2 \& \text{Diody}[q]$ po zmniejszeniu wartości q :



Następnie po wymuszeniu wartości 0xFB na porcie P3 w debuggerze możemy zaobserwować wykonanie pętli warunkowej odpowiadającej zresetowaniu układu do stanu początkowego (wartość q została zmieniona na 7):



Następnie układ kontynuuje swoją prawidłową pracę:



Po wymuszeniu przejścia do linijki 20 kodu (obsługa sytuacji, w której wartość licznik osiągnęła 450) możemy następnie zaobserwować kolejne wykonanie kodu

$P2 = P2 \& \text{Diody}[]$ po zwiększeniu wartości q :

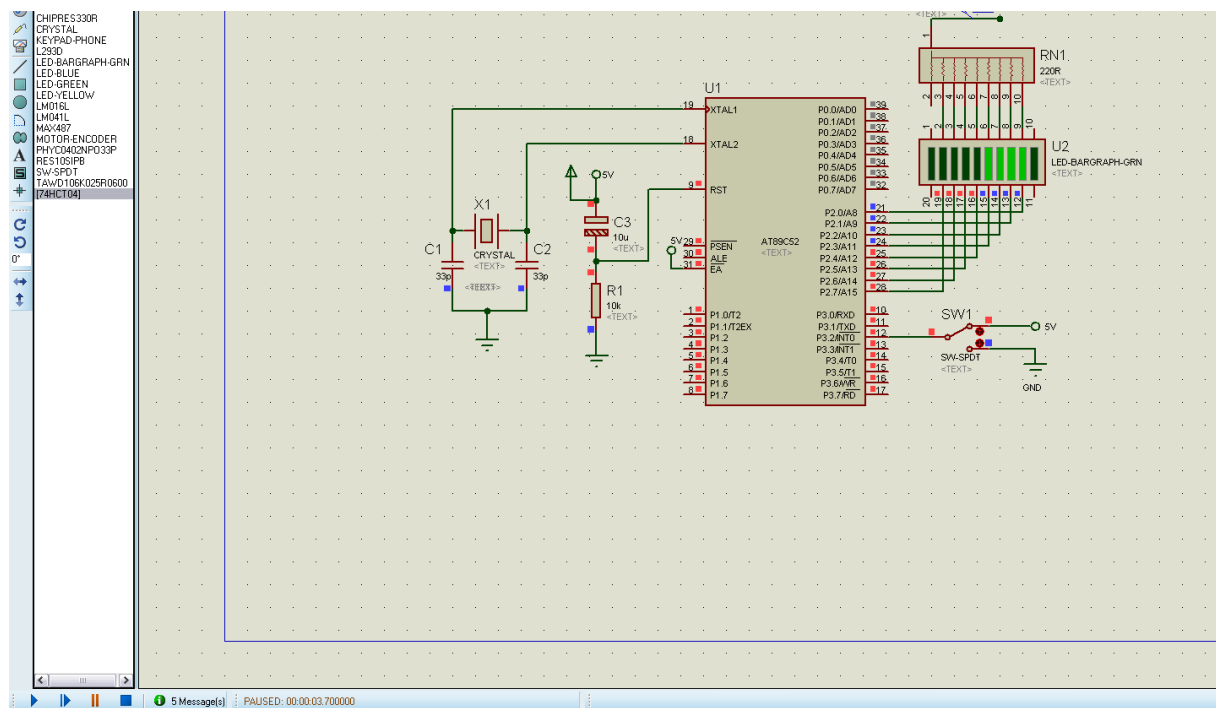
```

70
71     else if (P3 == 0xFB && r
72     {
73         q = 0;
74         licznik = 0;
75         reset = 0;
76         P2 = 0xFF;
77     }
78     if(zgasnij == 0)
79     {
80         P2 = P2 & Diody[q];
81     }
82     else
83     {
84         P2 = 0xFF;

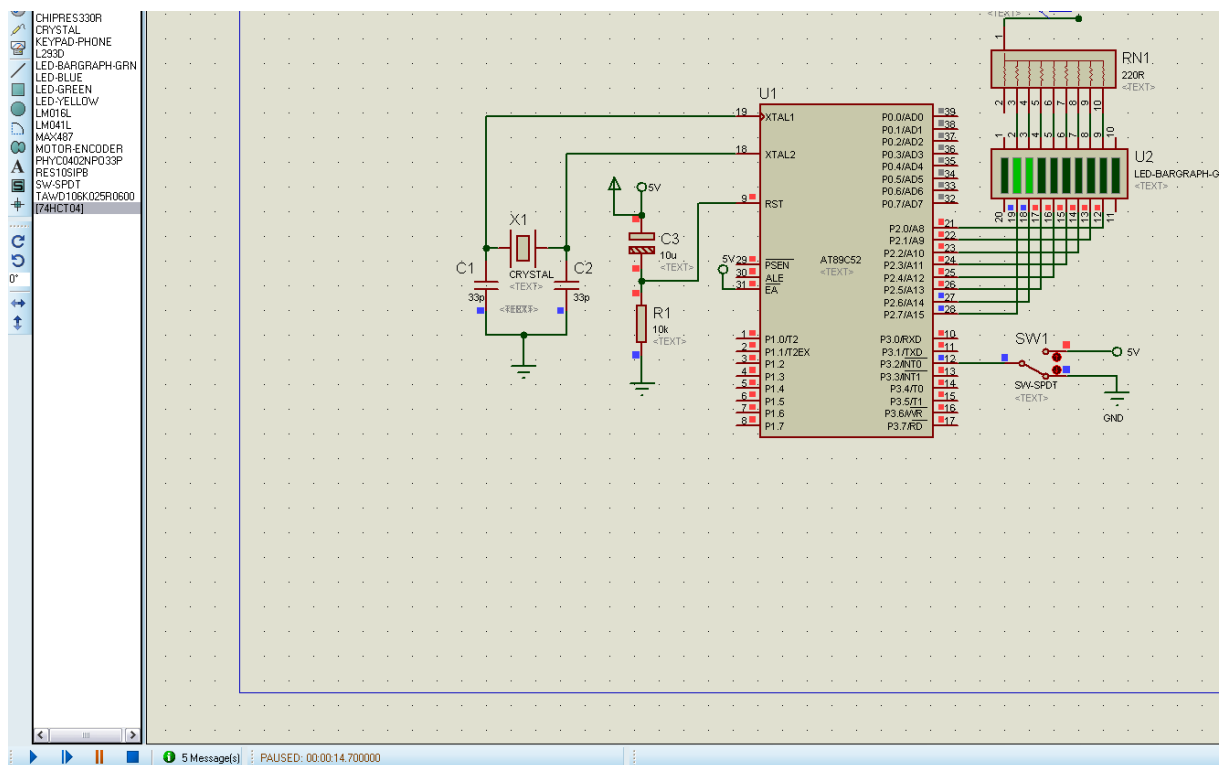
```

Zrzuty ekranu z symulatora Proteus:

Dla stanu układu, w którym $P3 = 0xFF$ następuje zapalanie kolejnych po sobie diod od strony prawej do strony lewej:



Po chwili od zgaszenia diod układ ponownie rozpoczyna swoją pracę:



Dla przeciwnego przypadku działanie układu wygląda identycznie:

