Spis treści

1 Zapoznaj się z informacjami dotyczącymi portów P0, P1, P2, P3:

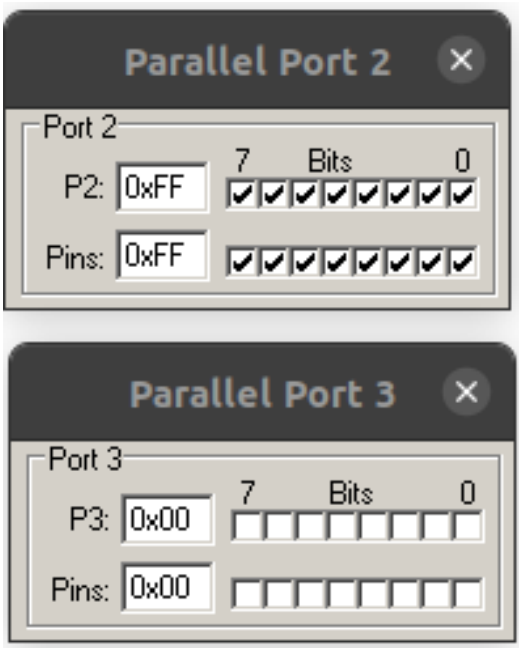
1.1 Specyfikacja zadania:

a. Zwróć uwagę na typ portu. Który z portów można skonfigurować jako wejścia lub wyjścia cyfrowe?
b. Sprawdzić kiedy dany bit port jest ustawiony jako wejście lub wyjście cyfrowe.

1.2 Prezentacja implementacji rozwiązania:

Port 1
Port 1 is also an 8-bit port directly controlled via the P1 SFR.
Port 1 digital output capability is not supported on this device.
Port 1 pins can be configured as digital inputs or analog inputs.

Rysunek 1: Wycinek dokumentacji



Rysunek 2: Stan portów

```
1000 #include <aduc831.h>

1002 int main(void)
{
1004   P2 = 0xFF; //wejscie
       P3 = 0x00; //wyjscie
1006   while(1)
       {
```

```
1008
    }
1010    return 0;
}
```

1.3 Prezentacja wyników testów oraz wnioski:

Zgodnie z informacjami z dokumentacji aduc831 wszystkie potry od P0 do P3 posiadają możliwość konfiguracji jako wejścia lub wyjścia oprucz portu P1, który może pracować tylko jako wejście zfodnie z informacja z rysunku numer 1.

Na 2 zostały przedstawione porty jako wejsca(górna część rysunku) oraz wyjście z listingu powyższego.

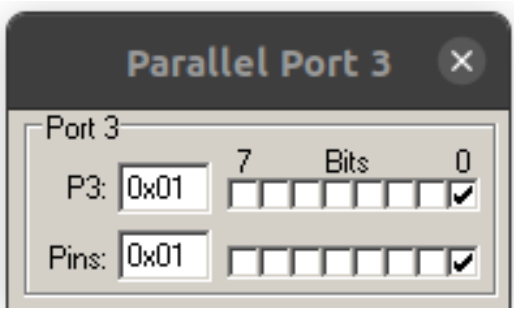
2 Programowa obsługa pojedynczego wejścia i wyjścia mikrokontrolera w symulatorze:

2.1 Zmieniaj cyklicznie wartości binarną pojedynczego wyjścia cyfrowego mikrokontrolera. Wykorzystaj operatory binarne. Sprawdź działanie programu w symulatorze.

2.1.1 Prezentacja implementacji rozwiązania:

```
1000 #include <aduc831.h>

1002 int main(void)
{
1004   P3 = 0x00; //outputs
    while(1)
1006   {
      P3 ^= 0x01;
1008   }
    return 0;
1010 }
```

2.1.2 Prezentacja wyników testów:



Rysunek 3: Wycinek dokumentacji

2.1.3 Wnioski:

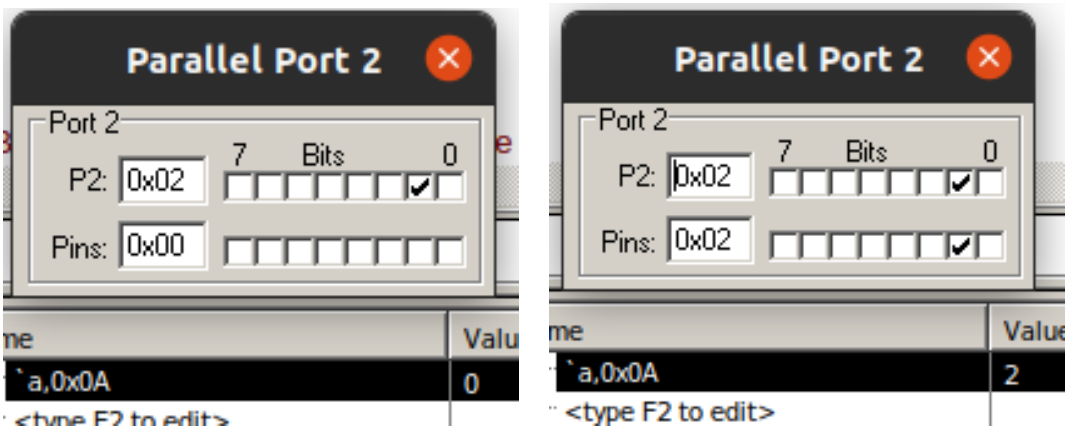
Na rysunku nr. 3 widzimy, że program z powyższego listingu powoduje zmianę stanu pinu 0 w porcie P3.

2.2 Skonfiguruj pojedynczą linie mikrokontrolera jako wejście cyfrowe. Sprawdzić czy ustawiony jest wybrany bit w bajcie. Zmień wartość zmiennej globalnej w zależności od stanu wejścia cyfrowego. Wykorzystaj operatory binarne. Sprawdź działanie programu w symulatorze.

2.2.1 Prezentacja implementacji rozwiązania:

```
1000 #include <aduc831.h>

1002 main(void)
{ unsigned int a = 0;
1004   P2 = 0x00; //wyjscie
   P2 |= (0x01 << 1); //wejscie
1006
   while(1)
1008   {
   if(P2&(0x01<<1))
1010     a = 1;
   else
1012     a = 0;
   }
1014   return 0;
}
```

2.2.2 Prezentacja wyników testów:



Rysunek 4: Left: Wartość 0.
Right: Wartość 2

2.2.3 Wnioski:

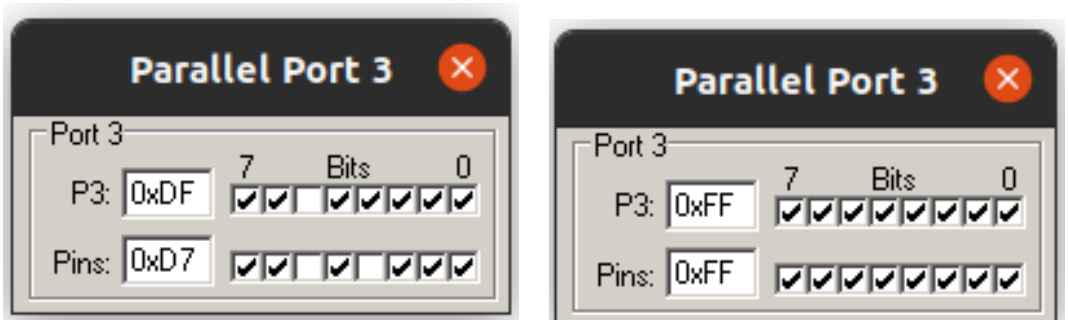
Powyższy listing przedstawia ustawienie Portu P2 jako wyjscie, pinu 1 na tym porcie jako wejsci. W pętli nieskączanej kiedy warunek zostanie spełniony, czyli na wejściu jest logiczna jedynka to zmienna zmienia wartość na 2(choc powinno być 1 według programu i nie wiem co jest przyczyną takiego stanu rzeczy) w przeciwnym wypadku wartość a wynosi 0.

2.3 Przygotuj trzy makrodefinicje: BitUstaw(bajt, nr bitu), BitKasuj(bajt, nr bitu) oraz BitSprawdz(bajt, nr bitu). Odpowiadające odpowiednio za: ustawienie wybranego bitu w bajcie, skanowanie wybranego bitu w bajcie oraz sprawdzenie wybranego bitu w bajcie. Wykorzystaj operatory binarne. Sprawdź działanie programu w symulatorze.

2.3.1 Prezentacja implementacji rozwiązania:

```
1000 #include <aduc831.h>

1002 #define BitUstaw(b, nr) b=b|(0x01<<nr)
     #define BitKasuj(b, nr) b=b&(~(0x01<<nr))
1004 #define BitSprawdz(b, nr) (b&(0x01<<nr))

1006 int main(void)
     {
1008
       while(1)
1010   {
        if(!BitSprawdz(P3,3))
1012      {
          BitKasuj(P3,5); //skasuj bit 5 z portu 3
1014       }
          else
1016      {
          BitUstaw(P3,5); // ustawia bit 3 na 1 reszta bez zmian
1018      }

1020   return 0;
     }
1022 }
```

2.3.2 Prezentacja wyników testów:



Rysunek 5: Left: Po zaniku sygnału P3.3. Right: Przed zanikiem sygnału P3.3

2.3.3 Wnioski:

Program przedstawia ustawienie często używanych funkcji jako definicji oraz wykorzystanie ich w pętli głównej. Jak przedstawiono na rysunku 5, kiedy zaniknie sygnał na wejściu P3.3 to kasuje się bit na P3.5. Gdy Sygnał się pojawi na P3.3 to również pojawi się na P3.5.

3 Programowa obsługa pojedynczego wejścia i wyjścia mikrokontrolera zestawu edukacyjnego:

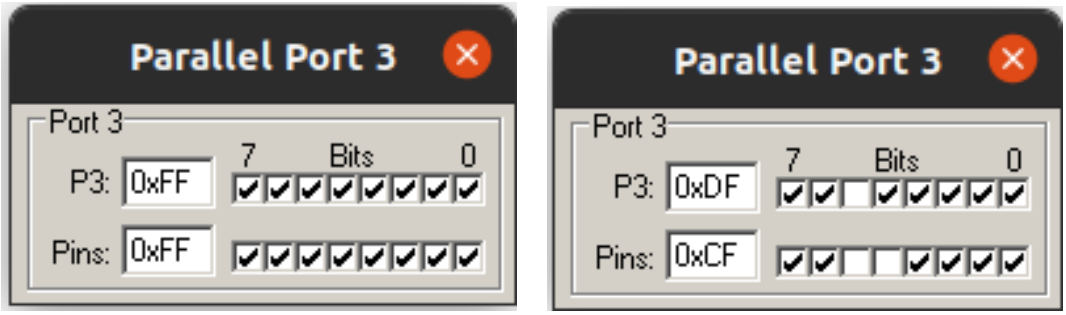
3.1 Stan dowolnej diody z Rys. 1 ma odpowiadać stanowi przycisku PR1. Jeśli przycisk jest wciśnięty to dioda jest zapalona, w innym przypadku dioda jest wyłączona. Wykorzystaj napisane makrodefinicje. Sprawdź działanie programu w symulatorze.

3.1.1 Prezentacja implementacji rozwiązania:

```
1000 #include <aduc831.h>
     #define ustaw(bajt,nr_bitu) (bajt|=(1<<nr_bitu))
1002 #define skasuj(bajt,nr_bitu) (bajt&=~(1<<nr_bitu))
     #define sprawdz(bajt,nr_bitu) (bajt&(1<<nr_bitu))
1004
     #define D1off ustaw(P3,5)
1006 #define D1on skasuj(P3,5)

1008 #define PR1 (!sprawdz(P3,4))

1010 int x=0;
     int main()
1012 {
       while(1)
1014   {
         if(PR1)
1016       x=1;
         if(!PR1)
1018       x=0;
         if(x==1)
1020       D1on;
```

```
    else
1022      D1off;
  }
1024  return 0;
}
```

3.1.2 Prezentacja wyników testów:



Rysunek 6: Left: Dioda zapalona. Right: Dioda nieaktywna.

3.1.3 Wnioski:

Linie 1-8 Definiowanie makr. W pętli while Jeżeli warunek PR1 rowny True to x = 1 a wtedy Dioda zapalona, w przeciwnym wypadku nieaktywna. Przedstawiono to na Rysunku 6.

3.2 Przygotuj układ do programowania wykonując następujące czynności:

Brak układu

4 Prosty układ kombinacyjny:

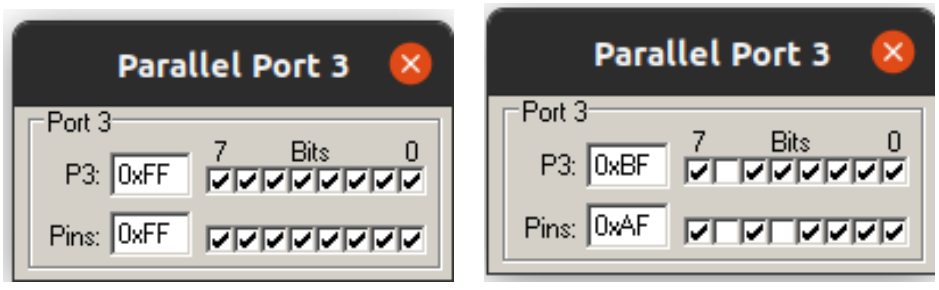
4.1 Złożony program wykorzystujący wiele wejść i wiele wyjść cyfrowych.

4.1.1 Wykonaj zadane operacje logiczne na bitach (np. d=(ab|c), gdzie a,b,c,... są pojedynczymi bitami portu). Sprawdź działanie programu w symulatorze.

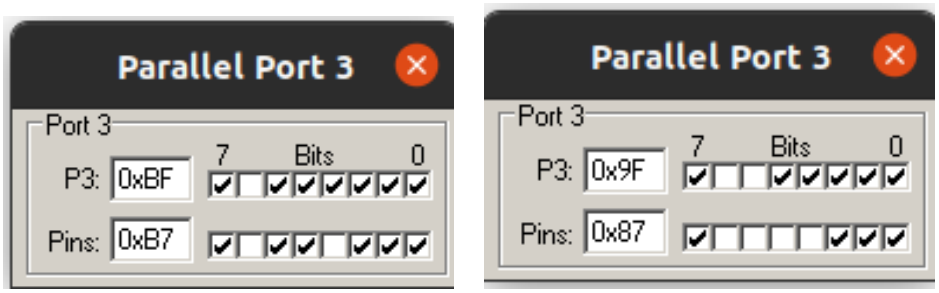
4.1.2 Prezentacja implementacji rozwiązania:

```
1000 #include <aduc831.h>

1002 #define BitUstaw(b, nr) b=b|(0x01<<nr)
     #define BitKasuj(b, nr) b=b&(~(0x01<<nr))
1004 #define BitSprawdz(b, nr) (b&(0x01<<nr))

1006 int main(void)
     {
1008
       while(1)
1010   {
        if ((!BitSprawdz(P3,3))&(!BitSprawdz(P3,4)))
1012
          {
1014      BitKasuj(P3,5);
          }
1016      else
          {
1018      BitUstaw(P3,5);
          }
1020

1022     if ((!BitSprawdz(P3,3))|(!BitSprawdz(P3,4)))

1024      {
          BitKasuj(P3,6);
1026      }
          else
1028      {
          BitUstaw(P3,6);
1030      }

1032
       return 0;
1034 }
     }
```

4.1.3 Prezentacja wyników testów:



Rysunek 7: Left: Warunki niespełnione. Right: Spełniony warunek |.



Rysunek 8: Left: Spełniony warunek |. Right: Spełniony warunek .

4.1.4 Wnioski:

Program w przedziale linii 11-19 kasuje P3.5 gdy spełniony jest warunek, że P3.3 i P3.4 sa jednocześnie rowne 0, kiedy tak nie jest to ustawia P3.5. Przedstawia tą sytuację rysunek 8 prawy. W przypadku kodu z lini 22-30 wystarczy, że tylko jedna część warunku zostanie spełniona aby P3.6 został skasowany lub ustawiony.

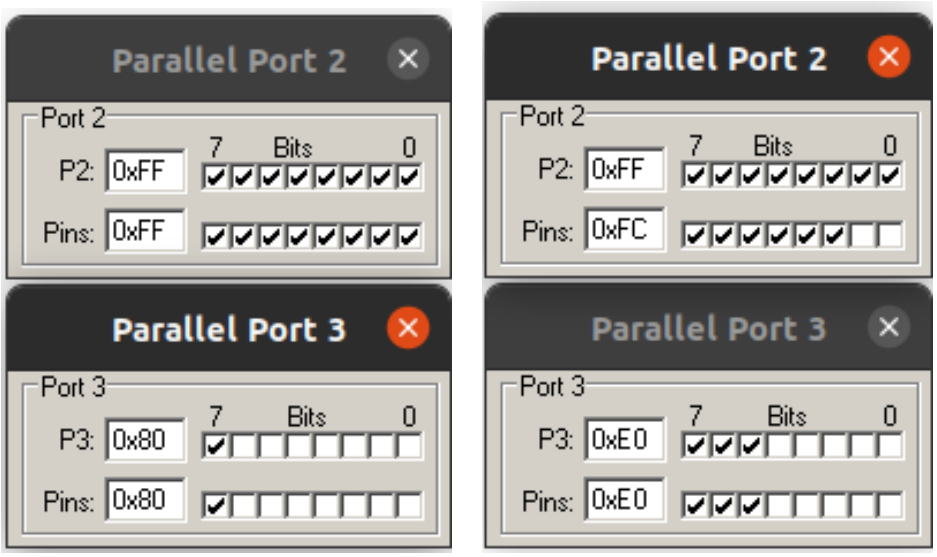
4.1.5 Wykorzystaj diody z Rys. 1 oraz przycisk PR1 oraz PR2. Napisz makroinstrukcje: ustawiające oraz kasujące stan poszczególnych diod, sprawdzające stan poszczególnych przycisków. Zaproponuj dla wszystkich czterech kombinacji stanu przycisków różne stany diod.

4.1.6 Prezentacja implementacji rozwiązania:

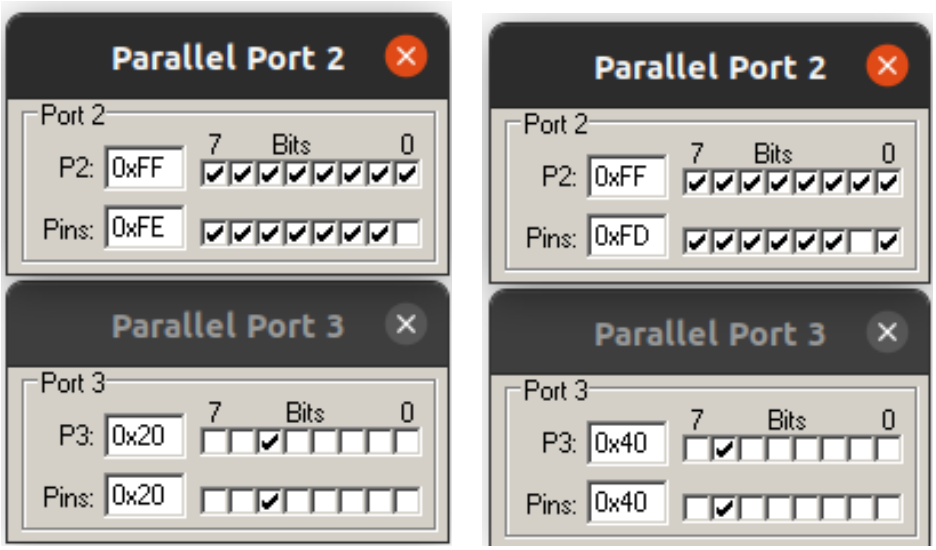
```
1000 #include <aduc831.h>

1002 #define ustaw(bajt,nr_bitu) (bajt|=(1<<nr_bitu))
     #define skasuj(bajt,nr_bitu) (bajt&=~(1<<nr_bitu))
1004 #define sprawdz(bajt,nr_bitu) ((bajt&(1<<nr_bitu))&&1)


1006
     #define D1on ustaw(P3,5)//ustaw diode P3.5
1008 #define D1off skasuj(P3,5)//skasuj diode P3.5

1010 #define D2on ustaw(P3,6)
     #define D2off skasuj(P3,6)
1012
     #define D3on ustaw(P3,7)
1014 #define D3off skasuj(P3,7)

1016 #define PR1 (sprawdz(P2,0))
     #define PR2 (sprawdz(P2,1))
1018
     void opoznienie()
1020 {
       int x,y;
1022   for (x=0;x<100;x++)
         y = 0;
1024 }

1026 int main() //funkcja glowna programu
```

```
{    P2 = 0xFF;
1028   P3 = 0x00;

1030   while(1)
       {
1032
       //Stan 1
1034   if (!(PR1)&(PR2))
       {
1036     D1off;
         D2off;
1038     D3off;
         D1on;
1040     opoznienie();
       }
1042   //Stan 2
       else if((PR1)&!(PR2))
1044   {
         D1off;
1046     D2off;
         D3off;
1048     D2on;
         opoznienie();
1050   }

1052   //Stan 3
       else if((PR1)&(PR2))
1054   {
         D1off;
1056     D2off;
         D3off;
1058     D3on;
         opoznienie();
1060   }

1062   //Stan 4
       else if(!(PR1)&!(PR2))
1064   {
         D1off;
1066     D2off;
         D3off;
1068     D1on;
         D2on;
1070     D3on;
         opoznienie();
1072   }


1074
       }
1076   return 0;
}
```

4.1.7 Prezentacja wyników testów:



Rysunek 9: Left: Obydwa przyciski załączone. Right: Obydwa przyciski rozłączone



Rysunek 10: Left: PR1 rozłączony, PR2 załączony. Right: PR2 rozłączony, PR1 załączony

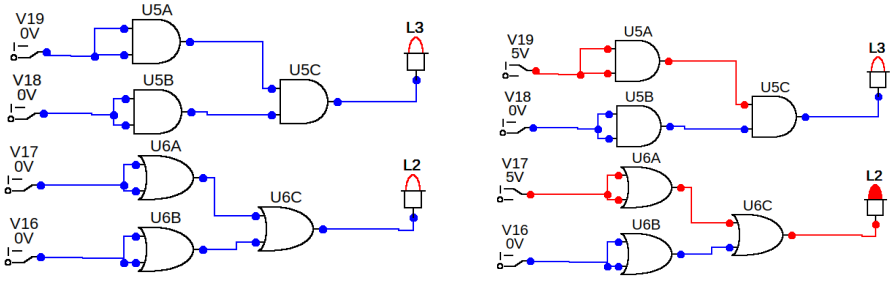
4.1.8 Wnioski:

Linie 7-17 makrodefinicje okreslajace diody oraz przyciski na bazie makr z poczatku programu 2-4. Linie 19-24 funkcja opóźniająca zastosowana do podtrzymania diody podczas jej świecenia. W pętli while mamy przedstawione różne stany trzech naszych diod w zależnośći od kombinacji załączonych przycisków. Stan1 PR1 nie załączony, PR2 załączony, Diody wszystkie wyłączone następnie załączenie diody 1 i podtrzymanie jej załączenia. Ten sam mechanizm odnosi się do kolejnych instrukcjii warunkowych else if.

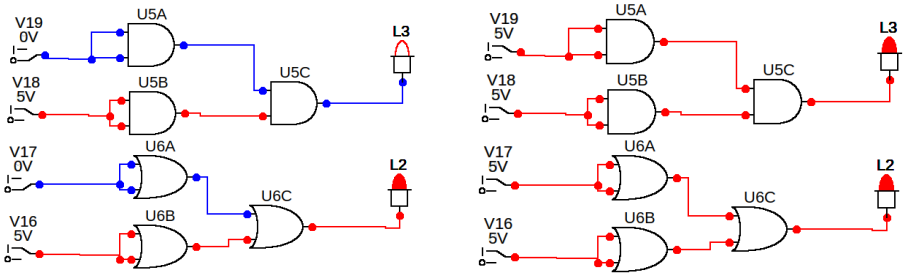
4.2 Realizacja sprzętowa.

4.2.1 Zrealizuj za pomocą bramek logicznych zadanie logiczne przedstawione w punkcie 4.a. Wykorzystaj logiczne przełączniki oraz elementy wyświetlające stan logiczny wyjść. Przetestuj działanie układu w symulatorze.

4.2.2 Prezentacja implementacji rozwiązania oraz wyników testów:



Rysunek 11: Left: Obydwa przyciski załączone. Right: Obydwa przyciski rozłączone



Rysunek 12: Left: PR1 rozłączony, PR2 załączony. Right: PR2 rozłączony, PR1 załączony

4.2.3 Wnioski:

Rysunek 11 oraz 12 przedstawiają układy złożone z bramek AND (górny) oraz OR (dolny). Składają się również z lampek L3, L2 oraz przełączników nadających stan logiczny na wejściu kazdej bramki (1 lub 0). Kolejno Na rysunku 11 w części lewej widać na obu układach lampki wyłączone, ponieważ sygnały wchodzace do bramek są równe zero. W rysunku po prawej stronie górna część przedstawiająca układ z bramek AND dostaje tylko sygnał 1 na jedną bramkę co według algebry Bool'a jest niewystarczające aby lampka L3 została załączona. Dolny rysunek wykorzystuję bramki OR, więc niezależnie która bramka ma sygnał na wejściu 1, lampka L2 zapali się. Tak samo będzie na rysunku 12 prawym dolnym i lewym dolnym oraz lewym górnym. W przypadku prawego górnego zaszedł warunek logiczny 11=1 co spowodowało zapalenie się lampki L2.

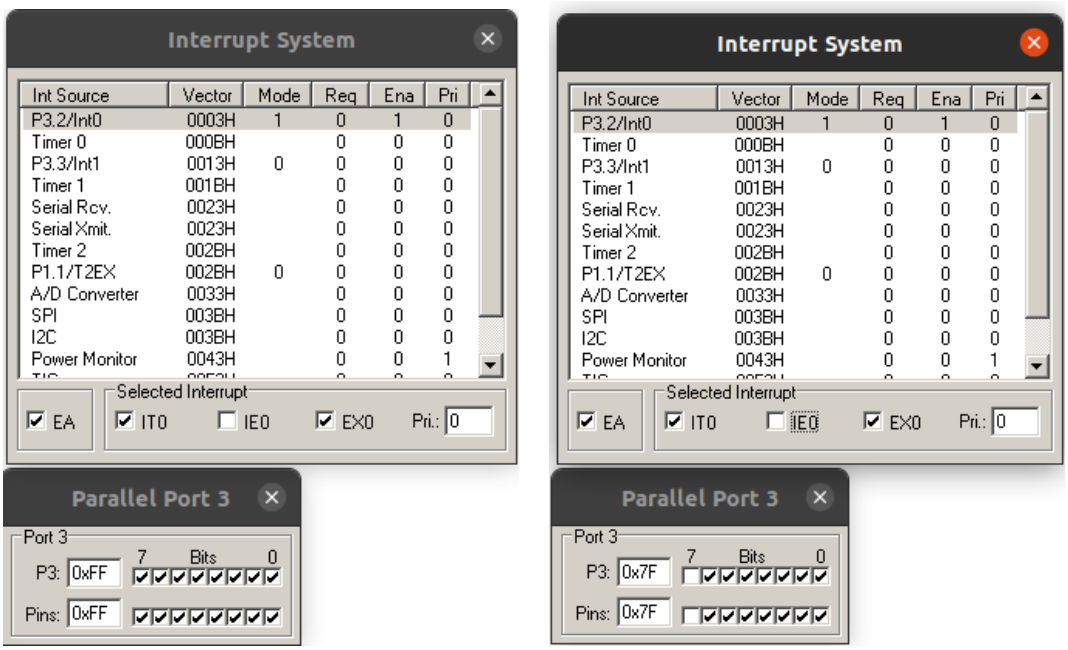
5 Obsługa przerwań mikrokontrolera:

5.1 Napisz szkielet funkcji obsługującej przerwanie od zdarzenia zewnętrznego – wejście cyfrowe przycisku. Pamiętaj o prawidłowym numerze przerwania. Prezentacja implementacji rozwiązania. Zmień stan wybranej diody na przeciwny w funkcji obsługującej przerwanie. Sprawdź działanie programu w symulatorze, następnie sprawdź działanie na zestawie edukacyjnym:

5.1.1 Prezentacja implementacji rozwiązania:

```
1000 #include <aduc831.h>

1002 #define ustaw(bajt,nr_bitu) (bajt|=(1<<nr_bitu))
     #define skasuj(bajt,nr_bitu) (bajt&=~(1<<nr_bitu))
1004 #define sprawdz(bajt,nr_bitu) ((bajt&(1<<nr_bitu))&&1)

1006 #define D1on ustaw(P3,7)//ustaw diode P3.5
     #define D1off skasuj(P3,7)//skasuj diode P3.5
1008
     #define PR1 (sprawdz(P3,3))
1010


1012 void przerwanie1() interrupt 0
     {
1014   D1off;


1016
     }
1018 int main() //funkcja glowna programu
     {
1020
       IT0 = 1;    // Configure interrupt 0 for falling edge on /INT0 (P3.2)
1022   EX0 = 1;    // Enable EX0 Interrupt
       EA = 1;     // Enable Global Interrupt Flag
1024   while(1)
       {
1026
       }
1028   return 0;
     }
```

5.1.2 Prezentacja wyników testów:



Rysunek 13: Left: Przed nastąpieniem przerwania. Right: Po nastąpieniu przerwania

5.1.3 Wnioski:

Kod z ostatniego listingu przedstawia szkielet przerwania od P3.2. Kiedy nastąpi przerwanie zasymulowane jako zaznaczenie IE0 na rysunku 13 prawym to wtedy dioda symulowana na porcie P3.7 zgaśnie.