Politechnika Poznańska

Instytut Automatyki, Robotyki i Inżynierii Informatycznej

Zakład Sterowania i Elektroniki Przemysłowej

Wprowadzenie do: zintegrowanego środowiska programistycznego języka C dla mikrokontrolerów oraz symulatora układów logicznych

Technika cyfrowa i mikroprocesorowa

Materiały do zajęć laboratoryjnych

dr inż. Dominik Łuczak

Dominik.Luczak@put.poznan.pl

I. CEL

WIEDZY

Celem zajęć jest zapoznanie z budową oraz terminologią zintegrowanego środowiska programistycznego dla systemów mikroprocesorowych oraz symulatora układów logicznych.

UMIEJĘTNOŚCI

Celem zajęć jest nabycie umiejętności obsługi:

1) zintegrowanego środowiska programistycznego dla systemów mikroprocesorowych:

- utworzenie szablonu projektu,
- konfiguracja ustawień projektu oraz programatora,
- prosta weryfikacja prawidłowego działania kodu,
- złożona weryfikacja prawidłowego działania kodu,

2) oraz symulatora układów logicznych:
- utworzenie prostego układu logicznego,
- analiza prostego układu logicznego.

KOMPETENCJI SPOŁECZNYCH

Celem zajęć jest kształtowanie właściwych postaw:

- tworzenie kodu programu w sposób czytelny,
- stosowanie stosownych komentarzy,
- weryfikacja poprawności działania programu oraz układu logicznego.

II. POLECENIA KOŃCOWE

Powtórz kilkukrotnie w domu zadania zrealizowane w trakcie zajęć, aż nabędziesz wprawę w tworzeniu oraz weryfikowaniu programu. Przygotuj szablon projektu, który będziesz używać na kolejnych zajęciach.

III. SCENARIUSZ DO ZAJĘĆ

a) ŚRODKI DYDAKTYCZNE

Sprzętowe:
- komputer.

Programowe:
- zintegrowane środowisko programistyczne języka C dla mikrokontrolerów (Keil uVision),
- symulatora układów logicznych (Circuit Maker).

b) PRZEBIEG ZAJĘĆ

1. Przygotuj schemat układu logicznego i wykonaj testy dla poszczególnych bramek logicznych:
    a. NOT,
    b. OR,
    c. NOR,

   d. AND,
   e. NAND,
   f. XOR,
   g. XNOR.
2. Przygotuj projekt w zintegrowanym środowisku programistycznym języka C dla mikrokontrolerów:
   a. utwórz pusty projekt,
   b. utwórz plik main.c – przeznaczony na program główny,
   c. zmień ustawienia projektu,
   d. przebuduj projekt.
3. Obsługa odpluskwiacza.
   a. Uruchom i przeanalizuj działanie symulatora wejść/wyjść cyfrowych mikrokontrolera.
   b. Uruchom i przeanalizuj działanie dynamicznego podglądu zmiennych.
   c. Uruchom i przeanalizuj działanie odpluskwiacza w trybie ciągłym i krokowym.
   d. Uruchom i przeanalizuj działanie opcji zaawansowanych.
4. Napisz prosty program pozwalający na wykonanie wybranych operacji matematycznych.
   a. napisz w kodzie programu odpowiednie funkcję,
   b. zastosuj komentarze zgodne z składnią generatora dokumentacji,
   c. przetestuj działanie programu,
   d. napisz makroinstrukcję odpowiadające napisanym funkcjom,
   e. przetestuj działanie programu,
   f. przećwicz obsługę symulatora.

IV. PRZYGOTOWANIE DO ZAJĘĆ

a) ZAPOZNANIE Z PRZEPISAMI BHP

Wszystkie informacje dotyczące instrukcji BHP laboratorium są zamieszczone w sali laboratoryjnej oraz u prowadzącego zajęcia. Wszystkie nieścisłości należy wyjaśnić z prowadzącym laboratorium. Wymagane jest zaznajomienie i zastosowanie do regulaminu.

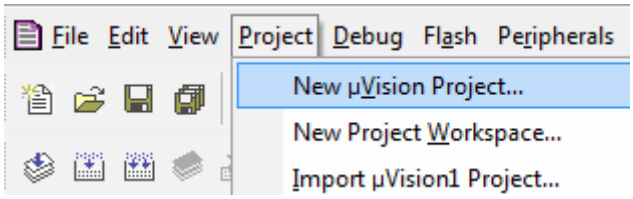
Na zajęcia należy przyjść przygotowanym zgodnie z tematem zajęć. Obowiązuje również materiał ze wszystkich odbytych zajęć.

b) WPROWADZENIE DO ZINTEGROWANEGO ŚRODOWISKA PROGRAMISTYCZNEGO

Keil µVision jest to zintegrowane środowisko programistyczne stworzone dla architektury mikrokontrolerów z rodziny 51 dla profesjonalnych aplikacji [1]. Środowisko wyposażone jest w kompilator języka C oraz odpluskwiacz (ang. debugger). Program w wersji demo można bezpłatnie pobrać ze strony producenta [2]. W tym celu z menu po lewej stronie należy wybrać **Evaluation Software->C51 Evaluation Software** i wypełnić ankietę. Po zatwierdzeniu ankiety zostanie udostępniony link.

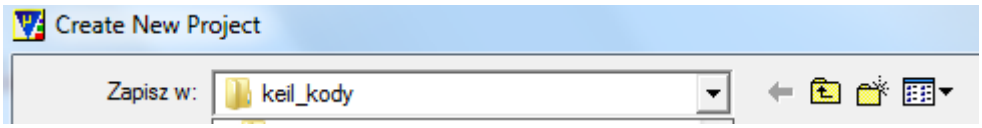
c) TWORZENIE NOWEGO PROJEKTU

Uruchom aplikację *Keil uVision* umieszczoną domyślnie na pulpicie komputera. W nowo otwartym oknie kliknij na **Project -> New µVision Project…** (Rys. 1)



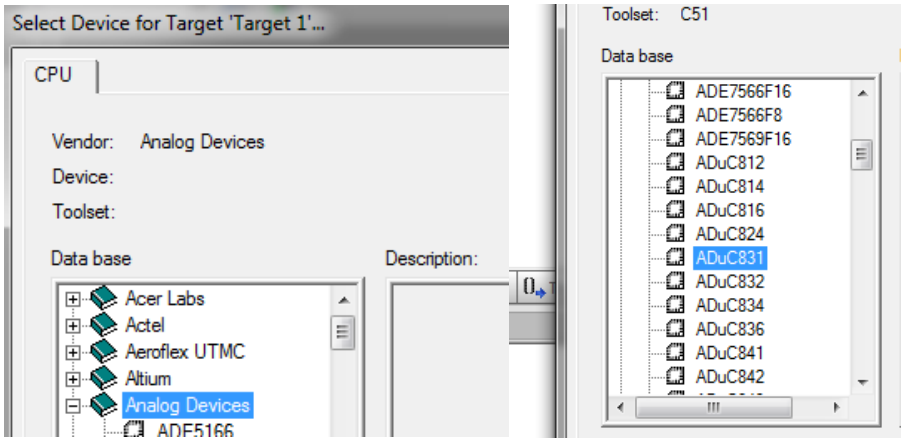
Rys. 1 Tworzenie nowego projektu.

W nowo otwartym oknie wybierz miejsce zapisu nowego projektu oraz podaj nazwę (Rys. 2).



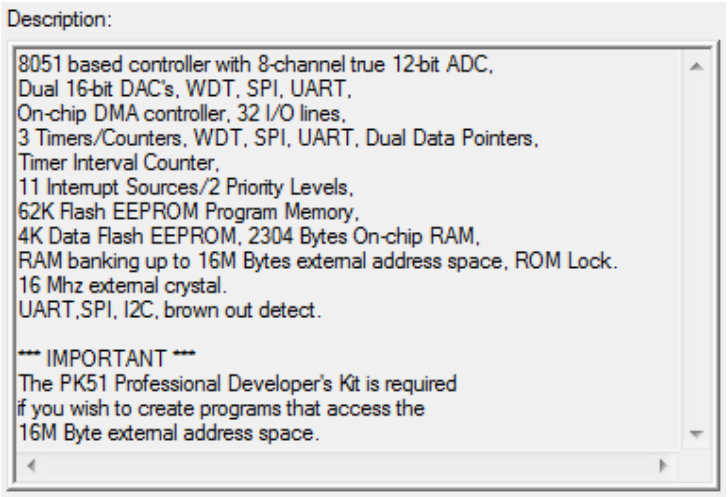
Rys. 2 Wybór lokalizacji nowego projektu .

W kolejnym oknie wybierz urządzenie (mikrokontroler) dla którego jest tworzony program. Na potrzeby zajęć wykorzystywany jest zestaw edukacyjny z mikrokontrolerem **ADuC831** [3, 4]**.** Wybierz firmę **Analog Devices** następnie mikrokontroler i zatwierdź wybór (Rys. 3).



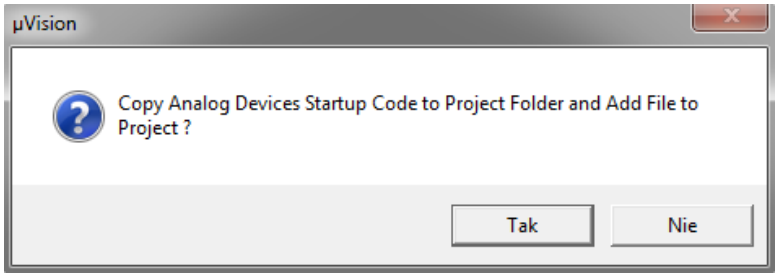
Rys. 3 Wybór urządzenia.

Warto zauważyć iż z boku okna podczas wyboru urządzenia jest wypisana specyfikacja danego mikrokontrolera (Rys. 4)



Rys. 4 Specyfikacja mikrokontrolera.

Po potwierdzeniu wyboru pojawia się okno kopiowania kodu startowego firmy Analog Devices. Wybierz opcję NIE (Rys. 5).



Rys. 5 Zapytanie o kopiowanie kodu startowego, wybieramy opcję NIE.

Właśnie ukończyłeś tworzenie nowego projektu. Projekt jest pusty i nie posiad żadnego pliku z kodem źródłowym. W celu dodania pliku z kodem w języku C do projektu należy go uprzednio utworzyć.

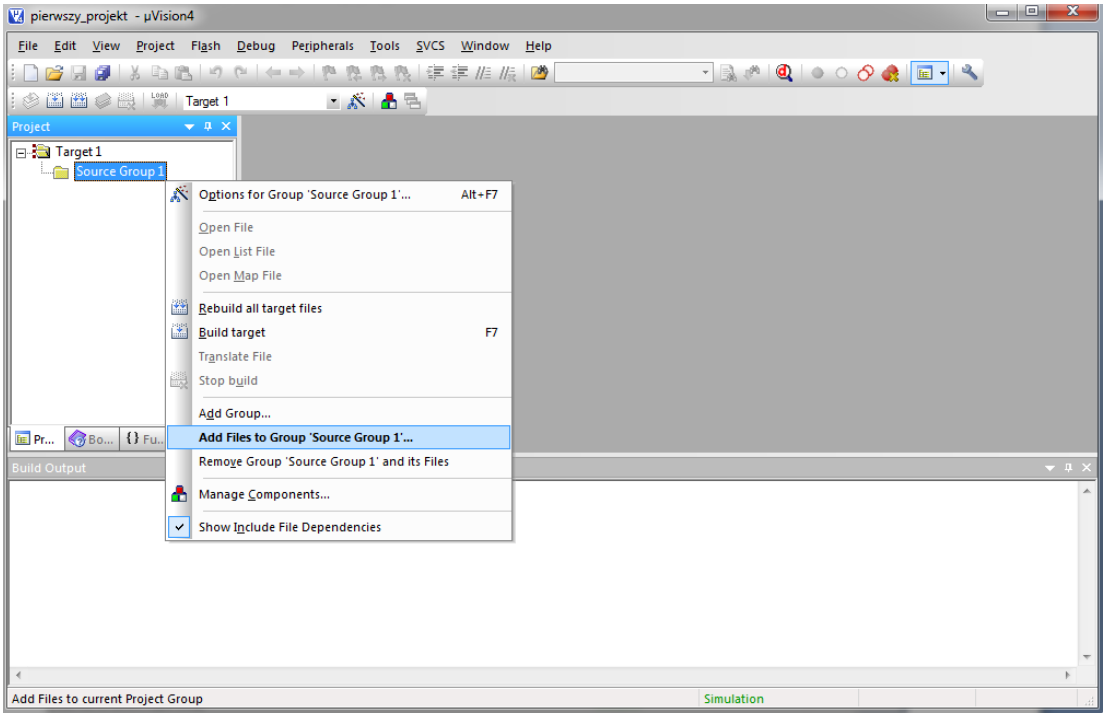
i. TWORZENIE PLIKU *.C

W celu stworzenia pliku *.c wybieramy kolejno:

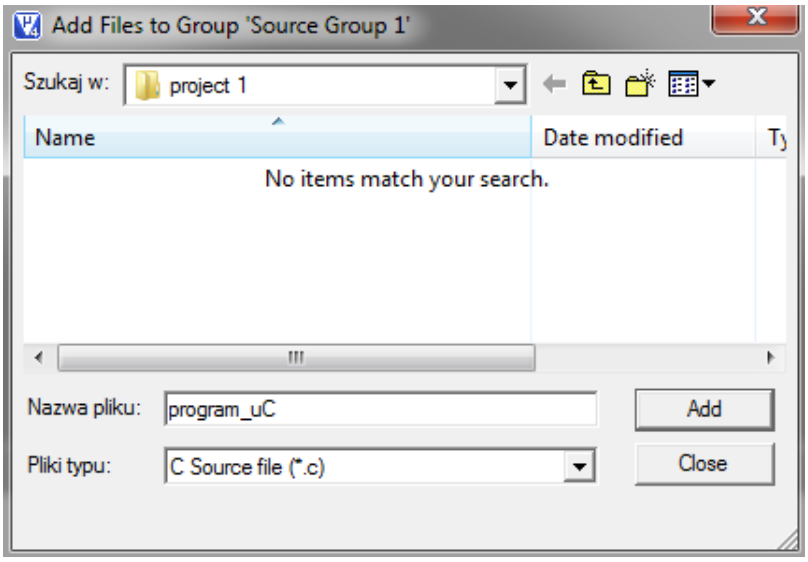
File->New(Ctrl+n)

File->Save(Ctrl+s)->Pulpit->main.c->Zapisz

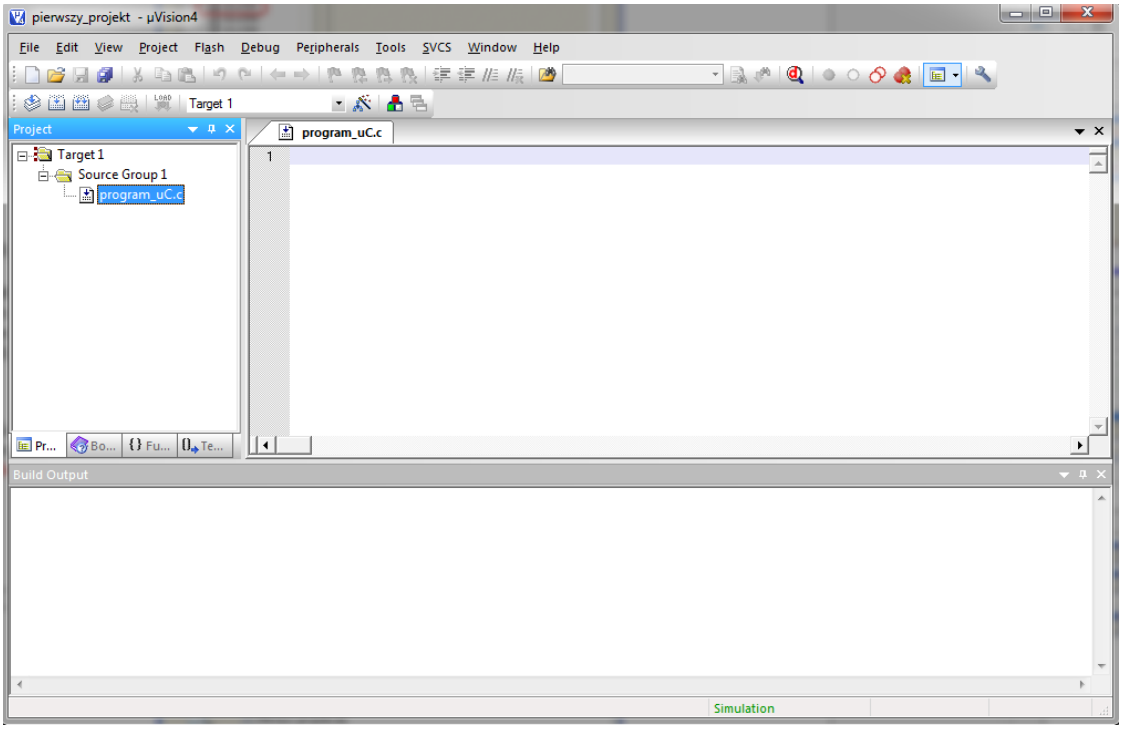
Należy zwrócić uwagę aby w nazwie pliku nie występowały znaki **‘spacji’** - wystarczy użyć znaku podkreślenia. Również w tym punkcie trzeba pamiętać o podaniu rozszerzenia **.c**. Nowy plik nie jest jeszcze powiązany z projektem, należy go dodać do projektu. W tym celu kliknij LPM (lewy przycisk myszy) w bocznym oknie projektu na **Target1**, następnie PPM (prawy przycisk myszy) na **Source Group 1** i z rozwiniętego menu wybierz opcję **Add Files to ‘Group 1’** (Rys. 6). W nowo otwartym oknie wybierz nazwę utworzonego pliku - Rys. 7, następnie kliknij LPM **Add**. Plik został dodany do projektu, jeśli nie masz więcej plików do dodania kliknij LPM **Close**. W drzewie projektu widoczny jest plik z rozszerzeniem .c gotów do pisania kodu programu. Wybieramy plik z listy klikając dwukrotnie LPM. Plik zostaje otwarty do edycji w oknie głównym (Rys. 8).



Rys. 6 Dodawanie nowego pliku do projektu.



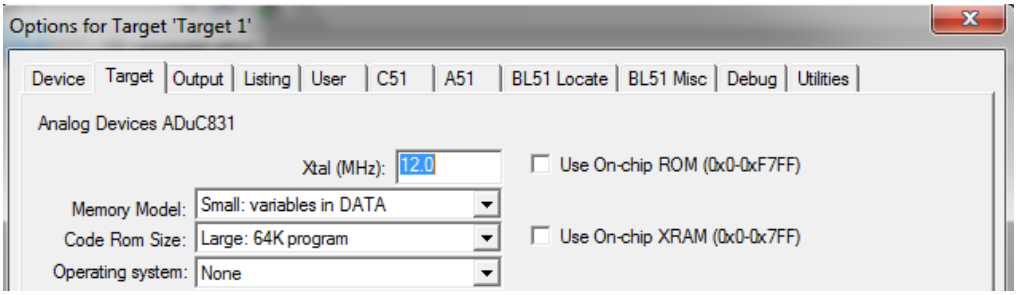
Rys. 7 Wybór nazwy pliku.



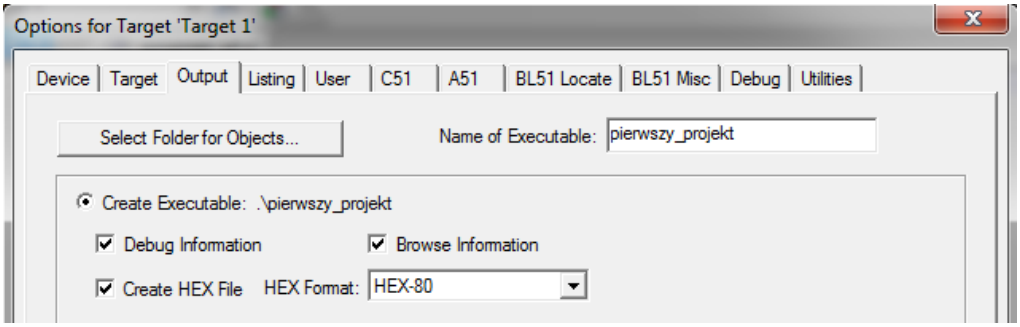
Rys. 8 Otwarcie stworzonego pliku.

ii. ZMIANA USTAWIEŃ PROJEKTU

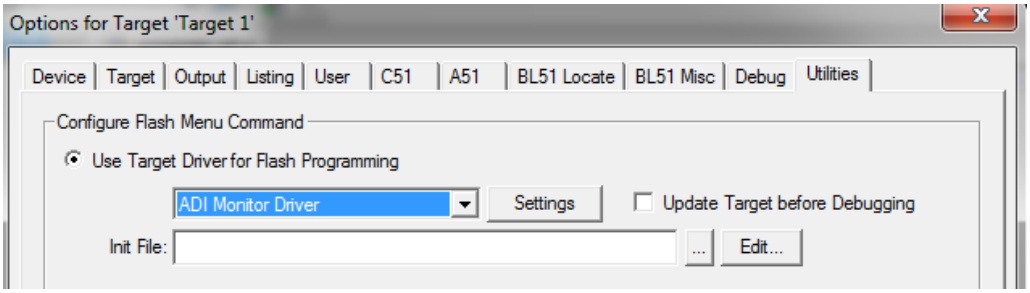
Każdy system mikroprocesorowy, dla którego pisać będzie program może posiadać różne parametry i opcje. Przed przystąpieniem do pisania kodu należy uprzednio zmienić ustawienia klikając PPM na **Target 1** i wybierając **Options for Target 'Target 1'.** Na potrzeby pierwszych zajęć ustaw wartość kwarcu (**Xtal**) na 12 MHz (Rys. 9) i przejdź do zakładki **Output**. W **Output** zaznacz tworzenie pliku Hex wybierając **Create HEX File** (Rys. 10). Później przejdź do zakładki **Utilities** (Rys. 11). W **Utilities** z działu **Configue Flash Menu Command** zaznacz **Use Target for Flash Programming**, następnie wybierz **ADI Monitor Driver**.



Rys. 9 Zmiana ustawień kwarcu.



Rys. 10 Tworzenie pliku typu Hex.



Rys. 11 Wybór ADI Monitor Driver w Utilities.

Po ustawieniu powyższych opcji zatwierdź zmiany klikając LPM na przycisk OK. Projekt został skonfigurowany i możesz przejść do pisania kodu dla mikrokontrolera.

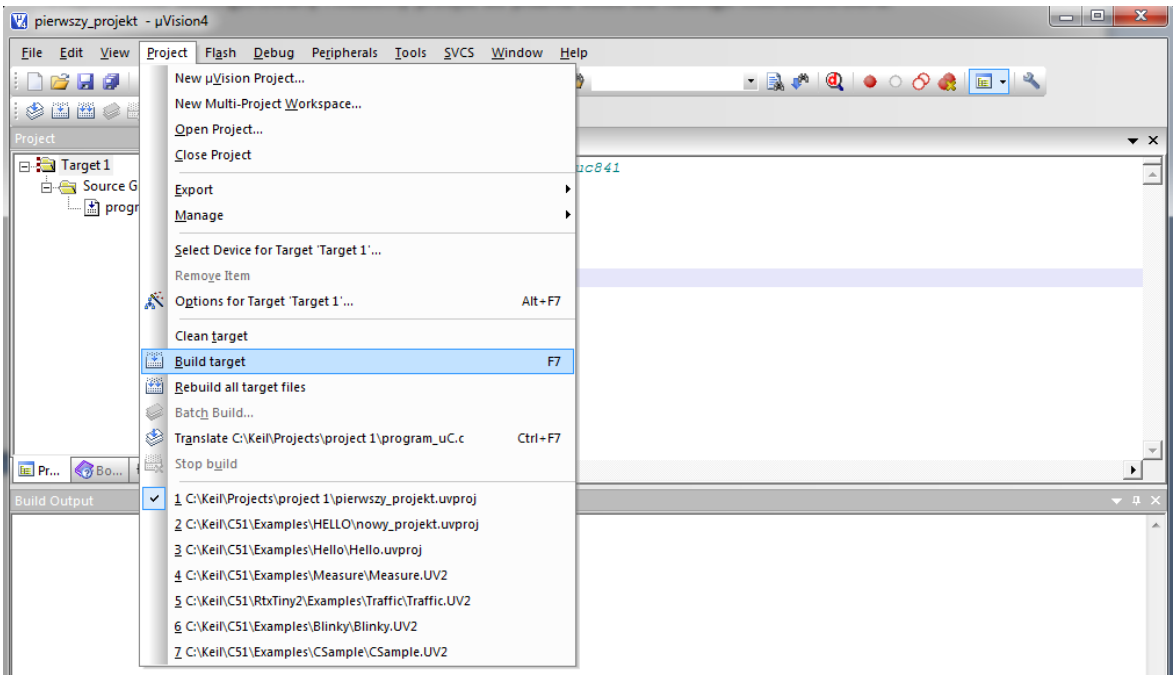
iii. PIERWSZY KOD PROGRAMU

Wybierz z drzewa projektu plik z rozszerzeniem .c do edycji. Wpisz kod źródłowy z Listing 1.

Listing 1 Przykładowy program

```
#include <aduc831.h> //plik nagłówkowy
int main(void)
{
P3=0x0F;
return 0;
}
```

Program ma za zadanie ustawić na porcie P3 cztery najmłodsze bity na ‘1’ a cztery najstarsze na ‘0’. Nazwa P3 jest zdefiniowana w pliku nagłówkowym <ADuC831.h>. W celu kompilacji i konsolidacji programu wybierz z menu **Project->Build target**. Pierwszy program utworzony - Rys. 12. Pierwszy program został stworzony. W celu weryfikacji działania programu należy skorzystać z odpluskwiacza.



Rys. 12 Utworzony pierwszy program.

d) OBSŁUGA ODPLUSKWIACZA

Odpluskwiacz (ang. Debug tool, Debugger, czytaj dibager) – program komputerowy służący do dynamicznej analizy innych programów, w celu odnalezienia i identyfikacji zawartych w nich błędów, zwanych z angielskiego bugami (robakami). Proces nadzorowania wykonania programu za pomocą odpluskwiacza (debuggera) określa się mianem odpluskwiania (debugowania).

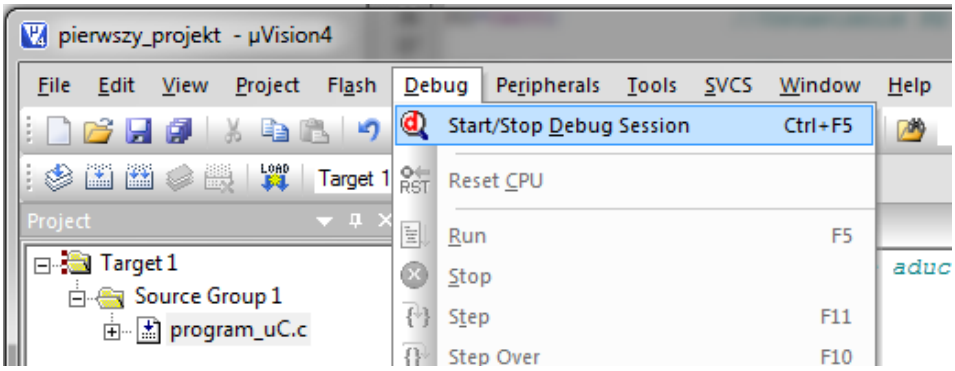
W celu przetestowania wybranych funkcji wbudowanych w odpluskiwacz programu Keil µVision wykorzystaj kod źródłowy z Listing 2.

Listing 2 Program do analizy

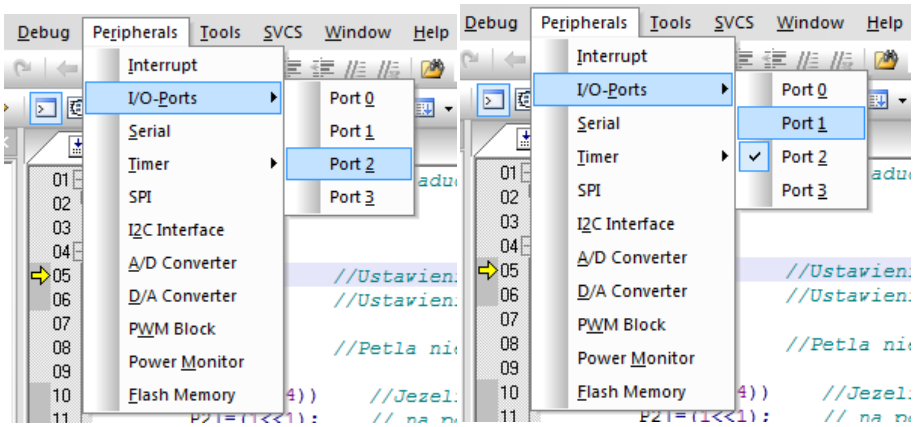
```
#include<aduc831.h>
int main(void)
{
P0=0xFF;  //Ustawia P0 jako wejścia
P2=0x00;  //Ustawia P2 jako wyjścia
while(1)   //Pętla główna programu
          {
          if(P0&(0x01<<0))       //Jeżeli wejście cyfrowe nr 0 z P0 ma wartość bitową równą 1
            P2|=(0x01<<0);              //Ustaw bit 0 z portu P2
          else
            P2&=~(0x01<<0);     //Skasuj bit 0 z portu P2
          }
return 0;
}
```

i. WEJŚCIA/WYJŚCIA MIKROKONTROLERA

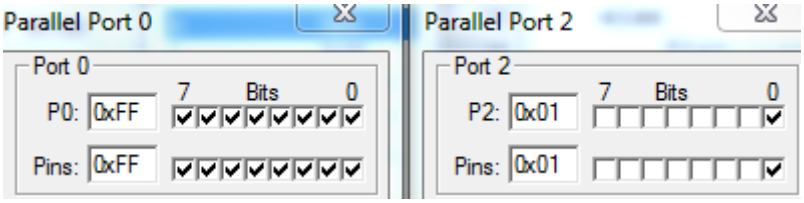
Uruchom odpluskiwacz wybierając z menu **Debug** pozycję **Start/Stop Debuging Session -** Rys. 13. Pojawi się komunikat o używaniu darmowej wersji programu i maksymalnym rozmiarze kodu możliwym do analizy – 2 kB. Kliknij przycisk **OK** i sprawdź działanie programu. Z menu **Peripherals** wybierz **I/O-Ports** odpowiadające za port **P0 i P2** użyte w programie (Rys. 14). Po wybraniu portów pojawią się okna symulujące stan na portach P0 i P2 (Rys. 15). Zapamiętaj, że okna symulatora zamkną się automatycznie po wyjściu z trybu odpluskwiania i pojawią się ponownie po wejściu do trybu odpluskwiania. Przetestuj wspomnianą funkcjonalność wybierając z menu **Debug** pozycję **Start/Stop Debuging Session**. Wejdź w tryb odpluskwiania. Uruchom program w trybie ciągłym w symulatorze klikając na **Debug->Run** (ikona z paska podręcznego ). Po uruchomieniu symulatora wartości portów zmienią się – Rys. 15. Pin P0.0 został ustawiony programowo jako wejście. Zmieniając stan na linii **pins P0** na **bicie nr 0** zmieniamy stan linii portu bitu nr 0 na **P2**.



Rys. 13 Uruchomienie Debuggera.



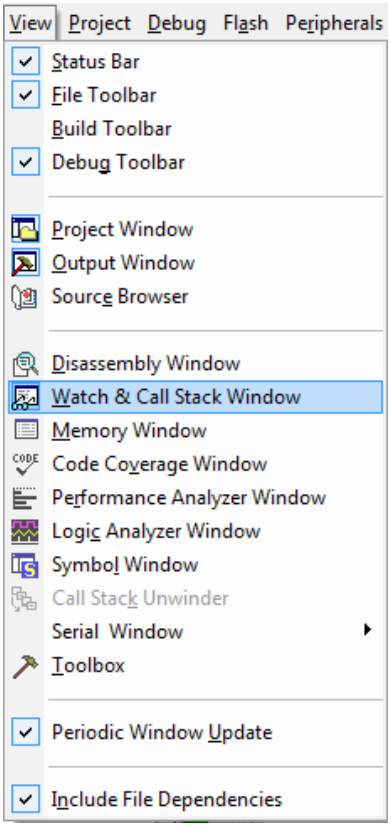
Rys. 14 Wybór portów.



Rys. 15 Zmiana stanów pinów.

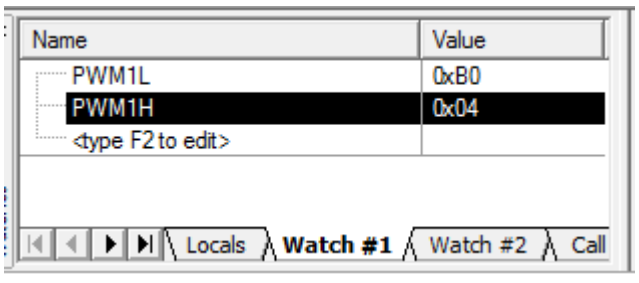
ii. DYNAMICZNY PODGLĄD ZMIENNYCH

Odpluskwiacz pozwala na dynamiczny podgląd zmiennych użytych w programie. Podczas uruchomionej symulacji programu wybierz z menu **View** opcję **Watch & Call Stack Window** (Rys. 16). Pojawi się nowe okno, w którym wybierz opcję **Watch #1** – Rys. 17. W aktywnej zakładce okna podejrzyj wartość wybranej zmiennej lub rejestru mikrokontrolera wciskając **klawisz F2** i wpisując nazwę zmiennej lub rejestru (Rys. 18). W celu zmiany wartości zmiennej klikamy LPM w pole **Value** i wpisujemy wartość (Rys. 19). Wartość zmiennej można przedstawić w różnych formatach. W celu zmiany formatu na system dziesiętny kliknij PPM na wybrana zmienną i z rozwiniętego menu wybierz właściwą podstawę (Rys. 20).

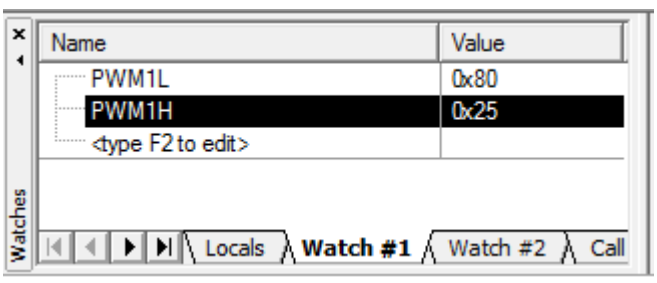


Rys. 16 Wybór okna do dynamicznego podglądu zmiennych.

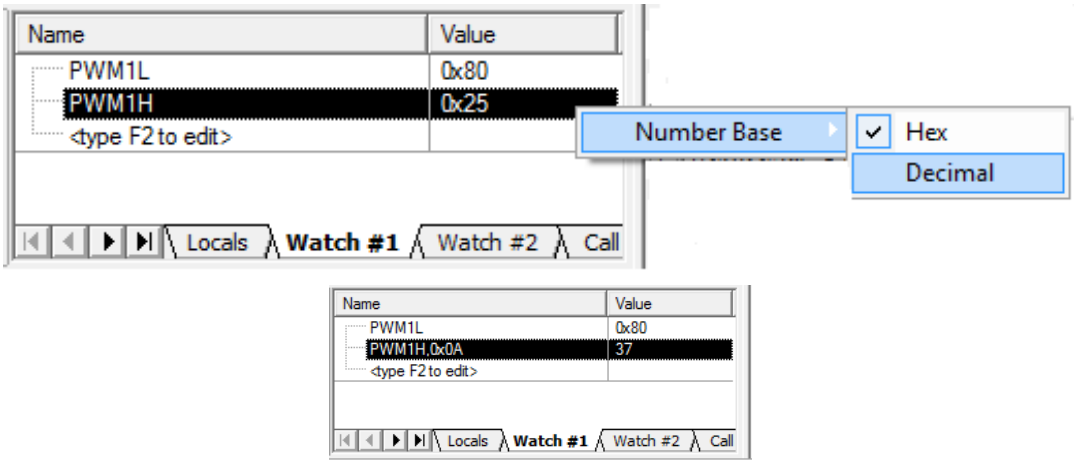
Rys. 17 Wybór opcji Watch #1.



Rys. 18 Zapis nazwy zmiennej.



Rys. 19 Zmiana Value.



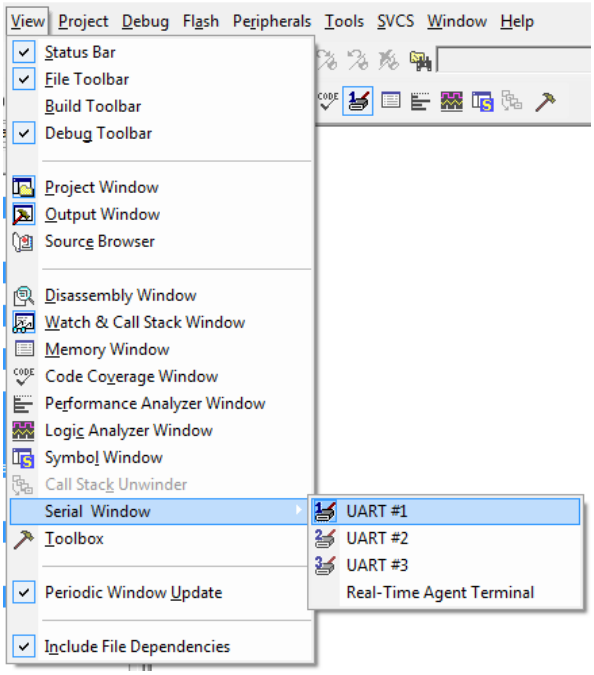
Rys. 20 Wybór podstawy wyświetlania zmiennej.

iii. ANALIZATOR SYGNAŁÓW

W poprzednim punkcie dowiedziałeś się jak podejrzeć obecną wartość zmiennej w trakcie działania programu. Informacja ta jest bardzo przydatna, lecz w niektórych sytuacjach warto uzyskać informację o zmianie danej wartości w czasie. Narzędziem posiadającym wspomnianą funkcjonalność jest **Logic Analyser Window**, dostępny w menu **View**.

iv. KOMUNIKACJA SZEREGOWA

Podgląd i wysyłanie znaków w komunikacji szeregowej zapewnia **Serial Window**. Serial Window otwiera oko do przesyłania i odbierania znaków do symulowanego UARTa. Symulator UARTa wykorzystasz na późniejszych zajęciach. W menu **View** należy wybrać opcję **Serial Window** a następnie **UART #1** - Rys. 21



Rys. 21 Serial Window.

e) KOMENTARZE W JĘZYKU C

Komentarz jest częścią kodu źródłowego, który nie podlega kompilacji. Służy do wstawiania informacji pozwalających na łatwiejsze zrozumienie zapisanego algorytmu. Komentarz pozwala na dokumentację kodu źródłowego na potrzeby własne jak i innych członków zespołu. Wyróżniamy dwa sposoby komentowania:

liniowy oraz blokowy. Stosowanie komentarza liniowego pozwala na wstawianie krótkich notatek na końcu poszczególnych linii kodu. Wstawienie komentarza odbywa się z wykorzystaniem składni //. Przykład użycia komentarza liniowego przedstawiono w Listing 3.

Listing 3 Komentarz zwykły

```
int a,b;
a=5; b=1; //określenie wartości początkowych
a=a+b ;//najpierw wykonana zostanie operacja dodawania, następnie wynik zostanie zapisany w zmiennej
```

Komentarz blokowy stosujemy do wprowadzenia dłuższych wypowiedzi, które są mało czytelne w przypadku zapisania w jednej linii. Treść komentarza umieszczamy między znacznikami początku i końca komentarza odpowiednio **/*** oraz ***/**. Przykład komentarza blokowego (Listing 4).

Listing 4 Komentarz blokowy

```
/*
Przykład dodawania dwóch zmiennych.
Przydziel pamięć na dwie zmienne.
Określ wartości początkowe poszczególnych zmiennych.
Najpierw wykonana zostanie operacja dodawania a+b. Wykorzystane zostaną wartości bieżące w zmiennych. Zostanie dodana wartość 5+1, następnie wynik operacji zostanie zapisany w zmiennej a.
*/
int a,b;
a=5; b=1;
a=a+b ;
```

Stosowanie właściwych komentarzy jest równie istotne co sam kod programu. Należy stosować komentarze liniowe oraz blokowe do opisu każdego algorytmu. Pojawia się pytanie gdzie umieszczać informacje oraz jaka powinna być ich treść. Zbyt krótkie komentarze mogą być równie niezrozumiałe dla odbiorcy co sam kod, natomiast zbyt długie komentarze są czasochłonne w czytaniu i tworzeniu jak również mogą zawierać zbyt wiele zbędnych informacji. Warto zapoznać się z systemami pozwalającymi na automatyczną generację dokumentacji na podstawie kodu źródłowego zaopatrzonego w stosowny komentarz. Treść komentarza zapisana w odpowiednim formacie poddawana jest analizie w wyniku, której generowana jest dokumentacja. Doxygen jest przykładem systemu generującego dokumentację na podstawie kodu źródłowego [5]. Zapoznaj się z składnią oraz przetestuj działanie generatora we własnym zakresie. Przykładowy komentarz przedstawia Listing 5.

Listing 5 Użycie stylu dokuemntjącego funkcję

```
double PI;          ///< stała PI

/**
 * Funkcja odwrotna do sinusa hiperbolicznego (area sinus hiperboliczny) \operatorname {arsinh}\ x=\ln(x+{\sqrt {x^{2}+1}}).
 * Dziedziną i przeciwdziedziną funkcji jest zbiór liczb rzeczywistych.
 * \param[in] x argument funkcji
 * \return wartość funkcji asinh w punkcie x
 */
double asinh(double x);
```

f) POLECENIE PREPROCESORA

Preprocesor jest narzędziem, które analizuje kod źródłowy poszukując dyrektyw preprocesora w celu ich wykonania. Pozwala to na zmianę kodu źródłowego przed kompilacją zależnie od dyrektyw preprocesora. Wynikowy kod źródłowy po procesie prekompilacji zostaje następnie poddany kompilacji. Podstawowe dyrektyw preprocesora to:

- #include - dyrektywa włącza zawartość innego pliku źródłowego w miejscu jej wystąpienia w pliku podlegającym aktualnie przetwarzaniu. Wyróżniamy dwie odmiany #include <nazwa pliku z rozszerzeniem> oraz #include „nazwa pliku z rozszerzeniem”. W przypadku użycia znaków <> prekompilator szuka pliku w folderze kompilatora, natomiast w sytuacji wykorzystania znaków „” prekompilator najpierw szuka pliku w folderze projektu jeśli go nie znajdzie to w folderze kompilatora.
- #define – definicja stałych wartości oraz makroinstrukcji,
- #undef – usuwa definicje,
- #if – dyrektywy kompilacji warunkowej,
- #elif – oznacza else if,
- #endif – oznacza koniec bloku kompilacji warunkowej,
- #ifdef – sprawdza czy istnieje podana definicja,
- #ifndef – sprawdza czy nie istnieje podana definicja,
- #warning – zwraca informację o ostrzeżeniu,
- #error – przerywa kompilację i zwraca informację o błędzie.

Pamiętaj by na końcu linii zawierającej makroinstrukcję nie wstawiać znaku **;**. Przykład wykorzystania makroinstrukcji przedstawia Listing 6.

Listing 6 Przykład makroinstrukcji

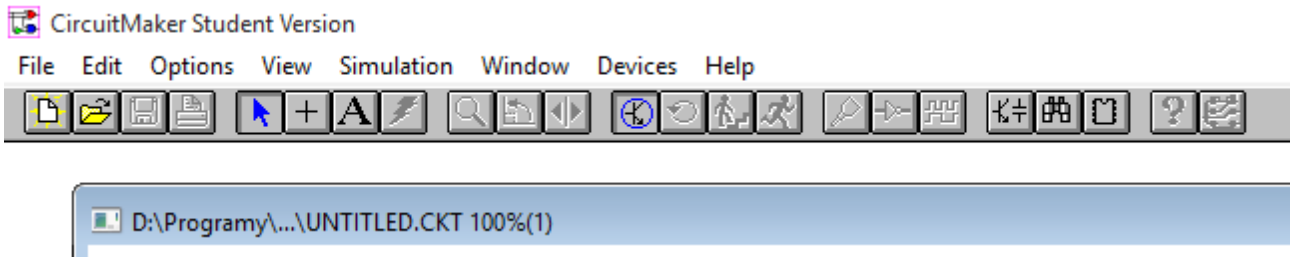
```
#define SUMA(a,b,c) (a+b+c)  //makroinstrukcja obliczająca sumę z 3 składników

int main(void){
int wynik;
wynik = SUMA(1,4,6); //po prekompilacji linia przyjmie postać wynik = (1+4+6);
return 0;
}
```

g) SYMULATOR UKŁADÓW LOGICZNYCH

i. PRACA SYMULATORA W TRYBIE CYFROWYM

Program CircuitMaker pozwala na wygodną analizę układów elektronicznych analogowych i cyfrowych. Po uruchomieniu programu widoczne jest główne okno programu Rys. 22 Główne okno programupatrz Rys. 22) pracującego w trybie analogowym. W celu zmiany trybu pracy symulatora z analogowego na cyfrowy wybrać należy z **Simulation** pozycję **Analog mode** lub klikając na ikonę tranzystora na pasku narzędzi (Rys. 23). Praca w trybie cyfrowy sygnalizowana jest przez znak bramki logicznej na pasku narzędzi (Rys. 24).



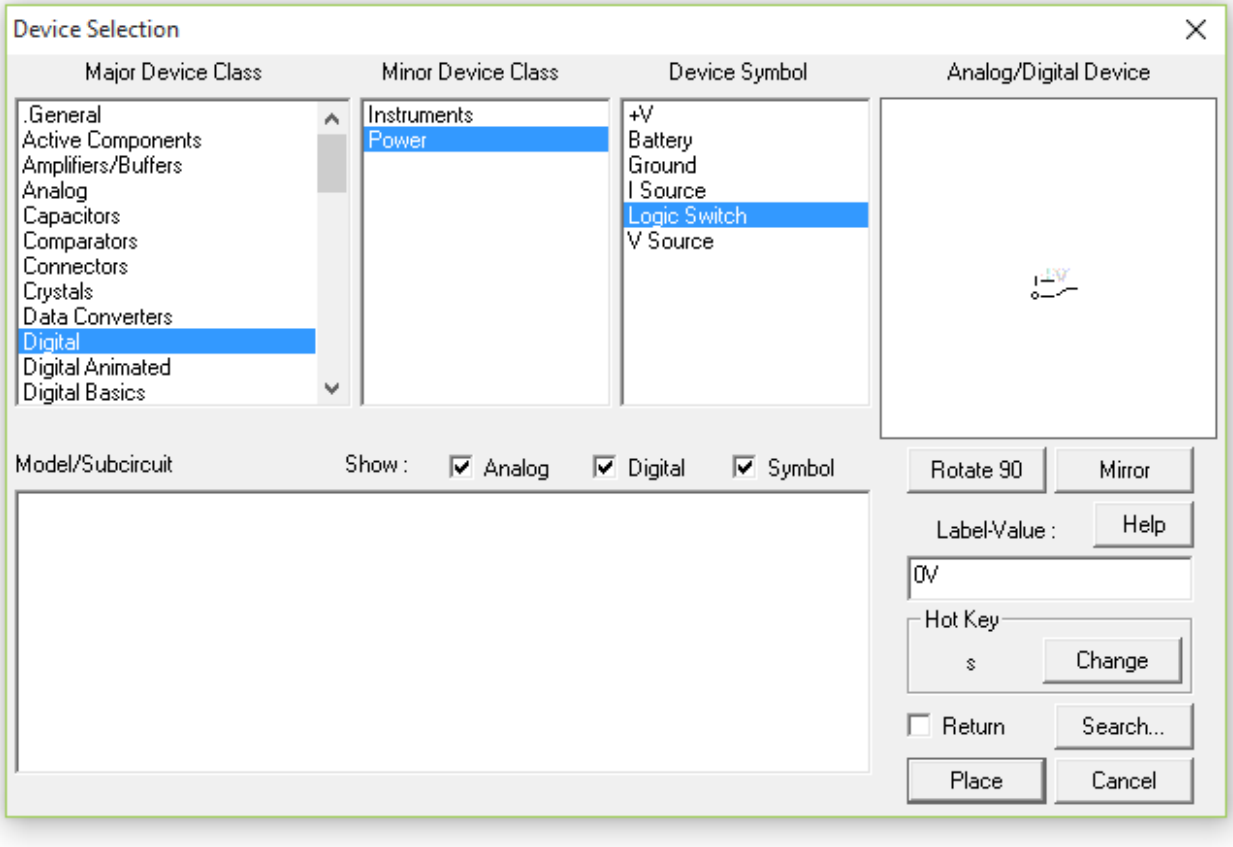
Rys. 22 Główne okno programu

Rys. 23 Praca w trybie analogowym

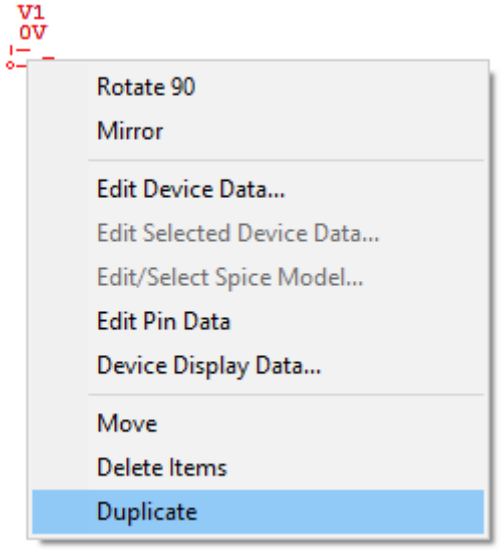
Rys. 24 Praca w trybie cyfrowym

ii. PRZYGOTOWANIE UKŁADU LOGICZNEGO

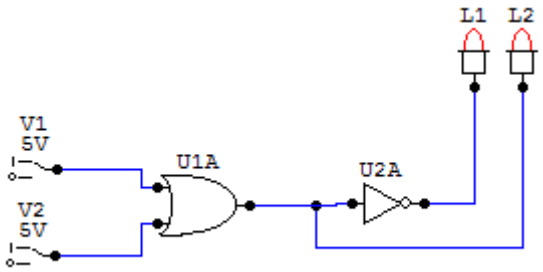
Elementy przygotowywanego układu logicznego dodać można klikając z paska narzędzi  . W nowo otwartym oknie (patrz Rys. 25) wybrać należy kategorię główną, pośrednią oraz konkretny element, którego dodanie zatwierdza się klikając **Place**. Na każdym z elementów można wykonać wybraną operację z menu kontekstowego (Rys. 26). Menu pojawia się po kliknięciu prawego przycisku myszy na elemencie. Dodawanie połączeń między elementami jest możliwe po wybraniu  . Połączenia dodawane są poprzez najechanie myszką końcówki elementu źródłowego (zostanie obramowane czerwonym prostokątem) a następnie kliknięcie i przytrzymanie lewego przycisku myszy w tym obszarze i przesunięcie do końcówki docelowej. Zmiana położenia połączeń oraz elementów jest dostępna po wybraniu  . Wykorzystując dostępne mechanizmy przygotowano prosty schemat układu logicznego przedstawionego na Rys. 27.



Rys. 25 Okno wyboru elementu



Rys. 26 Menu kontekstowe wybranego elementu

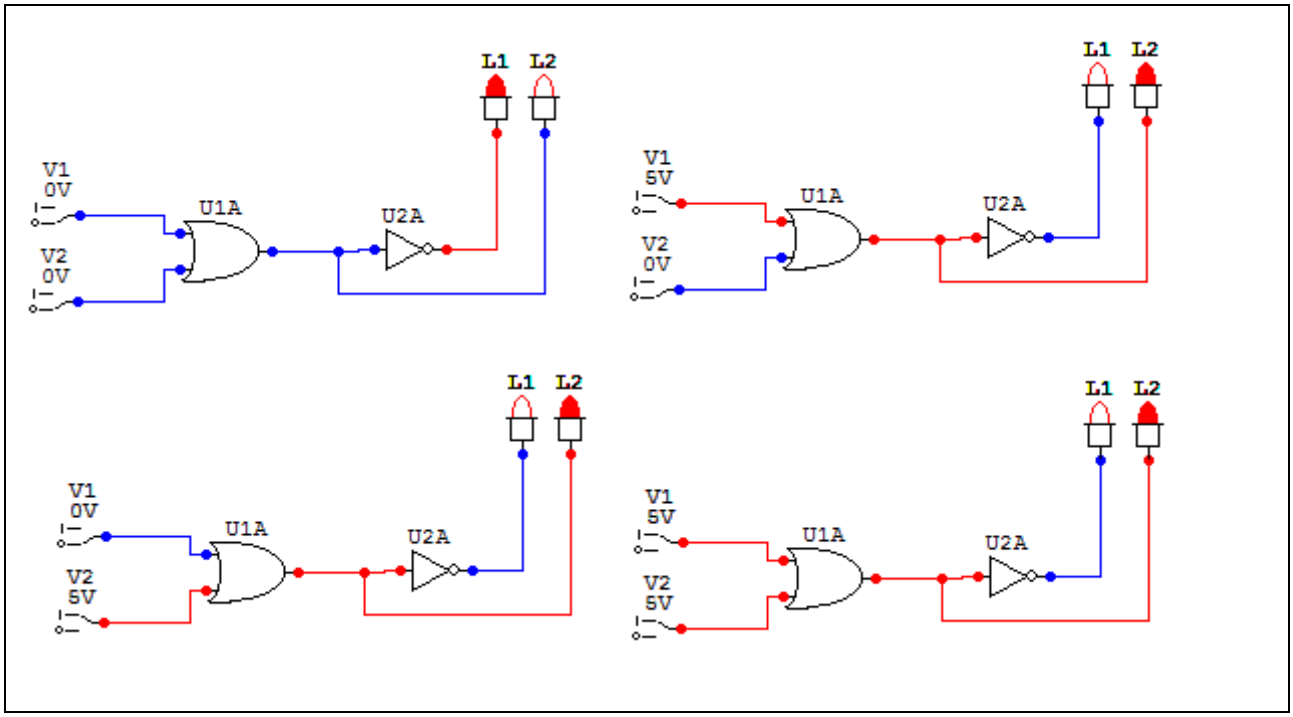


Rys. 27 Schemat prostego układu logicznego

iii. SYMULACJA UKŁADU LOGICZNEGO

Przygotowany schemat układu logicznego można przetestować w trybie symulacji, który uruchamiany jest poprzez wybranie  . Wyłączenie trybu symulacyjnego dokonuje się przez wybranie ikony z napisem STOP. Wybranie ikony  pozwala na podgląd stanów logicznych na liniach łączeniowych. Widok paska narzędzi w trybie symulacyjnym z włączonym podglądem stanów logicznych został przedstawiony na Rys. 28. Wyniki przeprowadzonych symulacji dla układu z Rys. 27 przedstawiono dla wszystkich możliwych stanów wejściowych na Rys. 29.

Rys. 28 Uruchomienie symulacji z podglądem stanów logicznych



Rys. 29 Symulacja przykładowego układu logicznego dla różnych stanów wejściowych

LITERATURA

1. Keil 8051 Microcontroller Development Tools [online]. [udostępniono 19.11.2014]. Pobrano: http://www.keil.com/c51/.

2. C51 Version 9.53 Evaluation Software Request [online]. [udostępniono 19.11.2014]. Pobrano: https://www.keil.com/demo/eval/c51.htm.

3. ADUC831 datasheet and product info | Precision Analog Microcontroller: 1.3MIPS 8052 MCU + 62kB Flash + 8-Ch 12-Bit ADC + Dual 12-Bit DAC | Analog Microcontrollers | Analog Devices [online]. [udostępniono 19.11.2014]. Pobrano: http://www.analog.com/en/processors-dsp/analog-microcontrollers/aduc831/products/product.html.

4. *ADuC831 MicroConverter, 12-Bit ADCs and DACs with Embedded 62 kBytes Flash MCU Data Sheet, (Rev. 0) - ADUC831.pdf* [online]. s.l.: s.n. [udostępniono 19.11.2014]. Pobrano: http://www.analog.com/static/imported-files/data_sheets/ADUC831.pdf.

5. Doxygen: Main Page [online]. [udostępniono 19.11.2014]. Pobrano: http://www.stack.nl/~dimitri/doxygen/.