Aleksander DZIUDA
Waldemar KRUPA

LABORATORIUM TECHNIKI CYFROWEJ

KOLEGIUM KARKONOSKIE
Państwowa Wyższa Szkoła Zawodowa
w Jeleniej Górze

mgr inż. Aleksander Dziuda

mgr inż. Waldemar Krupa

LABORATORIUM TECHNIKI CYFROWEJ

JELENIA GÓRA 2006

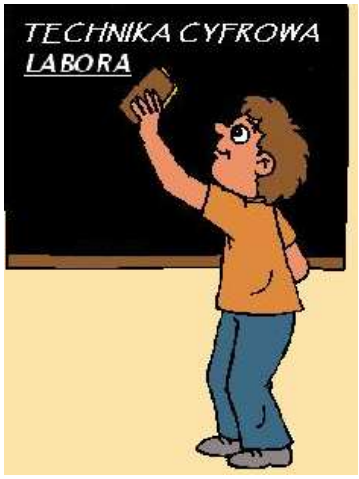
PRZEWODNICZĄCY RADY
PROGRAMOWO - WYDAWNICZEJ
dr hab. Henryk GRADKOWSKI

RECENZENT
prof. dr hab. inż. Jan Jagielski

Niniejsze wydawnictwo można kupić w Bibliotece Uczelnianej
Kolegium Karkonoskiego PWSZ
w Jeleniej Górze
ul. Lwówecka 18 tel. 0 75 645 33 52

Druk i oprawa: INTROLIGATORSTWO - MAŁA POLIGRAFIA
58-503 ul. Podchorążych 1/6
tel. 0 75 6474006 tel. kom. 0 501 181516

ISBN 83-912031-7-4
ISBN 978-83-912031-7-0



SPIS TREŚCI

OD AUTORÓW. 5

1. REGULAMIN LABORATORIUM TECHNIKI CYFROWEJ. 7

2. MAKIETA LABORATORYJNA ML-1. 19

2.1. Struktura sprzętowa systemu. 19
2.1.1. Moduł układów cyfrowych. 20
2.1.2. Tryby pracy mikroprocesorowego sterownika systemu. 24
2.1.3. Zasilacz. 27
2.1.4. Inne elementy. 27
2.2. Program „Makieta laboratoryjna 1.0”. 28
2.2.1. Interfejs użytkownika. 28
2.2.2. Komunikaty występujące w programie. 34
2.2.3. Pomoc programu. 37

3. ĆWICZENIA LABORATORYJNE. 39

3.1. Cele kształcenia. 39
3.2. Charakterystyka ćwiczeń. 39
3.2.1. Wprowadzenie do ćwiczeń laboratoryjnych, wymagania, literatura, zasady BHP. 40
3.2.2. Zapoznanie się z makietą laboratoryjną, możliwościami, właściwościami, oprogramowaniem sterownika. 40
3.2.3. Podstawowe elementy logiczne rodziny 74 LS. 41
3.2.4. Układy konwersji kodów. 43
3.2.5. Układy komutacyjne. 44
3.2.6. Rejestry równoległe i przesuwne. 46
3.2.7. Liczniki asynchroniczne. 48
3.2.8. Liczniki synchroniczne. 49
3.2.9. Układy arytmetyczne. 50
3.2.10. Termin obróbczy. 53

3.3. Elementy logiczne i bloki funkcjonalne występujące na stanowisku laboratoryjnym. 54
3.3.1. Podstawowe oznaczenia graficzne. 54
3.3.2. Wykaz układów scalonych 74LS serii TTL dostępnych na stanowisku laboratoryjnym. 57
3.3.3. Funkcje układów scalonych 74LS występujących na stanowisku laboratoryjnym. 58
3.3.4. Wyciąg z not katalogowych układów scalonych 74LS występujących na stanowisku laboratoryjnym. 61
3.4. Literatura do ćwiczeń laboratoryjnych. 76
4. NARZĘDZIA WSPOMAGAJĄCE PROJEKTOWANIE UKŁADÓW CYFROWYCH W STRUKTURACH PLD. 77
4.1. Programy systemu WinCUPL. 77
4.2. Podstawy języka CUPL. 79
4.2.1. Elementy języka i składnia. 80
4.2.2. Preprocesor. 104
4.2.3. Opis automatów. 107
4.2.4. Format pliku źródłowego. 110
4.3. Program WinCUPL. 115
4.3.1. Obsługa WinCUPL. 116
4.3.2. Praca z przykładowym projektem. 126
4.3.3. Pierwszy program. 132
4.4. Programator LabTool-48XP. 146
4.3.1. Praca z programatorem. 149
4.3.2. Programowanie układu. 153
4.5. Literatura do rozdziału 4. 157

OD AUTORÓW

Niniejsze opracowanie jest przeznaczona dla studentów przygotowujących się i wykonujących ćwiczenia laboratoryjne w ramach przedmiotów „*Technika cyfrowa*” na kierunku ELEKTRONIKA I TELEKOMUNIKACJA oraz „*Elektrotechnika i elektronika*” z zakresu układów cyfrowych na kierunku EDUKACJA TECHNICZNO-INFORMATYCZNA, prowadzonych w Instytucie Techniki Kolegium Karkonoskiego.

Przyjmuje się, że studenci znają działanie i właściwości projektowanych układów logicznych z informacji przekazywanych na wykładach i ćwiczeniach audytoryjnych poprzedzających zajęcia laboratoryjne.

Opracowanie zawiera Regulamin Laboratorium Techniki Cyfrowej, opis Makiety Laboratoryjnej ML-1 na której realizowane są ćwiczenia laboratoryjne oraz programu „Makieta laboratoryjna 1.0”, materiały poświęcone każdemu ćwiczeniu i opis narzędzi wspomagających projektowanie układów cyfrowych w strukturach PLD.

W regulaminie laboratorium określono cel ćwiczeń laboratoryjnych, zasady przygotowania się i zaliczenia ćwiczeń oraz wskazówki o sposobie wykonania sprawozdań z ćwiczeń. Ponadto przedstawiono podstawowe zasady BHP obowiązujące w Laboratorium Techniki Cyfrowej podczas wykonywania ćwiczeń.

Materiały poświęcone każdemu ćwiczeniu obejmują charakterystykę badanych układów, proponowany program ćwiczenia oraz przykładowe problemy do samodzielnego rozwiązania.

Narzędzia wspomagające projektowanie układów cyfrowych w strukturach PLD zapoznają z podstawami języka CUPL, programami systemu WinCUPL oraz programatorem LabTool-48XP.

Informacje zawarte w pracy pozwalają na wykonanie ćwiczeń zgodnie z podanym programem. Załączony spis literatury powinien zachęcić studentów do samodzielnego pogłębiania i rozszerzania swojej wiedzy z zakresu Techniki Cyfrowej.

1. REGULAMIN

LABORATORIUM TECHNIKI CYFROWEJ

Laboratorium z Techniki Cyfrowej jest ilustracją wykładów i ćwiczeń audytoryjnych o tym samym tytule, prowadzonych w Instytucie Techniki Kolegium Karkonoskiego.

Zajęcia laboratoryjne mają na celu:

- Praktyczną weryfikację, ugruntowanie i poszerzenie nabytej wiedzy teoretycznej oraz kształtowanie praktycznych umiejętności wykorzystania scalonych układów cyfrowych, poprawnego odczytywania wyników i prowadzenia odpowiedniej dokumentacji pomiarów.
- Kształtowanie umiejętności metrologicznych, takich jak: prawidłowe zaplanowanie pomiarów, celem przeprowadzenia doświadczenia, poprawna organizacja stanowiska pomiarowego i właściwa eksploatacja aparatury oraz ocena i dyskusja wyników pod względem ich wiarygodności.
- Zapoznanie studentów z elementami cyfrowymi klasy TTLS, stosowanymi strukturami układów cyfrowych, a także metodyką projektowania urządzeń cyfrowych na poziomie stosowania bloków funkcjonalnych.

Oczekuje się, że uczestnictwo w zajęciach pozwoli studentom na wykonywanie samodzielnych projektów z użyciem układów cyfrowych oraz interpretację i dyskusję otrzymanych wyników w trakcie badań.

Podstawą przygotowania się do ćwiczeń laboratoryjnych są wiadomości nabyte przez studentów podczas wykładów i ćwiczeń audytoryjnych w ramach kursów „Technika cyfrowa I” i „Technika cyfrowa II” oraz korzystanie z dostępnej, obowiązkowej i zalecanej literatury. Realizacja ćwiczeń laboratoryjnych, to samodzielna działalność i podejmowanie decyzji przez studentów w trakcie wykonywania ćwiczeń.

Przygotowanie się studentów do ćwiczeń laboratoryjnych polega na:

- zapoznaniu się z tematem i celem ćwiczenia,
- przyswojeniu teoretycznych podstaw zachodzących w badanych układach,
- poznaniu zasad badania określonych zjawisk i wynikającego z niej układu,
- określeniu przewidywanych efektów pomiarów (charakterystyk, wyników, itp.),
- przyswojeniu sobie wiadomości dotyczących budowy i zasady działania użytych w ćwiczeniu przyrządów pomiarowych,
- przygotowaniu protokołu pomiarów (tabelki, schematy, wstępne obliczenia).

Każdy student przed rozpoczęciem ćwiczeń laboratoryjnych obowiązany jest zapoznać się z *Regulaminem Laboratorium Techniki Cyfrowej* oraz *Przepisami BHP obowiązującymi w laboratorium*, co winno być udokumentowane podpisem.

Studenci wykonują ćwiczenia w Laboratorium Techniki Cyfrowej w podgrupach składających się z zespołów dwuosobowych lub w szczególnych przypadkach trzyosobowych pod kierunkiem pracownika Instytutu Techniki.

Ćwiczenia wykonywane są w systemie całościowym, tzn. ustawiony jest w laboratorium jeden temat ćwiczeń, realizowany przez wszystkie podgrupy.

Podgrupy laboratoryjne wykonują ćwiczenie według ustalonego przez prowadzącego schematu.

Po wykonaniu każdego etapu ćwiczenia należy uzyskane wyniki zaprezentować prowadzącemu w celu sprawdzenia i potwierdzenia realizacji programu.

Każda podgrupa laboratoryjna prowadzi jeden protokół, w którym notuje wyniki pomiarów, uwagi dotyczące przebiegu ćwiczenia, schematy połączeń, obliczenia oraz polecenia prowadzącego zajęcia dotyczące danego ćwiczenia. Zatwierdzenie protokołu oznacza zaliczenie części praktycznej danego ćwiczenia. W sprawozdaniu zawarte winny być tylko wyłącznie wyniki z zatwierdzonego protokołu.

Studenci przedstawiają sprawozdanie z ćwiczenia w formie pisemnej. Szczegółowe zasady wykonywania sprawozdania określa prowadzący kurs.

Sprawozdanie, wraz z zatwierdzonym protokołem pomiarów, studenci mają obowiązek przedstawić prowadzącemu do oceny na następnych zajęciach. W przypadku niespełnienia tego warunku studenci nie są dopuszczoni do wykonywania ćwiczenia.

W ciągu semestru odbywa się 8 ćwiczeń (8 spotkań). Każde ćwiczenie trwa 3 godziny lekcyjne. Pierwsze spotkanie organizacyjne stanowi wprowadzenie oraz dokonywany jest podział na podgrupy laboratoryjne. Dziesiąte spotkanie przeznaczone jest na ewentualne odrobienie zaległego ćwiczenia.

W przypadkach usprawiedliwionej nieobecności na ćwiczeniach istnieje możliwość odrobienia ćwiczenia z inną grupą laboratoryjną, jeżeli będą wolne miejsca. Przypadek taki należy uzgodnić z prowadzącym ćwiczenia w danej grupie.

W czasie ćwiczenia obowiązuje znajomość instrukcji oraz teorii z wykładu i ćwiczeń audytoryjnych obejmujących tematykę ćwiczenia.

W przypadku niedostatecznego przygotowania teoretycznego prowadzący może niedopuścić studenta do wykonywania ćwiczenia. Przypadek taki jest równoznaczny z nieobecnością nieusprawiedliwioną.

Student, który nie został dopuszczony do wykonywania ćwiczenia ma prawo uczestniczyć w wykonywaniu ćwiczenia, jednak bez możliwości oddania protokołu i sprawozdania. Takie ćwiczenie musi być powtórzone w czasie wyznaczonym przez prowadzącego w ramach limitu nieobecności nieusprawiedliwionych.

W przypadku rażąco nieudolnego wykonywania ćwiczenia, nieumiejętnego posługiwania się sprzętem laboratoryjnym, niszczenia sprzętu pomiarowego a także innego wyposażenia laboratorium lub innego postępowania szkodliwego z punktu widzenia pozostałych osób korzystających z laboratorium studenci mogą zostać usunięci z laboratorium.

Sposób oceniania wykonania ćwiczenia przez studentów jest przedstawiany przez prowadzącego w czasie zajęć wprowadzających do ćwiczeń laboratoryjnych.

W laboratorium obowiązuje następujący regulamin:

1. Płaszcze i inne okrycia wierzchnie należy pozostawić w szatni.
2. W laboratorium należy zachować ciszę. Konwersację z innymi studentami należy ograniczyć do niezbędnego minimum i prowadzić głosem ściszonym, tak aby nie zakłócić ciszy i nie rozpraszać pozostałych ćwiczących.
3. Nie wolno samowolnie zamieniać ani przenosić przyrządów oraz korzystać ze sprzętu laboratoryjnego nie wchodzącego w zestaw danego ćwiczenia.
4. Modelowania realizowanych w ramach ćwiczeń laboratoryjnych projektów układów studenci dokonują używając wyłącznie układów scalonych znajdujących się na danym stanowisku laboratoryjnym.

5. Studenci mogą włączać napięcia zasilające oraz włączać i wyłączać komputery wyłącznie za zgodą prowadzącego.
6. Studenci są zobowiązani do zachowania szczególnej ostrożności w użytkowaniu przyrządów i komputerów. O wszelkich uszkodzeniach należy natychmiast powiadomić prowadzącego zajęcia.
7. Winni zniszczenia lub uszkodzenia układu wskutek samowolnego włączenia napięcia lub niestosowania się do wskazówek prowadzącego ćwiczenia – będą usuwani z zajęć.
8. Rozłączenie badanego układu tak w trakcie ćwiczenia jak i po zakończeniu pomiarów jest dozwolone po zatwierdzeniu wyników przez prowadzącego ćwiczenie.
9. Po wykonaniu ćwiczenia studenci wykonują jedno sprawozdanie na podgrupę.
10. Zaliczenie ćwiczeń laboratoryjnych następuje na podstawie;
    - ✓ wykonania wszystkich ćwiczeń przewidzianych programem laboratorium,
    - ✓ zaliczenia sprawozdań ze wszystkich wykonanych w laboratorium ćwiczeń.
11. Zabrania się dokonywania jakichkolwiek zmian w konfiguracji systemu Windows 98SE (dotyczy to również pulpitu!) i w zainstalowanych programach.
12. Zakazuje się wgrywania, „ściągania” z Internetu oraz instalowania oprogramowania na komputerach znajdujących się w laboratorium.
13. Zabrania się tworzenia nowych katalogów oraz wgrywania plików do komputerów znajdujących się w laboratorium.
14. Nie wolno umieszczać na pulpicie systemu Windows 98SE żadnych skrótów, folderów czy plików.

15. W trakcie zajęć zabrania się z korzystania z Internetu (przeglądarek WWW, poczty elektronicznej, a szczególnie z usług typu: Gadu-Gadu, IRC, Chat, itp.). Można korzystać z przeglądarek WWW w przypadku, gdy jest to wymagane do zaliczenia danego ćwiczenia.
16. Wykonując dane ćwiczenie należy pracować wyłącznie w katalogu roboczym wskazanym w instrukcji tego ćwiczenia i zgodnie z tą instrukcją.

Celem uniknięcia nieszczęśliwym wypadkom należy przestrzegać następujących zasad (*wyciąg z instrukcji BHP*):

1. Pracę na stanowiskach laboratoryjnych powinna cechować szczególna ostrożność i rozwaga.
2. Przed przystąpieniem do łączenia układu pomiarowego, należy sprawdzić czy źródła zasilania są odłączone.
3. Zabrania się używania przewodów łączących z uszkodzoną izolacją lub oberwanymi końcówkami.
4. Zabrania się włączenia napięcia bez sprawdzenia układu przez prowadzącego ćwiczenia i uzyskania jego zgody.
5. Wszystkie zmiany w układzie, jak również jego rozłączenie jest dozwolone jedynie przy odłączeniu napięcia.
6. Włączenie napięcia po dokonanej zmianie w układzie może nastąpić dopiero po sprawdzeniu układu przez prowadzącego ćwiczenia.
7. Zabrania się dotykać części elementów układu znajdującego się pod napięciem, a w szczególności zacisków przyrządów i końcówek przewodów łączących.
8. W razie odłączenia źródeł zasilania przez bezpiecznik automatyczny, znajdujący się na listwie zasilającej, należy niezwłocznie powiadomić o tym fakcie prowadzącego ćwiczenia.
9. W przypadku porażenia prądem elektrycznym należy:

- natychmiast wyłączyć napięcie zasilające za pomocy wyłącznika głównego lub wyłączników znajdujących się na listwach zasilających,
- uwolnić porażonego spod działania prądu elektrycznego,
- powiadomić prowadzącego ćwiczenia oraz lekarza,
- przystąpić do udzielenia pierwszej pomocy zgodnie z zasadami udzielania pomocy w przypadku porażenia prądem,
- poddać poszkodowanego opiece lekarskiej, niezależnie od tego czy stracił przytomność czy nie.

11. W razie pożaru należy:
    - wyłączyć napięcie zasilające,
    - nieść pomoc osobom zagrożonym, zachować spokój, nie wywoływać paniki,
    - przystąpić do gaszenia pożaru za pomocy dostępnych środków gaśniczych, np.: kocy gaśniczych, gaśnic. Uwaga! do gaszenia urządzeń elektrycznych nie wolno używać gaśnic pianowych,
    - powiadomić straż pożarną – telefon 998,
    - studenci, którzy nie biorą udziału w akcji ratowniczej muszą bezwzględnie opuścić laboratorium.
12. W razie powstania innego wypadku (urazy mechaniczne, chemiczne, itp.) należy:
    - usunąć czynniki szkodliwe,
    - zawiadomić o wypadku pracownika prowadzącego ćwiczenia,
    - udzielić pierwszej pomocy,
    - w razie potrzeby wezwać lekarza, ewentualnie przewieźć poszkodowanego do szpitala.

UWAGA!

- Wykonując ćwiczenie **PRZESTRZEGAJ** przepisów BHP związanych z obsługą urządzeń elektrycznych.
- Uszkodzenia bądź nieprawidłowości w funkcjonowaniu urządzeń **ZGŁOŚ** prowadzącemu zajęcia. Urządzenia uszkodzone **ODSTAW** w miejsce z opisem "Urządzenia uszkodzone".
- Po wykonaniu ćwiczenia laboratoryjnego:
  - **ROZŁĄCZ** badane układy,
  - **WYŁĄCZ** zasilanie urządzeń i stołu,
  - **UŁÓŻ** przewody w uchwytach,
  - **ODSTAW** urządzenia przestawione z innych stanowisk na ich pierwotne miejsce.

Zalecany układ sprawozdania:

- ✓ *tabela nagłówkowa sporządzona według wzoru;*

KOLEGIUM KARKONOSKIE INSTYTUT TECHNIKI			
LABORATORIUM TECHNIKI CYFROWEJ			
Numer grupy:	*Numer ćwiczenia:*	*Prowadzący:*	
Skład podgrupy: *1.* *2.*	*Temat ćwiczenia:*		
	Data wykonania:	*Ocena:*	*Podpis:*

- ✓ *cel ćwiczenia;*
- ✓ *rozwiązania zadań otrzymanych w ramach przygotowania się do ćwiczeń laboratoryjnych;*
- ✓ *schematy połączeń zaprojektowanych układów faktycznie stosowanych w ćwiczeniu;*
- ✓ *tabele z wynikami badań i obliczeń;*
- ✓ *wzory stosowane do obliczeń i obliczenia;*
- ✓ *przebiegi czasowe;*
- ✓ *wnioski i uwagi z przebiegu ćwiczenia i otrzymanych wyników.*

Oceniając sprawozdanie bierze się pod uwagę:

- ✓ *poprawność otrzymanych wyników i wykresów;*
- ✓ *stopień wyczerpania tematu we wnioskach;*
- ✓ *staranność wykonania schematów, tabel i wykresów.*

Uwaga: do sprawozdania winien być dołączony protokół z pomiarów zawierający:

- *krótką charakterystykę przeprowadzonych badań, uwzględniając cel ćwiczenia oraz zastosowaną metodę pomiaru;*
- *dyskusję otrzymanych wyników i analizę wykonanych wykresów w oparciu o teoretyczne uzasadnienie zjawisk;*
- *uwagi i spostrzeżenia poczynione w trakcie ćwiczenia.*

Sprawdzone i zaliczone sprawozdania przechowywane są do końca semestru przez prowadzących zajęcia laboratoryjne.

Zasady zaliczenia laboratorium

Ustala się następujące zasady zaliczenia laboratorium:

1. Zaliczenie wszystkich ćwiczeń laboratoryjnych przewidzianych programem studiów.
2. Dopuszczalna jest nieobecność na dwóch ćwiczeniach pod warunkiem ich wykonania w wyznaczonych terminach odróbczych.
   Liczba nieobecności przewyższająca 2 będzie traktowana zgodnie z regulaminem studiów Kolegium Karkonoskiego.
3. Ocena końcowa będzie wyznaczona jako średnia arytmetyczna ocen cząstkowych uzyskanych z każdego ćwiczenia.
4. Ocenie może podlegać aktywność studentów podczas laboratorium.

Ad.1. Warunkiem zaliczenia ćwiczenia jest:

a) Przygotowanie do ćwiczenia, zgodne z wykazem materiału obowiązującego wykazanego w instrukcji do ćwiczenia.

b) Pozytywna ocena ze sprawdzianu weryfikującego przygotowanie do ćwiczenia (jeśli ćwiczenie zostało objęte sprawdzianem).

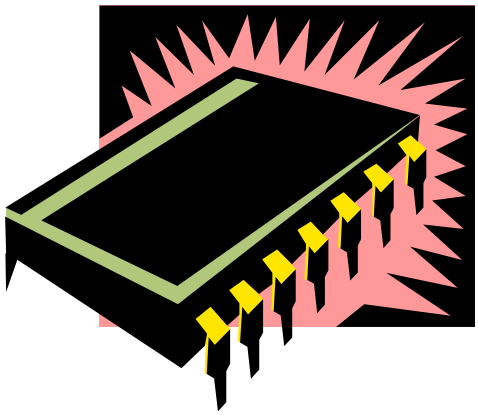
   Dopuszcza się przeprowadzenie kolokwium z zakresu materiału obejmującego wykonane ćwiczenia w terminie ustalonym z co najmniej 14 dniowym wyprzedzeniem oraz po uzgodnieniu wagi jego oceny. Zaliczenie tematyki ćwiczenia może nastąpić w trakcie wykonywania ćwiczenia lub poprzez kolokwium po każdej serii ćwiczeń

c) Rozliczenie sprawozdania z ćwiczenia poprzedniego.

d) Uzyskanie pozytywnej oceny z wykonywania ćwiczenia.

Uwaga:

Niespełnienie warunków, o których mowa w punktach a) - d), może spowodować usunięcie ćwiczącego z zajęć i równoczesne wykreślenie z listy obecności w danym dniu.



2. MAKIETA LABORATORYJNA ML-1

2.1. Struktura sprzętowa systemu

Rys. 2.1. Widok stanowiska laboratoryjnego

Stanowisko laboratoryjne w przedstawionej powyżej konfiguracji (rysunek 2.1) przeznaczone jest do wykonywania ćwiczeń laboratoryjnych z „*Techniki cyfrowej*” na kierunku ELEKTRONIKA I TELEKOMUNIKACJA oraz „*Elektrotechniki i elektroniki*” z zakresu układów cyfrowych na kierunku EDUKACJA TECHNICZNO-INFORMATYCZNA.

Do właściwych zasobów sprzętowych systemu należą:

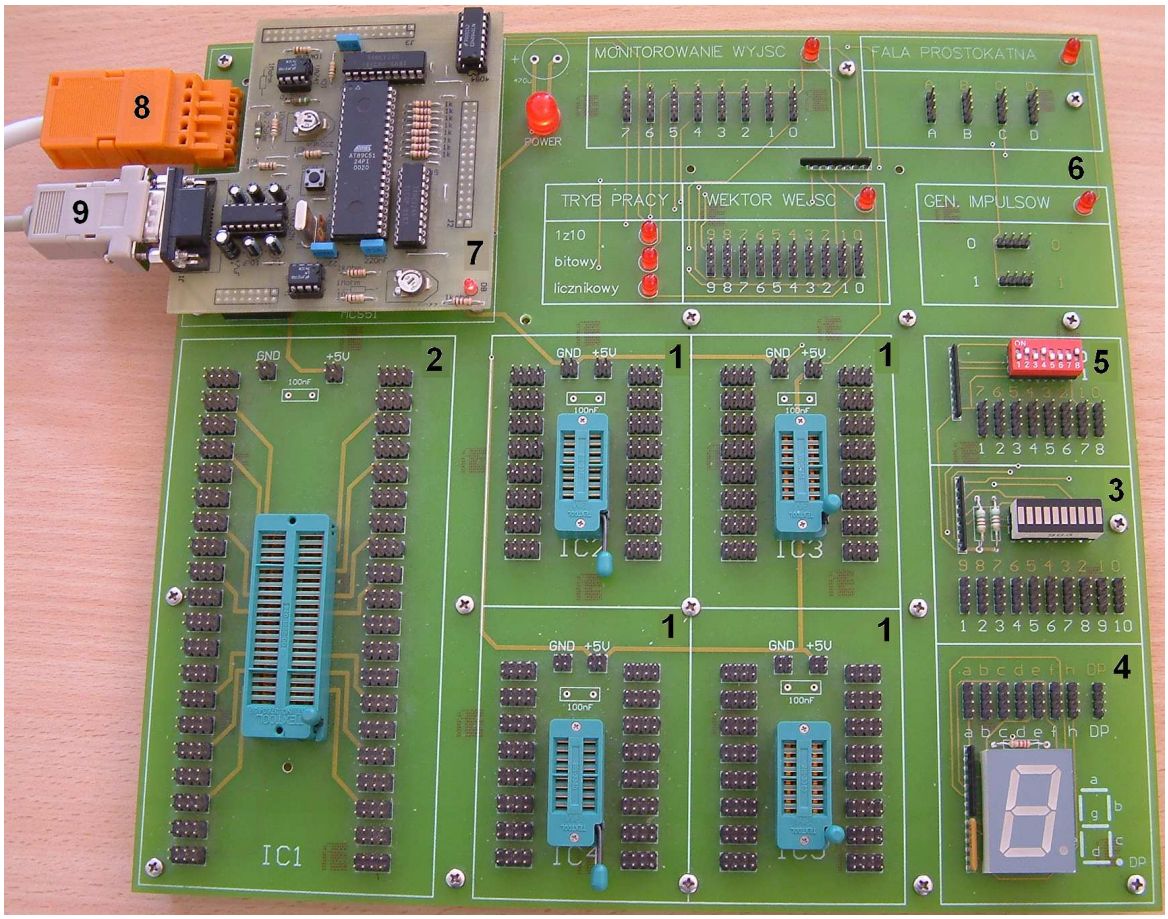
* Moduł układów cyfrowych wraz z mikroprocesorowym sterownikiem systemu;
* Komputer klasy PC wraz z oprogramowaniem;
* Zasilacz;
* Inne elementy: zestaw układów scalonych klasy TTLS, przewody połączeniowe.

2.1.1. Moduł układów cyfrowych

Moduł układów cyfrowych jest uniwersalnym układem zawierającym podstawowe elementy umożliwiające szybkie zamodelowanie badanego układu cyfrowego i dokonanie niezbędnych prób. Służy do modelowania układów cyfrowych projektowanych oraz badanych w ramach ćwiczeń laboratoryjnych. Wygląd modułu przedstawia rysunek 2.2.

Moduł zawiera następujące elementy:

1. 4 podstawki 16 nóżkowe zerowego docisku (TEXTOOL);
2. 1 podstawkę 40 nóżkową zerowego docisku (TEXTOOL);
3. 10 diod elektroluminescencyjnych LED zespolonych w jednym bloku;
4. 7-segmentowy wskaźnik cyfrowy;
5. mikroprzełącznik DIP Switch 8-bitowy;
6. pola sygnałów zerojedynkowych i sygnałów impulsowych:
   - monitorowanie wyjść;
   - fala prostokątna;
   - tryb pracy;
   - wektor wejść;
   - generator impulsów;
7. mikroprocesorowy sterownik systemu;
8. złącze zasilające;
9. łącze szeregowe (RS – 232).



Rys. 2.2. Moduł układów cyfrowych.

Moduł układów cyfrowych jest zasilany napięciem stabilizowanym + 5V.

Każda podstawka (1, 2) pod układ scalony ma wyprowadzone wszystkie nóżki na zgrupowane wokół niej szpilki. Zasilanie układu scalonego włożonego do podstawki można dołączyć w sposób elastyczny, zależnie od typu, wykorzystując szpilki zasilania GND i +5V doprowadzone bezpośrednio w pobliże podstawki. Dźwigienka dociskowa umożliwia łatwą wymianę układu scalonego.

Zespolone diody LED (3) *(linijka 10-diodowa koloru białego)* służą do bezpośredniej obserwacji stanów logicznych badanego układu (max. 10 wejść/wyjść). Sygnałem aktywnym (świecenie diody) jest poziom niski LOW – odpowiada to 0 logicznemu.

Wskaźnik 7-segmentowy (4) umożliwia badanie układów transkoderów kodu BCD (lub innych) na kod wskaźnika. Poszczególne segmenty wskaźnika

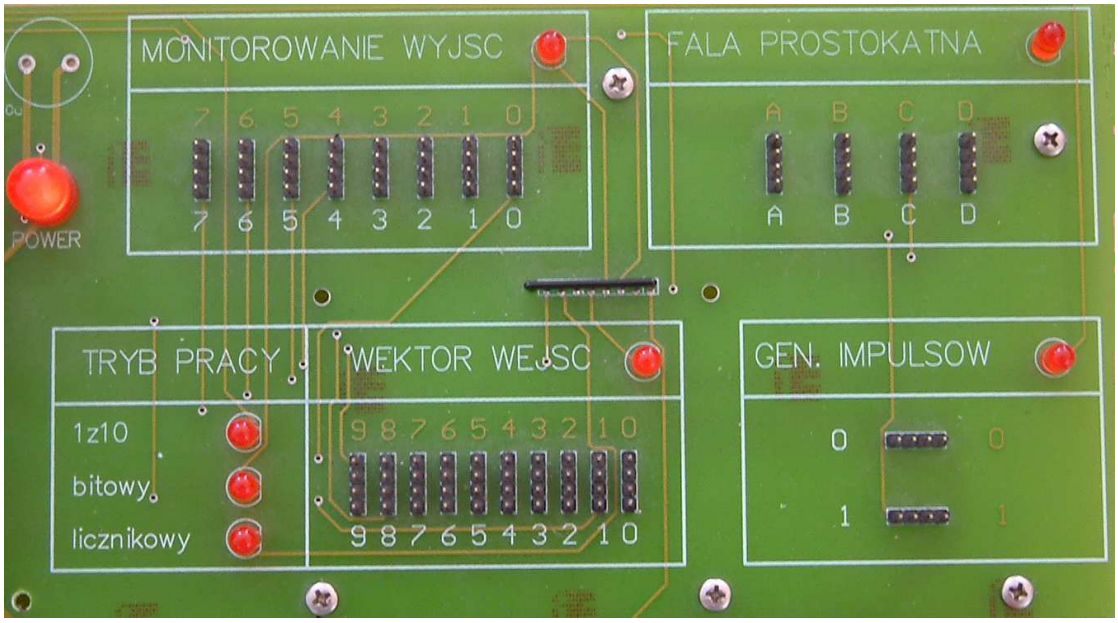
zostały oznaczone literami a, b, c, d, e, f, g, h. Sygnałem aktywnym (świecenie segmentu) jest poziom niski LOW.

Mikroprzełącznik 8-segmentowy (5) umożliwia dołączenie, zależnie od potrzeb, stanów logicznych 0 i 1 (poziomów LOW i HIGH). Odpowiadające tym stanom poziomy napięć generowane są na wyjściach 1 – 8 mikroprzełącznika stosownie do ustawiania odpowiedniego segmentu.

UWAGA: w pozycji 1 segmentu na stosownym wyjściu pojawia się poziom HIGH (1 logiczna), a w pozycji 0 segmentu na wyjście dołączony jest poziom LOW (0 logiczne).

Pola sygnałów zerojedynkowych i sygnałów impulsowych (6) ilustrują stan i pracę mikroprocesorowego sterownika systemu (rysunek 2.3):

- tryby pracy,
- wektora stanów wejściowych,
- generatora impulsów,
- falę prostokątna,
- monitorowanie wyjść.

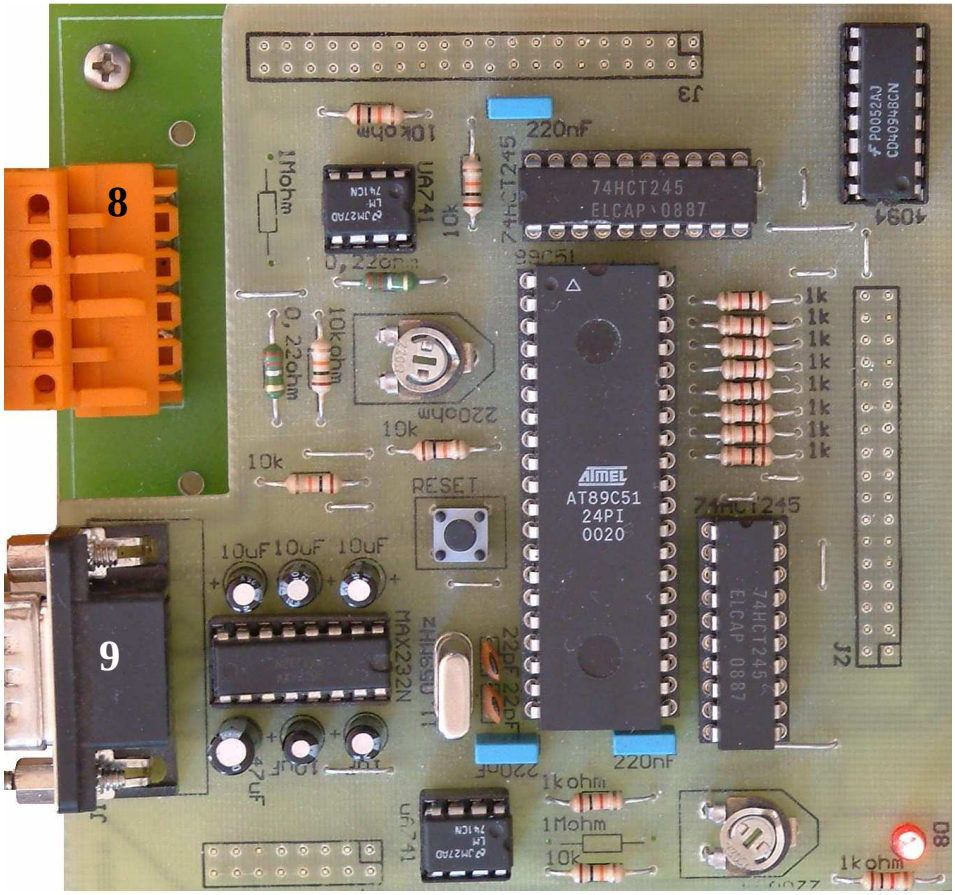


Rys. 2.3. Pole sygnałów związanych ze sterownikiem.

Mikroprocesorowy sterownik systemu (7) stanowi rozwiązanie na bazie mikrokontrolera AT89C51. Tworzą go dwa elementy: elektroniczny sterow-

nik makiety układów cyfrowych oraz rezydujące w komputerze PC oprogramowanie. Oprogramowanie praktycznie steruje całą pracą elektronicznego sterownika.

Sterownik systemu, przedstawiony na rysunku 2.4, umożliwia programową realizację podstawowych funkcji niezbędnych do badania modelowanych struktur cyfrowych.



Rys. 2.4. Mikroprocesorowy sterownik systemu.

Sterownik sprzęgany jest z makietą układów cyfrowych za pomocą trzech złącz – J1, J2 i J3.

Złącze zasilające (8) doprowadza napięcie zasilające do makiety oraz mikroprocesorowego sterownika systemu z zasilacza modułu układów cyfrowych.

Łącze szeregowe RS – 232 (9) odpowiada za współpracę PC z interfejsem sterownika.

2.1.2. Tryby pracy mikroprocesorowego sterownika systemu

Tryb ręczny

Ustalona wartość wektora pojawia się na wyjściach dopiero po zaakceptowaniu i obowiązuje do następnej akceptacji (jedna operacja).

* Wektor 10-bitowy kodu „1 z10”:
  - przesuwanie „1” - aktywny poziom wysoki [HIGH];
  - przesuwanie „0” - aktywny poziom niski [LOW].
* Wektor bitowy o zmiennej długości – swobodne ustalenie wartości wektora przez ustawienie pojedynczego bitu $\in \{0,1\}$. Długość wektora z zakresu od 2 do 8 bitów.
* Wektor 5-bitowy licznikowy:
  - opcja zliczanie w przód w NKB (inkrementacja);
  - opcja zliczania w tył w NKB (dekrementacja).

Tryb automatyczny

Zmiana wartości wektora i jego akceptacja dokonywane są automatycznie (przez program) po upływie czasu t, spójny zakres zmian wartości wektora.

Wybór tej opcji powoduje automatyczne cykliczne generowanie wartości wektora i jego akceptację po czasie t. Czas t jest wybieralnym parametrem w menu trybu pracy, $t \in \{1s, 2s, 4s\}$.

* Wektor 10-bitowy kodu „1 z10”:
  - przesuwanie „1” - aktywny poziom wysoki [HIGH];
  - przesuwanie „0” - aktywny poziom niski [LOW].
* Wektor 5-bitowy licznikowy:
  - opcja zliczanie w przód w NKB (inkrementacja);
  - opcja zliczania w tył w NKB (dekrementacja).

Generator pojedynczych impulsów.

* generator zadanej liczby impulsów:

- generuje zaprogramowaną liczbę impulsów z zakresu od 2 do 255 na jednym programowo wyróżnionym wyjściu, z wybraną częstotliwością *f*. Na wyjściu drugim - stały, ustawiany programowo, poziom napięcia (LOW lub HIGH);
- częstotliwość *f* wybierana programowo o wartościach 1Hz, 1kHz.

* generator pojedynczych impulsów:
  - dwa funkcjonalnie zróżnicowane wyjścia: jedno impulsowe, drugie stałopoziomowe (programowo wybierany poziom LOW lub HIGH);
  - programowe ustalenie czasu t trwania impulsu na wyjściu impulsowym: t∈ {50μs; 100μs; 500μs; 1ms; 50ms; 100ms; 0,5s; 4s};
  - przełączanie (zmiana funkcji) wyjść za pomocą jednej operacji;
  - generowanie impulsów według programowych ustaleń (jedna operacja odpowiada jednemu impulsowi na wyjściu);
  - licznik liczby wygenerowanych impulsów na ekranie monitora (z możliwością zerowania).
* generator fali prostokątnej:
  - częstotliwość *f* wybierana programowo o wartościach 1Hz, 1kHz, 10kHz;
  - częstotliwość na wyjściach A, B, C, D odpowiednio równa:

$$f_A = \frac{1}{2} f \; ; \; f_B = \frac{1}{4} f \; ; \; f_C = \frac{1}{8} f \; ; \; f_D = \frac{1}{16} f \; .$$

Monitorowanie wyjść badanego układu

- wizualizacja na monitorze PC stanów logicznych wybranych wyjść badanego układu cyfrowego (dołączonych do końcówek 7÷0) w dwóch opcjach:
  - w kodzie NKB;
  - w kodzie szesnastkowym (HEX).

Zakłada się, że na wszystkich końcówkach (7÷0) istnieją stabilne stany (LOW lub HIGH),

Funkcje realizowane przez komputer PC

* Tryb ręczny:
  - „1 z 10” – wysyłanie i odbieranie po każdym naciśnięciu przycisku,
  - „Wektor bitowy” – wysłanie i odbieranie po każdym naciśnięciu przycisku,
  - „5-bitowy licznik” – wysłanie i odbieranie po każdym naciśnięciu przycisku.
* Tryb automatyczny:
  - „1 z 10” – po naciśnięciu przycisku uruchamiany jest zegar, który realizuje wysyłanie komend i odbieranie w określonych odstępach czasu,
  - „5-bitowy licznik” – wysyłanie i odbieranie realizowane przez zegar.

Funkcje realizowane przez komputer wysyłają, określony dla danej funkcji pakiet komend, który jest odpowiednio interpretowany przez mikrokontroler i odbierają sygnały zwrotne po upływie określonego czasu.

Funkcje realizowane przez interfejs modułu

Funkcje realizowane przez interfejs modułu, zawierają się w trzecim trybie pracy.

Stanowią je Generatory:

- „Generator pojedynczych impulsów” – wysyłanie i odbieranie po każdym naciśnięciu przycisku,
- „Generator zadanej liczby impulsów” – odbiera komendy wysłane przez komputer i uruchamia generator,
- „Generator fali prostokątnej” – komendy otrzymane przez komputer, uruchamiają generator fali prostokątnej.

Realizowane jest również wysyłanie sygnałów zwrotnych. Zależnie od częstotliwości, sygnał jest wysyłany po każdym wygenerowanym impulsie (częstotliwość 1 Hz) lub po zatrzymaniu procesu generacyjnego (częstotliwości powyżej 1 Hz).

2.1.3. Zasilacz

Dla układów TTL standardowym napięciem zasilania jest + 5V. Maksymalna odchyłka napięcia zasilania wynosi ±0,25V. Pobierane jest z modułu zasilacza μM-UNI wyposażonego w stabilizator napięcia +5V.

2.1.4. Inne elementy

Wszystkie połączenia zamodelowanego układu cyfrowego realizowane są za pomocą giętkich przewodów o zróżnicowanej długości, dołączonych w odpowiedniej liczbie do każdego stanowiska laboratoryjnego. Do modelowania układów cyfrowych służy zestaw elementów i bloków funkcyjnych w postaci układów scalonych klasy TTL.

Dostępna jest również sonda logiczna TTL i CMOS typu Instek GLP-1A (rysunek 2.5). Umożliwia ona sprawdzanie stanów logicznych badanych układów cyfrowych.

Rys. 2.5.Sonda logiczna Instek GLP-1A.

2.2. Program Makieta laboratoryjna 1.0

Program *Makieta laboratoryjna 1.0*© jest integralną częścią modułu układów cyfrowych. Został napisany w ramach pracy inżynierskiej przez Bartłomieja Denisa. Wszelkie prawa do programu posiada Kolegium Karkonoskie.

2.2.1. Interfejs użytkownika

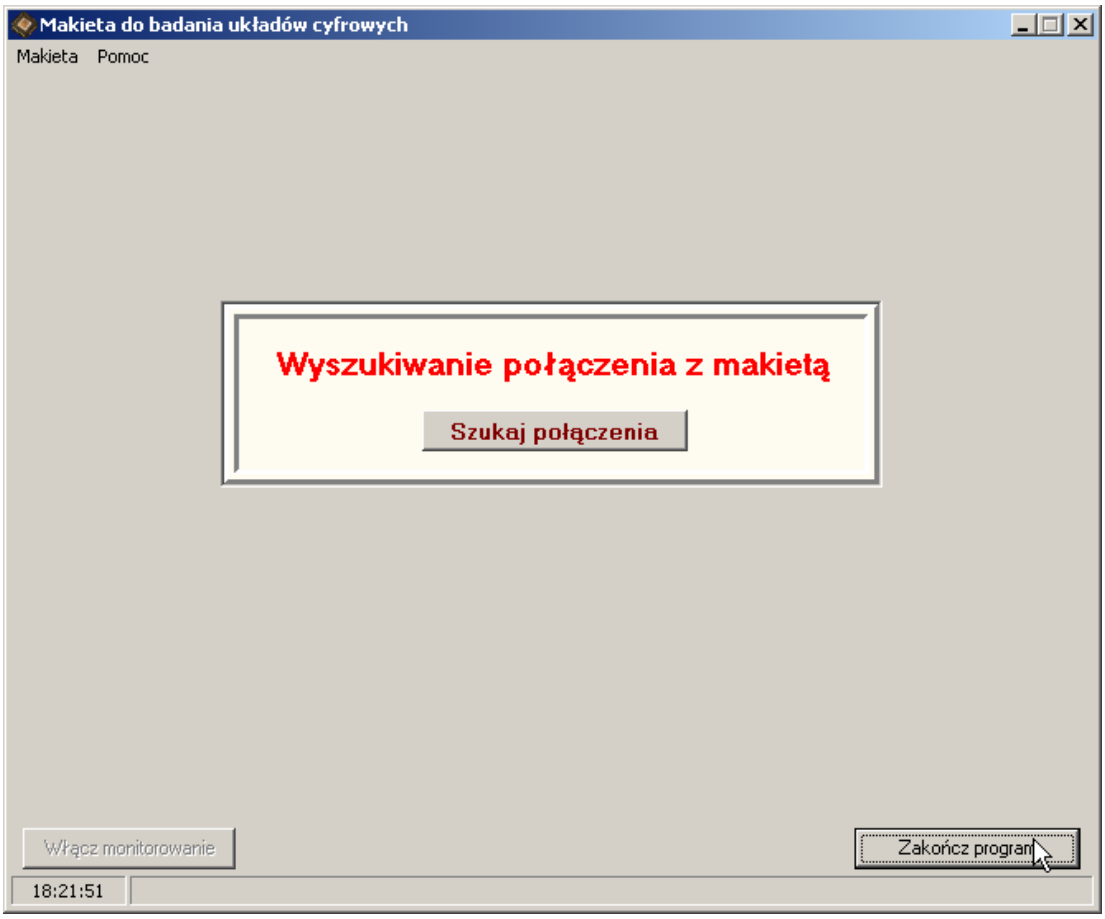
Interfejs użytkownika składa się z trzech zakładek, z których każda reprezentuje określony tryb pracy, a w związku z tym ściśle określone funkcje. Każdy tryb charakteryzuje się odpowiednią dla niego funkcjonalnością, gdzie każde naciśnięcie przycisku powoduje wykonanie pojedynczej akcji lub uruchomienie procedury generacyjnej. Dodatkowo, istnieją różnorodne zabezpieczenia w każdym z trybów, spełniające zarówno funkcje eliminacji możliwości wystąpienia błędów, jak również funkcje czysto informacyjne.

Dwukrotne kliknięcie ikony *Makieta.exe*, 

znajdującej się na pulpicie monitora, spowoduje pojawienie się okna „*Makieta do badania układów cyfrowych*”, przedstawionego na rysunku 2.6.

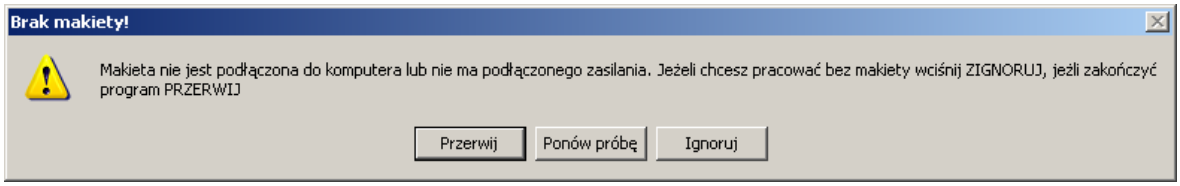
Program posiada górne menu. Zakładka *Makieta* zawiera opcję ponownego wyszukania połączenia z makietą, zaś zakładka *Pomoc* zawiera informacje typu proceduralnego oraz informację o programie.

Wciśnięcie przycisku „*Szukaj połączenia*” rozpoczyna procedurę wyszukiwania połączenia z modułem układów cyfrowych. Operacja przebiega automatycznie bez ingerencji użytkownika.



Rys. 2.6.Okno główne programu *Makieta laboratoryjna*.

W przypadku braku połączenia z modułem zostanie wygenerowany komunikat (rysunek 2.7):

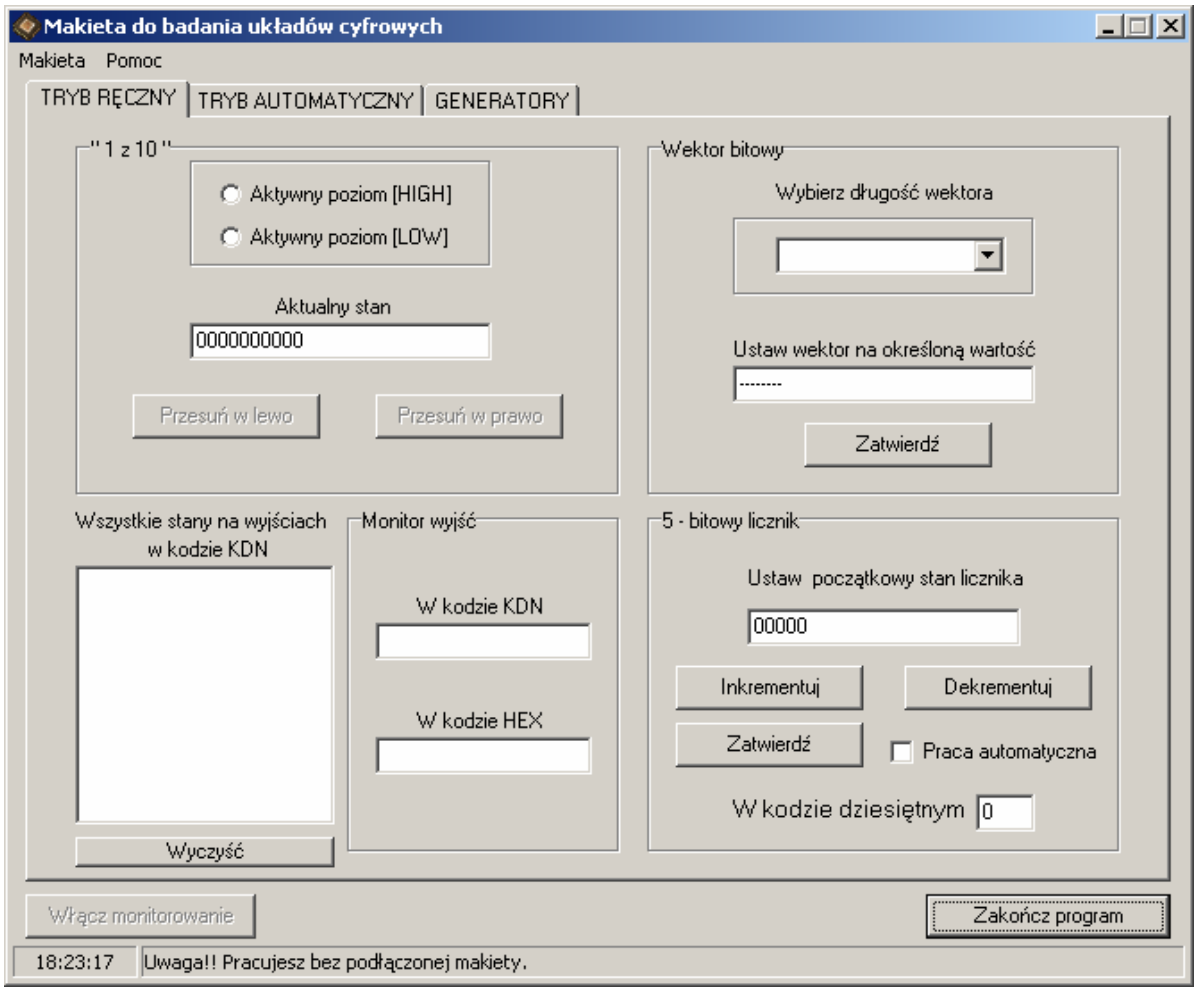


Rys. 2.7. Komunikat braku połączenia z modułem.

Wciśnięcie przycisku *„Przerwij”* zakończy działanie programu. Naciśnięciu przycisku *„Ponów próbę”* spowoduje ponowne uruchomienie procedury wyszukiwania modułu. Wciśnięcie przycisku *„Ignoruj”* pozwala na pracę bez podłączonego modułu, umożliwiając zapoznania się z interfejsem programu.

Gdy program „znajdzie” połączenie z mikroprocesorowym sterownikiem systemu, pojawia się użytkownikowi zestaw trybów pracy z całą gamą funkcji. Jako pierwsza zakładka pojawia się TRYB RĘCZNY (rysunek 2.8), gdzie

każde naciśnięcie odpowiedniego przycisku wysyła sygnał do sterownika modułu. Podzielona jest na kilka sekcji, z których każda realizuje ściśle określoną funkcję. Zestaw przycisków jest jednoznacznie przypisany do odpowiedniej funkcji, zaś ich opisy informują o wykonywanych akcjach.



Rys. 2.8. Zakładka TRYB RĘCZNY.

Tryb „1 z 10” pozwala na przesuwanie stanu aktywnego w prawą bądź lewą stronę. Określenie rodzaju stanu aktywnego pozostawione jest użytkownikowi. Poniżej znajdują się okna monitorów, czyli pola tekstowe, w których rejestrowane są sygnały wyjściowe z modułu, reprezentowane w kodzie binarnym oraz szesnastkowym. Możliwość włączenia lub wyłączenia monitorowania, realizowane jest za pomocą przycisku „Włącz monitorowanie”. Przy uruchomieniu programu monitorowanie jest domyślnie ustawione w trybie wyłączonym.

Po prawej stronie mamy do dyspozycji dwa funkcjonalnie zróżnicowane obszary:

- ***„Wektor bitowy***”
- ***„5-bitowy licznik”***

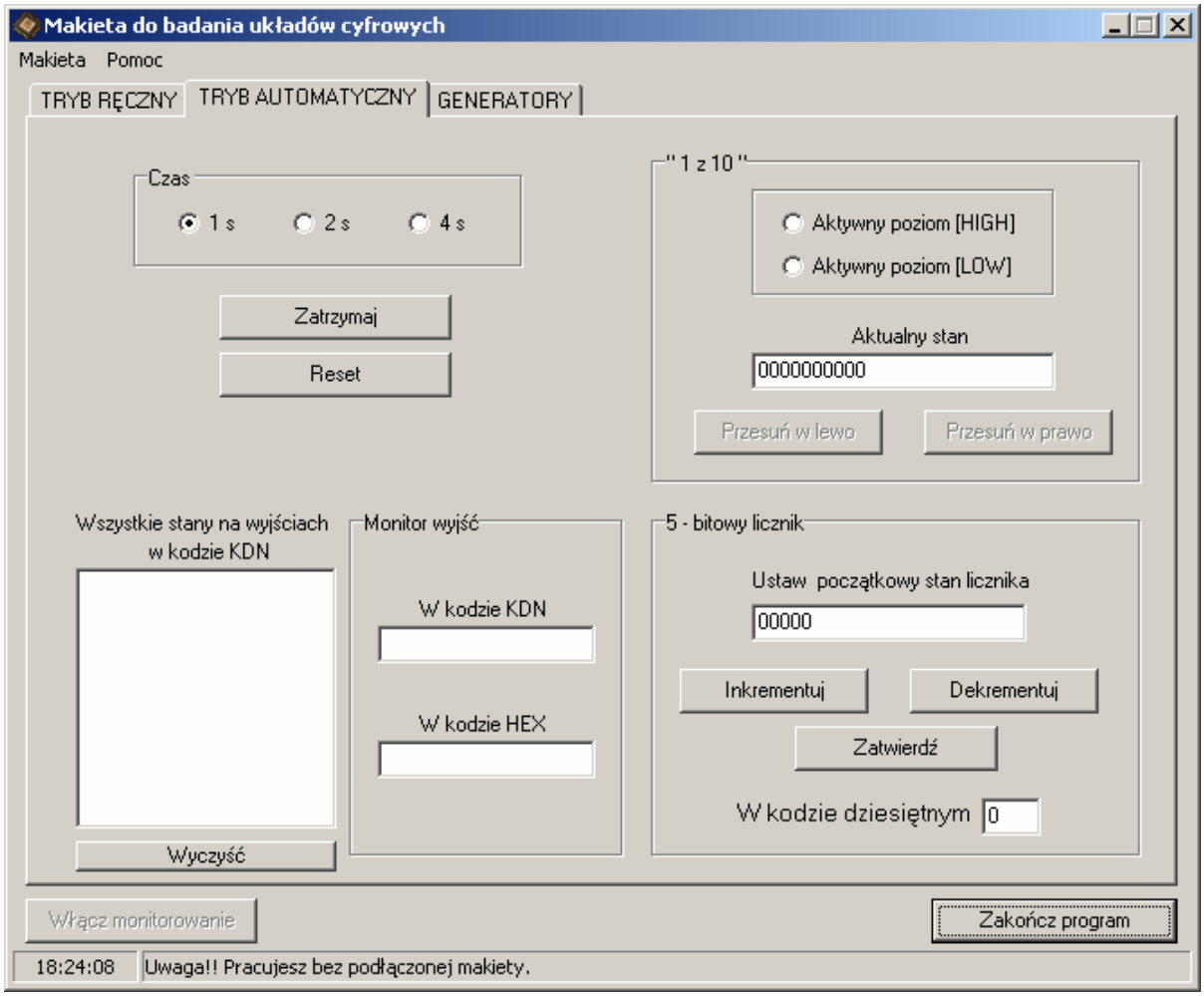
„Wektor bitowy” pozwala na wygenerowanie słowa bitowego, którego wybrana powyżej długość zmusza użytkownika do ustawienia poszczególnych bitów jako „1” lub „0”. Naciśnięcie przycisku *„Zatwierdź”* wysyła wektor do modułu, na wejścia badanego układu.

„5-bitowy licznik, umożliwia ustawienie stanu początkowego, a poprzez wciśnięcie przycisku *„Zatwierdź*” wysłania bezpośrednio do sterownika modułu. Aby ułatwić wysyłanie kolejnych stanów licznika mamy do dyspozycji przyciski inkrementacji lub dekrementacji stanów. Możemy po każdej zmianie skorzystać z przycisku zatwierdzającego lub użyć opcji pracy automatycznej. Włączając automatyczną pracę licznika każdy kolejny stan jest od razu wysyłany bez konieczności każdorazowego zatwierdzania.

Kolejna zakładka to TRYB AUTOMATYCZNY (rysunek 2.9). Składa się z dwóch, podobnych jak w trybie ręcznym sekcji: *„1 z 10”* oraz *„5-bitowy licznik”*.

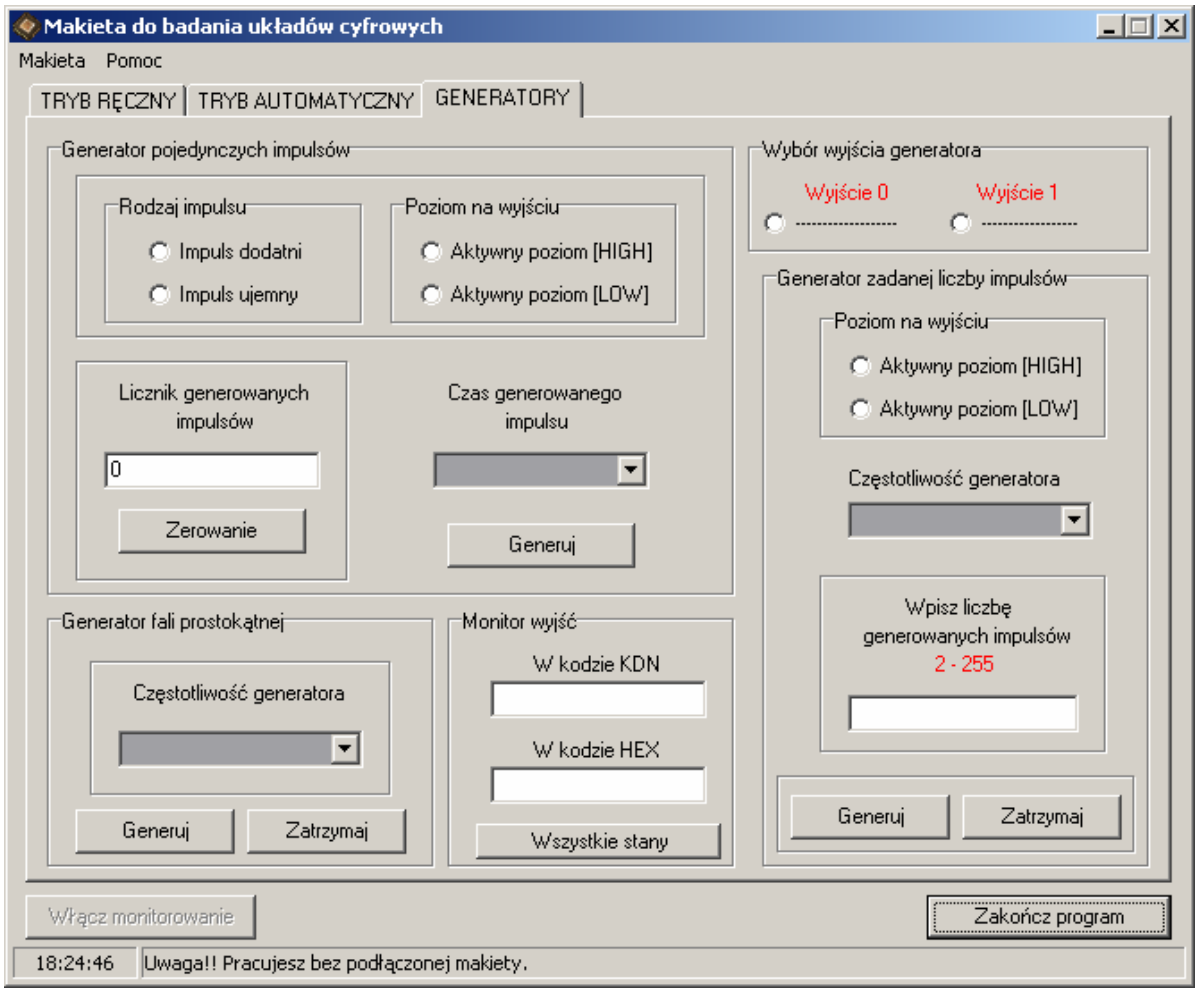
Zasada działania automatycznego trybu pracy oparta jest na automatycznej procedurze generacji określonych stanów w określonych odstępach czasu. Funkcja „1 z 10” działa podobnie jak w trybie ręcznym, jednak tutaj naciśnięcie któregokolwiek z przycisków uruchamia automat generacyjny. Do dyspozycji są trzy przedziały czasu: 1, 2 lub 4 sekundy. Wysyłanie sygnałów do sterownika następuje każdorazowo, co konkretny przedział czasu, do momentu zatrzymania. Użytkownik ma możliwość przerwania procedury w każdym momencie jej działania. W obszarze *„5-bitowy licznik”* znajduje się dodatkowy przycisk *„Zatwierdź”*, który powoduje zatwierdzenie ustawionego początkowego stanu licznika. Oczywiście jak we wszystkich trybach pracy do

dyspozycji mamy monitor, na podstawie którego możemy śledzić zmiany występujące w badanym układzie.



Rys. 2.9. Zakładka TRYB AUTOMATYCZNY.

Najważniejszym i najbardziej rozbudowanym trybem pracy są GENERATORY (rysunek 2.10). Tryb ten stanowi grupę różnych generatorów, z których każdy jest nieodzownym elementem badań układów cyfrowych. Interfejs na pierwszy rzut oka, może wydawać się niezwykle skomplikowany, jednak po bardziej wnikliwym przyjrzeniu się dostępnym opcjom, staje się czytelny i przejrzysty.



Rys. 2.10. Zakładka GENERATORY pracy oprogramowania.

Największą ilość ustawianych parametrów daje nam generator pojedynczych impulsów. Oprócz podstawowego wyboru wyjścia generatora, dokonujemy również wyboru rodzaju impulsu, stanu na wyjściu stałopoziomowym oraz czasu generowanego impulsu. Każde naciśnięcie przycisku *„Generuj”* powoduje generację pojedynczego impulsu, który jednocześnie jest zliczany przez licznik umieszczony z lewej strony zakładki. Ilość generowanych impulsów nie jest ograniczona.

Po prawej stronie zakładki znajduje się generator zadanej liczby impulsów. W przeciwieństwie do generatora pojedynczych impulsów, naciśnięcie przycisku *„Generuj”* uruchamia procedurę generacyjną określonej liczby impulsów. Proces trwa do momentu wygenerowania zadanej liczby impulsów lub wcześniejszego przerwania przez użytkownika. Funkcja ta umożliwia do-

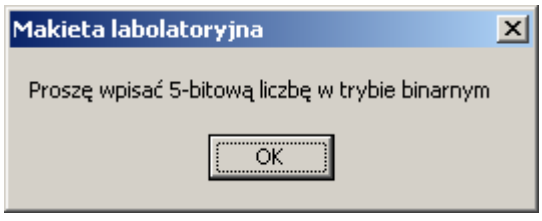
konanie wyboru wyjścia generatora, stanu na wyjściu stałopoziomowym oraz wartości częstotliwości. Generator pozwala na ustawienie ilości impulsów w przedziale od 2 do 255.

Ostatnią funkcją w tym trybie pracy jest generator fali prostokątnej. W swoich opcjonalnych możliwościach pozwala na ustawienie częstotliwości. Uruchomiony, działa do momentu przerwania przez użytkownika lub zakończenia programu. Dostępny jest również monitor, ale w odróżnieniu od poprzednich trybów pracy monitor zbiorowy pojawia się po naciśnięciu przycisku *„Wszystkie stany”*. Działanie tego trybu pracy, poza generatorem pojedynczych impulsów, polega na wysłaniu odpowiedniej komendy do mikrokontrolera, który uruchamia odpowiadający jej generator.

2.2.2. Komunikaty występujące w programie

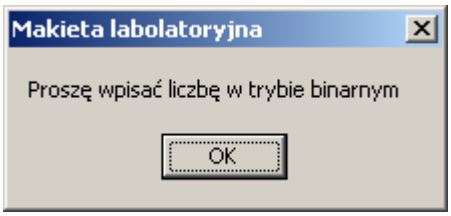
W programie „*Makieta laboratoryjna*” występują zarówno komunikaty informujące o wystąpieniu błędu, jak również komunikaty wymagające od użytkownika podjęcia określonej decyzji.

Wszystkie wpisywane przez użytkownika dane muszą być w reprezentacji binarnej, stąd każda próba zatwierdzenia nieprawidłowo wpisanych danych jest odrzucana. Wizualizacja tego rodzaju przypadków realizowana jest w postaci dwóch komunikatów. Pierwszy z nich (rysunek 2.11) występuje wówczas, gdy wpisane słowo jest niewłaściwej długości (stosowane przy funkcji „wektor bitowy”).



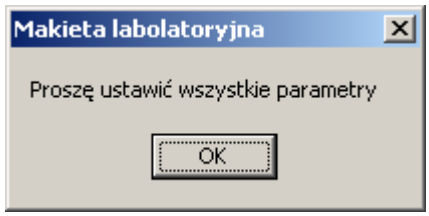
Rys. 2.11. Niewłaściwa długość słowa.

Dodatkowo informuje użytkownika, jaka jest prawidłowa długość słowa. Natomiast, komunikat drugi (rysunek 2.12) przypomina użytkownikowi, iż wpisywana liczba musi być w systemie binarnym.



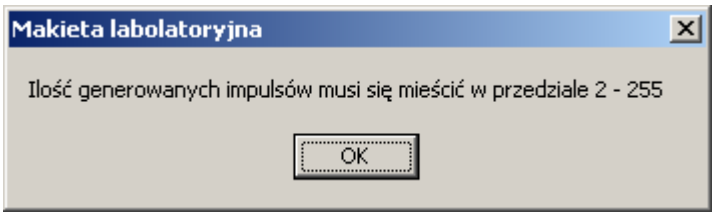
Rys. 2.12. Konieczność wpisywania danych w systemie binarnym.

Działanie generatorów polega na wygenerowaniu określonej, zależnie od rodzaju generatora, liczby impulsów o ustalonych parametrach. Szczególne znaczenie ma ustawienie wszystkich parametrów, gdyż brak jakiegokolwiek spowodowałoby nieprawidłową pracę a nawet brak reakcji sterownika makiety. Działanie generatorów realizowane jest przez mikrokontroler, umieszczony w sterowniku makiety, a więc nieotrzymanie dokładnej listy komend nie uruchomi żadnej funkcji generacyjnej. Dodatkowo brak ustawienia jakiejkolwiek z opcji powoduje wygenerowanie komunikatu informacyjnego (rysunek 2.13), zmuszającego użytkownika do ustawienia wszystkich parametrów pracy określonego generatora.



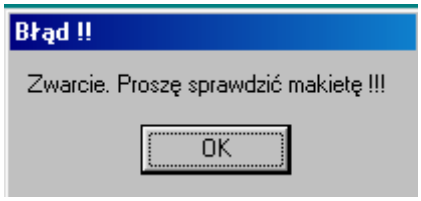
Rys. 2.13. Ustawienie wszystkich parametrów.

W generatorze zadanej liczby impulsów, poza informacją o parametrach pojawia się jeszcze jeden komunikat. Ze względu na przedział liczbowy, w jakim musi się zawierać ilość generowanych impulsów, należało zablokować możliwość wpisania liczby z poza przedziału. Jeżeli wpiszemy liczbę mniejszą od 2 lub większą od 255 wygenerowany zostanie komunikat monitujący użytkownika do zmiany liczby (rysunek 2.14).



Rys. 2.14. Liczba generowanych impulsów.

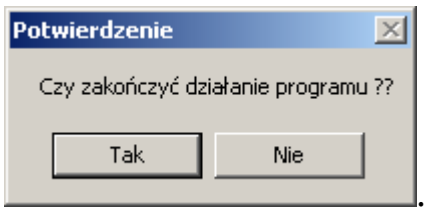
Najważniejszym komunikatem, stanowiącym integralną część systemu zwarciowego sterownika modułu jest wizualizacja wystąpienia zwarcia. W momencie, kiedy podczas podłączania układu cyfrowego, na którym będziemy przeprowadzać badania, wystąpi zwarcie, na monitorze pojawi się komunikat (rysunek 2.15).



Rys. 2.15. Zwarcie na module.

Komunikat jednocześnie zmusza użytkownika do sprawdzenia połączeń i wyeliminowania zwarcia. Będzie do tej pory się pojawiał dopóki zwarcie nie zostanie wyeliminowane. Rozwiązanie niejednokrotnie może ustrzec użytkownika przed uszkodzeniem badanego układu a nawet modułu.

Gdy skończyliśmy badania i chcemy zakończyć pracę programu, po naciśnięciu przycisku „*Zakończ program*” pojawi się jeszcze jeden komunikat (rysunek 2.16), proszący o potwierdzenie decyzji o zakończeniu pracy.



Rys. 2.16. Zakończenie pracy programu.

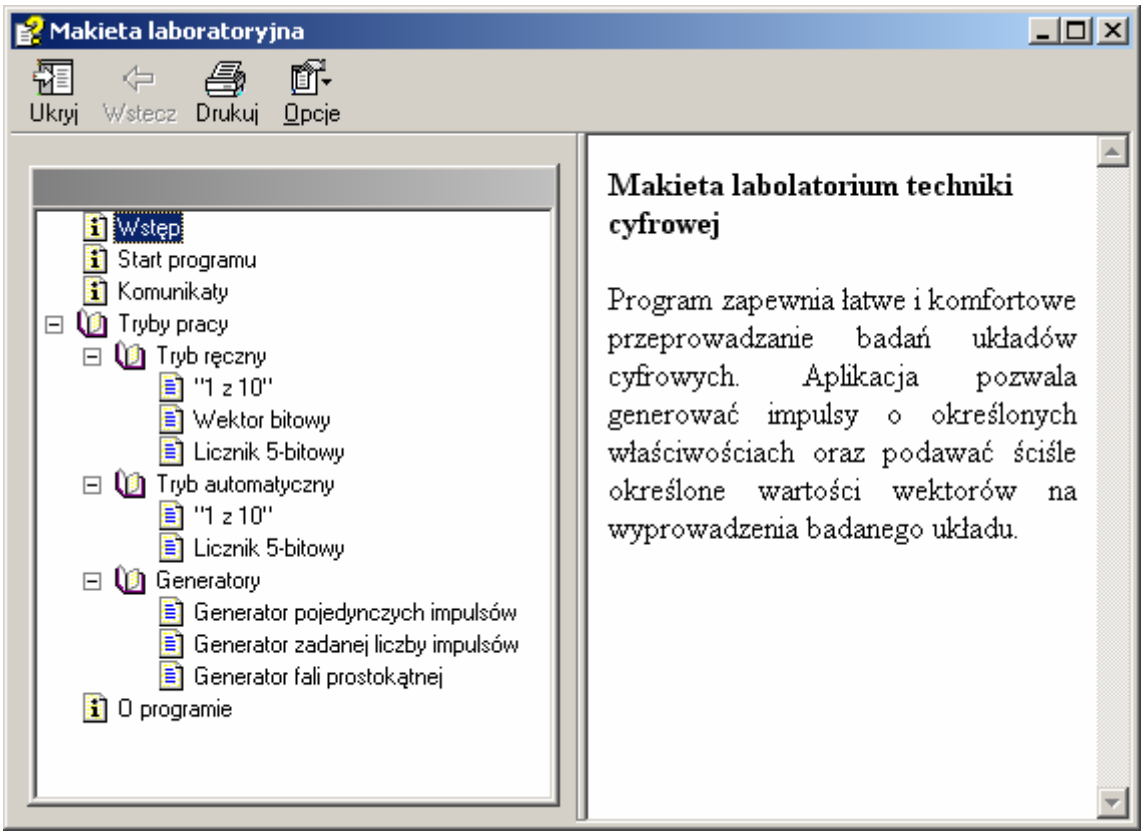
2.2.3. Pomoc programu

System pomocy programu *„Makieta laboratoryjna*” jest typu proceduralnego. Zakładka *Pomoc*, w górnym menu okna programu, umożliwia dostęp do dokumentacji tekstowej popartej ilustracjami umożliwiającej odwołanie się do programu (rysunek 2.17). Opisane są krok po kroku czynności, jakie musi wykonać użytkownik, aby wykonać konkretne zadanie, np. wygenerować 20 impulsów o konkretnych parametrach za pomocą generatora zadanej liczby impulsów.

Jest podzielona na trzy grupy, odpowiadające podziałowi interfejsu, czyli tryb ręczny, tryb automatyczny oraz generatory. Ponadto znajdują się informacje o instalacji oraz procesie uruchamiania oprogramowania.

Ze względu na zawartość i przeznaczenie system pomocy można podzielić na kilka kategorii, spełniających zapotrzebowanie na różnego rodzaju informacje. Są to:

* pomoc proceduralna - opisuje metody wykonywania konkretnych zadań,
* pomoc koncepcyjna - wyjaśnia mechanizmy działania poszczególnych trybów pracy i leżące u ich podstaw zasady,
* pomoc instruktażowa - zawiera informacje, spełniające funkcje nauczania użytkownika metod korzystania z programu.



Rys. 2.17. Okno „Pomocy”.

3. ĆWICZENIA LABORATORYJNE

3.1. Cele kształcenia

Zajęcia laboratoryjne mają na celu praktyczną weryfikację, ugruntowanie i poszerzenie nabytej wiedzy teoretycznej oraz kształtowanie umiejętności praktycznych łączenia układów pomiarowych, poprawnego odczytywania wyników i prowadzenia odpowiedniej dokumentacji pomiarów.

Kształtowanie umiejętności metrologicznych, takich jak: prawidłowe zaplanowanie pomiarów celem przeprowadzenia doświadczenia, poprawna organizacja stanowiska pomiarowego i właściwa eksploatacja aparatury oraz ocena i dyskusja wyników pod względem ich wiarygodności.

Zapoznanie studentów z układami cyfrowymi klasy TTLS, stosowanymi strukturami układów cyfrowych, a także metodyką projektowania urządzeń cyfrowych na poziomie stosowania bloków funkcjonalnych.

Oczekuje się, że uczestnictwo w zajęciach pozwoli studentom na wykonywanie samodzielnych projektów z użyciem cyfrowych układów oraz interpretację i dyskusję otrzymanych wyników w trakcie badań.

3.2. Charakterystyka ćwiczeń

Treści programowe

Określenie właściwości logicznych badanych monolitycznych cyfrowych układów scalonych realizowanych w klasie TTLS, poznanie możliwości realizacji dowolnych wyrażeń logicznych na podstawie wybranych elementów scalonych małej skali integracji.

Zagadnienie praktycznej realizacji zadanej funkcji logicznej rozważa się w dwóch aspektach:

- z punktu widzenia przypadkowego zestawu wolnych funktorów,
- z punktu widzenia doboru właściwych funktorów, aby otrzymany układ zawierał minimalną liczbę elementów scalonych.

3.2.1. Wprowadzenie do ćwiczeń laboratoryjnych, wymagania, literatura, zasady BHP

Omówienie spraw organizacyjnych związanych z wykonaniem i zaliczeniem ćwiczeń laboratoryjnych: podział na grupy i podgrupy, zapoznanie z *Regulaminem Laboratorium Techniki Cyfrowej* oraz *Przepisami BHP obowiązującymi w laboratorium*, przedstawienie tematyki i zasad zaliczenia ćwiczeń laboratoryjnych, zapoznanie z literaturą podstawową i uzupełniającą.

3.2.2. Zapoznanie się z makietą laboratoryjną, możliwościami, właściwościami oraz oprogramowaniem sterownika

Celem ćwiczenia jest poznanie środowiska projektowego:

- makiety laboratoryjnej ML-1;
- programu „*Makieta laboratoryjna 1.0*.”

Program ćwiczenia obejmuje:

- Poznanie budowy makiety laboratoryjnej ML-1 oraz możliwości modelowania układów logicznych z jej wykorzystaniem.
- Poznanie programu *Makieta laboratoryjna 1.0*.
- Zaprojektowanie i sprawdzenie poprawności działania trójwejściowych elementów logicznych zrealizowanych z elementów dwuwejściowych.
- Zrealizowanie funkcji logicznych z bramek NAND i NOR.

Przygotowanie do ćwiczeń.

Studenci winni zapoznać się z instrukcją laboratoryjną „Laboratorium Techniki Cyfrowej”, ze szczególnym zwróceniem uwagi na rozdział 2 - Ma-

kieta laboratoryjna ML-1. Wymagana jest znajomość podstaw matematycznych układów przełączających: algebra Boole'a, aksjomaty, definicje i twierdzenia, podstawowe funkcje logiczne - suma, iloczyn, negacja, suma zanegowana, iloczyn zanegowany, suma modulo 2, postać kanoniczna sumy i iloczynu dla funkcji logicznej.

3.2.3. Podstawowe elementy logiczne rodziny 74LS

Celem ćwiczenia jest zapoznanie się z:

- strukturą i właściwościami logicznymi podstawowych elementów logicznych badanych monolitycznych układów scalonych: bramek i przerzutników;
- problemami związanymi z syntezą kombinacyjnych układów logicznych opartyą na zadanym zbiorze bramek logicznych;
- wyrobienie umiejętności budowy układów kombinacyjnych i sekwencyjnych z podstawowych elementów kombinacyjnych na podstawie opisu słownego, tablicowego lub numerycznego.

Program ćwiczenia obejmuje:

- Sprawdzenie poprawności działania wskazanych bramek.
- Zaprojektowanie i sprawdzenie prostych cyfrowych sieci kombinacyjnych.
- Sprawdzenie poprawności działania przerzutników.
- Transformację różnych typów przerzutników.
- Syntezę jednowyjściowych układów kombinacyjnych oraz sposób ich realizacji.

Badane układy scalone:

* 74LS00 74LS02 74LS04 74LS08
* 74LS10 74LS20 74LS32 74LS38
* 74LS86 74LS74 74LS107 74LS244

Przygotowanie do ćwiczeń.

Każda grupa otrzymuje indywidualne zadania projektowe. Treść i formę prezentacji zadań określa prowadzący zajęcia. Do każdego zadania studenci przygotowują rozwiązania w postaci wyrażeń logicznych i schematu ideowego układu, zaprojektowanego z dostępnych w zestawie elementów cyfrowych. Rozwiązania winny być przedstawione w protokole, który jest podstawą wykonania ćwiczenia. Wymagana jest znajomość podstawowych pojęć i tożsamości algebry Boole'a, praw De Morgana, metod syntezy funkcji logicznych jednowyjściowych na elementach małej skali integracji, rodzaje przerzutników, realizacja przerzutnika RS w oparciu o bramki NAND lub NOR, pojęcie przerzutnika synchronicznego i asynchronicznego, konwersja przerzutników synchronicznych.

 Zagadnienia do opracowania:

1. Zdefiniować podstawowe funkcje logiczne.
2. Omówić sposoby przedstawiania funkcji logicznych.
3. Przedstawić postać sumacyjną i iloczynową wyrażeń logicznych tej samej funkcji przełączającej.
4. Przedstawić konstrukcję mapy Karnaugha i wykorzystanie jej do minimalizacji funkcji.
5. Zdefiniować podstawowe parametry statyczne układów cyfrowych i podać ich typowe wartości dla układów TTLS.
6. Podać klasyfikację przerzutników.
7. Podać równania funkcyjne przerzutników: SR, JK, D, T.
8. Omówić budowę i działanie przerzutnika wyzwalanego impulsem JK-MS.
9. Podać schematy funkcjonalne konwersji przerzutnika T w przerzutniki JK i D.
10. Wyjaśnić pojęcie dwójki liczącej oraz podać przykłady realizacji z wykorzystaniem przerzutników.

3.2.4. Układy konwersji kodów

Celem ćwiczenia jest zapoznanie się z:

- projektowaniem prostych układów logicznych realizujących funkcje koderów, dekoderów i transkoderów z wykorzystaniem wybranych elementów scalonych małej skali integracji;
- budową i zasadą działania układów konwersji kodów realizowanych z bramek logicznych lub wykonanych w postaci monolitycznych układów scalonych;
- sposobami realizacji układów konwersji kodów techniką TTL i metodami ich badania.

Program ćwiczenia obejmuje:

- Zaprojektowanie z bramek i sprawdzenie poprawności działania wskazanych układów:
  - kodera zwykłego lub priorytetowego;
  - dekodera kodu BCD na kod „1 z n”;
  - transkodera zadanego kodu wejściowego na wyróżniony kod wyjściowy.
- Zapoznanie się z budową i badanie scalonego enkodera priorytetowego 74LS148.
- Zapoznanie się z budową i badanie scalonych dekoderów 74LS138 i 74LS145.
- Zapoznanie się z budową i zasadą działania dekodera kodu BCD na kod wskaźnika siedmiosegmentowego 74LS47.

Badane układy scalone:

* 74LS47 74LS138 74LS145 74LS148

Do realizacji układów konwersji kodów z układów scalonych małej i średniej skali integracji można zastosować inny zestaw laboratoryjny, po wcześniejszym uzgodnieniu z prowadzącym zajęcia.

Przygotowanie do ćwiczeń.

Każda grupa otrzymuje indywidualne zadania projektowe. Treść i formę prezentacji zadań określa prowadzący zajęcia. Do każdego zadania studenci przygotowują rozwiązania w postaci wyrażeń logicznych i schematu ideowego układu zaprojektowanego z dostępnych w zestawie elementów cyfrowych. Rozwiązania winny być przedstawione w protokole, który jest podstawą wykonania sprawozdania. Wymagana jest znajomość podstawowych rodzajów kodów binarnych: NKB, BCD, 1 z n, Greya, metod syntezy funkcji logicznych wielowyjściowych na elementach małej i średniej skali integracji.

 Zagadnienia do opracowania:

1. Omówić zasady konwersji liczb dziesiętnych na liczby binarne i odwrotnie.
2. Omówić kody naturalne i kody BCD.
3. Zdefiniować i dokonać podziału konwerterów kodów.
4. Omówić budowę i zasadę działania iteracyjnego enkodera priorytetowego.
5. Omówić budowę i właściwości enkodera 74LS148.
6. Omówić budowę i przeznaczenie dekodera.
7. Omówić budowę i właściwości dekoderów scalonych 74LS138 i 74LS145.
8. Przedstawić zasady syntezy transkoderów.
9. Przedstawić zasady konwersji liczb binarnych na liczby zapisane w kodzie Graya.
10. Omówić przeznaczenie wejść funkcyjnych układu scalonego 74LS47.

3.2.5. Układy komutacyjne

Celem ćwiczenia jest:

- poznanie budowy, zasady działania i zastosowań scalonych układów multiplekserów i demultiplekserów;

- praktyczna realizacja i badanie zadanych układów komutacyjnych;
- praktyczna realizacja i badanie generatorów funkcji logicznych budowanych z bloków komutacyjnych.

Program ćwiczenia obejmuje:

- Zaprojektowanie z podstawowych bramek logicznych multipleksera czterobitowego i sprawdzenie poprawności jego działania.
- Sprawdzenie działania logicznego scalonych układów 74LS151, 74LS153 i 74LS155.
- Zastosowanie multiplekserów i demultiplekserów do realizacji zadanych funkcji boolowskich.
- Zaprojektowanie i sprawdzenie układu generatora impulsów zegarowych:
  - o programowanym współczynniku wypełnienia,
  - wielofazowego (2 lub więcej faz), realizowanego za pomocą bramek, przerzutnika i multipleksera.
- Symulację 4 kanałowego przesyłania informacji jedną linią od nadajnika do odbiornika.

Badane układy scalone:

* 74LS153 74LS155

Do realizacji układów komutacyjnych z układów scalonych małej i średniej skali integracji można zastosować inny zestaw laboratoryjny, po wcześniejszym uzgodnieniu z prowadzącym zajęcia.

Przygotowanie do ćwiczeń.

Każda grupa otrzymuje indywidualne zadania projektowe. Treść i formę prezentacji zadań określa prowadzący zajęcia. Do każdego zadania studenci przygotowują rozwiązania w postaci wyrażeń logicznych i schematu ideowego układu, zaprojektowanego z dostępnych w zestawie elementów cyfrowych. Rozwiązania winny być przedstawione w protokole, który jest podstawą wy-

konania sprawozdania. Wymagana jest znajomość budowy, zasady działania, metod syntezy funkcji logicznych wielowyjściowych oraz sposoby ich realizacji na elementach średniej skali integracji.

 Zagadnienia do opracowania:

1. Omówić właściwości multipleksera.
2. Scharakteryzować metodę syntezy układów kombinacyjnych z użyciem multiplekserów wykorzystująca tablice Karnaugha.
3. Omówić budowę i właściwości multipleksera 74LS153.
4. Omówić budowę multipleksera piramidalnego.
5. Omówić zasady budowy multiplekserowych układów wielopoziomowych.
6. Omówić właściwości demultipleksera.
7. Podać różnicę między dekoderem a demultiplekserem.
8. Scharakteryzować metodę syntezy układów kombinacyjnych z użyciem demultiplekserów.
9. Omówić budowę i właściwości demultipleksera/dekodera 74LS155.
10. Omówić zastosowanie multipleksera i demultipleksera jako układów uniwersalnych.

3.2.6. Rejestry równoległe i przesuwające

Celem ćwiczenia jest zapoznanie się ze sposobami realizacji podstawowych rodzajów rejestrów równoległych i przesuwających, z ich budową, zasadą działania oraz możliwościami funkcjonalnymi scalonych układów rejestrów równoległych i szeregowych (przesuwnych).

Program ćwiczenia obejmuje:

- Sprawdzenie poprawności działania logicznego scalonych układów rejestrów równoległych: 74LS174 i 74LS175.
- Zaprojektowanie, realizację i sprawdzenie działania wskazanego układu rejestru przesuwnego:

– 3 bitowego rejestru rewersyjnego z przesuwem informacji;
– 6 bitowego rejestru z przesuwem informacji w prawo albo w lewo (należy zastosować bramki i układy 74LS174, 74LS175),
– licznika pierścieniowego z układem kombinacyjnym pełniącym rolę przełącznika ładowanie/zamknięcie pierścienia.

- Sprawdzenie poprawności działania logicznego scalonych układów rejestrów przesuwnych 74LS164, 74LS165 i 74LS194.

Badane układy scalone:

∗ 74LS164 74LS165 74LS174 74LS194

Przygotowanie do ćwiczeń.

Każda grupa otrzymuje indywidualne zadania projektowe. Do każdego zadania studenci przygotowują rozwiązania w postaci schematu ideowego układu, zaprojektowanego z dostępnych w zestawie elementów cyfrowych. Rozwiązania winny być przedstawione w protokole, który jest podstawą wykonania sprawozdania. Wymagana jest znajomość budowy, zasady działania, sposoby wprowadzania informacji do rejestrów i wyprowadzania informacji z rejestrów, budowy struktury wewnętrznej i zasady działania badanych układów.

Zagadnienia do opracowania:

1. Zdefiniować układ sekwencyjny.
2. Przerzutniki jako elementy pamięci w układach sekwencyjnych - opis słowny, tablicowy, za pomocą grafu i funkcji logicznej.
3. Napisać tablice wzbudzeń dla przerzutników JK, D, T.
4. Podać podstawowe rodzaje rejestrów.
5. Omówić układy równoległego wprowadzania informacji do rejestru.
6. Scharakteryzować rejestry przesuwające.
7. Podać strukturę rejestru liczącego.
8. Omówić układy odczytu równoległego informacji z rejestru.

9. Wyjaśnić zamianę informacji równoległej na szeregową i odwrotnie na przykładzie scalonych rejestrów 74LS164 oraz 74LS165.
10. Omówić budowę i właściwości rejestru uniwersalnego 74LS194.

3.2.7. Liczniki asynchroniczne

Celem ćwiczenia jest poznanie własności oraz zasad projektowania podstawowych liczników asynchronicznych modulo N, budowanych z odpowiednio ze sobą połączonych przerzutników TTLS oraz zapoznanie się z budową i zasadą działania scalonych liczników asynchronicznych dziesiętnych i binarnych.

Program ćwiczenia obejmuje:

- Zaprojektowanie i badanie poprawności działania zadanych liczników, realizowanych z bramek i przerzutników bistabilnych.
- Zapoznanie się z budową i możliwościami funkcjonalnymi asynchronicznych liczników scalonych 74LS90 i 74LS93.
- Zaprojektowanie i sprawdzenie poprawności działania liczników modulo N, realizowanych za pomocą bramek i liczników scalonych. Wartość N zadaje prowadzący zajęcia (należy tu uwzględnić liczniki o pojemności z zakresu 2÷255).

Badane układy scalone:

∗ 74LS90 74LS93

Przygotowanie do ćwiczeń.

Każda grupa otrzymuje indywidualne zadania projektowe. Do każdego zadania studenci przygotowują rozwiązania w postaci schematu ideowego układu, zaprojektowanego z dostępnych w zestawie elementów cyfrowych. Rozwiązania winny być przedstawione w protokole, który jest podstawą wykonania sprawozdania. Wymagana jest znajomość budowy, metod syntezy liczników mod. N, struktury wewnętrznej i zasady działania badanych układów.

Zagadnienia do opracowania:

1. Wymienić główne kryteria podziału i rodzaje liczników.
2. Przedstawić schemat logiczny asynchronicznego licznika liczącego w przód.
3. Przedstawić schemat logiczny asynchronicznego licznika liczącego w tył.
4. Narysować i omówić schematy funkcjonalne liczników scalonych 74LS90 i 74LS93.
5. Omówić właściwości liczników scalonych 74LS90 i 74LS93.
6. Narysować schemat logiczny licznika asynchronicznego o zadanym modulo N, stosując metodę rozkładu liczby N na czynniki.
7. Narysować schemat układu licznika o zadanym modulo wykorzystując układy 74LS90 lub 74LS93.
8. Przedstawić rozwiązanie programowalnego dzielnika częstotliwości opartego o licznik o pojemności 8.
9. Omówić szeregowy sposób łączenia liczników asynchronicznych dla uzyskania licznika o większej pojemności.
10. Omówić równoległy sposób łączenia liczników asynchronicznych dla uzyskania licznika o większej pojemności.

3.2.8. Liczniki synchroniczne

Celem ćwiczenia jest kształtowanie umiejętności projektowania liczników synchronicznych realizowanych z przerzutników i bramek logicznych oraz zapoznanie się z budową, zasadą działania i możliwościami funkcyjnymi wybranych scalonych układów liczników synchronicznych.

Program ćwiczenia obejmuje:

- Zaprojektowanie, realizację i sprawdzenie poprawności działania liczników synchronicznych o zadanej pojemności, budowanych z bramek i przerzutników:

– zliczających w przód, dla zadanego typu przerzutnika i rodzaju przeniesienia (szeregowego albo równoległego);
– zliczających w tył, dla zadanego typu przerzutnika i rodzaju przeniesienia;
– dwukierunkowych (rewersyjnych).

- Zapoznanie się z budową, możliwościami funkcjonalnymi i sprawdzenie działania scalonych liczników 74LS192 i 74LS193.
- Zaprojektowanie, realizację i badanie liczników o programowanej pojemności N z zakresu 1÷15, a także liczników o pojemności z zakresu 2÷255 (należy stosować bramki i liczniki scalone).
- Poznanie budowy i zasady działania programowalnych dzielników częstotliwości.

Badane układy scalone:

∗ 74LS163 74LS193 74LS197

Przygotowanie do ćwiczeń.

Każda grupa otrzymuje indywidualne zadania projektowe. Do każdego zadania studenci przygotowują rozwiązania w postaci schematu ideowego układu, zaprojektowanego z dostępnych w zestawie elementów cyfrowych. Rozwiązania winny być przedstawione w protokole, który jest podstawą wykonania sprawozdania. Wymagana jest znajomość budowy, sposoby projektowania na przerzutnikach JK, D i T, struktury wewnętrznej i zasady działania badanych grup układów, zasady projektowania liczników mod. N.

Zagadnienia do opracowania:

1. Wyjaśnić podstawowe różnice pomiędzy licznikami asynchronicznymi a synchronicznymi.
2. Omówić budowę jednokierunkowego synchronicznego licznika dwójkowego zliczającego w naturalnym kodzie dwójkowym.

3. Omówić właściwości liczników jednokierunkowych na przykładzie układu 74LS163.
4. Przedstawić zasady kaskadowego łączenia liczników 74LS163.
5. Scharakteryzować synchroniczne liczniki rewersyjne.
6. Omówić właściwości licznika scalonego 74LS193.
7. Omówić rozwiązania licznika zliczającego mod. N wykorzystując układ 74LS193.
8. Przedstawić zasady kaskadowego łączenia liczników 74LS193 w licznik o dużej pojemności.
9. Omówić właściwości licznika scalonego 74LS197.

3.2.9. Układy arytmetyczne

Celem ćwiczenia jest doświadczalne poznanie:

- budowy, właściwości funkcyjnych i logicznych badanych sumatorów dwójkowych;
- możliwości wykorzystania sumatorów liczb dwójkowych i dwójkowo-dziesiętnych;
- budowy i przeznaczenia komparatorów.

Program ćwiczenia obejmuje:

- Zbudowanie i zbadanie półsumatora i sumatora jednobitowego.
- Zaprojektowanie i sprawdzenie działania:
  - układu sumatora dwóch liczb dwubitowych,
  - układu subtraktora dwóch liczb dwubitowych.
- Zapoznanie się z budową i sprawdzenie działania:
  - scalonego układu sumatora dwóch liczb czterobitowych 74LS83,
  - scalonego układu komparatora dwóch liczb czterobitowych 74LS85.
- Zaprojektowanie i sprawdzenie działania wskazanego układu:

– realizującego dodawanie i odejmowanie dwóch liczb czterobitowych w kodzie uzupełnienia do 2,
– komparatora szeregowego dwóch liczb czterobitowych A i B w kodzie naturalnym binarnym określającego relacje A=B, A≠B.

Badane układy scalone:

∗ 74LS83 74LS85

Przygotowanie do ćwiczeń.

Każda grupa otrzymuje indywidualne zadania projektowe. Treść i formę prezentacji zadań określa prowadzący zajęcia. Do każdego zadania studenci przygotowują rozwiązania w postaci wyrażeń logicznych i schematu ideowego układu, zaprojektowanego z dostępnych w zestawie elementów cyfrowych. Rozwiązania winny być przedstawione w protokole, który jest podstawą wykonania sprawozdania. Wymagana jest znajomość systemów zapisu liczb stosowanych w arytmetyce urządzeń liczących, zasady działania badanych grup układów oraz realizacji podstawowych operacji arytmetycznych na liczbach dwójkowych i dwójkowo-dziesiętnych, przedstawionych w różnych kodach.

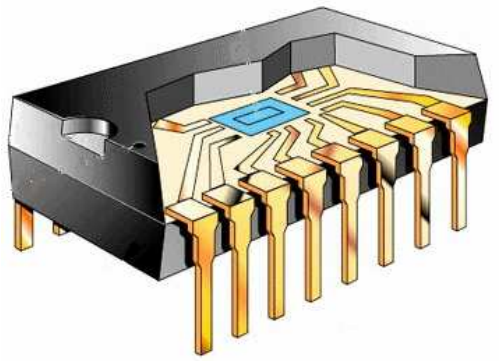
Zagadnienia do opracowania:

1. Zdefiniować i dokonać podziału układów arytmetycznych.
2. Opisać budowę i działanie sumatora jednobitowych liczb dwójkowych.
3. Podać funkcje logiczne sumatora i subtraktora pełnego.
4. Sumator binarny jako przykład układu iteracyjnego. Realizacja dodawania w zapisie ZU1i ZU2.
5. Wyjaśnić zasadę szeregowego dodawania wielobitowych liczb dwójkowych.
6. Wyjaśnić zasadę równoległego dodawania wielobitowych liczb dwójkowych.
7. Opisać budowę i działanie sumatora akumulującego.
8. Omówić budowę układu dodająco-odejmujacego liczb czterobitowych.

9. Omówić budowę komparatora równoległego.
10. Omówić budowę komparatora szeregowego.

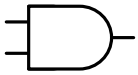
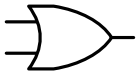
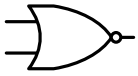
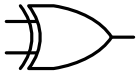
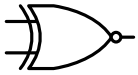
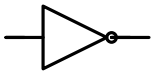
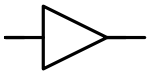
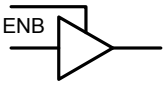
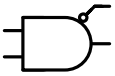
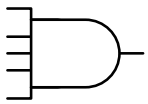
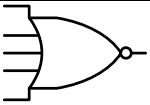
3.2.10. Termin odróbczy

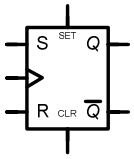
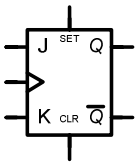
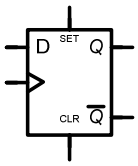
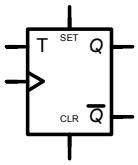
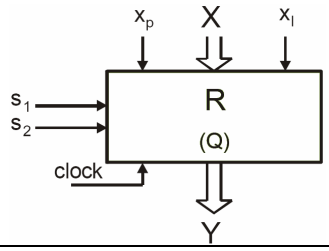
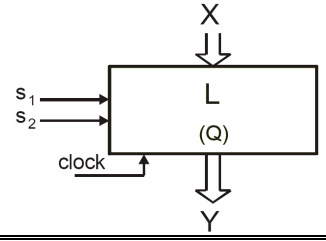
Możliwość poprawienia i odrobienia zaległości. W terminie obróbczym istnieje możliwość poprawienia jednego ćwiczenia laboratoryjnego.

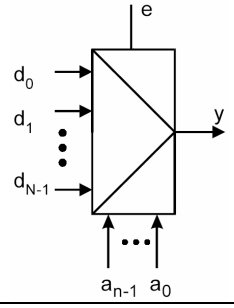
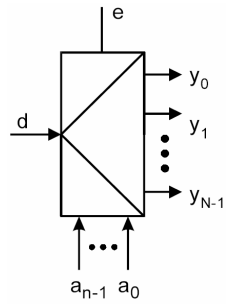
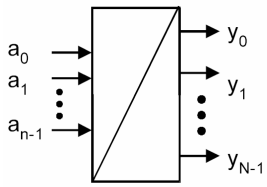
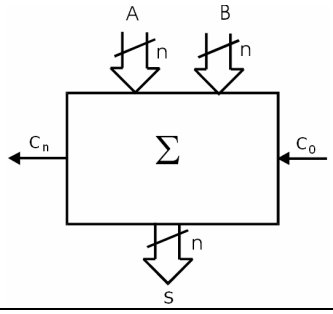
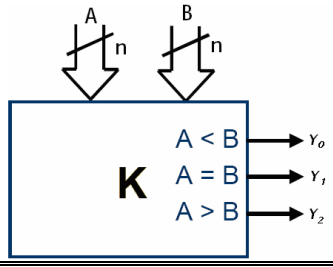


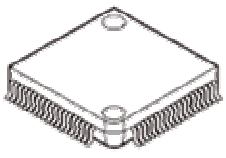
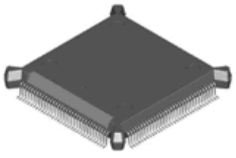
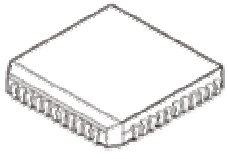
3.3. Elementy logiczne i bloki funkcjonalne występujące na stanowisku laboratoryjnym

3.3.1. Podstawowe oznaczenia graficzne

Nazwa elementu	Oznaczenie graficzne	Realizowana funkcja
AND (I)		$y = x_1 \cdot x_2$
OR (LUB)		$y = x_1 + x_2$
NAND (NIE-I)		$y = \overline{x_1 \cdot x_2}$
NOR (NIE-LUB)		$y = \overline{x_1 + x_2}$
EXOR (Wyłącznie LUB)		$y = x_1 \oplus x_2 = \overline{x_1} x_2 \vee x_1 \overline{x_2}$
EXNOR		$y = x_1 \otimes x_2 = \overline{x_1}\,\overline{x_2} \vee x_1 x_2$
NOT (NIE)		$y = \overline{x}$
BUFOR		$y = x$
BUFOR TRÓJSTANOWY		$y = x$
Ekspander AND		$y = x_1 \cdot x_2$ zwiększenie liczby wejść
AND bramka wielowejściowa		$y = x_1 \cdot x_2 \cdot \ldots \cdot x_n$
NOR bramka wielowejściowa		$y = \overline{x_1 + x_2 + \ldots + x_n}$

Nazwa elementu	Oznaczenie graficzne	Realizowana funkcja
Przerzutnik synchroniczny SR		$Q_{n+1} = S + \overline{R}_n Q_n$
Przerzutnik synchroniczny JK		$Q_{n+1} = J_n \overline{Q_n} + \overline{K_n} Q_n$
Przerzutnik synchroniczny D		$Q_{n+1} = D_n$
Przerzutnik synchroniczny T		$Q_{n+1} = T_n \overline{Q_n} + \overline{T_n} Q_n = T_n \oplus Q_n$
Rejestr		$Y := SHR(x_p, Y)$ $Y := SHL(Y, x_l)$
Licznik		$Y := Y + 1 = INC(Y)$ $Y := Y - 1 = DEC(Y)$

Nazwa elementu	Oznaczenie graficzne	Realizowana funkcja
Multiplekser		$y_k = e\sum_{k=0}^{N-1} P_k(A)d_k$
Demultiplekser		$y_k = eP_k(A)d$
Dekoder		$y_k = P_k(A)$
Sumator		$Y = A + B + c_0$
Komparator		$Y_i = f_i(A-B)$

3.3.2. Wykaz układów scalonych 74LS dostępnych na stanowisku laboratoryjnym

Elementy logiczne i bloki funkcjonalne

Lp.	Typ elementu	Ilość (szt.)	Lp.	Typ elementu	Ilość (szt.)
1.	74LS00	2	16.	74LS138	1
2.	74LS02	2	17.	74LS145	2
3.	74LS04	1	18.	74LS148	1
4.	74LS08	2	19.	74LS153	1
5.	74LS10	2	20.	74LS155	1
6.	74LS20	2	21.	74LS163	1
7.	74LS32	2	22.	74LS164	1
8.	74LS38	2	23.	74LS165	1
9.	74LS47	1	24.	74LS174	1
10.	74LS74	2	25.	74LS193	2
11.	74LS83	2	26.	74LS194	1
12.	74LS86	2	27.	74LS197	1
13.	74LS90	2	28.	74LS244	2
14.	74LS93	2	29.	74LS245	2
15.	74LS107	2	30.	74LS259	1

Struktury PLD

Lp.	Typ elementu	Ilość (szt.)
1.	GAL16V8	2
2.	GAL22V10	2

3.3.3. Funkcje układów scalonych 74LS występujących na stanowisku laboratoryjnym

7400 - ***Quad 2-input NAND gates.*** **–** Czterokrotne, 2-wejściowe bramki NAND.

7402 - ***Quad 2-input NOR gates*** - Czterokrotne, 2-wejściowe bramki NOR.

7404 - ***Hex inverters*** - Sześciokrotne inwertory.

7408 - ***Quad 2-input AND gates*** - Czterokrotne, 2-wejściowe bramki AND.

7410 - ***Triple 3-input NAND gates*** - Trzykrotne, 3-wejściowe bramki NAND.

7420 - ***Dual 4-input NAND gates*** - Dwukrotne, 4-wejściowe bramki NAND.

7432 - ***Quad 2-input OR gates*** - Czterokrotne, 2-wejściowe bramki OR.

7438 - ***Quad 2-input open-collector NAND gates with buffered output*** - Czterokrotne, 2-wejściowe bramki buforowe NAND z otwartym kolektorem tranzystora wyjściowego.

7447 - ***Open-collector BCD to 7-segment decoder/common-anode LED driver with ripple blank input*** - Dekoder kodu BCD na kod wskaźnika 7-segmentowego, z wyjściami typu otwarty kolektor.

7474 - **Dual D flip-flop with set and reset** - Dwukrotne przerzutniki typu D wyzwalane dodatnim zboczem.

7483 - ***4-bit binary full adder with fast carry*** - Pełny sumator 4-pozycyjny (binarny) z szybkim przeniesieniem.

7486 - ***Quad 2-input XOR gates*** - Czterokrotne, 2-wejściowe bramki exclusive-OR

7490 - ***4-bit asynchronous decade counter with /2 and /5 sections, set(9) and reset*** - 4-bitowy asynchroniczny licznik dziesiętny.

7493 - ***4-bit asynchronous binary counter with /2 and /8 sections and reset*** - 4-bitowy asynchroniczny licznik dwójkowy.

74107 - ***Dual J-K flip-flop with clear*** - Dwukrotne przerzutniki J-K MS z wejściem zerowania.

74138 - ***1-of-8 inverting decoder/demultiplexer.*** - Dekoder/demultiplekser z 3 linii na 8 linii.

74145 - ***1-of-10 open-collector inverting decoder/demultiplexer.*** - Dekoder kodu BCD na kod dziesiętny z wyjściami typu otwarty kolektor.

74148 - ***8-to-3 line inverting priority encoder with cascade inputs.*** - Enkoder priorytetowy typu "8 linii na 3 linie".

74153 - ***8-to-2 line noninverting data selector/multiplexer with separate enables.*** - Podwójne, 4-wejściowe multipleksery/selektory danych.

74155 - ***2-of-8 inverting decoder/demultiplexer with separate enables.*** - Podwójne dekodery/demultipleksery z 2 linii na 4 linie.

74163 - **4*-bit synchronous binary counter with load, reset, and ripple carry output.*** - Synchroniczny, 4-bitowy licznik dwójkowy z wejściem zerującym.

74164 - ***8-bit serial-in parallel-out shift register with asynchronous reset and two AND gated serial inputs.*** - 8-bitowy rejestr przesuwający, z wejściami szeregowymi i wyjściami równoległymi.

74165 - ***8-bit parallel-in serial-out shift register with asynchronous parallel load and two OR gated clock inputs.*** - 8-bitowy rejestr przesuwający, z wejściami równoległymi i wyjściami szeregowym.

74174 - ***8-bit D flip-flop with reset.*** - Ośmiokrotne przerzutniki typu D, z wejściem zerującym.

74193 - **4*-bit synchronous binary up/down counter with asynchronous load and reset, and separate up and down clocks. Carry and borrow outputs.*** - Synchroniczny binarny licznik rewersyjny o 2-wejściach zegarowych z kasowaniem.

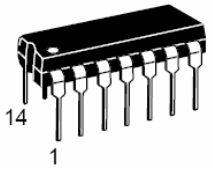
74194 - ***4-bit bidirectional universal shift register with asynchronous reset.*** - 4-bitowy, dwukierunkowy rejestr przesuwający.

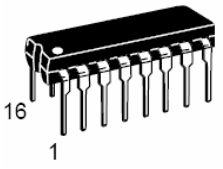
74197 - ***4-bit asynchronous binary counter with /2 and /8 sections, load and reset.*** - 30 MHz, ustawialny, 4-bitowy, dwójkowy licznik/zatrzask.

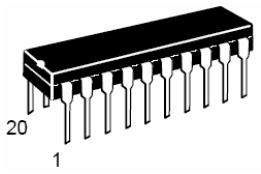
74244 - ***Dual 4-bit 3-state noninverting buffer/line driver***. - 8-bitowy (dwukrotny, 4-bitowy), buforowany odbiornik linii, wyjście proste.

74245 - ***8-bit 3-state noninverting bus transceiver. Enable and direction pins control output enables*****.** - 8-bitowa brama, wyjścia 3-stanowe, proste.

74259 - ***1-of-8 addressable latch with reset.*** **-** 8-bitowy adresowany zatrzask.

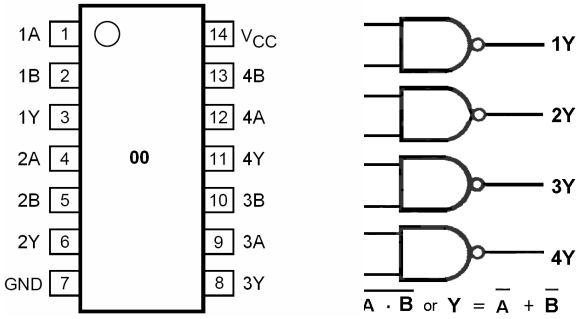




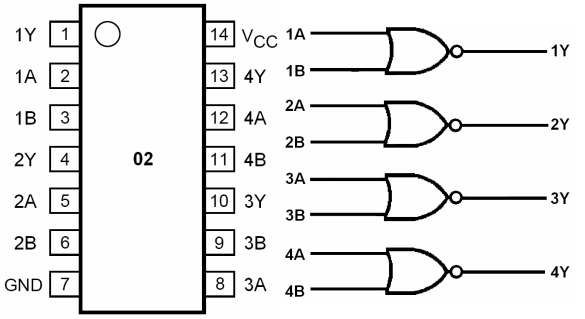


3.3.4. Wyciąg z not katalogowych układów scalonych 74LS występujących na stanowisku laboratoryjnym

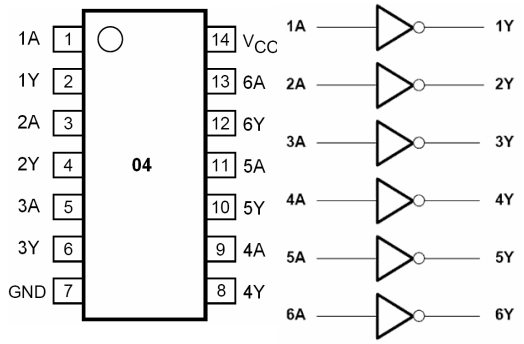
SN7400, SN74LS00, SN74S00
QUADRUPLE 2-INPUT POSITIVE-NAND GATES



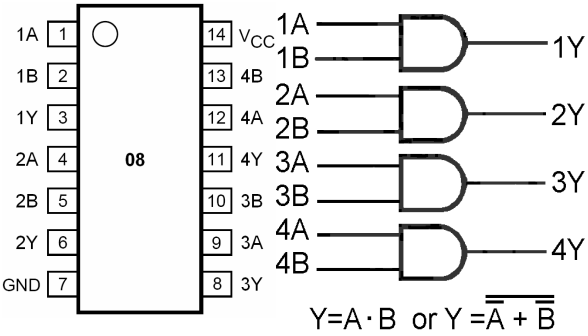
SN7402, SN74LS02, SN74S02
QUADRUPLE 2-INPUT POSITIVE-NOR GATES



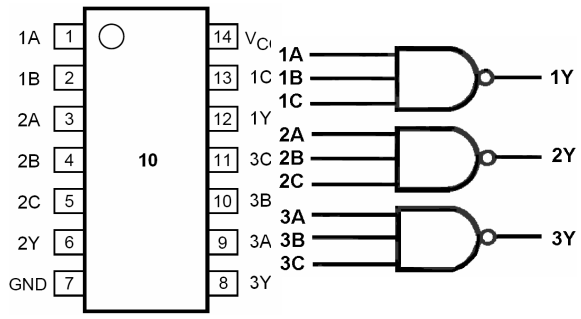
SN7404, SN74LS04, SN74S04 HEX INVERTERS



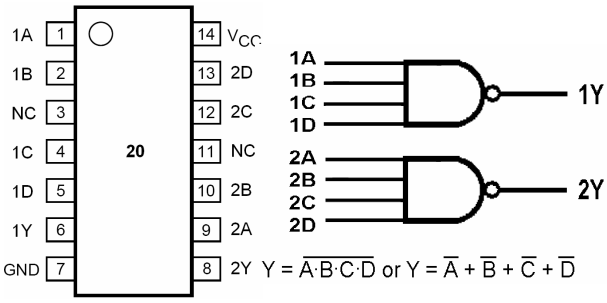
SN7408, SN74LS08, SN74S08
QUADRUPLE 2-INPUT POSITIVE-AND GATES



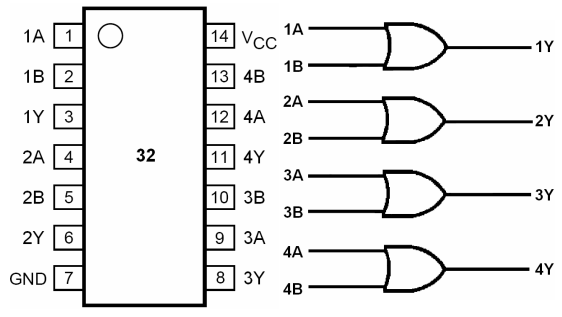
SN7410, SN74LS10, SN74S10
TRIPLE 3-INPUT POSITIVE-NAND GATES



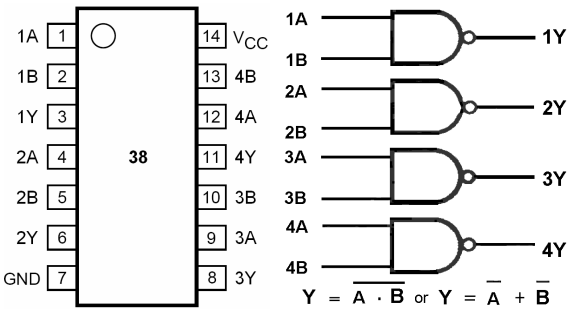
SN7420, SN74LS20, SN74S20
DUAL 4-INPUT POSITIVE-NAND GATES



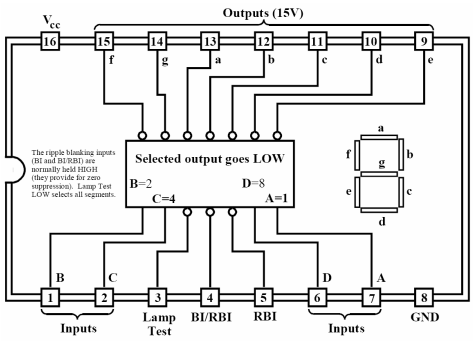
SN7432, SN74LS32, SN74S32
QUADRUPLE 2-INPUT POSITIVE OR GATES



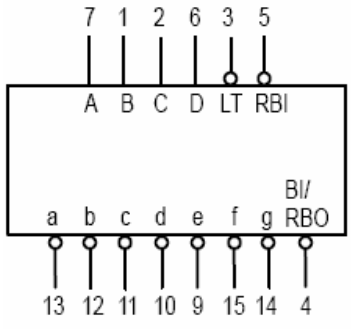
SN7438, SN74LS38, SN74S38
QUADRUPLE 2-INPUT POSITIVE-NAND BUFFERS
WITH OPEN-COLLECTOR OUTPUTS



SN7446A, '47A, '48, SN74LS47, 'LS48, 'LS49
BCD-TO-SEVEN-SEGMENT DECODERS/DRIVERS



LOGIC SYMBOL

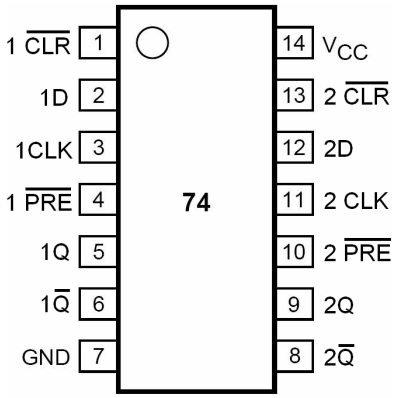


TRUTH TABLE

	INPUTS							OUTPUTS							
DECIMAL OR FUNCTION	$\overline{LT}$	$\overline{RBI}$	D	C	B	A	$\overline{BI/RBO}$	$\overline{a}$	$\overline{b}$	$\overline{c}$	$\overline{d}$	$\overline{e}$	$\overline{f}$	$\overline{g}$	NOTE
0	H	H	L	L	L	L	H	L	L	L	L	L	L	H	A
1	H	X	L	L	L	H	H	H	L	L	H	H	H	H	A
2	H	X	L	L	H	L	H	L	L	H	L	L	H	L	
3	H	X	L	L	H	H	H	L	L	L	L	H	H	L	
4	H	X	L	H	L	L	H	H	L	L	H	H	L	L	
5	H	X	L	H	L	H	H	L	H	L	L	H	L	L	
6	H	X	L	H	H	L	H	H	H	L	L	L	L	L	
7	H	X	L	H	H	H	H	L	L	L	H	H	H	H	
8	H	X	H	L	L	L	H	L	L	L	L	L	L	L	
9	H	X	H	L	L	H	H	L	L	L	H	H	L	L	
10	H	X	H	L	H	L	H	H	H	H	L	L	H	L	
11	H	X	H	L	H	H	H	H	H	L	L	H	H	L	
12	H	X	H	H	L	L	H	H	L	H	H	H	L	L	
13	H	X	H	H	L	H	H	L	H	H	L	H	L	L	
14	H	X	H	H	H	L	H	H	H	H	L	L	L	L	
15	H	X	H	H	H	H	H	H	H	H	H	H	H	H	
BI	X	X	X	X	X	X	L	H	H	H	H	H	H	H	B
RBI	H	L	L	L	L	L	L	H	H	H	H	H	H	H	C
LT	L	X	X	X	X	X	H	L	L	L	L	L	L	L	D

H = HIGH Voltage Level
L = LOW Voltage Level
X = Immaterial

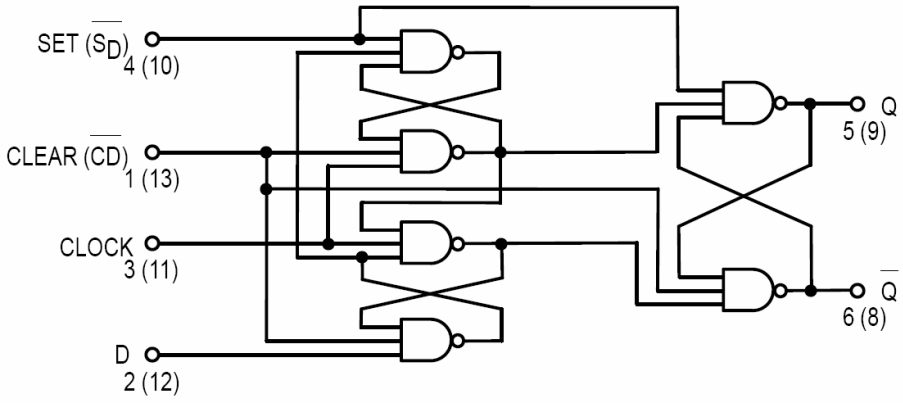
SN7474, SN74LS74A, SN74S74
DUAL D-TYPE POSITIVE-EDGE-TRIGGERED FLIP-FLOPS WITH PRESET AND CLEAR



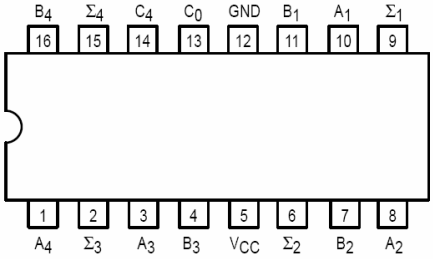
Inputs				Outputs	
$\overline{PR}$	$\overline{CLR}$	CLK	D	Q	$\overline{Q}$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H*	H*
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q_0	$\overline{Q}_0$

H = High Logic Level
X = Either Low or High Logic Level
L = Low Logic Level
↑ = Positive-going Transition
* = This configuration is nonstable; that is, it will not persist when either the preset and/or clear inputs return to their inactive (high) level.
Q_0 = The output logic level of Q before the indicated input conditions were established.

LOGIC DIAGRAM (Each Flip-Flop)

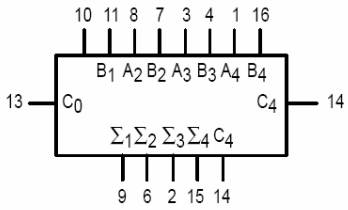


SN7483, SN74LS83, SN74S83
4-BIT BINARY FULL ADDERS WITH FAST CARRY

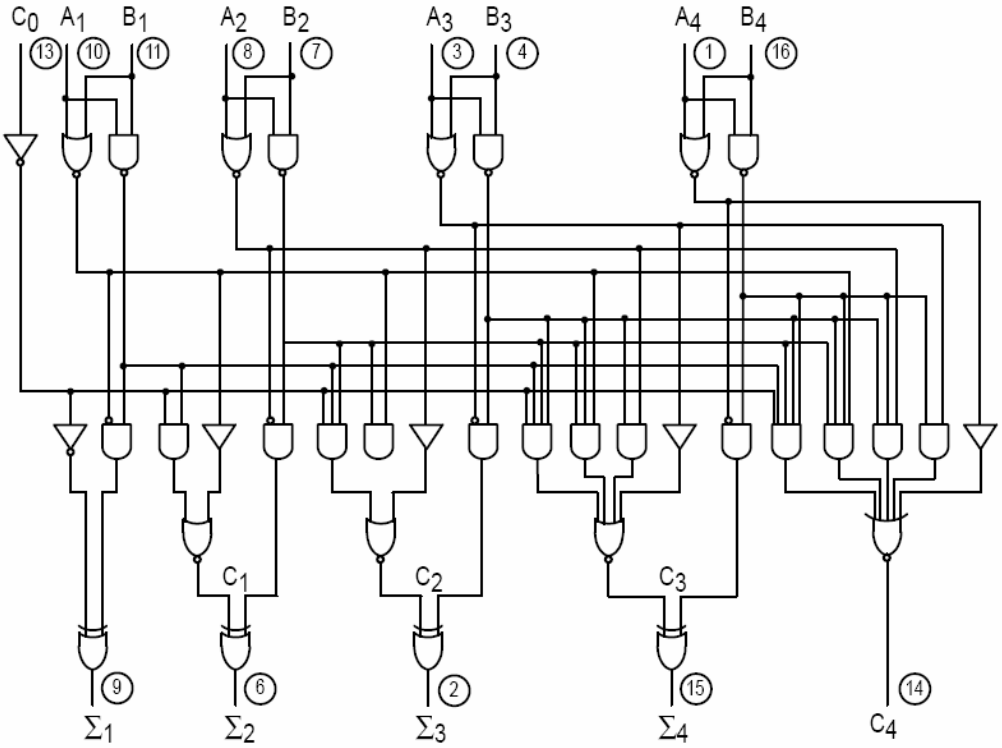


$A_1 - A_4$	Operand A Inputs
$B_1 - B_4$	Operand B Inputs
C_0	Carry Input
$\Sigma_1 - \Sigma_4$	Sum Outputs (Note b)
C_4	Carry Output (Note b)

LOGIC SYMBOL



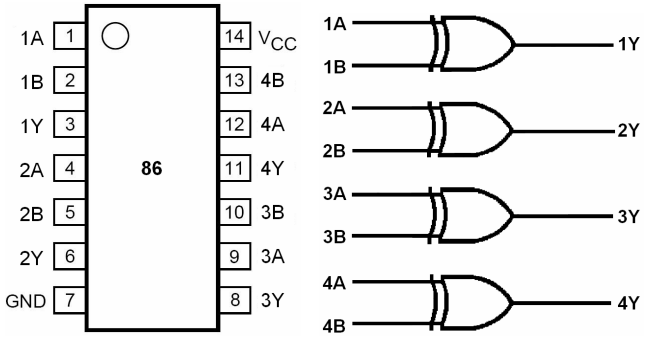
LOGIC DIAGRAM



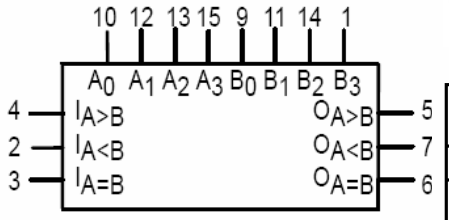
$C_0 + (A_1+B_1)+2(A_2+B_2)+4(A_3+B_3)+8(A_4+B_4) = \Sigma_1+2\Sigma_2+4\Sigma_3+8\Sigma_4+16C_4$

	C_0	A_1	A_2	A_3	A_4	B_1	B_2	B_3	B_4	Σ_1	Σ_2	Σ_3	Σ_4	C_4	
Logic Levels	L	L	H	L	H	H	L	L	H	H	H	L	L	H	
Active HIGH	0	0	1	0	1	1	0	0	1	1	1	0	0	1	(10+9 = 19)
Active LOW	1	1	0	1	0	0	1	1	0	0	0	1	1	0	(carry+5+6 = 12)

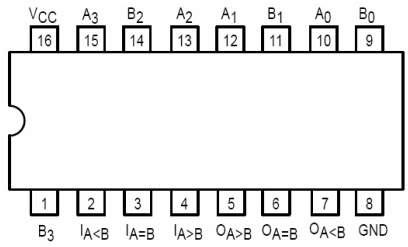
SN7486, SN74LS86A, SN74S86
QUADRUPLE 2-INPUT EXCLUSIVE-OR GATES



SN7485, SN74LS85, SN74S85
4-BIT MAGNITUDE COMPARATORS

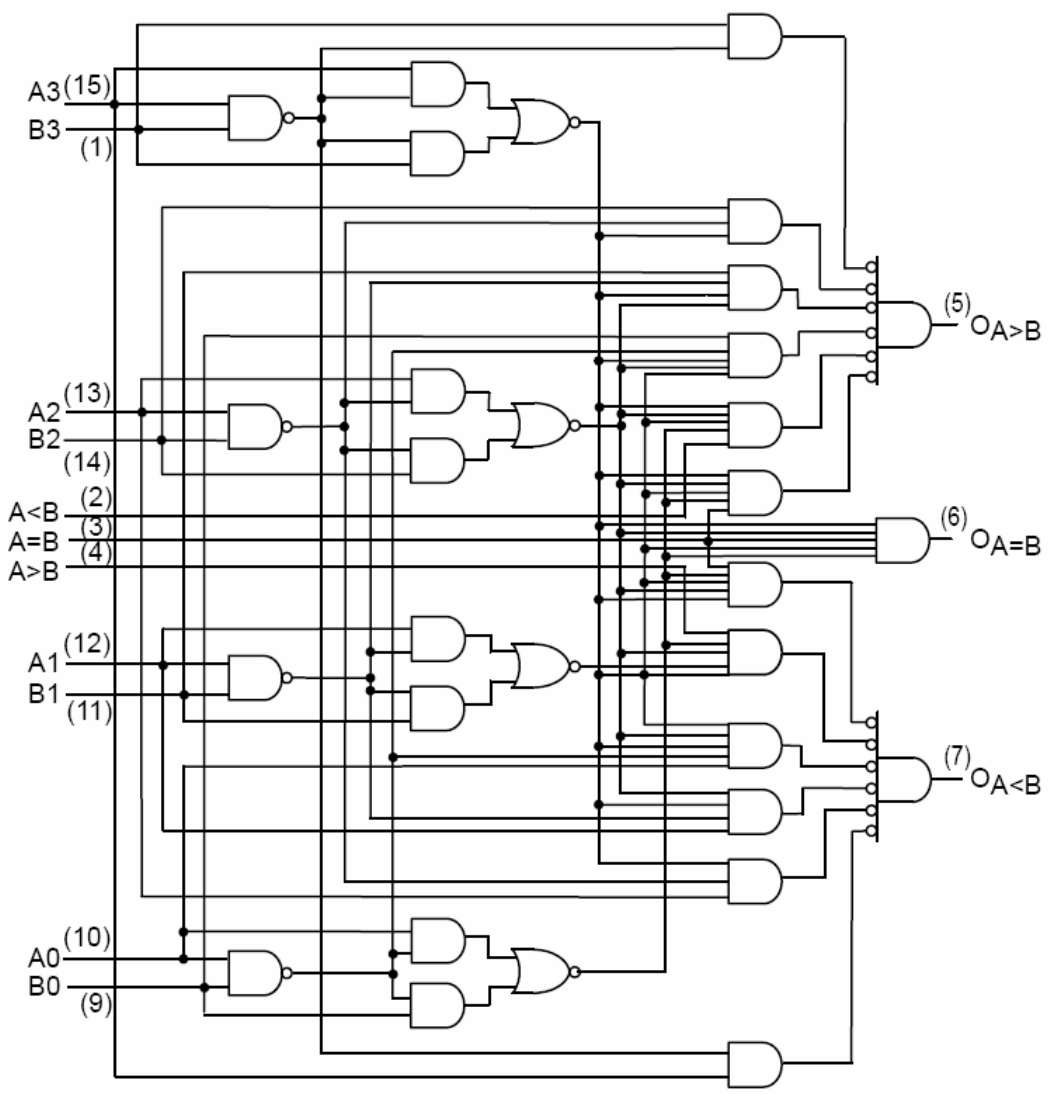


V_{CC} = PIN 16
GND = PIN 8



TRUTH TABLE

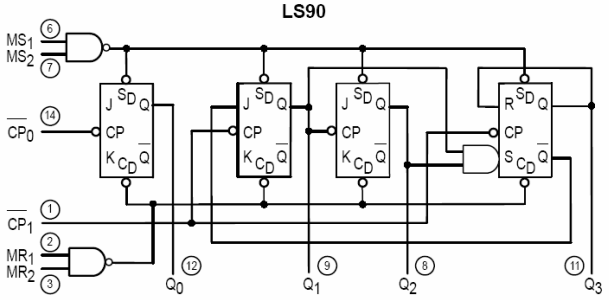
COMPARING INPUTS				CASCADING INPUTS			OUTPUTS		
A_3,B_3	A_2,B_2	A_1,B_1	A_0,B_0	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$	$O_{A>B}$	$O_{A<B}$	$O_{A=B}$
$A_3>B_3$	X	X	X	X	X	X	H	L	L
$A_3<B_3$	X	X	X	X	X	X	L	H	L
$A_3=B_3$	$A_2>B_2$	X	X	X	X	X	H	L	L
$A_3=B_3$	$A_2<B_2$	X	X	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1>B_1$	X	X	X	X	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1<B_1$	X	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B1$	$A_0>B_0$	X	X	X	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0<B_0$	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	H	L	L	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	L	H	L	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	X	X	H	L	L	H
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	H	H	L	L	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	L	L	L	H	H	L



SN7490A, SN74LS90, DECADE, COUNTERS

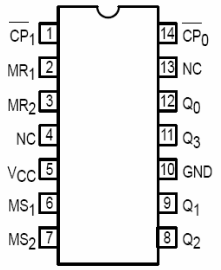
LOGIC DIAGRAM

LS90



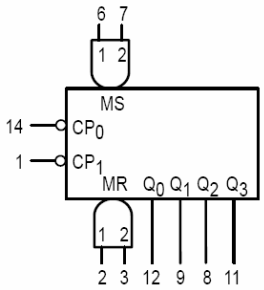
○ = PIN NUMBERS
V_{CC} = PIN 5
GND = PIN 10

CONNECTION DIAGRAM
DIP (TOP VIEW)



NC = NO INTERNAL CONNECTION

NOTE:
The Flatpak version has the same pinouts (Connection Diagram) as the Dual In-Line Package.



V_{CC} = PIN 5
GND = PIN 10
NC = PINS 4, 13

MODE SELECTION

RESET/SET INPUTS				OUTPUTS			
MR_1	MR_2	MS_1	MS_2	Q_0	Q_1	Q_2	Q_3
H	H	L	X	L	L	L	L
H	H	X	L	L	L	L	L
X	X	H	H	H	L	L	H
L	X	L	X	Count			
X	L	X	L	Count			
L	X	X	L	Count			
X	L	L	X	Count			

H = HIGH Voltage Level
L = LOW Voltage Level
X = Don't Care

BCD COUNT SEQUENCE

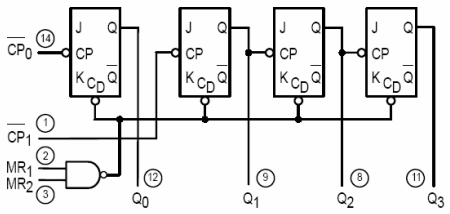
COUNT	OUTPUT			
	Q_0	Q_1	Q_2	Q_3
0	L	L	L	L
1	H	L	L	L
2	L	H	L	L
3	H	H	L	L
4	L	L	H	L
5	H	L	H	L
6	L	H	H	L
7	H	H	H	L
8	L	L	L	H
9	H	L	L	H

NOTE: Output Q_0 is connected to Input CP_1 for BCD count.

SN7493A, SN74LS93 BINARY COUNTERS

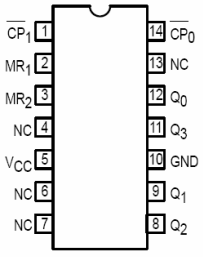
LOGIC DIAGRAM

LS93



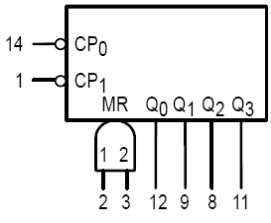
○ = PIN NUMBERS
V_{CC} = PIN 5
GND = PIN 10

CONNECTION DIAGRAM
DIP (TOP VIEW)



NC = NO INTERNAL CONNECTION

NOTE:
The Flatpak version has the same pinouts (Connection Diagram) as the Dual In-Line Package.



V_{CC} = PIN 5
GND = PIN 10
NC = PIN 4, 6, 7, 13

MODE SELECTION

RESET INPUTS		OUTPUTS			
MR_1	MR_2	Q_0	Q_1	Q_2	Q_3
H	H	L	L	L	L
L	H	Count			
H	L	Count			
L	L	Count			

H = HIGH Voltage Level
L = LOW Voltage Level
X = Don't Care

TRUTH TABLE

COUNT	OUTPUT			
	Q_0	Q_1	Q_2	Q_3
0	L	L	L	L
1	H	L	L	L
2	L	H	L	L
3	H	H	L	L
4	L	L	H	L
5	H	L	H	L
6	L	H	H	L
7	H	H	H	L
8	L	L	L	H
9	H	L	L	H
10	L	H	L	H
11	H	H	L	H
12	L	L	H	H
13	H	L	H	H
14	L	H	H	H
15	H	H	H	H

NOTE: Output Q_0 is connected to Input CP_1.

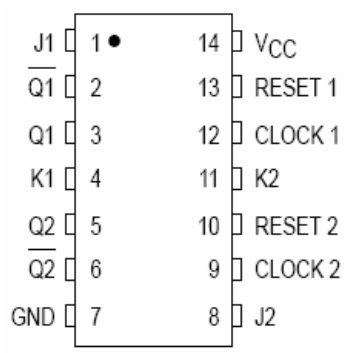
SN54107, SN54LS107A, SN74107, SN74LS107A DUAL J-K FLIP-FLOPS WITH CLEAR

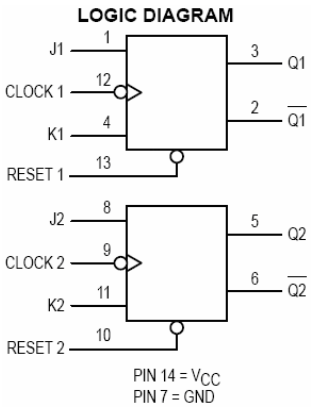
'107
FUNCTION TABLE

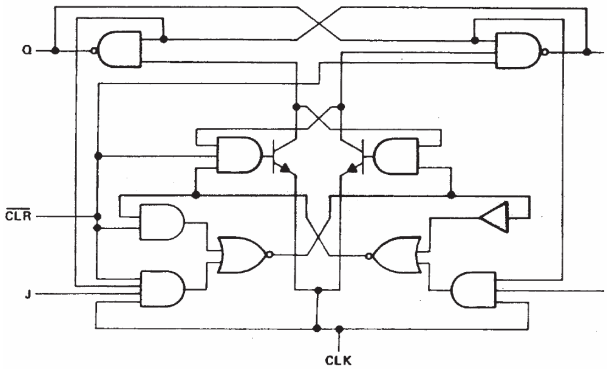
INPUTS				OUTPUTS	
$\overline{CLR}$	CLK	J	K	Q	$\overline{Q}$
L	X	X	X	L	H
H	↓	L	L	Q_0	$\overline{Q}_0$
H	↓	H	L	H	L
H	↓	L	H	L	H
H	↓	H	H	TOGGLE	

'LS107A
FUNCTION TABLE

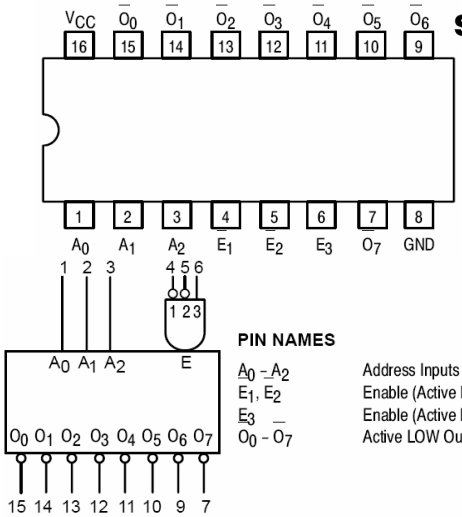
INPUTS				OUTPUTS	
$\overline{CLR}$	CLK	J	K	Q	$\overline{Q}$
L	X	X	X	L	H
H	↓	L	L	Q_0	$\overline{Q}_0$
H	↓	H	L	H	L
H	↓	L	H	L	H
H	↓	H	H	TOGGLE	
H	H	X	X	Q_0	$\overline{Q}_0$





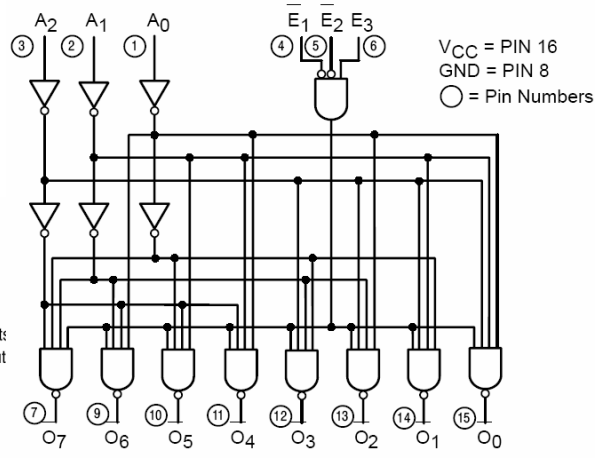


SN74LS138 1-of-8 Decoder/Demultiplexer



PIN NAMES

A_0 – A_2	Address Inputs
E_1, E_2	Enable (Active LOW) Inputs
E_3	Enable (Active HIGH) Input
O_0 – O_7	Active LOW Outputs

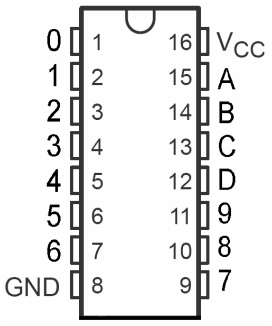


TRUTH TABLE

INPUTS						OUTPUTS							
$\overline{E_1}$	$\overline{E_2}$	E_3	A_0	A_1	A_2	$\overline{O_0}$	$\overline{O_1}$	$\overline{O_2}$	$\overline{O_3}$	$\overline{O_4}$	$\overline{O_5}$	$\overline{O_6}$	$\overline{O_7}$
H	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	L	X	X	X	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	H	L	L	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	H	H	L	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	L	H	H
L	L	H	L	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

H = HIGH Voltage Level
L = LOW Voltage Level
X = Don't Care

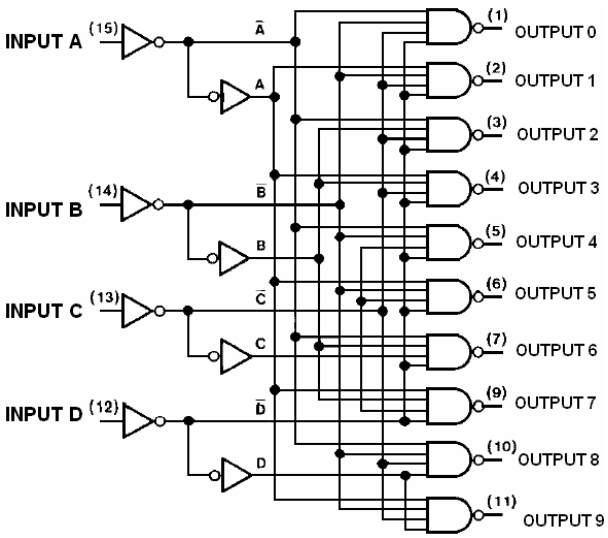
SN74145, SN74LS145
BCD-TO-DECIMAL DECODERS/ DRIVERS



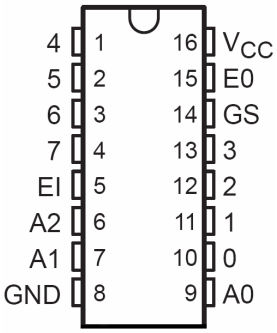
Function Table

No.	Inputs				Outputs									
	D	C	B	A	0	1	2	3	4	5	6	7	8	9
0	L	L	L	L	L	H	H	H	H	H	H	H	H	H
1	L	L	L	H	H	L	H	H	H	H	H	H	H	H
2	L	L	H	L	H	H	L	H	H	H	H	H	H	H
3	L	L	H	H	H	H	H	L	H	H	H	H	H	H
4	L	H	L	L	H	H	H	H	L	H	H	H	H	H
5	L	H	L	H	H	H	H	H	H	L	H	H	H	H
6	L	H	H	L	H	H	H	H	H	H	L	H	H	H
7	L	H	H	H	H	H	H	H	H	H	H	L	H	H
8	H	L	L	L	H	H	H	H	H	H	H	H	L	H
9	H	L	L	H	H	H	H	H	H	H	H	H	H	L
I	H	L	H	L	H	H	H	H	H	H	H	H	H	H
N	H	L	H	H	H	H	H	H	H	H	H	H	H	H
V	H	H	L	L	H	H	H	H	H	H	H	H	H	H
A	H	H	L	H	H	H	H	H	H	H	H	H	H	H
L	H	H	H	L	H	H	H	H	H	H	H	H	H	H
I	H	H	H	H	H	H	H	H	H	H	H	H	H	H
D														

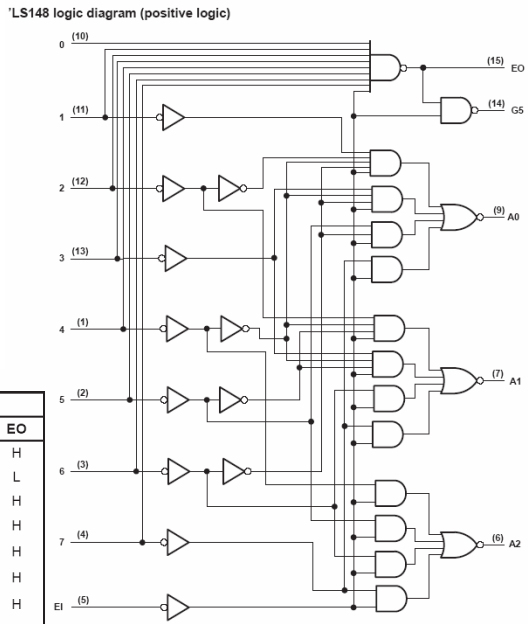
H = HIGH Level (OFF)
L = LOW Level (ON)



SN74LS148 -LINE TO 3-LINE PRIORITY ENCODERS



'LS148 logic diagram (positive logic)

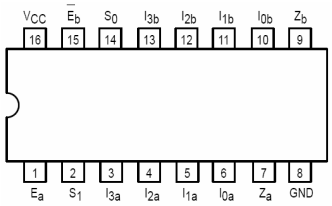


FUNCTION TABLE – '148, 'LS148

INPUTS									OUTPUTS				
EI	0	1	2	3	4	5	6	7	A2	A1	A0	GS	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

H = high logic level, L = low logic level, X = irrelevant

SN54/74LS153
DUAL 4-INPUT MULTIPLEXER



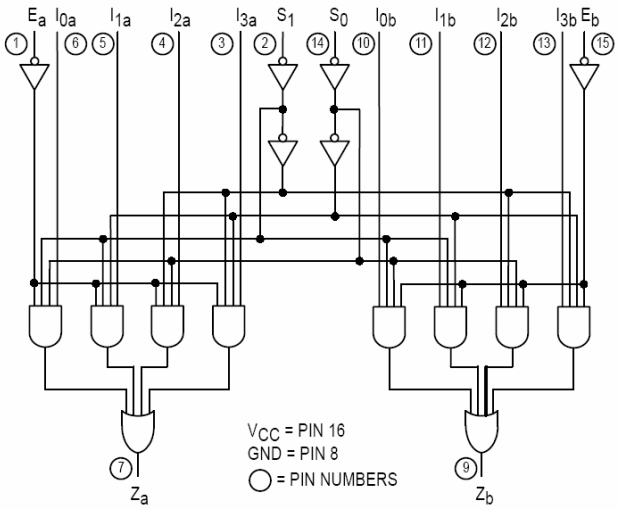
S_0 Common Select Input
$\overline{E}$ Enable (Active LOW) Input
I_0, I_1 Multiplexer Inputs
Z Multiplexer Output (Note b)

TRUTH TABLE

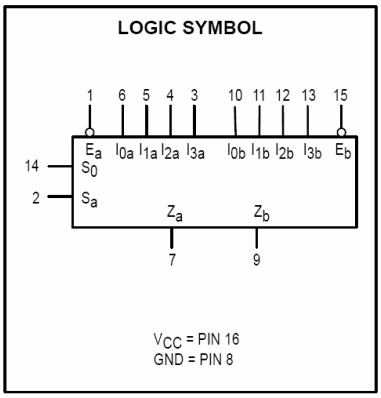
SELECT INPUTS		INPUTS (a or b)					OUTPUT
S_0	S_1	$\overline{E}$	I_0	I_1	I_2	I_3	Z
X	X	H	X	X	X	X	L
L	L	L	L	X	X	X	L
L	L	L	H	X	X	X	H
H	L	L	X	L	X	X	L
H	L	L	X	H	X	X	H
L	H	L	X	X	L	X	L
L	H	L	X	X	H	X	H
H	H	L	X	X	X	L	L
H	H	L	X	X	X	H	H

H = HIGH Voltage Level
L = LOW Voltage Level
X = Don't Care

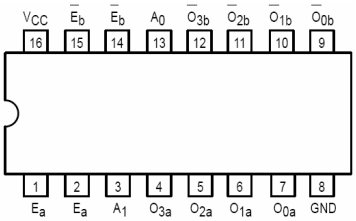
LOGIC DIAGRAM

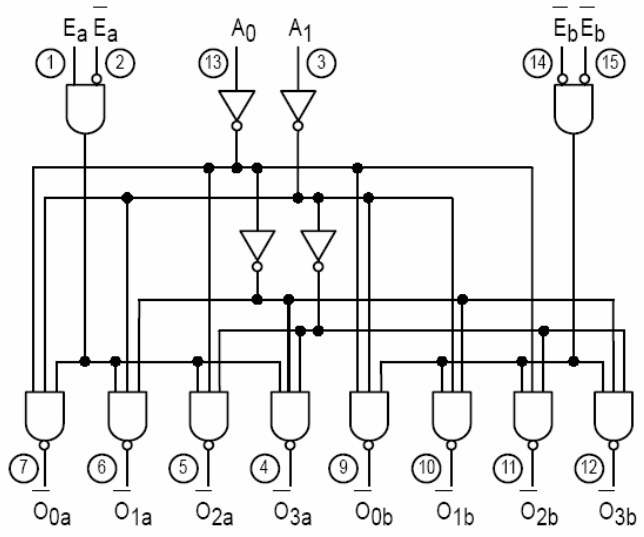


LOGIC SYMBOL

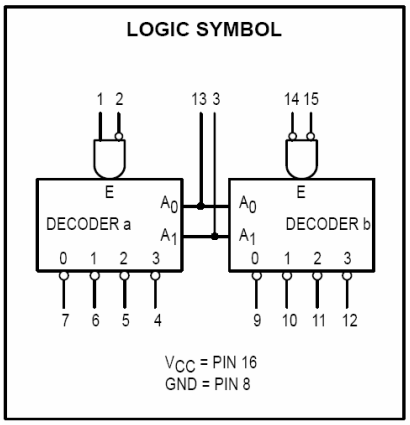


SN74155, SN74156, SN74LS155A, SN74LS156
DUAL 2-LINE TO 4-LINE DECODERS/DEMULTIPLEXERS





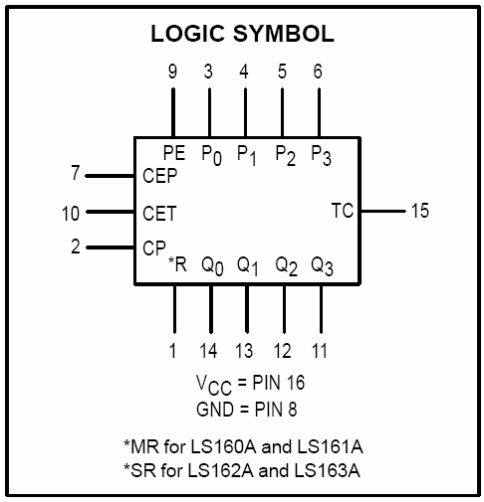
LOGIC SYMBOL

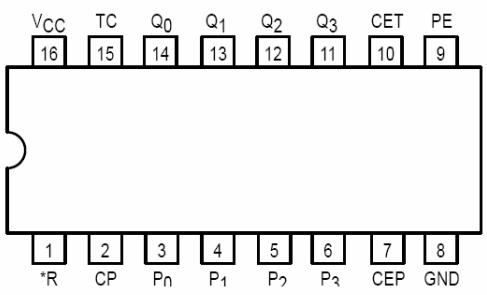


TRUTH TABLE

ADDRESS		ENABLE "a"		OUTPUT "a"				ENABLE "b"		OUTPUT "b"			
A_0	A_1	E_a	$\overline{E}_a$	$\overline{O}_0$	$\overline{O}_1$	$\overline{O}_2$	$\overline{O}_3$	E_b	$\overline{E}_b$	$\overline{O}_0$	$\overline{O}_1$	$\overline{O}_2$	$\overline{O}_3$
X	X	L	X	H	H	H	H	H	X	H	H	H	H
X	X	X	H	H	H	H	H	X	H	H	H	H	H
L	L	H	L	L	H	H	H	L	L	L	H	H	H
H	L	H	L	H	L	H	H	L	L	H	L	H	H
L	H	H	L	H	H	L	H	L	L	H	H	L	H
H	H	H	L	H	H	H	L	L	L	H	H	H	L

SN54/74LS163A
4-BIT BINARY COUNTERS



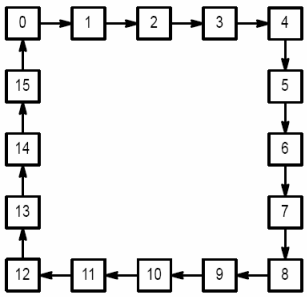


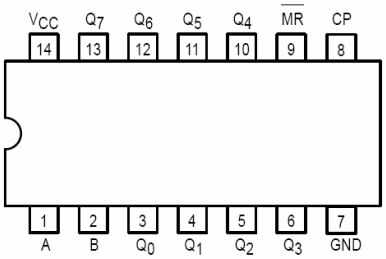
$\overline{PE}$	Parallel Enable (Active LOW) Input
P_0-P_3	Parallel Inputs
CEP	Count Enable Parallel Input
CET	Count Enable Trickle Input
$\underline{CP}$	Clock (Active HIGH Going Edge) Input
$\underline{MR}$	Master Reset (Active LOW) Input
SR	Synchronous Reset (Active LOW) Input
Q_0-Q_3	Parallel Outputs (Note b)
TC	Terminal Count Output (Note b)

MODE SELECT TABLE

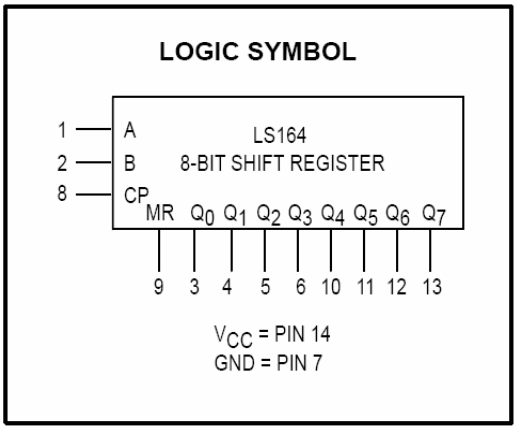
*SR	PE	CET	CEP	Action on the Rising Clock Edge (⌐)
L	X	X	X	RESET (Clear)
H	L	X	X	LOAD ($P_n \rightarrow Q_n$)
H	H	H	H	COUNT (Increment)
H	H	L	X	NO CHANGE (Hold)
H	H	X	L	NO CHANGE (Hold)

LS161A • LS163A





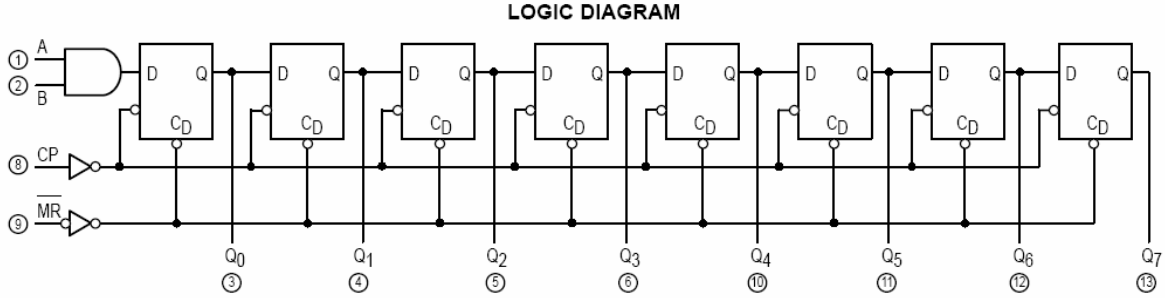
SN54/74LS164
8-Bit Serial In/Parallel Out Shift Registers



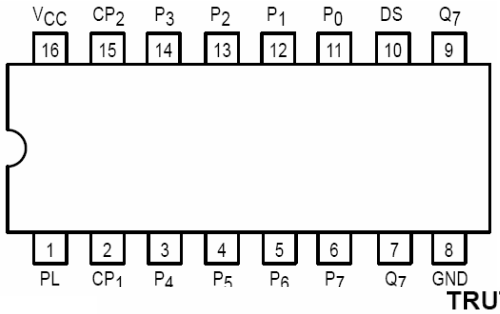
MODE SELECT — TRUTH TABLE

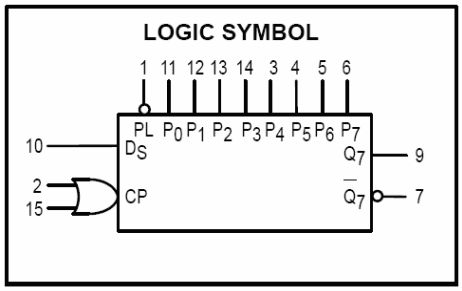
OPERATING MODE	INPUTS			OUTPUTS	
	$\overline{MR}$	A	B	Q_0	Q_1–Q_7
Reset (Clear)	L	X	X	L	L – L
Shift	H	l	l	L	$q_0 - q_6$
	H	l	h	L	$q_0 - q_6$
	H	h	l	L	$q_0 - q_6$
	H	h	h	H	$q_0 - q_6$

LOGIC DIAGRAM



SN54/74LS165
8-BIT PARALLEL-TO-SERIAL SHIFT REGISTER

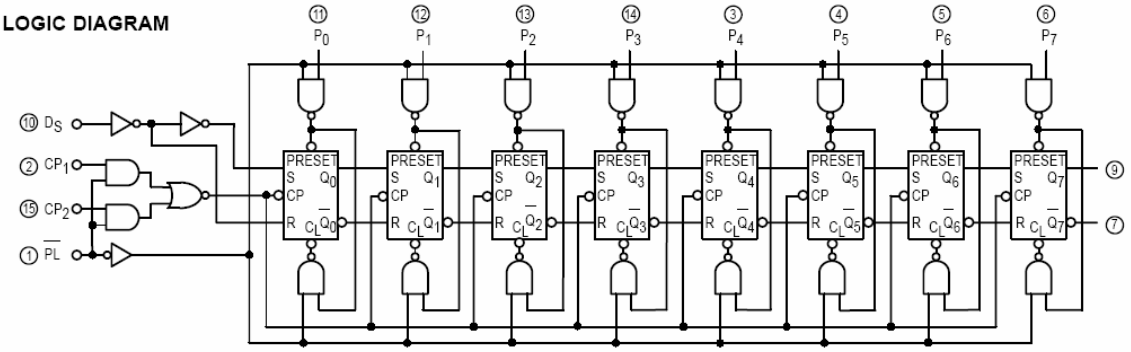




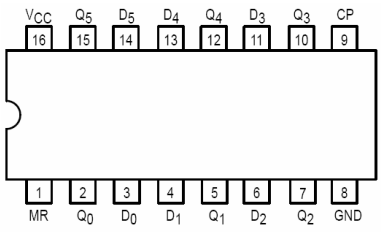
TRUTH TABLE

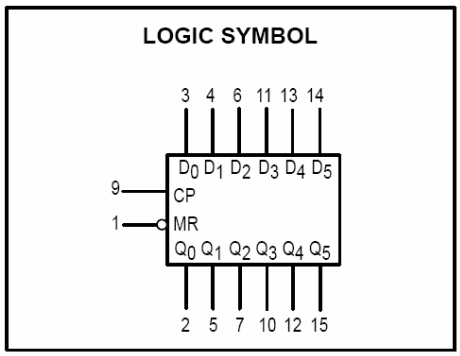
PL	CP		CONTENTS								RESPONSE
	1	2	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	
L	X	X	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	Parallel Entry
H	L	↓	D_S	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Right Shift
H	H	↓	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	No Change
H	↓	L	D_S	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Right Shift
H	↓	H	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	No Change

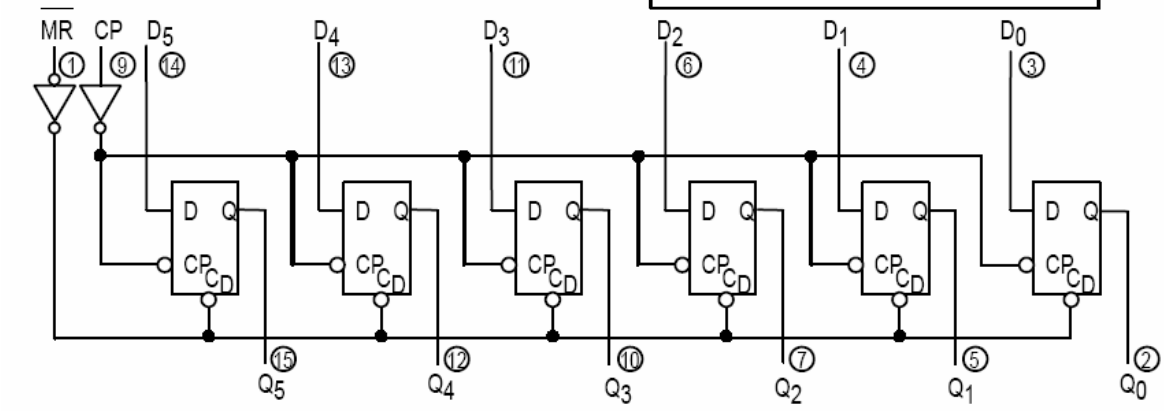
LOGIC DIAGRAM



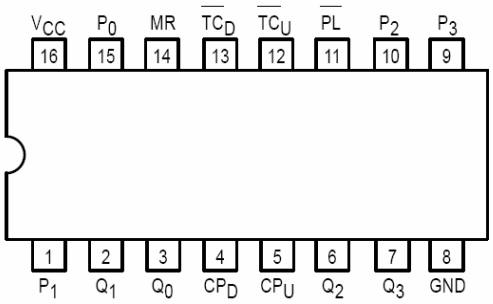
SN54/74LS174
HEX D FLIP-FLOP

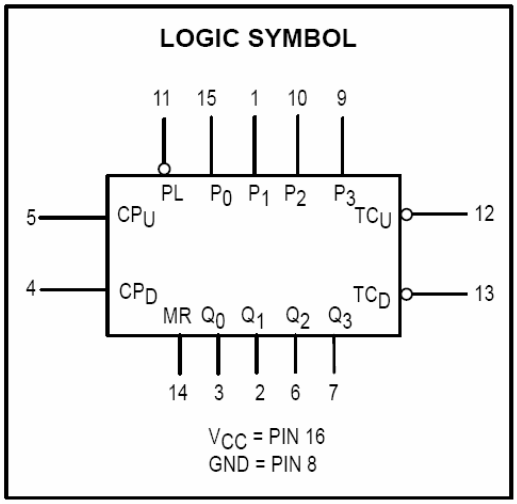






SN74192, SN74193, SN74LS192, SN74LS193
SYNCHRONOUS 4-BIT UP/DOWN COUNTERS (DUAL CLOCK WITH CLEAR)





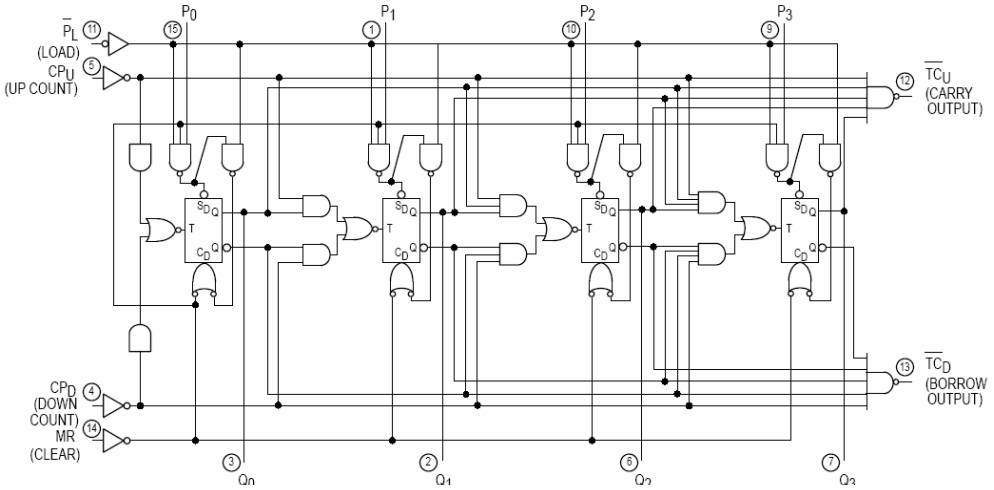
CP_U	Count Up Clock Pulse Input
CP_D	Count Down Clock Pulse Input
$\overline{MR}$	Asynchronous Master Reset (Clear) Input
$\overline{PL}$	Asynchronous Parallel Load (Active LOW) Ir
P_n	Parallel Data Inputs
$\overline{Q_n}$	Flip-Flop Outputs (Note b)
$\overline{TC_D}$	Terminal Count Down (Borrow) Output (Note b)
TC_U	Terminal Count Up (Carry) Output (Note b)

MODE SELECT TABLE

MR	$\overline{PL}$	CP_U	CP_D	MODE
H	X	X	X	Reset (Asyn.)
L	L	X	X	Preset (Asyn.)
L	H	H	H	No Change
L	H	⎍	H	Count Up
L	H	H	⎍	Count Down

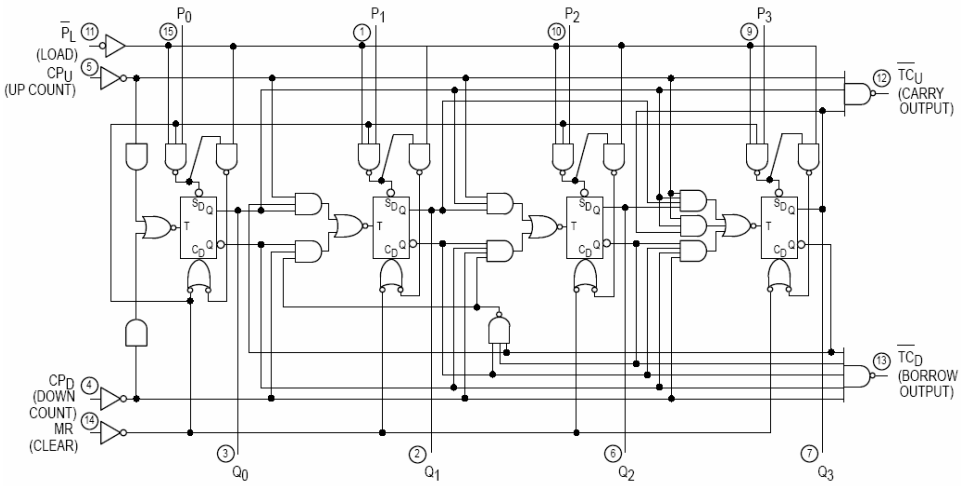
LOGIC DIAGRAMS (continued)

LS193

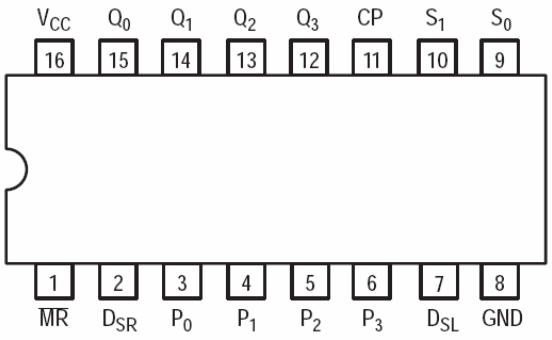


LOGIC DIAGRAMS

LS192



SN74LS194A
4-Bit Bidirectional Universal Shift Register

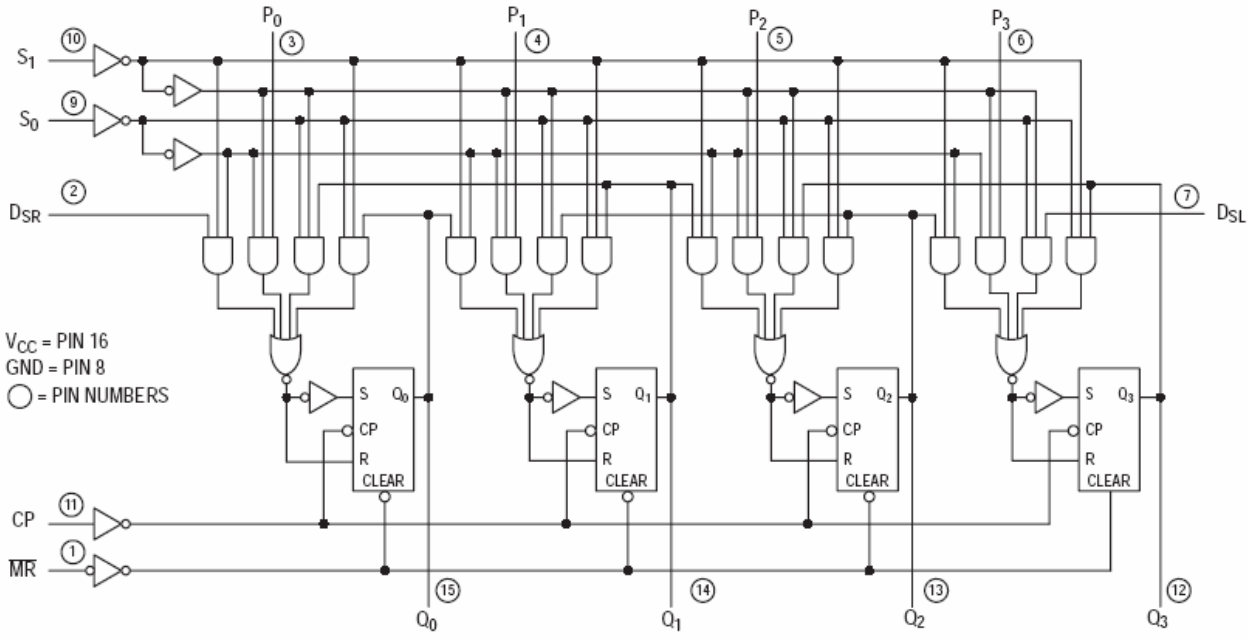


S_0, S_1	Mode Control Inputs
$P_0 - P_3$	Parallel Data Inputs
D_{SR}	Serial (Shift Right) Data Input
D_{SL}	Serial (Shift Left) Data Input
CP	Clock (Active HIGH Going Edge) Input
$\overline{MR}$	Master Reset (Active LOW) Input
$Q_0 - Q_3$	Parallel Outputs

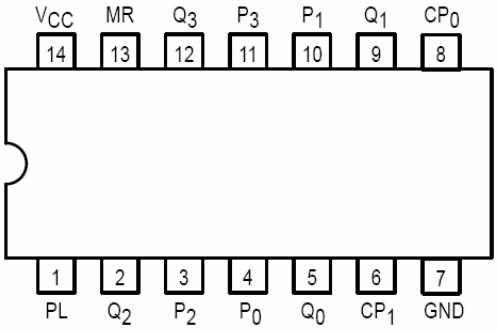
MODE SELECT — TRUTH TABLE

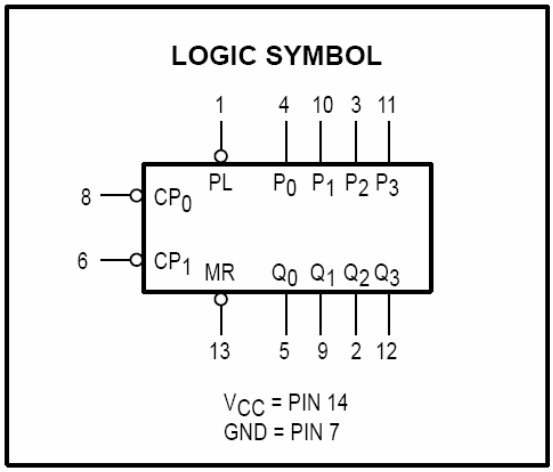
OPERATING MODE	INPUTS						OUTPUTS			
	$\overline{MR}$	S_1	S_0	D_{SR}	D_{SL}	P_n	Q_0	Q_1	Q_2	Q_3
Reset	L	X	X	X	X	X	L	L	L	L
Hold	H	l	l	X	X	X	q_0	q_1	q_2	q_3
Shift Left	H H	h h	l l	X X	l h	X X	q_1 q_1	q_2 q_2	q_3 q_3	L H
Shift Right	H H	l l	h h	l h	X X	X X	L H	q_0 q_0	q_1 q_1	q_2 q_2
Parallel Load	H	h	h	X	X	P_n	P_0	P_1	P_2	P_3

LOGIC DIAGRAM



SN 74197, SN74S197, SN74LS197
50/30/100MHz PRESETTABLE DECADE OR BINARY COUNTERS/LATCHES

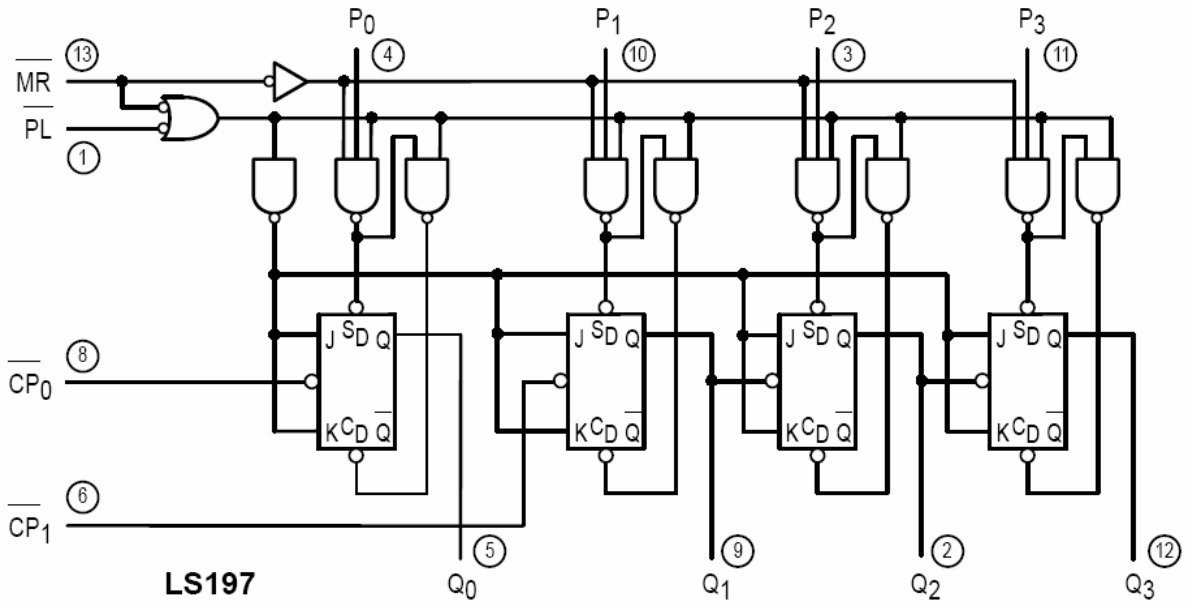




$\overline{CP_0}$	Clock (Active LOW Going Edge) Input to Divide-by-Two Section
$\overline{CP_1}$ (LS196)	Clock (Active LOW Going Edge) Input to Divide-by-Five Section
$\overline{CP_1}$ (LS197)	Clock (Active LOW Going Edge) Input to Divide-by-Eight Section
$\overline{MR}$	Master Reset (Active LOW) Input
$\overline{PL}$	Parallel Load (Active LOW) Input
P_0–P_3	Data Inputs
Q_0–Q_3	Outputs (Notes b, c)

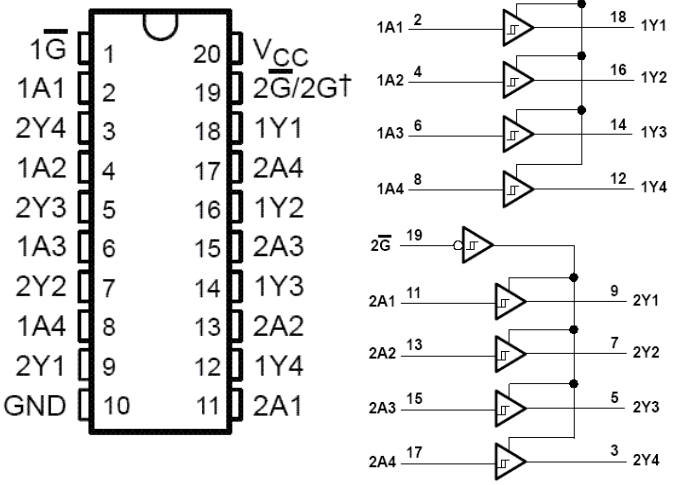
MODE SELECT TABLE

INPUTS			RESPONSE
$\overline{MR}$	$\overline{PL}$	$\overline{CP}$	
L	X	X	Reset (Clear)
H	L	X	Parallel Load
H	H	⌐	Count

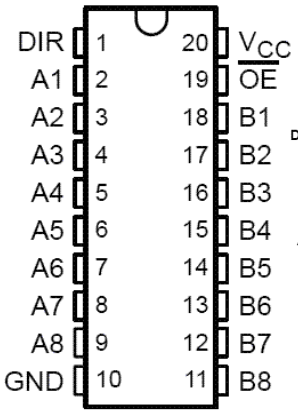


SN74LS240, SN74LS241, SN74LS244, SN74S240, SN74S241, SN74S244
OCTAL BUFFERS AND LINE DRIVERS WITH 3-STATE OUTPUTS

'LS244, 'S244

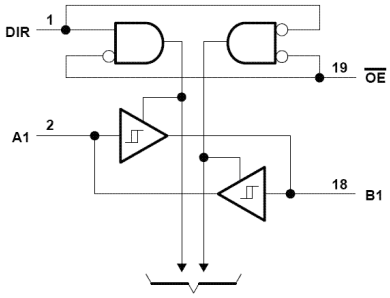


SN54LS245, SN74LS245
OCTAL BUS TRANSCEIVERS WITH 3-STATE OUTPUTS

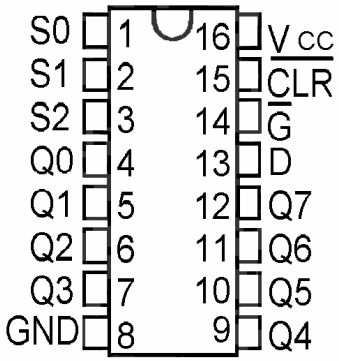


FUNCTION TABLE

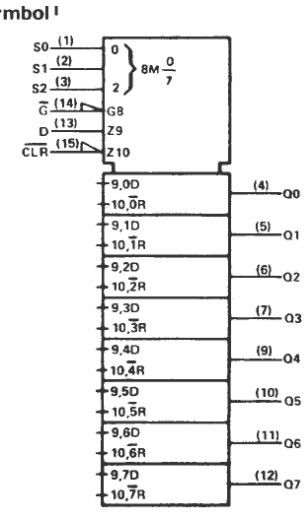
INPUTS		OPERATION
$\overline{OE}$	DIR	
L	L	B data to A bus
L	H	A data to B bus
H	X	Isolation



SN74259, SN74LS259B
8-BIT ADDRESSABLE LATCHES



logic symbol†



†This symbol is in accordance with ANSI/IEEE Std. 91-1984 and IEC Publication 617-12.
Pin numbers shown are for D, J, N, and W packages.

FUNCTION TABELE

INPUTS		OUTPUT OF ADDRESSED LATCH	EACH OTHER OUTPUT	FUNCTION
$\overline{CLR}$	$\overline{G}$			
H	L	D	Q_{i0}	Addressable Latch
H	H	Q_{i0}	Q_{i0}	Memory
L	L	D	L	8-Line Demultiplexer
L	H	L	L	Clear

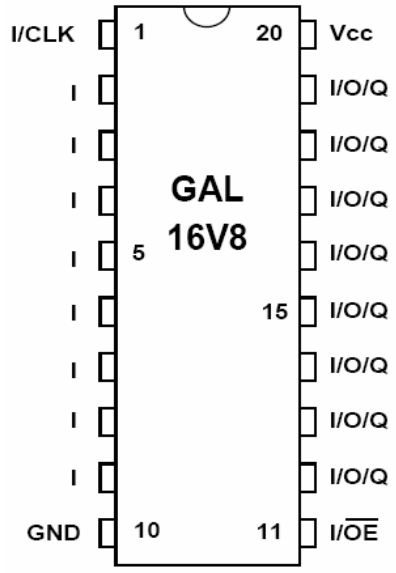
H = high level, L = low level
D = the level at the data input
Q = the level of Q (i = 0,1, ...,7 as appropriate) before the indicated steady-state input conditions established

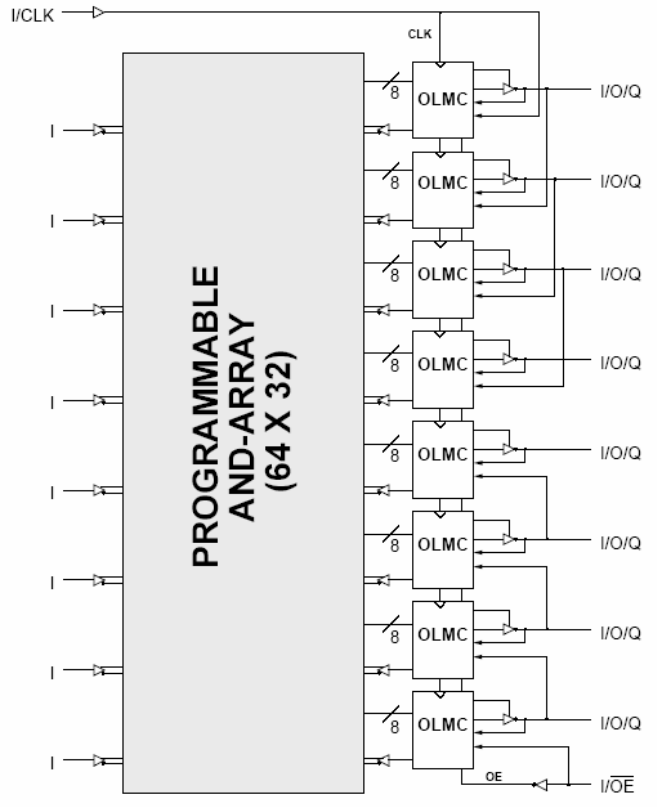
LATCH SELSCTION TABLE

SELSCT INPUTS			LATCH ADDRESSED
S2	S1	S0	
L	L	L	0
L	L	H	1
L	H	L	2
L	H	H	3
H	L	L	4
H	L	H	5
H	H	L	6
H	H	H	7

GAL16V8/883

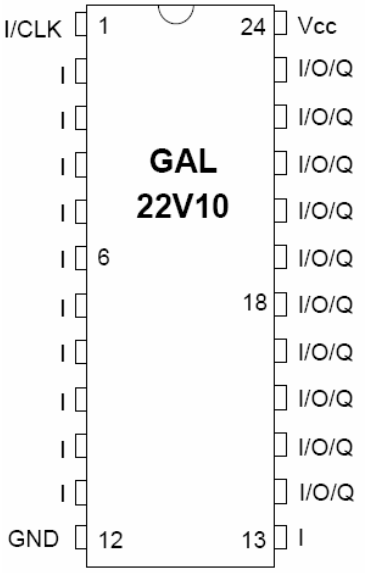
High Performance E²CMOS PLD
Generic Array Logic™

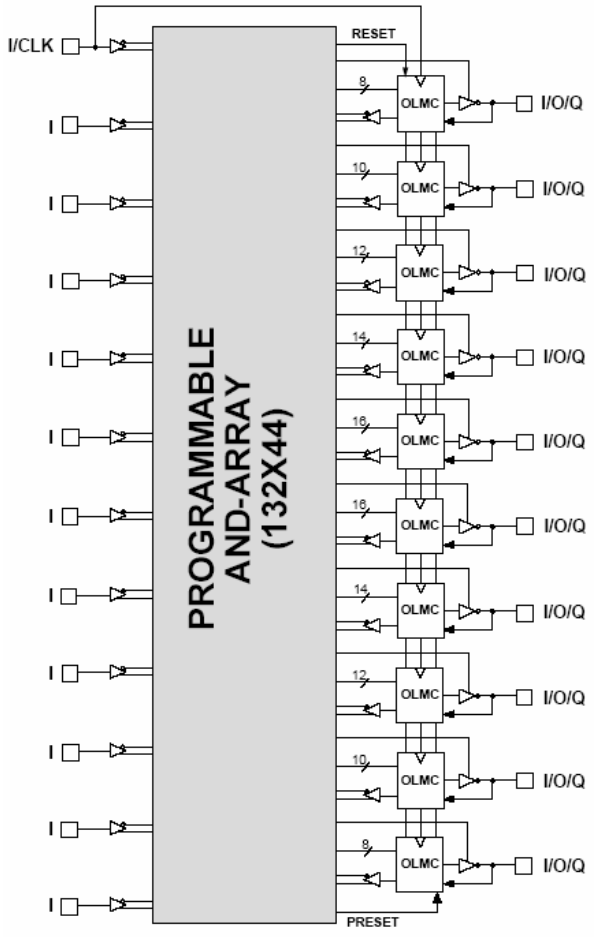




GAL22V10/883

High Performance E²CMOS PLD
Generic Array Logic™





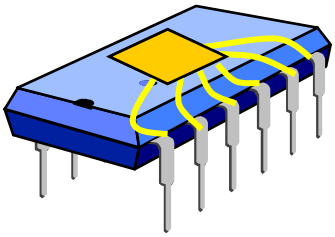
3.4. Literatura do ćwiczeń laboratoryjnych

Literatura podstawowa:

1. Dziuda A., Krupa W.: *Laboratorium Techniki Cyfrowej*. Introligatorstwo – Mała Poligrafia. Jelenia Góra 2006.
2. Łakomy M., Zabrodzki J.: *Cyfrowe układy scalone*. WNT. Warszawa 1980.
3. Pieńkos J., Turczyński J.: *Układy scalone TTL w systemach cyfrowych*. WKiŁ. Warszawa 1986.
4. Sasal W.: *Układy scalone serii UCY 74LS i UCY 74S. Parametry i zastosowania*. WKiŁ. Warszawa 1987.
5. Sasal W.: *Układy scalone serii UCA 74/ UCY 74. Parametry i zastosowania*. WKiŁ. Warszawa 1985.

Literatura uzupełniająca:

1. *Układy logiczne. Ćwiczenia laboratoryjne*. [red:] Łukowicz M. Oficyna Wydawnicza Politechniki Wrocławskiej. Wrocław 2002.
2. ATMEL - WinCUPL. User's Manual. Copyright Atmel Corporation 1997.
3. Advantech Equipment Corp.: LABTOOL-48XP. Intelligent Universal Programmer. User's Manual. Printed in Taiwan. July 2002.



4. NARZĘDZIA WSPOMAGAJĄCE PROJEKTOWANIE UKŁADÓW CYFROWYCH W STRUKTURACH PLD

4.1. Programy systemu CUPL

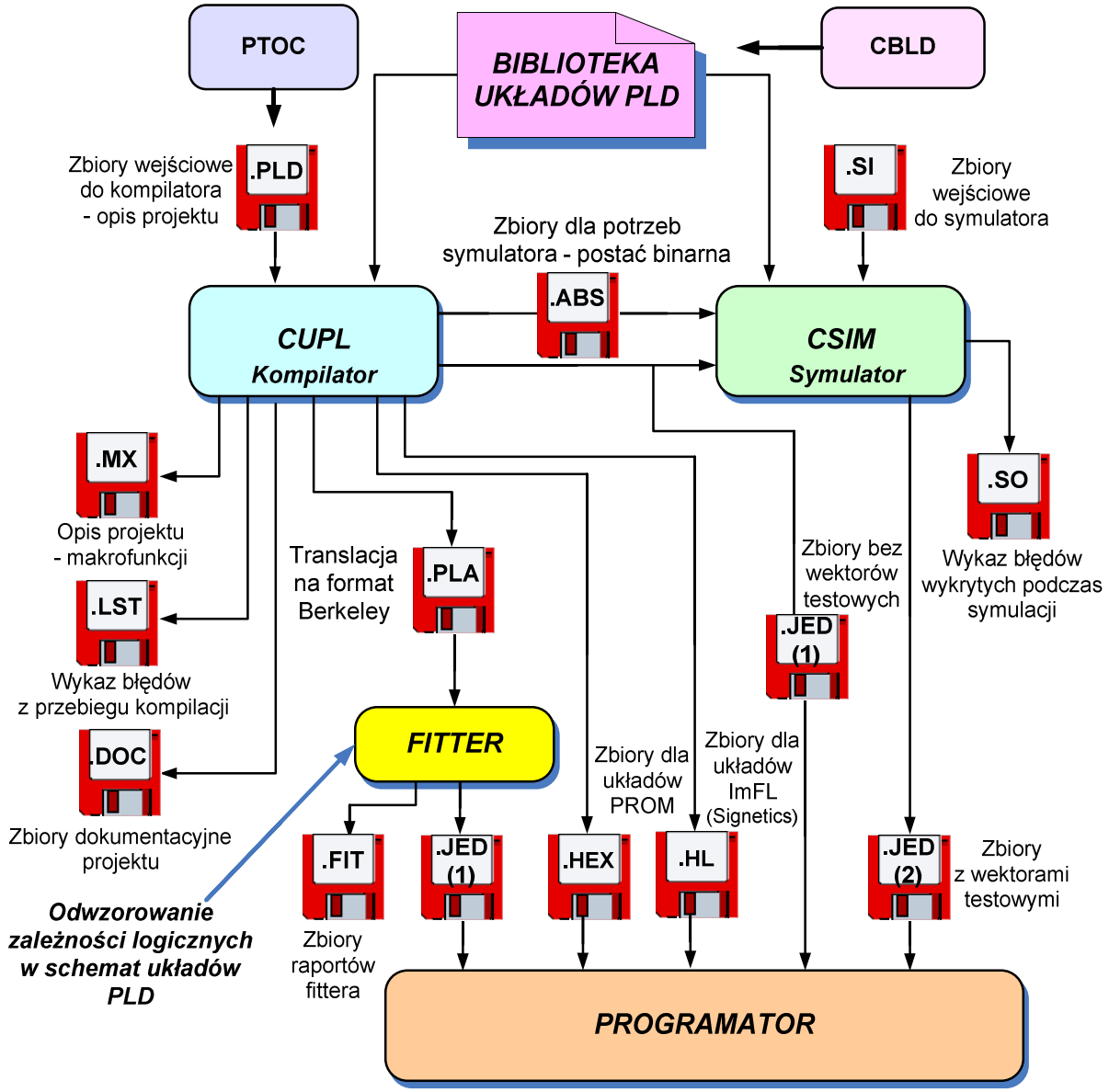
System CUPL (*Compiler Universal for Programmable Logic*) jest kompilatorem firmy LOGICAL DEVICES INC. Podstawowe jego zalety to:

- uniwersalność - możliwość zastosowania różnych typów układów PLD i pamięci pochodzących od różnych producentów;
- prosta składnia - maksymalne uproszczenie języka przy zachowaniu możliwie dużych możliwości;
- samodokumentowanie się - konstrukcja pliku opisu logicznego nie wymaga żadnych dodatkowych opisów;
- łatwość współpracy z programatorami – mapa przepaleń tworzona jest m.in. w formacie *jedec*, który jest najczęściej wykorzystywanym formatem w programatorach.

System CUPL tworzą następujące programy (rysunek 4.1):

- CUPL (*Universal Compiler for Programmable Logic*) - uniwersalny kompilator do programowalnych układów. Pozwala on na translację zbiorów dyskowych zapisanych w języku specyfikacji zadania na zbiory zawierające rozkazy dla programatorów programowalnych układów logicznych.

- CSIM (*CUPL Simulator*) - symulator układów logicznych zaprogramowanych kodem wytworzonym przez kompilator CUPL.
- CBLD (*CUPL Build*) - program zarządzający bibliotekami układów logicznych dla kompilatora CUPL.
- PTOC (*PALASM to CUPL Translator*) - program do tłumaczenia zbiorów dyskowych zapisanych w formacie programu PALASM na format wejściowy kompilatora CUPL.



Rys. 4.1. Schemat współpracy programów systemu CUPL.

Zbiory wejściowe do kompilatora (rozszerzenie *.PLD) i symulatora (rozszerzenie *.SI) są zbiorami zawierającymi wyłącznie znaki ASCII.

Zbiory wyjściowe mogą być wyprowadzone w formatach:

- *.JED - plik wejściowy dla programatora z wektorami testowymi (2) lub bez (1);
- *.HEX - plik wejściowy dla programatora układów PROM;
- *.HL - plik wejściowy dla programatora układów ImFL firmy Signetics;
- *.DOC - plik dokumentacji - zawiera rozszerzone równania logiczne i tablice symboli zmiennych;
- *.LST - plik błędów - lista błędów kompilacji;
- *.SO - plik błędów wykrytych podczas symulacji;
- *.MX - plik zawierający rozwinięcie makr.

Kompilator CUPL może służyć jako program translacji na format Berkeley (*.PLA) oraz dodatkowo produkuje zbiór dla potrzeb symulatora (*.ABS).[1]

4.2. Podstawy języka CUPL

Język CUPL służy do tekstowego opisu układu cyfrowego. Umożliwia jego specyfikację przez podanie równań logicznych lub przez opis procedury przy użyciu instrukcji własnych. Dodatkowe możliwości organizacji tekstu źródłowego udostępnia preprocesor podobny w składni do preprocesora języka C. Podczas tworzenia projektu dobrze jest zastosować metodę analizy: „góra-dół”. Oznacza to, że należy zacząć od ogólnego zarysu projektu, a później przesuwać się cały czas w głąb, definiując kolejne elementy.

[1] Łuba T., Markowski M. A., Zbierzchowski B. Kompilatory układów logicznych. Oficyna Wydawnicza PW. Warszawa 1995.

Tekst opisu przechowywany jest w niesformatowanych plikach tekstowych o rozszerzeniu *.PLD. Do symulacji niezbędny jest dodatkowo plik o rozszerzeniu *.SI.

4.2.1. Elementy składni języka

Komentarze

Komentarze można zamieszczać w pliku źródłowym. Rozpoczynają się i kończą znakami `/*  */`. Tekst umieszczony między tymi symbolami traktowany jest jako komentarz. Znaki te mogą obejmować jedną i więcej linii - koniec linii nie oznacza końca komentarza.

Przykłady:

1.
```
/*****************************************/
/*  To jest przyklad tworzenia tytulu lub  */
/*  bloku informacyjnego pliku zrodlowego*  /
/*****************************************/
```

2.
```
/*
Inny przyklad tworzenia bloku informacyjnego
*/
```

3.
```
out1=in1 # in2;       /* Funkcja OR  */
out2=in1 & in2;       /* Funkcja AND */
out3=in1 $ in2;       /* Funkcja XOR */
```

Komentarz jest ignorowany przez kompilator. Plik źródłowy nie powinien zawierać kodów rozszerzonych (m.in. polskich znaków).

Zmienne

Zmienne są ciągami znaków alfanumerycznych (cyfry, litery, podkreślenia) zawierające co najmniej jedną literę. Określają one wyprowadzenia układu *(pin)*, wewnętrzne węzły *(node)*, stałe, sygnały wejściowe, wyjściowe oraz pośrednie lub zbiory sygnałów. Nazwy zmiennych mogą składać się maksymalnie z 31 znaków (przy dłuższych nazwach są one obcinane do 31 zna-

ków). W języku CUPL wyróżnia się duże i małe litery. W skład nazwy zmiennej nie może wchodzić spacja. Nazwy zmiennych nie mogą być nazwami zarezerwowanych symboli języka CUPL (tabela 4.1) oraz nazwami zastrzeżonymi słów kluczowych (tabela 4.2).

Tabela 4.1. Zarezerwowane symbole języka CUPL.

&	#	(	)	-
*	+	[	]	/
:	.	..	/*	*/
;	,	!	'	=
@	$	^	%	

Tabela 4.2. Zarezerwowane słowa kluczowe języka CUPL.

APPEND	ASSEMBLY	ASSY	COMPANY
CONDITION	DATE	DEFAULT	DESIGNER
DEVICE	ELSE	FIELD	FLD
FORMAT	FUNCTION	FUSE	GROUP
IF	JUMP	LOC	LOCATION
MACRO	MIN	NAME	NODE
OUT	PARTNO	PIN	PINNODE
PRESENT	REV	REVISION	SEQUENCE
SEQUENCED	SEQUENCEJK	SEQUENCERS	SEQUENCET
TABLE			

Przykłady poprawnych nazw zmiennych:

```
a0,
A0,
8250_ENABLE,
Real_time_clock_interrupt_address;
```

Zmienne indeksowe

Są to zmienne zawierające w nazwie numer. Zakończone zawsze liczbą dziesiętną z zakresu od 0 do 31. Zmiennymi takimi mogą być nazwy opisujące linie adresowe, linie danych lub inne kolejno numerowane elementy. Indeksowanie najlepiej zaczynać od zera. Zmienna z indeksem 0 zawsze oznacza najmniej znaczący bit.

Przykłady poprawnych zmiennych indeksowych:

```
a23,
D07,
D7,
counter_bit_3.
```

Zmienne D07 i D7 są różnymi zmiennymi.

Listy

Listy są obiektami umożliwiającymi skrócony zapis grupy zmiennych. Są one powszechnie używane przy deklaracji wielu wyprowadzeń i węzłów, deklaracji pól bitowych, równań logicznych i zbioru operacji. Nawiasy prostokątne `[ ]` ograniczają zasięg deklaracji listy.

Format listy ma postać:

$[zm_1, zm_2, \ldots zm_n]$

Gdy zmienne są indeksowe, można zastosować skróconą notację;

$[zm_{m..n}]$

gdzie: `m` - pierwszy indeks w liście,

`n` – ostatni indeks w liście.

Przykłady:

```
[UP, DOWN, LEFT, RIGHT]
[A0, A1, A2, A3, A4, A5, A6, A7]
[A0..2, A3, A4, A5..7]
[A0..7, B0..3]
```

Stałe liczbowe

Liczby mogą być reprezentowane w jednym z czterech dostępnych formatów: binarnym - ‘b’, ósemkowym - ‘o’, dziesiętnym - ‘d’ oraz szesnastkowym - ‘h’. W pliku źródłowym przyjmuje się przez domniemanie, że używane stałe są w zapisie szesnastkowym, oprócz liczb określających numery wyprowadzeń (pin) i indeksów zmiennych indeksowych, dla których przyjmuje się, że są w systemie dziesiętnym.

Wszystkie operacje na liczbach realizowane są z dokładnością 32-bitową, daje to zakres wartości od 1 do 2^{32}- 1. W zapisach dwójkowych, ósemkowych i szesnastkowych liczby w swym zapisie mogą zawierać znak X - wartość nieoznaczona. Przykłady zapisu liczb w czterech formatach przedstawia tabela 4.3.

Tabela 4.3. Zapis liczb w języku CUPL.

Format	Liczba	Wartość dziesiętna
Dwójkowy	`‘b’0` lub `‘B’0`	0
Dwójkowy	`‘b’1101` lub `‘B’1101`	13
Ósemkowy	`‘o’663` lub `‘O’663`	435
Ósemkowy (zakres wartości)	`‘O’[300..477]`	192..314
Dziesiętny	`‘d’79` lub `‘D’79`	79
Szesnastkowy	`‘h’BA` lub `‘H’BA`	186

Deklaracje

Deklaracje wyprowadzeń z układu `pin`

Deklaracja `pin` przypisuje nazwy (zmienne) dla sygnału na wyprowadzeniu układu scalonego o podanym numerze oraz określa jego polaryzację.

Składnia ma postać:

```
PIN pin_n=[!]zm;
```

gdzie: `PIN` - wymagane słowo kluczowe,

`pin_n` - zapisany dziesiętnie numer wyprowadzenia obudowy,

`zm` - nazwa (zmienna) przypisanego sygnału,

`!` - definiowanie polaryzacji sygnału wejściowego lub wyjściowego, polaryzacja ujemna,

`=` - operator przypisania, który powoduje przypisanie wartości wyrażenia zmiennej lub liście zmiennych,

Przykłady:

```
pin 1 =   clock;  /* Register Clock */
pin 2 =   !enable; /* Enable I/O Port */
pin [3,4] =   ![stop,go];  /* Control Signals */
pin [5..7] =   [a0..2]; /* Address Bits 0-2 */
```

W deklaracji wyprowadzeń z układu nie podaje się, czy dane wyprowadzenie jest wejściem, wyjściem albo jest dwukierunkowe.

Deklaracje węzłów `node, pinnode`

Niektóre układy PLD zawierają zasoby wewnętrzne, które generują funkcje logiczne, ale nie są dostępne bezpośrednio z wyprowadzeń zewnętrznych układu. Dostęp do nich jest możliwy przez zdefiniowanie węzłów wewnętrznych (*buried*).

W celu zdeklarowania zmiennych dla ukrytych funkcji używa się słowa kluczowego NODE, którego format jest następujący:

```
NODE [!] zm;
```

gdzie: `NODE` - wymagane słowo kluczowe,

`!` – opcjonalnie, określa polaryzację sygnału,

`zm` – nazwa (zmienna) przypisanego sygnału.

Przykład:

```
NODE [State0..5];  /* wewnetrzne bity stanu */
NODE !Invert;      /* For Complement Array */
```

W celu wyraźnego zdefiniowania ukrytych węzłów używa się słowa kluczowego PINNODE. Umożliwia ono połączenie węzła o numerze n ze zmienną. Format deklaracji jest następujący:

```
PINNODE node_n = [!]zm;
```

gdzie: `PINNODE` - wymagane słowo kluczowe,

`node_n` – numer węzła w zapisie dziesiętnym,

`!` - opcjonalnie, określa polaryzację sygnału,

`zm` - nazwa (zmienna) przypisanego sygnału.

`=` - operator przypisania, który powoduje przypisanie wartości wyrażenia zmiennej lub liście zmiennych,

Przykład:

```
PINNODE [29..34] = [State0..5];
PINNODE 35 = !Invert;
```

Deklaracje pól bitowych

Deklaracja pola bitów łączy pojedynczą zmienną z grupą bitów. Gdy użyte zostanie słowo kluczowe `FIELD`, kompilator generuje pojedynczą 32-bitową zmienną, która jest używana do reprezentacji zmiennych w polu bitowym. Składnia ta jest głównie używana do definicji adresów i szyn danych. Format jest następujący:

```
FIELD zm = [zm1, zm2, ... zmn];
```

Przykład:

```
FIELD count=[Q3..0];
FIELD mode =[clr,dir];
```

Zmienna utworzona za pomocą słowa kluczowego FIELD może zawierać maksymalnie 32 bity.

Operatory logiczne

Operatorami logicznymi można działać na zmienne i listy - wynikami są odpowiednio zmienne i listy. W języku CUPL dostępne są cztery standardowe operatory logiczne: NOT, AND, OR, XOR.

Sposób zapisu operacji logicznych oraz kolejność wykonania przedstawia tabela 4.4.

Tabela 4.4. Operatory logiczne.

Operator	Opis	Przykład	Kolejność
!	NOT- negacja (dopełnienie)	!A	1
&	AND - koniunkcja	A&B	2
#	OR - alternatywa	A#B	3
$	XOR - różnica symetryczna	A$B	4

Przykłady:

bitowe operatory logiczne.

Wyrażenie:

```
[A0, Al, A2, A3] # [B0, B1, B2, B3];
```

jest równoważne wyrażeniu:

```
[AO # B0, A1 # B1, A2 # B2, A3 # B3];
```

listowe operatory logiczne.

Wyrażenie:

```
[D0, D1, D2, D3] &;
```

jest równoważne wyrażeniu ;

```
D0 & D1 & D2 & D3;
```

Operatory arytmetyczne i równań

Operatorami arytmetycznymi można działać na zmienne i listy - wynikami są odpowiednio zmienne i listy. W języku CUPL dostępnych jest sześć operatorów arytmetycznych: potęgowanie, mnożenie, dzielenie, moduł, dodawanie, odejmowanie.

Sposób zapisu operacji arytmetycznych oraz kolejność wykonania przedstawia tabela 4.5.

Tabela 4.5. Operatory arytmetyczne i równań.

Operator	Opis	Przykład	Kolejność
**	Potęgowanie	2**3	1
*	Mnożenie	2*4	2
/	Dzielenie	6/3	2
%	Modulo	7%5	2
+	Dodawanie	4+5	3
-	Odejmowanie	6-4	3

Równania logiczne

Równania logiczne umożliwiają połączenie zacisków wejściowych z wyjściowymi poprzez zaprojektowane funkcje. Sposób zapisu równań jest następujący:

```
[!]zm[.ext] = wyr;
```

gdzie: `zm` - pojedyncza zmienna lub lista zmiennych, zdefiniowana zgodnie z zasadami notacji listowej,

`.ext` - opcjonalne rozszerzenie umożliwiające przyłączenie funkcji do większości węzłów wewnątrz programowalnych układów,

`wyr` - wyrażenie będące kombinacją zmiennych i operatorów logicznych,

`=` - operator przypisania, który powoduje przypisanie wartości wyrażenia zmiennej lub liście zmiennych,

`!` – definiowanie polaryzacji sygnału, w tym kontekście jest to zanegowanie wyrażenia stojącego po prawej stronie znaku.

Jeśli `zm` nie jest nazwą (zmienną) zadeklarowaną uprzednio w instrukcji PIN i NODE, kompilator automatycznie przyjmie, że równanie definiuje nową zmienną wewnętrzną, do której można się odwołać w innych wyrażeniach logicznych.

```
Q0.D=Q1 & Q2 & Q3; /* Wyjscie w D flip-flop */
```

```
Q1.J = Q2 # Q3;     /* Wyjscie w JK flip-flop */
Q1.K = Q2 & !Q3;
MREQ=READ # WRITE; /* Zmienna posrednia */
[D0..3] = 'h'FF;   /* Przyporzadkowanie wartosci
                      stałej */
```

W miejsce zm może w równaniu wystąpić notacja listowa; w takim wypadku równanie logiczne stosuje się do każdego elementu listy indywidualnie, np.:

Przykłady:

```
[Q2..0].OE=!PinOE; ⇔   Q2.OE=!PinOE;
                       Q1.OE=!PinOE;
                       Q0.OE=!PinOE;
```

Instrukcja APPEND

Standardowo w równaniach logicznych tylko jedno wyrażenie może być połączone z jedną zmienną. Słowo kluczowe APPEND umożliwia połączenie jednej zmiennej z wieloma wyrażeniami. Sposób używania APPEND jest identyczny do definiowanych równań logicznych, z tym że słowo kluczowe występuje na początku każdego równania. Instrukcja ma postać:

```
APPEND [!]zm[.ext] = wyr;
```

Wynikiem stosowania słowa APPEND jest suma logiczna OR dla wszystkich wyrażeń.

Przykład:

```
 APPEND Y = A0 & A1;
 APPEND Y = B0 & B1;
 APPEND Y = C0 & C1;
```

Powyższe trzy równania są równoważne równaniu:

```
 Y = A0 & A1 # B0 & B1 # C0 & C1;
```

Operacje na zbiorach

Wszystkie operacje mogą być wykonywane na pojedynczych bitach (przykładowo na sygnałach wejściowych) lub na informacjach zgrupowanych w zbiory. Operacje na zbiorach można stosować pomiędzy zbiorem a zmienną lub wyrażeniem, lub pomiędzy dwoma zbiorami. Wynikiem operacji między zbiorem a zmienną (lub wyrażeniem) jest nowy zbiór, w którym operacje są wykonywane pomiędzy każdym elementem i zmienną (lub wyrażeniem). Jeżeli operacja wykonywana jest na dwóch zbiorach, muszą mieć one te same rozmiary (ilość elementów). W wyniku powstaje nowy zbiór złożony z elementów stanowiących wynik operacji pomiędzy elementami każdego ze zbiorów.

Przykład:

Wyrażenie

```
[D0, D1, D2, D3] & read
```

jest równoważne

```
[D0 & read, D1 & read, D2 & read, D3 & read]
```

Gdy operacje są wykonywane pomiędzy dwoma zbiorami, zbiory muszą mieć taki sam wymiar (tzn. taką samą liczbę elementów). Wynikiem operacji między zbiorami jest nowy zbiór, w którym operacje są wykonywane pomiędzy kolejnymi elementami każdego ze zbiorów.

Przykład:

Wyrażenie:

```
[A0, A1, A2, A3] & [B0, B1, B2, B3]
```

jest równoważne

```
[A0 & B0, A1 & B1, A2 & B2, A3 & B3]
```

Do grupowania zmiennych w zbiór, który może być identyfikowany poprzez jedną zmienną służy słowo kluczowe FIELD.

Przykład:

```
FIELD a_inputs = [A0, A1, A2, A3];
```

```
FIELD b_inputs = [B0, B1, B2, B3];
```

Operacja iloczynu logicznego pomiędzy zbiorami wygląda następująco:

```
a_inputs & b_inputs
```

Przy operacjach pomiędzy zbiorami i liczbami, liczby są traktowane jako maska bitów.

Przykładowo przedstawiono równania logiczne dla 4-bitowego licznika:

```
FIELD count=[Q3..0];
cound.d = ‘b’0001 & (!Q0)
 # ‘b’0010 & (Q1 $ Q0)
 # ‘b’0100 & (Q2 $ Q1 & Q0)
 # ‘b’1000 & (Q3 $ Q2 & Q1 &Q0);
```

Są one równoważne równaniom:

```
Q0.d = ! Q0;
Q1.d = Q1 $ Q0;
Q2.d = Q2 $ Q1 & Q0;
Q3.d = Q3 $ Q2 & Q1 & Q0;
```

Operator równości

Operator równości pozwala sprawdzić równość pomiędzy zbiorami zmiennych a stałą na poziomie poszczególnych pozycji bitowych. Realizuje pojedyncze wyrażenie logiczne. Pozycje binarne w zmiennej są porównywane z odpowiadającymi im pozycjami w zbiorze. Gdy pozycja binarna w zmiennej jest równa 1, element w zbiorze pozostaje niezmieniony, gdy jest równa 0 element zostaje zanegowany, gdy jest równa binarnemu X element zostaje usunięty. Wynikowe elementy zostają przekształcone w postać koniunkcji, celem stworzenia pojedynczych wyrażeń. Format operatora równości jest następujący:

1. [zm_1, zm_2, ... , zm_n]: constant;
2. bit_fileld_zm: constant;

gdzie:

[zm_1, zm_2, ... , zm_n] - zbiór zmiennych,

constant - stała (standardowo w kodzie szesnastkowym),

bit_field_zm - zmienna zdefiniowana za pomocą słowa kluczowego FIELD,

: - operator równości.

Przykład:

Wyrażenia

```
1. select = [A3..0]:’h’D;
2. select = [A3..0]:'b'1X0X;
```

są równoważne

```
1. select = A3 & A2 & !A1 & A0 ;
2. select = A3 & !A1 ;
```

Operator równości można również wykorzystać, gdy operacje wykonywane są na identyczności. Format jest następujący:

```
[zm, zm,..., zm]:op;
```

co jest równoważne

```
zm op zm op ... zm;
```

gdzie: op - operator logiczny,

zm – zmienna.

Przykład:

Wyrażenia

```
[A3,A2,A1,A0]:&
[B3,B2,B1,B0]:#
[C3,C2,C1,C0]:$
```

są równoważne wyrażeniom

```
A3 & A2 & A1 & A0
B3 # B2 # B1 # B0
C3 $ C2 $ C1 $ C0
```

Operator równości może być także stosowany do tworzenia funkcji tablicowej wyjściowych wartości. Dla przykładu przedstawiono zapis równań logicznych tabeli stanów zapisanej w tabeli 4.6.

Tabela 4.6. Tabela prawdy.

in2	in1	in0	out1	out0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	1	1
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

Przykład:

```
FIELD  input = [in2..0];

FIELD  output = [out1, out0] ;
output=input:0&[0,0]#input:1&[0,1]#input:2&[1,0]&input:3[0,1]
      #input:4&[1,1]#input:5&[0,1]#input:6&[1,0]&input:7[1,1]
```

Operacje na zakresach liczb

Operacje na zakresach liczb są podobne do operatora równości, przy czym stałą zastępuje zbiór stałych o określonym zakresie ograniczonym nawiasami [ ]. Operacja ta sprawdza równość pomiędzy zbiorem zmiennych a każdą ze stałych znajdujących się w obrębie rozważanego zakresu. Format jest następujący:

1. $[zm_1, zm_2, ..., zm_n]$:[stala_lo..stala_hi];
2. bit_field_zm:[stala_lo..stala_hi];

gdzie: $[zm_1, zm_2, ..., zm_n]$ - zbiór zmiennych,

stala_lo, stala_hi - liczby definiujące zakres operacji.

Przykład:

```
FIELD address =[A3..0]; /* definicja szyny adre-
                           sowej */
```

Wyrażenie

```
select = address:[C..F]; /* rownanie zakresu */
```

jest równoważne

```
select = address:C # address:D # address:E # ad-
dress:F;
```

co można rozwinąć do postaci:

```
    select = A3 & A2 & !A1 & !A0
           # A3 & A2 & !A1 & A0
           # A3 & A2 & A1 & !A0
           # A3 & A2 & A1 & A0;
```

Tablice prawdy

W niektórych przypadkach najłatwiejszym sposobem zapisu równań logicznych jest zastosowanie metody opisowej, czyli tablicy prawdy. CUPL umożliwia za pomocą słowa kluczowego TABLE utworzyć taką tablicę. W tablicy prawdy należy zdefiniować zależności pomiędzy kombinacjami zmiennych wejściowych i wyjściowych, a następnie sprecyzować jednoznaczne powiązania pomiędzy zdekodowanymi wartościami wejścia i listą zmiennych wyjściowych. Format użycia operatora TABLE jest następujący:

```
TABLE zm_list_1 ⇒ zm_list_2
{
input_0 => output_0;
.........................
input_n => output_n;
}
```

gdzie: zm_list_1 - określenie zmiennych wejściowych,

`zm_list_2` - określenie zmiennych wyjściowych,

`input_n` - wartość lub lista wartości zmiennych wejściowych odpowiadających stanowi n,

`output_n` - wartość lub lista wartości zmiennych wyjściowych odpowiadających stanowi n,

`=>` - operator powiązania zmiennych wejściowych z wyjściowymi.

Przykład:

opis dekodera kodu szesnastkowego na kod BCD

```
FIELD  input = [in3..0] ;

FIELD  output = [out4..0] ;

TABLE  input => output

{

0=>00; 1=>01;  2=>02;  3=>03;

4=>04; 5=>05;  6=>06;  7=>07;

8=>08; 9=>09;  A=>10;  B=>11;

C=>12; D=>13;  E=>14;  F=>15;

}
```

Instrukcja CONDITION

Słowo kluczowe `CONDITION` umożliwia opis funkcji logicznych na wyższym poziomie niż pozwala na to pisanie standardowych równań logicznych. Instrukcja ma postać:

```
CONDITION
{
IF wyr0 OUT  zm0;
      .
      .
```

```
  IF wyrn OUT  zmn;
  DEFAULT  OUT  zmM ;
  }
```

gdzie: wyrX - wyrażenie logiczne złożone ze zmiennych wejściowych funkcji,

zmX - nazwa (zmienna) zadeklarowana w instrukcji PIN lub grupa takich nazw (zmiennych) zapisanych w notacji listowej,

Zapis taki oznacza, że zmienna zmX ma być ustawiona w stan aktywny (jeden lub zero w zależności od polaryzacji podanej w deklaracji PIN), gdy spełniony jest warunek wyrX; jeżeli żaden z warunków nie jest spełniony, układ ustawia wyjścia wg opcjonalnej instrukcji DEFAULT.

Przykład:

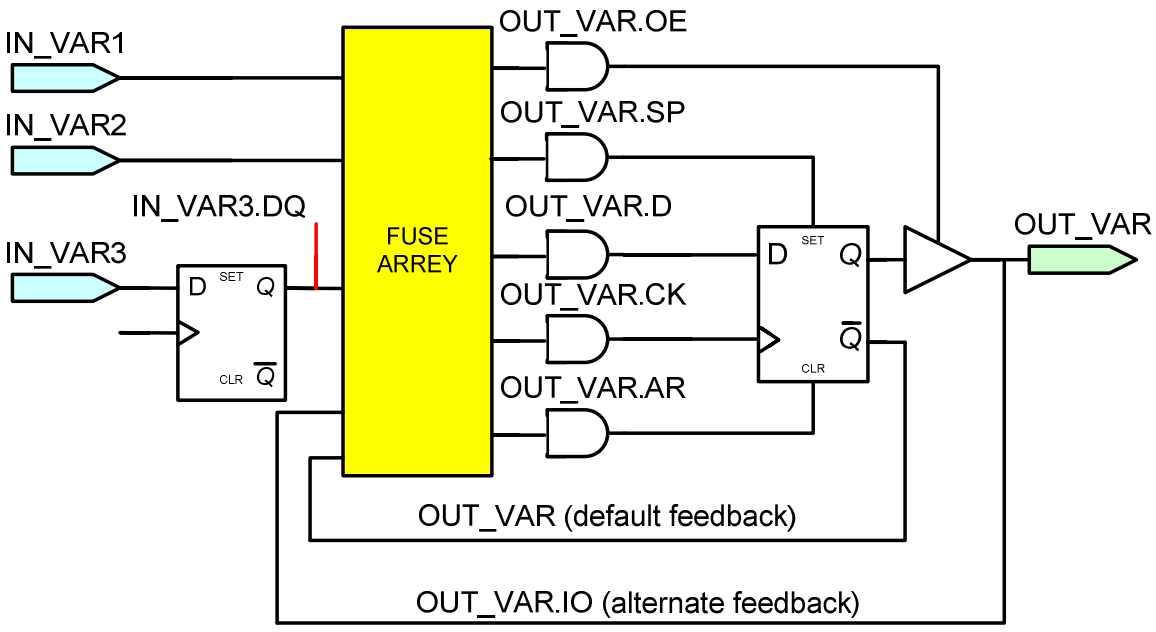
opis dekodera dwuwejściowego na kod 1 z 4

```
PIN [1,2] = [A,B]; /* Data Inputs */
PIN 3 = !enable;   /* Enable Input */
PIN [12..15] = [Y0..3]; /* Decoded Outputs */
PIN 14 = no_match; /* Match Output */
CONDITION
{
  IF enable & !B & !A  out  Y0;
  IF enable & !B & A   out  Y1;
  IF enable & B & !A   out  Y2;
  IF enable & B & A    out  Y3;
  }
```

Rozszerzenia

Rozszerzenia mogą być dodawane do nazw zmiennych w celu wskazania specyficznych funkcji powiązanych z głównymi węzłami wewnątrz programowalnego układu. Kompilator sprawdza rozszerzenie zmiennej w celu stwierdzenia, czy jest ono poprawne dla wybranego programowalnego układu oraz czy jego użycie nie jest w konflikcie z innymi użytymi rozszerzeniami. CUPL używa tych rozszerzeń, żeby skonfigurować makrokomórki układu PLD. Można w ten sposób wykorzystać wbudowane w strukturę układu przerzutniki oraz bufory trójstanowe.

Na rysunku 4.2 przedstawiono przykład użycia rozszerzeń zmiennych.




```
OUT_VAR.D = IN_VAR1 & OUT_VAR
          # !IN_VAR2 # IN_VAR3.DQ
          # !IN_VAR1 & OUT_VAR.IO
```

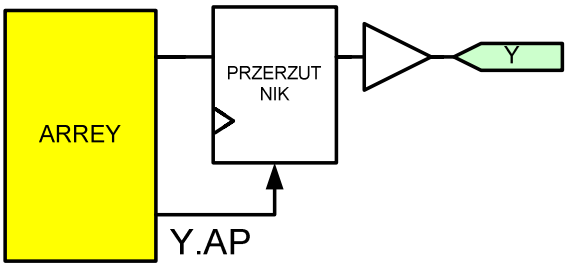
Rys. 4. 2. Obwód ilustrujący użycie rozszerzeń.

Tabela 4.7 przedstawia listę najczęściej używanych rozszerzeń. Kompletną listę można znaleźć w podręczniku „CUPL PLD/FPGA Language Compiler” (Wincupl2→Help→ CUPL Programmers Reference).

Tabela 4.7. Lista rozszerzenia zmiennych.

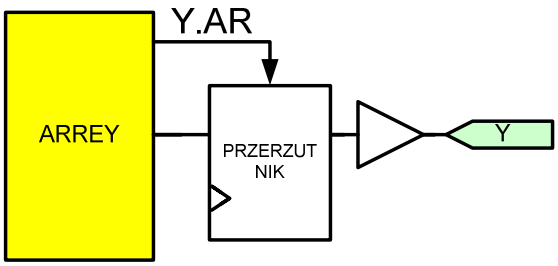
Rozszerzenie	Strona podstawienia	Opis
.AP	L	Asynchroniczne ustawienie przerzutnika.
.AR	L	Asynchroniczne zerowanie przerzutnika.
.CE	L	Wejście zezwalające CE.
.CK	L	Programowalny zegar przerzutnika.
.CKMUX	L	Wybór multipleksowanego zegara.
.D	L	Wejście informacyjne przerzutnika typu D.
.DFB	R	Wyjście przerzutnika D w trybie kombinacyjnym makroceli.
.DQ	R	Wyjście Q przerzutnika typu D.
.INT	R	Zwrotny sygnał z wewnętrznego wyjścia rejestrowego.
.IO	R	Sprzężenie zwrotne od końcówki wyjściowej.
.IOD	R	Sprzężenie zwrotne przez przerzutnik typu D.
.IOL	R	Sprzężenie zwrotne przez „latch”.
.J	L	Wejście J przerzutnika typu JK.
.K	L	Wejście K przerzutnika typu JK.
.L	L	Wejście informacyjne przerzutnika typu „latch”.
.LE	L	Wyjście Latch Enable przerzutnika typu „latch”.
.LQ	R	Wyjście Q przerzutnika typu „latch”.
.OE	L	Programowalne Output Enable.
.R	L	Wejście R przerzutnika typu RS.
.S	L	Wejście S przerzutnika typu RS.
.SP	L	Wejście synchronicznego ustawiania przerzutnika.
.T	L	Wejście T przerzutnika typu T.
.TFB	R	Wyjście przerzutnika T w trybie kombinacyjnym makroceli.

Rozszerzenie .AP:



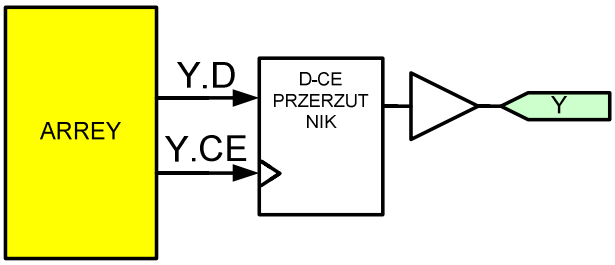
Rozszerzenie .AP stosowane jest celem zdefiniowania wyrażenia dla asynchronicznego sygnału ustawienia przerzutnika. Na przykład, wyrażenie `"Y.AP = A & B;"` powoduje, że przerzutnik jest asynchronicznie ustawiany, kiedy A i B są logicznie prawdziwe.

Rozszerzenie .AR:



Rozszerzenie .AR stosowane jest celem zdefiniowania wyrażenia dla asynchronicznego sygnału zerowania przerzutnika. Jest ono stosowane dla układów mających jedną lub więcej linii zależności podłączonych do asynchronicznego przerzutnika.

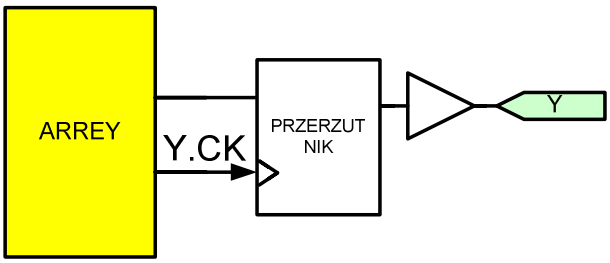
Rozszerzenie .CE:



Rozszerzenie .CE stosowane jest przy wykorzystaniu przerzutników typu D, które posiadają wejścia wyzwalane zegarem (przerzutniki D-CE). Służy do sprecyzowania wejść do termu CE (*Clock Enable*) przerzutnika. Jeżeli są nie

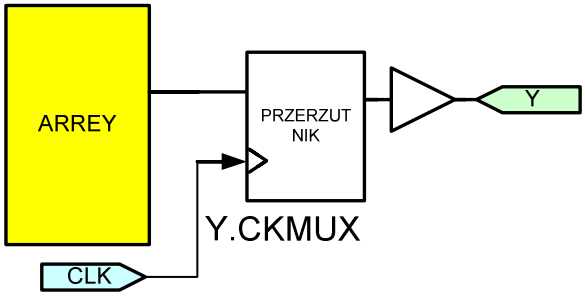
używane, winny być ustawione w stan 1 - przerzutnik zachowuje się jak przerzutnik typu D.

Rozszerzenie .CK:



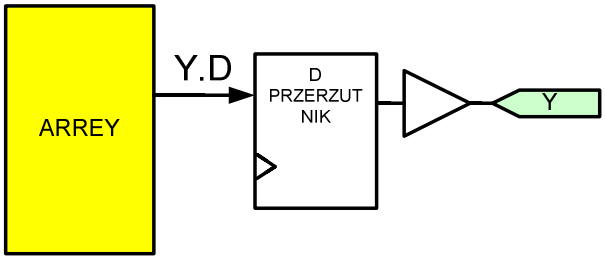
Rozszerzenie .CK stosowane jest do określenia zegara sterującego określonym termem. Używamy go celem podłączenia zegara przerzutnika do dedykowanego pinu.

Rozszerzenie .CKMUX:



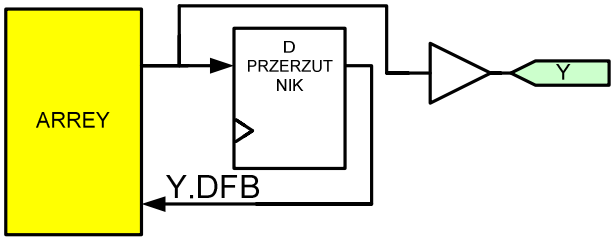
Rozszerzenie .CKMUX stosowane jest do podłączenia zewnętrznego zegara sterującego przerzutnikiem. Używanie rozszerzenia . CKMUX jest bardziej wydajne niż używanie rozszerzenia .CK.

Rozszerzenie .D:



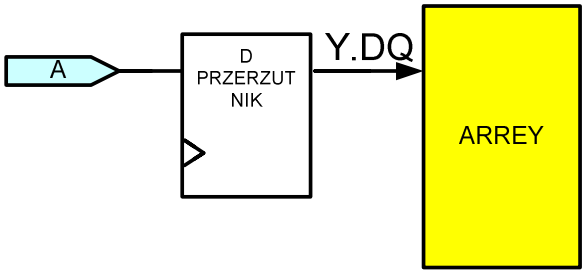
Rozszerzenie .D stosowane jest do określenia wejścia D do przerzutnika. Opcja ta powoduje skonfigurowanie przez kompilator makrokomórki w układach PLD jako przerzutnik typu D.

Rozszerzenie .DFB:



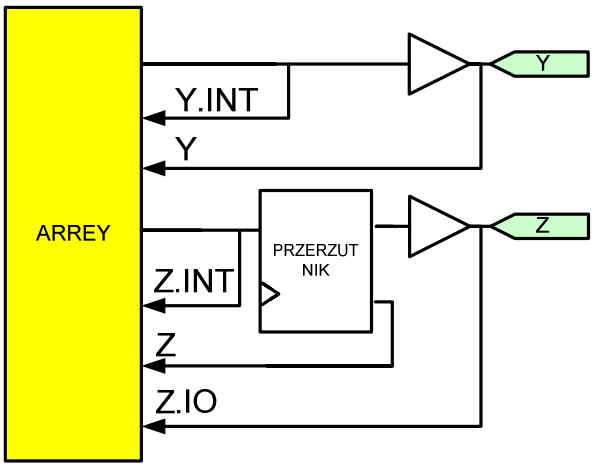
Rozszerzenie .DFB stosowane jest do skonfigurowania makrokomórek jako wyjście kombinacyjne w trybie kombinacyjnym. Rozszerzenie .DBF pozwala na realizację funkcji wejścia lub wyjścia jako podzestaw funkcji wejściowo-wyjściowej (*I/O function*).

Rozszerzenie .DQ:



Rozszerzenie .DQ stosowane jest do sprecyzowania wejścia D do przerzutnika. Opcja ta powoduje automatyczne ustawienie wejścia jako rejestrowe. Nie jest ono używane do sprecyzowania wyjścia Q przerzutnika D.

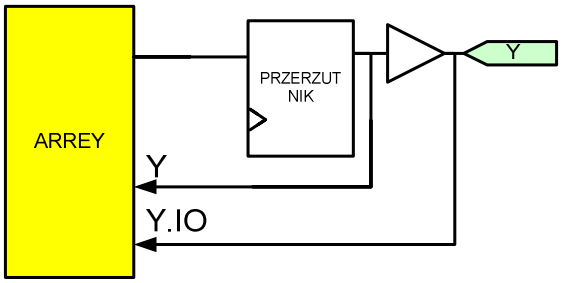
Rozszerzenie .INT:



Rozszerzenie .INT stosowane jest do określenia wewnętrznej ścieżki połączeń zwrotnych. Może być wykorzystane do określenia ścieżek zwrotnych

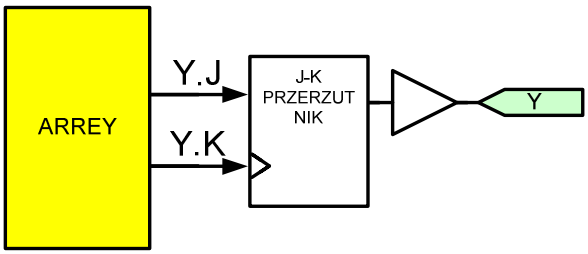
kombinatoryki typu „buried” zarówno dla wejść rejestrowych oraz kombinacyjnych.

Rozszerzenie .IO:



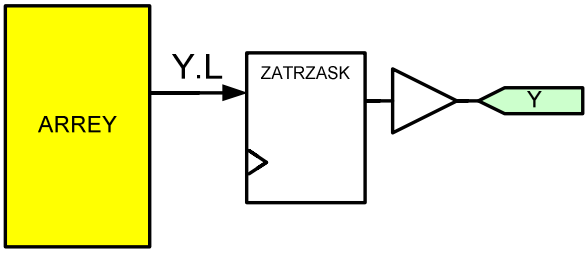
Rozszerzenie .IO stosowane jest do zaznaczenia połączeń zwrotnych pinów z komórkami. Jest ono przydatne gdy sam proces projektowy wymaga użycia pinów I/O oraz logiki typu „buried” w obrębie tej samej makrokomórki. Jest również użyteczne przy implementacji dwukierunkowych wyjść w CUPL.

Rozszerzenie .J i .K:



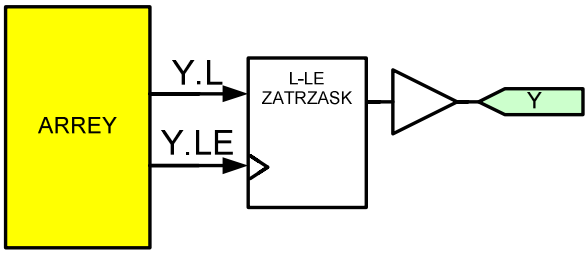
Rozszerzenie .J i .K stosowane jest do określenia wejść J-K do przerzutnika. Opcja ta powoduje skonfigurowanie przez kompilator makrokomórki w układach PLD jako przerzutnika typu J-K.

Rozszerzenie .L:



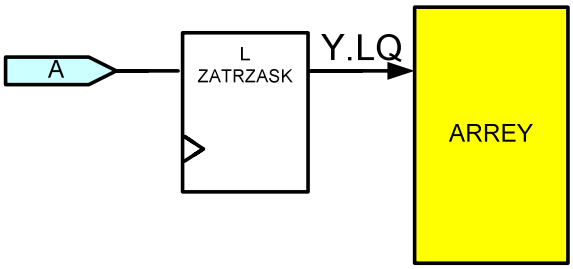
Rozszerzenie .L stosowane jest do określenia wejść typu „zatrzask” (latch). Opcja ta powoduje skonfigurowanie przez kompilator makrokomórki w układach PLD jako przerzutnika typu zatrzask.

Rozszerzenie .LE:



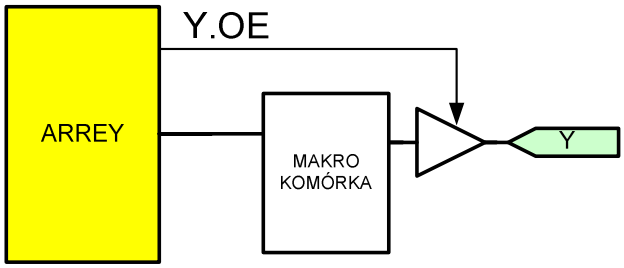
Rozszerzenie .LE stosowane jest do określenia równań obsługujących przerzutniki typu „zatrzask” (latch). Jest wymagane przy projektowaniu z wykorzystaniem możliwości wyzwalania przerzutników poziomem. Powoduje ono, że term produkcji łączony jest z blokiem LE.

Rozszerzenie .LQ:



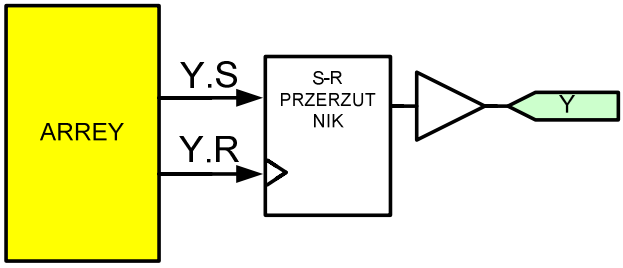
Rozszerzenie .LQ stosowane jest do specyfikacji zatrzasku wejściowego (*input latch*). Opcja ta powoduje automatyczne ustawienie wejścia jako zatrzaskowe (*latched*).

Rozszerzenie .OE:



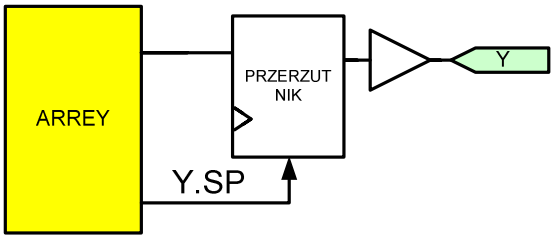
Rozszerzenie .OE stosowane jest do specyfikacji sygnału wyjściowego dla termu produkcji. Jest wymagane przy wykorzystywaniu dwukierunkowych pinów I/O oraz indywidualnie programowalnych termów wyjściowych.

Rozszerzenie .S i .R:



Rozszerzenie .S i .R stosowane jest do określenia wejść S-R do przerzutnika. Opcja ta powoduje skonfigurowanie przez kompilator makrokomórki w układach PLD jako przerzutniki typu S-R.

Rozszerzenie .SP:

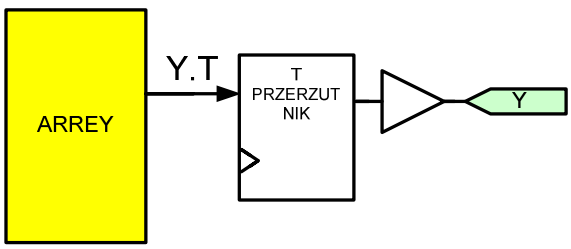


Rozszerzenie .SP stosowane jest do ustawienia synchronicznego wejścia ustawiającego preset przerzutnika na określone wyrażenie. Na przykład równanie

```
Y.SP = A & B; /*gdzie A i B sa wejsciami*/
```

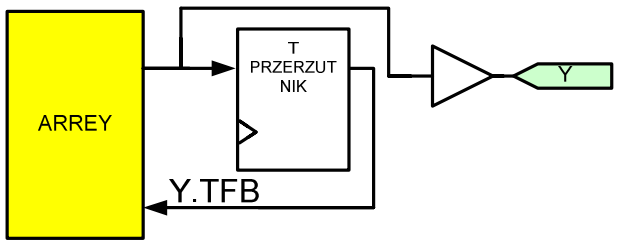
Powoduje, że wyjście Y jest ustawiane synchronicznie z zegarem lokalnym użytym w makroceli, gdzie wejścia A i B są logicznie prawdziwe.

Rozszerzenie .T:



Rozszerzenie .T stosowane jest do określenia wejścia T do przerzutnika. Opcja ta powoduje skonfigurowanie przez kompilator makrokomórki w układach PLD jako przerzutniki typu T.

Rozszerzenie .TFB:



Rozszerzenie .TFB stosowane jest wówczas, gdy mikrokomórka jest ustawiona na kombinacyjne wyjście, ale przerzutnik T pozostaje podłączony do wyjścia. Rozszerzenie .TEB pozwala na sprzężenie zwrotne rejestrowej reprezentacji kombinacyjnego wyjścia z powrotem do układu programowalnego.

4.2.2. Preprocesor

Kompilator języka CUPL zawiera preprocesor umożliwiający makrogenerację, generację warunkową oraz włączenie do programów zawartości wskazanych plików.

Wiersze programu zaczynające się znakiem $ służą do komunikacji z preprocesorem. W instrukcjach tych nie występuje znak średnika ani znak przypisania.

Instrukcje rozpoznawane przez preprocesor przedstawia tabela 4.8.

Tabela 4.8. Instrukcje preprocesora.

$DEFINE	$IFDEF	$UNDEF
$ELSE	$IFNDEF	$REPEAT
$ENDIF	$INCLUDE	$REPEND
$MACRO	$MEND	

W instrukcjach preprocesora duże i małe litery nie są rozróżnialne, ale zwyczajowo pisze się je dużymi literami.

Instrukcja $DEFINE

Instrukcja `$DEFINE` zastępuje ciąg znakowy przez określony operator, liczbę lub zmienną. Format instrukcji jest następujący:

```
$DEFINE argument1 argument2
```

gdzie: `argument1` - ciąg znakowy,

`argument2` - operator, liczba lub zmienna.

Przykład:

```
$DEFINE  ON    'b'1
$DEFINE  OFF   ‘b'0
$DEFINE  PORTC 'h'3F0
```

`$DEFINE` pozwala również tworzyć osobisty zbiór logicznych operatorów.

Przykład:

```
$DEFINE   {    /*   zastepuje poczatek komentarza
$DEFINE   }    */   zastepuje koniec komentarza
$DEFINE   /    !    zastepuje Negacje
$DEFINE   *    &    zastepuje AND
$DEFINE   +    #    zastepuje OR
$DEFINE   :+:  $    zastepuje XOR
```

Instrukcja $UNDEF

Instrukcja `$UNDEF` odwraca działanie komendy `$DEFINE`. Zleca zaprzestanie zastępowanie identyfikatora. Format instrukcji jest następujący:

```
$UNDEF argument
```

gdzie: `argument` - ciąg znakowy użyty w komendzie `$DEFINE`.

Instrukcję `$UNDEF` można użyć do przedefiniowania ciągu znakowego.

Przykład:

```
Pin [3..6] = [in3..0];
```

```
$DEFINE S0 ‘B’0010
select1 =[in3..0]:’S0;
$UNDEF S0
$DEFINE S0 ‘B’1000
select2=[in3..0]:S0;
```

Instrukcja $INCLUDE

Instrukcja `$INCLUDE` zleca wstawienie w jej miejsce zawartości pliku o podanej nazwie. Format instrukcji jest następujący:

```
$INCLUDE filename
```

gdzie: `filename` – nazwa zbioru dyskowego.

Instrukcja $IFDEF

Instrukcja `$IFDEF` sprawdza, czy identyfikator jest aktualnie zdefiniowany w preprocesorze. Format instrukcji jest następujący:

```
$IFDEF argument
```

Instrukcja $IFNDEF

Instrukcja `$IFNDEF` sprawdza, czy identyfikator nie jest aktualnie zdefiniowany w preprocesorze. Format instrukcji jest następujący:

```
$IFNDEF argument
```

Po instrukcjach sterujących `$IFDEF` oraz `$IFNDEF` następuje dowolna liczba wierszy programu, wśród których może wystąpić wiersz:

```
$ELSE
```

Całą konstrukcję kończy instrukcja:

```
$ENDIF
```

Przykład:

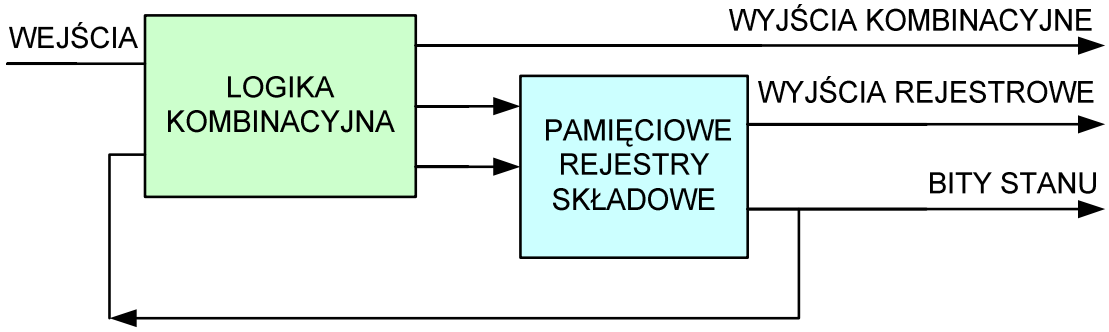
```
$DEFINE Prototype X/* define Prototype*/
$IFDEF Prototype
pin 1 = memreq;     /* memory request on  */
                         /* pin 1 of prototype*/
pin 2 = ioreq;           /* I/O request on*/
```

```
                              /* pin 2 of prototype*/
    $ELSE
    pin 1 = ioreq;            /* I/O request on*/
                              /* pin 1 of PCB*/
    pin 2 = memreq;      /* memory request on  */
                              /* pin 2 of PCB*/
    $ENDIF
```

Jeżeli sprawdzony warunek jest prawdziwy, to wszystkie wiersze pomiędzy `$ELSE` i `$ENDIF` są ignorowane. Natomiast, jeżeli warunek jest fałszywy, to są ignorowane wiersze pomiędzy testem a instrukcją `$ELSE` lub `$ENDIF` w przypadku braku `$ELSE.`

4.2.3. Opis automatów

Model układu sekwencyjnego w języku CUPL przedstawia rysunek 4.3. Używa on sześciu komponentów: wejść, logiki kombinacyjnej, rejestrów składowych, bitów stanu, wyjść rejestrowych i wyjść kombinacyjnych.



Rys. 4.3. Model układu sekwencyjnego.

Wejścia - sygnały wejściowe do układu mające swój początek w innym układzie.

Logika kombinacyjna - dowolna kombinacja bramek logicznych (najczęściej AND i OR), które generują sygnał wyjściowy o czasie opóźnienia T_{pd} (*propagation delay time*).

Rejestry składowe - przerzutniki, które otrzymują na wejście informacje z logiki kombinacyjnej układu sekwencyjnego. Niektóre przerzutniki konfigu-

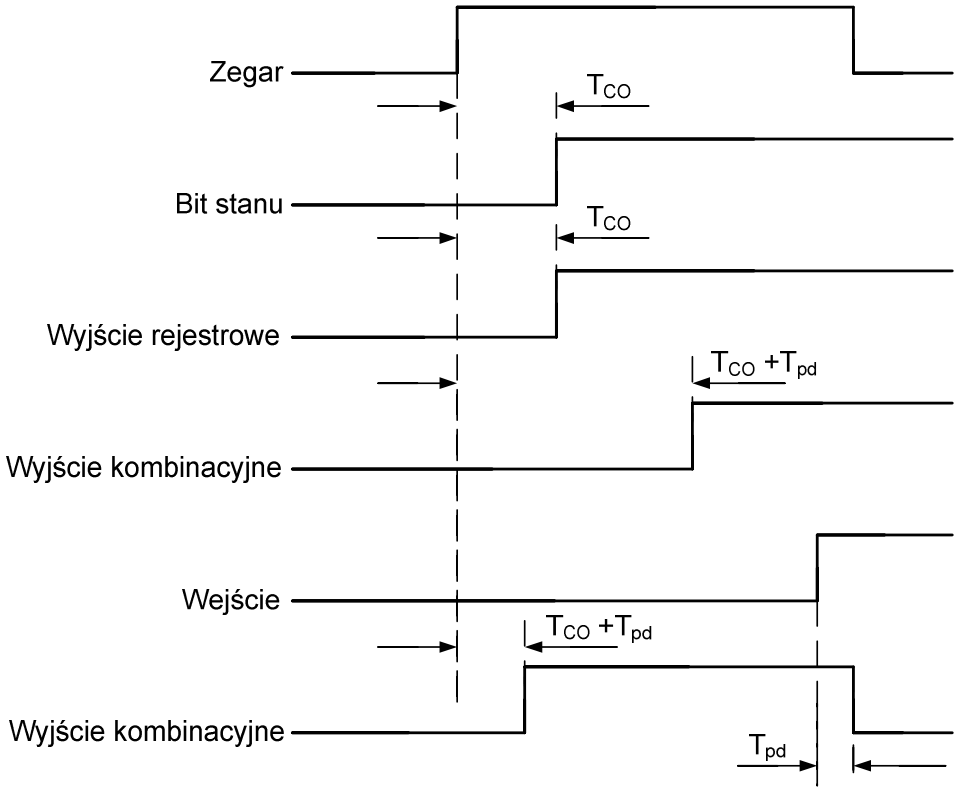
rowane są na potrzeby bitów stanu, inne na potrzeby rejestrowych wyjść. Rejestrowe wyjście charakteryzuje się czasem T_{co} (c*lock to out tim*e), który reprezentuje opóźnienie pomiędzy zboczem sterującym sygnału zegara a wystąpieniem zmian na odpowiednim wyjściu przerzutnika.

Bity stanu - wyjścia przerzutników, które używane są przy sprzężeniach zwrotnych. Zawierają informacje o aktualnym stanie przerzutników.

Wyjścia kombinacyjne – wyjścia bezpośrednie z układu logiki kombinacyjnej. Mogą być funkcją bitu stanu lub sygnału wejściowego. Generują sygnał wyjściowy o czasie opóźnienia $T_{co} + T_{pd}$.

Wyjścia rejestrowe – wyjścia z układu przerzutników składowych.

Rysunek 4.4 przedstawia przebiegi czasowe na poszczególnych wejściach i wyjściach układu sekwencyjnego.



Rys. 4.4. Przebiegi czasowe dla układu sekwencyjnego.

Aby zaimplementować układ sekwencyjny, CUPL udostępnia składnię, która pozwala wykonać każdą funkcję możliwą w automacie. Żeby wprowa-

dzić funkcję logiczną za pomocą grafu przejść stanów, stosuje się składnię złożoną ze słów kluczowych: SEQUENCE, PRESENT.

Opis układu sekwencyjnego jest możliwy przez zastosowanie instrukcji o następującym formacie:

```
SEQUENCE state_zm_list
{
PRESENT  state_n0 statements;
   .
   .
   .
PRESENT  state_nn statements;
}
```

gdzie: state_zm_list - lista zmiennych używanych w grafie przejść stanów,

state_n - numer stanu i zdekodowana wartość z listy state_zm_list, jest on liczbą, która jest unikalna dla każdego stanu w składni SEQUNECE ..PRESENT,

statements - warunki lub wyrażenia wyjściowe opisane w następnych sekcjach, słowa kluczowe NEXT, OUT, IF.

Przejścia warunkowe

W składni SEQUENCE .. PRESENT można umieścić warunki, od których będzie zależało przejście między stanami. Format składni jest następujący:

```
SEQUENCE state_zm_list
{
PRESENT state_n
IF expr NEXT state_n OUT [!]zm ... OUT [!]zm;
..............
IF expr NEXT state_n OUT [!]zm ... OUT [!]zm;
[[DEFAULT] NEXT state_n OUT [!]zm;]
}
```

gdzie: NEXT: - słowo kluczowe określające stan następny.

OUT: - słowo kluczowe określające stan wyjścia powiązanego z przejściem w określonym stanie.

IF: - słowo kluczowe określające warunkową zmianę stanu.

Listę warunkowych instrukcji NEXT może opcjonalnie kończyć instrukcja DEFAULT. Określa ona do jakiego stanu automat ma się przełączyć, gdy żaden z warunków wcześniejszych nie jest spełniony.

Instrukcję DEFAULT należy stosować z rozwagą, ponieważ powoduje ona dodanie warunku będącego negacją sumy wszystkich warunków wcześniejszych.

Sygnały generowane przez automat opisuje instrukcja OUT. Sygnały te mogą być wyprowadzane jako kombinacyjne lub zatrzaskiwane w przerzutnikach. Mówimy wówczas o instrukcji OUT asynchronicznej lub synchronicznej. Ponadto sygnały te mogą zależeć tylko od zmiennej stanu - sygnały bezwarunkowe - lub od zmiennych stanu oraz sygnałów wejściowych - sygnały warunkowe.

4.2.4. Format pliku źródłowego

Struktura opisu pliku źródłowego powinna być zgodna z ustalonym szablonem i powinna zawierać następujące elementy:

- ✓ nagłówek,
- ✓ blok tytułowy,
- ✓ deklarację wyprowadzeń układu (pinów),
- ✓ definicję zmiennych przejściowych,
- ✓ równania logiczne.

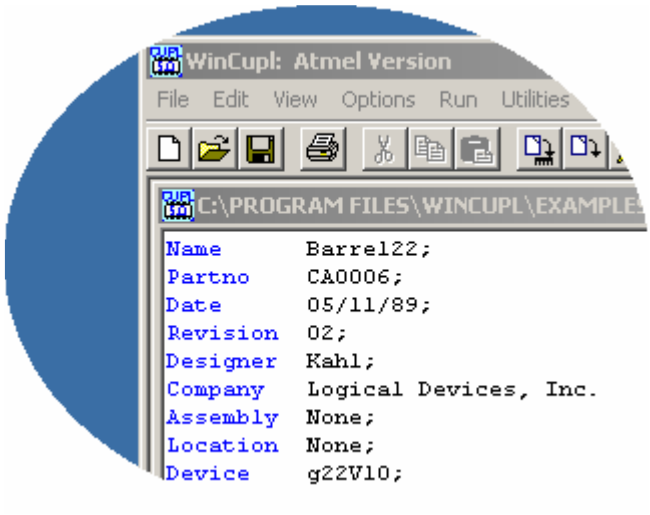
Opisy poszczególnych elementów pliku źródłowego zilustrowano rysunkami programu Wincupl znajdującego się w katalogu Wincupl\Examples\Atmel\Barrel22.pld.

Nagłówek

Nagłówek (tabela 4.9) umożliwia identyfikację pliku źródłowego dla celów archiwalnych. Jest on umieszczany na początku pliku. Na jego podstawie można określić: autora, czas powstania projektu, typ wykorzystywanego układu itp. CUPL dostarcza dziesięciu słów kluczowych do wykorzystania w nagłówku. W przypadku braku któregokolwiek z nich, kompilator wysyła komunikat z ostrzeżeniem. Po słowach kluczowych musi znajdować się informacja zakończona znakiem średnika (rysunek 4.5).

Tabela 4.9. Nagłówek pliku źródłowego.

Słowa kluczowe	Znaczenie
`Name`	opis nazwy pliku (do 32 znaków - w systemie DOS do 8 znaków). Jeśli pole *Name* nie zawiera nazwy pliku, CUPL nie wygeneruje pliku wynikowego.
`Partno`	określenie numeru części dla konkretnego projektu (np.: opis układu, w którym znajduje się element) - nie jest to nazwa typu elementu PLD.
`Date`	data ostatniej zmiany pliku źródłowego.
`Revision`	numer wersji kodu źródłowego (zaczynać od 01).
`Designer`	nazwisko i imię projektanta.
`Company`	nazwa firmy, dla której zaprojektowano układ.
`Assembly`	numer płytki drukowanej, na której będzie użyty układ.
`Location`	numer elementu na płytce.
`Device`	nazwa programowalnego układu logicznego (dla celów kompilacji).
`Format`	format pliku wyjściowego: h - plik w formacie ASCII-hex, i - plik w formacie Signetics HL, j - plik w formacie JEDEC.



Rys. 4. 5. Struktura nagłówka.

Blok tytułowy

Blok tytułowy (rysunek 4.6) zawiera dodatkowe informacje opisujące projektowany układ. Tekst ujęty jest w znaki początku i końca komentarza.

```
/*****************************************************************/
/*                                                               */
/* 8-Bit Registered Barrel Shifter                               */
/*                                                               */
/* This 8-bit registered barrel shifter takes 8 data inputs      */
/* and cyclically rotates the data from 0 to 7 places under      */
/* control of the select ( S0, S1, S2 ) inputs.  A SET input     */
/* can be used to initialize the outputs to the all ones state   */
/*****************************************************************/
/* Allowable Target Device Types :  PAL22V10                     */
/*****************************************************************/
```

Rys. 4.6. Struktura bloku tytułowego.

Deklaracje wyprowadzeń

W polu tym (rysunek 4.7) znajdują się deklaracje wyprowadzeń z układu: wejść i wyjść poprzedzonych słowem kluczowym PIN. Deklaracje wyprowadzeń występują opcjonalnie, gdy zadeklarowany jest typ układu.

```
/**  Inputs  **/

PIN 1           = clock;        /* Register Clock              */
PIN [2..9]      = [D7..0];      /* Data Inputs                 */
PIN [10,11,14]  = [S2..0];      /* Shift Count Select Inputs   */
PIN 13          = !out_enable;  /* Register Output Enable      */
PIN 23          = preset;       /* Set to Ones Input           */

/**  Outputs  **/

PIN [15..22] = [Q7..0];         /* Register Outputs            */
```

Rys. 4. 7. Struktura bloku deklaracyjnego.

Definicje zmiennych przejściowych

W polu tym (rysunek 4.8) znajdują się deklaracje wewnętrznych węzłów, pól bitowych oraz innych zmiennych, którym chce się przyporządkować wartości początkowe. Deklarowane są zmienne pomocnicze w postaci tablic (wektorów) zawierających nazwy zadeklarowanych wcześniej sygnałów lub będących wynikiem operacji na tych sygnałach.

```
/** Declarations and Intermediate Variable Definitions **/

field shift  = [S2..0];          /* Shift Width Field          */
field input  = [D7..0];          /* Inputs Field               */
field output = [Q7..0];          /* Outputs Field              */
```

Rys. 4. 8. Struktura zmiennych przejściowych.

Równania logiczne

W polu tym (rysunek 4.9) wpisywane są równania boolowskie, tablice prawdy układów logicznych oraz opis automatów (grafy przejść w formie tekstowej).

```
/** Logic Equations **/

output.d  =  [D7, D6, D5, D4, D3, D2, D1, D0]  &  shift:0
          #  [D0, D7, D6, D5, D4, D3, D2, D1]  &  shift:1
          #  [D1, D0, D7, D6, D5, D4, D3, D2]  &  shift:2
          #  [D2, D1, D0, D7, D6, D5, D4, D3]  &  shift:3
          #  [D3, D2, D1, D0, D7, D6, D5, D4]  &  shift:4
          #  [D4, D3, D2, D1, D0, D7, D6, D5]  &  shift:5
          #  [D5, D4, D3, D2, D1, D0, D7, D6]  &  shift:6
          #  [D6, D5, D4, D3, D2, D1, D0, D7]  &  shift:7;

output.sp = preset;              /* synchronous preset         */

output.oe = out_enable;          /* tri-state control          */

output.ar = 'h'00;               /* asynchronous reset not used */
```

Rys. 4. 9. Struktura opisu logicznego.

Przykładowy program  Barrel22.pld

Przykład opisu źródłowego 8-bitowego rejestru przesuwnego przedstawia listing programu *barrel22.pld*

```
Name      Barrel22;
Partno    CA0006;
Date      05/11/89;
Revision  02;
Designer  Kahl;
Company   Logical Devices, Inc.;
Assembly  None;
Location  None;
Device    g22V10;

/*************************************************************/
/*           8-Bit Registered Barrel Shifter                 */
/* This 8-bit registered barrel shifter takes 8 data inputs  */
/* and cyclically rotates the data from 0 to 7 places under  */
/* control of the select ( S0, S1, S2 ) inputs.  A SET input */
/* can be used to initialize the outputs to the all ones state*/
/*************************************************************/
/* Allowable Target Device Types :  PAL22V10                 */
/*************************************************************/

/**  Inputs  **/

PIN 1           = clock;       /* Register Clock            */
PIN [2..9]      = [D7..0];     /* Data Inputs               */
PIN [10,11,14]  = [S2..0];     /* Shift Count Select Inputs */
PIN 13          = !out_enable; /* Register Output Enable    */
PIN 23          = preset;      /* Set to Ones Input         */

/**  Outputs  **/

PIN [15..22] = [Q7..0];         /* Register Outputs         */

/** Declarations and Intermediate Variable Definitions **/

field shift  = [S2..0];         /* Shift Width Field        */
field input  = [D7..0];         /* Inputs Field             */
field output = [Q7..0];         /* Outputs Field            */

/** Logic Equations **/

output.d  =  [D7, D6, D5, D4, D3, D2, D1, D0]  &  shift:0
          #  [D0, D7, D6, D5, D4, D3, D2, D1]  &  shift:1
          #  [D1, D0, D7, D6, D5, D4, D3, D2]  &  shift:2
          #  [D2, D1, D0, D7, D6, D5, D4, D3]  &  shift:3
          #  [D3, D2, D1, D0, D7, D6, D5, D4]  &  shift:4
          #  [D4, D3, D2, D1, D0, D7, D6, D5]  &  shift:5
          #  [D5, D4, D3, D2, D1, D0, D7, D6]  &  shift:6
          #  [D6, D5, D4, D3, D2, D1, D0, D7]  &  shift:7;

output.sp = preset;           /* synchronous preset        */

output.oe = out_enable;       /* tri-state control         */

output.ar = 'h'00;            /* asynchronous reset not used */
```

4.3. Program WinCUPL

WinCUPL jest zaadaptowaną wersją dla Windows DOS-owego programu CUPL (*Compiler Universal for Programmable Logic*). Filozofia obsługi i sterowania bibliotekami pozostała niezmieniona. Pomimo niezbyt bogatej biblioteki układów obsługiwanych przez WinCUPL-a (dostępne są układy EPLD/CPLD firmy Atmel oraz standardowe układy PLD rodziny 16v8, 20v8, 22v10), nadal cieszy się sporym powodzeniem wśród użytkowników, ze względu na możliwości szybkiego i łatwego przygotowania projektów dla układów PLD. Program cechuje nieskomplikowana składnia i łatwość opisu przy jednocześnie dość dużych możliwościach.

Rys. 4.10. Okno powitalne WinCUPL i WinSim.

Program ten umożliwia kompilację tekstowych plików źródłowych opisujących układy, realizowane w strukturze programowalnej, do formatu *(jedec)* akceptowanego przez programatory oraz symulację działania zapisanej logiki przy wykorzystaniu programu *WinSim*. Projekt układu opisuje się przy pomocy języka wysokiego poziomu, który nosi nazwę CUPL (został on opisany w rozdziale 4.2). Przy jego pomocy możliwe jest tworzenie opisów cyfrowych układów kombinacyjnych i sekwencyjnych. WinCUPL udostępnia typowe dla opisów stosowanych w technice cyfrowej tablice prawdy, dowolne funkcje logiczne, grafy przejść, a także możliwości bliskie typowym makroasemblerom, tj. makropolecenia, predefiniowane funkcje oraz możliwość deklarowania stałych i zmiennych opisu.

Symulacja tworzonych projektów ogranicza się do symulacji funkcjonalnej, bez uwzględniania parametrów czasowych układów.

Ograniczoną funkcjonalnie do kompilatora i symulatora wersję oprogramowania narzędziowego udostępnia na swojej stronie internetowej firma Atmel w porozumieniu z firmą Logical Devices, która jest producentem tego oprogramowania. *(http://www.atmel.com.)*.

Pakiet WinCUPL zawiera następujące narzędzia:

WinCUPL - kompilator języka CUPL, zintegrowany z edytorem tekstowym. Umożliwia translację zbiorów zapisanych w języku CUPL na zbiory zawierające rozkazy dla programatora.

WinSim - symulator funkcjonalny projektów układów kompilowanych przez WinCUPL. Rezultat symulacji może być graficznie modyfikowany.

Po zainstalowaniu programu dostępne są dwie ikony, służące do uruchamiania:

- zintegrowanego edytora projektu WinCUPL,

- symulatora WinSim.

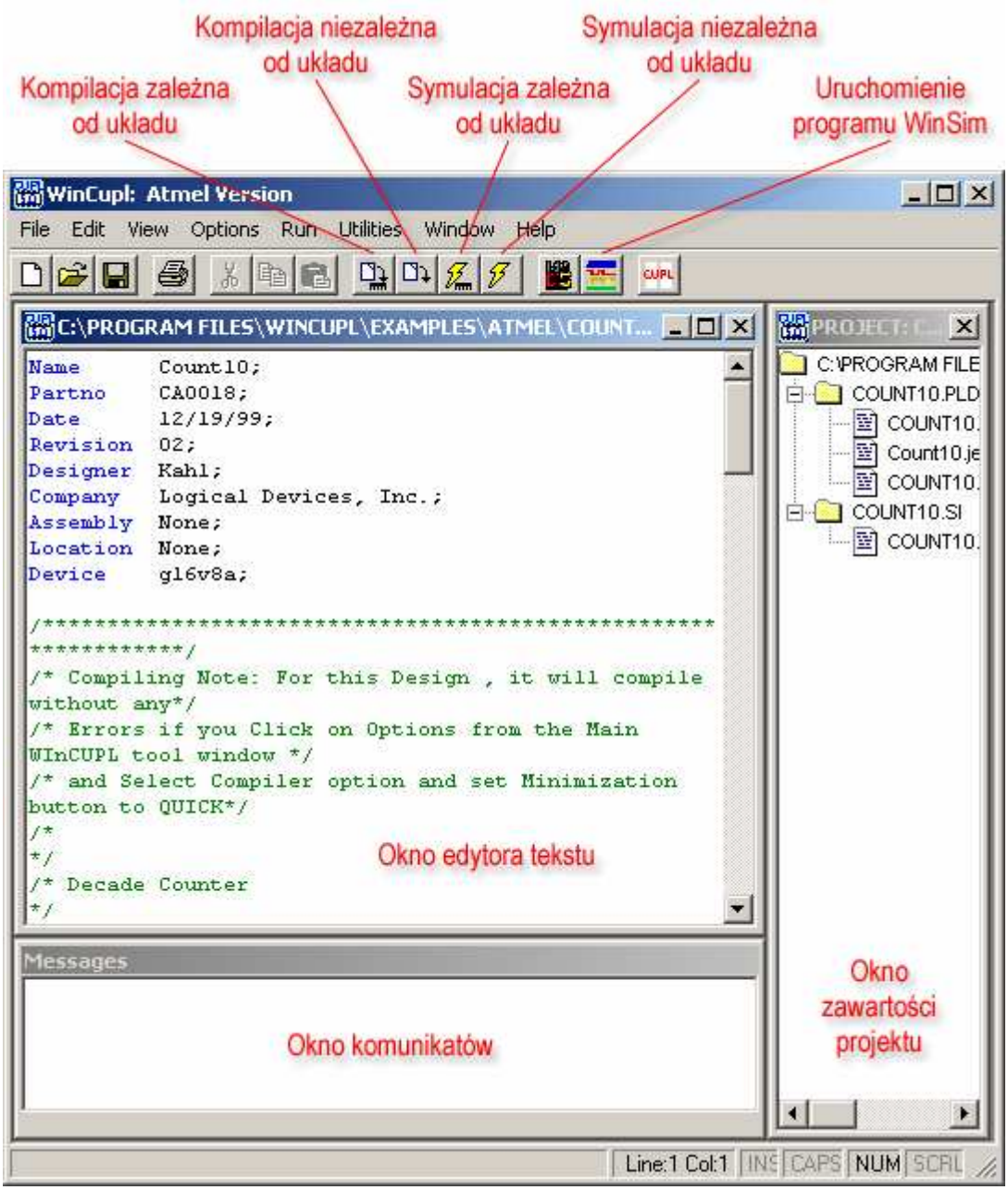
4.3.1. Obsługa WinCUPL

Aplikacja WinCUPL

Za pomocą aplikacji WinCUPL można tworzyć nowe projekty, wprowadzać tekst źródłowy, poddawać go kompilacji oraz wywoływać symulator.

Plikami źródłowymi są: plik *.PLD dla kompilatora oraz plik *.SI dla symulatora.

Po uruchomieniu programu WinCUPL pojawia się główne okno. Widok domyślnego okna głównego z otwartym przykładowym projektem przedstawia rysunek 4.11.



Rys. 4.11. Główne okno programu WinCUPL.

Okno główne składa się z paska menu i opcjonalnie z dodatkowego paska narzędzi oraz trzech części charakterystycznych dla pakietu WinCUPL:

- Okna edytora tekstowego, w którym znajduje się plik źródłowy, automatycznie kolorującego słowa kluczowe i inne elementy tworzonego opisu źródłowego, zgodnie z regułami języka.
- Okna komunikatów, w którym wyświetlane są raporty z działania poszczególnych plików będących efektem kompilacji, w tym komunikaty o błędach.
- Okna zawartości projektu, przedstawiającego listę plików źródłowych wykorzystywanych w projekcie.

Na pasku tytułu, znajdującego się w jego górnej części okna głównego, wyświetlana jest wersja WinCUPL-a udostępniana przez firmę Atmel Corporation. Pasek menu zawiera listę poleceń, przez wybranie których możemy tworzyć, drukować i zapisywać (w postaci odpowiednich plików) wyniki wykonywanych czynności, jak np. kompilacji, symulacji. Pasek narzędzi zawiera przyciski, z których korzysta się zamiast najczęściej używanych poleceń menu (rysunek 4.12).

Rys. 4.12. Pasek menu i narzędzi programu WinCUPL.

Menu ***File***

New - utworzenie nowego projektu.

Open - otwarcie istniejącego projektu.

Close - zamknięcie aktywnego okna projektu.

Save - zapisanie aktywnego projektu.

Save As - zapisanie obecnie modyfikowanego projektu pod nową nazwą.

Save All - zapisanie wszystkich otwartych projektów.

Open Project - otwarcie istniejącego projektu.

Close Project - zamknięcie projektu.

Print Setup - wybór ustawień drukowania.

Print - druk aktywnego dokumentu.

MRU… - lista ostatnio otwartych projektów.

Exit - wyjście z programu WinCUPL.

Menu ***Edit***

Undo - cofnięcie ostatnio dokonanej zmiany.

Cut - przeniesienie zaznaczonego tekstu do schowka.

Copy - skopiowanie zaznaczonego tekstu do schowka.

Paste - wklejenie zawartości schowka w miejsce wskazane przez kursor.

Find - wyszukanie tekstu w aktywnym dokumencie.

Find Next - wyszukanie następnego wystąpienia ostatnio poszukiwanej frazy.

Replace - wyszukanie i zastąpienie tekstu w aktywnym projekcie.

Insert CUPL Macro Reference - wstawienie referencji makra CUPL do aktywnego dokumentu.

Insert CUPL Macro Definition - wstawienie definicji makra CUPL do aktywnego dokumentu.

Update Macro Symbol Table - uaktualnienie tabeli symboli referencji makr.

Insert Table - wstawienie Tabeli prawdy.

Menu ***View***

Toolbar - włączenie i wyłączenie paska narzędzi.

Status Bar - włączenie i wyłączenie paska stanu.

Project - wyświetlenie lub ukrycie okna projektu.

Font - zmiana czcionki zaznaczonego tekstu.

Refresh - odświeżenie aktywnego projektu.

Menu ***Options***

Compiler - ustawienie opcji kompilatora.

Devices - wybór docelowego układu programowalnego z listy układów.

Simulator - ustawienie opcji symulatora.

WinCUPL - ustawienie opcji środowiska WinCUPL.

Menu ***Run***

Device Dependent Compile - kompilacja zależna od układu.

Device Independent Compile - kompilacja niezależna od układu.

Device Dependent Simulation - symulacja zależna od układu.

Device Independent Simulation - symulacja niezależna od układu.

Menu ***Utilities***

DOS - wywołanie wiersza poleceń DOS.

WinSim - uruchomienie programu WinSim.

CUPL Tools - uruchomienie wszystkich narzędzi WinCUPL.

Menu ***Window***

Cascade - kaskadowe rozmieszczenie okien.

Tile Horizontal - rozmieszczenie okien w poziomie.

Tile Vertical - rozmieszczenie okien w pionie.

Arrange Icons - rozmieszczenie ikon zamkniętych okien.

Auto Arrange - automatyczne rozmieszczenie okien.

Menu ***Help***

Contents - wyświetlenie tematów pomocy.

Search - wyświetlenie okna wyszukiwania w pliku pomocy.

CUPL Programmers Reference Guide - otwarcie Podręcznika Programisty CUPL.

Simulator - otwarcie pliku pomocy symulatora WinSim.

Atmel Info - wyświetlenie listy dokumentów.

About - wyświetlenie wersji aplikacji.

Pasek narzędzi

 - utworzenie nowego projektu,

 - otwarcie istniejącego projektu,

 - zapisanie na dysk aktywnego projektu,

 - wydruk aktywnego projektu na domyślnej drukarce,

 - przeniesienie do schowka aktualnie zaznaczonego obiektu,

 - skopiowanie do schowka aktualnie zaznaczonego obiektu,

 - wklejenie zawartości schowka do aktywnego projektu,

 - kompilacja zależna od układu,

 - kompilacja niezależna od układu,

 - symulacja zależna od układu,

 - symulacja niezależna od układu,

 - wywołanie wiersza poleceń DOS,

 - uruchomienie programu WinSim,

- uruchomienie wszystkich narzędzi WinCUPL.

Aplikacja WinSim 

WinSim jest środowiskiem umożliwiającym symulację funkcjonalną projektu opisanego w języku CUPL stworzonego w programie WinCUPL. Do jej pracy niezbędne jest utworzenie pliku *.SI, w którym zapisuje się pobudzenia testowe symulacji.

Jego struktura jest zbliżona do struktury pliku opisującego logikę i zawiera następujące elementy:

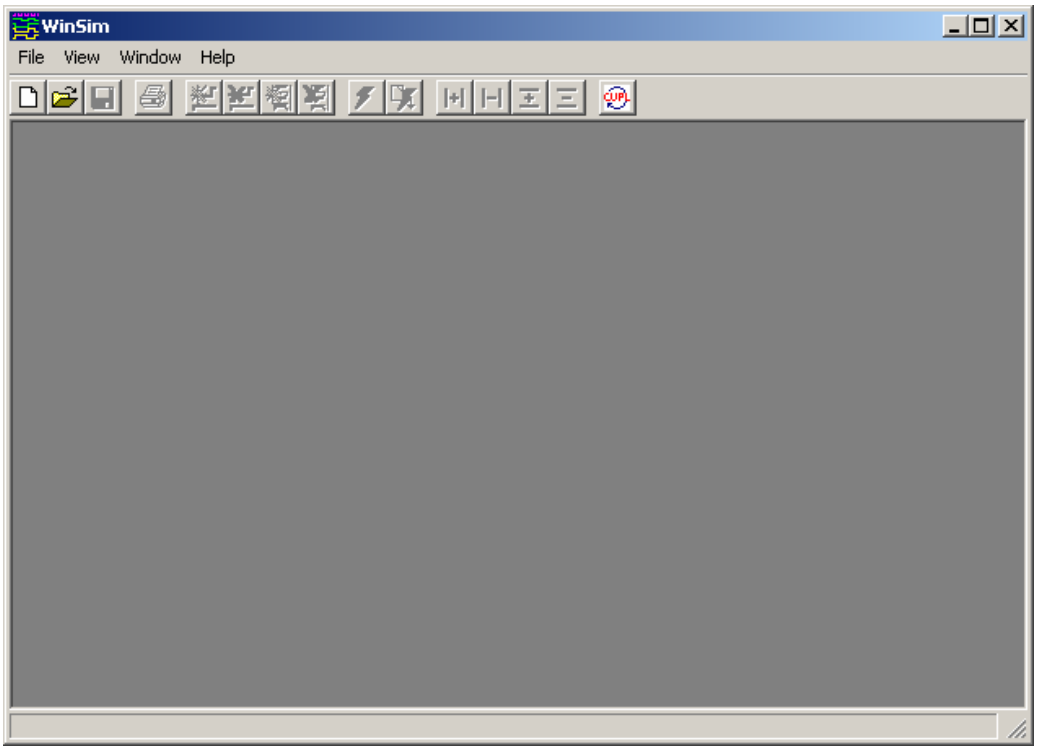
- ✓ nagłówek,
- ✓ blok deklaracji sygnałów wejściowych i wyjściowych do obserwacji po słowie kluczowym ORDER,
- ✓ blok wektorów testowych definiowany po słowie kluczowym VECTORS.

Blok nagłówka powinien być identyczny z tym określonym w pliku źródłowym z rozszerzeniem *.PLD.

Blok deklaracji sygnałów wejściowych i wyjściowych określa kolejność występowania w wektorach testowych poszczególnych zmiennych. Rozpoczyna go instrukcja ORDER.

Blok wektorów testowych umieszczony za słowem kluczowym VECTORS powinien uwzględniać wszystkie zmienne wejściowe sprawdzanych funkcji logicznych. Ilość wektorów testowych zależna jest od ilości kombinacji sygnałów wejściowych, na które chcemy znać odpowiedź projektowanych funkcji logicznych. Najczęściej uwzględnia się wszystkie możliwe kombinacje. Sygnały wejściowe muszą być wprowadzane do wektora testowego w kolejności ustalonej w bloku deklaracji sygnałów. W miejsce badanych sygnałów wyjściowych wstawia się znak *.

Po uruchomieniu aplikacji WinSim pojawia się główne okno. Widok domyślnego okna głównego przedstawia rysunek 4.13.



Rys. 4.13. Główne okno programu WinSim.

Pasek menu zawiera listę poleceń, przez wybranie których możemy tworzyć, drukować i zapisywać (w postaci odpowiednich plików) wyniki wykonywanych czynności, jak np. symulacji. Pasek narzędzi zawiera przyciski, z których korzysta się zamiast najczęściej używanych poleceń menu (rysunek 4.14).

Rys. 4.14. Pasek menu i narzędzi programu WinSim.

Menu ***File***

New - utworzenie nowego projektu.

Open - otwarcie istniejącego projektu.

Close - zamknięcie aktywnego okna.

Save - zapisanie aktywnego projektu.

Save As - zapisanie obecnie modyfikowanego projektu pod nową nazwą.

Save All - zapisanie wszystkich otwartych projektów.

Print Setup - wybór ustawień drukowania.

Print - druk aktywnego dokumentu.

MRU… - lista ostatnio otwartych projektów.

Exit - wyjście z programu WinSim.

Menu ***View***

Status Bar - włączenie i wyłączenie paska stanu.

Source - otwarcie okna tekstowe z aktywnym projektem z pliku *.SI.

Signal Definitions - otwarcie okna *Signal Definitions* dla aktywnej symulacji.

Grid - włączenie siatki w polu wyświetlania sygnałów.

Grid Size - ustawienie rozmiarów siatki w polu wyświetlania sygnałów.

Colors - zmiana kolorów wyświetlanych sygnałów na ekranie.

Wide Signals - przełączenie grubości linii wyświetlanego sygnału.

Refresh - odświeżenie aktywnego dokumentu.

Menu ***Signal***

Add Signal - dodaje nowe sygnały do symulacji.

Delete Signal - usuwa zaznaczone sygnały z symulacji.

Rename Signal - zmienia nazwę wybranego sygnału.

Add Vector - dodaje nowe wektory do symulacji.

Delete Vector - usuwa wybrane wektory z symulacji.

Add Message - dodaje wiersz wiadomości $MSG w pliku źródłowy stanowiący komentarz opisujący np. wektor testowy.

Bus Members - przełączenie na ekran szyny pamięci z wybranego sygnału.

Menu ***Simulator***

Run Simulator - uruchamia symulator CSIM dla aktywnego projektu.

Compile and Simulate - kompiluje plik źródłowy *.PLD i uruchamia symulator CSIM.

Assign Simulator Results - przyporządkowuje sygnały z symulacji do pliku symulującego.

Set Library - wybór bibliotek symulatora.

Menu ***Window***

Cascade - kaskadowe rozmieszczenie okien.

Tile Horizontal - rozmieszczenie okien w poziomie.

Tile Vertical - rozmieszczenie okien w pionie.

Arrange Icons - rozmieszczenie ikon zamkniętych okien.

Menu ***Help***

Contents - wyświetlenie tematów pomocy.

Search For Help On - wyświetlenie okna wyszukiwania w pliku pomocy.

About WinSim - wyświetlenie wersji aplikacji.

Pasek narzędzi

 - utworzenie nowego projektu,

 - otwarcie istniejącego projektu,

 - zapisanie na dysk aktywnego projektu,

 - wydruk aktywnego projektu na domyślnej drukarce,

 - dodanie nowego sygnału,

 - usunięcie zaznaczonych sygnałów,

 - dodanie wektorów testowych,

 - usunięcie zaznaczonych wektorów testowych,

 - uruchomienie symulatora,

 - kompilacja projektu i uruchomienie symulatora,

 - zwiększenie szerokości linii siatki,

 - zmniejszenie szerokości linii siatki,

 - zwiększenie wysokości linii siatki,

 - zmniejszenie wysokości linii siatki,

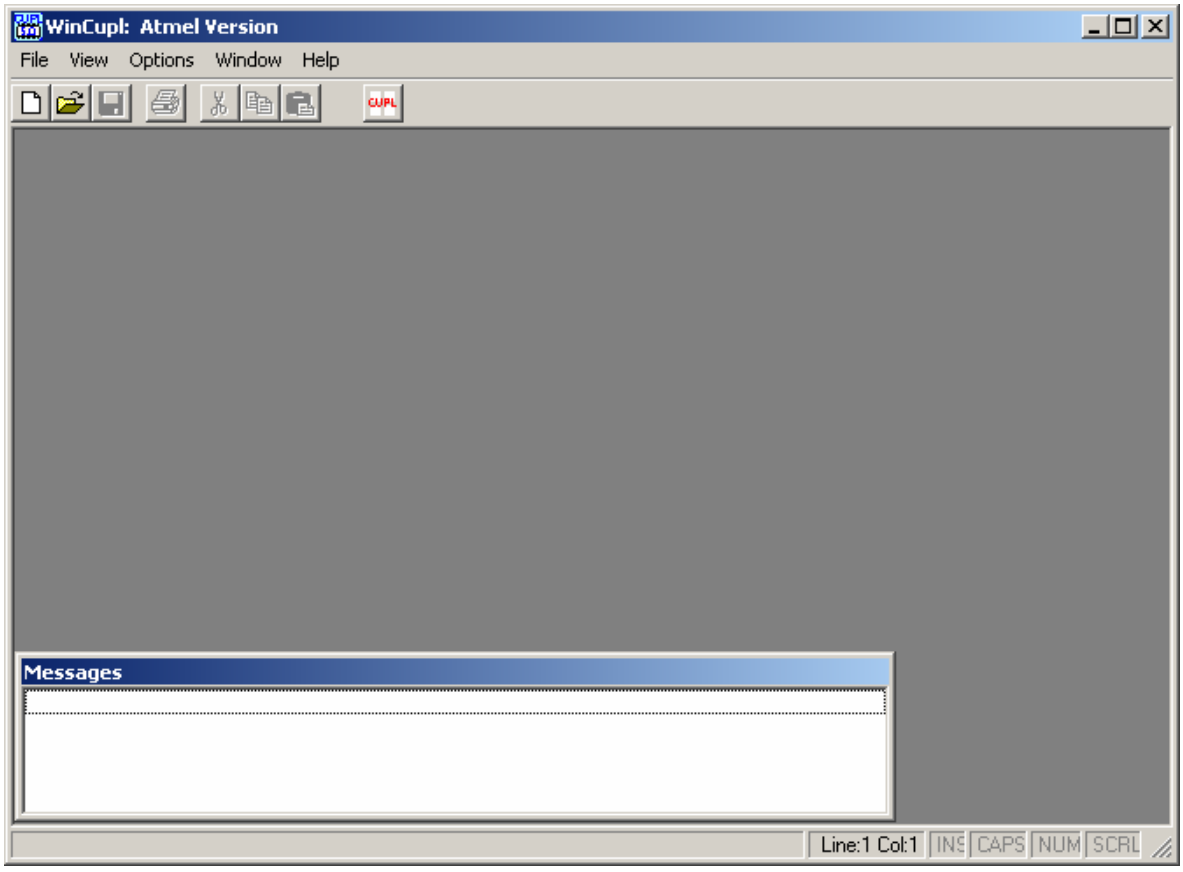
 - uruchomienie wszystkich narzędzi WinCUPL.

4.3.2. Praca z przykładowym projektem

W celu otwarcia, kompilacji oraz symulacji przykładowego projektu należy wykonać następujące czynności:

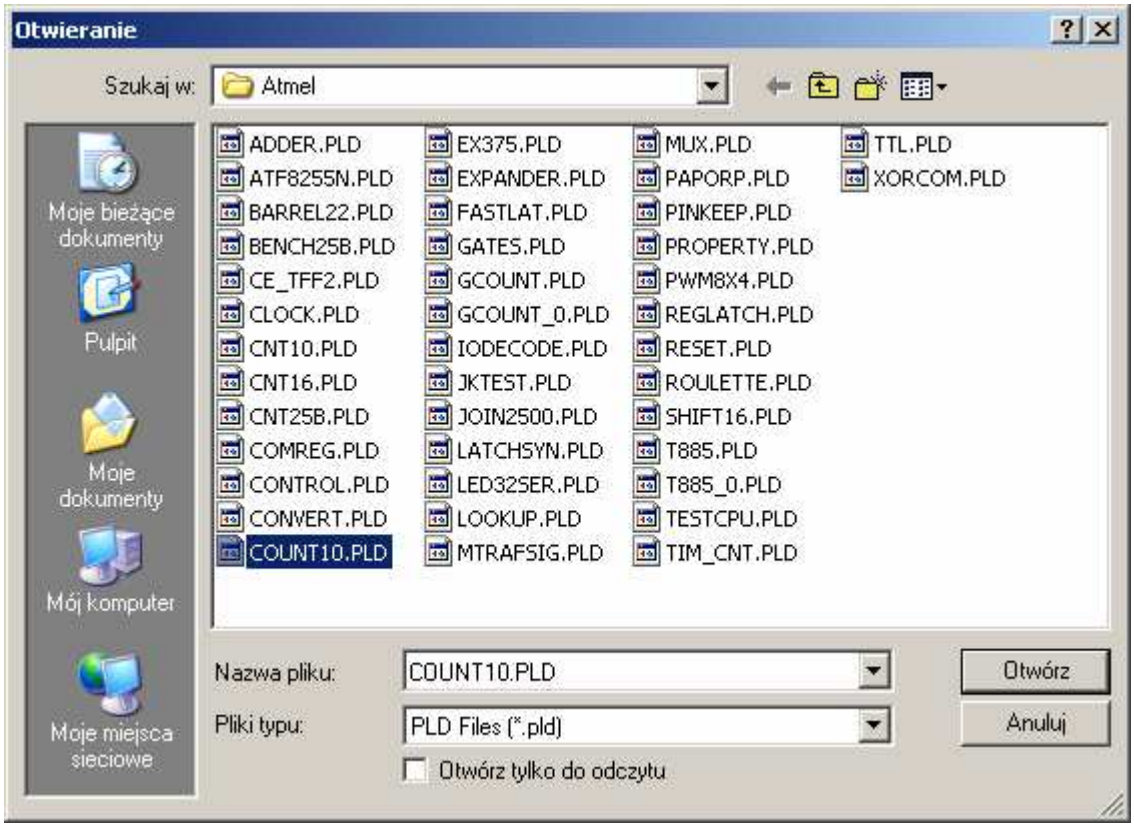
1. Uruchomić WinCUPL z listy programów menu Start lub używając ikony  służącej do uruchomienia aplikacji projektu.

Po uruchomieniu programu WinCUPL pojawia się główne okno. Widok domyślnego okna głównego przedstawia rysunek 4.15.



Rys. 4.15. Widok okna główne programu WinCUPL.

2.Otworzyć wybrany projekt umiejscowiony w: C:\Wincupl2\Examples\Atmel\COUNT10.PLD (rysunek 4.16). Plik ten przedstawia czterobitowy licznik dziesiętny liczący w górę i dół, z możliwością synchronicznego kasowania.



Rys. 4.16. Widok okna wyboru projektu.

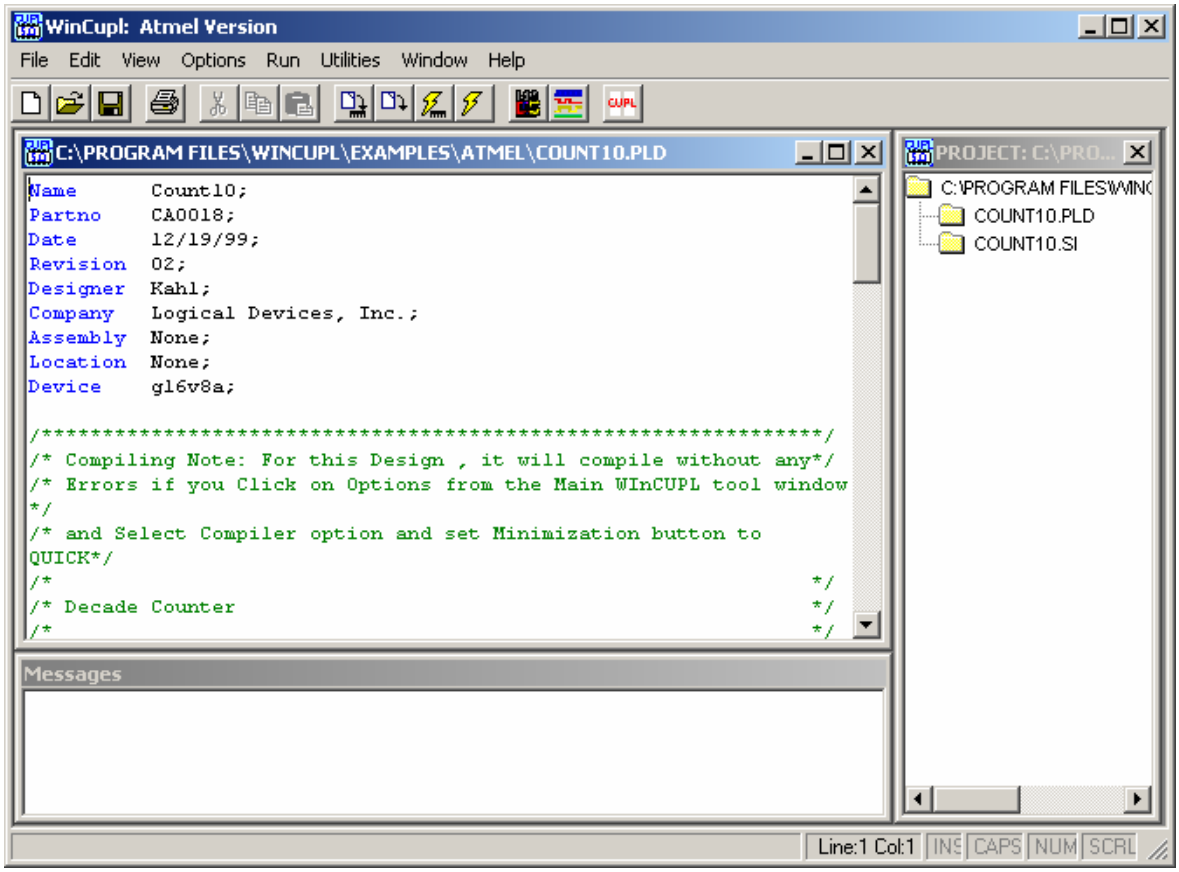
3. Po dokonaniu wyboru projektu pojawia się główne okno programu podzielone na trzy okna (rysunek 4.17), w których dostępne są podstawowe informacje.

Okno zawartości projektu *(PROJECT)* przedstawia wszystkie pliki skojarzone z otwartym projektem lub drzewo z plikami. Dwukrotne kliknięcie na nazwie danego pliku z tego drzewa powoduje otwarcie tego pliku w oknie edytora. Zamknięcie okna projektu powoduje zamknięcie wszystkich okien związanych z projektem.

Okno edytora tekstu *(Editor)* przedstawia pełny tekst tworzonego opisu pliku źródłowego licznika.

Okno komunikatów *(Messages)* jest czyste. Po przeprowadzeniu procesu kompilacji będzie przedstawiać raporty z działania plików i komunikaty o błędach podczas kompilacji. Dzięki ich wskazaniu i skomentowaniu błędy mogą być stosunkowo łatwo usunięte.

Wykaz wszystkich błędów generowanych podczas kompilacji i symulacji znajdziesz w pliku pomocy *CUPL Programmers Reference*, rozdział 4 *Error Messages.*



Rys. 4.17. Widok okien z projektem „Count 10”.

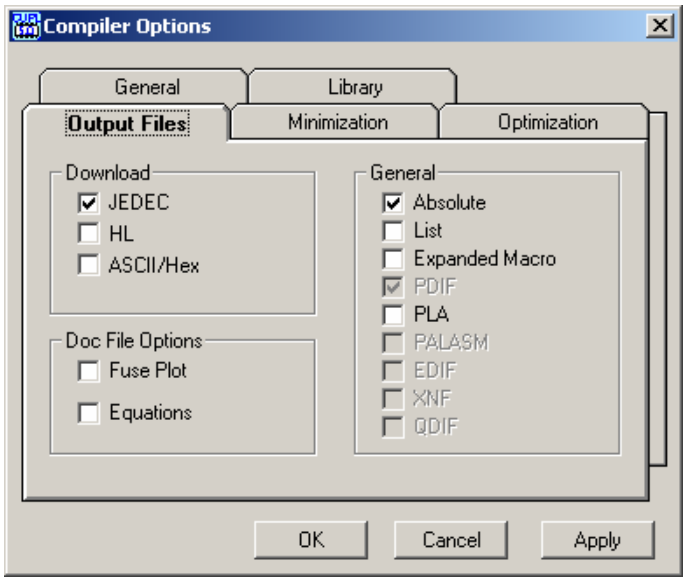
4. Wybór parametrów kompilacji.

Otworzyć okno *Compiler Options* poprzez wybranie z menu głównego *Options→Compiler*. Okno przedstawia opcje bezpośrednio wpływające na minimalizację, optymalizację i format plików wyjściowych (rysunek 4.18).

W zakładce *Output Files* w obszarze Downland powinien być zaznaczony *JEDEC*, w obszarze Doc File Options, jeżeli chcemy otrzymać dokument z rysunkiem układu, zaznaczamy *Fuse Plot* i *Equations*.

Również ustawiamy parametry w pozostałych zakładkach, np.: *General – Simulate, JEDEC name = PLD name.*

Zakładki *Minimization* oraz O*ptymization* umożliwiają wybór sposobu minimalizacji i optymalizacji, który będzie domyślny dla całego projektu.



Rys. 4.18. Widok okna konfiguracji kompilatora WinCUPL z aktywną zakładką *Output Files*.

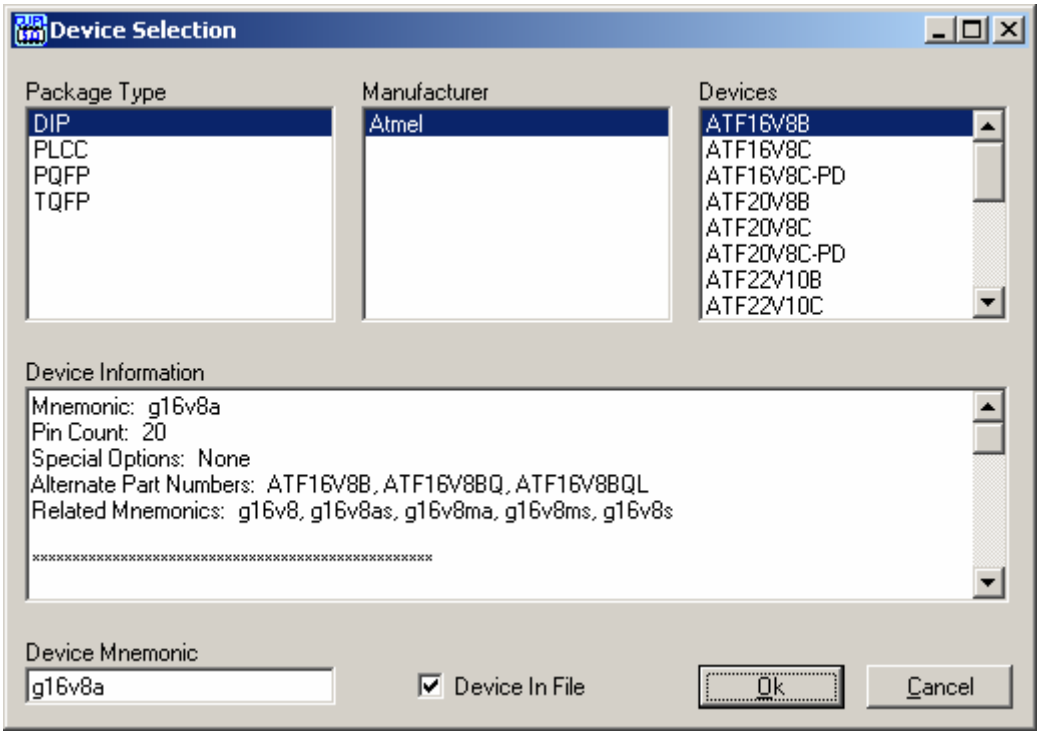
Minimalizację używa się do ustawienia stopnia minimalizacji dla poszczególnych zmiennych. Można wybrać jeden ze sposobów: brak minimalizacji, szybka, Quinn-McCluskey, Presto, Expresso.

5. Wybór układu dla projektu.

Otworzyć okno *Devices Selection* (rysunek 4.19) poprzez wybranie z menu głównego *Options→Devices*. W oknie ukażą się pola zawierające listę układów EPLD *(Devices)*, nazwę firmy producenta Atmel *(Manufacture)*, typ obudowy układu scalonego *(Package Type)*.

W dolnej części okna występuje pole edycyjne *Device mnemonic* oraz przycisk wielokrotnego wyboru *Device in file***.** Wybrać (zaznaczyć przycisk wyboru) z pola *Device in file* oraz wybrać symbol układu (dla układów GAL: g16v8 lub g22v10) w polu *Device*. Porównaj czy symbol układu w polu *Devi-*

ce Mnemonic jest taki sam jak w nagłówku pliku źródłowego w pozycji *Device* okna edytora tekstowego.

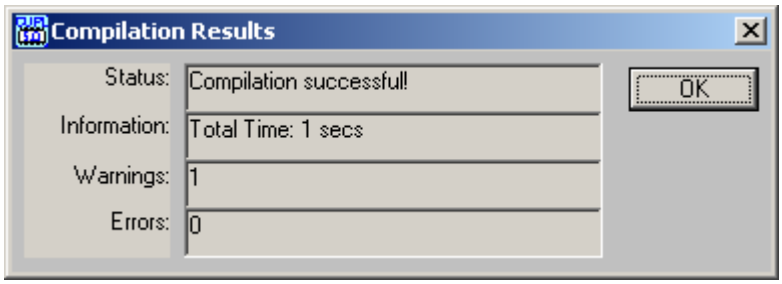


Rys. 4.19. Widok okna wyboru układu.

6. Kompilacja pliku źródłowego.

Z menu głównego wybierz *Run* → *Device Dependent Compile* (skrót F9 lub ). Jest to polecenie kompilacji projektu zapisanego w pliku źródłowym. Pamiętaj, że jeżeli plik źródłowy został zmodyfikowany, to przed kompilacją należy go zapisać.

Po zakończeniu kompilacji zostanie wyświetlony komunikat z przebiegu kompilacji (rysunek 4.20).



Rys. 4.20. Widok okna przebiegu kompilacji.

7. Symulacja skompilowanego projektu.

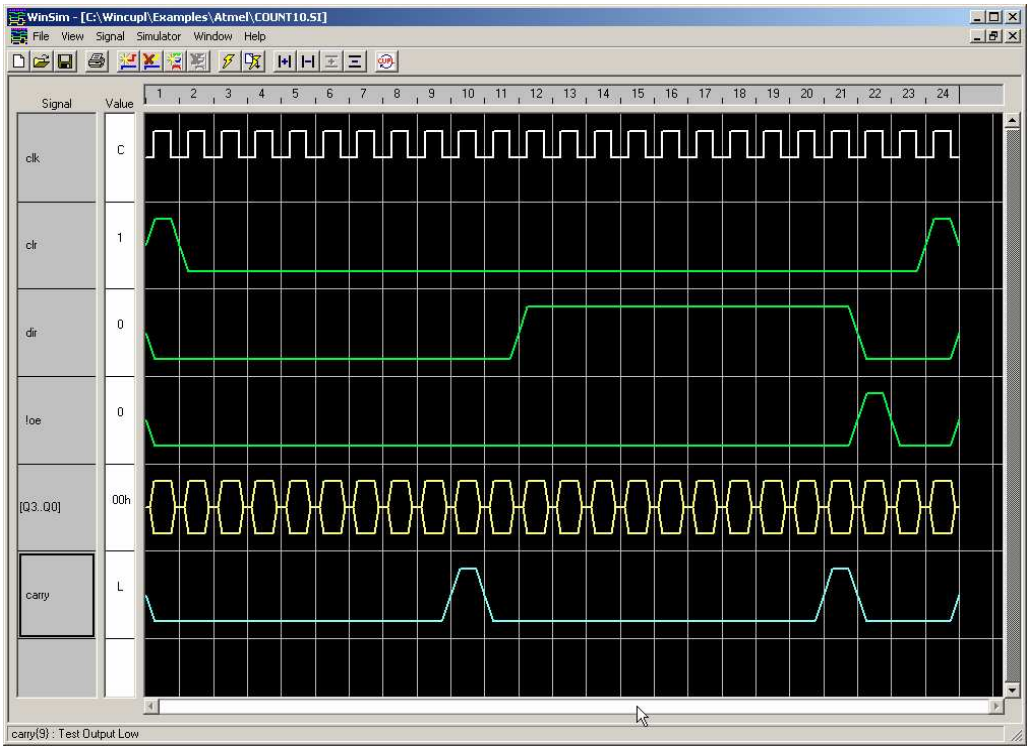
Kiedy kompilacja przebiegnie bez błędów, wykonaj dwukrotne kliknięcie na pliku z drzewa plików o nazwie Count10.SI. Otworzy się okno wywołania symulatora "*Open with Winsim*" (rysunek 4.21).

Rys. 4.21. Okno wywołania symulatora WinSim.

Wybierz przycisk *Tak*. Spowoduje to otwarcie symulatora funkcyjnego WinSim. Symulację pliku źródłowego możesz również wykonać wybierając zakładkę *Run → Device Dependent Simulate* (skrót F7 lub ).

Uwaga! Jeżeli wybierzesz przycisk *Nie* z okna "*Open with Winsim*" (rysunek 4.21), to w oknie edytora tekstowego otwarty zostanie plik .SI.

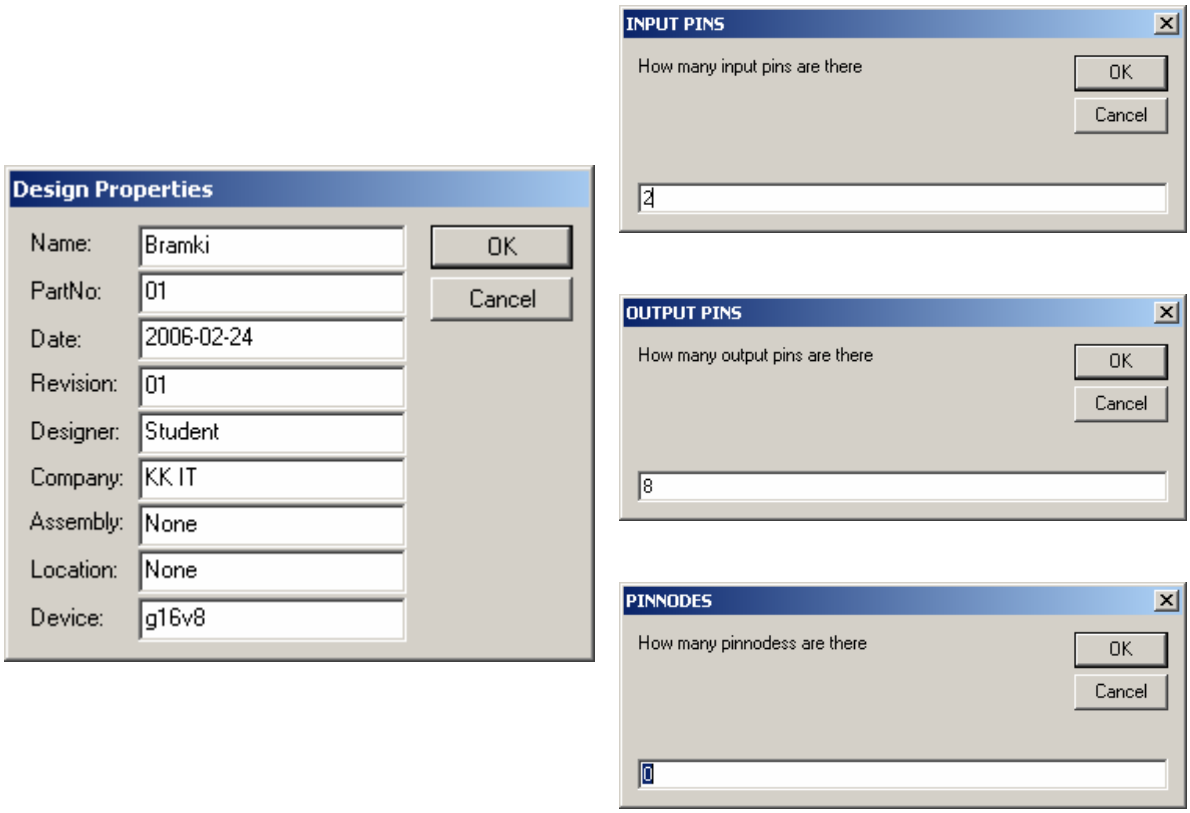
W otwartym oknie symulatora WinSim przedstawiony jest proces symulacji (rysunek 4.22) omawianego projektu.



Rys. 4.22. Widok okna z wynikiem symulacji.

4.3.3. Pierwszy program

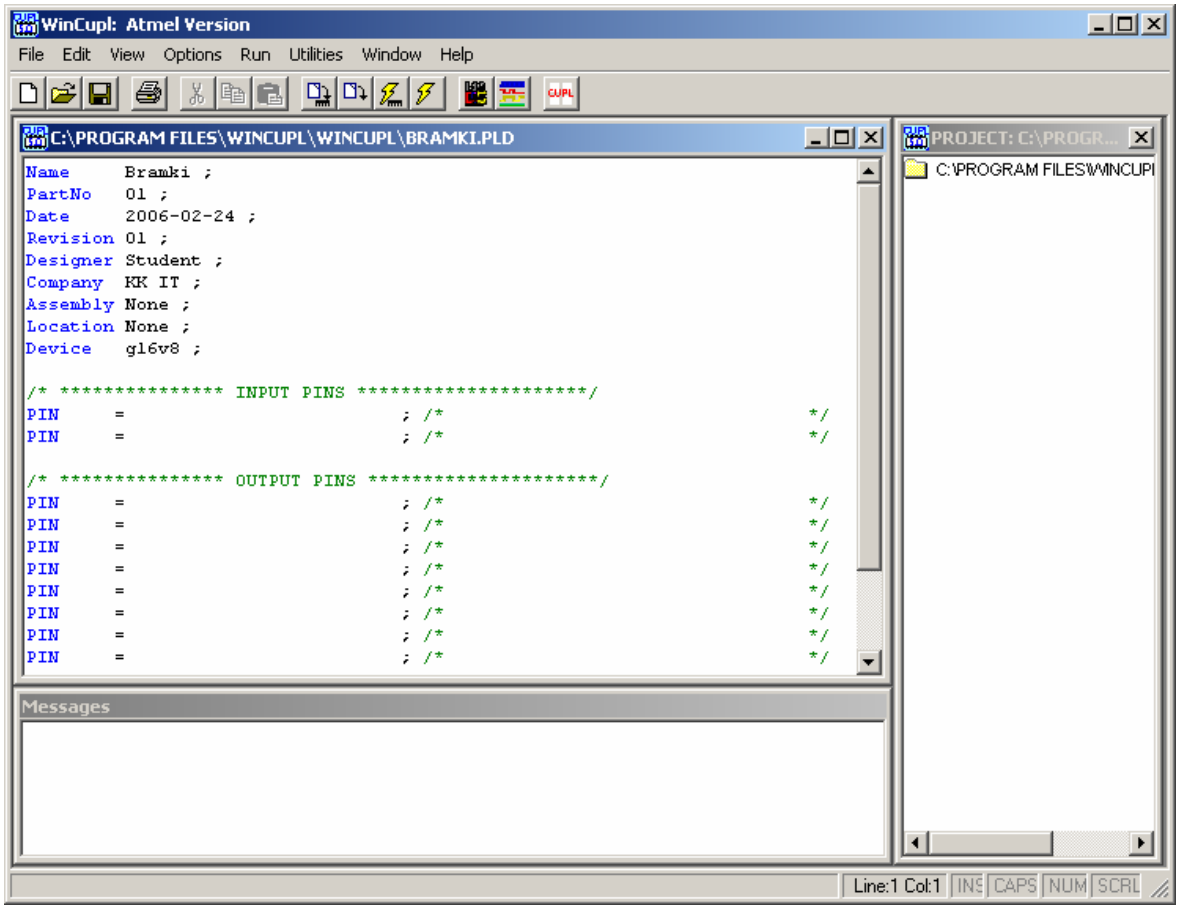
W celu utworzenia nowego projektu zapisanego w postaci pliku źródłowego należy uruchomić opcję *File/New/Project*. Pojawi się okno kreatora nagłówka (rysunek 4.23), w którym należy podać parametry identyfikacyjne nagłówka pliku źródłowego (*Desing Properties*), ilość końcówek wejściowych (*Input Pins*) i wyjściowych (*Output Pins*) wykorzystywanych w projekcie oraz ilość węzłów wewnętrznych (*Pinnodes*). Jeżeli w projekcie wykorzystujemy konkretny układ PLD należy wpisać go w pole *Device* nagłówka projektu. Lista układów, które można zastosować, dostępna jest w bibliotece CUPL po uruchomieniu opcji *Option/Devices* (patrz rozdział 4.3.2 - pkt 5).



Rys. 4.23. Okno kreatora nagłówka oraz deklaracji końcówek wyjściowych, wejściowych i węzłów wewnętrznych.

Po określeniu parametrów identyfikacyjnych pliku źródłowego oraz parametrów projektu: liczby wyprowadzeń WEJŚĆ, liczby wyprowadzeń WYJŚĆ oraz ilości węzłów wewnętrznych, zostanie utworzony pliku źródłowy z roz-

szerzeniem *Bramki.PLD* z pustym szablonem. Widok okna głównego z pustym szablonem przedstawia rysunek 4.24.

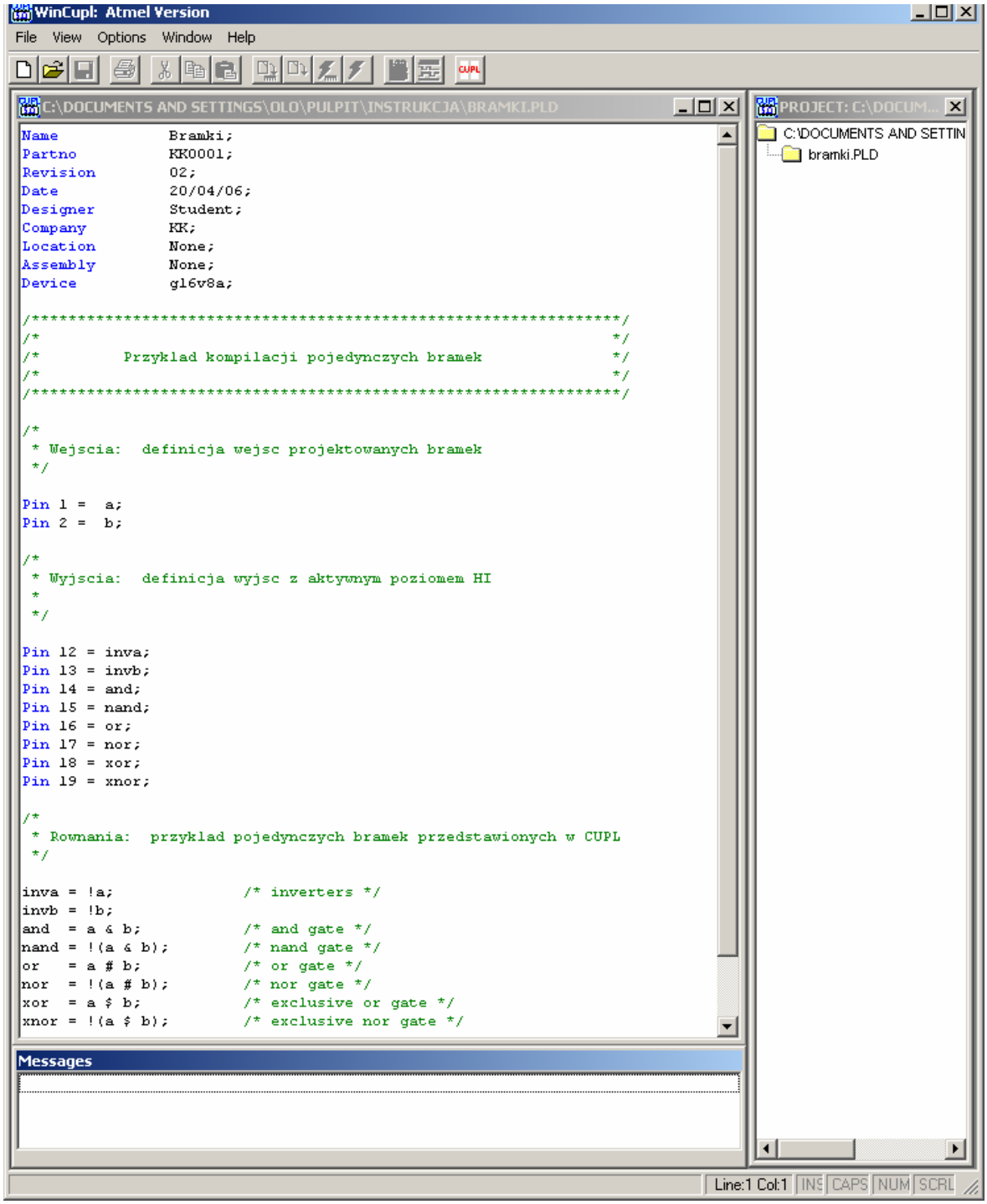


Rys. 4.24. Szablon pliku źródłowego z rozszerzeniem *.PLD.

W oknie edytora tekstowego, zachowując strukturę opisu pliku źródłowego, określamy: blok tytułowy, deklaracje wyprowadzeń z układu: wejść *(Input pins*) i wyjść *(Output pins)* - numery końcówek układu PLD, równania logiczne lub inne opisy funkcjonalne projektowanych układów - patrz rozdział 4.2.4.

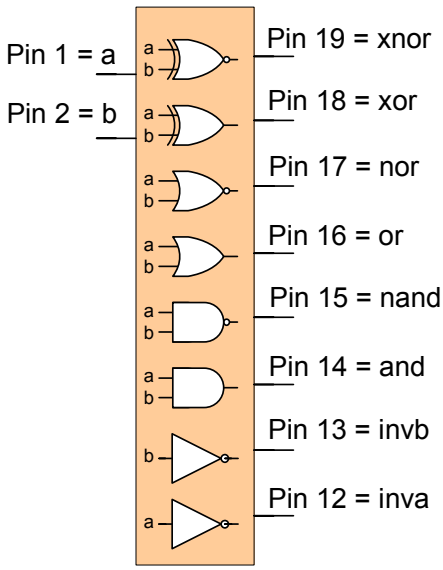
Podawane deklaracje wyprowadzeń w deklaracji PIN muszą być zgodne z rodzajem wyprowadzenia (WEJŚĆ, WYJŚĆ, WEJŚĆ/WYJŚĆ) dla użytego układu GAL16V8. Funkcje poszczególnych wyprowadzeń należy odczytać z jego schematu logicznego. Znajdziesz go w nocie katalogowej (*Specifications* ***GAL16V8*** *High Performance* E^2*CMOS PLD Generic Array Logic*™) znajdującej się na stronie internetowej *http://www.atmel.com/*.

Widok okna edytora tekstu z pełnym tekstem pliku źródłowego przedstawia rysunek 4.25.



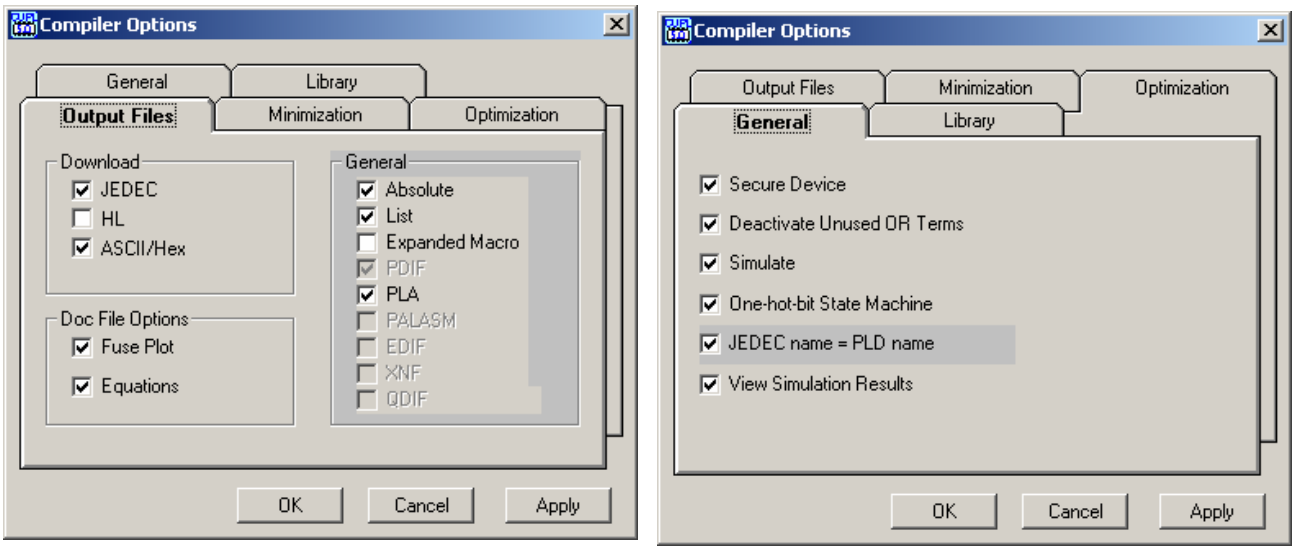
Rys. 4.25. Okno edytora tekstowego z plikiem źródłowym.

Opisana w pliku źródłowym struktura przedstawia osiem bramek logicznych. Schemat logiczny projektowanego układu przedstawia rysunek 4.26.



Rys. 4.26. Struktura omawianego przykładu.

Gdy opisane zostaną wszystkie elementy pliku źródłowego, należy wykonać kompilację pliku źródłowego *Bramki.PLD*. Istotną sprawą jest sprawdzenie parametrów kompilacji pliku źródłowego, które decydują o formacie pliku wynikowego oraz ewentualnej symulacji działania zrealizowanej logiki. Parametry kompilacji dostępne są po uruchomieniu opcji *Option/Compiller*. Wybór parametrów umożliwiający utworzenie pliku wynikowego w formacie *jedec*, przeprowadzenie symulacji oraz generację dokumentu z rysunkiem układu przedstawia rysunek 4.27.

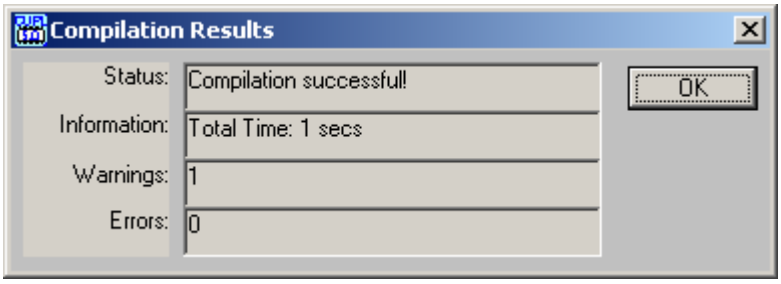


Rys. 4.27. Widok okna wyboru parametrów kompilacji.

Ustawienie parametrów w zakładce *General* – opcje *Simulate*, *JEDEC name = PLD name* oraz w zakładce *Output Files* w obszarze Downland opcji *JEDEC*, zaś w obszarze General opcji *Absolute*, spowoduje, że w wyniku kompilacji otrzymamy plik JEDEC z wektorami testowymi. Jeżeli w obszarze Doc File Options zakładki *Output Files* zaznaczamy opcje *Fuse Plot* i *Equations*, to otrzymamy dokument z rysunkiem układu. Bardzo pomocnym jest również zaznaczenie opcji *List* oraz *Expanded Macro* w obszarze General. Otrzymamy wówczas pełne informacje o procesie kompilacji.

Jeżeli dokonaliśmy wyboru opcji kompilatora, możemy dokonać kompilacji poleceniem *Run → Device Dependent Compile*, skrót F9 lub przycisk z paska narzędzi  . Ta funkcja skompiluje plik źródłowy, stworzy plik *Bramki.JED*, wykona symulację i doda do programu wektory testowe. Należy pamiętać, że jeżeli dokonaliśmy modyfikacji pliku, to należy go zapisać przed kompilacją.

Proces zakończenia kompilacji bez błędów jest potwierdzony komunikatem (rysunek 4. 28).



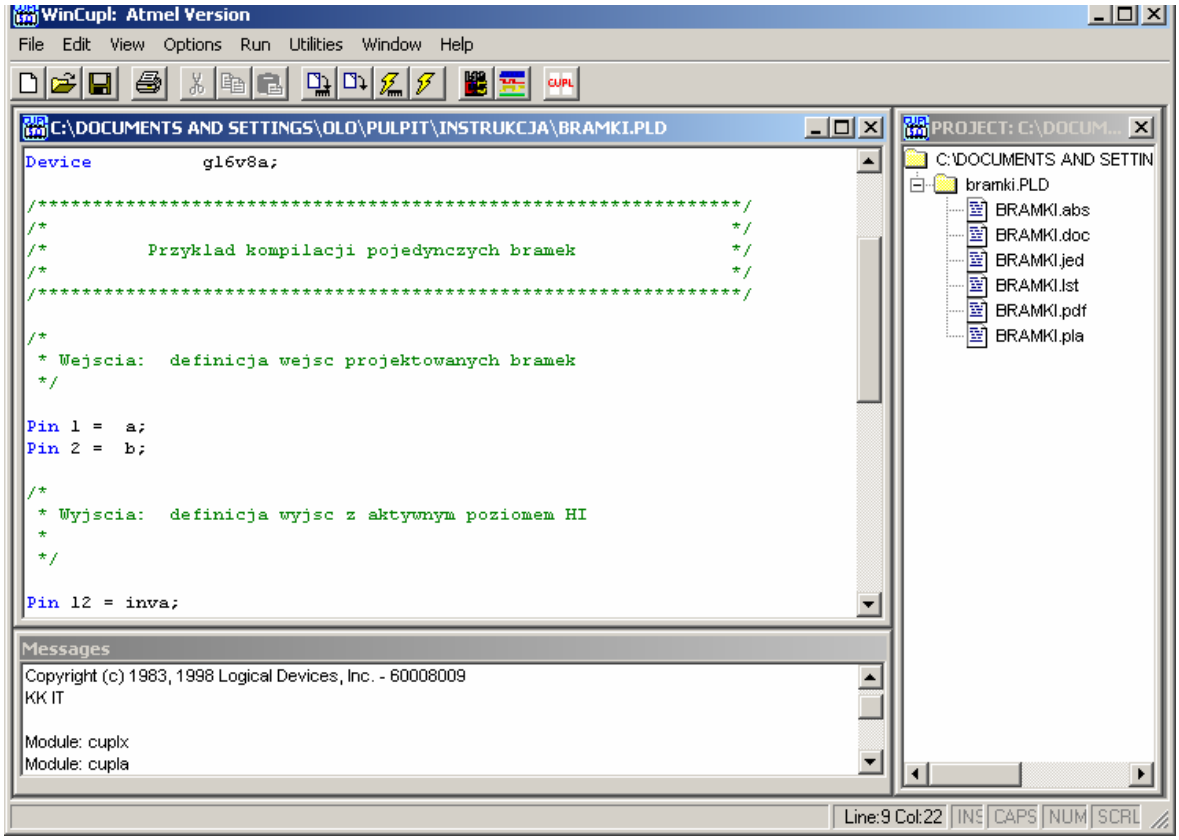
Rys. 4.28. Widok okna z komunikatem zakończenia kompilacji.

Gdy proces kompilacji przebiegnie bez błędów zostaną stworzone m.in. następujące pliki (rysunek 4.29):

- JEDEC (*Bramki.jed*) - plik wejściowy dla urządzenia programującego,
- plik bezwzględny (*Bramki*.abs) - do użycia z CSIM,
- plik błędów (*Bramki.lst*) - lista błędów w pliku źródłowym,

- plik dokumentacji (*Bramki.doc*) - zawiera rozszerzone równania logiczne i tablice symboli zmiennych,

- plik PLA (*Bramki.pla*) - używany przez filtry końcowe.

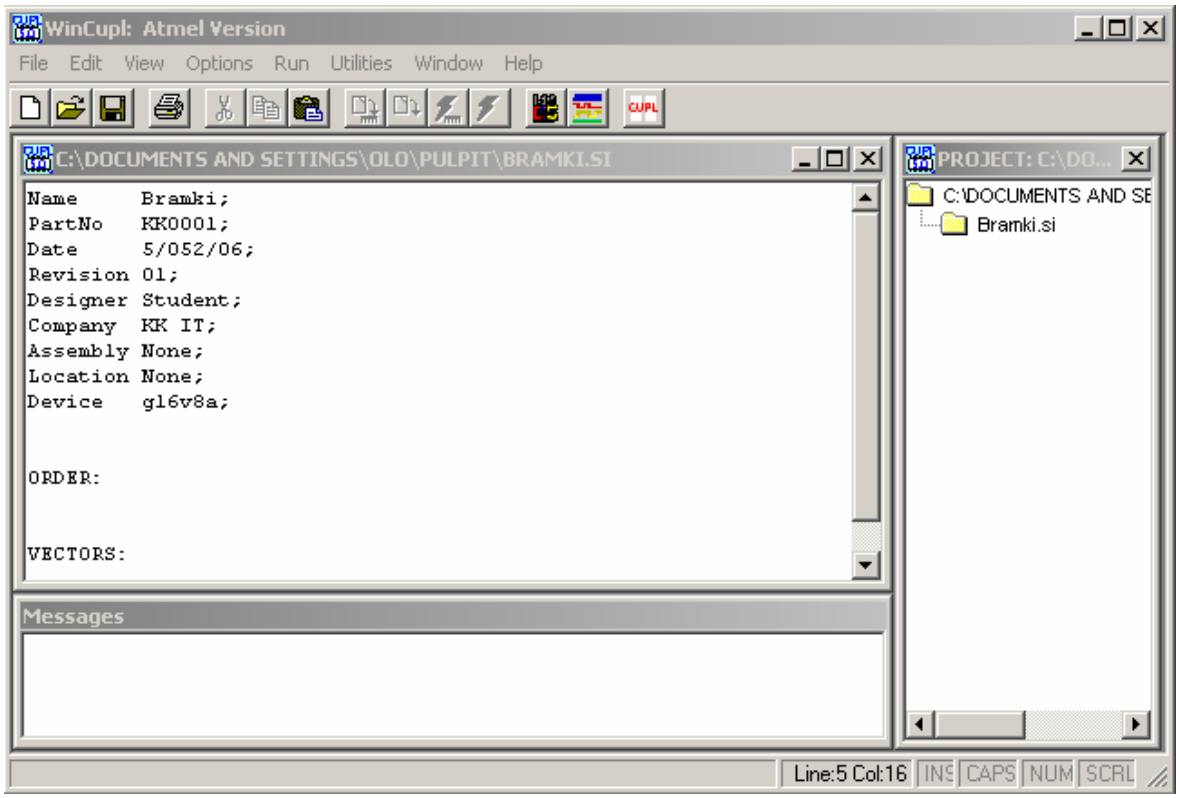


Rys. 4.29. Lista tworzonych plików.

Kolejnym etapem realizacji projektu jest jego symulacja. Uruchomienie symulatora realizujemy poprzez polecenie *Utilities* → *WinSim* lub przyciskiem z paska narzędzi  .

Nie należy uruchamiać symulatora bez uprzedniej kompilacji projektu. Uwaga ta dotyczy także wprowadzania poprawek do projektu: należy pamiętać, aby po modyfikacji pliku *Bramki.PLD* lub *Bramki.SI* wykonać kompilację przed wywołaniem WinSim. Uruchomienie symulatora otworzy domyślne okno główne aplikacji WinSim. Widok domyślnego okna głównego przedstawiony był na rysunku 4.13.

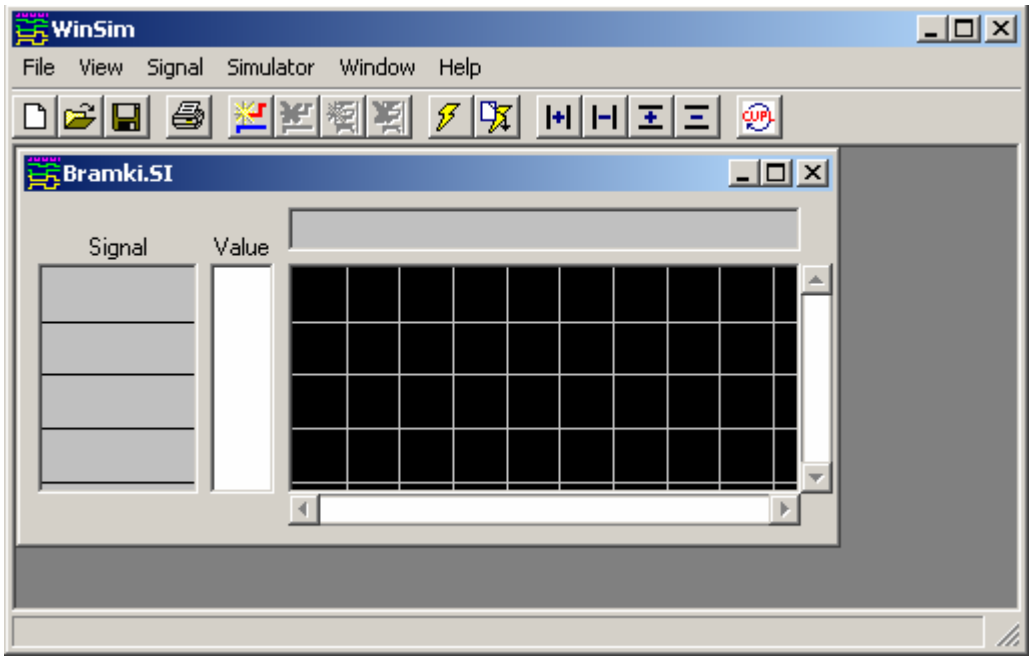
Po uruchomieniu symulatora rozpoczynamy od utworzenia pliku źródłowego z rozszerzeniem *Bramki.SI*. Widok okna głównego programu WinSim z pustym szablonem pliku źródłowego przedstawia rysunek 4.30.



Rys. 4.30. Szablon pliku źródłowego z rozszerzeniem *.SI.

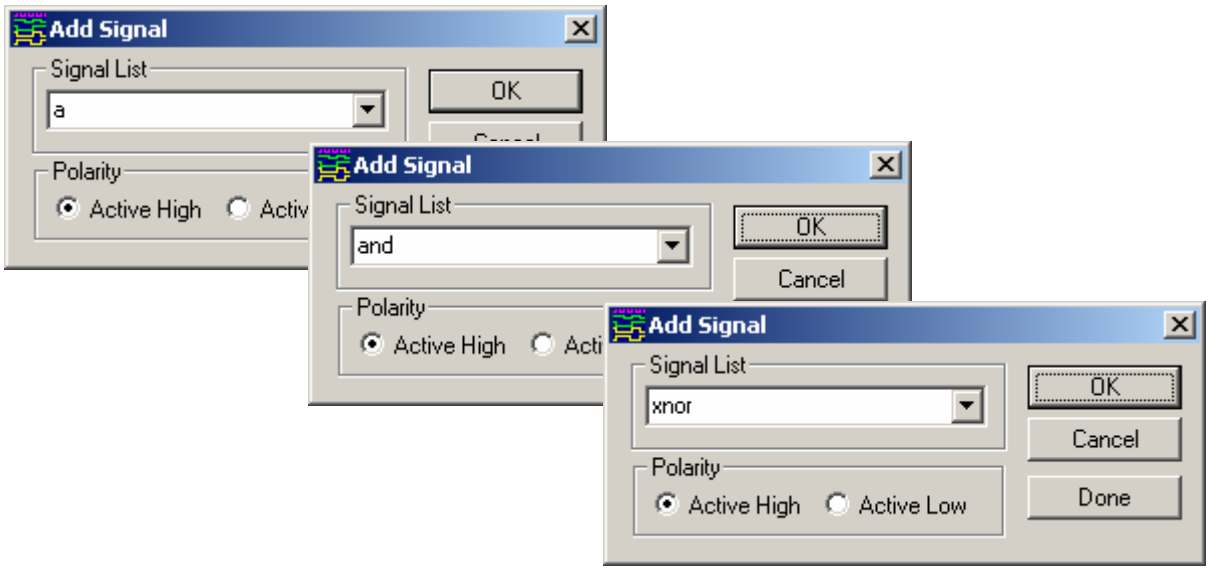
Polecenie *File→New* otwiera okno dialogowe kreatora nagłówka. Zamiast wprowadzać ponownie takie same parametry nagłówka, jak dla pliku źródłowego *Bramki.PLD*, można wybrać przycisk *Desing File…* i wskazać plik *Bramki.PLD* o odpowiedniej nazwie. Należy pamiętać, że dane w nagłówkach obu plików muszą być identyczne. W przeciwnym wypadu jest wysyłany komunikat ostrzegający o braku zgodności danych.

Po zatwierdzeniu danych pojawi się okno główne z otwartym oknem symulacji - rysunek 4.31.



Rys. 4.31. Otwarte okno symulacji.

Kolejnym krokiem jest dodanie śledzonych sygnałów oraz dodanie wektorów testowych. Śledzone sygnały – poszczególne zmienne - dodajemy poleceniem *Signal→Add Signal* lub przyciskiem z paska narzędzi  (rysunek 4.32). W pliku *Bramki.SI* zostanie dopisana instrukcja ORDER, która podaje, jakie sygnały projektu mają uczestniczyć w symulacji.



Rys. 4.32. Okno definiowania sygnałów.

Instrukcja ORDER

Instrukcja ORDER podaje kolejność występowania w wektorach testowych poszczególnych zmiennych. Deklaracja kolejności ma postać:

ORDER: zm_1, zm_2, output;

W deklaracji kolejności powinno stosować się te same identyfikatory co w specyfikacji układu. Jeżeli stosuje się skrócony zapis listowy, polaryzacja pierwszej zmiennej indeksowej determinuje polaryzację reszty zmiennych indeksowych.

Dla grupy sygnałów indeksowych można użyć skrótowej notacji listowej.

W deklaracji kolejności można wstawić stałe tekstowe, które oddzielają zmienne między sobą. Stała tekstowa będzie się pojawiać w wydruku w każdym wektorze. Stałe tekstowe należy ująć w znaki cudzysłowu:

„stala tekstowa”

Formatując wydruk można również zadeklarować wstawienie spacji za pomocą operatora %, po którym należy podać liczbę określającą liczbę spacji.

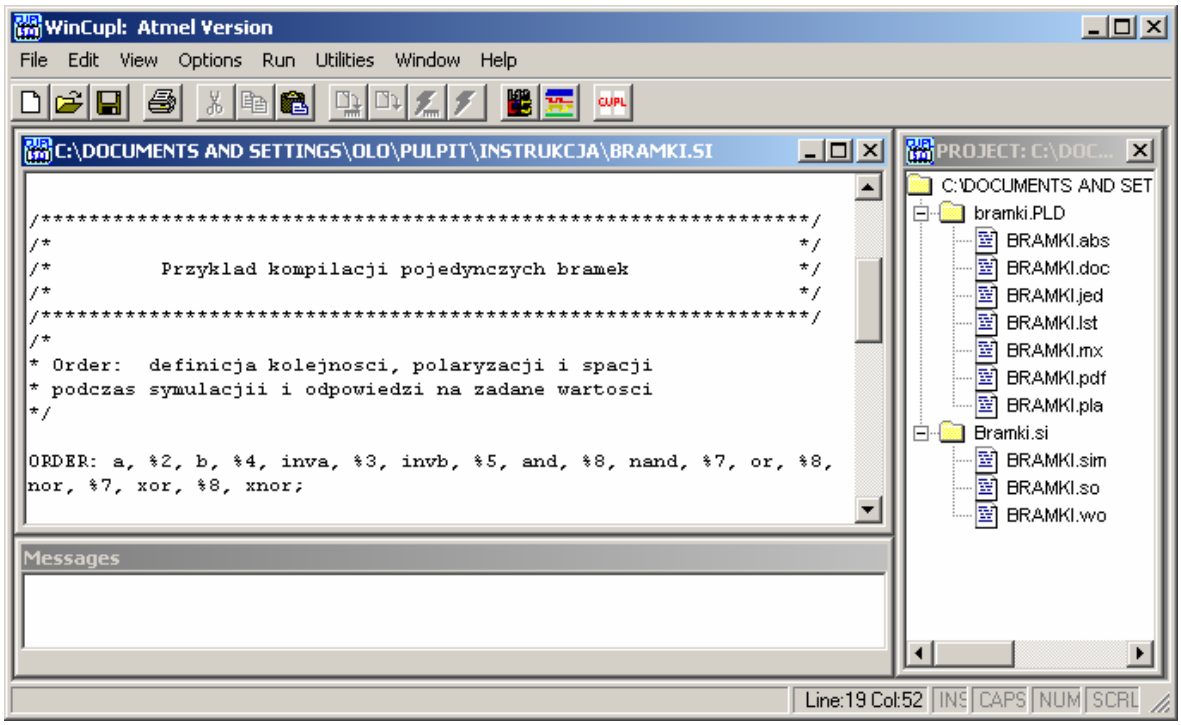
Przed deklaracją kolejności można wstawić podstawę systemu liczbowego (za pomocą słowa kluczowego BASE), który będzie się stosować do stałych liczbowych ujętych w apostrofy.[2]

Przykłady deklaracji kolejności:

```
ORDER: clock, input, output;
ORDER: A0, A1, A2, A3, SELECT, !OUT0, !OUT1;
ORDER: A0..3, SELECT, !OUT0..1;
ORDER: clock, %2, input, %2, output;
```

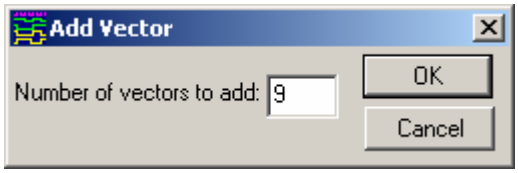
Dla rozpatrywanego przykładu postać instrukcji ORDER przedstawia rysunek 4.33.

[2] Łuba T., Markowski M. A., Zbierzchowski B. Kompilatory układów logicznych. Oficyna Wydawnicza PW. Warszawa 1995.



Rys. 4.33. Otwarte okno z instrukcją ORDER.

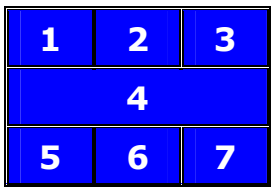
Dodawanie wektorów testowych (rysunek 4.34) spowoduje dopisanie instrukcji VECTORS w pliku źródłowym *Bramki.SI*. Każdy wektor testowy musi zawierać się w pojedynczej linii, bez znaku średnika. Można to uczynić poprzez wpisanie ich jako tekstu do pliku *Bramki.SI* po otwarciu go poleceniem *View→Source* lub skorzystać z interfejsu graficznego - polecenie *Signal→Add Victor* lub przycisk z paska narzędzi  .



Rys. 4.34. Okno definiowania wektorów.

Dodawanie wektorów testowych poprzez interfejs graficzny sprowadza się do klikania myszą w odpowiedni punkt wykresu (tabela 4.10). Należy pamiętać o zapisaniu zmian w pliku *Bramki.SI*.

Tabela 4.10. Wykorzystanie interfejsu graficznego do dodania wektorów testowych.

Sygnał/Wektor	*Obszar*	*Klawiatura*	*Wynik*
	1	*Nie*	*Wartość wejściowa HO (High)*
	2	*Nie*	*Wejście zegarowe (Low-High-Low)*
	3	*Nie*	*Wartość testowa (wyjściowa) High*
	4	*Nie*	*Wartość wysokiej impedancji (wejściowa i wyjściowa)*
	4	*Alt*	*Ustawienie stanu początkowego rejestr*
	4	*Shift*	*Wyjście nietestowane*
	4	*Ctr*	*Wartość nieokreślona*
	5	*Nie*	*Wartość wejściowa LO (Low)*
	6	*Nie*	*Wejście zegarowe (High-Low-High)*
	7	*Nie*	*Wartość testowa (wyjściowa) Low*

Instrukcja VECTORS

Instrukcja VECTORS rozpoczyna tę część pliku *.SI, w której podany zostaje ciąg wektorów z pobudzeniami testowymi. W wektorach określamy odpowiedni symbol dla każdego sygnału z listy ORDER - wejściowego i wyjściowego. Symbole dla sygnałów WEJŚĆ określają ich pobudzenia podczas symulacji, natomiast symbole dla sygnałów WYJŚĆ podają oczekiwane odpowiedzi.

Symbole dla sygnałów wejściowych (zadawane pobudzenia):

0 – zadaj stan niski (L) - *wartość wejściowa LO (Low).*

1 – zadaj stan wysoki (H) - *wartość wejściowa HO (High).*

C – zadaj impuls dodatni - *wejście zegarowe (Low-High-Low).*

K – zadaj impuls ujemny - *wejście zegarowe (High-Low-High).*

X – wartość nieokreślona.

P – ustawienie stanu początkowego rejestru.

‘ ‘ – wartość wejściowa, która będzie rozwinięta według zadanej podstawy liczbowej wyspecyfikowanej w deklaracji BASE, dozwolone znaki od 0 do F i X.

„ „ – wartość wyjściowa, która będzie rozwinięta według zadanej podstawy liczbowej wyspecyfikowanej w deklaracji BASE, dozwolone znaki od 0 do F, X oraz H, L, Z.

Symbole dla sygnałów wyjściowych (testy odpowiedzi):

L – testuj stan niski (L) - *wartość testowa (wyjściowa) Low.*

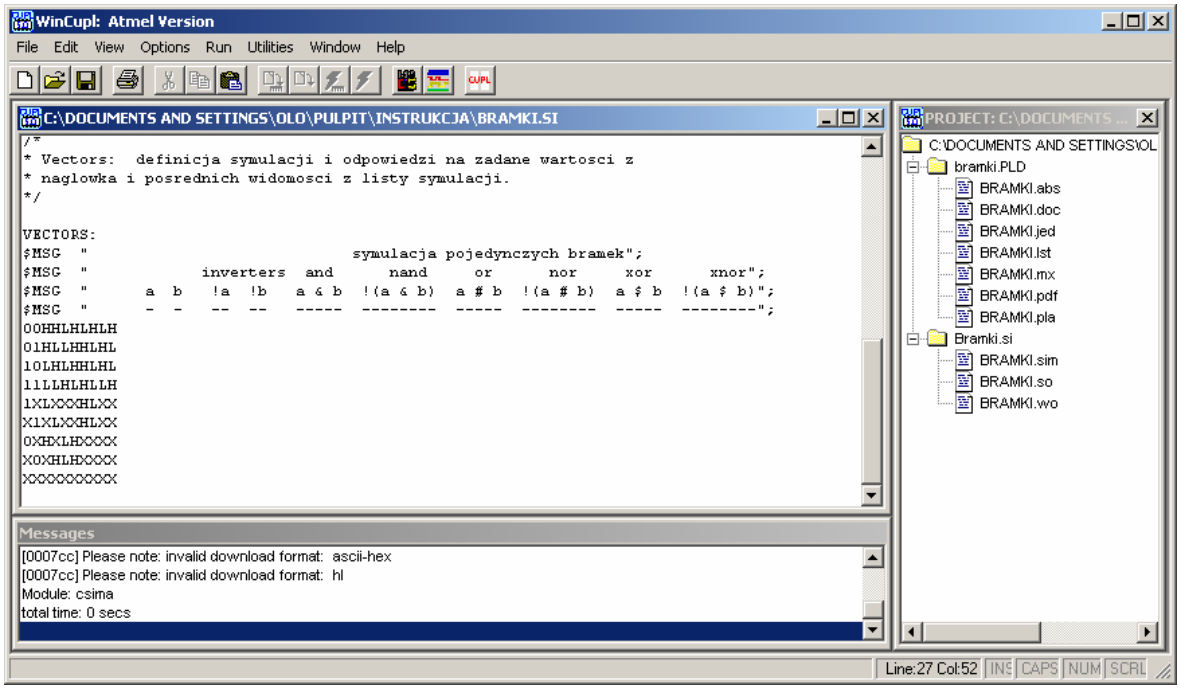
H – testuj stan wysoki (H) - *wartość testowa (wyjściowa) High.*

Z – testuj stan wysokiej impedancji - *wartość wysokiej impedancji (wejściowa i wyjściowa).*

N – nie testuj stanu - *wyjście nietestowane.*

* – wyznacz w symulacji stan sygnału i wstaw go do wektora - *wartość wstawiana przez symulator (tylko dla wyjść).*

Dla rozpatrywanego przykładu postać instrukcji VECTORS przedstawia rysunek 4.35.



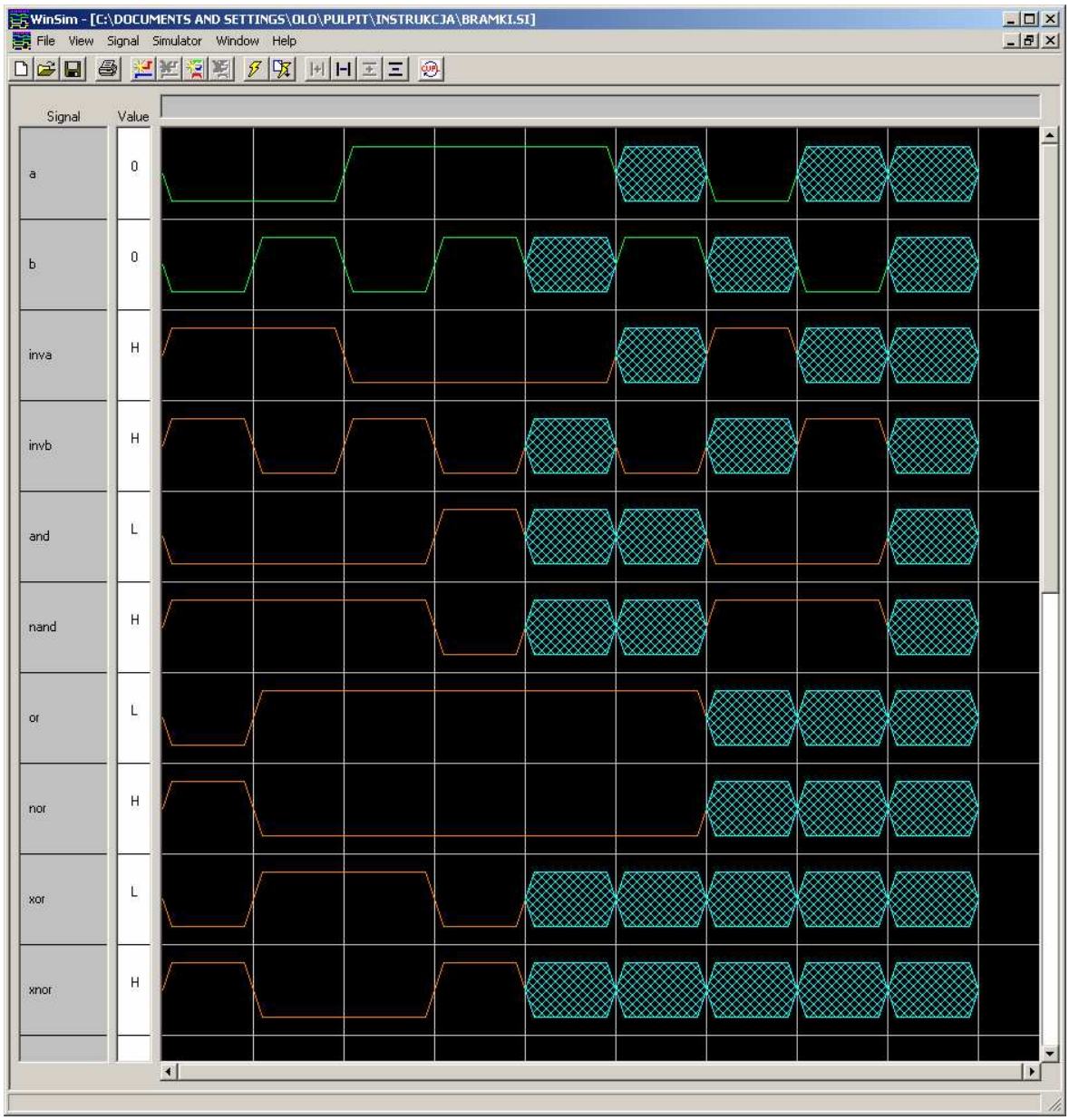
Rys. 4.35. Otwarte okno z instrukcją VECTORS.

Plik źródłowy *Bramki.SI* ma postać:

```
Name     Bramki;
PartNo   KK0001;
Date     20/04/06;
Revision 02;
Designer Student;
Company  KK;
Assembly None;
Location None;
Device   g16v8a;


/*************************************************************/
/*                                                           */
/*         Przyklad kompilacji pojedynczych bramek           */
/*                                                           */
/*************************************************************/
/*
* Order:  definicja kolejnosci, polaryzacji i spacji
* podczas symulacjii i odpowiedzi na zadane wartosci
*/

ORDER: a, %2, b, %4, inva, %3, invb, %5, and, %8, nand, %7,
or, %8, nor, %7, xor, %8, xnor;

/*
* Vectors:  definicja symulacji i odpowiedzi na zadane warto-
sci z naglowka i posrednich widomosci z listy symulacji.
*/

VECTORS:
$MSG  "                    symulacja pojedynczych bramek";
$MSG  "       inverters  and     nand     or      nor      xor     xnor";
$MSG  " a  b   !a  !b   a & b  !(a & b)  a # b  !(a # b)  a $ b  !(a $ b)";
$MSG  " -  -   --  --   -----  --------  -----  --------  -----  --------";
00HHLHLHLH
01HLLHHLHL
10LHLHHLHL
11LLHLHLLH
1XLXXXHLXX
X1XLXXHLXX
0XHXLHXXXX
X0XHLHXXXX
XXXXXXXXXX
```

Gdy plik *Bramki.SI* jest gotowy, należy uruchomić symulator poleceniem *Simulator→Run Simulator* lub przyciskiem z paska narzędzi  . Jeśli w pliku źródłowym nie było błędów, w oknie symulatora zostaną wyświetlone wykresy z przebiegami czasowymi.

Wykresy z przebiegami czasowymi dla projektu *Bramki* przedstawia rysunek 4.36.



Rys. 4.36. Wynik symulacji projektu *Bramki*.

4.4. Programator LabTool-48XP

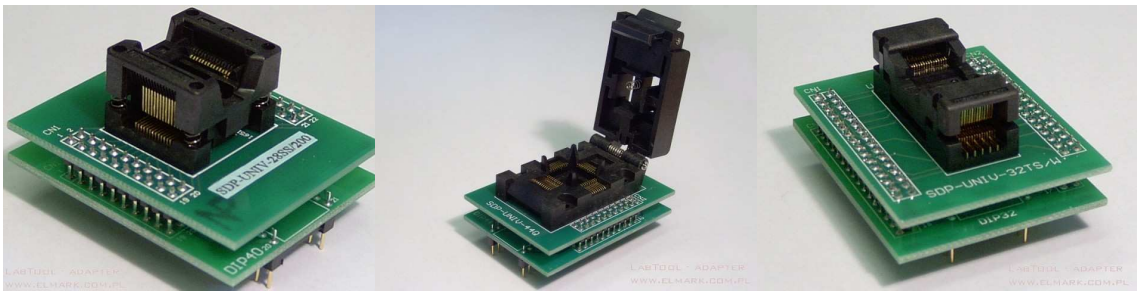
LabTool-48XP jest programatorem firmy Advantech Equipment cieszącym się dużą popularnością na całym świecie ze względu na dużą wydajność, uniwersalność, prostotę obsługi i niezawodność. Lista obsługiwanych przez niego układów zawiera ponad 5000 pozycji a producent zapowiada jej kwartalne rozszerzanie o ponad 100 nowych układów [3].

Rys. 4.37. Widok programatora LabTool48XP.

LabTool-48XP (rysunek 4.37) jest wysokiej jakości programatorem uniwersalnym współpracującym z komputerem PC przez port drukarki. Standardowo wyposażony jest w gniazdo 48-stykowe ZIF (*Zero Insertion Force*) do programowania układów w obudowach typu DIL. Każda nóżka gniazda programującego jest uniwersalna (można na nią podać 4 różne napięcia, masę, sygnał TTL, podciąg rezystorem PULL UP i PULL DOWN, szybkie sygnały - zegar, dane, adresy, sygnały strobujące - pozostawić w wysokiej impedancji, można również czytać jej stan logiczny). Dla układów w obudowach innych niż DIL (SO, SOIC, SSOP, TSOP, PSOP, PLCC, PQFP, BGA i innych) o różnych rozstawach nóżek oraz różnych szerokościach często z większą

[3] http://www.elmark.com.pl

liczbą wyprowadzeń niż 48 firma Advantech skonstruowała wiele rodzajów adapterów (rysunek 4.38). Ze względu na uniwersalność gniazda LabTool'a, wszystkie adaptery są jedynie prostymi przejściówkami nie zawierającymi elementów aktywnych. Zmniejsza to późniejsze ewentualne koszty rozbudowy stanowiska pracy. Opis adapterów (tablica połączeń) jest dostępny na stronie producenta *http://www.aec.com.tw.* Do obsługi większości układów pamięciowych wystarczy kilka rodzajów adapterów (44-pin PLCC, 48-pin TSOP, 40-pin TSOP i 32-pin TSOP).



Rys. 4.38. Przykłady adapterów programatora LabTool48XP.

Podstawowe parametry programatora znajdują się w Instrukcji obsługi Advantech Equipment Corporation.: LABTOOL-48XP. Intelligent Universal Programmer. User's Manual. Copyright Notice, którą znajdziesz na stronie internetowej *http://www.aec.com.tw/* firmy Advantech.

Programator (rysunek 4.37) posiada trzy diody LED informujące o stanie pracy programatora.

Dioda zielona (GOOD) - ostatnia operacja programowania przebiegła pomyślnie.

Dioda żółta (BUSY) - programator jest zajęty (wykonywana jest operacja na układzie zamontowanym w podstawce).

Uwaga! Nie wolno wkładać i wyjmować układu z podstawki, dopóki nie zaświeci się dioda zielona lub czerwona. Wkładanie lub wyjmowanie układu w czasie, gdy świeci się dioda żółta, może spowodować jego uszkodzenie.

Dioda czerwona (ERROR) - operacja programowania przebiegła nieprawidłowo.

Pulsowanie diody z częstotliwością 5Hz sygnalizuje, że układ scalony został zaprogramowany i zweryfikowany, oczekuje na wyjęcie i włożenie nowego układu. Opcja ta występuje w przypadku stosowania trybu programowania seryjnego *„Mass Production Mode”*, dzięki któremu można przyśpieszyć programowanie większej liczby układów.

Programator obsługiwany jest przez program pod kontrolą systemu Windows *LT48XP_660.EXE*. Ze strony producenta (*http://www.atmel.com)* lub dystrybutorów *(http://www.elmark.com.pl; www.labtool.com)* można pobrać (bezpłatnie) zawsze aktualną wersję oprogramowania obsługującego programator. Wszystkie polecenia programu są dostępne poprzez kliknięcie myszką, wykorzystując widoczne na ekranie menu. Często jednak sekwencję kliknięć można zastąpić użyciem klawiatury, przyspieszając w ten sposób wykonanie wielu czynności. Informacje na temat dostępnych klawiszy skrótu są dostępne w menu po jego rozwinięciu.

Oprogramowanie programatora, oprócz standardowych funkcji, które posiada każdy programator (*Read*, *Program*, *Blank Check*, *Verify*, *Erase*), posiada dodatkowe, które znacznie usprawniają pracę. Należą do nich m.in.:

- *Insertion Test* - test poprawności włożenia układu do podstawki, jak również połączenia każdej nóżki z odpowiadająca jej nóżką podstawki. Sprowadza się to do sprawdzania kontaktu między programatorem a układem. W czasie tego testu jest wykrywane każde błędne włożenie układu do podstawki (adaptera), przesunięcie układu w podstawce, odwrotne włożenie układu. Dzięki temu zapobiega się uszkodzeniom programatora i programowanego układu.

- *Auto ID* - automatyczne rozpoznawanie 8-bitowych pamięci Flash lub EPROM. Większość producentów pamięci wyposaża swoje układy w funkcje

odczytu ID producenta (numer identyfikacyjny składający się z dwóch bajtów) i ID układu, starając się nie powtarzać numeracji zajętej już przez innych producentów. Programator po odczytaniu ID układu porównuje je ze swoją bazą danych i pokazuje użytkownikowi do zatwierdzenia znaleziony układ. Wcześniej jest wykonywany dodatkowo test kontaktu (ze względów bezpieczeństwa).

- *Mass Production Mode* - tryb programowania seryjnego, dzięki któremu można przyśpieszyć programowanie większej liczby układów. Funkcja może być zintegrowana z automatycznym numerowaniem (nadawanie numeru seryjnego programowanym układom). Obsługujący sam ustala, pod jakimi adresami pamięci oprogramowanie może wstawić numer seryjny i o jaką wartość ma być inkrementowany. Programator pracujący w tym trybie automatycznie wykrywa obecność układu w gnieździe i jego wymianę po zaprogramowaniu. Na tej podstawie przechodzi do obsługi kolejnego układu. Można tu również wyłączyć autonumerowanie i wszystkie układy będą programowane jednakową zawartością.

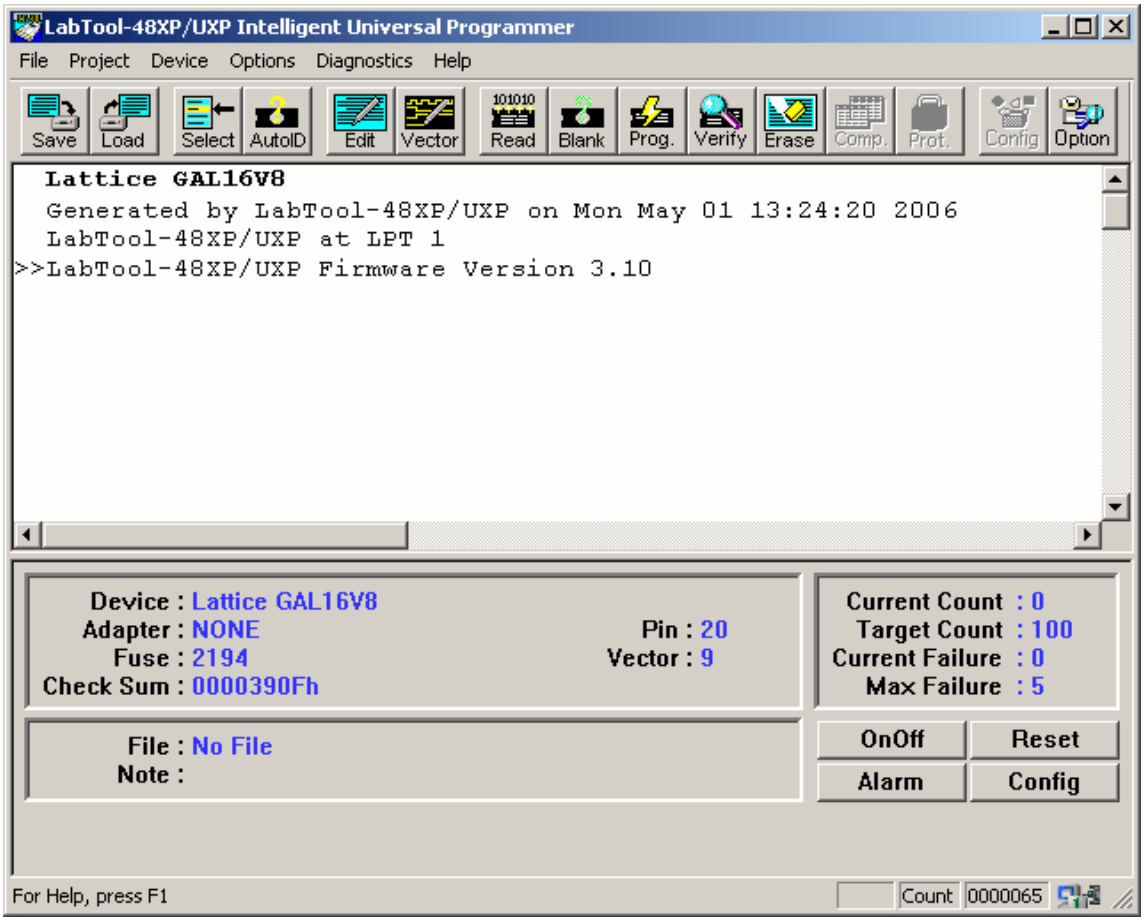
4.4.1. Praca z programatorem

Obsługa programatora LabTool-48XP/UXP nie wymaga żadnych skomplikowanych czynności. Przed przystąpieniem do pracy należy jedynie sprawdzić czy podłączenie programatora do komputera zrealizowane jest w sposób prawidłowy. Oprogramowanie sterujące automatycznie wykrywa podłączenie programatora do portu. Stosowny komunikat wyświetlany jest w polu tekstowym okna głównego programu.

Po uruchomieniu programu LabTool-48XP/UXP 

pojawia się główne okno programu sterującego pracą programatora. Widok domyślnego okna głównego programu przedstawia rysunek 4.39. W oknie przedstawione są komunikaty ukazujące gotowość programatora do pracy wraz z datą

użytkowania, numer portu równoległego/drukarkowego (LPT1) i wersja sprzętu (Firmware Version 3.10).



Rys. 4.39. Okno główne programu sterującego pracą programatora.

Na pasku tytułu wyświetlana jest nazwa programatora. Pasek menu zawiera listę poleceń umożliwiająca zaprogramowanie, wykonanie weryfikacji i zabezpieczenie układu. Pasek narzędzi zawiera przyciski, z których korzystamy zamiast najczęściej używanych poleceń menu (rysunek 4.40).

Rys. 4.40. Pasek menu i pasek narzędzi programu LabTool-48XP.

Polecenia menu głównego mogą być wybierane przy użyciu myszki lub klawisza ALT i wciśnięciu podkreślonej litery nazwy rozkazu (np. ALT-F dla

polecenia “File”). Po rozwinięciu menu poszczególne rozkazy wymagają tylko naciśnięcia klawisza odpowiadającego podkreślonej na ekranie literce.

Menu ***File*** (ALT-F)

Save Buffer (ALT-S) - zapis zawartości bufora do pliku na dysku.

Load File (ALT-L) - odczyt danych z pliku i wpisanie ich do bufora.

Exit (ALT-X) - wyjście z programu LabTool-48XP.

Menu ***Project*** (ALT-J)

Save Project (ALT-F1) - zapisuje aktualne ustawienia oprogramowania LabTool’a do wskazanego pliku projektu.

Load Project (ALT-F2) - przywraca ustawienia oprogramowania do takich, jak poprzednio używane i zapisane do wskazanego pliku projektu.

Menu ***Device*** (ALT-D)

Change (ALT-C) - wybór układu do programowania.

Auto Select EPROM (ALT-A) - automatyczne wyszukanie pamięci typu EPROM lub FLASH.

Mass Production Mode - tryb seryjnego programowania.

Edit (ALT-E) - edycja bufora.

Modify Vectors - modyfikacja wektora testującego (dostępna tylko przy wyborze układu PLD).

Read (ALT-R) - odczytanie zawartość układu i wpisanie jej do bufora.

Blank Check (ALT-B) - sprawdzanie czystości układu scalonego.

Program/Auto (ALT-P) - programowanie układu z aktualną zawartością bufora.

Verify (ALT-V) - weryfikacja poprawności zapisu bufora z zawartością układu scalonego.

Erase (Ctrl-F1) - kasowanie zawartości układu znajdującego się w podstawce.

Secure (ALT-U) - zabezpieczenie układu.

Compare (Ctrl-F3) - porównanie zawartości układu z zawartością bufora.

Configuration (ALT-G) - ustawienia konfiguracyjne rejestrów.

Menu ***Options*** (ALT-O)

Modify Programming Parameter (F3) - modyfikacja parametrów programowania układu.

Device Operation Options (F4) - ustawienia opcji operacji dla programowanego układu.

Paralel Port Selection (F5) - wybór portu równoległego.

Statistics (F6) - wybór funkcji statystycznych.

Menu ***Diagnostic***

Self Test (F7) - włączenie testów programatora.

Menu ***Help*** (ALT-H)

Help Topic (F1) - wyświetlenie tematów pomocy.

About - wyświetlenie informacji o aplikacji.

Pasek narzędzi

- zapisuje aktualne ustawienia oprogramowania LabTool'a do wskazanego pliku projektu,

- przywraca ustawienia oprogramowania do takich, jakie były poprzednio używane i zapisane do wskazanego pliku projektu,

- wybór układu do programowania,

- automatyczne wyszukanie i wybór pamięci EPROM lub FLASH,

- edycja bufora,

- modyfikacja wektora testującego,

- odczyt zawartości układu i zapisanie jej do bufora,

- sprawdzanie czystości układu scalonego,

- programowanie układu aktualną zawartością bufora,

- weryfikacja poprawności zapisu bufora z zawartością układu scalonego,

- kasowanie zawartości układu znajdującego się w podstawce,

- porównanie zawartości układu z zawartością bufora,

- zabezpieczenie zawartości układu,

- ustawienia konfiguracyjne rejestrów,

- ustawienia opcji operacji dla programowanego układu.

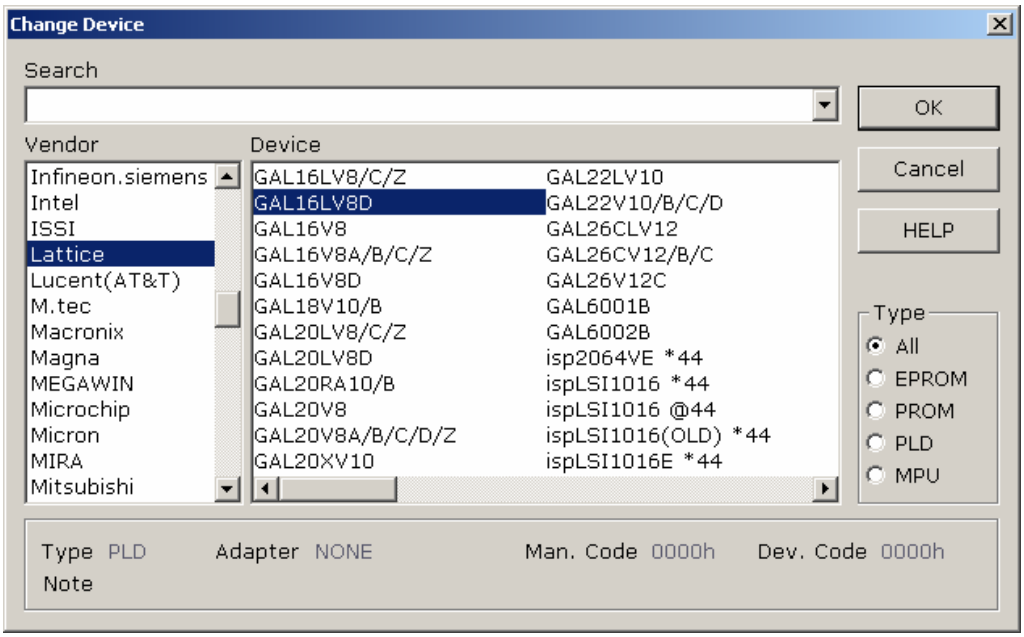
4.4.2. Programowanie układu

W celu zaprogramowania układu należy wykonać następujące czynności:

1. Wybrać układ programowany.

Użyj polecenia z menu *Device→Change* lub klawiszy skrótu „ALT-C” (ikona Select 

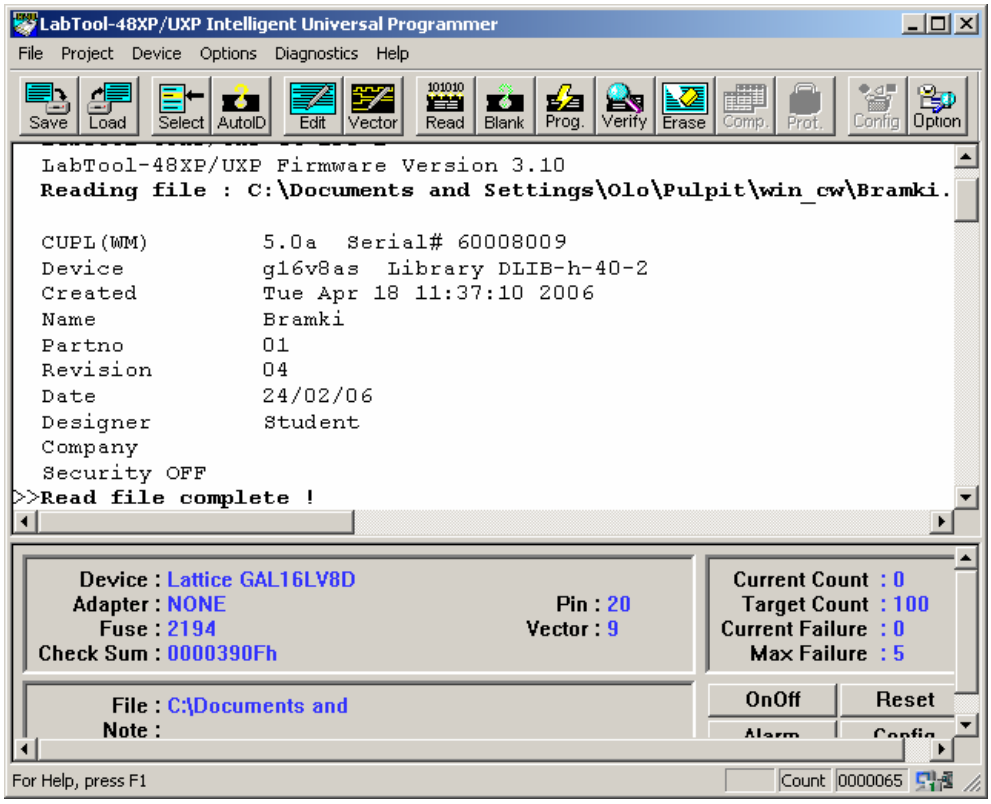
na pasku narzędzi). Wpisz pełną nazwę układu scalonego, który chcesz programować. Operację ta można wykonać używając myszki, dokonując wyboru układu z listy (rysunek 4.41).



Rys. 4.41. Okno *Change Device* wyboru układu.

2. Załaduj plik projektu do zapisania w układzie.

Użyj polecenia z menu *File→Load File* lub klawiszy skrótu „ALT-L” (ikona Load na pasku narzędzi). Wybierz plik do załadowania z rozszerzeniem JEDEC (rysunek 4.42). Plik typu JEDEC zawiera spis połączeń w macierzy elementów, reprezentowany przez adres oraz serię znaków „1” bądź „0” odpowiednio sygnalizujących obecność połączenia lub jego brak. Zawiera również wektory testowe (lista pinów oraz wartości każdego z nich dla każdego kroku testowego) umożliwiające przeprowadzenie przez programator testów funkcjonalności projektu.

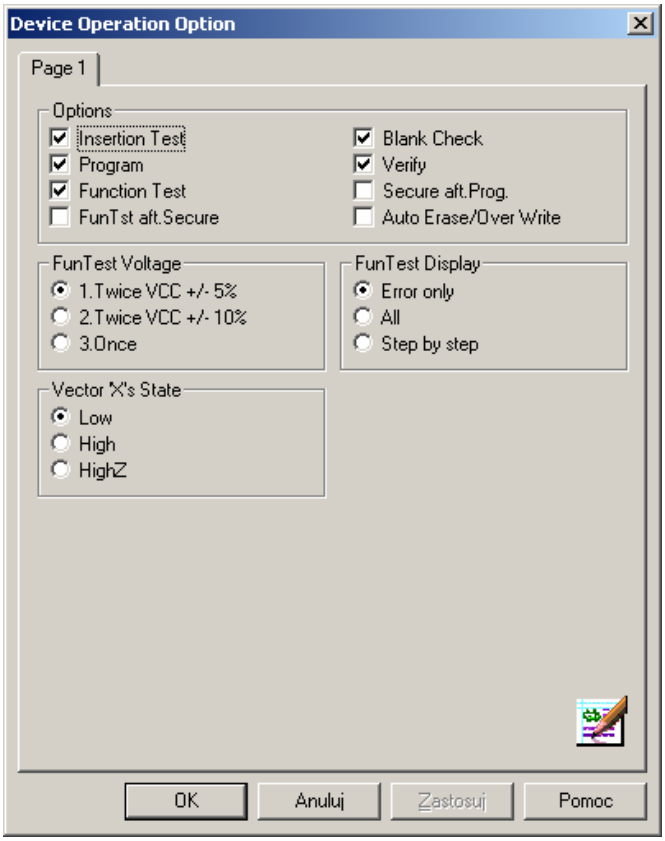


Rys. 4.42. Okno główne z załadowanym projektem (format jedec).

Opcjonalnie, można załadować bufor zawartością danych odczytanych z innego zaprogramowanego układu. Operację odczytu zawartości układu dokonujemy poleceniem z menu *Device→Read* lub przy użyciu klawiszy skrótu „ALT-R” (ikona Read na pasku narzędzi).

3. Włóż do podstawki ZIF czysty (*blank*) układ przeznaczony do zaprogramowania. Użyj polecenia z menu *Options→Operation* lub klawiszy skrótu „F4” (ikona Option na pasku narzędzi). Następnie wybierz opcje operacji dla programowanego układu (rysunek 4.43).

Po wykonaniu tej czynności, włączamy programowanie używając klawiszy skrótu „ALT-P” lub ikona Prog. na pasku narzędzi.

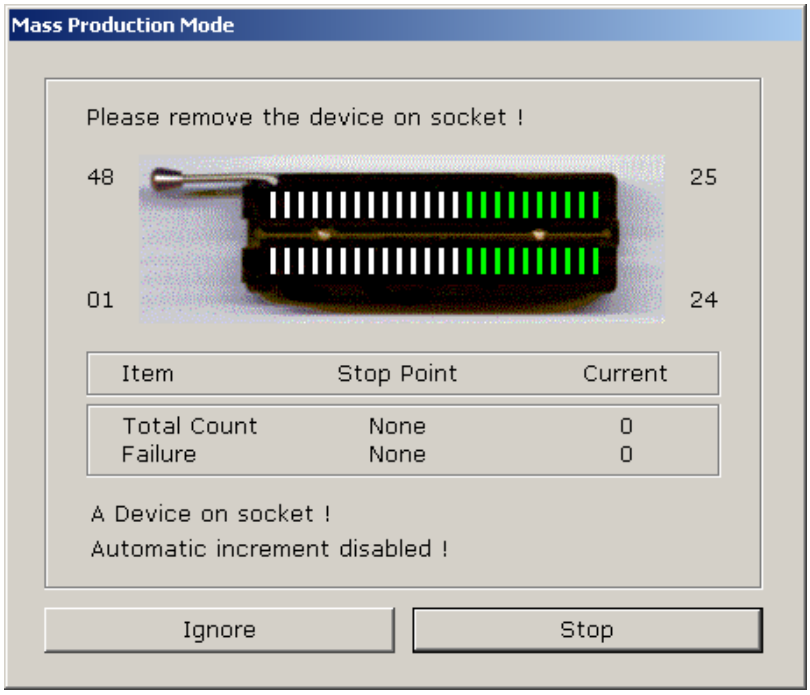


Rys. 4.43. Okno ustawienia opcji dla programowanego układu.

4. W przypadku programowania większej liczby układów należy zmienić tryb programowania na tryb programowania seryjnego *Mass production mode*, dzięki któremu można przyśpieszyć programowanie. Użyj polecenia z menu *Device→ Mass production mode* (rysunek 4.44). Po wejściu w ten tryb programowania, wszystkie funkcje (skróty) klawiatury i funkcje myszy są nieaktywne. Po włączeniu tej opcji LabTool-48XP/UXP biedzie automatycznie programować układ po poprawnym umieszczeniu w podstawce ZIF.

Zaprogramowanie układu jest sygnalizowane mruganiem zielonej diody LED. Wtedy należy wyjąć z podstawki zaprogramowany układ i włożyć do podstawki nowy. Po włożeniu do podstawki nowego układu, programator najpierw sprawdza poprawność kontaktów, ID i następnie przechodzi do programowania. Dzięki zablokowaniu funkcji klawiatury i myszy, nie istnieje ryzyko nieoczekiwanej zmiany zawartości bufora.

Pamiętaj, że nie wolno wyjmować lub wkładać układu do podstawki, dopóki nie zaświeci się dioda zielona lub czerwona, zgaśnie żółta.



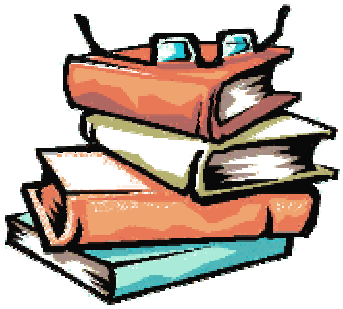
Rys. 4.44. Okno trybu programowania seryjnego.

5. Jeśli układ posiada konfigurowalne opcje oscylatora, watchdoga, zabezpieczenia, itp. Możesz włączyć konfigurowanie tych ustawień skrótem „ALT-G” (ikona Config ).

Opcje te są dostępne tylko wtedy, gdy układ ma takie możliwości.

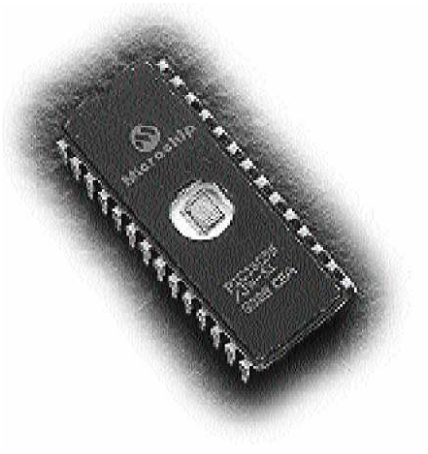
Programowanie konfiguracji lub zabezpieczenia układu może być także wykonywane po wciśnięciu ikony Proct  lub wraz z operacją programowania układu *Programm*. Konfigurację i zabezpieczenie układu w operacji programowania osiągamy używając polecenia z menu *Options→Secure* lub

klawiszy skrótu „ALT-U”. Należy zaznaczyć okno wielokrotnego wyboru „Secure aft. Prog”. Zapisanie zabezpieczenia odczytu w układach posiadających tą funkcję spowoduje brak możliwości weryfikacji poprawności programowania układu oraz odczytu zawartości układu przez programator.



4.5. Literatura do rozdziału 4

1. Advantech Equipment Corp.: LABTOOL-48XP. Intelligent Universal Programmer. User's Manual. Printed in Taiwan. July 2002.
2. Atmel: WinCUPL. User's Manual. Copyright Atmel Corporation 1997.
3. Atmel: ABEL/CUPL. Design File Conversion. Aplication Note. Atmel Corporation 2002.
4. Atmel: WinCUPL.Tutorial 1.
5. CUPL Programmers Reference Guide. Atmel Corporation. Library, ETC.
6. Łuba T., Markowski M. A., Zbierzchowski B. Kompilatory układów logicznych. Oficyna Wydawnicza PW. Warszawa 1995.
7. Pasierbiński J., Zbysiński P.: Układy programowalne w praktyce. WKŁ. Warszawa 2001.
8. Zbysiński P., Pasierbiński J.: Układy programowalne - pierwsze kroki. Wyd. II. BTC. Warszawa 2004.



ISBN 83-912031-7-4
ISBN 978-83-912031-7-0