Artur Cichowski
Paweł Szczepankowski
Wojciech Śleszyński

TECHNIKA CYFROWA I MIKROPROCESOROWA

LABORATORIUM

Wydawnictwo Politechniki Gdańskiej

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Materiał został przygotowany w związku z realizacją projektu
pt. „Zamawianie kształcenia na kierunkach technicznych,
matematycznych i przyrodniczych – pilotaż” współfinansowanego
ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego
nr umowy: 46/DSW/4.1.2/2008 – zadanie 018240 w okresie od 21.08.2008 – 15.03.2012

Artur Cichowski
Paweł Szczepankowski
Wojciech Śleszyński

TECHNIKA CYFROWA I MIKROPROCESOROWA

LABORATORIUM

Gdańsk 2011

PRZEWODNICZĄCY KOMITETU REDAKCYJNEGO
WYDAWNICTWA POLITECHNIKI GDAŃSKIEJ
Romuald Szymkiewicz

RECENZENT
Janusz Nieznański

PROJEKT OKŁADKI I SKŁAD
Katarzyna Olszonowicz

Wydano za zgodą
Rektora Politechniki Gdańskiej

ISBN 978–837348–400–9

WYDAWNICTWO POLITECHNIKI GDAŃSKIEJ

Wydanie I. Ark. wyd. 10,3, ark. druku 10,75, 981/658

Spis treści

Wstęp

Skrypt zawiera czternaście ćwiczeń laboratoryjnych przeznaczonych do realizacji w ramach przedmiotów związanych z techniką cyfrową i mikroprocesorową. Pierwsze siedem ćwiczeń dotyczy techniki cyfrowej. Są one realizowane w środowisku Quartus II z wykorzystaniem zestawu dydaktycznego DE2 firmy Altera. Kolejne siedem ćwiczeń z techniki mikroprocesorowej realizowanych jest w środowiskach WinAVR, AVRStudio i VMLAB przy wykorzystaniu zestawu laboratoryjnego AVR_edu.

Poszczególne ćwiczenia laboratoryjne zawierają przykładowe projekty umożliwiające analizę działania podstawowych układów logicznych oraz rozwiązań sprzętowych lub programowych typowych problemów. Skrypt zawiera także dużą liczbę zadań problemowych przeznaczonych do samodzielnego rozwiązania lub wymagających modyfikacji przykładowych projektów. W celu realizacji ćwiczenia laboratoryjnego należy zapoznać się przed zajęciami z częścią teoretyczną, zawierającą podstawowe informacje dotyczące tematu ćwiczenia oraz z zadaniami problemowymi przeznaczonymi do rozwiązania. Często wymagane będzie również opracowanie rozwiązania niektórych zadań w domu, co będzie wiązało się z przygotowaniem schematów lub programów i wykonaniem symulacji. Środowiska programistyczne używane w trakcie zajęć są darmowe i umożliwiają symulacyjne testowanie własnych rozwiązań. Zajęcia laboratoryjne przeznaczone są głównie na uruchamianie i testowanie projektów z wykorzystaniem zestawów laboratoryjnych.

Wykorzystywane w pierwszej części zajęć laboratoryjnych środowisko Quartus II umożliwia tworzenie, symulację oraz uruchamianie projektów dla układów programowalnych (ang. *Programmable Logic Devices* – PLD) firmy Altera. Zastosowany zestaw dydaktyczny DE2 z programowalną matrycą bramkową (ang. *Field Programmable Gate Array* – FPGA) i środowisko Quartus II oferują ogromne możliwości rozwijania i doskonalenia umiejętności z techniki cyfrowej i przetwarzania sygnałów, zaczynając od poznawania podstawowych funkcji i układów logicznych po tworzenie zaawansowanych projektów. Zestaw uruchomieniowy zawiera m.in. układ FPGA z rodziny Cyclone II (EP2C35F672C6), pamięć ulotną SDRAM (ang. *Synchronous Dynamic Random Access Memory*) i nieulotną typu flash, przyciski, przełączniki, diody LED, wyświetlacze siedmiosegmentowe, wyświetlacz alfanumeryczny, gniazdo karty pamięci SD, kodek audio, interfejsy komunikacyjne RS-232, USB, Ethernet, IrDA.

W ramach zajęć laboratoryjnych z podstaw techniki cyfrowej przewidziano następujące ćwiczenia:

- tworzenie, symulację i uruchamianie projektów w środowisku Quartus II – ćwiczenie 1,
- badanie działania bramek i przerzutników – ćwiczenie 2,
- projektowanie, implementację i uruchamianie układów kombinacyjnych – ćwiczenie 3,
- badanie działania, zastosowanie i projektowanie: rejestrów, dekodera adresów oraz multipleksera – ćwiczenie 4,
- badanie działania, zastosowanie i projektowanie: liczników i dzielników częstotliwości – ćwiczenie 5,
- podstawy języka opisu sprzętu VHDL – ćwiczenie 6,
- projektowanie, implementację i uruchomienie synchronicznego układu sekwencyjnego – ćwiczenie 7.

Ćwiczenia z zakresu techniki mikroprocesorowej polegają na programowaniu mikrokontrolera AVR w języku C z wykorzystaniem biblioteki AVR Libc. Podstawą realizacji sformułowanych w skrypcie zadań jest umiejętność programowania w języku C w stopniu podstawowym. Podczas opracowywania ćwiczeń laboratoryjnych kierowano się wyborem zagadnień o jak największym znaczeniu praktycznym. Kolejne ćwiczenia realizowane w ramach omawianego bloku programowego wykorzystują w dużym stopniu oprogramowanie opracowane w poprzednich ćwiczeniach.

W drugiej części ćwiczeń laboratoryjnych będzie wykorzystywany darmowy kompilator języka C dla mikrokontrolerów AVR – AVR-GCC, który wraz z edytorem kodów źródłowych – Programmers Notepad, zawarty jest w środowisku WinAVR. W celu symulacji i uruchomienia własnych programów będzie używane środowisko AVRStudio firmy Atmel oraz dodatkowo podczas symulacji – programy HAPSIM i VMLAB (ang. *Simulator Visual Micro Lab*). Platformę sprzętową stanowi zestaw laboratoryjny AVR_edu opracowany w Katedrze Energoelektroniki i Maszyn Elektrycznych przez autorów skryptu. Zestaw dydaktyczny AVR_edu zawiera popularny mikrokontroler AVR – ATmega128 wraz z często stosowanymi w technice mikroprocesorowej peryferiami takimi jak: przyciski, przełączniki, diody sygnalizacyjne LED, wyświetlacze siedmiosegmentowe LED, wyświetlacz alfanumeryczny i graficzny, zegar czasu rzeczywistego, kartę SD, interfejsy komunikacyjne RS232 i USB oraz inne.

Program ćwiczeń z techniki mikroprocesorowej jest następujący:

tworzenie, symulacja i uruchamianie projektów w środowisku WinAVR, AVRStudio, HAPSIM i VMLAB, obsługa portów mikrokontrolera oraz operacje bitowe w języku C – ćwiczenie 8,

tworzenie, symulacja i uruchamianie projektów wykorzystujących operacje bitowe i obsługę portów mikrokontrolera – ćwiczenie 9,

modyfikacja i uruchomienie oprogramowania zawierającego podprogram obsługi przerwania czasowego, obsługa zestawu wyświetlaczy siedmiosegmentowych LED – ćwiczenie 10,

uzupełnienie oprogramowania z poprzedniego ćwiczenia o programową realizację zegara – ćwiczenie 11,

obsługa wyświetlacza alfanumerycznego – ćwiczenie 12,

obsługa przetwornika analogowo-cyfrowego oraz licznika w trybie pracy modulacji szerokości impulsów – ćwiczenie 13 i 14.

1. Wprowadzenie do środowiska Quartus II

1.1. Cel ćwiczenia

Ćwiczenie ma charakter wprowadzający do systemu projektowego Quartus. Celem ćwiczenia jest zapoznanie się z zasadami tworzenia, symulacji i testowania projektów w środowisku Quartus II.

1.2. Wprowadzenie

W instrukcji przedstawiono poszczególne etapy tworzenie nowego projektu w środowisku Quartus II wraz z krótkimi komentarzami dotyczącymi podstawowych ustawień konfiguracyjnych. Wymieniono najczęściej wykorzystywane podczas zajęć laboratoryjnych zasoby biblioteczne omawianego środowiska.

Środowisko Quartus wyposażone jest obszerną pomoc. Dostęp do pomocy można uzyskać, korzystając z menu *Help > Contents* lub po wciśnięciu przycisku F1.

1.3. Metody definiowania układów cyfrowych

System Quartus II umożliwia przygotowanie opisu układu cyfrowego w postaci:

— pliku tekstowego w języku opisu sprzętu: VHDL, VERILOG, AHDL,
— schematu z wykorzystaniem symboli układów cyfrowych,
— przebiegów czasowych,

Na zajęciach laboratoryjnych najczęściej będzie wykorzystywany drugi ze sposobów.

1.4. Wybrane elementy biblioteczne dostępne w środowisku Quartus

Poniżej przedstawiono wybrane elementy biblioteczne dostępne w środowisku Quartus.

1.4.1. Elementy podstawowe (primitives)

1.4.1.1. Przerzutniki (*primitives/storage*)

Opis	Oznaczenie
Przerzutnik typu D	*dff* *dffe*
Przerzutnik typu RS	*srff* *srffe*
Przerzutnik typu JK	*jkffe* *jkff*
Przerzutnik typu T	*tff* *tffe*
Zatrzask	*dlatch*

1.4.1.2. Bramki (*primitives/logic*)

Opis	Oznaczenie
AND	*and2...and12*
NAND	*nand2...nand12*
NOT	*not*
OR	*or2...or12*
NOR	*nor2...nor12*
EX-OR	*xor*
EX-NOR	*xnor*
OR z zanegowanymi wejściami	*bor2...bor12*
NOR z zanegowanymi wejściami	*bnor2...bnor12*
AND z zanegowanymi wejściami	*band2...band12*

1.4.1.3. Porty wejścia/wyjścia (*primitives/pin*)

Opis	Oznaczenie
dwukierunkowy	*bidir*
wejściowy	*input*
wyjściowy	*output*

1.4.1.4. Bufory (*primitives/buffers*)

Opis	Oznaczenie
zmiana nazwy linii sygnałowej	*wire*
trójstanowy	*tri*

1.4.1.5. Inne (*primitives/other*)

Opis	Oznaczenie
stała	*constant*
masa sygnałowa (zero logiczne)	*gnd*
zasilanie (jedynka logiczna)	*vcc*

1.4.2. Makromoduły (others/maxplus2)

— układy z rodziny 74xxx,
— sumatory,
— bufory,
— rejestry,
— rejestry przesuwające,
— zatrzaski,
— jednostki ALU,
— komparatory,
— konwertery,

— liczniki,
— dekodery,
— kodery,
— dzielniki częstotliwości,
— mnożarki,
— multipleksery.

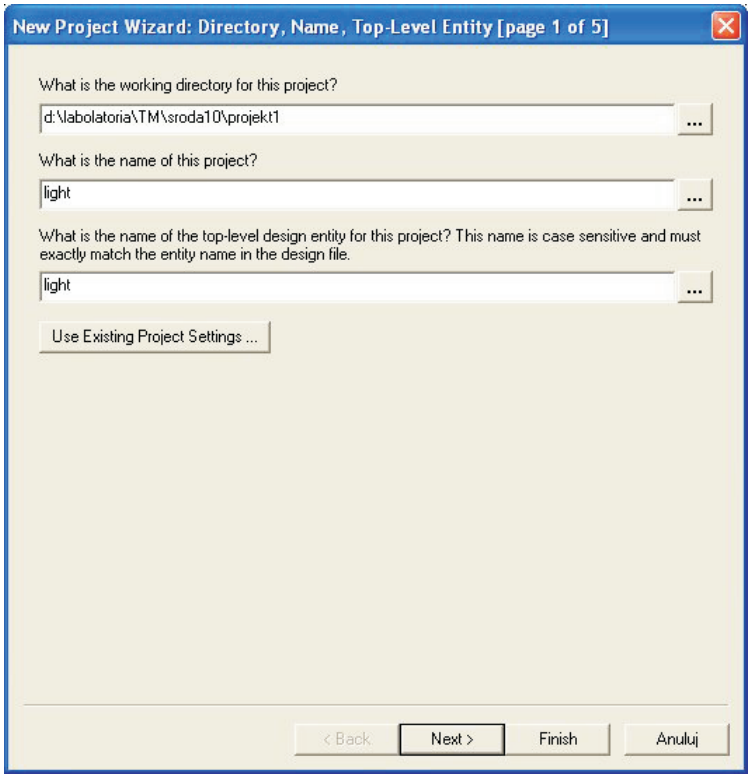
1.4.3. Funkcje parametryzowane lpm (megafunctions)

Oznaczenie	Opis
lpm_and *lpm_or* *lpm_xor*	Bramki wielobitowe (*megafunctions/gates*)
lpm_bustri	Wielobitowy bufor trójstanowy (*megafunctions/gates*)
lpm_clshift	Uniwersalny rejestr przesuwający z przesuwaniem logicznym, arytmetycznym lub obrotem (*megafunctions/gates*)
lpm_constans	stała (*megafunctions/gates*)
lpm_decode	Dekoder 1 z N (*megafunctions/gates*)
lpm_inv	Wielobitowy inwerter (*megafunctions/gates*)
busmux	Multiplekser grupowy „2 na 1” (*megafunctions/gates*)
lpm_mux	Multiplekser grupowy „N na M” (*megafunctions/gates*)
mux	Pojedynczy multiplekser „N na 1” (*megafunctions/gates*)
lpm_abs	Funkcja wartości bezwzględnej (*megafunctions/arithmetic*)
lpm_add_subb	Wielobitowy sumator/subtraktor (*megafunctions/arithmetic*)
lpm_compare	Komparator dwóch liczb n-bitowych (*megafunctions/arithmetic*)
lpm_counter	Wielobitowy licznik z różnymi wariantami sterowania (*megafunctions/arithmetic*)
lpm_mult	Wielobitowy multiplikator (*megafunctions/arithmetic*)
divide	Wielobitowa dzielarka (*megafunctions/arithmetic*)
lpm_ram_dq	RAM z rozdzielonymi portami zapisu i odczytu (pamięć dwuportowa) (*megafunctions/storage*)
lpm_ram_io	RAM z jednym portem dwukierunkowym (*megafunctions/storage*)
lpm_rom	ROM (tablica stałych wartości) (*megafunctions/storage*)
csdpram	Pamięć RAM jednocyklowa z oddzielonymi portami wejścia-wyjścia (*megafunctions/storage*)
csfifo	Pamięć FIFO (*megafunctions/storage*)

1.5. Tworzenie nowego projektu

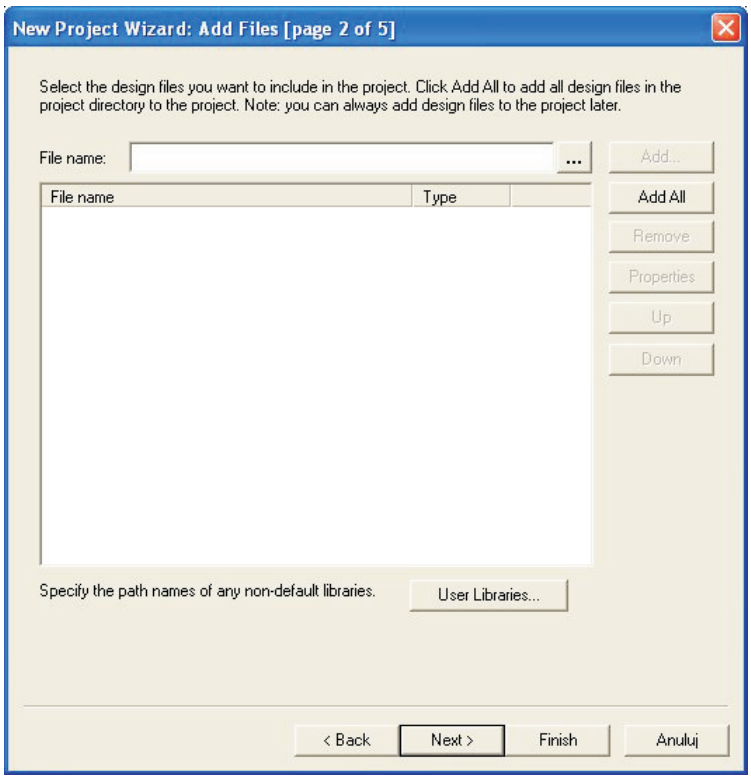
Projekty w środowisku Quartus mają budowę hierarchiczną, dla której jeden blok programu może zawierać inne. Tworzenie nowego projektu zostanie opisane z wykorzystaniem kreatora projektów.

Należy wybrać z menu *File* > *New Project Wizard*. Uzyskane okno dialogowe w przyszłości możemy pominąć, zaznaczając opcję: *Don't show me this introduction again.* Po wybraniu opcji *Next* uzyskuje się okno umożliwiające definiowanie katalogu i nazwy dla nowego projektu (rys. 1.1). Należy wpisać katalog roboczy zgodnie z zaleceniami prowadzącego zajęcia laboratoryjne.



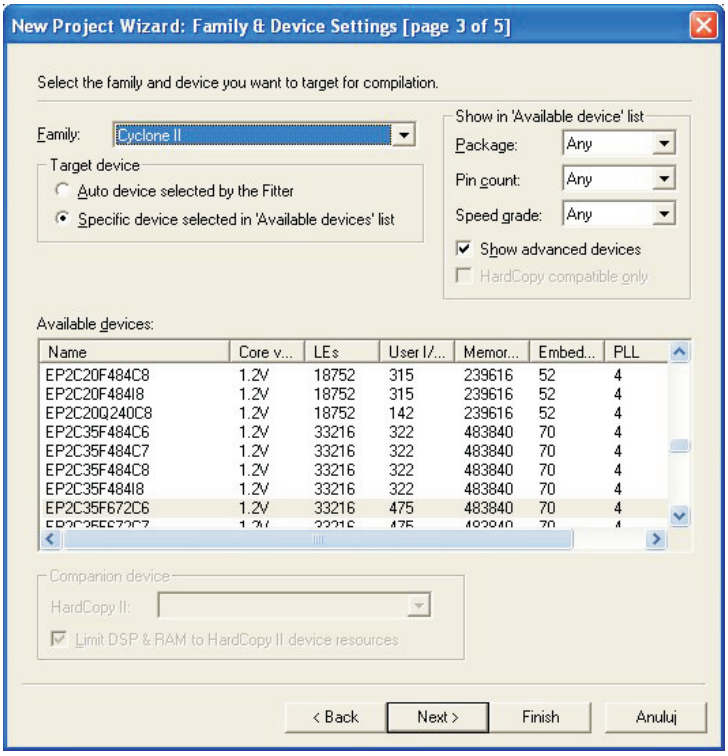
Rys. 1.1. Wprowadzanie katalogu i nazwy nowego projektu

Kolejne okno dialogowe (rys. 1.2) umożliwia dodanie do tworzonego projektu istniejących plików. Dla tworzonego projektu nie dodajemy gotowych plików źródłowych wybierając *Next*.



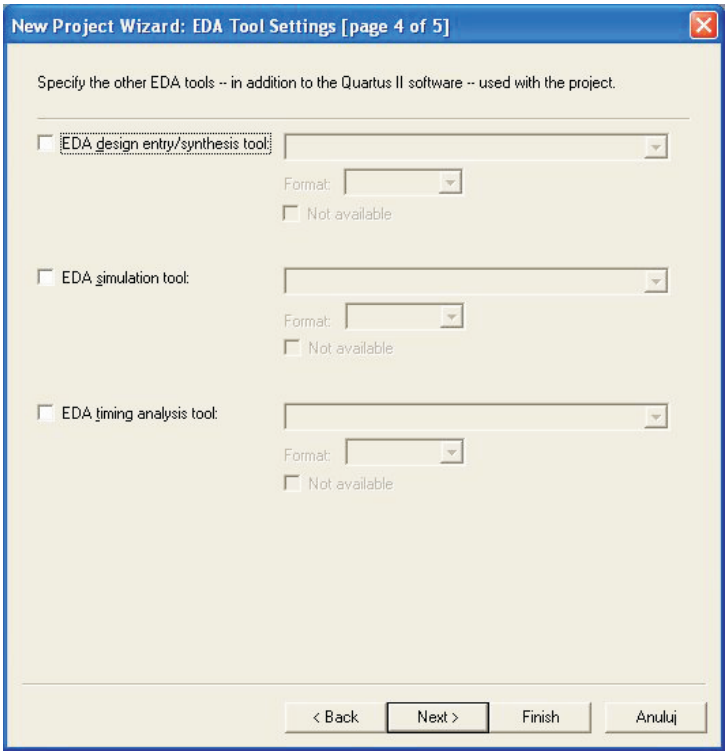
Rys. 1.2. Dodawanie istniejących plików do projektu

Okno dialogowe (rys. 1.3) umożliwia wybór rodziny układów programowalnych, a następnie konkretnego układu. Należy wybierać rodzinę *Cyclone II* i układ *EP2C35F672C6.*



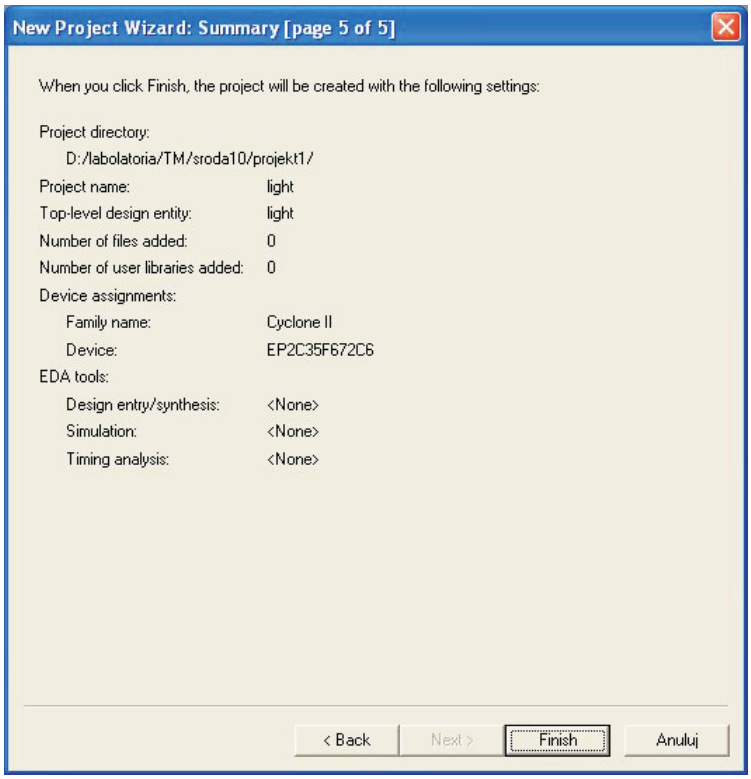
Rys. 1.3. Wybór programowalnego układu logicznego

W kolejnym oknie dialogowym (rys. 1.4), należy pominąć zaznaczenia pól wyboru i wybrać *Next*.



Rys. 1.4. Specyfikacja narzędzi firm innych niż zawartych w środowisku Quartus II

W ostatnim oknie dialogowym kreatora (rys. 1.5) przedstawiono wybrane ustawienia projektu. Należy wybrać opcję *Finish*.



Rys. 1.5. Podsumowanie ustawień tworzonego projektu

1.6. Przykład schematu realizowanego w edytorze graficznym

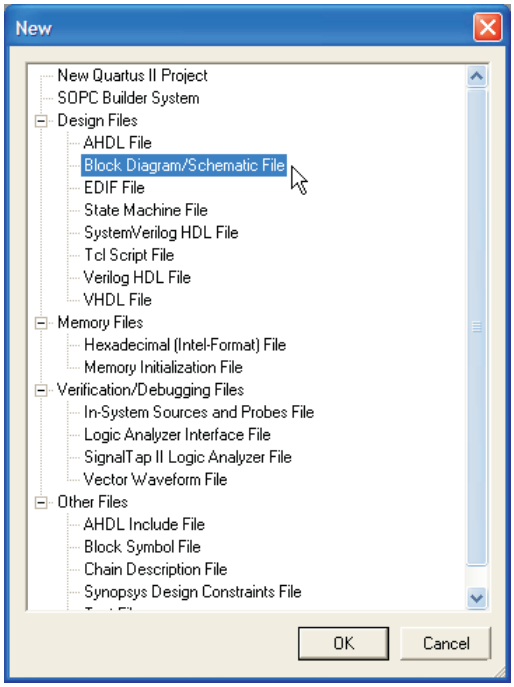
Realizacja pierwszego projektu utrwala znajomość zasad działania podstawowych bramek logicznych. Poniżej przedstawiono tablice prawdy dla poszczególnych bramek (tabela 1.).

Tabela 1.1

Tablice prawd dla operacji logicznych

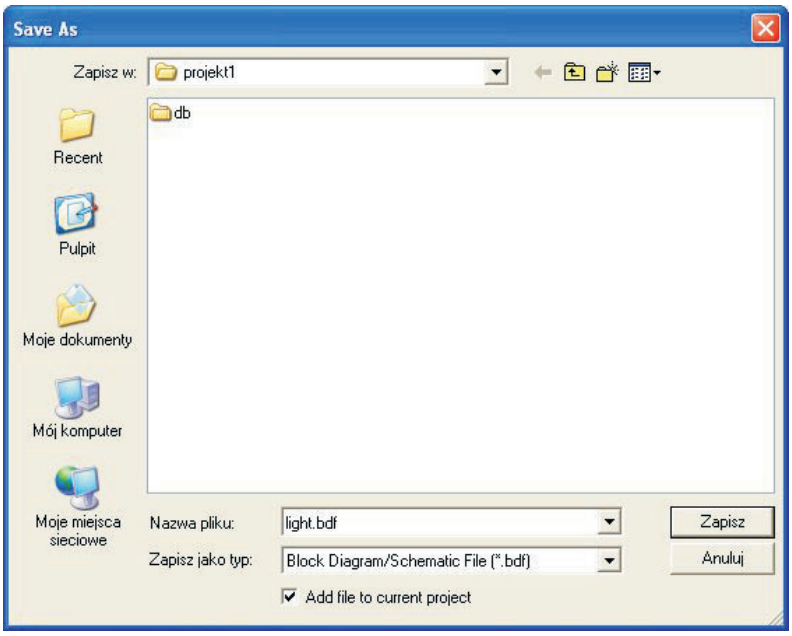
p	q	NOT p	NOT q	p AND q	p OR q	p NAND q	p NOR q	p XOR q
0	0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0	1
1	0	0	1	0	1	1	0	1
1	1	0	0	1	1	0	0	0

Tworzenia schematu badanego układu należy zacząć od wybrania menu: *File > New*, a następnie, korzystając z okna dialogowego (rys. 1.6), należy wybrać *Block Diagram /Schematic File*.



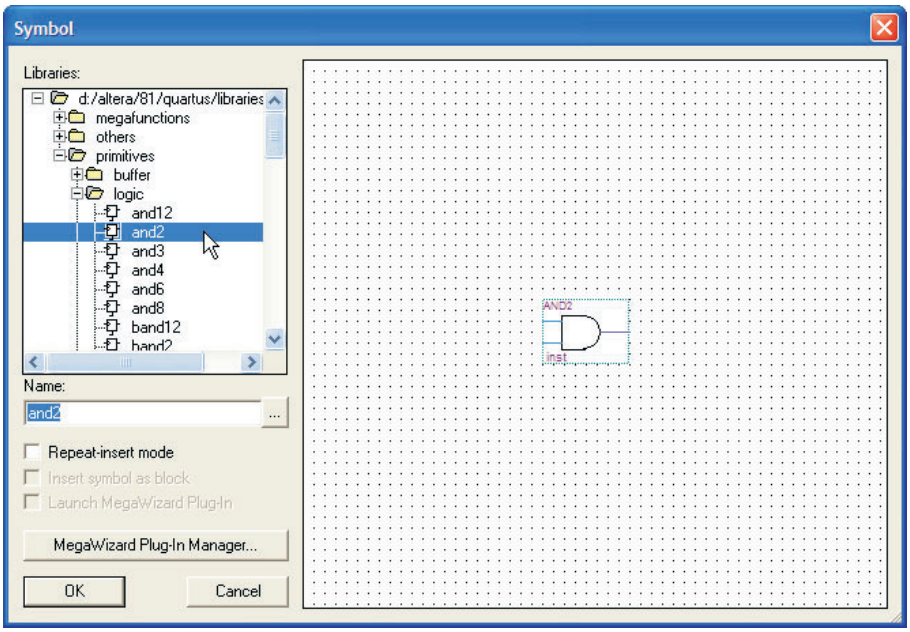
Rys. 1.6. Wybór sposobu opisu układu cyfrowego

Schemat należy zapisać, wybierając komendą *File* > *Save As*, pod nazwą „light”, w katalogu podanym przez prowadzącego (rys. 1.7). Po zapisaniu pliku należy dodać go do projektu, wybierając *Project* > *Add current file to project.*



Rys. 1.7. Zapis tworzonego schematu

W celu wprowadzenia symboli elementów tworzonego schematu klikamy dwukrotnie lewym przyciskiem myszy pole edytora i wybieramy żądany blok funkcjonalny (rys. 1.8).

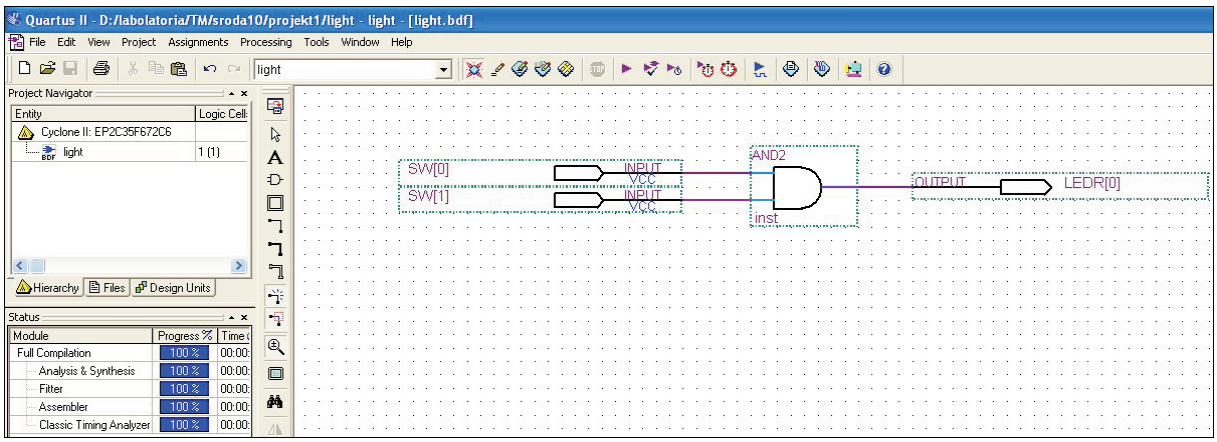


Rys. 1.8. Biblioteki bloków funkcjonalnych

Pierwszym badanym komponentem jest bramka AND (*and2*), można ją znaleźć w bibliotece *primitives/logic*. W polu edytora graficznego należy ponadto umieścić dwa wejścia (*input*) oraz jedno wyjście (*output*) z biblioteki *primitives/pin*.

Następnie należy wykonać połączenia pomiędzy pinami a badaną bramką. Wejściom nadajemy nazwy SW[0] i SW[1] a wyjściu LEDR[0]. Nazwy wprowadzamy poprzez dwukrotne kliknięcie lewym przyciskiem myszy na wejścia i wyjście.

Stworzony projekt należy skompilować, wybierając ikonę:  . Po kompilacji wyświetla się okno dialogowe z podsumowaniem. Po zapoznaniu się z nim należy je zamknąć. Po poprawnym zrealizowaniu opisanych powyżej czynności uzyskany wygląd okna edytora schematów powinien być zbliżony do przedstawionego narys. 1.9.

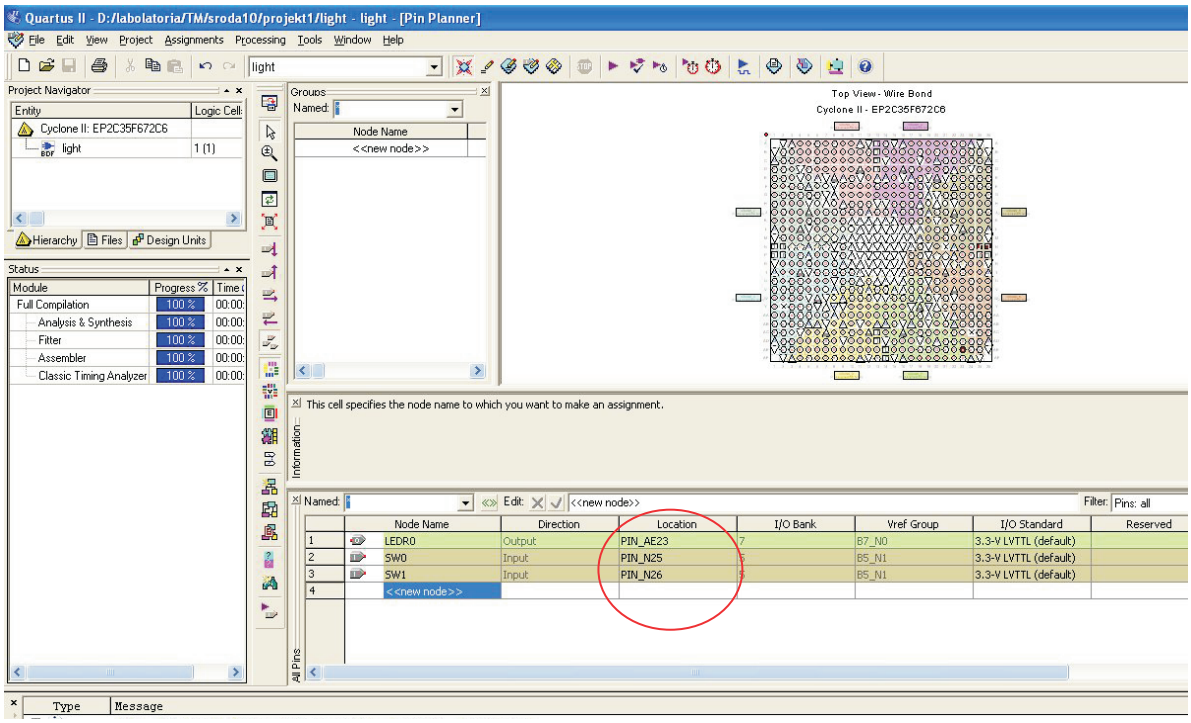


Rys. 1.9. Edytor schematów

Sugeruje się pominięcie przyporządkowania poszczególnych wyprowadzeń układu FPGA do portów wejścia/wyjścia, a jedynie zapoznanie się z nim. Korzystając z informacji zawartych w dokumentacji [1], można przyporządkować wejściom SW[0] (PIN_N25) i SW[1] (PIN_N26) oraz wyjściu LEDR[0] (PIN_AE23) fizyczne wyprowadzenia układu *EP2C35F672C6*. W tym celu należy wybrać *Assignments* > *Pins*, następnie w oknie dialogo-

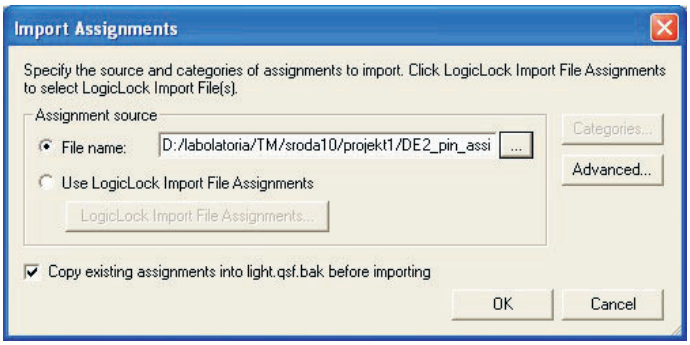
wym (rys. 1.10), w kolumnie *Location* wskazać odpowiadające danemu wejściu/wyjściu wyprowadzenie.

UWAGA: Błędne zdefiniowanie wyprowadzenia, jako wyjściowego, może spowodować po uruchomieniu programu uszkodzenie zestawu laboratoryjnego DE2. Należy zwrócić uwagę, iż kompilator nie zgłosi żadnego ostrzeżenia użytkownikowi o źle ustawionej kierunkowości wyprowadzenia.



Rys. 1.10. Przyporządkowanie wyprowadzeń

Alternatywnym podejściem do opisywanego powyżej jest zaimportowanie gotowego przyporządkowania wyprowadzeń z pliku: *DE2_pin_assignments.csv*. W pliku tym zdefiniowano wszystkie wyprowadzenia układu *EP2C35F672C6*, przy czym nadano im nazwy zgodne z opisami wejść/wyjść zamieszczonymi na pakiecie DE2 oraz w dokumentacji [1]. Należy skopiować wymieniony plik z katalogu wskazanego przez prowadzącego do katalogu roboczego, a następnie wybieramy z menu *Assignments* > *Import Assignments*. Należy wybrać plik z katalogu roboczego (rys. 1.11) i potwierdzić, wybierając *OK*.

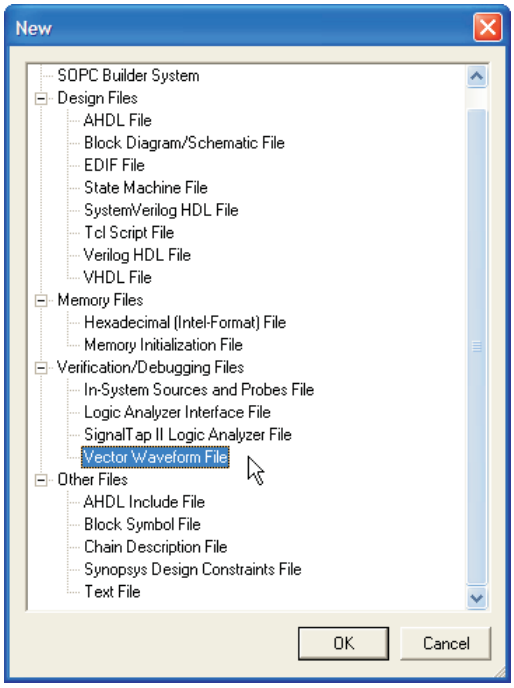


Rys. 1.11. Importowanie przyporządkowania pinów

Ponownie należy wykonać kompilację stworzonego projektu, wybierając ikonę:  .

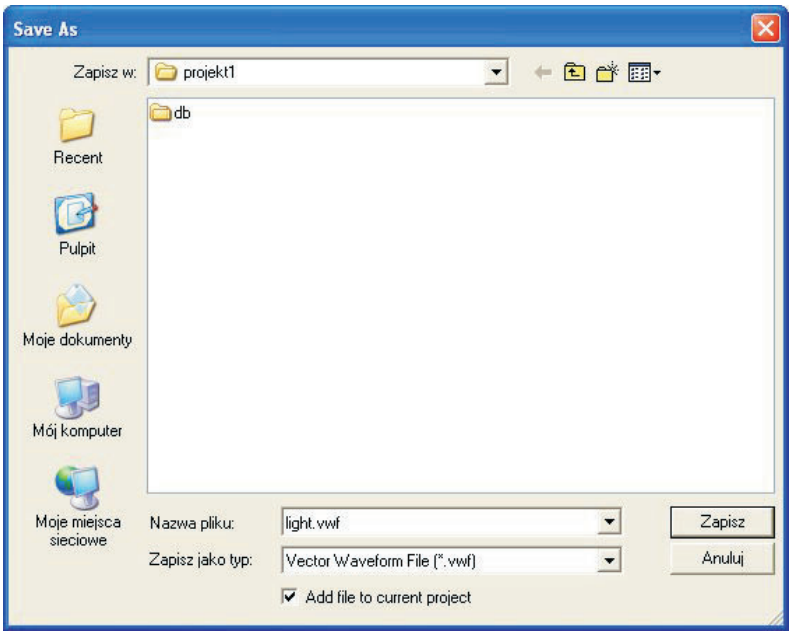
1.7. Symulacja projektu

Należy wybierając z menu *File* > *New*, po czym ukaże się okno dialogowe jak na rys. 1.12, w którym należy wybrać *Vector Waveform File*.



Rys. 1.12. Otwieranie nowego pliku *Vector Waveform File*

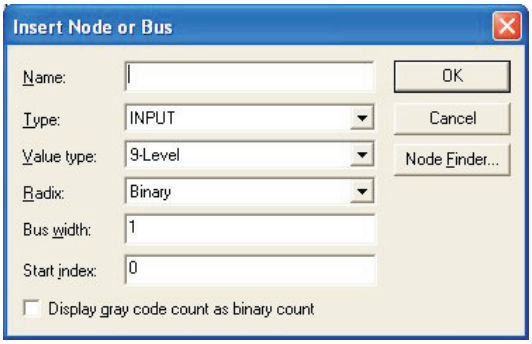
Należy wybrać z menu *File* > *Save As* i zapisać plik symulacyjny pod nazwą „Light.vwf” (rys. 1.13).



Rys. 1.13. Zapisywanie pliku *Vector Waveform File*

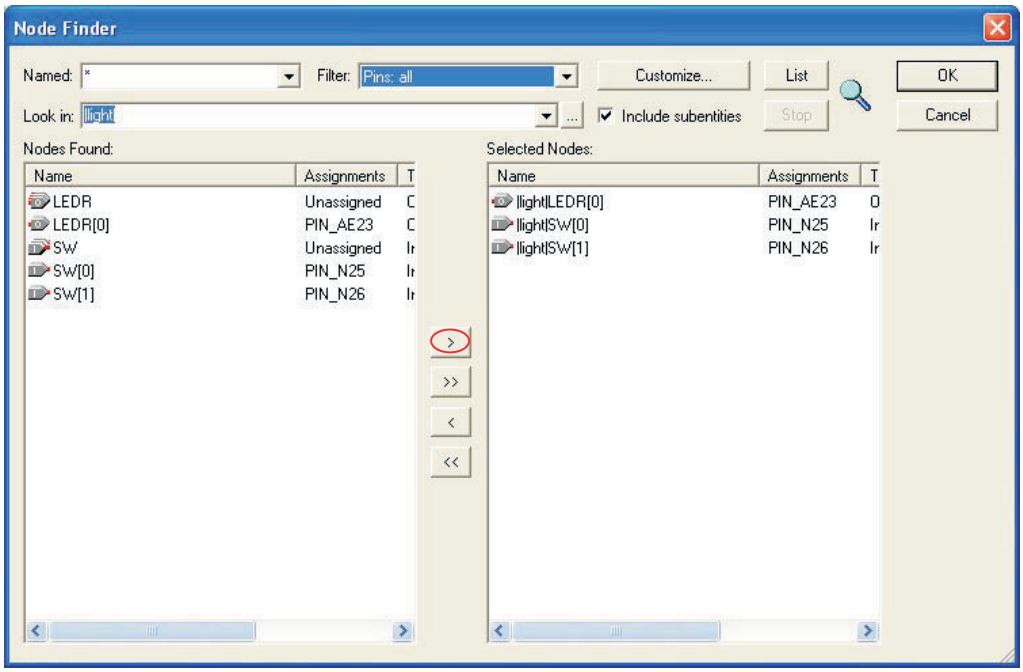
Należy zmienić końcowy czas symulacji, wybierając z menu *Edit* > *End Time* i wpisując w wywołanym oknie 200 ns.Chcąc obserwować przebieg symulacji dla ustawionego okresu symulacji, należy wybrać z menu *View* > *Fit in Window*.

Następnie należy wprowadzić porty wejściowe i wyjściowe. Należy kliknąć dwukrotnie lewym przyciskiem myszy dla kursor ustawiony w kolumnie *Name* (lewa strona edytora), uzyskując okno *Insert Node or Bus* (rys. 1.14).



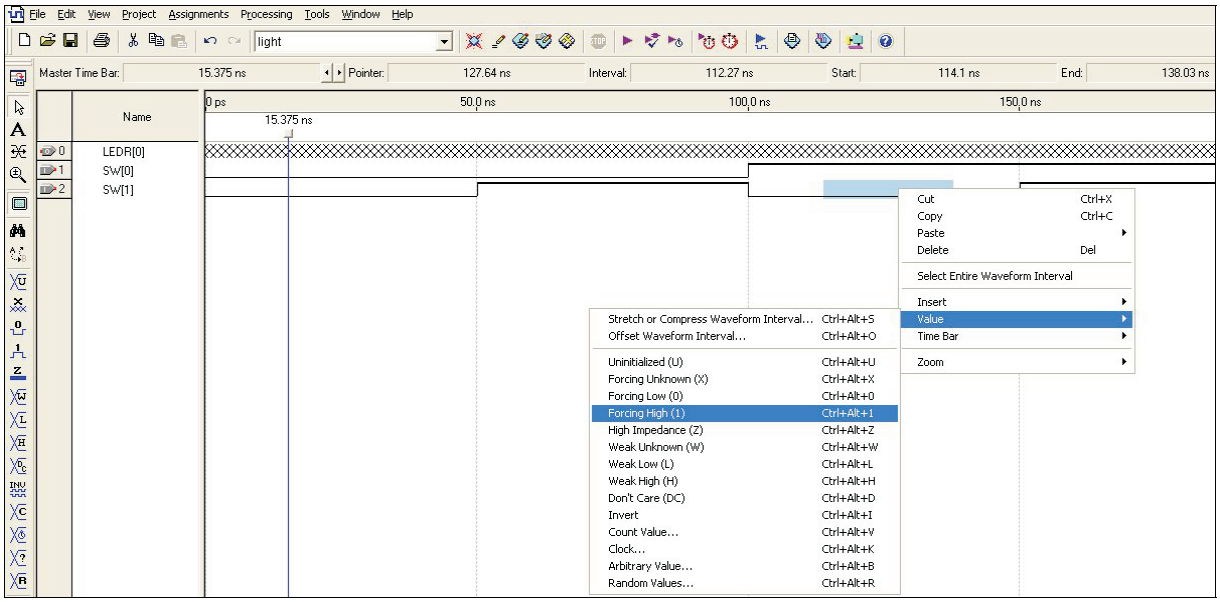
Rys. 1.14. Wprowadzanie portów wejściowych i wyjściowych

Należy wybrać *Node Finder* i i z rozwijanej listy *Filter* wybierać *Pins*: *all*. Należy kliknąć lewym przyciskiem myszy na przycisk *List* i wybierać z listy port wyjściowy LEDR[0] oraz porty wejściowe SW[0] i SW[1], a następnie wybrać przycisk > (rys. 1.15).



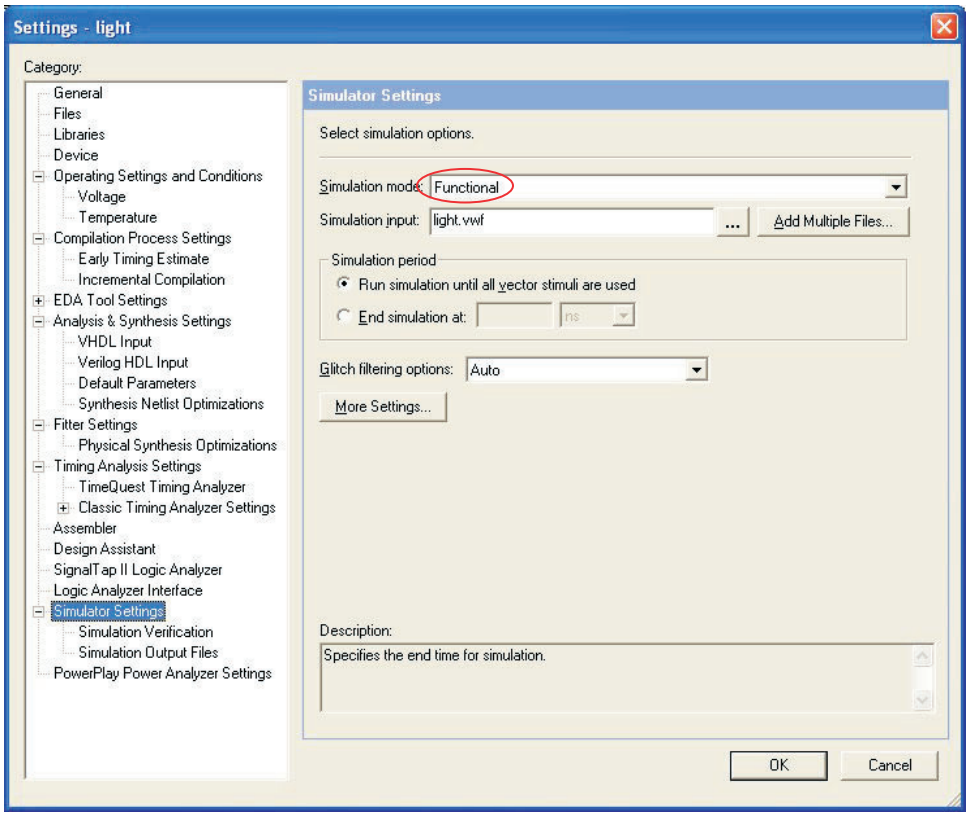
Rys. 1.15. Wybór portów wejściowych i wyjściowych

Siatkę w podglądzie symulacji należy ustawić na 50 ns, wybierając *Edit* > *Grid Size*. Należy zdefiniować stany portów wejściowych, zaznaczając myszką wymagany przedział czasu, a następnie klikając prawym przyciskiem myszy i z menu *Value* wybierając żądany stan/typ sygnału. W celu np. ustawienia stanu wysokiego należy wybrać *Value* > *Forcing High* (*1*) (rys. 1.16) lub wybrać ikonę 1. Należy ustawić stan wysoki dla przedziału czasu od 100 ns do 200 ns dla portu SW[0] oraz dla przedziałów od 50 ns do 100 ns i od 150 ns do 200 ns dla portu SW[1].



Rys. 1.16. Ustawianie stanów portów wejściowych

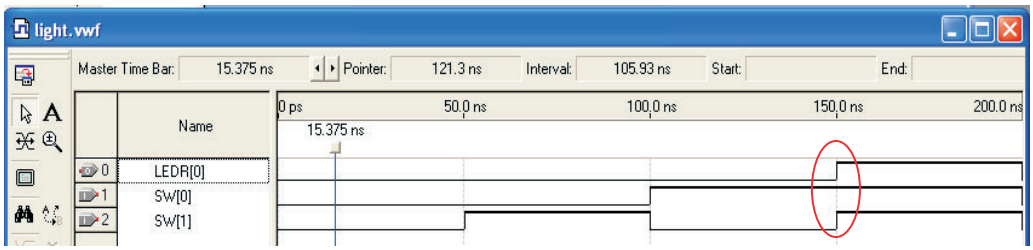
W środowisku Quartus możemy wykonać dwa typy symulacji:
Funkcjonalną – *functional simulation*, która nie uwzględnia opóźnień propagacyjnych. Typ symulacji mażna ustawić, wybierając z menu *Assignments* > *Settings*, a następnie dla *Simulator Setings* w polu *Simulation mode* w omawianym przypadku, ustawiając *Functional* (rys. 1.17).
Czasową – *timing simulation*, która uwzględnia opóźnienia propagacyjne występujące w układzie. W tym przypadku w polu *Simulation mode* należy ustawić *timing* (rys. 1.17).



Rys. 1.17. Ustawianie typu symulacji

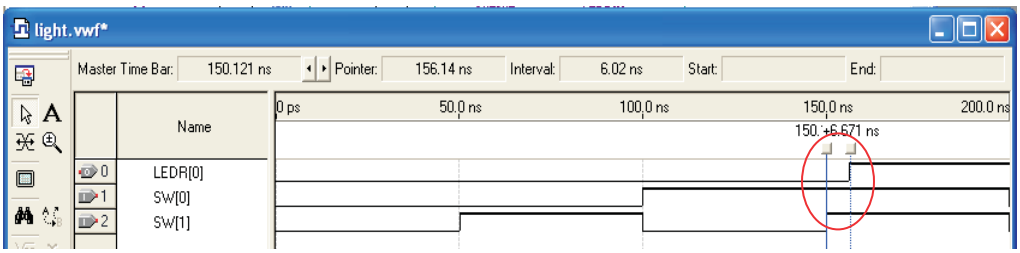
Przed wykonaniem symulacji funkcjonalnej nalezy wygenerować listę etykiet, wybierając z menu *Processing* > *Generate Functional Simulation Netlist*. Należy wykonać symulację, wybierając z menu *Processing* > *Start Simulation* lub ikonę .

Po zrealizowaniu symulacji funkcjonalnej należy przystąpić do analizy przebiegu sygnału wyjściowego LEDR[0] (rys. 1.18). Zmiana stanu na wyjściu następuje bez żadnego opóźnienia w stosunku do chwili, w której wejście SW[1] zmienia po raz drugi stan ze stanu niskiego na stan wysoki (rys. 1.18).



Rys. 1.18. Wynik symulacji funkcjonalnej (ang. *functional simulation*)

Po zrealizowaniu kolejnej symulacji (czasowej) można zauważyć (rys. 1.19) opóźnienie propagacyjne, które występuje w rzeczywistym układzie. Zmiana stanu na wyjściu LEDR[0] następuje po ok. 6,6 ns w stosunku do chwili, w której wejście SW[1] zmienia po raz drugi stan ze stanu niskiego na stan wysoki.

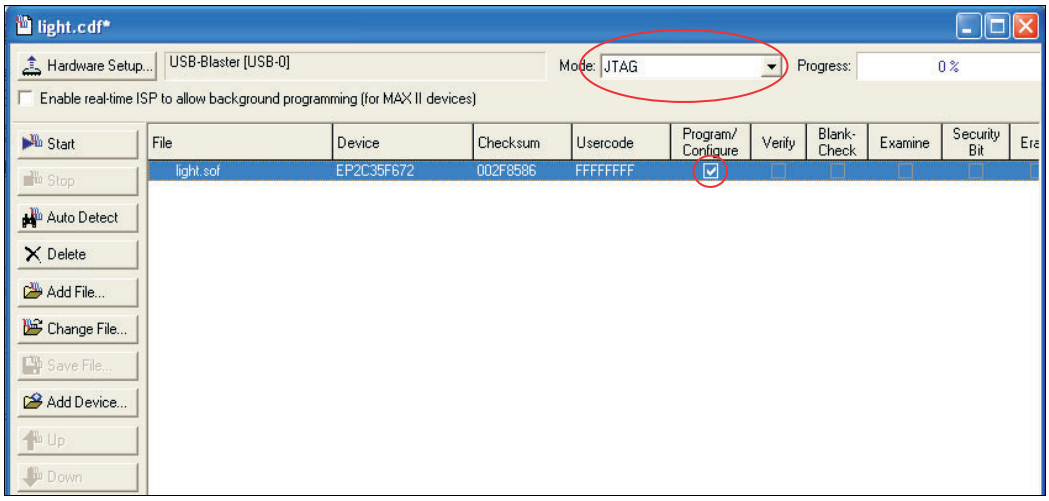


Rys. 1.19. Wynik symulacji czasowej (ang. *timing simulation*)

1.8. Konfiguracja układu FPGA i pamięci flash

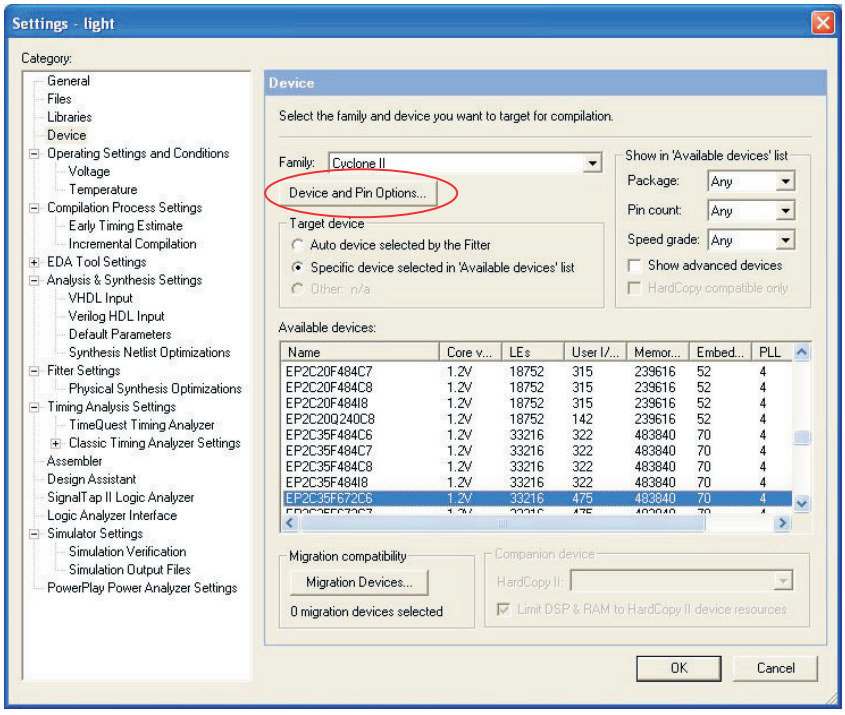
Aby umożliwić konfigurację układu FPGA lub pamięci flash znajdujących się na pakiecie DE2 należy zainstalować sterownik *USB-Blaster*. Procedura instalowania wymienionego sterownika została opisana w dokumentacji [2]. Konfiguracja może być realizowana w dwu trybach *JTAG* i *Active Serial Programming*.

W przypadku wykorzystania trybu *JTAG* kod wynikowy kompilacji przesyłany jest bezpośrednio do pamięci SRAM układu FPGA – *EP2C35F672C6*. Aby to umożliwić należy przełącznik znajdujący się obok wyświetlacza alfanumerycznego ustawić w pozycję „RUN”.
Wymieniony tryb programowania będzie wykorzystywany głównie na zajęciach laboratoryjnych. Zaletą tego rozwiązania jest możliwość przeprowadzenia nieskończonej liczby poprawnych przeprogramowań pamięci SRAM, natomiast wadą utrata programu po wyłączeniu zasilania. Moduł programatora umożliwiający konfigurację należy wywołać, wybierając z menu *Tools* > *Programmer* lub ikonę . Następnie należy ustawić tryb *JTAG* w polu *Mode* (rys. 1.20). W polu *File* możemy zauważyć plik o nazwie „light.sof” oraz w polu *Device* znajdujący się na płycie DE2 układ FPGA – *EP2C35F672C6* (rys. 1.20). W przypadku braku na liście wymienionego pliku należy go dodać, wybierając *Add File*. W polu dialogowym przedstawionym na rys. 1.20 należy zaznaczyć opcję w polu *Program/Configure*. Konfigurację układu *EP2C35F672C6* rozpoczynamy, wybierając przycisk *Start* (rys. 1.20).

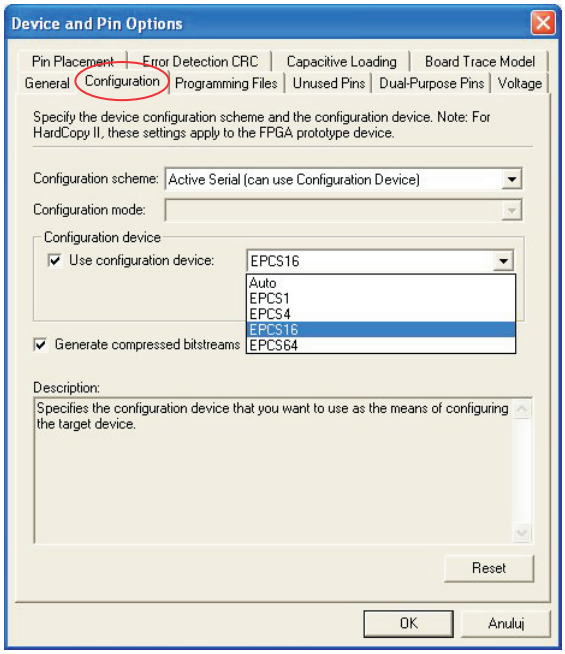


Rys. 1.20. Okno dialogowe modułu programatora (ang. *Programmer*)

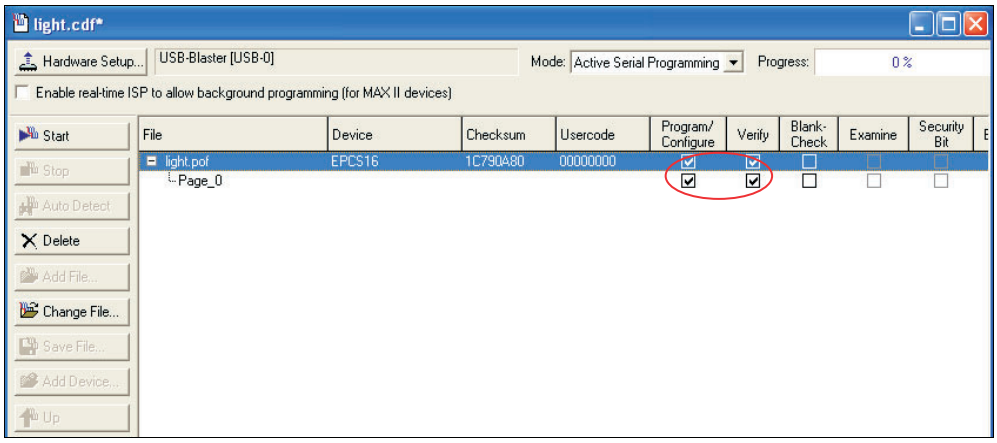
Zaleca się pominięcie programowania pamięci flash na zajęciach laboratoryjnych. W celu konfiguracji pamięci flash – *EPCS16* należy przełącznik znajdujący się obok wyświetlacza alfanumerycznego ustawić w pozycję „PROG”. Konfiguracyjną pamięć typu flash można wybrać z menu *Assignments* > *Device*, a następnie w uzyskanym oknie dialogowym (rys. 1.21) należy wybrać przycisk *Device and Pin Options*. Należy wybrać zakładkę *Configuration* (rys. 1.22) i wskazać zastosowaną pamięć *EPCS16*. Należy uruchomić moduł programatora (*Programmer*) i zmienić w polu *Mode* tryb *JTAG* na *Active Serial Programming* (rys. 1.23). Należy wprowadzić plik wynikowy „Light.pof”, wybierając *Add File*. Następnie należy zaznaczyć (rys. 1.23) opcje zawarte w polach *Program/Configure* i *Verify*. Nastepnie należy wybierać przycisk *Start* w celu przeprogramowania pamięci konfiguracyjnej flash (rys. 1.23). Po prawidłowym przeprogramowaniu wyłączamy zasilanie DE2 zmieniamy pozycję przełącznika z „PROG” na „RUN” i ponownie testujemy zaprogramowany projekt. Wadą tego rozwiązania jest ograniczona zdolności przeprogramowania, typowo 100 000 razy.



Rys. 1.21. Przyporządkowanie do projektu programowalnego układu logicznego



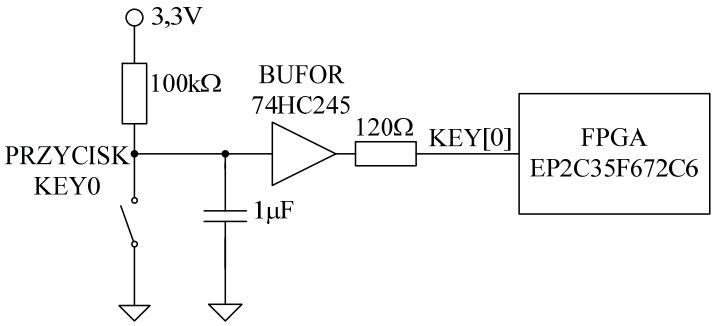
Rys. 1.22. Specyfikacja pamięci konfiguracyjnej *EPCS16*



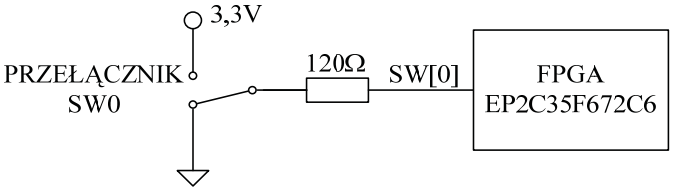
Rys. 1.23. Okno dialogowe modułu programatora *Programmer*

1.9. Testowanie działania projektu z wykorzystaniem pakietu DE2

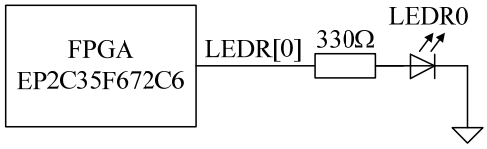
Po przeprowadzeniu konfiguracji należy przystąpić do fizycznego testowania przygotowanego projektu „Light”. W celu poprawnego zrozumienia działania badanej bramki należy zapoznać się ze schematami zawartymi narys. 1.24, rys. 1.25 i rys. 1.26 lub w dokumentacji [1].
Przyciski KEY[0] do KEY[3] (styki normalnie otwarte) są szeregowo połączone z rezystorami o wartości 100 kΩ (definiującymi domyślny stan logiczny i wartość prądu płynącego przez zamknięty styk przycisku). Wciskając przycisk, podajemy na odpowiednie wejście bufora 74HC245 stan niski (0V). Układ 74HC245 jest nieodwracającym, buforem cyfrowym. Wyjścia buforu są podłączone (poprzez rezystory o wartości 120 Ω) do wyprowadzeń układu FPGA. W przypadku przełączników od SW[0] do SW[17] dla pozycji suwaka skierowanej w kierunku użytkownika wprowadzany jest stan niski (0 V) natomiast dla położenia suwaka bliżej diod LED wprowadzany jest stan wysoki (3,3 V). Diody LED (od LEDR[0] do LEDR[17] i od LEDG[0] do LEDG[7]) świecą w przypadku wystawienia przez układ FPGA na odpowiednim wyjściu stanu wysokiego (3,3 V).



Rys. 1.24. Schemat podłączenia przycisku (ang. *push button*): KEY0 do układu programowalnego FPGA w pakiecie DE2



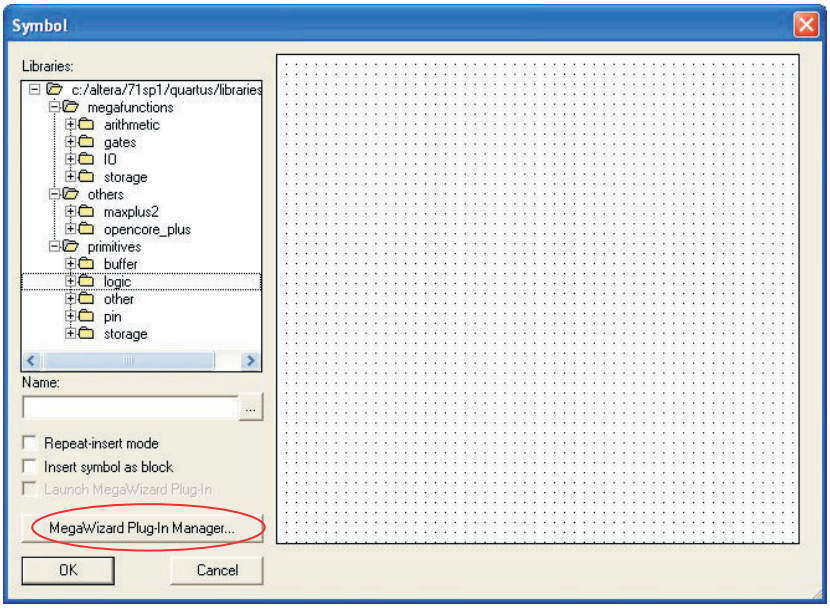
Rys. 1.25. Schemat podłączenia przełącznika (ang. *toggle switch*) SW0 do układu programowalnego FPGA w pakiecie DE2



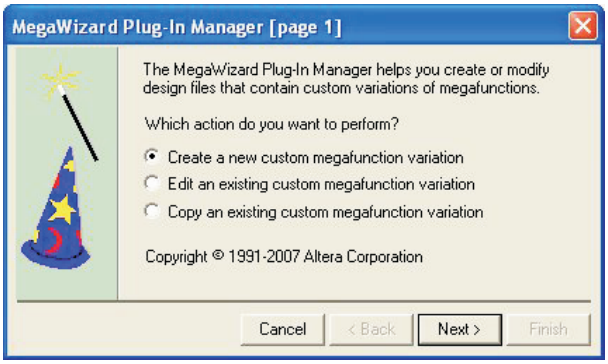
Rys. 1.26. Schemat podłączenia diody LEDR0 do układu programowalnego FPGA w pakiecie DE2

1.10. Zastosowanie narzędzia testowego *In-System Sources and Probes*

Kolejnym blokiem funkcjonalnym, jaki należy zastosować w projekcie „Light" jest *In-System Sources and Probes*. Narzędzie to ułatwia użytkownikowi uruchamianie projektów. Pozwala ono monitorować oraz wymuszać stany logiczne sygnałów (zarówno wejściowych, wyjściowych jak i wewnątrz projektu). Należy wybrać w menu *Edit* > *Insert Symbol*, a następnie uaktywnić kreator *MegaWizard Plug-In Manager* (rys. 1.27). Należy wybrać *Create a New custom megafunction variation* (rys. 1.28).

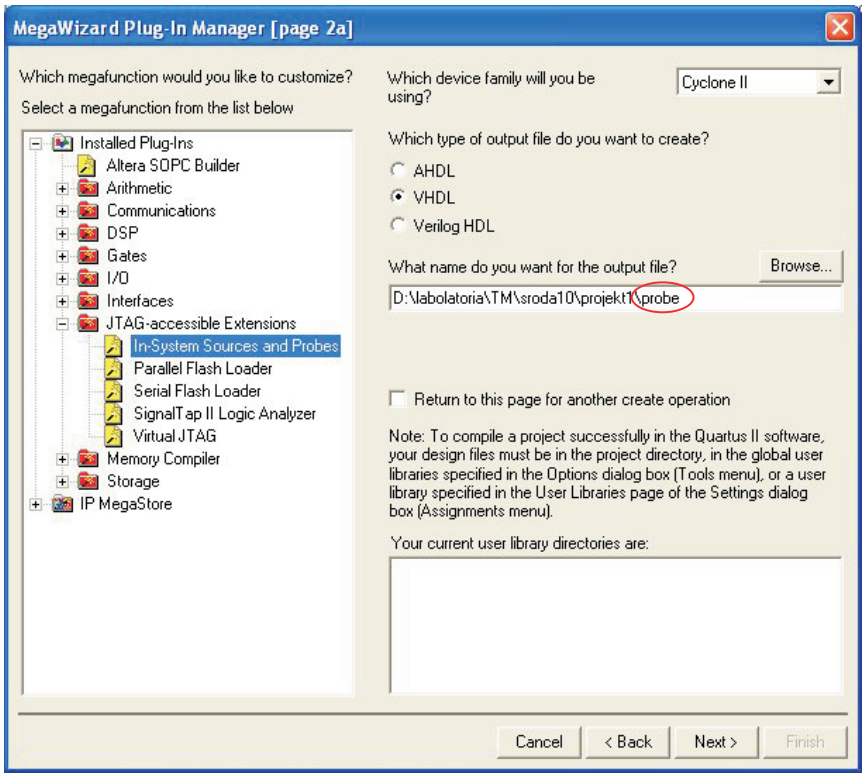


Rys. 1.27. Biblioteki bloków funkcjonalnych



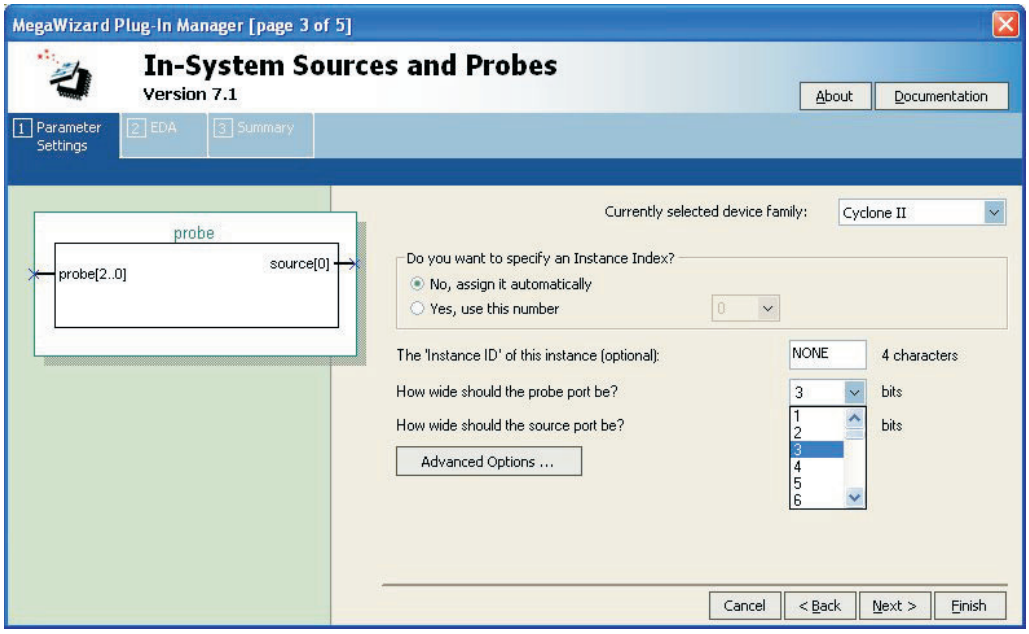
Rys. 1.28. Okno dialogowe *MegaWizard Plug-In Manager*

W kolejnym oknie dialogowym należy wybierać *JTAG-accessible Extensions* > *In-System Sources and Probes* oraz nadać nazwę dla wprowadzanego bloku („probe” –rys. 1.29).



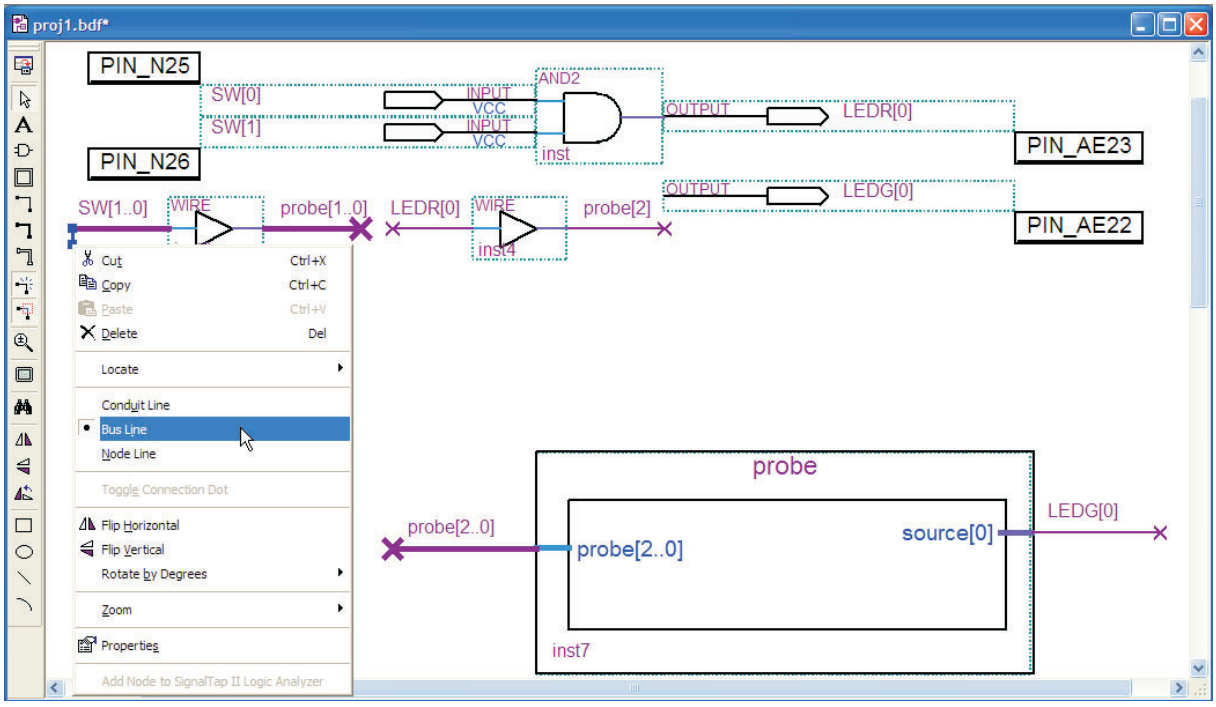
Rys. 1.29. Wybór z biblioteki bloku *In-System Sources and Probes*

W uzyskanym oknie dialogowym przedstawionym na rys. 1.30 należy wybrać trzybitowe wejście *probe* i jednobitowe wyjście *source*.



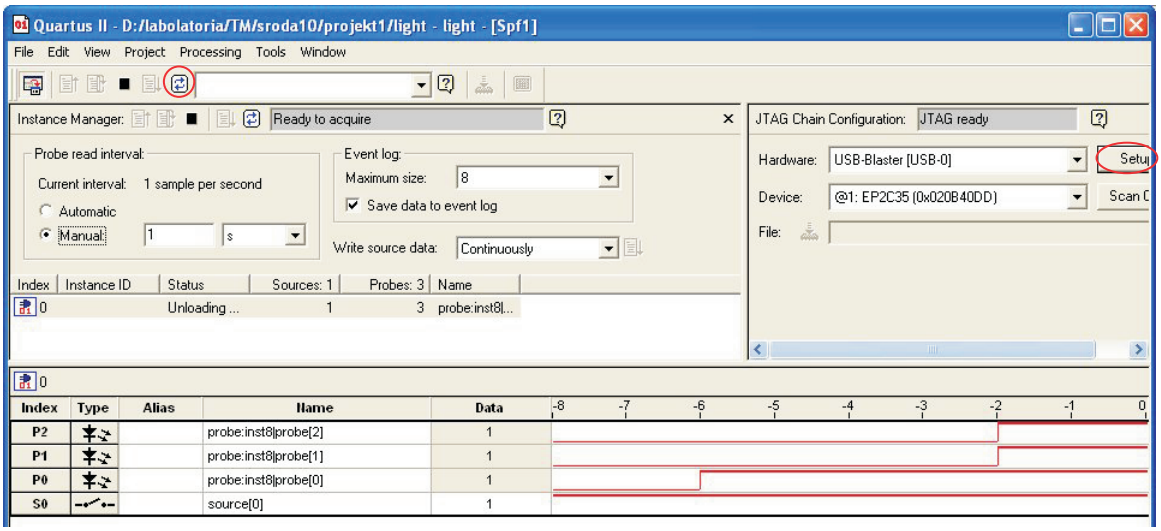
Rys. 1.30. Definiowanie parametrów bloku *In-System Sources and Probes*

Ostatecznie należy umieścić utworzony blok „probe” na schemacie zgodnie z rys. 1.31. Należy wykonać niezbędne połączenia, ustawiając ich typ na *Node Line* lub *Bus Line* odpowiednio w przypadku połączenia jednobitowego i wielobitowego. W etykiecie połączenia wielobitowego (magistrali) należy wyszczególnić w nawiasach kwadratowych numer najbardziej znaczącego bitu (ang. *MSB*) oraz numer najmniej znaczącego bitu (ang. *LSB*). Bity te należy oddzielić dwiema kropkami np. SW[1…0]. Następnie należy wykonać kompilację projektu i zaprogramować pamięć SRAM układu *EP2C35F672C6*.



Rys. 1.31. Schemat projektu „Light” po wprowadzeniu narzędzia *In-System Sources and Probes*

Należy wybrać z menu *Tools* > *In-System Sources and Probes Editor*. Należy wybrać przycisk *Setup* i następnie wskazać *USB-Blaster*. Należy ustawić parametry transmisji np. *Manual 1s* i wybrać ikonę *Continuously Read Probe Data* (rys. 1.32). Omawiane narzędzie testowe może się okazać szczególnie użyteczne w przypadku uruchamiania bardziej złożonych projektów.



Rys. 1.32. Okno dialogowe *In-System Sources and Probes Editor*

1.11. Przebieg ćwiczenia

W ramach ćwiczenia laboratoryjnego należy wykonać zadania zawarte powyżej z uwzględnieniem wskazówek prowadzącego, **poza programowaniem pamięci flash *EPCS16* i ręcznym przypisaniem wyprowadzeń do portów wejściowych i wyjściowych**.

1.12. Pytania kontrolne

Wymienić sposoby opisu układu cyfrowego w systemie projektowym Quartus.
Przedstawić tablice prawd dla bramek NOT, AND, OR, NAND, NOR, XOR.
Jaki stan logiczny pojawi się na odpowiednim wejściu układu FPGA po wciśnięciu jednego z przycisków KEY0 ÷ KEY4? Odpowiedź uzasadnić na podstawie schematu przedstawionego na rys. 1.24.
Jaki stan wyjścia układu FPGA spowoduje świecenie diody LED? Na podstawie schematu przedstawionego na rys. 1.26 oszacować prąd diod LEDR/LEDG.
Dlaczego system projektowy Quartus nie sygnalizuje błędnego przyporządkowania wyprowadzeń dla pakietu DE2? Wyjaśnić, jakie mogą być negatywne skutki tych błędów.
Omówić różnice zachodzące między bezpośrednią konfiguracją układu FPGA, a programowaniem pamięci flash.

1.13. Literatura

[1] DE2 Development and Education Board User Manual, ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf
[2] Getting Started with Altera's DE2 Board ftp://ftp.altera.com/up/pub/Tutorials/DE2/Digital_Logic/tut_initialDE2.pdf
[3] Quartus II Introduction Using Schematic Design (tut_quartus_intro_schem.pdf)
Szczepankowski P.: Wprowadzenie do systemu projektowego MAX+plus II (cw1i2.pdf)

2. Bramki i przerzutniki

2.1. Cel ćwiczenia

Głównym celem ćwiczenia jest poznanie własności podstawowych układów logicznych: bramek i przerzutników.

2.2. Sygnały i układy logiczne

Układ logiczny można przedstawić, jako „czarną skrzynkę" ze zdefiniowaną liczą wejść i wyjść. Sygnały na wejściach i wyjściach są dwuwartościowe (inaczej: binarne, dwójkowe), mogą przyjmować jedynie dwie wartości: zero (0) lub jeden (1). Kombinację wartości sygnałów na wejściach układu nazywa się wektorem (słowem, stanem) wejściowym, lub wzbudzeniem układu, natomiast kombinację wartości sygnałów wyjściowych wektorem (słowem, stanem) wyjściowym lub odpowiedzią układu. Działanie układu opisane jest zależnością między zbiorem wektorów wejściowych i wyjściowych. Można wyróżnić dwie grupy układów:

— układy kombinacyjne (bez pamięci), dla których stan dowolnego wyjścia w danej chwili czasowej zależy wyłącznie od aktualnej kombinacji stanów na wejściach,
— układy sekwencyjne (z pamięcią), dla których wartość przynajmniej jednego wyjścia w danej chwili zależy od stanu wejść w tej chwili oraz od poprzednich kombinacji wejściowych.

Odpowiedni poziom napięcia reprezentuje logiczną jedynkę i logiczne zero. Jeśli napięcie wysokie (np. 3.3V) wykorzystano do reprezentowania stanu wysokiego (H – *High*) a napięcie niskie (0V) do reprezentowania stanu niskiego (L – *Low*) to przyjęto konwencję logiki dodatniej. Natomiast stosując wyższe napięcie do reprezentacji logicznego zera i napięcie niższe do reprezentacji logicznej jedynki, przyjmuje się konwencję logiki ujemnej.

2.3. Algebra Bool'a

Algebra Boole'a oparta jest na trzech podstawowych operacjach na argumentach binarnych:

— jednoargumentowa operacja negacji, oznaczana kreską nad zmienną logiczną (np. $\overline{X_1}$), operacja negacji zmienia wartość argumentu na przeciwny;
— n-argumentowa operacja sumy logicznej, oznaczana „+", która jest zdefiniowana następująco: jeżeli jeden z argumentów jest równy jeden to wynik operacji jest równy jeden;
— n-argumentowa operacja iloczynu logicznego, oznaczana „·", zdefiniowana następująco: jeżeli wszystkie argumenty są równe jeden, to wynik operacji jest równy jeden;

Działania sumy i iloczynu mają następujące własności [2]:

1.	przemienność	$A+B=B+A$	$A\cdot B=B\cdot A$
2.	łączność	$(A+B)+C=A+(B+C)$	$(A\cdot B)\cdot C=A\cdot(B\cdot C)$
3.	rozdzielczość	$A+(B\cdot C)=(A+B)\cdot(A+C)$	$A\cdot(B+C)=A\cdot B+A\cdot C$
4.	tożsamość	$A+0=A$	$A\cdot 0=0$
		$A+1=1$	$A\cdot 1=A$
		$A+A=A$	$A\cdot A=A$
5.	komplementarność	$A+\overline{A}=1$	$A\cdot\overline{A}=0$

Działania sumy i iloczynu spełniają też następujące prawa:

1. prawo de Morgana $\overline{A+B}=\overline{A}\cdot\overline{B}$ $\overline{A\cdot B}=\overline{A}+\overline{B}$
2. prawo sklejania $A\cdot\overline{B}+A\cdot B=A$ $(A+\overline{B})\cdot(A+B)=A$
3. prawo pochłaniania $A\cdot\overline{B}+B=A+B$

Operacje sumy, iloczynu i negacji przyporządkowują argumentom operacji wartości, czyli określają funkcje pomiędzy argumentami wejściowymi a wyjściem.
Podstawowe funkcje logiczne są następujące:

— negacja (NOT, NIE) $Y=\overline{X_1}$,
— suma (OR, LUB) $Y=X_1+X_2$,
— iloczyn (AND, I) $Y=X_1\cdot X_2$,
— zanegowany iloczyn (NAND, NIE-I) $Y=\overline{X_1\cdot X_2}$,
— zanegowana suma (NOR, NIE-LUB) $Y=\overline{X_1+X_2}$,
— nierównoważność (EX-OR, ALBO) $Y=\overline{X_1}\cdot X_2+X_1\cdot\overline{X_2}=X_1\oplus X_2$
— równoważność (EX-NOR) $Y=\overline{X_1}\cdot\overline{X_2}+X_1\cdot X_2=\overline{X_1\oplus X_2}$

2.4. Bramki

Układ elektroniczny realizujący funkcję logiczną nazywa się bramką (*gate*) lub funktorem.
Jednowejściowa bramka NOT (negator, inwerter) realizuje negację: stan na wyjściu (Y) jest przeciwny niż stan na wejściu bramki (X_1), czyli $Y=\overline{X_1}$.

Bramka AND realizuje iloczyn logiczny: na wyjściu bramki jest stan wysoki tylko wtedy, gdy na wszystkie wejścia podany jest stan wysoki. Dwuwejściowa bramka AND jest opisana funkcją $Y=X_1\cdot X_2$.

Bramka OR realizuje sumę logiczną: na wyjściu bramki jest stan wysoki wtedy, gdy przynajmniej jedno wejście jest w stanie wysokim – dwuwejściowa bramka OR jest opisana funkcją $Y=X_1+X_2$.

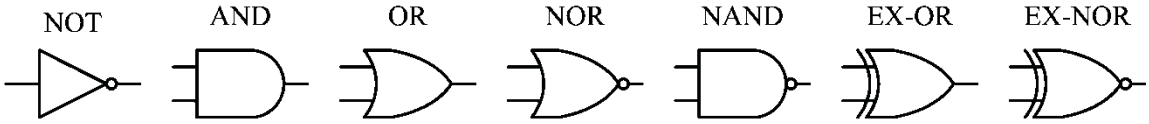
Na podstawie praw de Morgana:
$X_1+X_2=\overline{\overline{X_1}\cdot\overline{X_2}}$ oraz $X_1\cdot X_2=\overline{\overline{X_1}+\overline{X_2}}$.

Operacja sumy może zostać zastąpiona poprzez operacje negacji i iloczynu, a operacja iloczynu przez operacje negacji i sumy. Połączone operacje NOT i AND realizowane są za pomocą bramki NAND natomiast połączone operacje NOT i OR za pomocą bramki NOR.
Na wyjściu bramki NAND jest stan niski tylko wtedy, gdy wszystkie wejścia są w stanie wysokim. Dwuwejściowa bramka NAND jest opisana funkcją $Y=\overline{X_1\cdot X_2}$.

Stan wyjścia bramki NOR jest niski wtedy, gdy przynajmniej jedno wejście jest w stanie wysokim. Dwuwejściowa bramka NOR realizuje funkcje $Y=\overline{X_1+X_2}$.

Używane są jeszcze dwuwejściowe bramki EX-OR (*exclusive*-OR – wyłącznie-OR, nierówno ważność, XOR) i EX-NOR (*exclusive*-NOR, równoważność, XNOR). Wykorzystywane są najczęściej do sprawdzenia czy w sygnałach występują jednocześnie takie same stany logiczne. Stan wyjścia bramki EX-OR jest wysoki tylko wtedy, gdy stan wejść jest różny, natomiast na wyjściu bramki EX-NOR jest stan wysoki tylko wtedy gdy stan wejść jest jednakowy. Operacje realizowane przez bramki opisuje się funkcjami: $Y=X_1\oplus X_2$ dla bramki EX-OR oraz $Y=\overline{X_1\oplus X_2}$ dla bramki EX-NOR.

Na rys. 2.1 przedstawiono symbole opisanych wcześniej bramek, a w tab. 2.1 zamieszczono wyniki operacji logicznych realizowanych przez poszczególne bramki dla wszystkich możliwych wektorów wejściowych, tzw. tablice prawdy (*truth table*).



Rys. 2.1. Najczęściej stosowane symbole bramek

Tabela 2.1

Tablice prawdy bramek

Wejścia		NOT	AND	OR	NAND	NOR	EX-OR	EX-NOR
X_1	X_2	$\overline{X_1}$	$X_1 \cdot X_2$	$X_1 + X_2$	$\overline{X_1 \cdot X_2}$	$\overline{X_1 + X_2}$	$X_1 \oplus X_2$	$\overline{X_1 \oplus X_2}$
0	0	1	0	0	1	1	0	1
0	1	1	0	1	1	0	1	0
1	0	0	0	1	1	0	1	0
1	1	0	1	1	0	0	0	1

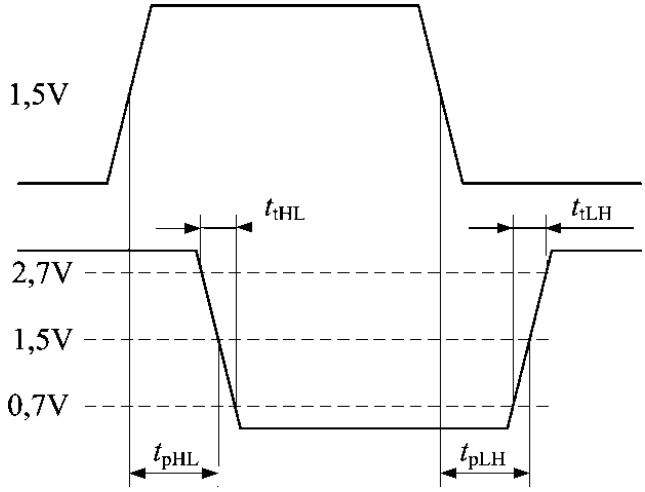
Bramki są dostępne w postaci półprzewodnikowych układów scalonych, które wykonywane są w kilku skalach integracji i różnych technologiach. Obecnie najbardziej rozpowszechnionymi rodzinami układów cyfrowych są TTL (*transistor-transistor logic*) oraz CMOS (*complementary MOS*). Poniżej przedstawiono podstawowe parametry statyczne (określające własności w stanie ustalonym) i dynamiczne (określające własności w stanach przejściowych) bramek wykonanych w technologii TTL.

Tabela 2.2

Parametry statyczne i dynamiczne bramek

Parametry statyczne	
U_{cc}	napięcie zasilania
U_{IH}	napięcie wejściowe w stanie 1
U_{IL}	napięcie wejściowe w stanie 0
U_{OH}	napięcie wyjściowe w stanie 1
U_{OL}	napięcie wyjściowe w stanie 0
I_{IH}	prąd wejściowy w stanie 1
I_{IL}	prąd wejściowy w stanie 0
I_{OH}	prąd wyjściowy w stanie 1
I_{OL}	prąd wyjściowy w stanie 0
I_{OS}	prąd wyjściowy zwarcia
I_{CCL}	prąd zasilania w stanie 0 na wyjściu
I_{CCH}	prąd zasilania w stanie 1 na wyjściu
N	obciążalność bramki
P_S	straty mocy w bramce

Parametry dynamiczne	
t_{pLH}	czas propagacji do stanu 1 na wyjściu; jest to czas mierzony od chwili osiągnięcia przez zbocze przebiegu wejściowego wartości 1,5V do chwili, kiedy napięcie na wyjściu wzrośnie od poziomu L do wartości 1,5V (rys. 2)
t_{pHL}	czas propagacji do stanu 0 na wyjściu; jest to czas mierzony od chwili osiągnięcia przez zbocze przebiegu wejściowego wartości 1,5V do chwili, kiedy napięcie na wyjściu zmniejszy się od poziomu H do wartości 1,5V (rys. 2)
t_p	średni czas propagacji $t_p = \frac{t_{pLH} + t_{pHL}}{2}$
t_{tLH}	czas narastania impulsu wyjściowego
t_{tHL}	czas opadania impulsu wyjściowego



Rys. 2.2. Definicje parametrów dynamicznych bramki TTL

Należy zaznaczyć, że operacje logiczne wykonywane są również z wykorzystaniem innych układów, np. elektrycy bardzo często realizują podstawowe operacje logiczne z wykorzystaniem przekaźników i styczników.

2.5. Przerzutniki

Przerzutniki są elementarnymi układami logicznymi z pamięcią czyli fizycznymi reprezentantami elementarnych układów sekwencyjnych. Przy użyciu przerzutników i bramek można budować złożone układy sekwencyjne. Podobnie jak wszystkie układy logiczne z pamięcią przerzutniki można podzielić na dwie podstawowe grupy:

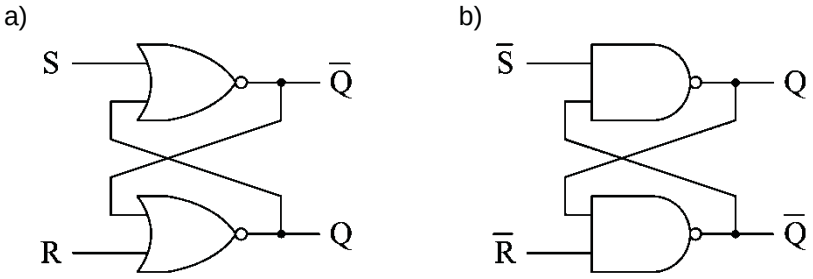
— przerzutniki asynchroniczne,
— przerzutniki synchroniczne.

Przerzutniki asynchroniczne mają tylko jeden typ wejść, tzw. wejścia informacyjne, podczas gdy przerzutniki synchroniczne mają dodatkowo tzw. wejścia zegarowe. Wejściami w sensie teorii automatów są tylko wejścia informacyjne, tzn. że tylko stan tych wejść wraz z aktualnym stanem wewnętrznym, określa następny stan przerzutnika.
W przerzutnikach asynchronicznych stan wewnętrzny zmienia się równocześnie (z dokładnością do czasu propagacji) ze stanem wejść informacyjnych. W przerzutnikach synchronicznych chwile zmian stanów wyznaczane są przez sygnał zegarowy na wejściu synchronizującym.

2.5.1. Przerzutniki asynchroniczne

Do najbardziej znanych przerzutników asynchronicznych należy przerzutnik RS. Schemat logiczny przerzutnika RS przedstawia rys. 2.3. Oznaczenia wejść pochodzą od angielskich słów SET (zapis) i RESET (zerowanie). Stan aktywny na wejściu SET przerzutnika ustawia wyjście przerzutnika w stan wysoki, natomiast stan aktywny na wejściu RESET przerzutnika zeruje przerzutnik, czyli wpisuje na wyjście Q stan niski. Dla przerzutnika zbudowanego z bramek NOR stanem (poziomem) aktywnym wejścia jest logiczna jedynka (stan wysoki), a poziomem spoczynkowym logiczne zero (stan niski). Dla przerzutnika zrealizowanego z bramek NAND poziomem aktywnym wejścia jest logiczne zero (stan niski), a poziomem spoczynkowym logiczna jedynka (stan wysoki). Wejście aktywne poziomem niskim przyjęto oznaczać kreską nad nazwą wejścia. Gdy oba wejścia są w stanie spoczynkowym, przerzutnik pamięta ostatnio wpisany stan wyjścia. W przypadku, gdy na obu wejściach przerzutnika pojawi się stan aktywny, na wyjściach bramek przerzutnika wystąpi ten sam stan logiczny. Stan ten uważany

jest za niedozwolony, gdyż stany wyjść przerzutnika nie są w tym przypadku przeciwne względem siebie.



Rys. 2.3. Schemat logiczny przerzutnika RS, realizacja przy użyciu bramek: a) NOR, b) NAND

Działanie przerzutników najczęściej definiowane jest tablicą przejść i wyjść, tablicą prawdy lub tablicą wymuszeń. Wymienione sposoby opisu zostaną wyjaśnione w sposób uproszczony na przykładzie przerzutnika RS. Wiersze tablicy przejść (tab. 2.3) odpowiadają stanom wewnętrznym przerzutnika (wartościom logicznym pamiętanym przez przerzutnik) a kolumny stanom wejściowym. Dla wszystkich przerzutników stan wyjścia (zero lub jeden) jest tożsamy ze stanem wewnętrznym, więc do jego opisu wystarcza podanie jedynie tablicy przejść. W tablicy przejść na przecięciu wiersza odpowiadającego danemu stanowi wewnętrznemu (a zarazem wyjściowemu) i kolumny odpowiadającej kombinacji sygnałów wejściowych jest podawany stan następny przerzutnika. Dalej rozważono tablicę przejść dla przerzutnika RS zbudowanego z bramek NOR (tab. 2.3a). Aktualny stan wewnętrzny przerzutnika, a zarazem stan jego wyjścia jest oznaczony literą Q i umieszczony w pierwszej kolumnie tablicy, natomiast wszystkie możliwe kombinacje wejść są umieszczone w pierwszym wierszu tablicy. W drugim wierszu tablicy pokazano, do jakich stanów wewnętrznych układ przechodzi ze stanu 0. Jeżeli wektor wejściowy jest 00 (czyli wejście S = 0 i R = 0), to przerzutnik przechodzi do stanu 0, jeśli wzbudzenie układu jest równe 01, to przerzutnik przechodzi do stanu 0, jeśli stan wejściowy jest równy 10, to przerzutnik przechodzi do stanu 1, słowo wejściowe 11 jest niedozwolone (po podaniu tej kombinacji stan przerzutnika nie jest zdefiniowany, może być dowolny i zależy od jego realizacji sprzętowej). W drugim wierszu tabeli jest opisane do jakich stanów przechodzi przerzutnik ze stanu 1.

Tabela 2.3

Tablica przejść przerzutnika RS złożonego z bramek: a) NOR, b) NAND

a)

Q/SR	00	01	11	10
0	0	0	×	1
1	1	0	×	1

b)

$Q/\overline{RS}$	00	01	11	10
0	×	1	0	0
1	×	1	1	0

gdzie: × – stan niedozwolony, Q_n i Q_{n+1} – aktualny i następny stan wyjścia przerzutnika

Tablicą prawdy przerzutnika nazywa się tablicowe przedstawienie funkcji logicznej, której wartość Q_{n+1} (stan następny przerzutnika) zależy od stanu wejść oraz od aktualnego stanu przerzutnika Q_n.

Tabela 2.4

Tablica prawdy przerzutnika RS złożonego z bramek: a) NOR, b) NAND

a)

S	R	Q_n	Q_{n+1}
□	0	0	0
0	0	1	1
	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	x
1	1	1	x

b)

$\overline{S}$	$\overline{R}$	Q_n	Q_{n+1}
0	0	0	x
0	0	1	x
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

gdzie: × – stan niedozwolony, Q_n i Q_{n+1} – aktualny i następny stan wyjścia przerzutnika

Tablica wymuszeń przerzutnika pokazuje, jakie wartości sygnałów wejściowych należy podać na wejścia przerzutnika, aby zmienił on stan na pożądany. Tablice wymuszeń wykorzystywane są podczas projektowania układów sekwencyjnych.

Tabela 2.5

Tablica wymuszeń przerzutnika RS złożonego z bramek: a) NOR, b) NAND

a)

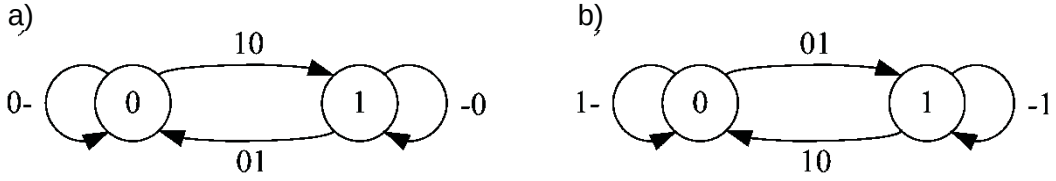
$Q_n \rightarrow Q_{n+1}$	SR
$0 \rightarrow 0$	0–
$0 \rightarrow 1$	10
$1 \rightarrow 0$	01
$1 \rightarrow 1$	–0

b)

$Q_n \rightarrow Q_{n+1}$	$\overline{SR}$
$0 \rightarrow 0$	1–
$0 \rightarrow 1$	01
$1 \rightarrow 0$	10
$1 \rightarrow 1$	–1

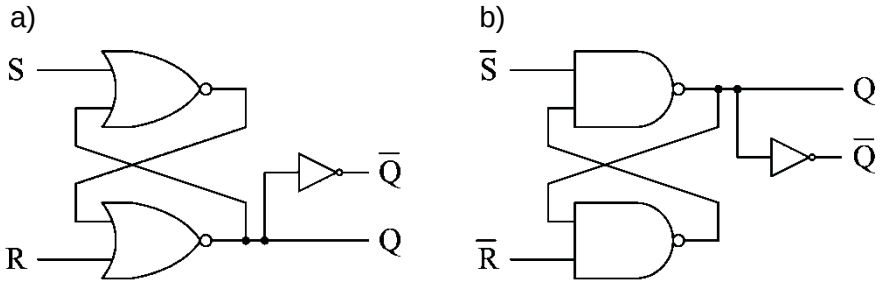
gdzie „–” oznacza stan dowolny wejścia

Działanie przerzutników (tak jak innych układów sekwencyjnych) można opisać także za pomocą grafów i przebiegów czasowych. Graf układu sekwencyjnego składa się z węzłów i strzałek. Węzłów jest tyle, ile jest stanów układu, a strzałki pomiędzy węzłami pokazują przejścia pomiędzy stanami. W przypadku przerzutników w węzłach podajemy stany wewnętrzne (stany wyjścia) przerzutnika, a obok strzałek stany wejściowe, odpowiadające danemu przejściu.



Rys. 2.4. Graf przerzutnika RS złożonego z bramek: a) NOR, b) NAND

Na rys. 2.4a przedstawiono graf przerzutnika RS złożonego z bramek NOR (rys. 2.3a), a na rys. 2.4b graf przerzutnika RS zbudowanego z bramek NAND (rys. 2.3b). Stany wejść podane są dla kombinacji SR ($\overline{SR}$).
Przerzutnik, w którym wyjścia pozostają komplementarne dla każdej kombinacji zmiennych wyjściowych, można zrealizować w sposób podany poniżej.

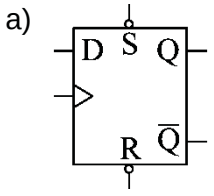


Rys. 2.5. Realizacja przerzutnika z dominacją a) zerowania, b) zapisywania

Jeśli sygnał S dominuje nad sygnałem R, mówi się o dominacji zapisywania, w przeciwnym przypadku – o dominacji zerowania.

2.5.2. Przerzutniki synchroniczne

W przerzutnikach synchronicznych można wyróżnić następujące wejścia i wyjścia:
wejścia informacyjne – np. D, T, J, K;
asynchroniczne wejścia: ustawiające (SET, S, PR) i zerujące (RESET, R, CLR);
wejście synchronizujące (inaczej: zegarowe, taktujące, wyzwalające) – oznaczane trójkątnym symbolem lub CLK, CL, CP, C, T; wyjścia: proste (Q) i zanegowane ($\overline{Q}$).
Asynchroniczne wejścia ustawiające i kasujące mają najwyższy priorytet. Używa się ich do ustawienia bądź wyzerowania wyjścia niezależnie (asynchronicznie) od innych sygnałów wejściowych. Wejścia te mogą być aktywne stanem wysokim lub niskim. Jeśli wejście jest aktywne stanem niskim, to jest zazwyczaj oznaczone znakiem negacji w postaci kółka umieszczonego na symbolu.
Na rys. 2.6 przedstawiono symbol przerzutnika D wyzwalanego zboczem oraz jego tablicę przejść. W momencie pojawienia się zbocza narastającego przebiegu zegarowego przerzutnik D zapamiętuje stan, jaki był na jego wejściu D.



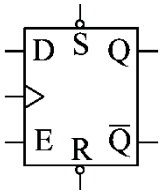
b)

Q\D	0	1
0	0	1
1	0	1

Rys. 2.6. Symbol przerzutnika D wyzwalanego zboczem oraz jego tablica przejść

Oprócz przerzutników typu D synchronizowanych dodatnim zboczem przebiegu zegarowego dostępne są również przerzutniki D wyzwalane wysokim poziomem przebiegu zegarowego (rys. 2.7). W przypadku przerzutnika wyzwalanego poziomem informacja z wejścia D jest przepisywana na wyjście przez cały czas trwania wysokiego poziomu na wejściu zegarowym (oznaczanym: E, G, CLK). Powrót przebiegu zegarowego do poziomu niskiego powoduje zapamiętanie („zatrzaśniecie", stąd też nazwa typu „zatrzask" – ang. *latch)* stanu wejścia D.

a)



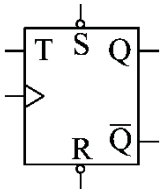
b)

Q\DE	00	01	11	10
0	0	0	1	0
1	1	0	1	1

Rys. 2.7. Symbol przerzutnika D wyzwalanego poziomem i tablica przejść

Kolejnym przerzutnikiem jednowejściowym jest przerzutnik T (rys. 2.8). Jeżeli na wejściu T jest stan wysoki, to przy każdym narastającym zboczu przebiegu zegarowego stan wyjścia przerzutnika zmienia się na przeciwny. Jeśli stan wejścia T zostanie zmieniony na niski, to przerzutnik pamięta stan wymuszony poprzednio.

a)



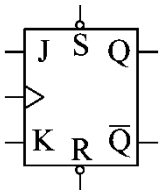
b)

Q\T	0	1
0	0	1
1	1	0

Rys. 2.8. Symbol przerzutnika T i tablica przejść

Przerzutnik JK (rys. 2.9) działa podobnie jak przerzutnik RS. Wejście J (nazwę wejścia można kojarzyć z ustawianiem **J**edynki) odpowiada wejściu S (SET), a wejście K (jak **K**asowanie) działa jak wejście R (RESET). Różnica w działaniu dotyczy stanu niedozwolonego dla przerzutnika RS, czyli wówczas, gdy R = S = 1. W przypadku przerzutnika JK dla stanu wejść J = K = 1 wyjście zmienia stan na przeciwny przy każdym narastającym zboczu przebiegu zegarowego. W tej sytuacji działanie przerzutnika JK jest identyczne do działania przerzutnika T dla T = 1.

a)



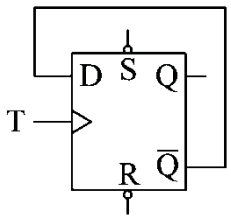
b)

Q\JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

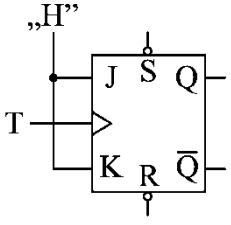
Rys. 2.9. Symbol przerzutnika JK i tablica przejść

Przerzutnik typu D oraz przerzutnik typu JK można wykorzystać do realizacji asynchronicznego przerzutnika typu T (rys. 2.10). W tym przypadku następuje zmiana funkcji wejścia zegarowego na wejście informacyjne T.

a)



b) „H”



Rys. 2.10. Realizacja przerzutnika T z przerzutnika a) D, b) JK

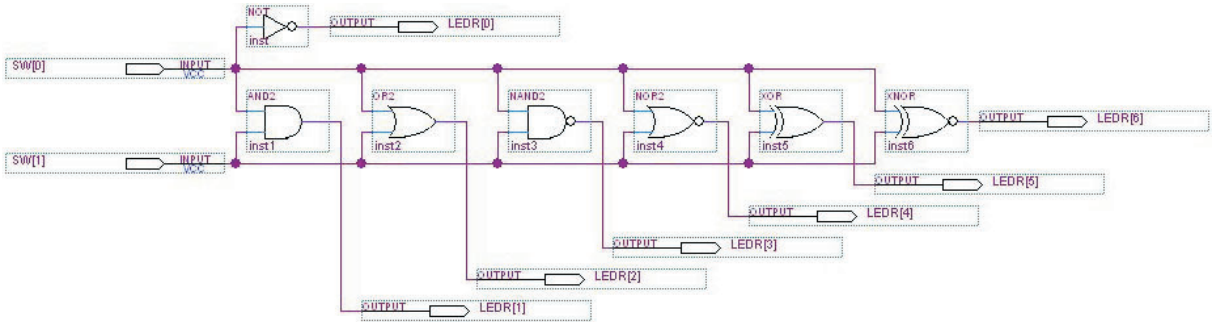
2.5.3. Parametry przerzutników

Parametry statyczne podane dla bramek TTL odnoszą się również do przerzutników. Natomiast podstawowe parametry dynamiczne podane dla przerzutników scalonych, to:

— czasy propagacji sygnałów od wejścia zegarowego do wyjść Q i $\overline{Q}$;
— czasy propagacji sygnałów od wejść asynchronicznych do wyjść Q i $\overline{Q}$;
— czas ustalania t_s (*t setup*) – minimalny czas, w którym sygnał wejściowy musi być obecny na wejściach informacyjnych synchronicznych przed nadejściem wyzwalającego zbocza impulsu zegarowego;
— czas przetrzymywania t_h (*t hold*) – minimalny czas, w którym sygnał wejściowy musi pozostawać na wejściu informacyjnym synchronicznym po wystąpieniu wyzwalającego zbocza impulsu zegarowego;
— maksymalna częstotliwość przebiegu zegarowego;
— maksymalny czas narastania (opadania) zbocza przebiegu zegarowego.

2.6. Przygotowanie do zajęć

1. Na stronie internetowej przedmiotu oraz na komputerach w laboratorium udostępniony jest projekt (bramki.zip) zawierający m.in. schemat przedstawiony na rys. 2.11 (bramki.bdf). Wykorzystując ten schemat i plik symulacyjny bramki.vwf (lub utworzone przez siebie), należy przeprowadzić symulacje działania bramek wymienionych w tab. 2.1 dla wszystkich możliwych kombinacji sygnałów wejściowych (wykonanie symulacji wyjaśniono w instrukcji do ćwiczenia 1) i przynieść na zajęcia wyniki symulacji (w postaci elektronicznej lub papierowej). Należy być przygotowanym na dyskusję otrzymanych wyników z prowadzącym zajęcia.



Rys. 2.11. Proponowany schemat do symulacyjnego sprawdzenia funkcji logicznych realizowanych przez bramki

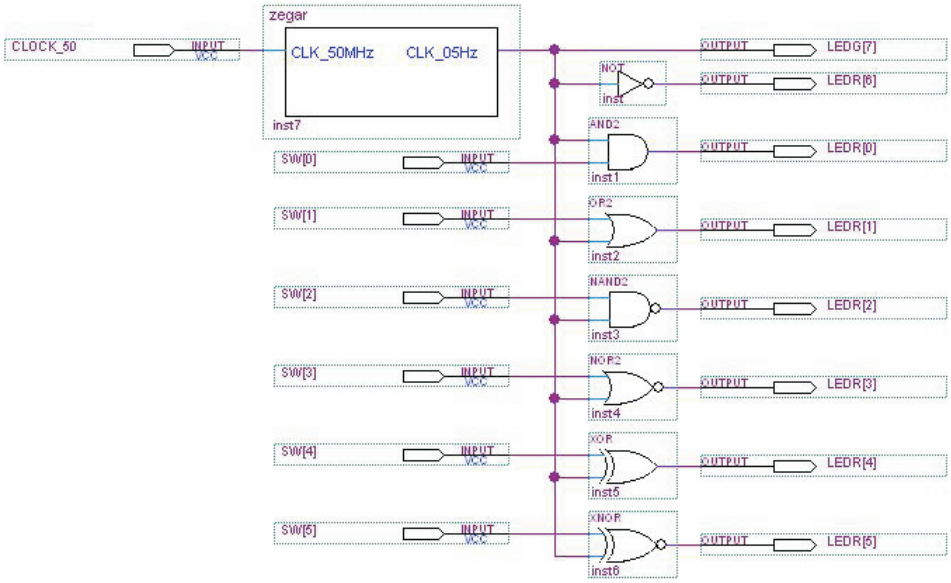
Uwaga: W projekcie znajduje się kilka schematów. Aby skompilować wybrany schemat (jest to konieczne do przeprowadzenia symulacji), należy w oknie *Project Navigator* wybrać zakładkę *Files*, aby wyświetlić wszystkie pliki znajdujące się w projekcie, następnie prawym przyciskiem myszy należy kliknąć na wybranym pliku (np. bramki.bdf) i wybrać opcję „*Set as Top-Level Entity*" – można wówczas skompilować wybrany schemat oraz przetestować go symulacyjnie w sposób opisany w instrukcji do ćwiczenia 0. Po utworzeniu pliku symulacyjnego należy sprawdzić w ustawieniach modułu Symulatora (menu główne „*Assigments*" –> „*Settings*" –> kategoria „*Simulator Settings*" –> pole „*Simulator input*"), czy właściwy plik został ustawiony jako wejściowy, jeśli nie – należy go zmienić.

2. Wykorzystać schemat przerzutniki.bdf oraz plik symulacyjny przerzutniki.vwf do wykonania symulacji działania przerzutników asynchronicznych RS, zrealizowanych na bram-

kach NOR i NAND oraz przerzutników synchronicznych: D (wyzwalanego zboczem i poziomem przebiegu zegarowego), T, JK, RS. Przynieść na zajęcia wyniki symulacji (w postaci elektronicznej lub papierowej). Należy być przygotowanym na dyskusję otrzymanych wyników z prowadzącym zajęcia.

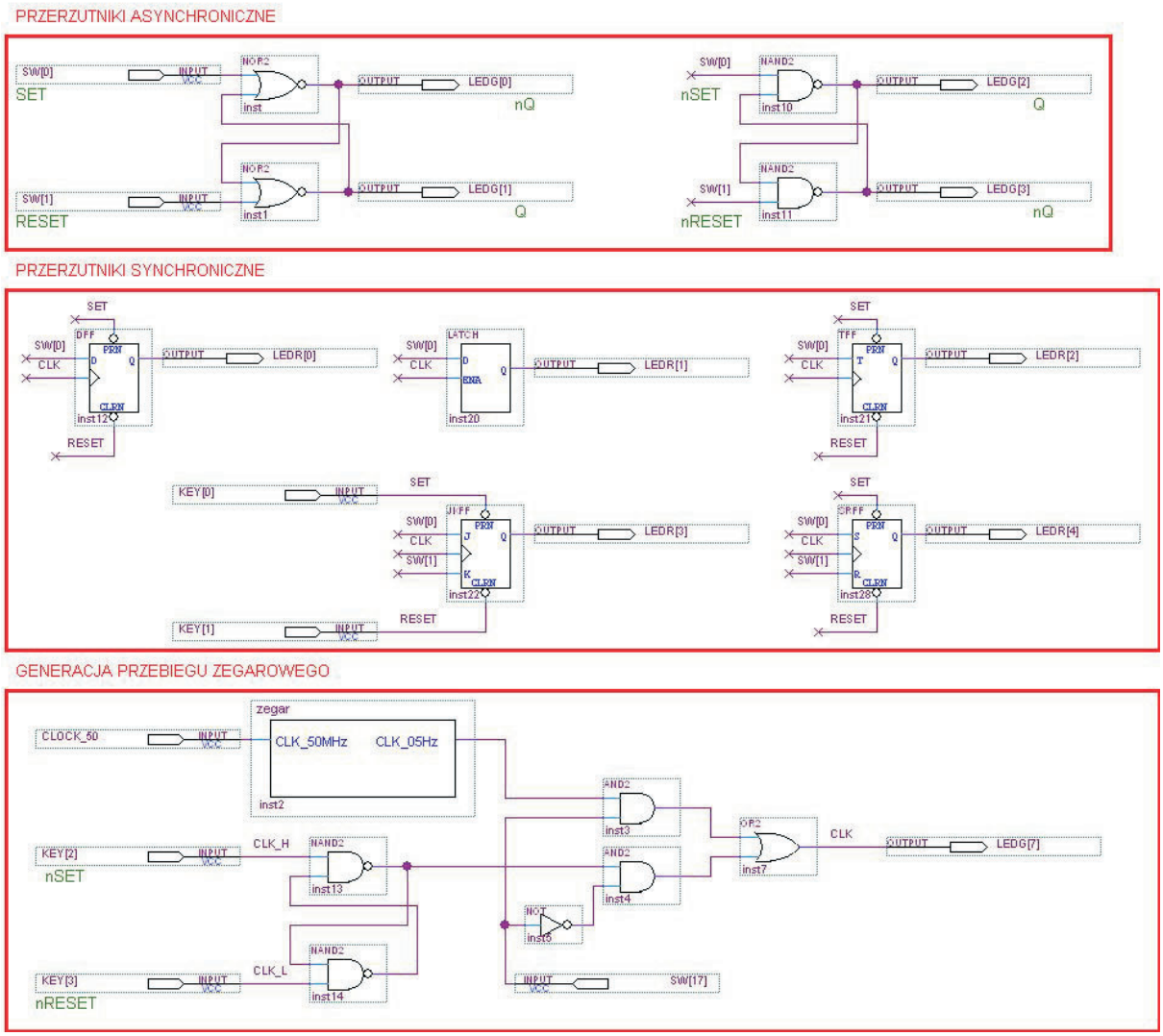
2.7. Przebieg ćwiczenia

1. Otworzyć projekt bramki.qpf, skompilować schemat bramki_clk.bdf (rys. 2.12) i skonfigurować układ programowalny. Następnie przystąpić do testowania działania bramek NOT, AND, NAND, OR, NOR, EXOR i EXNOR (rys. 2.12). Do jednego wejścia (każdej bramki) podłączony jest sygnał prostokątny o częstotliwości 0,5Hz i wypełnieniu 50%, a do drugiego (za wyjątkiem bramki NOT) wejścia, nazwanego sterującym, odpowiedni przełącznik (przełącznik steruje diodą, która jest bezpośrednio nad nim umieszczona). Zaobserwować i naszkicować kształt sygnałów na wyjściach bramek w przypadku wymuszenia stanu niskiego i wysokiego na wejściu sterującym.



Rys. 2.12. Proponowany schemat do sprawdzenia w układzie edukacyjnym funkcji logicznych realizowanych przez bramki

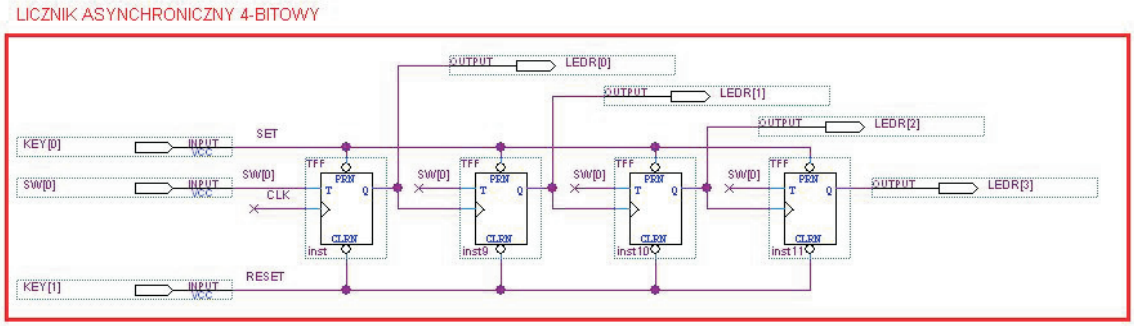
2. Skompilować schemat przerzutniki_clk.bdf (rys. 2.12) i skonfigurować układ programowalny. Przetestować działanie przerzutników asynchronicznych RS (zrealizowanych na bramkach NAND i NOR) oraz przerzutników synchronicznych D, *latch*, T, JK, RS (rys. 2.12), ustalić tablice prawd przerzutników, tablice wymuszeń i narysować grafy dla każdego przerzutnika. Tablice zamieszczone w załączniku (rozdział 2.11) należy wypełnić i pokazać prowadzącemu zajęcia.
   Przełącznik SW[17] umożliwia wybór przebiegu zegarowego, oznaczonego etykietą CLK. Dla wartości SW[17] równej 0 przebieg zegarowy będzie podawany z wyjścia przerzutnika asynchronicznego RS. Wówczas wciśnięcie przycisku KEY[2] spowoduje wymuszenie stanu wysokiego sygnału CLK, a wciśnięcie KEY[3] stanu niskiego. Co się stanie jeśli żaden z przycisków nie będzie wciśnięty i jeśli będą wciśnięte oba? Gdy przełącznikiem SW[17] zostanie wymuszony stan wysoki, wówczas przebieg zegarowy o częstotliwości 0,5Hz będzie podawany z wyjścia bloku „zegar”.



Rys. 2.13. Proponowany schemat do sprawdzenia działania przerzutników

Dla przerzutników synchronicznych D, T, JK i RS sprawdzić działanie asynchronicznych wejść ustawiających dla różnych stanów na wejściach informacyjnych i dla obu poziomów przebiegu zegarowego. Wyniki obserwacji zanotować i umieścić w sprawozdaniu.

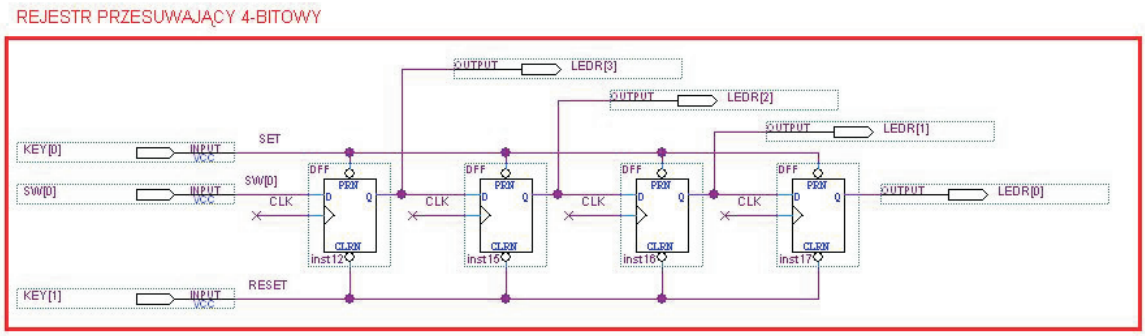
3. Skompilować schemat licznik_as.bdf (rys. 2.14) i skonfigurować układ programowalny.



Rys. 2.14. Schemat do sprawdzenia działania licznika asynchronicznego

Układ zbudowany z czterech przerzutników T jest licznikiem. Na wyjściach przerzutników można zaobserwować czterobitową liczbę w kodzie dwójkowym. Najbardziej znaczący bit, nazywany również najstarszym jest oznaczany jako MSB (ang. *Most Significant Bit*) i znajduje się na wyjściu przerzutnika podłączonego do diody LEDR[3]. Najmniej znaczący bit, czyli najmłodszy – LSB (ang. *Least Significant Bit*) to wyjście przerzutnika podłączonego do diody LEDR[0]. Należy podać zakres liczb pojawiających się na wyjściu licznika oraz ich kolejność.

4. Skompilować schemat rejestr_prz.bdf (rys. 2.15) i skonfigurować układ programowalny. Sprawdzić i opisać działanie układu, zwrócić uwagę na to, jak zmienia się stan sygnału na wyjściu LEDR[0] w zależności od stanu na wejściu SW[0].



Rys. 2.15. Schemat do sprawdzenia działania rejestru przesuwającego

5. Zbudować układ umożliwiający włączenie i wyłączenie diody LED jednym przyciskiem (np. KEY[0]), wykorzystać w tym celu przerzutnik T.
6. Korzystając z przerzutnika D i bramek, zrealizować dwa układy: pierwszy ma wykrywać zbocze opadające, a drugi zbocze narastające i opadające sygnału. W celu realizacji zadania można wykorzystać schemat ris_trig.bdf, realizujący wykrywanie zbocza narastającego.

2.8. Opracowanie wyników

Po wykonaniu ćwiczenia należy wykonać sprawozdanie (jedno na grupę laboratoryjną) według podanych niżej wytycznych. Wykonanie wymienionych zadań jest punktowane (z rozdzielczością 0.25 punktu). Na podstawie uzyskanych punktów wystawiane są oceny: poniżej 3 pkt. – ocena 2; (3,0 ÷ 3,25) pkt. – 3,0; (3,5 ÷ 3,75) pkt. – 3,5; (4,0 ÷ 4,25) pkt. – 4,0; (4,5 ÷ 4,75) pkt. – 4,5; (5,0 ÷ 5,25) pkt. – 5,0; 5,5 pkt. i więcej – 5,5.

Sprawozdanie z zajęć laboratoryjnych powinno zawierać:

— symulacje działania bramek i przerzutników opisane w paragrafie 0; tablice przejść, tablice prawdy i tablice wymuszeń przerzutników (wypełnione na zajęciach, lecz jeśli są mało czytelne, dodatkowo należy dołączyć przepisane na czysto (bądź wypełnione komputerowo i wydrukowane), dla każdego przerzutnika należy narysować graf; za zrealizowanie wszystkich wymienionych zadań można otrzymać 3,5 punktu; za brak lub błędne wykonanie zadań grupa otrzymuje punkty ujemne: –0,25 pkt za każdy błąd.
— rozwiązanie zadanie 0.3: opis działania układu, ew. symulacja – 0,5 pkt;
— rozwiązanie zadanie 0.4: opis działania układu, ew. symulacja – 0,5 pkt;
— rozwiązanie zadania 0.5 (schemat, symulacja, komentarz) – 0,5 pkt;
— rozwiązanie zadania 0.6 (schemat, symulacja, komentarz) – 0,5 pkt za każdy układ;

Prowadzący może zwolnić grupę z wykonania sprawozdania lub z pewnych jego części.

2.9. Pytania kontrolne

1. Zdefiniować pojęcia: czas narastania (opadania) i czas propagacji dla bramki TTL.
2. Zdefiniować pojęcia: czas ustalania i czas przetrzymywania dla przerzutników synchronicznych.
3. Przedstawić tablice prawd dla bramek NOT, AND, OR, NAND, NOR, XOR.
4. Dane są dwuwejściowe bramki AND, OR, NAND, NOR, XOR. Do jednego wejścia (każdej bramki) podłączono przebieg prostokątny o częstotliwości 1Hz, a do drugiego wejścia (sterującego) przełącznik. Określić kształt sygnałów na wyjściach bramek dla stanu niskiego i wysokiego na wejściu sterującym.
5. Wyjaśnić działanie przedstawionych w instrukcji przerzutników.
6. Narysować schematy logiczne asynchronicznych przerzutników RS, określić stan wyjść przerzutników (prostego i zanegowanego) dla stanu niedozwolonego.
7. Przeanalizować działanie układu z rys. 2.13, który służy do generowania przebiegu zegarowego za pomocą przycisków KEY[2] i KEY[3].
8. W jaki sposób można zrealizować przerzutnik T, mając do dyspozycji przerzutnik D lub JK?

2.10. Literatura

[1] Haras A., Nieznański J.: *Komputery i programowanie II. Materiały pomocnicze do laboratorium*. Gdańsk 1990.
[2] Skorupski A.: *Podstawy techniki cyfrowej*. Warszawa 2001.
[3] Kalisz J.: *Podstawy elektroniki cyfrowej*. Warszawa 1998.
[4] Majewski W.: *Układy logiczne*. Warszawa 1999.
[5] Górecki P.: *Układy cyfrowe, pierwsze kroki*. Warszawa 2004.

2.11. Załącznik

Tablice prawdy przerzutników

Q_n i Q_{n+1} – aktualny i następny stan wyjścia przerzutnika

Prz. asynchr. RS (NOR)			
S	R	Q_n	Q_{n+1}
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Prz. asynchr. RS (NAND)			
$\overline{S}$	$\overline{R}$	Q_n	Q_{n+1}
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Dla przerzutników asynchronicznych RS należy sprawdzić i zanotować stan wyjść (prostego i zanegowanego) w stanie niedozwolonym

Przerz. D		
D	Q_n	Q_{n+1}
0	0	
0	1	
1	0	
1	1	

Przerz. T		
T	Q_n	Q_{n+1}
0	0	
0	1	
1	0	
1	1	

Zatrzask			
D	E	Q_n	Q_{n+1}
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Przerz. JK			
J	K	Q_n	Q_{n+1}
0	0	0	
0	0	1	
0	1		
0	1	1	
1	0	0	
	0	1	
1	1	0	
1	1	1	

Przerz. RS synchr.			
S	R	Q_n	Q_{n+1}
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Tablice wymuszeń przerzutników:

$Q_n \rightarrow Q_{n+1}$	D	T	SR (synchr.)	JK
$0 \rightarrow 0$				
$0 \rightarrow 1$				
$1 \rightarrow 0$				
$1 \rightarrow 1$				

Grafy przerzutników:

3. Układy kombinacyjne

3.1. Cel ćwiczenia

Celem ćwiczenia jest zaznajomienie się z funkcjami boolowskimi (logicznymi) oraz opanowanie wiadomości dotyczących ich minimalizacji za pomocą tablic Karnaugh'a.

W ramach ćwiczenia wprowadzono multiplekser i zastosowano do realizacji funkcji logicznych.

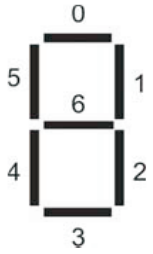
3.2. Wprowadzenie

Instrukcję do niniejszego ćwiczenia opracowano na podstawie [1]. W przypadku układów kombinacyjnych, stan dowolnego wyjścia w każdej chwili czasu zależy wyłącznie od aktualnej kombinacji stanów na wejściach.

Minimalizacja funkcji boolowskich ma na celu doprowadzanie do najprostszego jej zapisu (najmniej liter w zapisie). W konsekwencji jej realizacja wymaga najmniejszej liczby bramek z najmniejszą liczbą wejść. Na zajęciach laboratoryjnych proponowana jest metoda minimalizacji wykorzystująca tablice Karnaugh'a.

3.3. Przykład minimalizacji funkcji boolowskich

Jako przykład układu kombinacyjny wykorzystano transkoder kodu BCD (binarnego kodu dziesiętnego) na kod wyświetlacza siedmiosegmentowego – układ 7447. Tablicę prawdy dla omawianego układu (tab. 3.1) oraz rozmieszczenie poszczególnych segmentów w wyświetlaczu siedmiosegmentowym przedstawiono poniżej (rys. 3.1).



Rys. 3.1. Usytuowanie poszczególnych segmentów wyświetlacza siedmiosegmentowego

W omawianym przykładzie zrealizowano tylko podstawową funkcję transkodera 7447 – przekształcenie kodu BCD na kod wyświetlacza siedmiosegmentowego (zaznaczone w tabeli kolorem czerwonym). Pominięto działanie wejść BIN, RBIN i LTN oraz wyjścia RBON. Funkcje poszczególnych wejść:

— D, C, B, A – definiują w kodzie BCD wartość, która jest transkodowana na kod wyświetlacza siedmiosegmentowego,
— LTN – (ang. *lamp-test*) wejście aktywne stanem niskim, gdy pozostawimy je niepodłączone jest nieaktywne, w przypadku, gdy na tym wejściu jest stan niski i jednocześnie na wejściu BIN jest stan wysoki, wówczas wszystkie segmenty włączą się,

— BIN – (ang. *blanking input*) wejście aktywne stanem niskim, gdy pozostawimy je niepodłączone jest nieaktywne. W przypadku podania na to wejście stanu niskiego wszystkie segmenty zostaną wyłączone bez względu na stany pozostałych wejść,
— RBIN – (ang. *ripple-blanking input*) wejście aktywne stanem niskim, gdy pozostawimy je niepodłączone jest nieaktywne. Jeżeli omawiane wejście oraz wejścia A, B, C, D są w stanie niskim i jednocześnie wejście LTN jest w stanie wysokim, wówczas wszystkie segmenty są wyłączone, a wyjście RBON jest w stanie niskim.
— Funkcje poszczególnych wyjść:
— OA (0), OB. (1), OC (2), OD (3), OE (4), OF (5), OG (6) – wyjścia sterujące poszczególnymi segmentami wyświetlacza, aktywne stanem niskim (stan niski powoduje świecenie danego segmentu).
— RBON – aktywne stanem niskim, może być podłączone do wejścia RBIN kolejnego wyświetlacza w celu ewentualnego wyłączenia jego segmentów (w przypadku, gdy na bieżącym wyświetlaczu transkodowana jest cyfra zero oraz na kolejnym także). Wejście to pozwala wyłączyć wyświetlacze, dla których od skrajnie lewego począwszy pojawiłyby się cyfry zero.

Tabela 3.1

Tablica prawdy dla transkodera 7447

	numer segmentu							**0**	**1**	**2**	**3**	**4**	**5**	**6**		
Funkcja	**WEJŚCIA**							**WYJŚCIA**								
	LTN	*RBIN*	BIN	*D*	*C*	*B*	*A*	*OA*	*OB*	*OC*	*OD*	*OE*	*OF*	*OG*	*RBON*	*znak*
0	H	H	H	L	L	L	L	L	L	L	L	L	L	H	H	0
1	H	X	H	L	L	L	H	H	L	L	H	H	H	H	H	1
2	H	X	H	L	L	H	L	L	L	H	L	L	H	L	H	2
3	H	X	H	L	L	H	H	L	L	L	L	H	H	L	H	3
4	H	X	H	L	H	L	L	H	L	L	H	H	L	L	H	4
5	H	X	H	L	H	L	H	L	H	L	L	H	L	L	H	5
6	H	X	H	L	H	H	L	H	H	L	L	L	L	L	H	6
7	H	X	H	L	H	H	H	L	L	L	H	H	H	H	H	7
8	H	X	H	H	L	L	L	L	L	L	L	L	L	L	H	8
9	H	X	H	H	L	L	H	L	L	L	H	H	L	L	H	9
10	H	X	H	H	L	H	L	H	H	H	L	L	H	L	H	c
11	H	X	H	H	L	H	H	H	H	L	L	H	H	L	H	ɔ
12	H	X	H	H	H	L	L	H	L	H	H	H	L	L	H	u
13	H	X	H	H	H	L	H	L	H	H	L	H	L	L	H	ɔ
14	H	X	H	H	H	H	L	H	H	H	L	L	L	L	H	r
15	H	X	H	H	H	H	H	H	H	H	H	H	H	H	H	
BI	X	X	L	X	X	X	X	H	H	H	H	H	H	H	X	
RBI	H	L	X	L	L	L	L	H	H	H	H	H	H	H	L	
LT	L	X	H	X	X	X	X	L	L	L	L	L	L	L	H	8

Do realizacji wybrano funkcję kombinacyjną odpowiadającą wyjściu *OG* do, którego podłączony jest segment numer 6. Tablicową reprezentację funkcji boolowskiej dla omawianego wyjścia przedstawiono poniżej.

Tabela 3.2

Tablica wartości funkcji boolowskiej dla wyjścia *OG* (segment numer 6)

D	*C*	*B*	*A*	*OG*
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

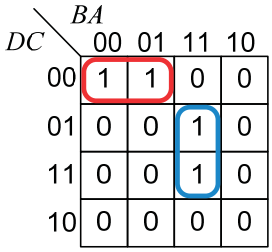
Minimalizację funkcji kombinacyjnych można realizować metodami algebraicznymi stosując tzw. reguły sklejania:

$$\begin{aligned} AB + A\overline{B} &= A \\ (A+B)(A+\overline{B}) &= A \end{aligned} \tag{3.1}$$

gdzie: *A* i *B* oznaczają zmienne lub funkcje boolowskie. Zgodnie z regułami sklejania sumę lub iloczyn dwóch wyrażeń różniących się tylko znakiem negacji nad jedną zmienną można zastąpić jednym wyrażeniem, odrzucając z zapisu zmienną, która ulega zmianie (występowanie negacji i jej brak).

Poniżej przedstawiono tablicę Karnaugh'a. Należy zauważyć, iż kolejne wiersze i kolumny tablicy opisane są w kodzie Graya. Dwa kolejne słowa kodowe (kratki) oraz ostatnie i pierwsze słowo kodowe (w dowolnej kolumnie lub wierszu) różnią się tylko stanem jednego bitu. Główną zaletą zastosowania tablic Karnaugh'a do minimalizacji funkcji boolowskich jest łatwość dostrzeżenia wyrażeń sąsiednich, które przylegają bezpośrednio do siebie lub poprzez przeciwległe krawędzie. Do opisanej kodem Gray'a tablicy wpisuje się wartości funkcji odpowiadające poszczególnym kombinacjom zmiennych.

Poniżej przedstawiono tablicę Karnaugh'a dla wyjścia *OG* w przypadku sklejania jedynek:



Wykorzystując tablice Karnaugh'a do minimalizacji funkcji kombinacyjnych, realizujemy sklejanie **wyrażeń sąsiednich**. Pierwszym etapem sklejania jest wyodrębnienie jak **największych grup**. Grupy składają się z kratek o jednakowych symbolach (1 lub 0), tworząc kształty prostokątów. Boki grup muszą mieć wymiary 2^m, m = 0, 1, 2, 3, … . Poszczególne kratki można wykorzystywać wielokrotnie w celu uzyskania jak największych grup. Wyodrębnienie grup kończymy, gdy każdy sklejany symbol (1 lub 0) będzie przynajmniej raz wykorzystany.

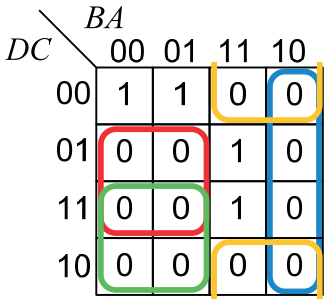
Niekiedy wartość funkcji boolowskiej dla pewnych kombinacji jest nieistotna (ang. *don't care condition*), tzn. funkcja jest częściowo określona. W tablicy Karnaugh'a należy ten fakt uwidocznić, wpisując w odpowiednie kratki specjalne symbole, rożne od 0 i 1 (np. x, –). Wyodrębniając w tablicy grupy wyrażeń podlegające sklejaniu, można te symbole traktować dowolnie, jako zera lub jedynki, co stwarza możliwość uzyskania większych grup, a tym samym prostszej postaci algebraicznego zapisu funkcji.

Każdej grupie (także jednokartkowej) odpowiada jeden iloczyn (dla grup z jedynkami) lub jedna suma (dla grup z zerami). W każdym iloczynie lub sumie występują tylko zmienne odpowiadające współrzędnym kratek, które nie ulegają zmianie w obrębie całej rozpatrywanej grupy (rozpatrując wartości zmiennych wzdłuż boków prostokąta obejmującego grupę). Współrzędnym o wartości jeden odpowiada zmienna prosta w iloczynie oraz zmienna zanegowana w sumie. Współrzędnym o wartości zero odpowiada zmienna zanegowana w iloczynie i zmienna prosta w sumie.

W omawianym przykładzie dla sklejania jedynek uzyskujemy pierwszą postać normalną:

$$OG = \overline{D} \cdot \overline{C} \cdot \overline{B} + C \cdot B \cdot A \tag{3.2}$$

Poniżej przedstawiono tablicę Karnaugh'a dla wyjścia *OG* (segment numer 6) dla sklejanie zer. Wyodrębniono jeden z czterech optymalnych układów grup.



Dla sklejania zer uzyskaną funkcję nazywamy drugą postacią normalną (w tym przypadku istnieją cztery równie dobre rozwiązania):

$$OG = (\overline{C} + B) \cdot (\overline{D} + B) \cdot (C + \overline{B}) \cdot (\overline{B} + A) \tag{3.3}$$

W omawianym przypadku prostszą postać zapisu funkcji kombinacyjnej uzyskano dla sklejania jedynek (w ogólnym przypadku może być odwrotnie).

3.4. Realizacja układów kombinacyjnych za pomocą bramek NAND lub NOR

Schemat układu logicznego odpowiadającego bezpośrednio postaci funkcji uzyskanej w wyniku minimalizacji (np. (3.2) i (3.3)) zawiera funktory: AND, OR oraz NOT. Najczęściej wykorzystywanymi w praktyce bramkami są NAND i NOR. Uzyskane formuły funkcji (3.2) i (3.3) można przekształcić algebraicznie, aby w sposób bezpośredni odpowiadały realizacji za pomocą bramek NAND (rys. 3.2 i rys. 3.4) lub NOR (rys. 3.3 i rys. 3.5). W tym celu wykorzystano prawa de Morgana:

$$\begin{aligned} \overline{A+B} &= \overline{A}\cdot\overline{B} \\ \overline{A\cdot B} &= \overline{A}+\overline{B} \end{aligned} \tag{3.4}$$

oraz możliwość podwójnego zanegowania zmiennej lub funkcji boolowskiej:

$$\overline{\overline{A}} = A \tag{3.5}$$

W przypadku sumy iloczynów (3.2) dla realizacji funkcji za pomocą bramek NAND (rys. 3.2) można uzyskać opis:

$$\begin{aligned} OG &= \overline{\overline{\overline{D}\cdot\overline{C}\cdot\overline{B}+C\cdot B\cdot A}} \\ OG &= \overline{(\overline{\overline{D}\cdot\overline{C}\cdot\overline{B}})\cdot\overline{(C\cdot B\cdot A)}} \end{aligned} \tag{3.6}$$

Formuła odpowiadająca bezpośrednio realizacji funkcji opisanej zależnością (3.2) za pomocą bramek NOR (rys. 3.3) może być przedstawiona następująco:

$$\begin{aligned} OG &= \overline{\overline{(\overline{D}\cdot\overline{C}\cdot\overline{B})}+\overline{(C\cdot B\cdot A)}} \\ OG &= \overline{(\overline{\overline{D}}+\overline{\overline{C}}+\overline{\overline{B}})}+\overline{(\overline{C}+\overline{B}+\overline{A})} \\ OG &= \overline{\overline{(D+C+B)}+\overline{(\overline{C}+\overline{B}+\overline{A})}} \end{aligned} \tag{3.7}$$

Natomiast w przypadku iloczynu sum (3.3) i jego realizacji za pomocą bramek NAND (rys. 3.4) można uzyskać zależność:

$$\begin{aligned} OG &= \overline{\overline{(\overline{C}+B)}\cdot\overline{(\overline{D}+B)}\cdot\overline{(C+\overline{B})}\cdot\overline{(\overline{B}+A)}} \\ OG &= \overline{(\overline{\overline{C}}\cdot\overline{B})}\cdot\overline{(\overline{\overline{D}}\cdot\overline{B})}\cdot\overline{(\overline{C}\cdot\overline{\overline{B}})}\cdot\overline{(\overline{\overline{B}}\cdot\overline{A})} \\ OG &= \overline{\overline{(C\cdot\overline{B})}\cdot\overline{(D\cdot\overline{B})}\cdot\overline{(\overline{C}\cdot B)}\cdot\overline{(B\cdot\overline{A})}} \end{aligned} \tag{3.8}$$

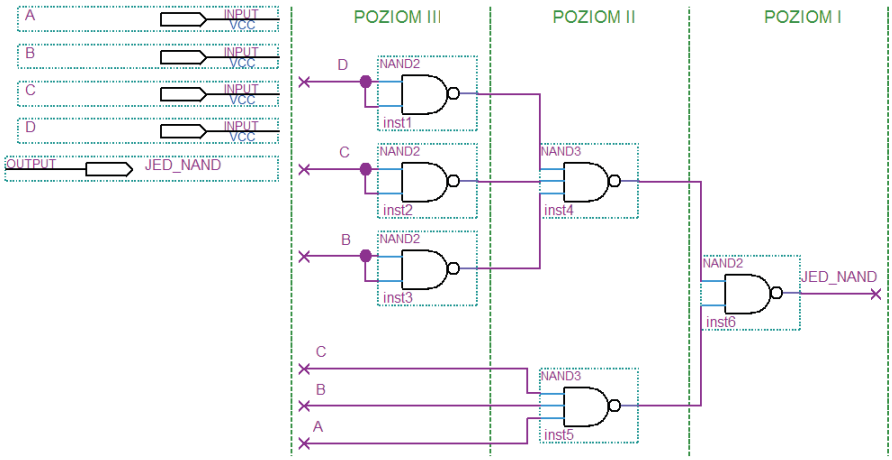
Opis funkcji (3.3) bezpośrednio odpowiadający realizacji za pomocą bramek NOR (rys. 3.5) można przedstawić następująco:

$$\begin{aligned} OG &= \overline{\overline{(\overline{C}+B)}\cdot\overline{(\overline{D}+B)}\cdot\overline{(C+\overline{B})}\cdot\overline{(\overline{B}+A)}} \\ OG &= \overline{\overline{(\overline{C}+B)}+\overline{(\overline{D}+B)}+\overline{(C+\overline{B})}+\overline{(\overline{B}+A)}} \end{aligned} \tag{3.9}$$

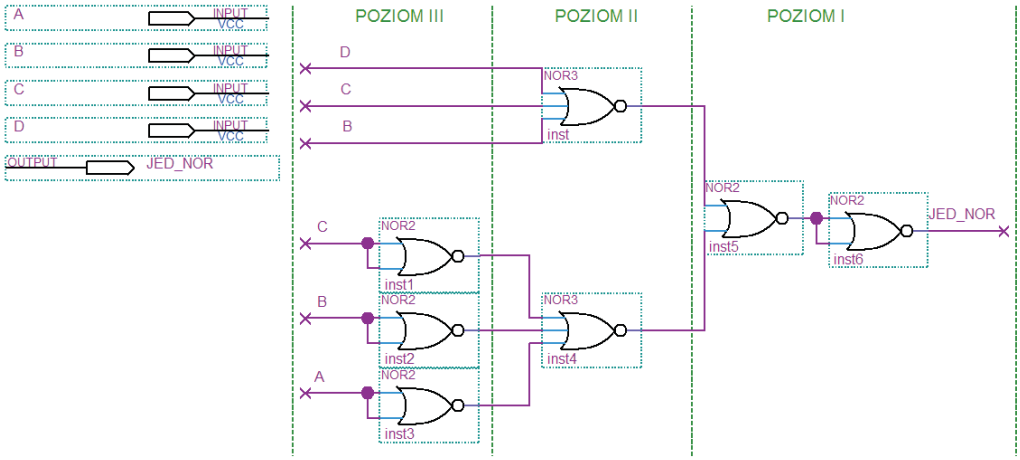
Możliwe jest także bezpośrednie rysowanie schematów (rys. 3.2 ÷ 3.5) na podstawie formuł funkcji uzyskanych z minimalizacji ((3.2) i (3.3) – bez realizowania wyżej zaprezentowanych przekształceń algebraicznych). Poniższe zasady formalizują kolejne etapy budowy schematu.

Jeżeli zapis algebraiczny jest postaci sumy iloczynów (3.2) to:

— poziom I buduje się z jednego elementu NAND lub dwóch elementów NOR;
— poziom II zawiera tyle elementów NAND lub NOR, ile elementarnych iloczynów występuje w wyrażeniu;
— poziom III w przypadku elementów NAND wytwarza negacje tych zmiennych, które **są** zanegowane w zapisie algebraicznym, a w przypadku elementów NOR – negacje tych zmiennych, które w zapisie algebraicznym **nie są** zanegowane.



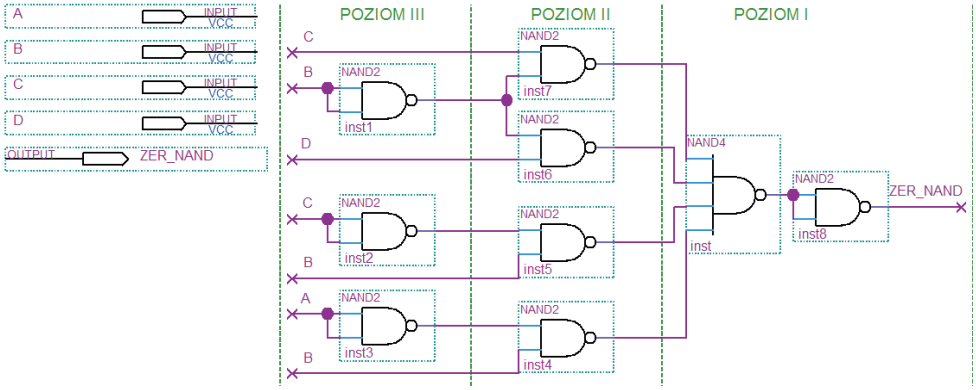
Rys. 3.2. Realizacja funkcji kombinacyjnej (3.2) lub (3.6) za pomocą bramek NAND



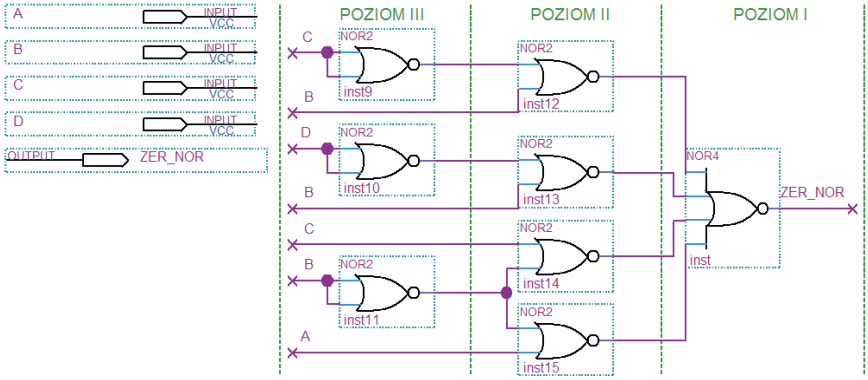
Rys. 3.3. Realizacja funkcji kombinacyjnej (3.2) lub (3.7) za pomocą bramek NOR

Jeśli zapis algebraiczny jest postaci iloczynu sum (3.3) to:

— poziom I buduje się z dwóch elementów NAND lub jednego elementu NOR;
— poziom II zawiera tyle elementów NAND lub NOR, ile elementarnych sum występuje w wyrażeniu;
— poziom III w przypadku elementów NAND wytwarza negacje tych zmiennych, które **nie są** zanegowane w zapisie algebraicznym, a w przypadku elementów NOR – negacje tych zmiennych, które w zapisie algebraicznym **są** zanegowane.



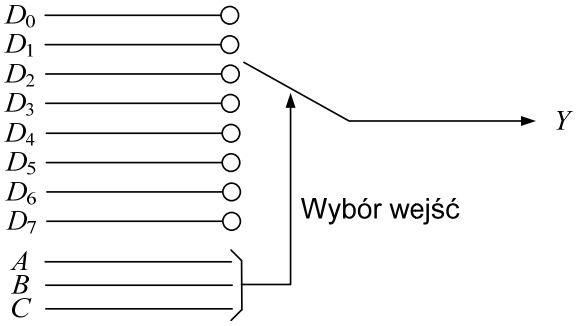
Rys. 3.4. Realizacja funkcji kombinacyjnej (3.3) lub (3.8) za pomocą bramek NAND



Rys. 3.5. Realizacja funkcji kombinacyjnej (3.3) lub (3.9) za pomocą bramek NOR

3.5. Realizacja układu kombinacyjnego za pomocą multipleksera

Multiplekser to układ o n wejściach wybierających (adresowych), 2^n wejściach informacyjnych i jednym wyjściu. Każdej z 2^n kombinacji wejść wybierających odpowiada jedno wejście informacyjne. Gdy na wejściu wybierającym występuje kombinacja odpowiadająca danemu wejściu informacyjnemu, to wejście to zastaje przyłączone do wyjścia (rys. 3.6).



Rys. 3.6. Multiplekser o ośmiu wejściach informacyjnych D_7–D_0, trzech wejściach wybierających *C*, *B*, *A* i jednym wyjściu *Y*, jako przełącznik wielopozycyjny

Tablicę prawdy funkcji logicznej możemy interpretować w ten sposób, że dla każdej kombinacji wejść należy **wybrać** 0 albo 1. Jeśli więc *D*, *C*, *B*, *A* interpretować, jako wejścia wybierające, to wartość funkcji *OG*, staje się równa wejściom informacyjnym, odpowiadającym poszczególnym adresom *DCBA*. Multiplekser można wiec traktować, jako układ realizujący dowolną funkcję *n* zmiennych, gdzie *n* jest liczbą wejść adresowych. Realizacja funkcji polega na odpowiednim przypisaniu wejściom informacyjnym wartości 0 albo 1.

Można zmniejszyć o jeden potrzebną liczbę wejść adresowych, tzn. zrealizować dowolną funkcję *n* zmiennych za pomocą multipleksera o (*n*–1) wejściach wybierających. Dla dowolnej funkcji *n* zmiennych każdej kombinacji (*n*–1) zmiennych odpowiadają dokładnie dwa wiersze w tablicy kombinacji. Istnieją dokładnie cztery warianty wartości, jakie funkcja może przyjąć dla pary kombinacji wejściowych (tab. 3.3).

Tabela 3.3

Fragment tablicy z czterema wariantami wartości funkcji boolowskiej

X_3	X_2	X_1	X_0	Y	Wartości wejść informacyjnych
			0	0	0
			1	0	
			0	0	X_0
			1	1	
			0	1	$\overline{X_0}$
			1	0	
			0	1	1
			1	1	

Realizacja funkcji *n* zmiennych za pomocą multipleksera o (*n*–1) wejściach adresowych polega na przypisaniu (*n*–1) zmiennych do wejść adresowych oraz na odpowiednim podaniu 0, 1, X_0 lub $\overline{X_0}$ na wejścia informacyjne. W przypadku, gdy dla obu kombinacji zmiennych (wierszy w tablicy) wartość funkcji wynosi 0, nie zależy ona od najmniej znaczącej ze zmiennych (X_0). Zatem do wejścia informacyjnego wybieranego dla tej kombinacji należy podłączyć wartość 0. Analogicznie można skomentować sytuację, gdy dla pary wierszy funkcja przyjmuje wartość 1. W przypadku, gdy dla pary kombinacji zmiennych funkcja przyjmuje wartości 0 – dla pierwszej i 1 – dla drugiej, zauważamy, iż są one takie same, jak wartości zmiennej najmniej znaczącej – X_0. Do wejścia informacyjnego w tym przypadku należy podłączyć zmienną najmniej znaczącą – X_0. Gdy dla pary kombinacji zmiennych funkcja przyjmuje wartości 1 – dla pierwszej i 0 – dla drugiej, należy dla odpowiadającego jej wejścia informacyjnego podłączyć negację zmiennej najmniej znaczącej – $\overline{X_0}$.

Na rys. 3.7 przedstawiono schemat projektu dotyczącego badania układów kombinacyjnych. Zostały na nim umieszczone następujące komponenty:

- dekoder 7447
- cztery realizacje układu kombinacyjnego dla segmentu numer sześć w postaci bloków funkcjonalnych:
  - *jedynkiNAND* – za pomocą bramek NAND w przypadku sklejania jedynek,
  - *jedynkiNOR* – za pomocą bramek NOR w przypadku sklejania jedynek,
  - *zeraNAND* – za pomocą bramek NAND w przypadku sklejania zer,
  - *zeraNOR* – za pomocą bramek NOR w przypadku sklejania zer,
- Multiplekser 74151.

Tabela 3.4

Tablica wartości funkcji boolowskiej dla wyjścia *OG* (sterującego segmentem numer sześć) i wartości wejść informacyjnych zapewniających jej realizację za pomocą multipleksera o trzech wejściach adresowych

D	*C*	*B*	*A*	*OG*	Wartości wejść informacyjnych	
0	0	0	0	1	1	D_0
0	0	0	1	1		
0	0	1	0	0	0	D_1
0	0	1	1	0		
0	1	0	0	0	0	D_2
0	1	0	1	0		
0	1	1	0	0	*A*	D_3
0	1	1	1	1		
1	0	0	0	0	0	D_4
1	0	0	1	0		
1	0	1	0	0	0	D_5
1	0	1	1	0		
1	1	0	0	0	0	D_6
1	1	0	1	0		
1	1	1	0	0	*A*	D_7
1	1	1	1	1		

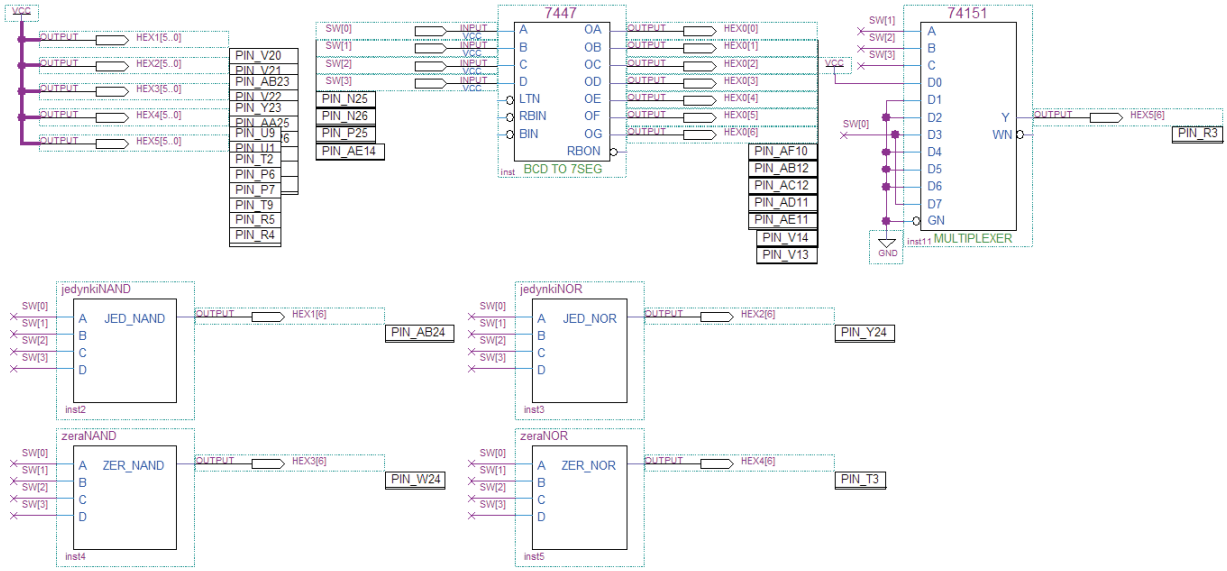
Aby zrealizować własny blok funkcjonalny, należy:

— utworzyć nowy schemat (*File –> New –> Block Diagram /Schematic File*), a następnie zrealizować odpowiedni dla numeru stanowiska laboratoryjnego układ kombinacyjny w sposób analogiczny do zaprezentowanego powyżej (rys. 3.2 ÷ 3.5);

— utworzony blok należy skompilować i sprawdzić symulacyjnie. W tym celu w oknie *Project Navigator* (lewa kolumna) należy wybrać zakładkę *Files*, aby wyświetlić wszystkie pliki znajdujące się w projekcie; następnie prawym przyciskiem myszy należy wybrać plik ze schematem i wybrać opcję *Set as Top-Level Entity*; można wówczas skompilować utworzony schemat oraz przetestować go symulacyjnie w sposób opisany w punkcie 0;

— następnie należy utworzyć symbol, który zostanie wstawiony, jako blok do głównego programu przedstawionego na rys. 3.7; mając otwarty i aktywny schemat *.bdf należy wybrać polecenie *File –> Create / Update –> Create Symbol Files for Current File* a następnie należy zapisać symbol z domyślną nazwą;

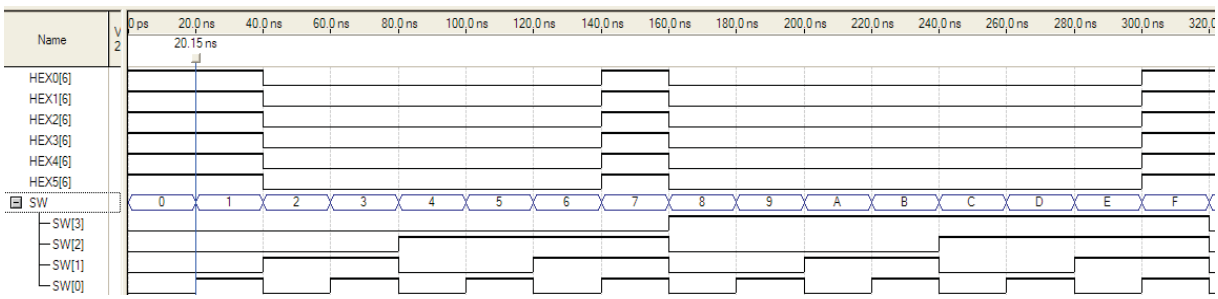
— ostatni etap to wstawienie symbolu do schematu; należy wybrać polecenie wstawienia symbolu, a następnie w oknie *Symbol* wyświetlić pliki znajdujące się w katalogu *Project* i wybrać odpowiedni plik.



Rys. 3.7. Schemat projektu do badania układów kombinacyjnych

Na rys. 3.8 przedstawiono wyniki badań symulacyjnych układów kombinacyjnych zamieszczonych na rys. 3.7. Należy zwrócić uwagę, iż w celu ułatwienia generacji przebiegów wejściowych oraz ułatwienia w prowadzeniu analizy wykonanych symulacji wykorzystano zgrupowanie wejść **SW[3..0]** (należy wybrać *Grouping* z menu kontekstowego). Po wybraniu opcji *Value –> Count Value* (menu kontekstowe lub wybierając ikonę:) możliwe jest jednoczesne zdefiniowanie wartości na wszystkich liniach rozważanej magistrali. Wybór ten jest możliwy po wcześniejszym uaktywnieniu edytowanej magistrali (należy kliknąć na opis numeru wejścia/wyjścia znajdujący się po lewej stronie opisu magistrali). W uzyskanym oknie dialogowym w zakładce *Counting* można ustawić, od jakiej wartości zaczynamy inkrementowanie oraz co jaką wartość, a następnie korzystając z zakładki *Timing* można wyspecyfikować, dla jakiego przedziału czasu ma być realizowana inkrementacja i z jakim krokiem czasowym.

Analizując poniższą symulację, można zauważyć, że uzyskane wartości wyjść poszczególnych układów kombinacyjnych dla wszystkich wartości kombinacji wejściowych są takie same, a ponadto są zgodne z prezentowanymi w tablicy prawdy. W celu ułatwienia analizy wyników symulacji rozważane wyjścia układów kombinacyjnych zestawiono w bezpośrednim sąsiedztwie (wraz z wyjściem HEX0[6], które w tym przypadku jest wyjściem referencyjnym).



Rys. 3.8. Wyniki badania symulacyjnego układów kombinacyjnych zamieszczonych na schemacie z rys. 3.7

3.6. Przygotowanie do zajęć

1. Obowiązuje znajomość minimalizacji funkcji boolowskich za pomocą tablicy Karnaugh'a (przewidywana jest wejściówka z tego zakresu), działania multipleksera wraz z zastosowaniem do realizacji funkcji logicznych.
2. Wymagana jest znajomość realizacji funkcji boolowskich za pomocą bramek NAND i NOR w przypadku zapisów uzyskanych dla sklejania jedynek i zer (przewidywana jest wejściówka z tego zakresu).
3. Obowiązuje znajomość działania dekodera 7447.
4. Na stronie internetowej przedmiotu jest udostępniony do pobrania skompilowany projekt (kombinacyjne.zip) zawierający miedzy innymi schematy przedstawione na rys. 3.7 (kombinacyjne.bdf) oraz plik symulacyjny przedstawiony na rys. 3.8 (kombinacyjne.vwf). Na podstawie schematu kombinacyjne.bdf należy **przygotować własny schemat dla numeru segmentu k-1**, gdzie k oznacza numer stanowiska laboratoryjnego (stanowisko po lewej stronie prowadzącego, przylegające do biurka prowadzącego ma numer 1, numery pozostałych stanowisk wzrastają zgodnie z ruchem wskazówek zegara, zatem stanowisko po prawej stronie prowadzącego ma numer 6). Każda grupa laboratoryjna zobligowana jest do **przeprowadzenia syntezy czterech układów kombinacyjnych** (rys. 3.2 ÷ 3.5). Następnie należy **zrealizować wskazaną funkcję boolowską za pomocą multipleksera** 74151. Należy, też przeprowadzić **symulację działania zbudowanych układów kombinacyjnych** dla wszystkich możliwych kombinacji sygnałów wejściowych w sposób analogiczny, do zaprezentowanego na rys. 3.8. Należy **przynieść na zajęcia uzyskane schematy oraz wyniki ich symulacji** (w postaci elektronicznej i papierowej). Należy być **przygotowanym na dyskusję otrzymanych wyników** z prowadzącym zajęcia. Utworzony projekt należy skompilować (plik ze schematem analogicznym do pokazanego na rys. 3.7 należy z powrotem ustawić, jako najwyższy w hierarchii – opcja: *Set as Top-Level Entity*) i przynieść na zajęcia.

3.7. Przebieg ćwiczenia

1. Uruchomić projekt kombinacyjne.qpf udostępniony na stronie internetowej laboratorium przedmiotu techniki mikroprocesorowe i pokazać do wglądu prowadzącemu zajęcia.
2. Uruchomić projekt realizowany w ramach przygotowania do zajęć (dla odpowiedniego numeru segmentu) i pokazać do wglądu prowadzącemu zajęcia.
3. Dla chętnych: zrealizować układ kombinacyjny dla wskazanego numeru segmentu za pomocą multipleksera *lpm_mux* (należy wykorzystać tylko porty *data*[][] – wejścia informacyjne, *sel*[] – wejścia wybierające i *result*[] – wyjście) i stałej *lpm_constant*.
4. Dla chętnych: zrealizować komparator dwu liczb dwubitowych (posiadający cztery wejścia i trzy wyjścia A > B, A < B i A = B) za pomocą dowolnych bramek lub multiplekserów.

3.8. Opracowanie wyników

Po wykonaniu ćwiczenia należy wykonać sprawozdanie (jedno na grupę laboratoryjną) według podanych niżej wytycznych. Wykonanie (bądź nie wykonanie) wymienionych zadań jest punktowane (z rozdzielczością 0,25 punktu). Na podstawie uzyskanych punktów wystawiane są oceny według kryteriów zawartych w instrukcji do poprzedniego ćwiczenia.

Należy wykonać sprawozdanie z zajęć laboratoryjnych, które powinno zawierać:

1. Syntezę czterech układów kombinacyjnych dla odpowiedniego numeru segmentu zrealizowane za pomocą (tablicową reprezentację funkcji kombinacyjnej, tablice Karnaugh'a,

równania algebraiczne funkcji logicznych uzyskane dla sklejania jedynek i zer – pierwszą i drugą postać normalną, schematy):
   - bramek NAND w przypadku sklejania jedynek,
   - bramek NOR w przypadku sklejania jedynek,
   - bramek NAND w przypadku sklejania zer,
   - bramek NOR w przypadku sklejania zer.

   Za zrealizowanie wszystkich wymienionych zadań można otrzymać 4 punkty; za nie wykonanie zadań grupa otrzymuje punkty ujemne: za brak syntezy wymaganego układu kombinacyjnego –1 pkt., za każdy układ, za brak symulacji zaprojektowanych układów –1 pkt., za błąd w tablicy prawdy, w tablicy Karnaugh'a, w sklejaniu, zapisach algebraicznych i na schematach –0,25 pkt.
2. Realizację układu kombinacyjnego dla odpowiedniego numeru segmentu za pomocą multipleksera (tablica wartości wejść informacyjnych dla realizacji za pomocą multipleksera o trzech wejściach adresowych, schemat zawierający multiplekser z trzema wejściami wybierającymi i odpowiednią konfiguracją połączeń dla wejść informacyjnych). Za zrealizowanie wymienionych zadań można otrzymać 1 punkt; za niewykonanie zadań grupa otrzymuje punkty ujemne: za brak tablicy wartości wejść informacyjnych –0,5 pkt., za brak multipleksera na schemacie –0,5 pkt., za każdy błąd –0,25 pkt.
3. Dodatkowo – schemat z realizacją układu kombinacyjnego dla wskazanego numeru segmentu za pomocą multipleksera *lpm_mux* i jego symulacja. Za zrealizowanie wymienionego zadania można otrzymać +1 punkt; za każdy błąd –0,25 pkt.
4. Dodatkowo – schemat z realizacją komparatora dwu liczb dwubitowych i jego symulacja. Za zrealizowanie wymienionego zadania można otrzymać +2 punkty; za każdy błąd –0,25 pkt.

Prowadzący może **zwolnić grupę z wykonania sprawozdania**, jeśli zrealizuje ona poprawnie na zajęciach zadania opisane w punktach 0 i 0.

Umieszczając w sprawozdaniu wyniki **symulacji, należy je obszernie skomentować**, w przeciwnym razie nie będą one uwzględniane.

3.9. Literatura

[1] Nieznański Janusz, niepublikowane materiały z wykładu „Podstawy techniki cyfrowej i mikroprocesorowej"

4. Rejestry i pamięci

4.1. Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z podstawowymi typami rejestrów, sprawdzenie ich działania oraz zapoznanie się z przykładami ich zastosowań za pomocą programu Quartus II i zestawu laboratoryjnego DE2.

4.2. Wprowadzenie

4.2.1. Definicje i podział rejestrów

Rejestr jest układem, który służy do pamiętania (przechowywania) informacji cyfrowej. Jako blok funkcjonalny występuje niemal we wszystkich poważniejszych urządzeniach cyfrowych. Na przykład pamięć (RAM, ROM) może być traktowana jako zespół rejestrów. Jednostka centralna komputera (np. mikroprocesor) zawiera zwykle kilkadziesiąt rejestrów o różnym przeznaczeniu. Rejestry różnią się przede wszystkim sposobem wprowadzania i wyprowadzania pamiętanej informacji. Pod tym względem można je podzielić na:

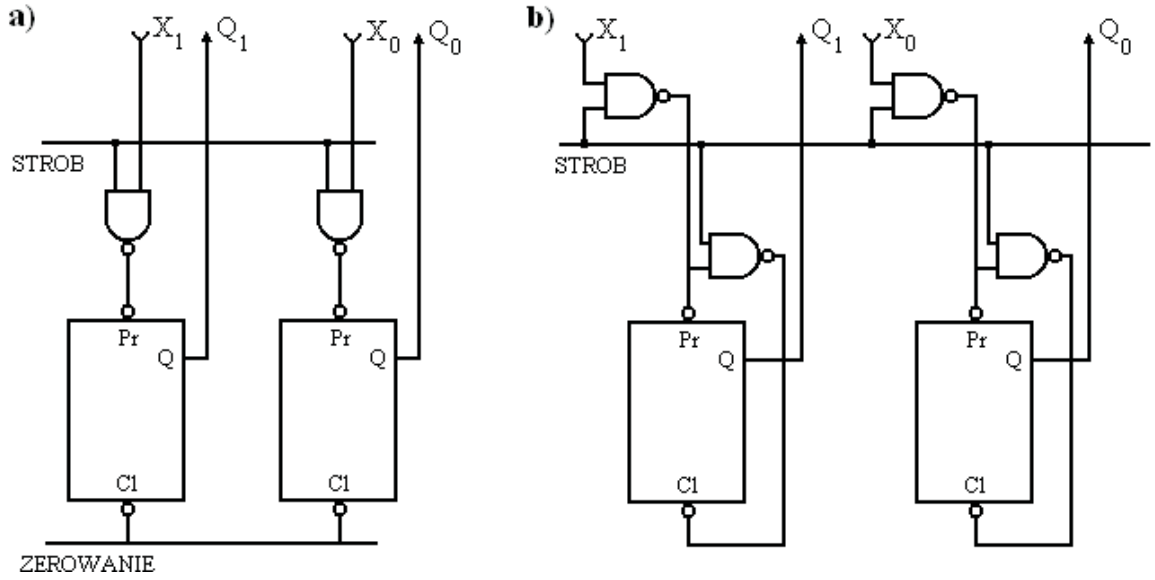
1) rejestry równoległe umożliwiające równoległe wprowadzanie i wyprowadzanie informacji (jednocześnie ze wszystkich i do wszystkich pozycji rejestru),
2) rejestry szeregowe umożliwiające szeregowe wprowadzanie i wyprowadzanie informacji, tzn. kolejno bit po bicie,
3) szeregowo-równoległe umożliwiające szeregowe wprowadzanie i równoległe wyprowadzanie informacji,
4) równoległo-szeregowe umożliwiające równoległe wprowadzanie i szeregowe wyprowadzanie informacji,
5) rejestry uniwersalne, które umożliwiają wprowadzanie i wyprowadzanie informacji zarówno w sposób równoległy, jak i szeregowy.

Podstawowym elementem rejestru jest przerzutnik, który umożliwia zapamiętanie jednego bitu informacji cyfrowej. Parametry charakteryzujące rejestr to (a) długość rejestru, równa liczbie przerzutników n (b) pojemność rejestru jest równa 2^n i określa maksymalną ilość różnych informacji, które mogą być zapamiętane w rejestrze (c) szybkość pracy rejestru; dla rejestru równoległego będzie to czas trwania wprowadzania i wyprowadzania informacji; w przypadku rejestru szeregowego będzie to maksymalna częstotliwość impulsów zegarowych, przy której przesuwanie kombinacji nie ulega jeszcze zniekształceniu.

4.2.2. Rejestry równoległe

Wprowadzenie równoległe informacji do rejestru odbywa się najczęściej za pomocą asynchronicznych wejść ustawiających CLEAR i PRESET przerzutnika. Informacja wyjściowa jest dostępna na wyjściach **Q** przerzutników. Informacja z wejść równoległych **X** jest wprowadzana do rejestru podczas wysokiego stanu sygnału STROB. Przy niskim stanie tego sygnału informacja wprowadzona do rejestru zostaje w nim zatrzaśnięta. Rejestry tego typu

noszą więc często nazwę zatrzasków (ang. latch). Przykłady realizacji tego typu rejestrów przedstawiono na rysunku 5.1 .



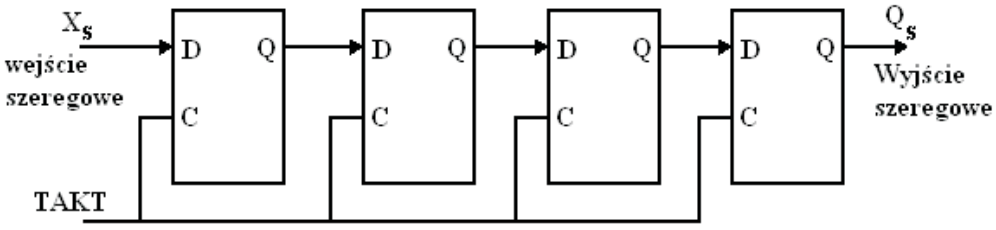
Rys. 4.1. Układy równoległego wprowadzania informacji do rejestru

W przypadku (a) z rys. 4.1 przed wprowadzeniem informacji należy rejestr wyzerować, w przypadku (b) nie jest to konieczne.

4.2.3. Rejestry przesuwające

Rejestry szeregowe

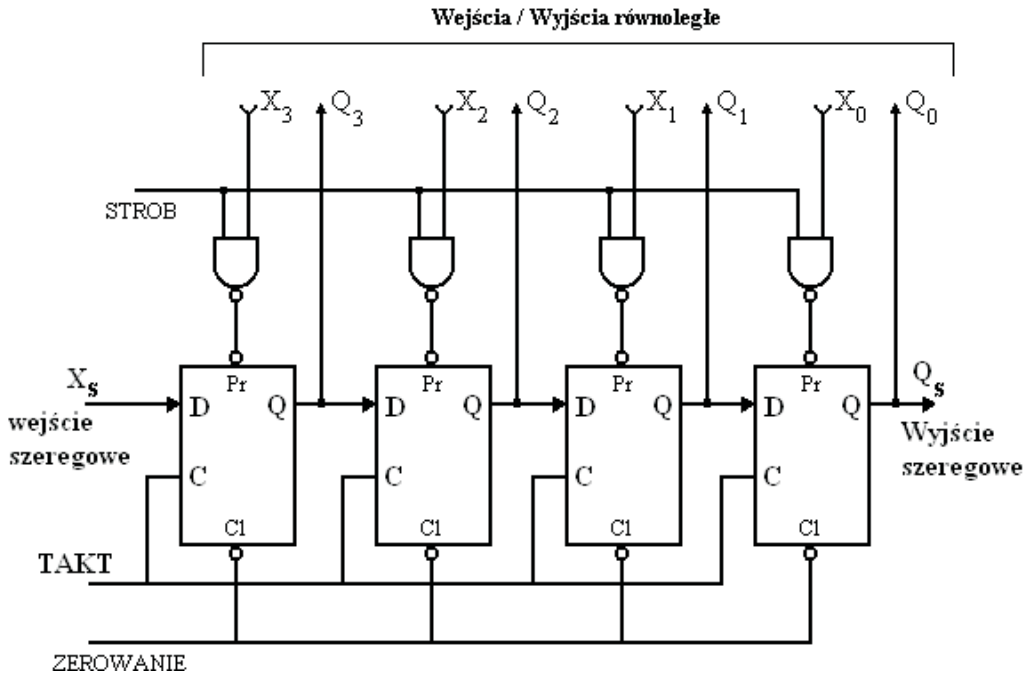
Rejestry szeregowe umożliwiają przesuwanie wprowadzonej, informacji w lewo lub w prawo (rejestry jednokierunkowe), bądź też zarówno w lewo, jak i w prawo (rejestry dwukierunkowe – rewersyjne). Przesuwanie informacji w rejestrze odbywa się synchronicznie w takt zmian sygnału taktującego, o jedną pozycję. Ogólną strukturę rejestru szeregowego przedstawiarys. 4.2.



Rys. 4.2. Struktura rejestru szeregowego

Rejestry równoległo-szeregowe i szeregowo-równoległe

Rejestry równoległo-szeregowe i szeregowo-równoległe są również rejestrami przesuwającymi. Konstrukcja tych rejestrów polega na kombinacji dwu poprzednich typów. W rejestrach równoległo-szeregowych wprowadzanie informacji następuje równolegle, a wyprowadzanie szeregowo. W rejestrach szeregowo-równoległych jest na odwrót. Rejestry o dowolnych kombinacjach wprowadzania i wyprowadzania informacji nazywane są rejestrami uniwersalnymi. Przykład takiego rejestru został przedstawiony na rys. 4.3.



Rys. 4.3. Struktura rejestru uniwersalnego

Jeśli STROB=1, to możemy wprowadzić równolegle informację do rejestru. Przed wprowadzeniem informacji rejestr musi być wyzerowany. Jeśli STROB=0, w czasie taktowania rejestr przesuwa informację w prawo. W rejestrach rewersyjnych wprowadza się dwa wejścia dla sygnału taktującego (zegarowego), osobne dla przesuwania w lewo i prawo. Stosowane jest także rozwiązanie z jednym wejściem sygnału taktującego oraz dodatkowym specjalnym wejściem, którego stan określa kierunek przesuwania.

Rodzaje przesuwania informacji

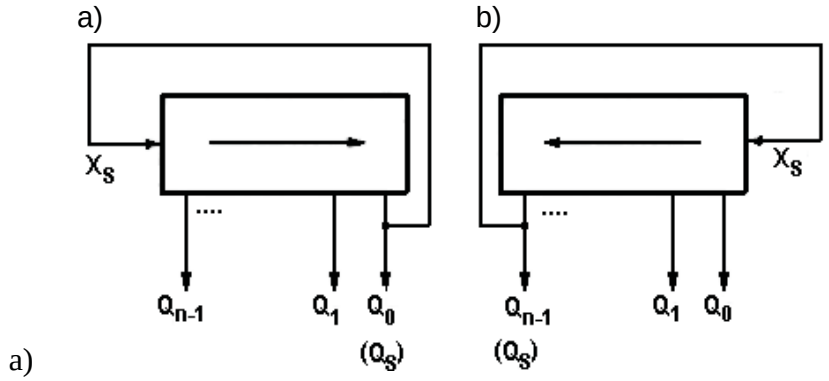
W rejestrach szeregowych można uzyskać różne rodzaje przesuwania. Są to:
- przesuwanie normalne (logiczne),
- przesuwanie arytmetyczne,
- przesuwanie cykliczne (rotacja).

Przesuwanie logiczne polega na tym, że na wejście szeregowe X_S podane jest 0. Informacja zawarta w rejestrze przesuwana jest w lewo bądź w prawo. Przy przesuwaniu arytmetycznym informacja zawarta w rejestrze traktowana jest jak liczba. Dlatego sposób realizacji przesuwania arytmetycznego zależy od kodu zapisu tej liczby. Jeżeli zawartość rejestru traktowana będzie jak liczba binarna bez znaku, to przesuwanie arytmetyczne realizuje się identycznie jak logiczne. W takim przypadku przesuwanie arytmetyczne odpowiada dzieleniu (przesuwanie w prawo) lub mnożeniu (w lewo) zawartości rejestru przez kolejne potęgi liczby 2; na przykład:

w prawo	
1100	12
0110	6
0011	3
0001	1

w lewo	
0011	3
0110	6
1100	12
1000	8

Operacje mnożenia i dzielenia nie zawsze wykonywane są dokładnie. Jest to skutkiem skończonej długości rejestru. Przesuwanie cykliczne polega na tym, że wyjście szeregowe rejestru Q_S połączone jest z wejściem szeregowym X_S.



a)

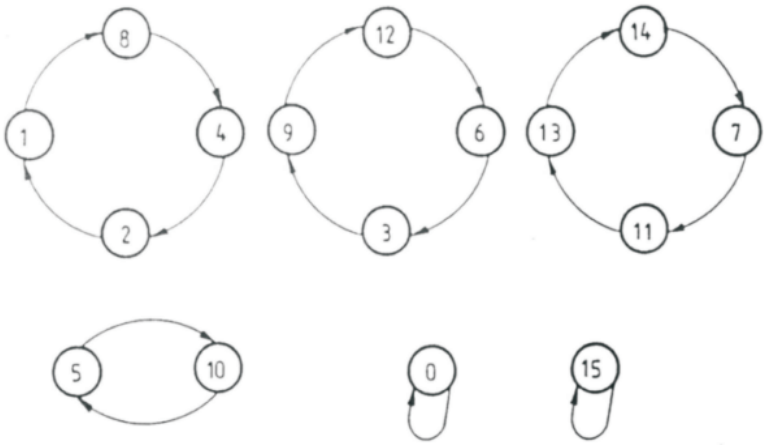
Rys. 4.4. Przesuwanie cykliczne: a) w prawo, b) w lewo

Przykład przesuwania cyklicznego:

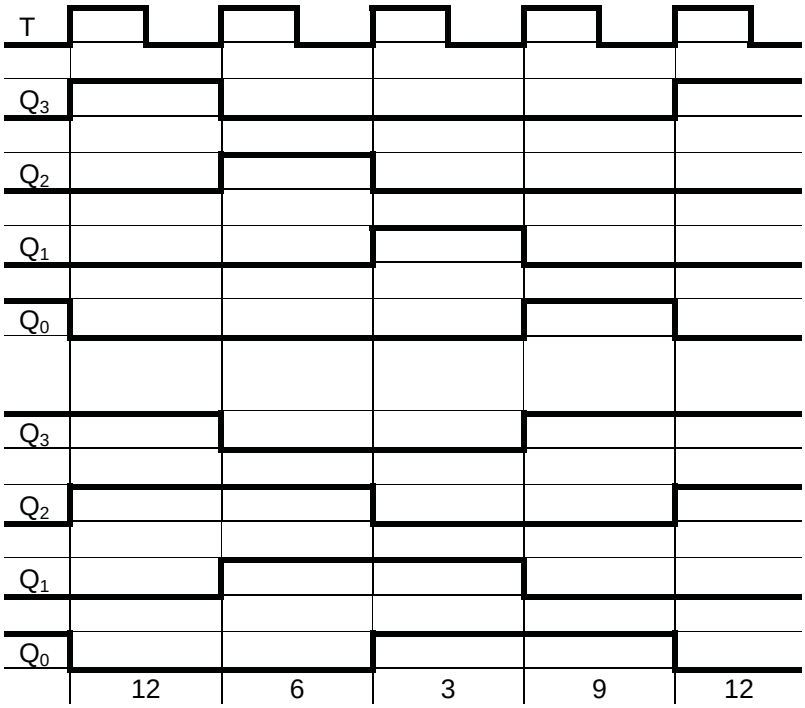
w prawo	
0110	6
0011	3
1001	9
1100	12
0110	6

w lewo	
0110	6
1100	12
1001	9
0011	3
0110	6

Kolejne stany rejestru najwygodniej jest przedstawiać w postaci grafu pracy rejestru. Przykład takiego grafu dla przesuwania cyklicznego w prawo przedstawiono na rys. 4.5. Należy zwrócić uwagę, że pełny graf opisujący pracę rejestru musi zawierać tyle stanów, ile wynosi pojemność rejestru (w tym przypadku 16). Inną metodą notowania pracy rejestru są przebiegi czasowe. Na rys. 4.6 przedstawiono przykładowe przebiegi czasowe w rejestrze czterobitowym podczas cyklicznego przesuwania w prawo. Należy podkreślić, że analiza przebiegów czasowych umożliwia często nietypowe zastosowanie rejestru, np. jako dzielnika częstotliwości, sterownika silnika skokowego czy generatora przebiegów trójfazowych.



Rys. 4.5. Przykład grafu pracy rejestru



Rys. 4.6. Przykładowe przebiegi czasowe w liczniku pierścieniowym

4.2.4. Rejestry liczące

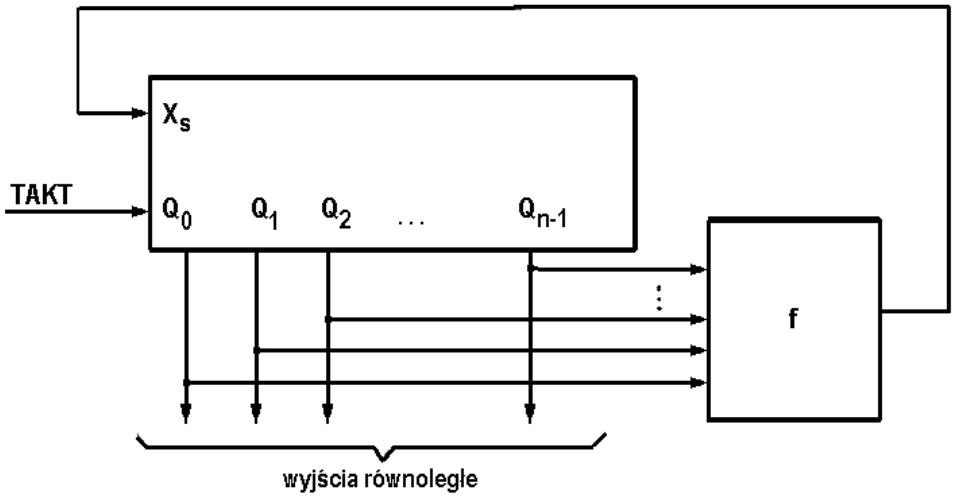
Rejestrem liczącym nazywamy rejestr przesuwający ze sprzężeniem zwrotnym, w którym sygnał podawany na wejście szeregowe X_S jest funkcją wyjść Q_i przerzutników rejestru

$$X_S = f(Q_0, Q_1, \ldots, Q_{n-1}) \tag{4.1}$$

Podział rejestrów liczących:

1) licznik pierścieniowy $X_S = Q_{n-1}$,
2) licznik Johnsona (pseudopierścieniowy) $X_S = \overline{Q_{n-1}}$,
3) rejestr liniowy $X_S = a_0Q_0 \oplus a_1Q_1 \oplus \ldots \oplus a_{n-1}Q_{n-1}$.

Ogólną strukturę rejestru liczącego przedstawiarys. 4.7.



Rys. 4.7. Ogólna struktura rejestru liczącego

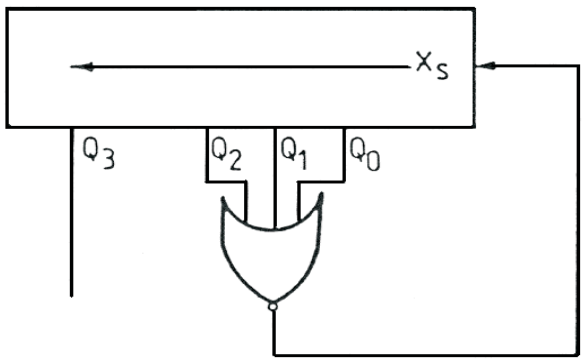
W liczniku pierścieniowym wyjście ostatniego przerzutnika jest podane na wejście pierwszego. W liczniku takim krąży najczęściej tylko jedno zero lub jedna jedynka (ustawione wcześniej), przesuwając się na sąsiednie wyjście po każdym impulsie zegarowym. Podczas przesuwania mogą pojawić się w rejestrze stany nieprawidłowe, które mogą być korygowane. Korekcję stanów nieprawidłowych zapewnia sprzężenie zwrotne typu

$$X_s = \overline{Q_{n-2} + ... + Q_1 + Q_0} \tag{4.2}$$

Przy takim sprzężeniu otrzymuje się licznik pierścieniowy samokorygujący. Sprzężenie takie zapewnia korekcję zbędnych jedynek, doprowadzając do tego, że w rejestrze krążyć będzie tylko jedna jedynka. Sprzężenie

$$X_s = \overline{Q_{n-2} \cdot Q_{n-3} + ... + Q_1 + Q_0} \tag{4.3}$$

zapewnia korekcję zbędnych zer, doprowadzając do tego, że w rejestrze krążyć będzie tylko jedno zero. Przykład rejestru samokorygującego podano na rys. 4.8.

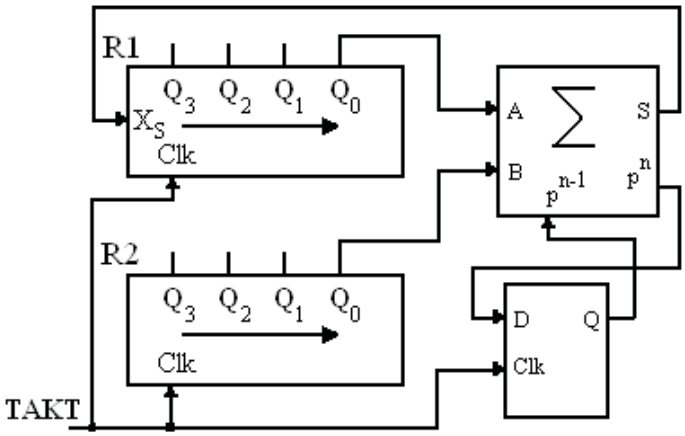


Rys. 4.8. Rejestr samokorygujący

Licznik Johnsona otrzymuje się, łącząc negację wyjścia ostatniego przerzutnika z wejściem pierwszego przerzutnika. Pojemność takiego licznika wynosi 2n, gdzie n jest liczbą przerzutników. Rejestr przesuwający ze sprzężeniem przez sumę modulo 2 nosi nazwę rejestru liniowego. Rejestr taki może pracować jako licznik o maksymalnej pojemności $2^n - 1$. Stan 00...00 nie jest prawidłowy w takim liczniku, ponieważ ze stanu tego nie na przejścia do innych stanów. Sprzężenie zwrotne w rejestrze liniowym nie musi wykorzystywać wszystkich wyjść rejestru. Uzyskanie maksymalnego cyklu pracy możliwe jest jedynie przy wykorzystaniu niektórych wyjść rejestru. Na przykład przy n = 4 należy podać $X_S = Q_3 \oplus Q_0$ lub $X_S = Q_3 \oplus Q_2$, aby uzyskać cykl pracy o 15 stanach ($2^4 - 1$).

4.2.5. Sumator szeregowy

Przykładem zastosowania rejestrów przesuwających jest sumator szeregowy. Jest to układ wykorzystujący dwa rejestry, sumator jednobitowy oraz przerzutnik w celu dodawania arytmetycznego dwóch liczb binarnych. Na rys. 4.9 przedstawiono schemat blokowy takiego sumatora.



Rys. 4.9. Struktura sumatora szeregowego

W rejestrach R1 i R2 znajdują się dwie liczby, które mają być dodane (składniki). Podczas dodawania zawartość tych rejestrów przesuwana jest w prawo, tzn. w kierunku najmniej znaczących bitów. Sumator jednobitowy dodaje w sensie arytmetycznym dwa bity składników. Wynik sumowania S jest pamiętany w rejestrze R1. Rejestr R1 pełni więc rolę akumulatora, tzn. pamiętany jest w nim zarówno składnik, jak i wynik dodawania. Przeniesienie powstałe w wyniku dodawania dwu bitów jest zapamiętywane w przerzutniku. Będzie ono wykorzystane podczas dodawania dwu kolejnych bitów składników. Funkcje pełnione przez sumator jednobitowy zebrano w tab. 4.1.

Tabela 4.1

Funkcje sumatora jednobitowego

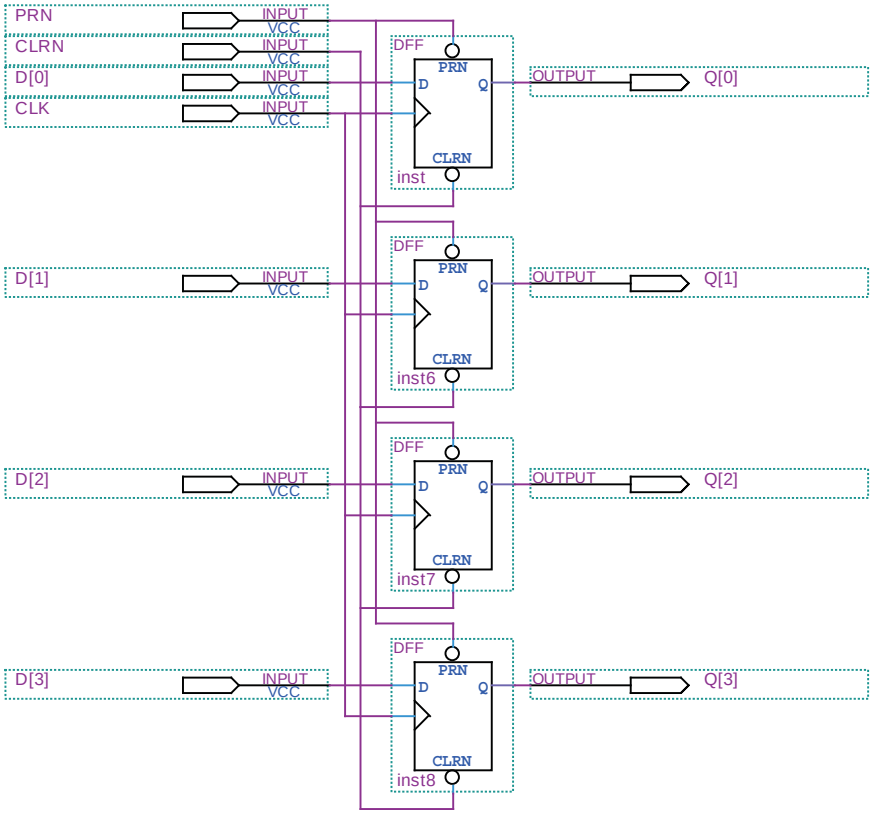
p^{n-1}	a	b	S	p^n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Stan przerzutnika pamiętającego przeniesienie może być ustalony za pomocą wejść ustawiających Pr i Cl. Po n taktach sygnału zegarowego (n – długość rejestrów) sumowanie jest zakończone. Jedną z podstawowych wad tego sumatora jest stosunkowo długi czas dodawania ograniczony maksymalną częstotliwością sygnału zegarowego.

4.3. Przygotowanie do zajęć

4.3.1. Rejestr równoległy 4-bitowy

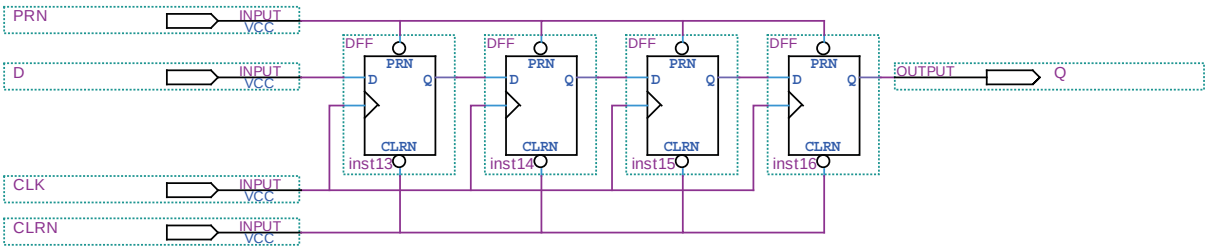
4-bitowy rejestr równoległy można zrealizować w strukturze programowalnego układu cyfrowego, bazując na przerzutnikach synchronicznych. Jednym z podstawowych jest przerzutnik DFF. Na rysunku poniższym przedstawiono przykładowe rozwiązanie rejestru równoległego. Dane wejściowe D[3...0] są zapisywane (i przepisywane na wyjście Q[3...0]) w rejestrze podczas narastającego zbocza sygnału CLK.



Rys. 4.10. Struktura 4-bitowego rejestru równoległego

4.3.2. Rejestr przesuwny 4-bitowy

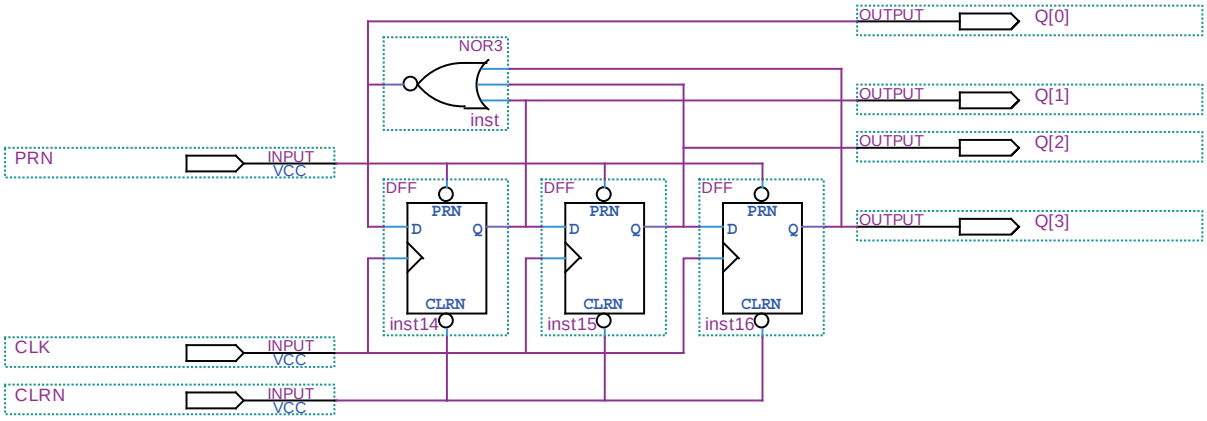
Rejestr przesuwny również może być zrealizowany z wykorzystaniem przerzutników DFF, które są połączone szeregowo. Informacja podana na wejście D dociera na wyjście Q po czterech taktach zegara CLK. Stąd też rejestr przesuwny może być wykorzystany jako linia opóźniająca zbocze sygnału lub pamięć próbek. Bazując na rejestrach przesuwnych, można również zrealizować pamięci szeregowe lub automaty. Jedną z takich realizacji jest rejestr samokorygujący opisany krótko w następnym podpunkcie.



Rys. 4.11. Struktura 4-bitowego rejestru przesuwnego

4.3.3. Rejestr samokorygujący

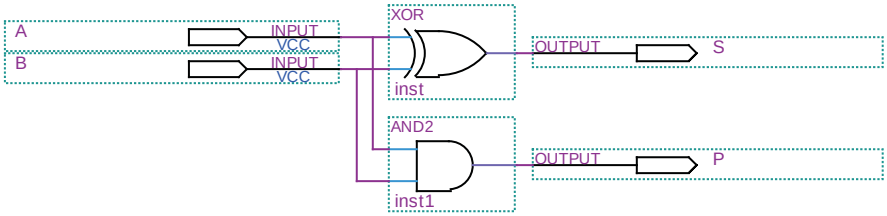
Rejestr samokorygujący to przykład prostego automatu, którego zadaniem jest generacja oczekiwanego ciągu impulsów. Jednym z przykładów może być tzw. 4-bitowy rejestr z krążącą jedynką z poniższego rysunku.



Rys. 4.12. Struktura 4-bitowego rejestru samokorygującego z krążącą jedynką

4.3.4. Półsumator jednobitowy

Aby zrealizować sumator szeregowy, należy zaprojektować blok sumujący, który może wykorzystywać jedynie dwie bramki; bramkę XOR oraz bramkę AND2.



Rys. 4.13. Struktura 4-bitowego rejestru samokorygującego z krążącą jedynką

4.3.5. Analiza i symulacja pamięci RAM zbudowanej z przerzutników DFFE

Wszystkie niezbędne pliki znajdują się w spakowanym projekcie „rejestry.zip”, który można pobrać ze strony katedralnej wraz z instrukcją. W niniejszym punkcie znajdują się opisy głównych elementów budowy pamięci.

Przerzutnik DFFE jako podstawowa komórka pamięci

Pojedynczy przerzutnik DFFE jest elementem pamiętającym 1 bit. Aby zapamiętać 8 bitów, należy użyć 8 przerzutników typu DFFE. W porównaniu z przerzutnikiem DFF, przerzutnik DFFE posiada dodatkowe wejście aktywujące „E” (ang. *enable*). Należy otworzyć plik „rejestry_punkt_3_1_1.bdf”, zapoznać się z jego budową oraz przeanalizować jego działanie na podstawie pliku „rejestry_punkt_3_1_1.vwf”.

Blok LPM_DECODE jako dekoder komórek pamięci

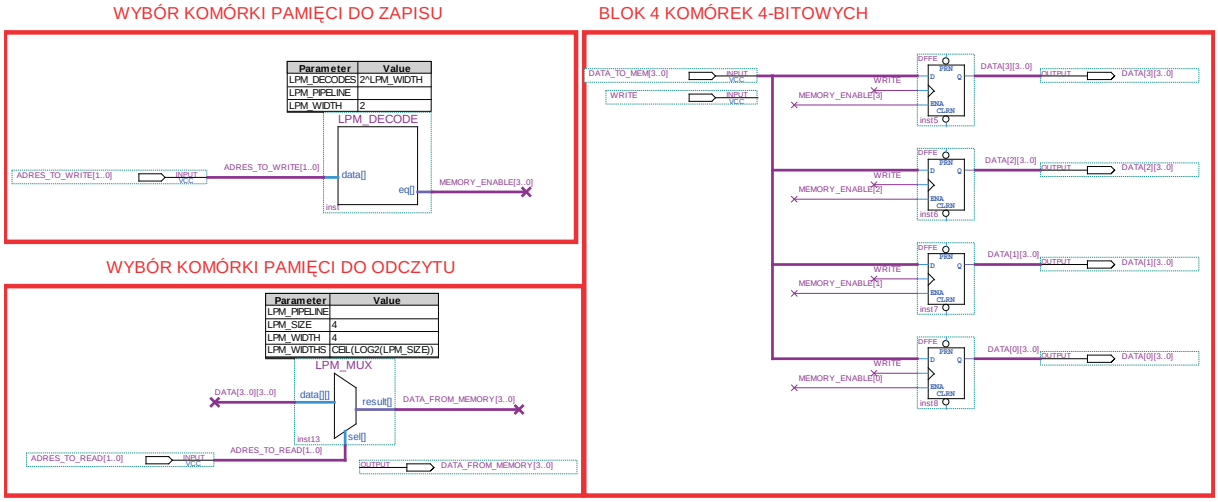
Pamięć RAM składa się z komórek. Jeśli chcemy zapisać wybraną komórkę, należy ją aktywować. Blok LPM_DECODE na podstawie adresu generuje sygnał aktywacji dla konkretnej grupy przerzutników DFFE reprezentujących komórkę. W omawianym przykładzie komórka jest 4-bitowa, a więc posiada 4 przerzutniki DFFE. Należy otworzyć plik „rejestry_punkt_3_1_2.bdf”, zapoznać się z jego budową oraz przeanalizować jego działanie na podstawie pliku „rejestry_punkt_3_1_2.vwf”.

Blok LPM_MUX jako multiplekser danych wyjściowych pamięci

W przypadku odczytu wcześniej zapisanych danych należy podać adres komórki, której zawartość ma się pojawić na szynie wyjściowej (w naszym omawianym przypadku szerokość szyny wynosi 4 bity). Blok LPM_MUX jest multiplekserem, który łączy zawartość określonej komórki z wyjściem pamięci na podstawie podanego adresu. Jeśli pamięć zawiera 4 komórki to szerokość adresu wynosi 2 bity, jeśli natomiast pamięć jest zbudowana z 16 komórek wymagana jest 4-bitowa szyna adresowa itd. Należy otworzyć plik „rejestry_punkt_3_1_3.bdf", zapoznać się z jego budową oraz przeanalizować jego działanie na podstawie pliku „rejestry_punkt_3_1_3.vwf".

Przykładowa realizacja pamięci 4-bitowej o rozmiarze 4 komórek

W poprzednich podpunktach omówione zostały główne elementy składowe bloku pamięci. Należy otworzyć plik „rejestry_punkt_3_1_4.bdf" i przeanalizować współdziałanie elementów składowych pamięci: multipleksera LPM_MUX oraz dekodera LPM_DECODE. Następnie należy sprawdzić poprawność działania proponowanej struktury na podstawie symulacji „rejestry_punkt_3_1_4.vwf". Struktura przykładowej pamięci przestawiona jest na rys. 4.14.



Rys. 4.14. Struktura 4-bitowej pamięci o rozmiarze 4 komórek

4.3.6. Analiza i symulacja przykładowej realizacji bloku pamięci dla zestawu DE2

Należy zapoznać się ze strukturą przedstawioną w pliku „rejestry_punkt_3_2.bdf". Następnie **należy przygotować symulację**, w której zostaną przedstawione przykładowe sekwencje zapisu do pamięci (na przykład 4 zapisy do każdej z 4 komórek) oraz odczytu z pamięci (na przykład odczyt z 1 oraz 2 komórki pamięci).

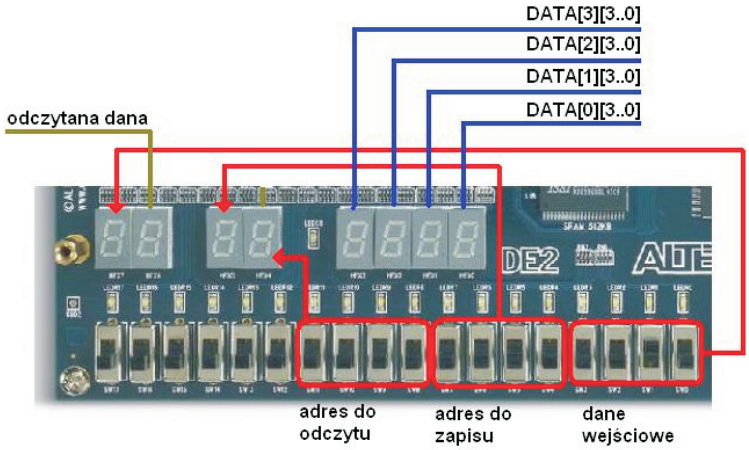
4.4. Przebieg ćwiczenia

Najczęściej rejestry wykorzystuje się do zapamiętania informacji. Bazując na rejestrach, można zaprojektować różne struktury pamięciowe, co jest tematem ćwiczeń praktycznych.

4.4.1. Przykładowa realizacja pamięci 4-bitowej o rozmiarze 4 komórek

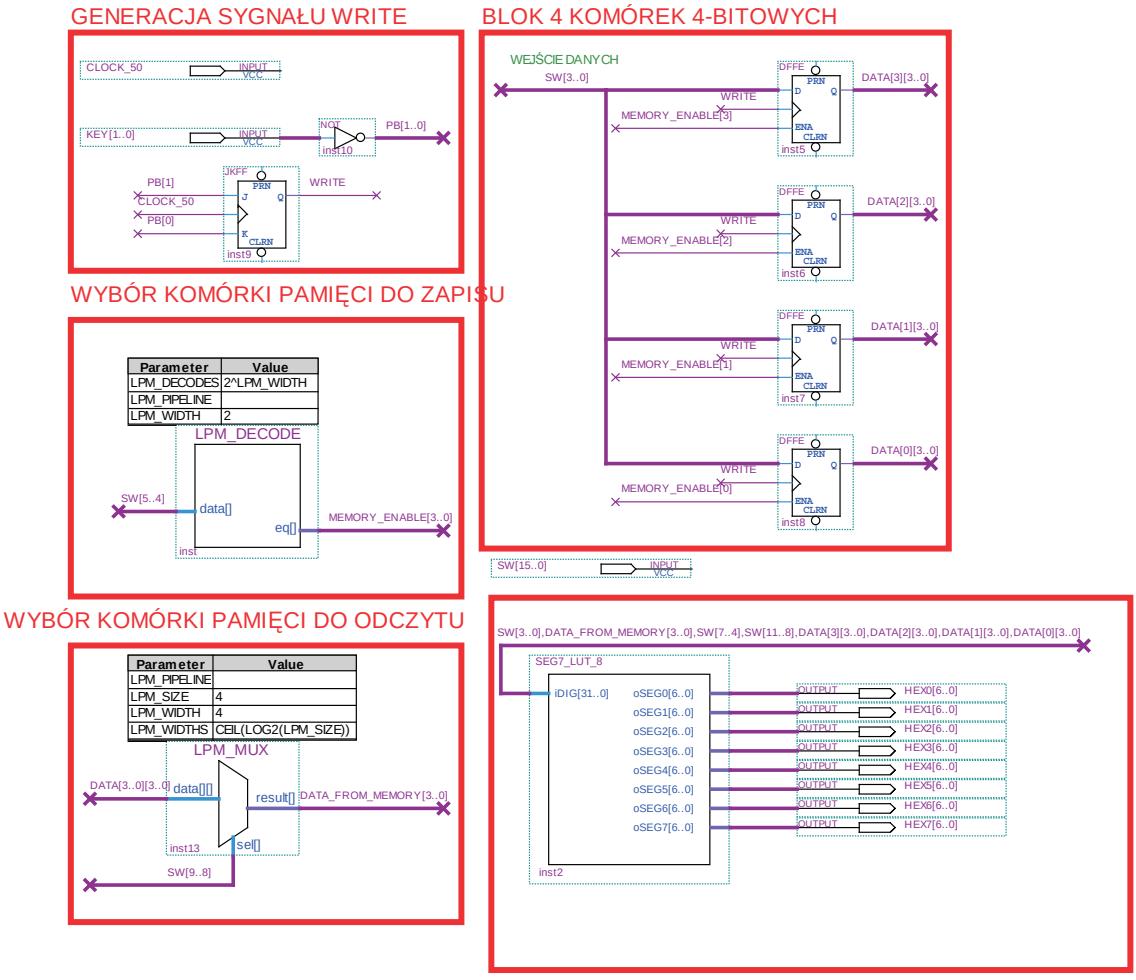
Adresy komórek pamięci do odczytu i zapisu ustala się przez odpowiednie położenie przełączników SW. Dodatkowo dla poprawy wizualizacji wprowadzono blok SEG7_LUT8,

który współpracuje z wszystkimi wskaźnikami segmentowymi HEX7 do HEX0. Rysunek 4.16 przedstawia strukturę pamięci. Znaczenie wskaźników ilustruje rys. 4.15.



Rys. 4.15. Funkcje przełączników oraz wskaźników siedmiosegmentowych

Należy przeprowadzić kilka testów polegających na wprowadzaniu danych do pamięci oraz ich odczytywaniu. Punkt zostaje uznany za ukończony, gdy pod adresem 0x0 będzie przechowywana wartość 0xA, a w pozostałych komórkach odpowiednio 0xB, 0xC i 0xD.



Rys. 4.16. Proponowana struktura pamięci o organizacji 4×4

4.4.2. Pamięć o organizacji 16×8

Zmodyfikować strukturę z poprzedniego punktu tak, by uzyskać pamięć o pojemności 16 słów 8-bitowych. Dodatkowo należy przeprojektować fragment struktury cyfrowej dla wskaźników tak, by na wskaźnikach były wyświetlane najważniejsze informacje dotyczące adresów oraz danych. Ćwiczenie zostaje uznane za ukończone, gdy w pamięci będzie przechowywany ciąg danych wskazany przez prowadzącego.

4.5. Pytania kontrolne

1. Podać podstawowe rodzaje rejestrów oraz ich parametry.
2. Co to są rejestry przesuwającej? Jakie rodzaje przesuwania realizują?
3. Podać strukturę rejestru liczącego oraz wymienić podstawowe ich rodzaje.
4. W jaki sposób zrealizować sumator szeregowy?
5. Jaką funkcję w pamięci pełni multiplekser, a jaką dekoder?

4.6. Literatura

[1] Haras A., Nieznański J: *Komputery i programowanie II. Materiały pomocnicze do laboratorium.* Gdańsk: Wyd. Polit. Gdańskiej 1990.

5. Liczniki i dzielniki częstotliwości

5.1. Cel i zakres ćwiczenia

Celem ćwiczenia jest utrwalenie i uzupełnienie wiedzy na temat liczników i dzielników częstotliwości oraz zapoznanie się z najczęściej stosowanymi rozwiązaniami praktycznymi tych układów.

5.2. Wprowadzenie

Licznikiem nazywa się układ cyfrowy służący do zliczania impulsów i pamiętania ich liczby. Każdy impuls wejściowy powoduję zmianę stanu licznika. Stan licznika utrzymuje się aż do chwili pojawienia się kolejnego impulsu. Liczba różnych stanów licznika P nazywana jest pojemnością (okresem, długością cyklu) licznika. Licznik o pojemności P nazywany jest licznikiem modulo P. Pojemność licznika jest zależna od liczby przerzutników, z których składa się licznik. Liczba przerzutników s potrzebna do zbudowania licznika o pojemności P musi spełniać warunek:

$$2^s \geq P \tag{5.1}$$

Liczbę s nazywa się długością lub liczbą bitów licznika. Zazwyczaj jeden spośród P stanów licznika jest przyjęty jako początkowy (zerowy) i może być ustawiany za pomocą specjalnego wejścia zerującego, działającego niezależnie od aktualnej zawartości licznika.

Ze względu na sposób doprowadzania zliczanych impulsów do wejść poszczególnych przerzutników rozróżnia się trzy typy liczników:

— asynchroniczne (szeregowe),
— synchroniczne (równoległe),
— asynchroniczno-synchroniczne (szeregowo-równoległe).

W licznikach asynchronicznych impulsy są podawane tylko na wejście pierwszego przerzutnika. Wyjście pierwszego przerzutnika steruje drugim przerzutnikiem itd. Zmiana stanu każdego członu następuje więc dopiero po zmianie poprzedniego członu, a nie synchronicznie z sygnałem wejściowym. Ostatni człon otrzymuje impulsy wejściowe opóźnione o sumę czasów przełączania wszystkich poprzednich członów.

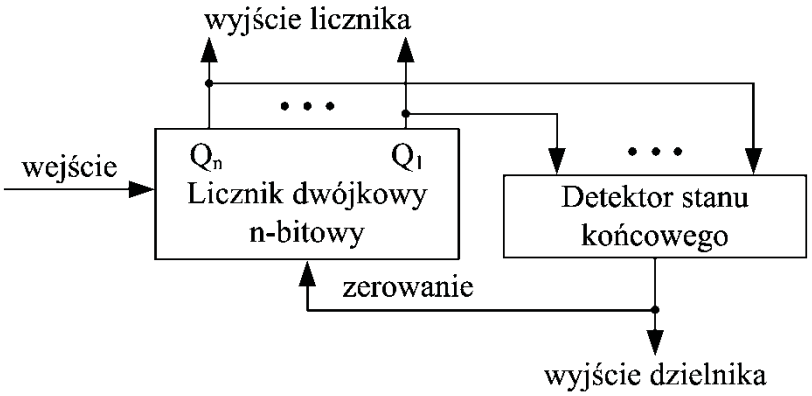
W licznikach synchronicznych sygnał wejściowy podawany jest równocześnie na wszystkie stopnie. Zmiana stanu danego stopnia następuje wówczas, gdy wszystkie poprzednie stopnie znajdują się w stanach końcowych. Informacja o tym, że dany stopień znajduje się w stanie końcowym jest przesyłana do następnych stopni w postaci sygnału przeniesienia za pośrednictwem dodatkowego układu kombinacyjnego. Zaletą liczników synchronicznych jest większa szybkość działania wynikająca z faktu, że zmiana stanu wszystkich przerzutników następuje jednocześnie (z dokładnością do czasów propagacji przeniesień przez układ kombinacyjny). Wadą liczników synchronicznych jest większa złożoność wynikająca z konieczności generowania przeniesień.

Układy asynchroniczno-synchroniczne są wykorzystywane w przypadku liczników o dużej pojemności (większej od 16), budowanych z liczników scalonych. Zazwyczaj łączy się szeregowo kilka scalonych liczników synchronicznych.

Każdemu stanowi licznika można przyporządkować liczbę odpowiadającą kombinacji zer i jedynek logicznych na wyjściach przerzutników. Jeśli liczby te są reprezentacjami naturalnego kodu dwójkowego, to licznik nazywamy dwójkowym lub binarnym. Jeśli natomiast reprezentują one wybrany kod dwójkowo-dziesiętny to licznik nazywamy dziesiętnym lub dekadą. W układach sterowania binarnego stosuje się inne kody pracy liczników, np. kod 1 z *n* (licznik pierścieniowy), kod Johnsona (licznik Johnsona) i inne.

Liczniki dwójkowe i dziesiętne mogą zliczać w przód (w górę), tzn. każdy kolejny impuls zwiększa o 1 liczbę reprezentującą stan licznika lub wstecz (w dół) – liczba odpowiadająca stanowi licznika jest zmniejszana o jeden po każdym impulsie. Liczniki zliczające tylko w przód lub tylko wstecz nazywane są licznikami jednokierunkowymi. Stosowane są również tzw. liczniki rewersyjne (dwukierunkowe), zliczające w przód lub wstecz w zależności od sygnału określającego kierunek zliczania.

Do syntezy liczników synchronicznych można wykorzystać dobrze opracowane metody algebraiczne. Przy użyciu tych metod można przeprowadzić syntezę licznika działającego zgodnie z dowolnym grafem zmian stanu. Dla liczników asynchronicznych (i układów asynchronicznych w ogóle) takiej ogólnej metody, jak dotąd, nie opracowano. Istnieje jednak praktyczna metoda pozwalająca na uzyskanie dowolnej pojemności licznika asynchronicznego (a także synchronicznego) przez skrócenie cyklu licznika dwójkowego o odpowiedniej długości. Metoda ta polega na detekcji żądanego stanu końcowego i zerowaniu licznika po wykryciu tego stanu (rys. 5.1).

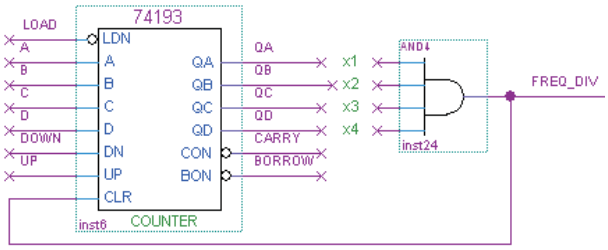


Rys. 5.1. Licznik/dzielnik częstotliwości z detekcją stanu końcowego i zerowaniem

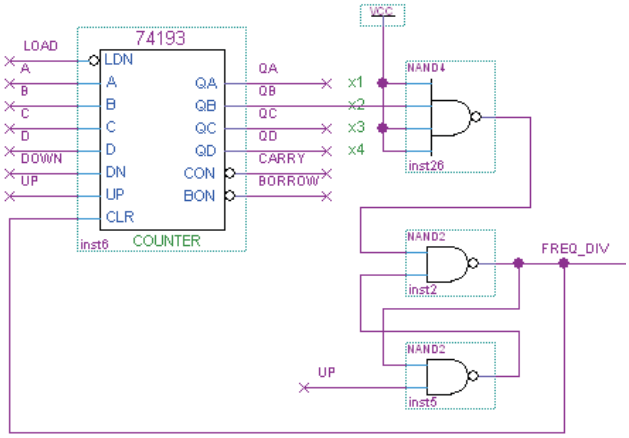
Metodę tą można stosować w odniesieniu do dowolnego licznika, także w wersji scalonej. Jeśli długość cyklu licznika ma być równa *P*, to detektor powinien wykrywać wystąpienie na wyjściach licznika stanu reprezentującego liczbę *P* w kodzie binarnym. Wykrycie stanu *P* powinno spowodować wyzerowanie licznika. Następuje to w tym samym takcie zegara, w którym licznik osiąga stan *P*, a więc w jednym takcie zegara na wyjściu licznika można obserwować dwa różne stany. Innymi słowy, w ciągu P taktów zegara licznik przyjmuje $P + 1$ stanów (0, 1, ..., *P*), z których jeden, stan *P*, należy traktować jak stan pasożytniczy. Jego obecność jest niezbędna dla właściwego działania układu, lecz niepożądana (w idealnym liczniku każdemu taktowi zegara odpowiada tylko jeden stan). Czas trwania tego stanu jest dla układów scalonych rzędu kilkunastu, kilkudziesięciu nanosekund i jest to suma czasu propagacji stanu *P* przez detektor stanu końcowego (wytworzenie impulsu zerującego) oraz czasu propagacji impulsu zerującego do wyjść licznika (zerowanie licznika). Na rys. 5.2a przedstawiono przykładowy schemat (z programu Quartus II) umożliwiający uzyskanie dowolnych (programowanych na podstawie tablicy z rys. 5.2rys. c) skróconych cykli licznika 74193. Jeśli

proces zerowania licznika przebiega nierównomiernie, tzn. niektóre przerzutniki w liczniku są zerowane wcześniej niż pozostałe, to w procesie przejściowym związanym z zerowaniem licznika mogą pojawić się także inne niż *P*, niepożądane stany. Należy zwrócić uwagę na fakt, że z chwilą gdy stan *P* przechodzi w inny stan na skutek działania impulsu zerującego, impuls ten przestaje być wytwarzany przez detektor, co może prowadzić do niepełnego wyzerowania licznika. Wystąpienie takiej sytuacji oznaczałoby poważne zakłócenie cyklu pracy licznika. Z tego względu często stosuje się środki zapewniające dostatecznie długi czas trwania impulsu zerującego. Jeden z popularnych sposobów przedstawiono na rys. 5.2b (działanie tego układu należy samodzielnie przeanalizować przed przystąpieniem do ćwiczenia).

a)



b)



c)

x4	x3	x2	x1	pojemność licznika / współczynnik podziału dzielnika
r	r	r	r	–
	r	r	z	1
r	r	z	r	2
r	r	z	z	3
r	z	r	r	4
r	z	r	z	5
r	z	z	r	6
r	z	z	z	7
z	r	r	r	8
z	r	r	z	
z	r	z	r	10
z	r	z	z	11
z	z	r	r	12
z	z	r	z	13
z	z	z	r	14
z	z	z	z	15

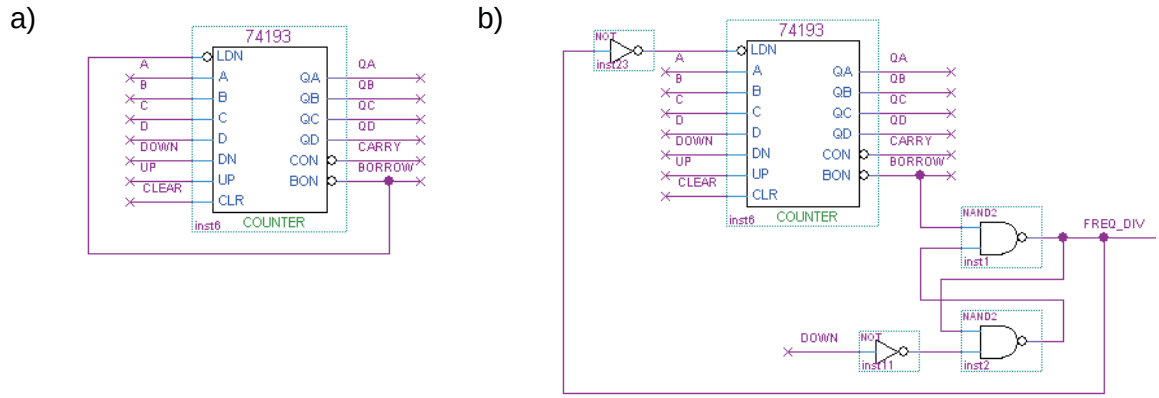
Rys. 5.2. Licznik/dzielnik nastawny z wykorzystaniem licznika scalonego 74193 (a); zastosowanie przerzutnika RS w celu wydłużenia impulsu zerującego licznika (b); tablica ilustrująca programowanie układu (r – rozwarcie, z – zwarcie) (c)

Dzielnikiem częstotliwości nazywany jest układ, na którego wyjściu otrzymuje się jeden impuls co *p* impulsów wejściowych, czyli częstotliwość wyjściowa dzielnika jest *p*–krotnie mniejsza od częstotliwości wejściowej. W odróżnieniu od liczników, dzielnik częstotliwości ma tylko jedno wyjście. Oczywiście każdy licznik może pełnić funkcję dzielnika częstotliwości, natomiast odwrotne twierdzenie nie jest prawdziwe. Sposób kodowania stanów nie jest w przypadku dzielnika częstotliwości istotny, co stwarza niekiedy możliwość uproszczenia struktury dzielnika.

Synchroniczne dzielniki częstotliwości można projektować przy użyciu tych samych metod algebraicznych, co w przypadku liczników synchronicznych. Dzielniki asynchroniczne można projektować metodą rozkładu współczynnika podziału *p* na czynniki [5]. Metoda rozkładu na czynniki jest nieekonomiczna i mało uniwersalna (nieprzydatna do syntezy dzielników synchronicznych i dzielników budowanych z liczników scalonych). Dlatego znacznie

częściej realizuje się dzielniki częstotliwości, wykorzystując liczniki (zazwyczaj scalone) z układem detekcji żądanego stanu końcowego i zerowaniem (rys. 5.1 i rys. 5.2). Wyjściem dzielnika jest w tym przypadku wyjście detektora stanu końcowego *P*, gdzie uzyskuje się jeden impuls zerujący na każde *P* taktów zegara (podział częstotliwości wejściowej przez *P*). Ponieważ czas trwania impulsu zerującego jest bardzo krótki i trudny do precyzyjnego określenia, a niezawodność całego układu może budzić pewne wątpliwości, często stosuje się wspomnianą już wersję układu z rys. 5.2b.

Dla uzyskania odpowiednio dużych pojemności liczników (współczynników podziału dzielników) budowanych z liczników scalonych często niezbędne jest kaskadowe łączenie kilku mikroukładów. W przypadku scalonych liczników asynchronicznych zazwyczaj łączy się najbardziej znaczące wyjście młodszego stopnia z wejściem starszego stopnia. Scalone liczniki synchroniczne posiadają zwykle specjalne wyprowadzenia służące do łączenia tych układów w bloki o większej pojemności (w ogólnym przypadku dla prawidłowego wykorzystania tych wyprowadzeń konieczne jest odwołanie się do literatury pomocniczej, np.: not aplikacyjnych zamieszczanych na stronach internetowych producentów). Liczniki zliczające wprzód mają zazwyczaj tzw. wyjście przeniesienia, na którym poziomem aktywnym, zwykle niskim, sygnalizowane jest osiągnięcie przez licznik stanu 11...1. Liczniki zliczające wstecz wyposażone są w wyjście pożyczki sygnalizujące osiągnięcie przez licznik stanu 00...0.



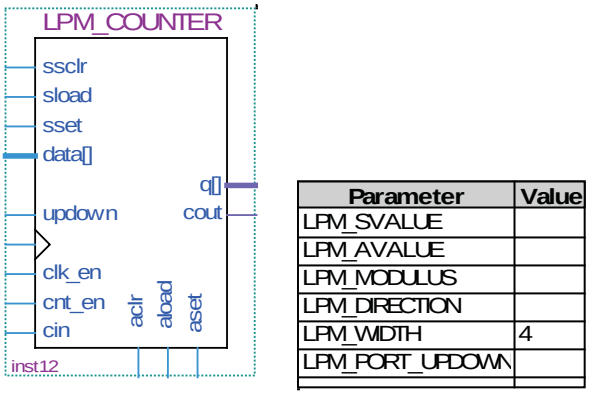
Rys. 5.3. Licznik/dzielnik o programowalnej długości cyklu z wykorzystaniem licznika 74193 (a); zastosowanie przerzutnika RS dla wyeliminowania wyścigów czasowych podczas ustawiania wejść (b)

Scalone liczniki synchroniczne wyposażone są często w tzw. wejścia ustawiające, umożliwiające ustalenie dowolnego stanu początkowego licznika. Stwarza to możliwość łatwej realizacji liczników o programowanej długości cyklu, szczególnie w przypadku liczników zliczających wstecz. Wyjście pożyczki można połączyć z wejściem wpisującym, tj. wejściem uaktywniającym ustawienie licznika, tak aby wystąpienie stanu 00...0 powodowało przepisanie do licznika stanu wejść ustawiających. Długość cyklu takiego licznika jest równa liczbie *P* reprezentowanej przez stan wejść ustawiających, względnie *P* + 1 zależnie od zastosowanego w wykorzystywanym liczniku scalonym sposobu ustawiania licznika (asynchroniczny lub synchroniczny) oraz od sposobu połączenia wyjścia pożyczki z wejściem wpisującym. Przykład licznika/dzielnika o programowanej długości cyklu pracującego według omawianej zasady przedstawiono na rys. 5.3a (wyjściem dzielnika jest wyjście pożyczki – BON). Zastosowany w układzie licznik scalony 74193 charakteryzuje się ustawianiem asynchronicznym. W przypadku połączenia wyjścia pożyczki układu 74193 z wejściem wpisującym, jak na rys. 5.3a, długość cyklu licznika jest równa *P*. Liczba stanów, jakie można zaobserwować na wyjściu, jest jednak równa *P* + 1 (dwa stany: 0 i *P* pojawiają się w tym samym takcie zegara). Stan *P* należy uznać za niepożądany, choć jego obecność, podobnie jak w poprzednim rozpatrywanym układzie z rys. 5.1, warunkuje prawidłowe działanie układu. Ponadto, analogicznie

jak w układzie z rys. 5.2a, istnieje w rozważanym układzie niebezpieczeństwo wystąpienia zakłóceń cyklu pracy licznika w przypadkach dużych różnic w czasach propagacji sygnałów od wejść ustawiających do wyjść poszczególnych przerzutników. Podobnie czas trwania impulsu wyjściowego dzielnika jest bardzo krótki, rzędu kilkudziesięciu nanosekund. W wielu przypadkach dla zwiększenia niezawodności działania układu należy zastosować środki wydłużające impuls wpisujący (rys. 5.3b).

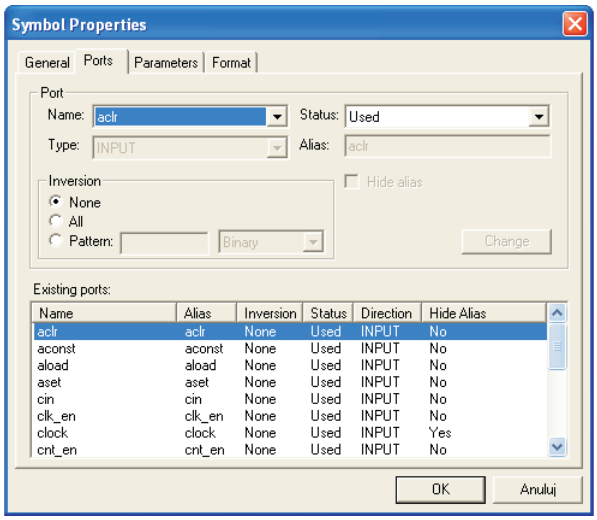
5.3. Opis modułu *LPM_COUNTER*

Moduł *LPM_COUNTER* (z biblioteki *megafunctions/arithmetic*) jest uniwersalnym licznikiem, który znajduje zastosowanie w wielu aplikacjach: liczniki impulsów, pomiar szerokości pulsu, pomiar liczby taktów zegara między impulsami, generacja ciągu impulsów w określonej sekwencji, opóźnienie sygnałów, generacja przebiegów nośnych i modulujących w technice modulacji szerokości impulsów (PWM). Moduł *LPM_COUNTER* występuje niemalże w każdym projekcie struktury cyfrowej realizowanej na bazie programowalnych układów logicznych firmy ALTERA. Symbol modułu pokazany jest na rys. 5.4.



Rys. 5.4. Symbol modułu *LPM_COUNTER* z aktywną tabelą parametrów i wszystkimi możliwymi wyprowadzeniami (wejścia oraz wyścia)

Konfigurację licznika, czyli wybór określonych wejść i wyjść oraz nadanie odpowiedniej wartości parametrom licznika odbywa się w oknie *Symbol Properties* (rys. 5.5).



Rys. 5.5. Okno edycji i konfiguracji modułu LPM_COUNTER

Okno pojawia się po dwukrotnym kliknięciu lewym przyciskiem myszy na tabelę parametrów (rys. 5.4). Posiada ono cztery zakładki, najważniejsze z nich to zakładka *Ports* oraz zakładka *Parameters*. W zakładce *Ports* określamy, które wyprowadzenie będzie przez nas wykorzystywane („*used*"), a które będzie nieaktywne („*unused*"). Możliwa jest również negacja poszczególnych sygnałów („*inversion*") lub ustawienie stałych poziomów („*pattern*"). Po zakończeniu edycji i zamknięciu okna *Symbol Properties* na symbolu *LMP_COUNTER* powinny być widoczne tylko wybrane wyprowadzenia. Zakładka *Parameters* służy do ustawiania poszczególnych parametrów licznika. Po wybraniu wskaźnikiem parametru z przewijanej listy, jego nazwa pojawia się w polu „*Parametr Name*". Zmieniają się również wpisy w polach „*Setting*", „*Type*" oraz „*Description*". Użytkownik może wpisać nową wartość w polu bądź też skorzystać z podpowiedzi – listy, która pojawia się po rozwinięciu danego pola. Szczegółowy opis wejść zamieszczony jest w tab. 5.1, wyjścia scharakteryzowane są w tab. 5.2, natomiast parametry opisano w tab. 5.3. Uproszczoną tablicę prawdy licznika zamieszczono w tab. 5.4.

Tabela 5.1

Wyprowadzenia wejściowe modułu *LPM_COUNTER*

Port	Wymagany	Opis
sclr	NIE	Synchroniczne kasowanie licznika przy pierwszym zboczu narastającym sygnału zegarowego. Domyślnie „0" (nieaktywne).
sload	NIE	Wejście synchronicznego ładowania licznika wartością z wejścia *data*[] przy pierwszym narastającym zboczu sygnału zegarowego. Domyślnie „0" (nieaktywne).
sset	NIE	Synchroniczne ustawianie. Ustawia licznik przy pierwszym zboczu narastającym sygnału taktującego *clock*. Domyślnie „0" (nieaktywne). Ustawia na wyjściu *q*[] wszystkie jedynki lub wartość podaną jako parametr *LPM_SVALUE*. Jeśli oba wejścia *sset* i *sclr* są wybrane i pojawią się równocześnie większy priorytet posiada *sclr*.
data[]	NIE	Równoległe wejście danych o szerokości (liczbie bitów) wynikającej z wartości parametru *LPM_WIDTH*. Szyna danych jest wykorzystywana z sygnałami *aload* lub *sload*.
updown	NIE	Kierunek zliczania. „1" – zliczanie w górę, „0" – zliczanie w dół. Jeśli parametr *LPM_DIRECTION* jest ustawiony na stały kierunek zliczania wówczas nie należy używać portu.
clock	TAK	Sygnał taktujący licznik. Zazwyczaj źródłem sygnału taktującego jest generator kwarcowy umieszczony na płytce razem z układem PLD.
clk_en	NIE	Aktywacja działania licznika w trybie synchronicznym. Domyślnie „1" (aktywny).
cnt_en	NIE	Wejście blokowania (0) i aktywacji (1) zliczania bez wpływu na *sload*, *sset* oraz *sclr*. Domyślnie „1".
cin	NIE	Wejście przepełnienia. Stosuje się w kaskadowym (szeregowym) połączeniu liczników. Dla licznika liczącego w przód zachowanie *cin* jest identyczne jak *cnt_en*. Domyślnie „1" (aktwny).
aclr	NIE	Asynchroniczne kasowanie licznika; *aclr* ma wyższy priorytet od *aset* jeśli oba wejścia są wybrane.
aload	NIE	Asynchroniczne ładowanie licznika wartością z portu *data*[].
aset	NIE	Domyslnie „0" (nieaktywne). Ustawia wyjscia *q*[] na jedynki lub na wartość podaną przez parametr *LPM_AVALUE*.

Tabela 5.2

Wyprowadzenia wyjściowe modułu *LPM_COUNTER*

Port	Wymagany	Opis
q[]	NIE	Wyjście licznika. Szerokość szyny wyjściowej zależna jest od wartości parametru LPM_WIDTH.
eq[15..0]	NIE	(Tylko w AHDL od wersji 9.1 programu Quartus). Wyjście dekodera stanów licznika – 1 z N. Z każdym taktem zliczania ustawiany jest tylko jeden bit magistrali *eq*[], zatem bity ustawiane są tylko przez 16 stanów na początku zliczania lub na końcu zliczania, jeśli licznik zlicza w dół (przesuwająca się jedynka). Szyna jest szczególnie przydatna w tworzeniu generatorów sekwencji.
cout	NIE	Wyjście przepełnienia.

Tabela 5.3

Parametry modułu *LPM_COUNTER*

Parametr	Wymagany	Opis
LPM_SVALUE	NIE	Stała wartość ładowana podczas narastającego zbocza sygnału taktującego gdy wejście *sset* jest w stanie wysokim.
LPM_AVALUE	NIE	Stała wartość ładowana asynchronicznie gdy sygnał aset jest aktywny.
LPM_MODULUS	NIE	Maksymalna wartość na wyjściu licznika powiększona o jeden. Parametr pozwala na tworzenie liczników o mniejszej pojemności niż wynika to z długości licznika.
LPM_DIRECTION	NIE	Kierunek zliczania: „*UP*” (w górę – wartość domyślna), „*DOWN*” (w dół), „*UNUSED*” (parametr nieużywany i wtedy kierunkiem zliczania steruje wejście *updown*).
LPM_WIDTH	TAK	Liczba bitów licznika lub inaczej szerokość magistrali wyjściowej *q*[] oraz wejściowej *data*[].
LPM_PORT_UPDOWN	NIE	Określenie formy sterowania kierunkiem zliczania (w górę lub w dół). Jeśli użytkownik decyduje się na sterowanie wyprowadzeniem *updown* powinien wybrać wartość „*PORT_USED*”.

Tabela 5.4

Tabela prawdy licznika *LPM_COUNTER*

Wejścia										Wyjścia	Funkcja
aclr	*aset*	*aload*	*clk_en*	*clock*	*sclr*	*sset*	*sload*	*cnt_en*	*updown*	*q*[*LPM_WIDTH*-1..0]	
1	x	x	x	x	x	x	x	x	x	000...	Asynchroniczne kasowanie licznika
0	1	x	x	x	x	x	x	x	x	111...	Asynchroniczne ustawienie licznika
0	1	x	x	x	x	x	x	x	x	*LPM_AVALUE*	Asynchroniczne ustawienie licznika wartością LPM_AVALUE
0	0	1	x	x	x	x	x	x	x	*data*[]	Asynchroniczne załadowanie licznika wartością z portu data[]
0	0	0	0	x	x	x	x	x	x	*q*[]	Podtrzymanie wartości licznika
0	0	0	1	⌜	1	x	x	x	x	000...	Synchroniczne kasowanie licznika
0	0	0	1	⌜	0	1	x	x	x	111...	Synchroniczne ustawienie licznika
0	0	0	1	⌜	0	1	x	x	x	*LPM_SVALUE*	Synchroniczne ustawienie licznika wartością LPM_SVALUE
0	0	0	1	⌜	0	0	0	0	0	*q*[]	Zatrzymanie zliczania i podtrzymanie ostatniej wartości licznika
0	0	0	1	⌜	0	0	1	x	x	*data*[]	Synchroniczne załadowanie licznika wartością z portu data[]
0	0	0	1	⌜	0	0	0	1	1	*q*[]+1	Zliczanie w górę
0	0	0	1	⌜	0	0	0	1	0	*q*[]-1	Zliczanie w dół

5.4. Przygotowanie do zajęć

W trakcie ćwiczenia wykorzystywany będzie licznik *lpm_counter* z biblioteki *maga-functions/arithmetic* programu *Quartus II*. Symulacyjnie można również sprawdzić funkcje licznika 74193. Jest to synchroniczny, rewersyjny, 4-bitowy licznik dwójkowy z wejściami ustawiającymi, który jest odpowiednikiem scalonego układu SN 74193.

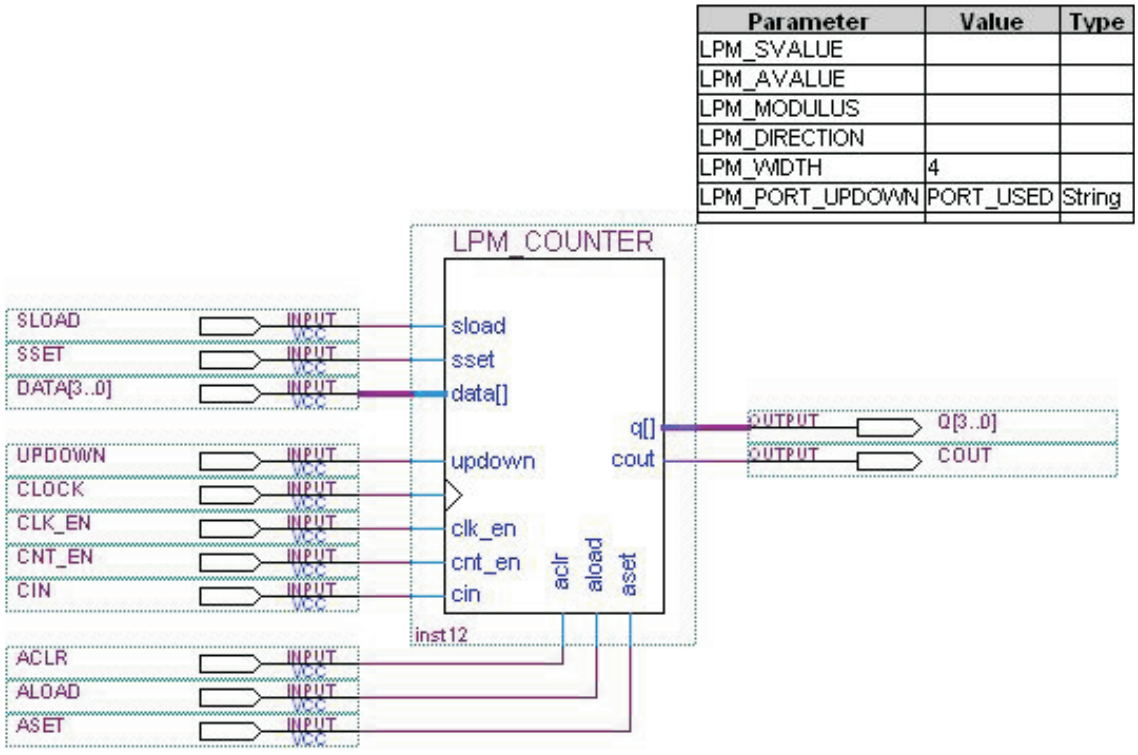
1. Na stronie internetowej przedmiotu wraz z instrukcją laboratoryjną znajduje się projekt liczniki.qpf umożliwiający symulację działania m.in licznika *lpm_counter* (schemat licznik_lpm.bdf (rys. 5.6), plik symulacyjny licznik_lpm.vwf).

   W zamieszczonej na rys. 5.7 symulacji działania licznika lpm_counter przedstawiono:

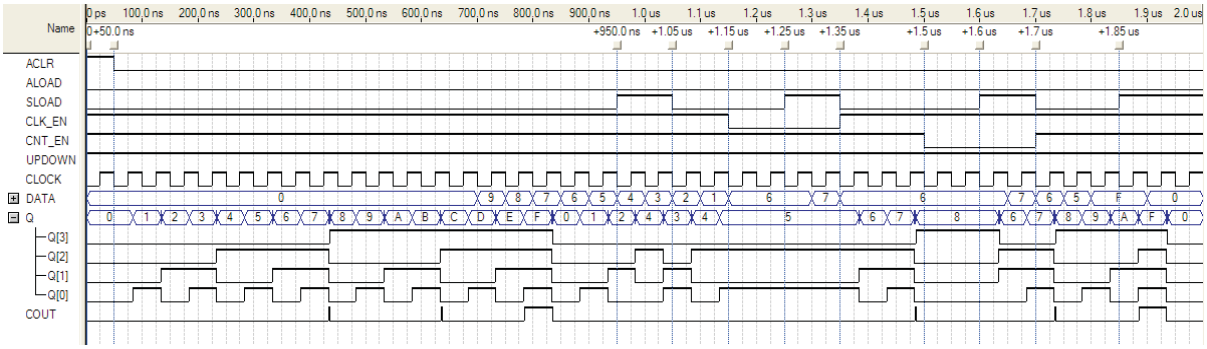
   – 0 ÷ 50ns – asynchroniczne zerowanie licznika poprzez wymuszenie stanu niskiego na wejściu *aclr* licznika;
   – 50ns ÷ 950ns, 1,05µs ÷ 1,15µs, 1,35µs ÷ 1,5µs, 1,7µs ÷ 1,85µs – zliczanie impulsów w przód – wejście updown, clk_en, cnt_en w stanie wysokim;
   – 950ns ÷ 1,05µs, 1,85µs ÷ 2,0µs – synchroniczny wpis równoległy danych do licznika z wejścia data (wpis następuje tylko podczas dodatnich zboczy zegarowych);
   – 1,15µs ÷ 1,35µs – blokada trybu synchronicznego: synchroniczny wpis nie jest aktywny (1,25µs ÷ 1,35µs);
   – 1,5µs ÷ 1,7µs – blokada zliczania impulsów zegarowych: synchroniczny wpis jest aktywny (1,6µs ÷ 1,7µs).

Korzystając z przygotowanego schematu (rys. 5.6) i symulacji, należy sprawdzić symulacyjnie:

– cykl pracy licznika dla kierunku zliczania licznika wstecz (zaobserwować stan wyjścia przeniesienia cout);
– asynchroniczny wpis równoległy danych do licznika.



Rys. 5.6. Schemat do symulacyjnego sprawdzenia funkcji licznika *lpm_counter*



Rys. 5.7. Przykładowe wyniki symulacji licznika *lpm_counter*

Uwaga: Na przebiegach symulacyjnych z rys. 5.7 w sygnale *cout* widoczne są krótkotrwałe impulsy pasożytnicze (ang. *glitch*) występujące w stanach przejściowych (w tym przypadku przy każdej zmianie stanu licznika z 7 na 8 oraz z B na C). Nie należy wykorzystywać wyjścia *cout* w sposób asynchroniczny, gdyż może to skutkować nieprawidłowym działaniem projektowanego układu.
Wyniki symulacji w formie wydruku należy przynieść na zajęcia do wglądu dla osoby prowadzącej laboratorium – zostaną one potem dołączone do sprawozdania.

2. (Dla zainteresowanych) Zapoznać się z funkcjami realizowanymi przez licznik 74193.

Tabela 5.5

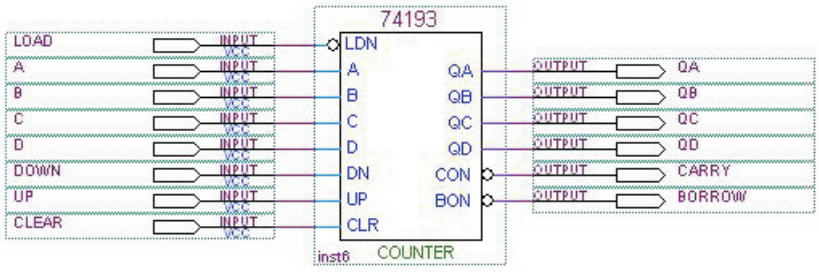
Tablica stanów licznika 74193

Wejścia licznika 74193								Wyjścia licznika 74193					
CLR	UP	DN	LDN	D	C	B	A	QD	QC	QB	QA	CON	BON
H	X	X	X	X	X	X	X	L	L	L	L	X	X
L	X	X	L	d	c	b	a	d	c	b	a	X	X
L	⎍	H	H	X	X	X	X	Zliczanie wprzód				H	H
L	H	⎍	H	X	X	X	X	Zliczanie wstecz				H	H
L	⎍	H	H	X	X	X	X	H	H	H	H	L	H
L	H	⎍	H	X	X	X	X	L	L	L	L	H	L
L	H	H	H	X	X	X	X	Stan poprzedni				X	X
L	⎍	⎍	H	X	X	X	X	Stan niedozwolony				X	X

H, L, X – stan wysoki, niski, dowolny; ⎍ – zmiana stanu z niskiego na wysoki; a, b, c, d – wartości na wejściach A, B, C, D (a, A – LSB; d, D – MSB);CLR (CLEAR) – zerowanie; UP (COUNT UP) – zliczanie wprzód; DN (COUNT DOWN) – zliczanie wstecz; LDN (LOAD) – wprowadzanie równoległe danych; A, B, C, D – wejścia danych; QA, QB, QC, QD – wyjścia danych; CON (CARRY) – wyjście przeniesienia; BON (BORROW) – wyjście pożyczki.

3. (Dla zainteresowanych) Na stronie internetowej przedmiotu wraz z instrukcją laboratoryjną znajduje się projekt liczniki.qpf umożliwiający testowanie m.in licznika 74193 (rys. 5.8). Korzystając z przygotowanego schematu, sprawdzić symulacyjnie (można wykorzystać plik symulacyjny licznik_74193.vwf):
   - wpis równoległy danych do licznika oraz zerowanie licznika;
   - cykl pracy licznika dla obu kierunków zliczania (wraz z wyjściem przeniesienia Carry i pożyczki Borrow);
   - typ wejść: zerującego CLEAR i ustawiającego LOAD licznika (synchroniczne czy asynchroniczne);

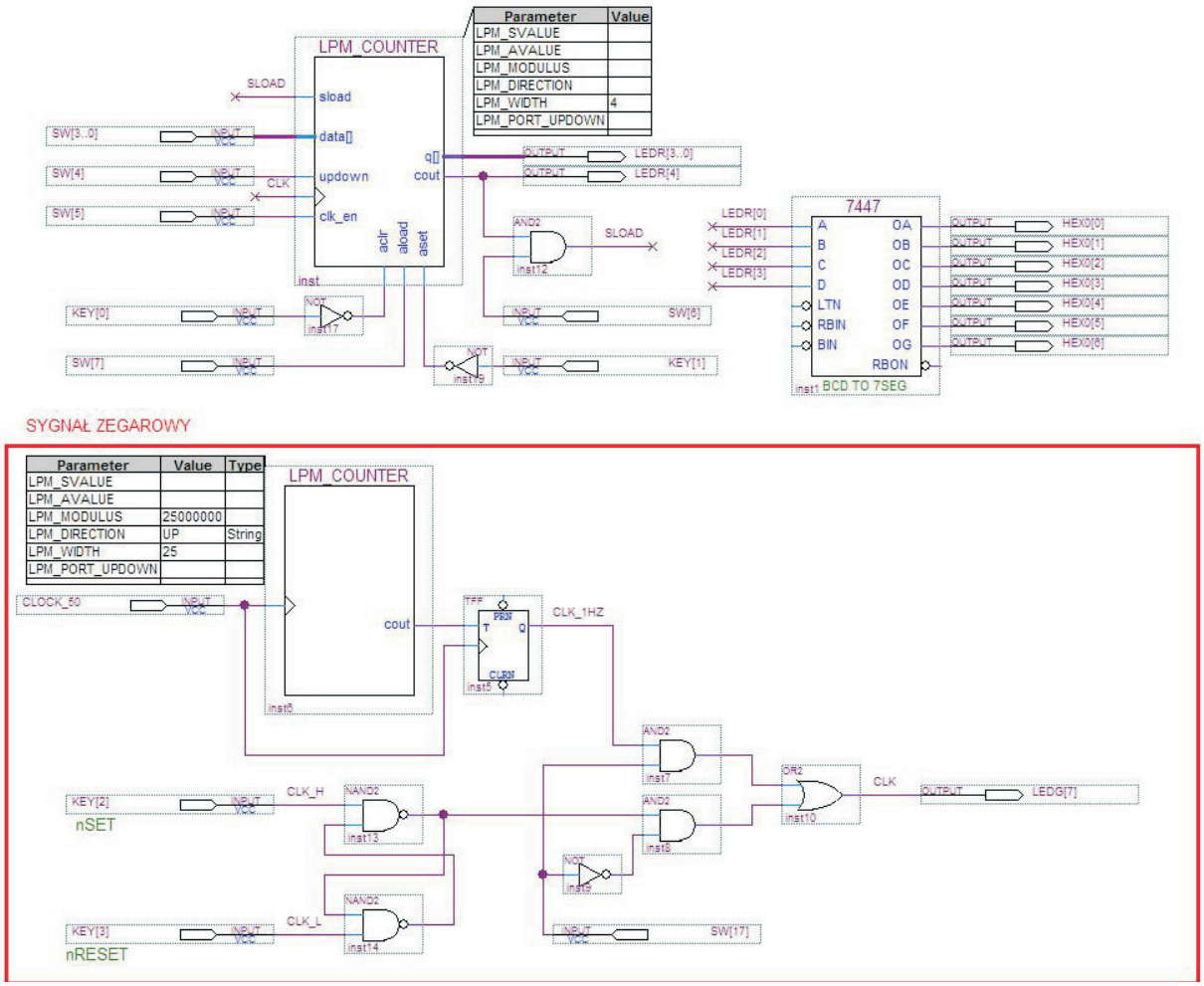
– zachowanie układu przy jednoczesnym podaniu impulsów na wejścia zliczania w górę COUNT UP i zliczania w dół COUNT DOWN.



Rys. 5.8. Schemat do symulacyjnego sprawdzenia funkcji licznika 74193

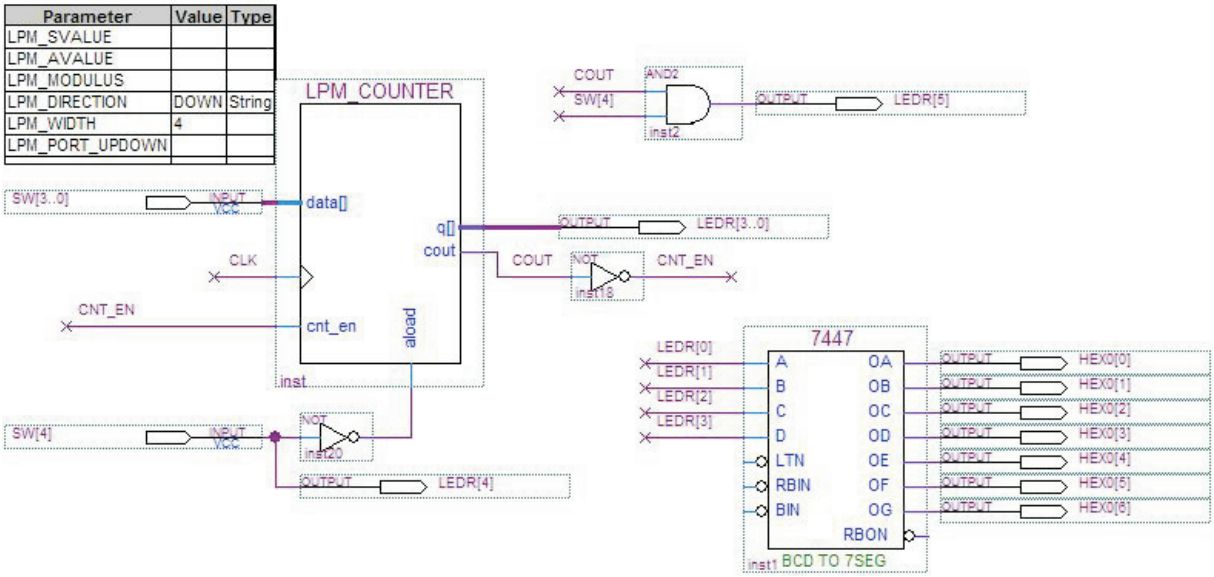
5.5. Program ćwiczenia

1. Należy zapoznać się z funkcjami realizowanymi przez licznik lpm_counter, korzystając ze schematu lpm_counter_lab.bdf (fragment schematu przedstawiono na rys. 6).
   a. Po zaprogramowaniu układu należy ustawić następujące przełączniki w stan wysoki: SW[17], SW[5], SW[4]; pozostałe przełączniki należy ustawić w stan niski. W tej konfiguracji licznik będzie zliczał do góry impulsy podawane na jego wejście zegarowe z generatora przebiegu prostokątnego o częstotliwości 1Hz. Wartość zliczonych impulsów (w kodzie binarnym) jest widoczna na diodach LED[3..0] (LEDR[3] – bit najbardziej znaczący) oraz na wyświetlaczu siedmiosegmentowym HEX0. Należy zaobserwować i zanotować stan wyjścia *cout* (wyjście przeniesienia dla tego kierunku zliczania) dla minimalnej (0) i maksymalnej (15) liczby impulsów zliczonych przez licznik.
   b. Przełącznikiem SW[4] ustawić zliczanie wstecz i zaobserwować działanie licznika, a w szczególności wyjścia *cout* (wyjście pożyczki dla tego kierunku zliczania) dla stanu licznika 0 i 15.
   c. Przyciskając przycisk KEY[0], wykonać asynchroniczne zerowanie licznika. Dlaczego przycisk jest podłączony do wejścia przez bramkę NOT?
   d. Przyciskając przycisk KEY[1], zaobserwować wyjście *q* licznika. Jaki jest skutek przyciśnięcia przycisku?
   e. Ustawić przełącznik SW[7], aby wymusić stan wysoki na wejściu *aload* licznika. Zmieniając ustawienia przełączników SW[3..0], obserwować wyjścia licznika. Do czego służą wejścia *aload* i *data* licznika? Czym różni się wejście *aload* od *sload*?
   f. Sprawdzić funkcje wejść *clk_en* i *cnt_en*. Wynik obserwacji zanotować.
   g. Ustawić następujące przełączniki w stan wysoki: SW[17], SW[6], SW[5] i SW[1]; pozostałe przełączniki należy ustawić w stan niski. Układ z rys. 5.9 w tej konfiguracji jest dzielnikiem częstotliwości przez 2. Wyjściem dzielnika jest wyjście *cout* licznika. Należy przeanalizować i opisać w sprawozdaniu działanie układu w konfiguracji dzielnika częstotliwości.
   h. Zrealizować dzielnik częstotliwości przez 5 w układzie scharakterowanym w punkcie g.
   i. Jak będzie działał dzielnik częstotliwości z paragrafu g) jeśli zostanie zmieniony kierunek zliczania na w przód?



Rys. 5.9. Testowanie wybranych funkcji licznika lpm_counter

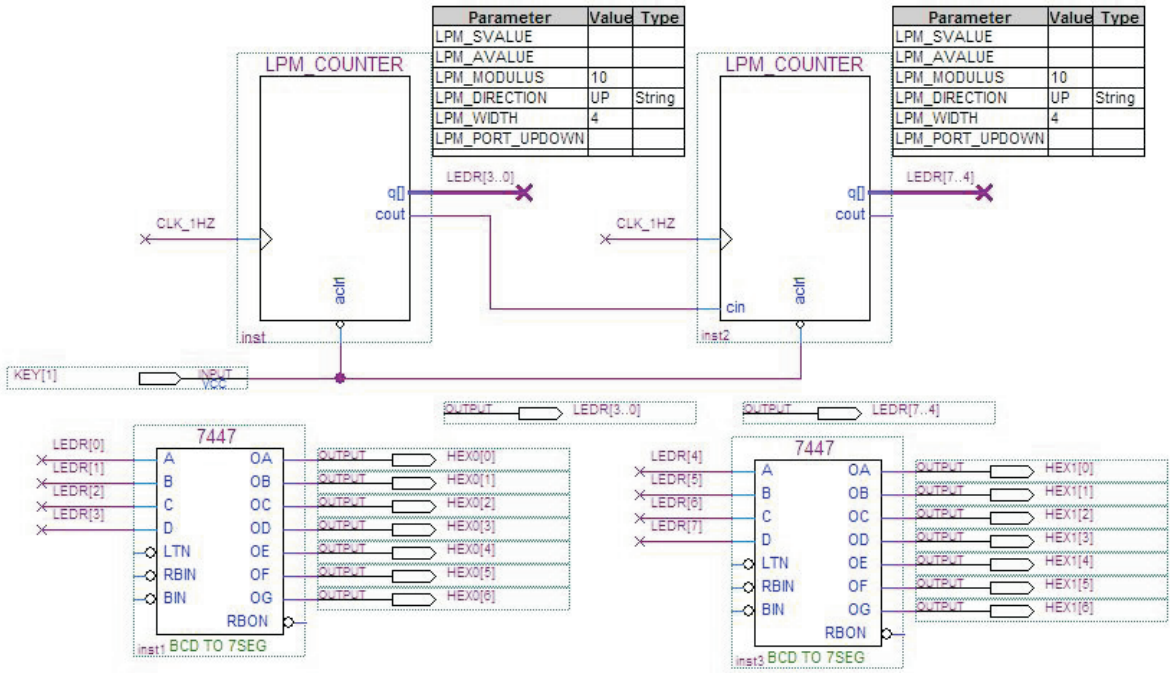
2. Analiza i badanie układu umożliwiającego załączenie wyjścia po opóźnieniu (schemat on_delay.bdf). Czas opóźnienia załączenia wyjścia, które jest podłączone do diody LEDR[5], jest nastawiany (w sekundach) przełącznikami SW[0] ÷ SW[3]. Sygnał wejściowy wymuszany jest za pomocą przełącznika SW[4]. Fragment schematu przedstawiono na rys. 5.10. Jeśli przełącznik SW[4] jest w stanie niskim, to wykonywany jest asynchroniczny wpis liczby z wejścia *data* do licznika. Wówczas każda zmiana wartości wejściowej zostaje wpisana do licznika bez względu na sygnał zegarowy. Jeśli liczba jest większa od 0 to sygnał CNT_EN z zanegowanego wyjścia *cout* licznika powoduje uaktywnienie zliczania w dół (wejście *cnt_en* odblokowane). Przełączenie przełącznika SW[4] ze stanu „0” do „1” wymusza stan niski na wejściu *aload* i rozpoczyna się dekrementacja wartości wpisanej do licznika przy każdym dodatnim zboczu zegarowym (sygnał zegarowy w rozpatrywanym przykładzie ma częstotliwość 1Hz). Jeśli licznik osiągnie stan 0 to sygnał CNT_EN z zanegowanego wyjścia *cout* licznika powoduje zablokowanie dalszego liczenia w dół. Następny cykl może rozpocząć się po ponownym wymuszeniu przełącznikiem SW[4] zmiany stanu z niskiego na wysoki. Sygnały COUT i SW[4] podane są na bramkę AND, która steruje wyjściem układu. Sygnał COUT, a tym samym wyjście układu, jest w stanie wysokim tylko wtedy, gdy w liczniku jest wartość zero, czyli po odliczeniu czasu.



Rys. 5.10. Przykładowa realizacja załączenia wyjścia po opóźnieniu
za pomocą licznika *lpm_counter*

3. Zaprojektować i połączyć wyzwalany generator impulsu o nastawianym czasie trwania oparty na liczniku *lpm_counter*. Wyzwolenie impulsu z przycisku KEY[0]. Nastawa czasu trwania impulsu za pomocą przełączników.
4. Zaprojektować generator sygnału prostokątnego o wypełnieniu 50% (przez połowę okresu stan wysoki sygnału a przez drugą połowę stan niski) z możliwością regulacji okresu. Do realizacji generatora należy wykorzystać ośmiobitowy licznik *lpm_counter*. Wyjście generatora wyprowadzić na diodę LED. Zakres częstotliwości należy tak dobrać aby można było obserwować pracę generatora bez użycia oscyloskopu. Częstotliwość sygnału należy zmieniać przełącznikami. Uruchomienie/zatrzymanie generatora za pomocą przełącznika.
5. (Dla zainteresowanych) Zaprojektować stoper odliczający czas. Wciśnięcie przycisku KEY[0] ma uruchamiać odliczanie czasu, ponowne wciśnięcie tego samego przycisku ma zatrzymywać odliczanie. Przycisk KEY[1] należy wykorzystać do kasowania czasu stopera. Czas ma być wyświetlany na wyświetlaczach siedmiosegmentowych w formacie MDJ, gdzie M – pojedyncze minuty (od 0 do 9), D – dziesiątki sekund (od 0 do 5), J – sekundy (od 0 do 9). W celu realizacji projektu można wykorzystać schemat licz_sekundy.bdf (rys. 5.11), w którym przedstawiono sposób zliczania i wyświetlania sekund i dziesiątek sekund.

Wykorzystano dwa czterobitowe liczniki liczące w przód modulo 10 (parametr LPM_MODULUS = 10). Wyjście *cout* pierwszego licznika (licznika sekund) podłączono do wejścia *cin* drugiego licznika (licznika dziesiątek sekund). Jeśli pierwszy licznik osiągnie stan 9 to na jego wyjściu *cout* pojawi się stan wysoki, który podany na wejście *cin* drugiego licznika odblokuje możliwość zliczania impulsów (można wykorzystać w tym celu również wejście *cnt_en*) – gdy licznik pierwszy zmieni stan z 9 na 0 to drugi zwiększy stan o jeden. Wejścia *aclr* wykorzystane w licznikach zostały zanegowane (symbolizują to niewielkie okręgi na wejściach).



Rys. 5.11. Przykładowa realizacja zliczania i wyświetlania sekund

5.6. Opracowanie wyników

W sprawozdaniu należy zamieścić swoje obserwacje działania różnych konfiguracji układów z punktów 0.0 i 0.0 oraz udzielić odpowiedzi na postawione tam pytania. Należy zamieścić rozwiązania (schemat + opis działania + symulacja) zadań 0.0, 0.0 (i 0.0 dla chętnych). Liczba punktów możliwych do uzyskania za rozwiązanie poszczególnych zadań jest następująca: 0.0 i 0.0 – 3 pkt, 0.0 – 1 pkt, 0.0 – 1 pkt, 0.0 – 1 pkt.

5.7. Pytania kontrolne

1. Wyjaśnić pojęcia licznik, pojemność licznika i długość licznika.
2. Wymienić podstawowe różnice między licznikami synchronicznymi i asynchronicznymi.
3. Do czego służą wyjścia przeniesienia i pożyczki licznika?
4. Co to jest dzielnik częstotliwości?
5. Podać zasadę działania licznika/dzielnika częstotliwości z detekcją stanu końcowego i zerowaniem.
6. Podać zasadę działania podzielnika częstotliwości z wykorzystaniem wejść wpisu równoległego i wyjścia pożyczki licznika.

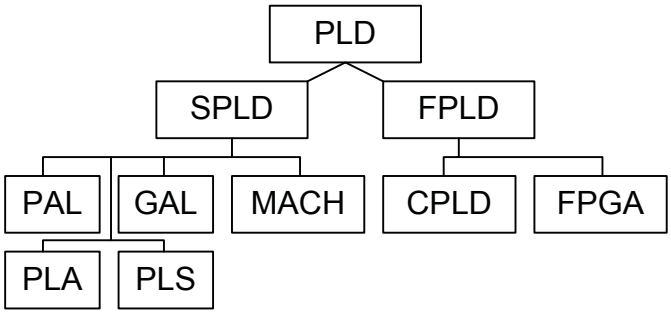
5.8. Literatura

[1] Materiały firmy Altera, www.altera.com.
[2] Haras A., Nieznański J: *Komputery i programowanie II. Materiały pomocnicze do laboratorium.* Gdańsk: Wyd. Polit. Gdańskiej 1990.
[3] Pienkoś J., Turczyński J.: *Układy scalone TTL w systemach cyfrowych*. Warszawa 1980.
[4] Misiurewicz P., Grzybek M.: *Półprzewodnikowe układy logiczne TTL*. Warszawa 1982.

6. Wprowadzenie do języka opisu sprzętu VHDL

6.1. Wprowadzenie

Współczesne urządzenia cyfrowe często projektuje się, bazując na programowalnych układach cyfrowych PLD (ang. Programmable Logic Devices). Układy PLD stosowane są zarówno dla zastosowań wielkoseryjnych, jak i w przypadku urządzeń o specyficznych wymaganiach technicznych [1]. Rodzina układów PLD składa się z dwóch podstawowych grup: układów SPLD (ang. Simple PLD) oraz FPLD (Field PLD). Na rys. 6.1 pokazano uproszczone struktury najważniejszych grup układów PLD.

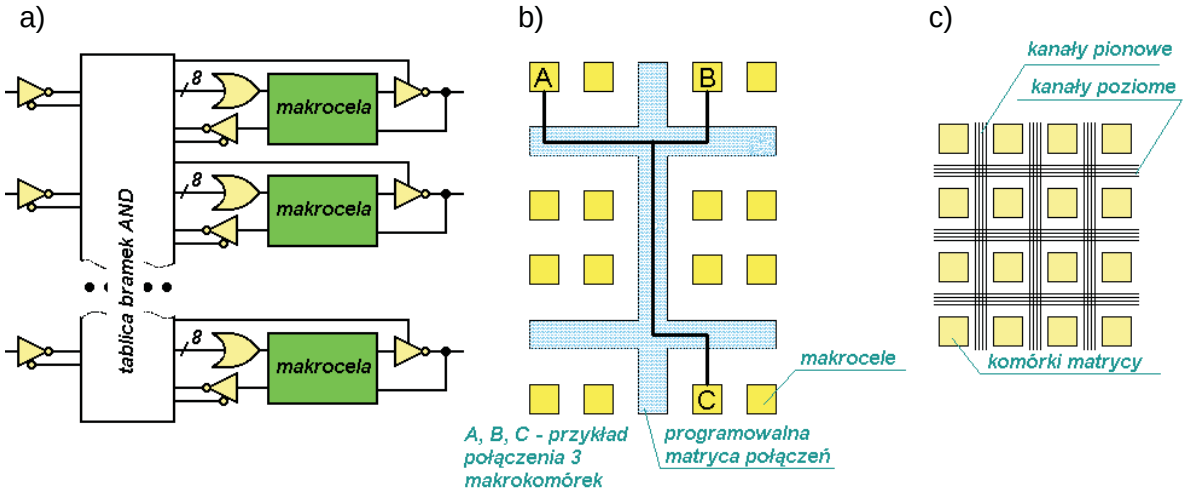


Rys. 6.1. Grupy układów PLD

Do celów dokumentacyjnych i projektowych niezbędne jest wykonanie opisu układu cyfrowego. Do najczęstszych metod opisu należą: (1) opis słowny w formie instrukcji lub podręcznika użytkowania (2) grafy przejść i stanów układu cyfrowego wraz z diagramami czasowymi (3) schemat struktury zawierający sformalizowane symbole i opisy (4) sformalizowany opis układu cyfrowego wg określonego standardu. Dla projektów o większej skali integracji lub nietypowych rozwiązań sprzętowych najbardziej wydajnym sposobem dokumentowania i projektowania jest użycie języka opisu sprzętu HDL (ang. Hardware Description Language) jak sformalizowanego sposobu opisu układu cyfrowego. Finalnym etapem prac z wykorzystaniem języków HDL jest zaprogramowanie i uruchomienie programowalnego układu logicznego ze strukturą i funkcjonalnością pisaną w tym języku. Jednym ze światowych standardów języka HDL jest VHDL. Skrót pochodzi od dwóch innych skrótów: **V**ery **H**igh Speed Integrated Circuit oraz HDL – Hardware **D**escription **L**anguage. Szczególnie rozwijany był na początku lat 80. do opisu niezależnych metod syntezy układów elektronicznych przez American Department of Defence (ang. DoD). Po raz pierwszy został ustandaryzowany w roku 1983, a następnie standaryzację powtórzono w latach 1987 (IEEE Std 1076) oraz 1993 (IEEE Std 1076, IEEE Std 1164).

Opis projektowanego układu w języku VHDL jest tworzony w postaci pliku tekstowego. Etap edycji pliku jest najczęściej wspomagany przez środowisko IDE (ang. Integrated Development Environment), które oferuje możliwość korzystania z tzw. wzorców (ang. wizzard template) tekstu czy projektu. Język VHDL jest platformą programowania wykładaną na wielu światowych uczelniach technicznych. Projekty tworzone w tym języku stanowią nieodłączny element nowoczesnych rozwiązań w technice cyfrowej. Mimo tego, że jezyk HDL jest

uniwersalnym językiem pozwalającym na przenoszenie kodu między różnymi platformami projektowymi, w językach programowania HDL fundamentalne znaczenie ma rzetelne rozpoznanie możliwości, budowy i funkcjonowania konkretnego układu PLD.



Rys. 6.2. Uproszczone struktury układów CPLD, FPGA oraz SPLD

6.1.1. Operatory

Tabela 6.1

Ważniejsze operatory w języku VHDL

Operator w VHDL	Opis operacji	Grupa
NOT	negacja logiczna	LOGICZNE
AND	iloczyn logiczny	
OR	suma logiczna	
NAND	odwrócony iloczyn logiczny	
NOR	odwrócona suma logiczna	
XOR	suma modulo 2	
XNOR	odwrócona suma modulo 2	
=	równość	RELACJA
/=	nierówność	
<	mniejszy	
>	większy	
<=	mniejszy równy	
>=	większy równy	
+	dodawanie	ARYTMETYCZNE
–	odejmowanie	
*	mnożenie	
/	dzielenie	
MOD	modulo	
REM	reszta	
ABS	moduł	
SLL	logiczne przesunięcie w lewo (wypełnienie zerami)	PRZESUNIĘCIA
SRL	logiczne przesunięcie w prawo (wypełnienie zerami)	
SLA	arytmetyczne przesunięcie w lewo (wypełnienie jedynkami)	
SRA	arytmetyczne przesunięcie w lewo (wypełnienie jedynkami)	
ROL	rotacja pierścieniowa w lewo	
ROR	rotacja pierścieniowa w prawo	
&	połączenie (sklejenie) bitów	SKLEJANIE BITÓW

6.1.2. Argumenty i typy danych

W języku VHDL istnieją trzy grupy argumentów: sygnały (ang. *signals*), zmienne (ang. *variable*) oraz stałe (ang. *constant*). Sygnały należy zawsze rozpatrywać jako rzeczywiste połączenia w strukturach cyfrowych. Zmienne są obiektami danych wirtualnych wspomagających pisanie i syntezę kodu; zmienne mogą być stosowane w procesach lub podprogramach. Ważniejsze typy zamieszczono w tab. 6.2 [2, 3, 4].

Tabela 6.2

Ważniejsze typy w języku VHDL

Typ	Przyjmowane wartości
STD_LOGIC	„U" (niezainicjalizowany), „X" (nieznany), „0" (silne zero), „1" (silne jeden), „Z" (wysoka impedancja), „W" (słaby nieznany), „L" (słabe zero), „H" (słabe jeden), „–„ (nieistotny)
BOOLEAN	FALSE, TRUE
BIT	'0', '1'
CHARACTER	NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL, BS, HT, LF, VT, FF, CR, SO, SI, DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB, CAN, EM, SUB, ESC, FSP, GSP, RSP, USP, ' ', '!', '"', '#', '$', '%', '&', ''', '(', ')', '*', '+', ',', '–', '.', '/', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>', '?', '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\', ']', '^', '_', '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~', DEL
INTEGER	–2147483648 … 2147483647
REAL	–1.7014110e+038 … 1.7014110e+038
NATURAL	0 … 2147483647
POSITIVE	1 … 2147483647
TIME	–9223372036854775808 … 9223372036854775807
STRING	Tablica obiektów CHARACTER
BIT_VECTOR	Tablica BIT

Przykłady:

stała

```
constant <constant_name> : <type> := <constant_value>
constant STALA : integer := 7
```

integer

```
type <name> is range <low> to <high>
type index_t is range 0 to 7
type addr_t is range 255 downto 0
```

tablica

```
type <name> is array(<range_expr>) of <subtype_indication>
type byte_t is array(7 downto 0) of std_logic
type mem_t is array(7 downto 0) of std_logic_vector(7 downto 0)
```

wyliczenia

```
type <name> is (<enum_literal>, <enum_literal>, ...)
type state_t is (IDLE, READING, WRITING, DONE)
```

sygnał

```
signal <name> : <type> := <default_value>
signal <name> : std_logic
signal <name> : std_logic_vector( 7downto 0)
signal <name> : integer
signal <name> : integer range 0 to 255
```

zmienna

```
variable <name> : <type> := <default_value>
variable <name> : std_logic
variable <name> : std_logic_vector(7downto 0)
variable <name> : integer
variable <name> : integer range 0 to 7
```

6.1.3. Instrukcje

W języku VHDL można wyróżnić dwa rodzaje instrukcji: sekwencyjne oraz równoległe. Do grupy instrukcji sekwencyjnych należą:

— Przypisanie do zmiennej :=
— Instrukcja czekania wait
— Instrukcja warunkowa if-then-else
— Instrukcja wyboru case
— Instrukcja pętli loop
— Instrukcja pusta null
— Instrukcja testowa assert

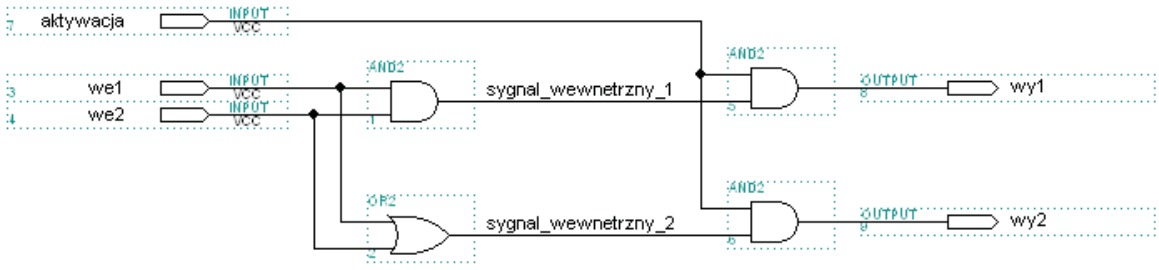
Instrukcje równoległe to:

— Przypisanie do sygnału <=
— Instrukcja procesu process
— Instrukcja łączenia komponentów port map
— Instrukcja powielania elementów generate
— Przypisanie warunkowe when-else
— Przypisanie selektywne with-select

Warto zapoznać się z działaniem funkcji *Insert Template* w trakcie edycji kodu VHDL. Po naciśnięciu prawego przycisku myszy i wybraniu tej opcji można skorzystać z gotowych szablonów kodów. Należy tylko wprowadzić w oznaczone miejsca w kodzie swoje nazwy (dotyczy to szczególnie nazw sygnałów oraz zmiennych) [3, 4].

6.1.4. Budowa programu

Proste projekty (tworzone na podstawie jednego źródłowego pliku tekstowego) składają się z trzech części: deklaracji bibliotek (LIBRARY), deklaracji wyprowadzeń bloku (ENTITY) oraz deklaracji architektury i działania bloku (ARCHITECTURE). Poniżej na rys. 6.3 zamieszczono prostą strukturę cyfrową z 3 wejściami (IN), 2 wyjściami (OUT), 3 bramkami AND oraz jedną bramką OR.



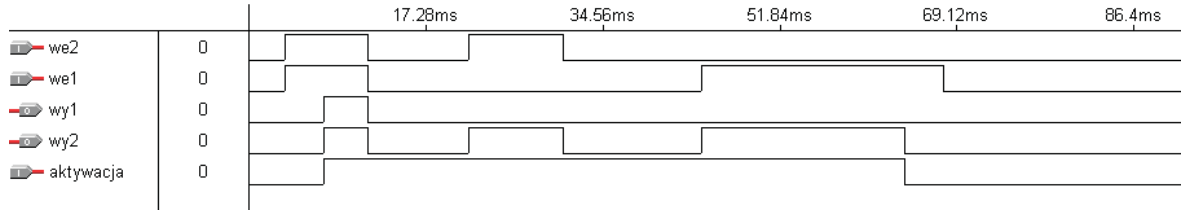
Rys. 6.3. Przykładowa struktura cyfrowa

Kod programu jest następujący.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
LIBRARY altera;
USE altera.maxplus2.ALL;
```

```
ENTITY bramki IS
PORT(
        we1, we2, aktywacja       : IN STD_LOGIC;
        wy1, wy2          : OUT    STD_LOGIC);
END bramki;

ARCHITECTURE moja_struktura OF bramki IS
      SIGNAL sygnal_wewnetrzny_1 : STD_LOGIC;
      SIGNAL sygnal_wewnetrzny_2 : STD_LOGIC;
BEGIN
      sygnal_wewnetrzny_1 <= we1 AND we2;
      sygnal_wewnetrzny_2 <= we1 OR we2;
      wy1 <= sygnal_wewnetrzny_1 AND aktywacja;
      wy2 <= sygnal_wewnetrzny_2 AND aktywacja;
END moja_struktura;
```

Symulacja funkcjonalna jest jednym ze sposobów sprawdzenia poprawności działania bloku cyfrowego. Przebiegi ilustrujące działanie struktury opisanej przez fragment kodu zamieszczono na rys. 6.4.



Rys. 6.4. Symulacja działania struktury z rys. 6.3

6.2. Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z elementami języka opisu sprzętu VHDL oraz ze sposobem edycji kodu z wykorzystaniem wbudowanych narzędzi wspomagających edycję kodu tzw. wzorców. Po sprawdzeniu poprawności składni języka weryfikacja funkcjonalna będzie przeprowadzana z użyciem symulacji i sprzętowych narzędzi testowych.

6.3. Przygotowanie do zajęć

Należy utworzyć projekt, do którego następnie należy dodać 3 pliki VHDL utworzone z wykorzystaniem wzorców. Każdy plik reprezentuje strukturę, którą należy następnie skompilować oraz sprawdzić symulacyjnie. Wzorce są następujące:

1. VHDL/Full Design/Shift Register/Basic Shift Register,
2. VHDL/Full Design/Arithmetic/Adders/Unsigned Adder,
3. VHDL/Full Design/Arithmetic/Counters/Binary Counter.

Nie wprowadzać modyfikacji kodu. Przeanalizować składnię kodu oraz budowę głównych fragmentów. Przygotować wszystkie pliki do wglądu prowadzącego.

6.4. Przebieg ćwiczenia

6.4.1. Układ detekcji zbocza narastającego oraz opadającego (poziom podstawowy)

Utworzyć symbol ze struktury „Basic Shift Register". Zaprojektować układ detekcji zbocza narastającego oraz opadającego, wykorzystując element „Basic Shift Register" z biblioteki użytkownika oraz bramkę XOR. Utworzenie symbolu na bazie aktualnego pliku powoduje dodanie nowo powstałego symbolu do listy elementów w bibliotece użytkownika osadzonej w katalogu projektu. Omówić wyniki na podstawie przebiegów symulacyjnych.

6.4.2. Układ detekcji określonego wyniku dodawania (poziom podstawowy)

Utworzyć symbol ze struktury „Unsigned Adder". Wykorzystać podstawowe bramki i zaprojektować układ detekcji wyniku sumowania 0xAA. Wystąpienie stanu powinno być wskazywane przez pojawienie się jedynki na wyjściu sygnalizacyjnym. Omówić wyniki na podstawie przebiegów symulacyjnych.

6.4.3. Układ 4-bitowego licznika binarnego zliczającego w górę (poziom podstawowy)

Zaprojektować 4-bitowy licznik binarny zliczający w górę. Strukturę cyfrową zakodować w języku VHDL. Przeprowadzić sprzętową weryfikację funkcjonalną z wykorzystaniem narzędzia In-System Source and Probes z menu Tools. Dostosować układ bazowy do wymagań postawionych przez prowadzącego.

6.4.4. Wizualizacja wyjścia licznika na wskaźniku siedmiosegmentowym (poziom rozszerzony)

Zaprojektować układ generacji sygnału zegarowego w sposób statyczny z użyciem przycisków lub układ generacji sygnału zegarowego 1Hz z oscylatora kwarcowego. Zaprojektować dekoder kody binarnego na kod BCD dla wskaźnika siedmiosegmentowego. Połączyć strukturę generatora przebiegu zegarowego z licznikiem z punktu 7.4.3. Do wyjścia licznika Q podłączyć jeden wybrany wskaźnik.

6.5. Wskazówki

1. Układ statycznej generacji sygnału zegarowego można zrealizować na przerzutniku JKFF i dwóch przyciskach.
2. Układ generacji zegara z oscylatora kwarcowego można oprzeć na dzielniku częstotliwości. Najprostszym dzielnikiem częstotliwości jest przerzutnik TFF, który dzieli wartość częstotliwości prze 2. Wygodnie jest wprowadzić licznik o rozmiarze 24 bitów zliczający modulo do wartości wynikającej z częstotliwości oscylatora kwarcowego. Jeśli wartość tej częstotliwości wynosi 10MHz, to licznik może zliczać modulo 5 000 000. W omawianym przypadku licznik będzie zliczał od nowa co 0,5 s. Wystarczy na jego wyjście podłączyć przerzutnik TFF jako dzielnik częstotliwości przez 2, a otrzyma się przebieg o wypełnieniu 50% i częstotliwości 1Hz.
3. Jeśli zaimportuje się mapę nazw wyprowadzeń *DE2_default.qsf* nie będzie trzeba ręcznie wpisywać nazw wyprowadzeń w menu Assiggments.

6.6. Pytania kontrolne

1. Jakie części (fragmenty) kodu można wyróżnić w jezyku VHDL. Która część kodu odpowiada za liczbę i kierunek wyprowadzeń bloku?
2. Opisać instrukcję warunkową if-then-else. Podać jej zastosowania.
3. Opisać instrukcję warunkową case-when. Podać jej zastosowania.
4. Omówić pojęcia: procesy w strukturze cyfrowej, operacje równoległe, operacje sekwencyjne. Podać przykłady.
5. Podać i pogrupować instrukcje wykonywane przez potencjalną jednostkę arytmetyczno – logiczną ALU zaimplementowaną w strukturze cyfrowej.
6. Jakie słowo kluczowe powinno pojawić się w tekście kody VHDL, aby po wygenerowaniu symbolu użytkownik miał możliwość konfiguracji wybranych parametrów bloku?

6.7. Literatura

[1] Praca zbiorowa, red. J. Kalisz: *Język VHDL w praktyce*. Warszawa: WKŁ 2002, ISBN 83-206-1440-6

[2] Praca zbiorowa, red. T. Łuba: *Programowalne układy przetwarzania sygnałów i informacji*. Warszawa: WKŁ 2008, ISBN 978-83-206-1711-5

[3] Pasierbiński J., Zbysiński P.: *Układy programowalne w praktyce*. Warszawa: WKŁ 2001, ISBN 83-206-1393-0

[4] Hamblen J. O., Hall T. S., Furman M. D., Rapid prototyping of Digital Systems SOPC Edition, Springer, ISBN 978-0-387-72670-0

7. Projektowanie synchronicznego układu sekwencyjnego

7.1. Cel i zakres ćwiczenia

Celem ćwiczenia jest zaprojektowanie, implementacja i analiza pracy synchronicznego układu sekwencyjnego.

7.2. Wprowadzenie

Układy sekwencyjne (automaty) to układy cyfrowe, w których stan wyjść zależy od aktualnego stanu wejść oraz od stanu wejść w poprzednich chwilach czasowych. Układy te posiadają zdolność pamiętania stanów logicznych z poprzednich chwil. Tą możliwość pamiętania modeluje się poprzez określenie stanu elementów pamięciowych układu, tzw. stanu wewnętrznego (lub po prostu – stanu). Pod wpływem sygnałów wejściowych układ może zmienić stan na inny, a jego sygnały wyjściowe zależą:

— wyłącznie od stanu układu – w przypadku tzw. automatu Moora,
— od stanu układu i sygnałów wejściowych – dla tzw. automatu Mealy'ego.

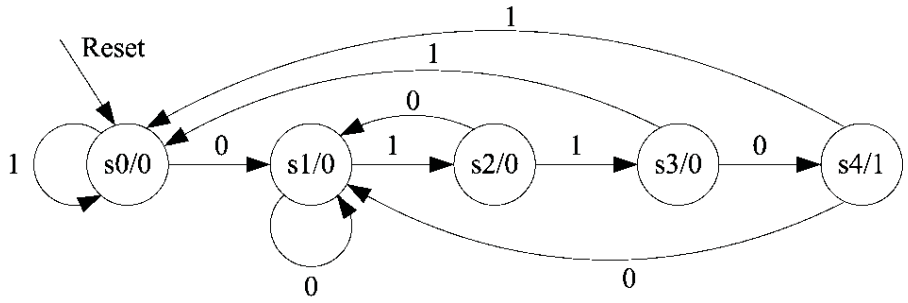
Z określonego stanu układ przechodzi do innego ściśle określonego stanu tylko po pobudzeniu odpowiednim słowem wejściowym. Układy sekwencyjne opisuje się w inny sposób niż układy kombinacyjne ze względu na konieczność uwzględnienienia stanu. Dla każdego stanu układu i każdego słowa wejściowego należy określić do jakiego stanu układ przejdzie i jaki będzie stan wyjść. Oprócz opisu słownego, najczęściej przedstawia się to za pomocą tablicy przejść i wyjść, grafu lub wykresu czasowego. Wymienione formy opisu opisują zachowanie układu we wszystkich możliwych przypadkach. Przykładową tablicę przejść i wyjść oraz graf można znaleźć w dalszej części instrukcji (tab. 7.1 i rys. 7.1); tablice przejść oraz grafy były również stosowane w ćwiczeniu 0 do opisu działania przerzutników – elementów pamięciowych o dwóch stanach. Przy wykorzystaniu przerzutników oraz bramek można budować złożone układy sekwencyjne, posługując się technikami opisanymi w dalszej części instrukcji.

Tablica przejść automatu opisuje przejścia od stanu do stanu na skutek określonych wektorów wejściowych we wszystkich możliwych przypadkach. Sporządzone na podstawie opisu słownego tablice przejść i wyjść często zawierają więcej stanów niż jest to wymagane do zaprojektowania poprawnie działającego automatu. Wówczas przeprowadza się minimalizację liczby stanów polegającą na wyszukaniu stanów, które mogą być zastąpione jednym. W niniejszym ćwiczeniu zagadnienie minimalizacji liczby stanów nie będzie rozpatrywane, informacje na ten temat można znaleźć m.in. w [5].

Tablica wyjść zależy od rodzaju automatu, ćwiczenie niniejsze będzie dotyczyć wyłącznie automatów synchronicznych Moore'a, dla tego typu automatu w tablicy wyjść (tab. 7.1, tab. 7.2) umieszcza się wartości sygnałów wyjściowych odpowiadających danemu stanowi. Należy podkreślić, że nie zależą one od sygnałów wejściowych, a jedynie od stanów wewnętrznych automatu.

7.3. Proces projekowania automatu synchronicznego

Proces projektowania automatu synchronicznego zostanie przedstawiony na przykładzie rozwiązania zadania detekcji ciągu (sekwencji) bitów 0110. Układ ma generować na wyjściu oznaczonym jako „wy" stan wysoki po wystąpieniu na wejściu „*we*" sekwencji 0110. Zostaną zaprojektowane trzy układy sekwencyjne, każdy z wykorzystaniem innego przerzutnika synchronicznego, kolejno D, JK oraz T. Projektowanie automatu synchronicznego zaczyna się od analizy opisu słownego. Narysowano graf automatu (rys. 7.1), na podstawie którego można w prosty sposób wyznaczyć tablicę przejść i wyjść.



Rys. 7.1. Graf układu detekcji ciągu bitów

Znaczenie poszczególnych stanów jest następujące:
- s0 – pierwszy element sekwencji nie został podany na wejście, wartość wyjścia *wy* = 0;
- s1 – wykryty pierwszy element sekwencji, *wy* = 0;
- s2 – wykryte dwa pierwsze elementy sekwencji, *wy* = 0;
- s3 – wykryte trzy pierwsze elementy sekwencji, *wy* = 0;
- s4 – wykryta cała sekwencja, *wy* = 1.

Na podstawie grafu automatu przedstawionego na rys. 7.1 sporządzono tablicę przejść i wyjść automatu Moore'a (tab. 7.1), a następnie, posługując się naturalnym kodem binarnym (NKB), zakodowano ją (tab. 7.2) w taki sposób, że stanowi s0 przyporządkowano ciąg 000, a kolejnym stanom przypisano kolejne wartości NKB. Oprócz NKB mogą być stosowane także inne kody np. kod Graya.

Tabela 7.1

Tablica przejść i wyjść

s \ *we*	0	1	*wy*
s0	s1	s0	0
s1	s1	s2	0
s2	s1	s3	0
s3	s4	s0	0
s4	s1	s0	1

Tabela 7.2

Zakodowana tablica przejść i wyjść

s \ *we*	*we* = 0	*we* =1	*wy*
$Q_2Q_1Q_0$	$Q_2Q_1Q_0$	$Q_2Q_1Q_0$	
000	001	000	0
001	001	010	0
010	001	011	0
011	100	000	0
100	001	000	1
101	xxx	xxx	x
110	xxx	xxx	x

Minimalna długość ciągu binarnego zależy od liczby stanów automatu. W przypadku tego zadania mamy pięć stanów, musimy więc przyjąć ciąg trzybitowy. Minimalna liczba przerzutników wystarczająca do realizacji automatu jest tym samym równa trzy. W zakodowanej tablicy przejść i wyjść stany (a zarazem wyjścia) przerzutników oznaczono literami Q z odpowiednim indeksem dolnym. Bardziej znaczącemu bitowi (pierwszy z lewej) przypisano przerzutnik Q_2, a mniej znaczącemu (pierwszy z prawej) przerzutnik Q_0. Przy użyciu trzech przerzutników można zakodować osiem stanów, zakodowaną tablicę przejść rozszerzono uwzględniając pozostałe stany, którym przypisano dowolne przejścia i wyjścia. Ułatwi to projektowanie funkcji wzbudzeń przerzutników metodą minimalizacji wykorzystującej tablice Karnaugha.

Następnie należy wybrać typ stosowanego przerzutnika, decydującym czynnikiem odnośnie wyboru typu przerzutnika jest najczęściej najprostsza postać zaprojektowanego automatu. Aby wybrać najprostsze rozwiązanie, warto jest zaprojektować układy realizujące ten sam problem z wykorzystaniem różnych przerzutników, a następnie je porównać.

Kolejnym krokiem projektowania automatu jest synteza wzbudzeń, czyli wyznaczenie funkcji boolowskich do sterowania wejściami przerzutników w taki sposób, aby stany przerzutników zmieniały się zgodnie z zakodowaną tabelą przejść. Argumentami funkcji boolowskich są zmienne wejściowe i wyjścia przerzutników. Do wyznaczania funkcji wzbudzeń przerzutników wykorzystuje się ich tablice wzbudzeń, które zamieszczono (łącznie) poniżej (tab. 7.3). W tablicy użyto skrótów literowych: m0, s, r oraz m1 na oznaczenie poszczególnych typów wzbudzeń przerzutników.

Tabela 7.3

Tablice wzbudzeń przerzutników D, T, JK

Q → Q'	wzb.	D	T	JK
0 → 0	m0	0	0	0x
0 → 1	s	1	1	1x
1 → 0	r	0	1	x1
1 → 1	m1	1	0	x0

Dla wybranego tybu przerzutnika tworzy się tablicę wzbudzeń automatu na podstawie zakodowanej tablicy przejść. W prezentowanym rozwiązaniu utworzono tablicę wzbudzeń automatu, którą wypełniono oznaczeniami poszczególnych typów wzbudzeń (tab. 7.4). Na jej podstawie utworzono następnie tablice wzbudzeń układów wykorzystujących określone przerzutniki.

Tabela 7.4

Tablica wzbudzeń układu wypełniona ogólnymi typami wzbudzeń

	Prz. 2		Prz. 1		Prz. 0	
s \ we		1	0	1	0	1
000	m0	m0	m0	m0	s	m0
001	m0	m0	m0	s	m1	r
010	m0	m0	r	m1	s	s
011	s	m0	r	r	r	r
100	r	r	m0	m0	s	m0
101	x	x	x	x	x	x
110	x	x	x	x	x	x

Tabela 7.5

Tablica wzbudzeń układu z przerzutnikami D

	D2		D1		D0	
s \ we	0	1	0	1	0	1
000	0	0	0	0	1	0
001	0	0	0	1	1	0
010	0	0	0	1	1	1
011	1	0	0	0	0	0
100	0	0	0	0	1	0
101	x	x	x	x	x	x
110	x	x	x	x	x	x

W analizowanym przykładzie tablice wzbudzeń automatów składają się z czterech głównych kolumn (obramowanych grubą linią). W pierwszej kolumnie umieszone są zakodowane stany układu (identycznie jak w tablicy przejść). W następnych kolumnach podane są typy wzbudzeń kolejnych przerzutników dla wszystkich kombinacji sygnałów wejściowych (w tym przypadku tylko dwóch: 0 i 1). Wyznaczanie typów wzbudzeń zostanie wytłumaczone na przykładzie. Z tabeli przejść i wyjść odczytujemy, że dla stanu „000”, przy pobudzeniu we = 0 następnym stanem ma być „001”, czyli w przypadku przerzutnika 2 – ma on nie zmienić swego stanu – należy podać na jego wejścia stany określone przez typ wzbudznia „m0” (0 na wejście D przerzutnika D, 0 na wejście T przerzutnika T, 0 na wejście J oraz x na wejście K przerzutnika JK); w przypadku przerzutnika 1 również „m0”, a dla przerzutnika 0 – „s” (1 na wejście D przerzutnika D, 1 na wejście T przerzutnika T, 1 na wejście J oraz x na wejście K przerzutnika JK), ponieważ ma zmienić stan z 0 na 1.

Tabela 7.6

Tablica wzbudzeń układu z przerzutnikami JK

	JK2		JK1		JK0	
s \ we	0	1	0	1	0	1
000	0x	0x	0x	0x	1x	0x
001	0x	0x	0x	1x	x0	x1
010	0x	0x	x1	x0	1x	1x
011	1x	0x	x1	x1	x1	x1
100	x1	x1	0x	0x	1x	0x
101	xx	xx	xx	xx	xx	xx
110	xx	xx	xx	xx	xx	xx

Tabela 7.7

Tablica wzbudzeń układu z przerzutnikami T

	T2		T1		T0	
s \ we	0	1	0	1	0	1
000	0	0	0	0	□	0
001	0	0	0	1	0	1
010	0	0	1	0	1	1
011	1	0	1	1	1	1
100	1	1	0	0	1	0
101	x	x	x	x	x	x
110	x	x	x	x	x	x

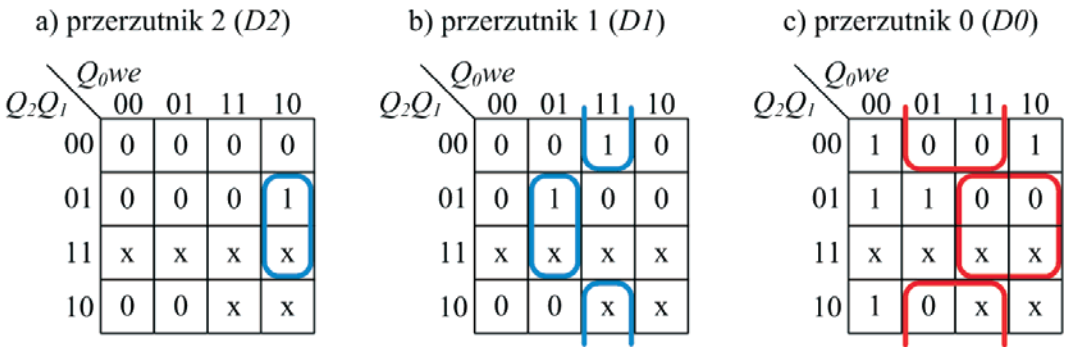
Następnie na podstawie tablic wzbudzeń układu należy wyznaczyć funkcje boolowskie sterujące wejściami przerzutników, można wykorzystać w tym celu tablice Karnaugha.

Po określeniu funkcji wzbudzeń należy wyznaczyć funkcję wyjść na podstawie tablicy wyjść, korzystając np. z tablic Karnaugha. W rozważanym przykładzie funkcję wyjść można uzależnić tylko od stanu przerzutnika 2:

$$wy = Q_2 \tag{7.1}$$

7.3.1. Wyznaczenie funkcji wzbudzeń przerzutników D

Na rysunku 7.2 zamieszczono tablice Karnaugh’a w celu wyznaczenia funkcji wzbudzeń przerzutników typu D.



Rys. 7.2. Tablice Karnaugh’a funkcji wzbudzeń przerzutników typu D

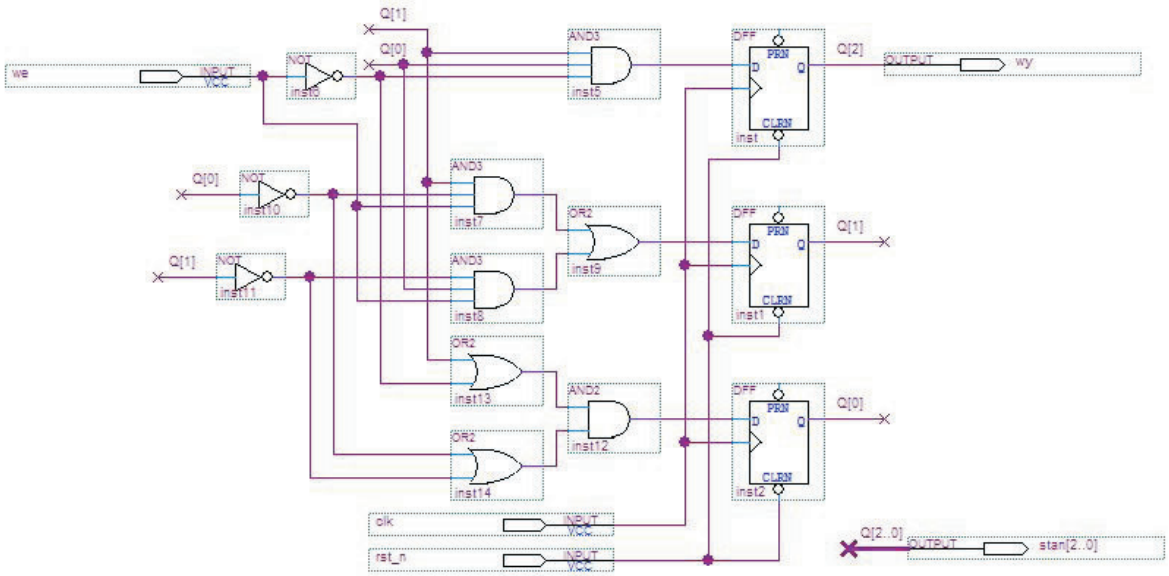
Szukane funkcje wzbudzeń to:

$$D_2 = Q_1 Q_0 \overline{we} \tag{7.2}$$

$$D_1 = Q_1 \overline{Q_0} we + \overline{Q_1} Q_0 we \tag{7.3}$$

$$D_0 = \left(Q_1 + \overline{we}\right)\left(\overline{Q_1} + \overline{Q_0}\right) \tag{7.4}$$

Na rys. 7.3 przedstawiono schemat układu detekcji sekwencji 0110 z wykorzystaniem przerzutników typu D.



Rys. 7.3. Detektor ciągu bitów 0110 zrealizowany z wykorzystaniem przerzutników typu D

7.3.2. Wyznaczenie funkcji wzbudzeń przerzutników JK

Na rysunku 7.4 zamieszczono tablice Karnaugh'a w celu wyznaczenia funkcji wzbudzeń przerzutników typu JK.
Funkcje wzbudzeń są następujące:

$$J_2 = Q_1 Q_0 \overline{we} \tag{7.5}$$

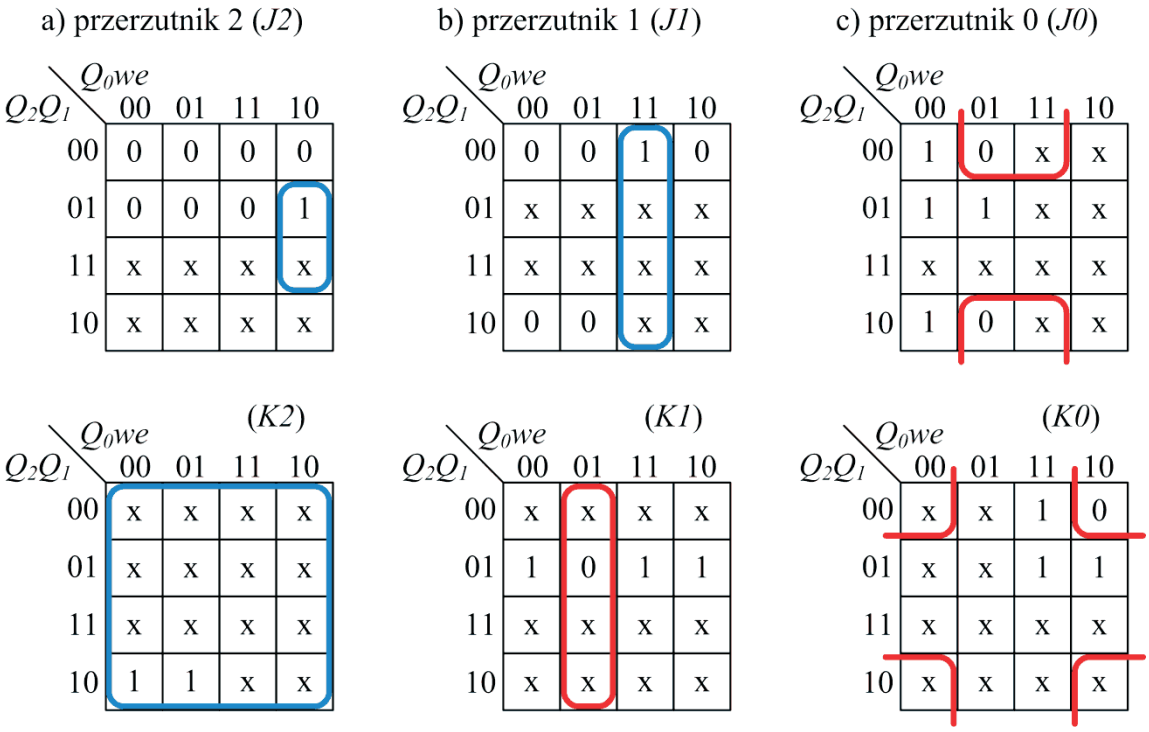
$$K_2 = 1 \tag{7.6}$$

$$J_1 = Q_0 we \tag{7.7}$$

$$K_1 = Q_0 + \overline{we} \tag{7.8}$$

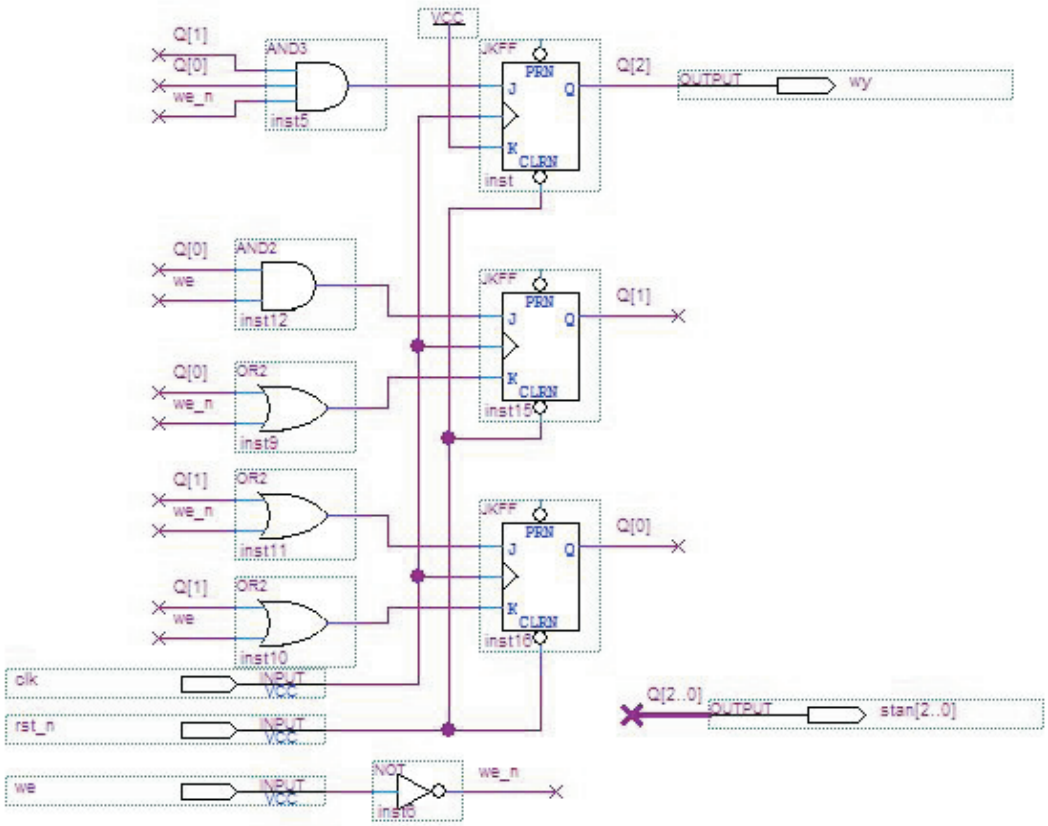
$$J_0 = Q_1 + \overline{we} \tag{7.9}$$

$$K_0 = Q_1 + we \tag{7.10}$$



Rys. 7.4. Tablice Karnaugh'a funkcji wzbudzeń przerzutników typu JK

Na rysunku 7.5 przedstawiono rozwiązanie układu detekcji sekwencji 0110 z wykorzystaniem przerzutników typu JK.



Rys. 7.5. Detektor ciągu bitów 0110 zrealizowany z wykorzystaniem przerzutników typu JK

7.3.3. Wyznaczenie funkcji wzbudzeń przerzutników T

Tablice Karnaugh'a funkcji wzbudzeń przerzutników T przedstawiono na rys. 7.6.

a) przerzutnik 2 (*T2*)

Q_2Q_1 \ Q_0we	00	01	11	10
00	0	0	0	0
01	0	0	0	1
11	x	x	x	x
10	1	1	x	x

b) przerzutnik 1 (*T1*)

Q_2Q_1 \ Q_0we	00	01	11	10
00	0	0	1	0
01	1	0	1	1
11	x	x	x	x
10	0	0	x	x

c) przerzutnik 0 (*T0*)

Q_2Q_1 \ Q_0we	00	01	11	10
00	1	0	1	0
01	1	1	1	1
11	x	x	x	x
10	1	0	x	x

Rys. 7.6. Tablice Karnaugh'a funkcji wzbudzeń przerzutników typu T

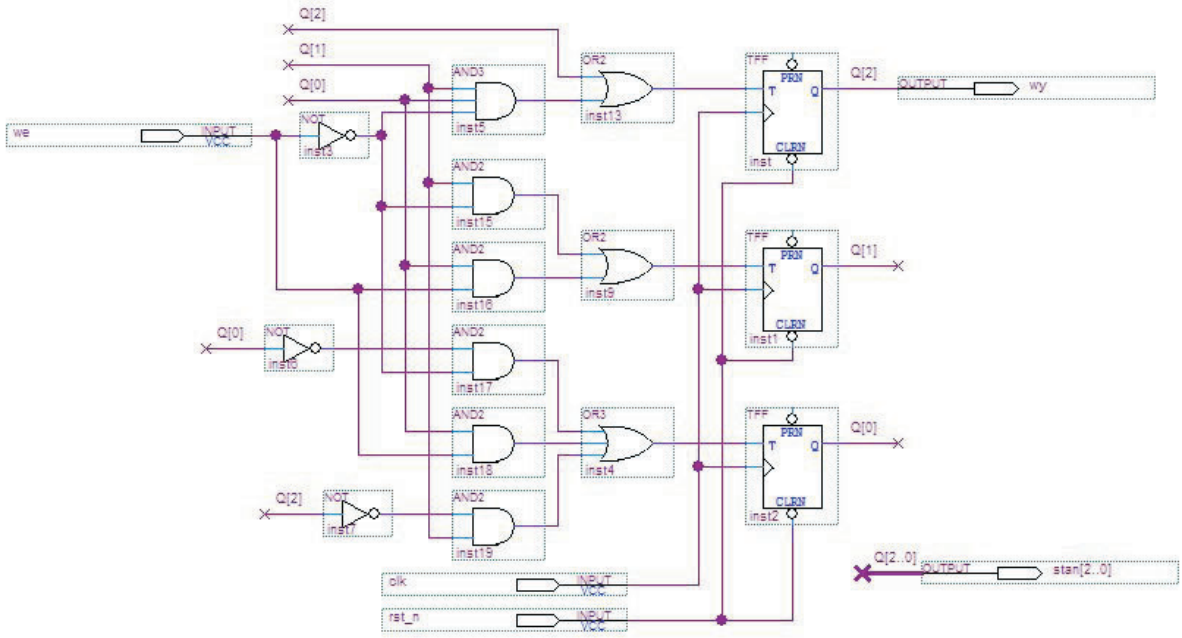
Funkcje wzbudzeń są następujące:

$$T_2 = Q_2 + Q_1 Q_0 \overline{we} \tag{7.11}$$

$$T_1 = Q_1 \overline{we} + Q_0 we \tag{7.12}$$

$$T_0 = \overline{Q_0}\,\overline{we} + Q_0 we + Q_1 \tag{7.13}$$

Na rysunku 7.7 przedstawiono rozwiązanie układu detekcji sekwencji 0110 z wykorzystaniem przerzutników typu T.



Rys. 7.7. Detektor ciągu bitów 0110 zrealizowany na przerzutnikach typu T

7.4. Opis działania układu detekcji sekwencji bitów w języku VHDL

W tym punkcie wyjaśniono, jak można szybko zrealizować automat synchroniczny w języku VHDL na bazie gotowego szablonu. Aby skorzystać z gotowego wzorca automatu należy utworzyć nowy plik w języku VHDL (*File* > *New* > *VHDL File*). Następnie, klikając

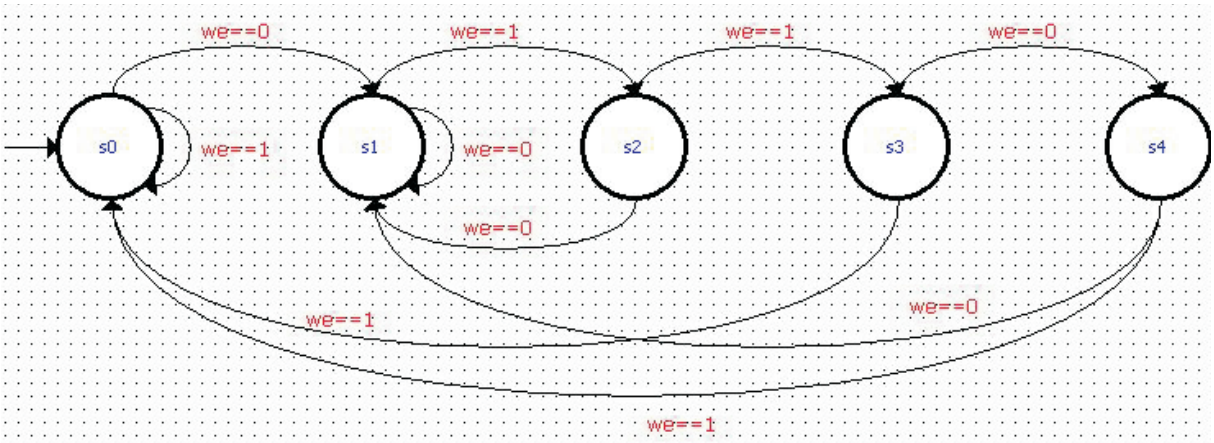
prawym przyciskiem myszy w obszarze strony, można wywołać rozwijane menu podręczne i wybrać opcję *Insert Template* (lub poprzez zakładkę *Edit > Insert Template*). Należy ustawić *VHDL > Full Designs >State Machines > Four State Moore State Machine* i kliknąć na przycisk *Insert*. Następnie należy dostosować szablon do własnych wymagań, zapisać projekt i utworzyć symbol (*File > Create/Update > Create Symbol Files for Current File*), który można wykorzystać w projekcie nadrzędnym. Poniżej przedstawiono plik źródłowy układu detekcji sekwencji 0110, który został utworzony w ten sposób.

```
-- Quartus II VHDL Template
-- A Moore machine's outputs are dependent only on the current state.
-- The output is written only when the state changes.  (State
-- transitions are synchronous.)

library ieee;
use ieee.std_logic_1164.all;

entity d_0110 is

     port(
       clk    : in   std_logic;
       we     : in   std_logic;
       rst_n     : in   std_logic;
       wy     : out  std_logic;
       stany     : out  std_logic_vector (2 downto 0)
     );

end entity;

architecture det_0110 of d_0110 is
     -- Build an enumerated type for the state machine
     type typ_stanu is (s0, s1, s2, s3, s4);
     -- Register to hold the current state
     signal stan   : typ_stanu;

begin
     -- Logic to advance to the next state
     process (clk, rst_n)
     begin
       if rst_n = '0' then
          stan <= s0;
       elsif (rising_edge(clk)) then
          case stan is
             when s0=>
                if we = '1' then
                   stan <= s0;
                else
                   stan <= s1;
                end if;
             when s1=>
                if we = '1' then
                   stan <= s2;
                else
```

```
                stan <= s1;
            end if;
        when s2=>
            if we = '1' then
                stan <= s3;
            else
                stan <= s1;
            end if;
        when s3 =>
            if we = '1' then
                stan <= s0;
            else
                stan <= s4;
            end if;
        when s4 =>
            if we = '1' then
                stan <= s0;
            else
                stan <= s1;
            end if;
      end case;
    end if;
  end process;

  -- Output depends solely on the current state
  process (stan)
  begin
    case stan is
      when s0 =>
        wy <= '0';
        stany <= "000";
      when s1 =>
        wy <= '0';
        stany <= "001";
      when s2 =>
        wy <= '0';
        stany <= "010";
      when s3 =>
        wy <= '0';
        stany <= "011";
      when s4 =>
        wy <= '1';
        stany <= "100";
    end case;
  end process;

end det_0110;
```

W oprogramowaniu Quartus II można też zaprojektować automat synchroniczny, korzystając z dedykowanego do tych celów edytora (*State Machine Editor*), należy wówczas utworzyć nowy plik automatu (*File > New > State Machine File*). W celu wprowadzenia projektu do systemu Quartus posługujemy się opisem automatu w postaci grafu, najwygodniej jest użyć opcji *State Machine Wizard* (*Tools > State Machine Wizard* lub przycisk z lewej strony pola

roboczego). Po narysowaniu grafu można utworzyć plik opisu automatu w jednym z trzech języków opisu sprzętu (*Tools* > *Generate HDL File* lub przycisk z lewej strony pola roboczego). Graf utworzony w celu rozwiązania prezentowanego przykładu przedstawiono na rys. 7.8.



Rys. 7.8. Graf automatu utworzony w systemie projektowym Quartus II

7.5. Przygotowanie do zajęć

1. Obowiązuje znajomość działania bramek, przerzutników i minimalizacji funkcji logicznych metodą tablic Karnaugh'a.
2. Należy wyznaczyć sekwencję bitów do detekcji na podstawie numeru indeksu dowolnej osoby wybranej z grupy laboratoryjnej (max. trzyosobowej). W tym celu należy liczbę utworzoną z dwóch najmniej znaczących cyfr numeru indeksu zapisać w naturalnym kodzie binarnym i wziąć z niej pięć najmłodszych bitów. Na przykład dla numeru indeksu 1252**85** zapisujemy liczbę **85** binarnie jako 10**10101**b i przyjmujemy jako sekwencję zadaną pięć najmłodszych bitów, czyli **10101**.
3. Narysować graf układu detekcji zadanej sekwencji pięciu bitów, które pojawiają się na wejściu danych „*we*". Układ ma mieć wyjściu „wy" stan wysoki po wystąpieniu zadanej sekwencji. We wszystkich innych przypadkach wyjście ma mieć stan niski.
4. Zaprojektować automat synchroniczny z wykorzystaniem przerzutników D i JK oraz opisać go w języku VHDL.
5. Utworzyć schemat do testowania przygotowanych układów. W tym celu można wykorzystać projekt *automat*, który znajduje się w folderze razem z instrukcją laboratoryjną.

7.6. Program ćwiczenia

Należy sprawdzić działanie zaprojektowanych układów wykorzystując zestaw uruchomieniowy. Ewentualne błędy należy wykryć i usunąć.

7.7. Opracowanie wyników

Na zajęcia należy przynieść projekty zrealizowanych automatów. Działanie układów zostanie sprawdzone na zajęciach.

7.8. Pytania kontrolne

1. Wyjaśnić pojęcie synchroniczny układ sekwencyjny.
2. Przerzutniki D, T, JK – symbole, opis słowny działania, tablice przejść i wymuszeń.
3. Omówić proces projektowania automatów synchronicznych z wykorzystaniem przerzutników.

7.9. Literatura

[5] Skorupski A.: *Podstawy techniki cyfrowej*. Warszawa 2001.
[6] Pawłowski M., Skorupski A.: *Projektowanie złożonych układów cyfrowych*. Warszawa 2010.
[7] Nieznański J.: *Technika mikroprocesorowa.* niepublikowane materiały z wykładu.
[8] Praca zbiorowa, red. J. Kalisz: *Język VHDL w praktyce*. Warszawa: WKŁ 2002, ISBN 83-206-1440-6.

8. Wprowadzenie do środowisk: WinAVR, AVRStudio, VMLAB, HAPSIM i obsługa portów mikrokontrolera ATmega128

8.1. Cel ćwiczenia

Celem ćwiczenie jest zapoznanie się z omawianymi środowiskami, realizacja oprogramowania obsługi portów mikrokontrolera oraz utrwalenie wiadomości dotyczących wykorzystania operatorów bitowych w języku C.

8.2. Wprowadzenie

W niniejszym opracowaniu przedstawiono wybrane czynności dotyczące konfiguracji i obsługi omawianych środowisk. Do instrukcji dołączono projekty zrealizowane w omawianych środowiskach.

Podczas zajęć laboratoryjnych wykorzystywane są darmowe środowiska, które należy pobrać i zainstalować:

— Pakiet WinAVR zawierający kompilator AVR-GCC dla Windows:
http://sourceforge.net/projects/winavr/files/
— Debugger, firmy ATMEL – AVRStudio (najnowsza wersja wymaga rejestracji):
http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=2725
— Symulator HAPSIM:
http://www.helmix.at/hapsim/
— Symulator Visual Micro Lab (VMLAB):
http://www.amctools.com/download.htm

8.3. Wybrane zagadnienia z języka C

Poniżej przedstawiono wybrane aspekty niezbędne w realizacji kodów źródłowych w języku C [4]. Opisano wykorzystywane w projekcie makra z biblioteki avr-libc [5]. Należy podkreślić, iż biblioteka avr-libc **nie** jest zgodna ze standardem ANSI C (opisanym np. w [4]), jednak jest ona zalecana do stosowania podczas programowania procesorów AVR z wykorzystaniem kompilatora AVR-GCC. Biblioteka ta zapewnia dużą szybkość działania programów i mały rozmiar kodu wynikowego.

Operatory zgodne ze standardem ANSI-C [4]:

— Operatory relacji: >, >=, <, <=, ==, !=,
— Operatory logiczne: iloczynu logicznego – &&, sumy logicznej – ||, negacji logicznej – !,
— Operatory bitowe: iloczynu – &, sumy – |, XOR – ^, przesunięcia w lewo – <<, przesunięcia w prawo – >>, negacji – ~.

Tabela 8.1

Tablice prawdy dla poszczególnych operacji logicznych

p	*q*	NOT *p*	NOT *q*	*p* AND *q*	*p* OR *q*	*p* XOR *q*
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	1	0

Często stosowany operator bitowy z biblioteki avr-libc – `_BV` (ang. *byte value*) jest zdefiniowany następująco [5]: `#define _BV(bit) (1<<(bit))`. Konwertuje on adres bitu na odpowiadającą mu wartość bajtu, Przykłady:

_BV(**2**) = 0b00000**1**00 = 0×04 (zapis 0**b**X – jest zapisem binarnym specyficznym dla biblioteki avr-libc),

~_BV(**1**) = 0b11111**0**1 = 0×FD (zapis 0**x**FD jest zapisem szesnastkowym i jest zgodny ze standardem ANSI-C).

Jedną z podstawowych umiejętności, doskonaloną podczas prowadzenie zajęć laboratoryjnych, jest wprawne wykorzystywanie operacji logicznych**.** Stosowanie bitowych operatorów logicznych pozwala selektywnie pozyskać lub zmodyfikować stan logiczny bitu lub stany logiczne bitów w rozpatrywanej jednostce alokacji (np. bajcie). Modyfikacja wybranych bitów w dowolnej jednostce alokacji polega na nadpisaniu dla nich wysokiego lub niskiego stanu logicznego albo ich negacji. Przykłady:

— Ustawienie stanu wysokiego dla bitu o adresie 3 w bajcie (trzy alternatywne rozwiązania):
  $0bX_7X_6X_5X_4$**1** $X_2X_1X_0 = 0bX_7X_6X_5X_4X_3X_2X_1X_0$ | 0b0000**1**000
  $0bX_7X_6X_5X_4$**1** $X_2X_1X_0 = 0bX_7X_6X_5X_4X_3X_2X_1X_0$ | _BV(**3**)
  $0bX_7X_6X_5X_4X_3X_2X_1X_0$ |= _BV(**3**)

— Ustawienie stanu niskiego dla bitu o adresie 0 w bajcie (trzy alternatywne rozwiązania):
  $0bX_7X_6X_5X_4X_3X_2X_1$**0** $= 0bX_7X_6X_5X_4X_3X_2X_1X_0$ **&** 0b1111111**0**
  $0bX_7X_6X_5X_4X_3X_2X_1$**0** $= 0bX_7X_6X_5X_4X_3X_2X_1X_0$ **&** ~_BV(**0**)
  $0bX_7X_6X_5X_4X_3X_2X_1X_0$ **&** = ~_BV(**0**)

— Zanegowanie bitu o adresie 5 w bajcie (trzy alternatywne rozwiązania):
  $0bX_7X_6\overline{X_5}X_4X_3X_2X_1X_0 = 0bX_7X_6X_5X_4X_3X_2X_1X_0$ ^ 0b00**1**00000
  $0bX_7X_6\overline{X_5}X_4X_3X_2X_1X_0 = 0bX_7X_6X_5X_4X_3X_2X_1X_0$ ^ _BV(**5**)
  $0bX_7X_6X_5X_4X_3X_2X_1X_0$ ^ = _BV(**5**)

— Zanegowanie bitowe bajtu:
  ~$0bX_7X_6X_5X_4X_3X_2X_1X_0$ = $0b\overline{X_7}\,\overline{X_6}\,\overline{X_5}\,\overline{X_4}\,\overline{X_3}\,\overline{X_2}\,\overline{X_1}\,\overline{X_0}$

— Zanegowanie logiczne bajtu:
  !$0bX_7X_6X_5X_4X_3X_2X_1X_0$ = 0b**0** = 0×**0** – jeżeli dowolny X_i od X_0 do X_7 ma stan logiczny „**1**” (wartość bajtu jest **różna od zera**)
  = 0b**1** = 0×**1** – jeżeli każdy X_i od X_0 do X_7 ma stan logiczny „**0**” (wartość bajtu **wynosi zero**)

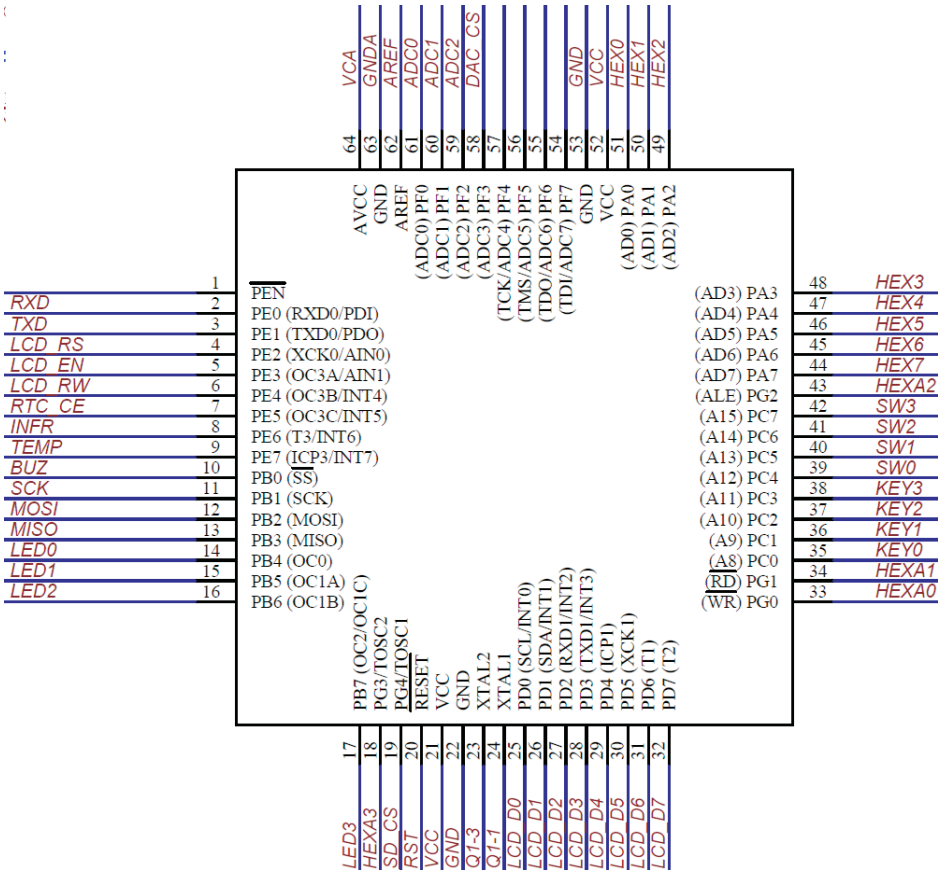
— Ustawienie stanu wysokiego „1” jednocześnie dla bitów o adresach 3, 2, 1 i 0 w bajcie (cztery alternatywne rozwiązania):
  $0bX_7X_6X_5X_4$**1111** $= 0bX_7X_6X_5X_4X_3X_2X_1X_0$ | 0b0000**1111**
  $0bX_7X_6X_5X_4X_3X_2X_1X_0$ |= _BV(**3**)| _BV(**2**)| _BV(**1**)| _BV(**0**)
  0×Y**F** = 0×YX | 0x0**F**

0×YX | = 0×0**F**

— Ustawienie stanu niskiego „0” jednocześnie dla bitów o adresach 7, 6, 5 i 4 w bajcie (pięć alternatywnych rozwiązań):
0b**0000**$X_3X_2X_1X_0$ = 0b$X_7X_6X_5X_4X_3X_2X_1X_0$ **&** 0b**0000**1111
0b$X_7X_6X_5X_4X_3X_2X_1X_0$ **&= ~(** _BV(**7**)| _BV(**6**)| _BV(**5**)| _BV(**4**))
0b$X_7X_6X_5X_4X_3X_2X_1X_0$ **&= (**~ _BV(**7**))&(~_BV(**6**))&(~ _BV(**5**))&**(**~ _BV(**4**))
0x**0**X = 0xYX **&** 0x**0**F
0xYX **&** = 0×**0**F

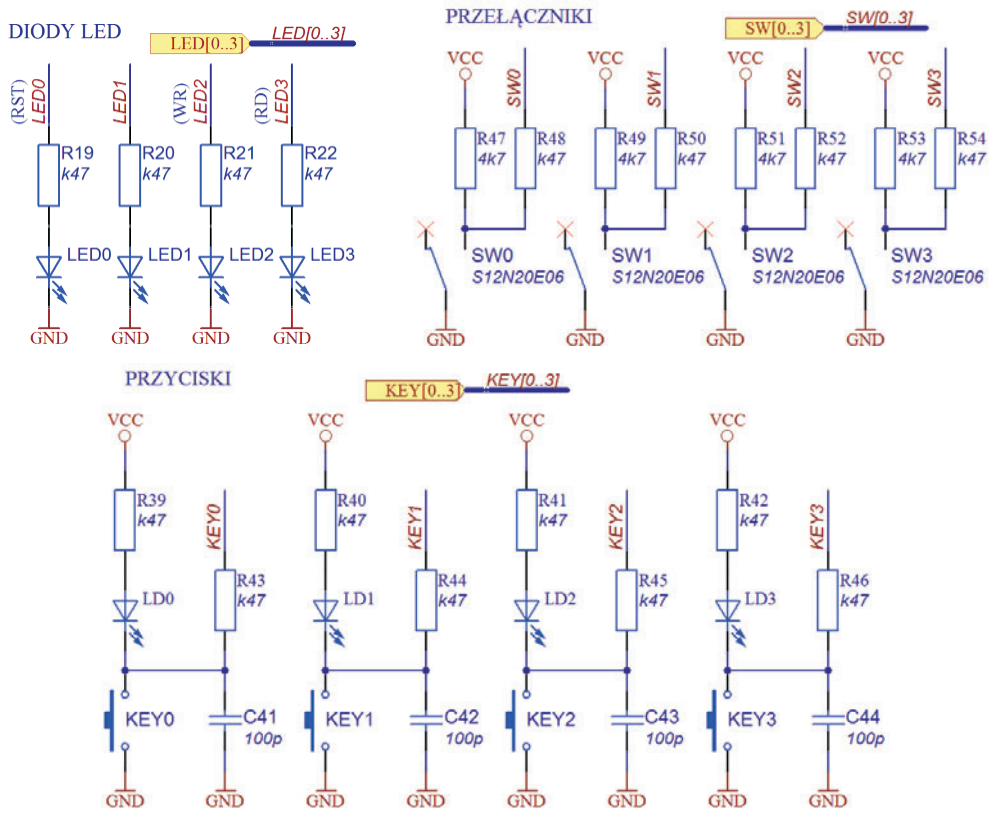
8.4. Uwarunkowania sprzętowe

Na rysunku 8.1 przedstawiono fragment schematu układu dydaktycznego AVR_edu prezentujący wyprowadzenia portów mikrokontrolera ATmega128 i etykietami sygnałów peryferyjnych w tym z: poszczególnymi diodami LEDx, przełącznikami SWx oraz przyciskami KEYx.



Rys. 8.1. Schemat podłączenia sygnałów wejściowych i wyjściowych do mikrokontrolera ATmega128

Na płytce dydaktycznej AVR_edu zrealizowano połączenia diod LEDx, iż są włączone, gdy na odpowiednich wyprowadzeniach wyjściowych o etykietach od LED0 do LED3 pojawi się **wysoki** stan logiczny (rys. 8.2). W przypadku naciśnięcia przycisków zostanie wprowadzony **niski** stan logiczny na odpowiednich wejściach mikrokontrolera (o etykietach od KEY0 do KEY3). Natomiast przełączniki mają oba stany stabilne, w położeniu bliżej krawędzi płytki wprowadzany będzie **wysoki** stan logiczny, a w przeciwnym **niski** stan logiczny.



Rys. 8.2. Fragmenty schematu prezentujące sposób podłączenia diod LED (LED), przycisków (KEY) i przełączników (SW)

Szczegółowy opis realizacji sprzętowej i programowej obsługi portów mikrokontrolera AVR można znaleźć np. w [6]. W niniejszym opracowaniu przedstawiono tylko główne aspekty dotyczące budowy i konfiguracji wyprowadzeń ogólnego przeznaczenia mikrokontrolera. Budowa portu wejścia/wyjścia została przedstawiona na rys. 8.3.

Omawiany schemat (rys. 8.3) zawiera szereg buforów trójstanowych, dla których stan wyjścia równy jest stanowi wejścia w przypadku podania na jego wejście zezwalające stanu aktywnego, w przeciwnym przypadku bufor przyjmuje na wyjściu stan wysokiej impedancji. W stanie wysokiej impedancji bufor nie stanowi obciążenie dla linii wyjściowej (nie wywołuje na niej ani stanu niskiego ani stanu wysokiego).

Z każdym portem x mikrokontrolera związane są trzy rejestry DDRx, PORTx, PINx, do których można zapisywać informacje oraz odczytywać z nich informacje.

Rejestr DDRx odpowiada za konfigurację portu x. Jeżeli pod określony adres bitu w tym rejestrze wpiszemy wysoki stan logiczny wówczas wskazane wyprowadzenie będzie wyjściem mikrokontrolera, w przeciwnym razie będzie wejściem. Na przykład, jeżeli do rejestru DDRA zostanie wpisana wartość 0x0F wówczas wyprowadzenia od PA4 do PA7 będą wejściami natomiast wyprowadzenia od PA0 do PA3 będą wyjściami.

Rejestr PORTx umożliwia ustawianie stanów w przypadku wyprowadzeń wyjściowych. Jeżeli dane wyprowadzenie jest ustawiony, jako wejściowe, wówczas wpisanie stanu wysokiego do rejestru PORTx skutkuje aktywacją rezystora podciągającego wejście do napięcia zasilania (wartość rezystancji tego rezystora zawiera się w granicach od 20kΩ do 50kΩ). Dla omawianego przypadku, jeżeli ponadto do rejestru PORTA zostanie wpisana wartość 0×AA to uzyskamy dla wyprowadzeń PA1 i PA3 na wyjściu stan wysoki, dla wyprowadzeń PA0 i PA2 na wyjściu stan niski, a dla pinów wejściowych PA5 i PA7 włączenie rezystorów pod-

ciągających. Warto zwrócić uwagę, iż budowa portu (rys. 83) umożliwia odczyt poprzednio wpisanej wartości do rejestrów DDRx oraz PORTx.

Rejestr PINx umożliwia odczyt stanów z wyprowadzeń wejściowych. W omawianym przykładzie odczyt rejestru PINA skutkowałby: dla wyprowadzeń od PA0 do PA3 wartości wcześniej wystawione na wyjścia (dla wyprowadzeń PA1 i PA3 stan wysoki, dla wyprowadzeń PA0 i PA2 stan niski), dla wejść w przypadku braku podłączenia zewnętrznych źródeł sygnałów dla wyprowadzeń PA7 i PA5 stan wysoki (aktywne rezystory podciągające), natomiast dla pinów PA6 i PA4 stan nieokreślony (taką ewentualność należy eliminować, stosując zewnętrzne rezystory definiujące domyślne stany logiczne). W przypadku wymuszenia przez zewnętrzne urządzenia stanów logicznych dla wyprowadzeń wejściowych (od PA4 do PA7) ich kopia zostanie odczytana poprzez rejestr PINA.

W tabeli 8.2 przedstawiono wpływ ustawień rejestrów na konfigurację wyprowadzeń ogólnego przeznaczenia mikrokontrolera Atmega128 [6].

Tabela 8.2

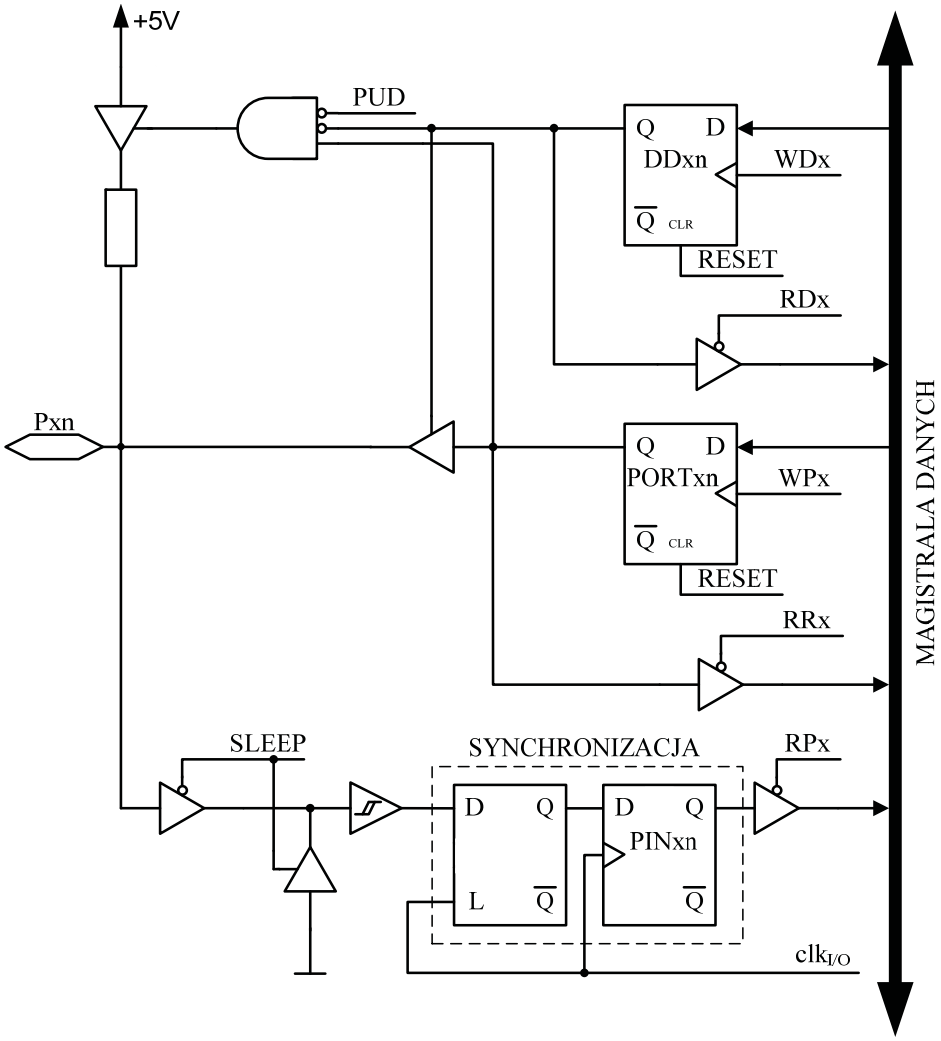
Konfiguracja wyprowadzeń mikrokontrolera ATMega128 [6]

DDxn	PORTxn	PUD	I/O	Wewnętrzny rezystor podciągający	Komentarz
0	0	X	wejście	nie	Stan wysokiej impedancji
0	1	0	wejście	tak	Utworzy się dzielnik rezystancyjny w przypadku zastosowania zewnętrznego rezystora definiującego stan niski
0	1	1	wejście	nie	Stan wysokiej impedancji
1	0	X	wyjście	nie	Wyjście w stanie niskim
1	1	X	wyjście	nie	Wyjście w stanie wysokim

Przerzutniki typu Latch i D (rys. 8.3) służą do synchronizacji sygnału wejściowego z sygnałem zegara systemowego. Ponadto eliminują one wpływ narastającego zbocza zegarowego na zmianę stanu sygnału wejściowego. Wprowadzają one opóźnienie od 0,5 do 1,5 okresu przebiegu zegara systemowego.

Sygnał sterujący SLEEP przyjmuje stan wysoki w trybach oszczędzania energii mikrokontrolera. Umożliwia on oszczędzanie energii w przypadku niepodłączonych wejść lub gdy wartość napięcia na wejściach wynosi około połowy napięcia zasilania. W przypadku stanu aktywnego dla sygnału SLEEP wyprowadzenia zostają odłączone od przerzutników Schmitta (charakteryzujących się pętlą histerezy) i jednocześnie zostaje zdefiniowany stan niski na ich wejściach.

Wyprowadzenia portów pełnią także wachlarz alternatywnych funkcji na przykład: interfejsów komunikacyjnych, wejść wbudowanego przetwornika analogowo-cyfrowego, przerwań zewnętrznych.



Rys. 8.3. Budowa wejścia/wyjścia ogólnego przeznaczenia w ATmega128 [6]. PUD – dezaktywacja rezystorów podciągających, SLEEP – sygnał generowany w trybach oszczędzania energii, $clk_{I/O}$ – sygnał zegarowy peryferii, WDx – zapis do rejestru DDRx, RDx – odczyt z rejestru DDRx, WPx – zapis do rejestru PORTx, RRx – odczyt z rejestru PORTx, RPx – odczyt stanów wyprowadzeń PINx

8.5. Opis projektu

1. Działanie projektu ma polegać na włączaniu się diod LEDx (gdzie x ∈ <0÷3>) wyłącznie w przypadku, gdy na odpowiadającym im wejściach SWx występuje wysoki stan logiczny (wejściu SW0 odpowiada dioda LED0 itd.). Założenie to spełnia pierwsza, druga, trzecia i czwarta realizacja kodu źródłowego zamieszczona w pliku main.c.
2. W przypadku zmiany stanu na dowolnym wejściu SWx następuje tylko i wyłącznie zmiana stanu wyjścia sterującego odpowiednią diodą LEDx (niedopuszczalne jest sterowanie pozostałymi wyprowadzeniami portu). Założenie to spełnia druga, trzecia i czwarta realizacja kodu źródłowego zamieszczona w pliku main.c.
3. Poszczególne grupy urządzeń (LED[3..0], KEY[3..0] i SW[3..0]) podłączone są w sposób dowolny do jednego z portów mikrokontrolera (jest to oczywiście zgodne z rys. 8.1). Projekt ma zapewnić przenośność kodu źródłowego main.c na inne procesory ATmega oraz inne układy połączeń spełniające powyższe założenia. Założenie to spełnia trzecia i czwarta realizacja kodu źródłowego zamieszczona w pliku main.c.

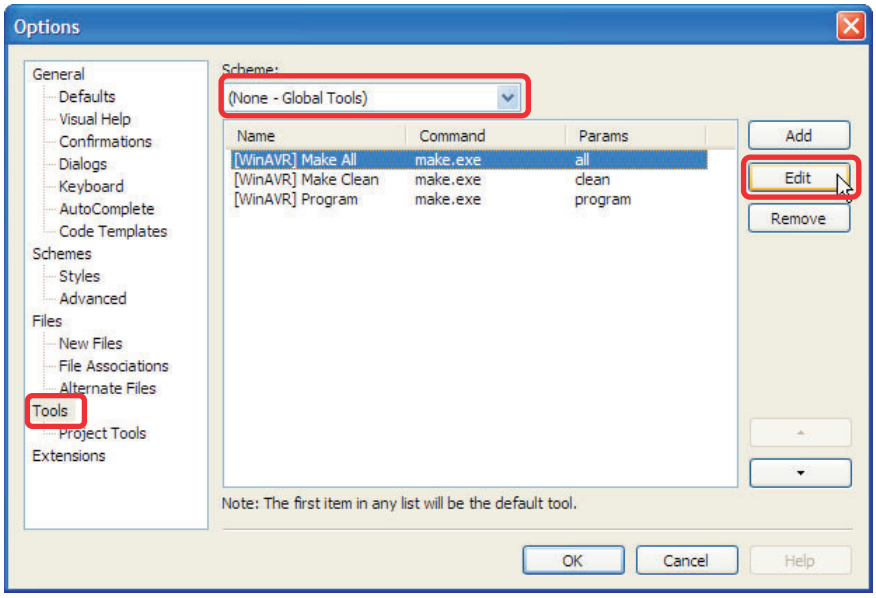
Aby spełnić założenia projektowe należy w przypadku wprowadzenia stanu wysokiego na wejście lub wejścia (o etykietach od SW0 do SW3) wystawić **wyłącznie na odpowiednie** wyjście lub wyjścia (o etykietach od LED0 do LED3) stan logiczny 1. Innymi słowy należy zrealizować „programowe połączenia" pomiędzy sygnałami LED0 i SW0, LED1 i SW1, LED2 i SW2 oraz LED3 i SW3.

8.6. Pakiet WinAVR i tworzenie projektu oprogramowania

WinAVR zawiera między innymi: darmowy kompilator AVR-GCC oraz edytor kodów źródłowych (ang. *Programmer's Notepad*).

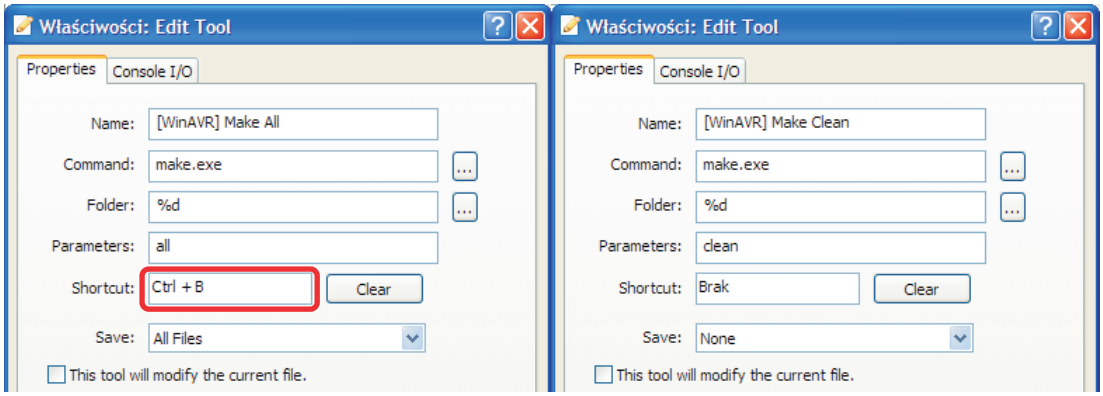
8.6.1. Konfiguracja środowiska WinAVR [1]

Aby skompilować projekt lub usunąć pliki wynikowe można utworzyć, posługując się notatnikiem odpowiednie pliki wsadowe (rozszerzenie .bat – [2], część 1, str. 38). Zalecanym sposobem kompilacji projektu na zajęciach laboratoryjnych jest wywoływanie polecenia kompilatora – *make.exe* z parametrem *all* bezpośrednio za pomocą edytora *Programmer's Notepad*. Dodatkowo można wprowadzić skrót klawiszowy dla tej czynności (rys. 8.5). W tym celu wybieramy zakładkę *Tools -> Options*, a następnie *Tools* i (*None – Global Tools*) (rys. 8.4).



Rys. 8.4. Dodawanie, usuwanie i edycja narzędzi wykorzystywanych przez *Programmer's Notepad*

Aby skonfigurować wybrane polecenie, aktywujemy przycisk *Edit* (rys. 8.4). Uzyskujemy okno dialogowe, które m. in. umożliwia dodawanie skrótów klawiszowych (rys. 8.5).



Rys. 8.5. Konfiguracja poleceń kompilatora wywoływanych za pomocą edytora *Programmer's Notepad*

8.6.2. Edycja pliku makefile

Bieżący punkt został opisany na podstawie [1]–[3]. Polecenie *make* jest interpreterem informacji zapisanych w pliku o nazwie makefile (bez rozszerzenia) znajdującym się w bieżącym katalogu. Do edycji plików makefile można wykorzystywać program *MFile*, który jest zawarty w pakiecie WinAVR. Edycja pliku makefile za pomocą wymienionego programu sprowadza się do wybrania z rozwijanych list następujących parametrów:

— nazwa pliku źródłowego, który zawiera funkcję `main`,
— nazwy procesora – `MCU type`,
— format pliku wynikowego,
— typu optymalizacji,
— opcji dla debuggera,
— odmiany języka C,
— nazw plików źródłowych napisanych w języku C, C++,
— nazw plików źródłowych napisanych w asemblerze.

Uzyskany makefile jest bardzo bogaty w komentarze, co ułatwia jego analizę. Innym sposobem jest edytowanie gotowego makefile, który jest dołączony do WinAVR i znajduje się w podkatalogu \sample katalogu, w którym zainstalowano WinAVR.

Na zajęciach laboratoryjnych pierwszy analizowany projekt będzie zawierał plik makefile, który w przyszłości będzie kopiowany do katalogów zawierających kolejne projekty i ewentualnie zostanie poddany modyfikacji. Poniżej przedstawiono wybrane fragmenty tego pliku opatrzone komentarzem.

— Nazwa procesora:
```
# MCU name
MCU = atmega128
```

— Częstotliwość pracy procesora:
```
   # Processor frequency
F_CPU = 16000000
```

— Format pliku wynikowego:
```
# Output format. (can be srec, ihex, binary)
FORMAT = ihex
```

— Nazwa pliku źródłowego, zawierającego funkcję main, a tym samym plików wynikowych:
```
# Target file name (without extension).
TARGET = main
```

— Nazwy plików źródłowych napisanych w języku C (pierwszy projekt zawiera wyłącznie plik `main.c`):

```
# List C source files here. (C dependencies are automatically gener-
ated.)
SRC = $(TARGET).c
```

— Nazwy plików źródłowych napisanych w asemblerze (projekt nie zawiera plików asemblera):

```
# List Assembler source files here.
ASRC =
```

— Typ optymalizacji. Dla pierwszego projektu wybrano optymalizację kompilacji pod względem minimalizacji rozmiaru kodu wynikowego. Poziom optymalizacji `s` aktywuje także funkcje minimalizujące czas wykonywania programu, ale jednocześnie niepowiększających rozmiaru kodu wynikowego.

```
# Optimization level, can be [0, 1, 2, 3, s].
OPT = s
```

— Opcje dla debuggera. W przypadku wykorzystywania do uruchamiania projektu środowiska AVRStudio od wersji 4.10 zaleca się ustawienie opcji `dwarf-2`. Podczas uruchamiania projektu w środowisku AVRStudio wykorzystuje się wówczas plik obiektowy z rozszerzeniem `.elf`. Alternatywnym podejściem jest ustawienie opcji dla debuggera `stabs` i generacja pliku obiektowego o rozszerzeniu `.cof [1]`.

```
# Debugging format
DEBUG = dwarf-2
```

— Odmiana języka C. W celu pełnego wykorzystania możliwości kompilatora AVR-GCC zaleca się ustawienie opcji `gnu99`.

```
# Compiler flag to set the C Standard level.
CSTANDARD = -std=gnu99
```

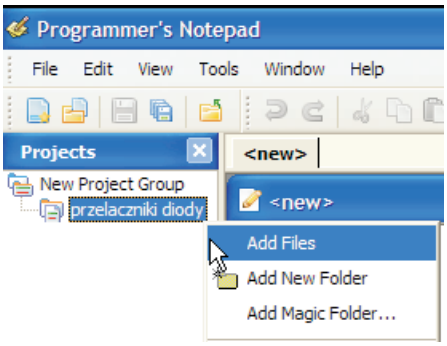
— Przekazanie nazwy procesora, częstotliwości taktowania.

```
# Combine all necessary flags and optional flags.
# Add target processor to flags.
ALL_CFLAGS=-mmcu=$(MCU) -I. $(CFLAGS) $(GENDEPFLAGS) -DF_CPU=$(F_CPU)
```

Więcej informacji na temat pliku makefile można znaleźć np. w [3].

8.6.3. Tworzenie projektu w środowisku WinAVR

Należy wybrać z menu *File -> New -> Project*. W pole *Name* należy wpisać nazwę projektu np. *Przelaczniki_diody*. Projekt należy utworzyć na partycji D:\TM\GrupaXY\Przelaczniki. Do wymienionego katalogu należy skopiować plik makefile, a następnie dodać go do projektu, wybierając z menu kontekstowego dla tworzonego projektu *Add Files* (rys. 8.6).



Rys. 8.6. Dodawanie plików do projektu

8.6.4. Tworzenie i opis pliku nagłówkowego main.h

Należy wybierć menu *File -> New -> C/C++* i utworzyć plik nagłówkowy o nazwie main.h. Po zapisaniu (*File -> Save As*) pod nazwą main.h w analogiczny sposób jak poprzednio należy dodać utworzony plik do projektu (rys. 8.6).

Omawiany plik nagłówkowy main.h rozpoczyna się od instrukcji preprocesora języka C – `#include`, oznaczającego dołączenie zawartości wskazywanego pliku ([4], str. 125), w tym przypadku `avr/io.h`. Nazwa pliku zamieszczona jest pomiędzy znakami <>, co wskazuje, iż plik ten znajduje się w zasobach bibliotecznych, w odróżnieniu, gdy nazwa pliku podawana jest w cudzysłowie, gdy znajduje się on w tym samym katalogu, co plik źródłowy. Dokumentację do biblioteki avr-libc można znaleźć w [5], aby dotrzeć do dokumentacji avr/io.h należy wybrać *Library Reference* i *<avr/io.h>: AVR device-specific IO definitions*, a następnie lub bezpośrednio *<avr/sfr_defs.h>: Special function registers*.

Tworząc projekt, zaleca się posługiwać definicjami, które pozwalają przypisać nazwy funkcjonalne odpowiadające danym urządzeniom wejścia/wyjścia do odpowiednich portów mikrokontrolera (np. `#define NAZWA_URZĄDZENIA PORTx`). W celu zapewnienia w jak najszerszym zakresie przenośności projektu dla platform sprzętowych z mikrkontrolerami AVR, należy umiejętnie wykorzystywać definicje portów i poszczególnych wyprowadzeń we wszystkich plikach źródłowych. Poniżej przedstawiono zawartość pliku main.h.

```
#include <avr/io.h>
/* dołączenie pliku nagłówkowego avr/io.h z biblioteki avr-libc pozwala na
   korzystanie z nazw rejestrów i flag bitowych oraz poniższych makr */
#include <util/delay.h>
/* dołączenie pliku nagłówkowego util/delay.h z biblioteki avr-libc pozwala
   na korzystanie z funkcji _delay_ms (maksymalne możliwe opóźnienie z
   maksymalną rozdzielczością to 262.14 ms / F_CPU w MHz, ponad tą wartość
   rozdzielczość zmniejsza się do 0.1 ms, z możliwym do uzyskania
   opóźnieniem 6.5535 s i _delay_us dla, której maksymalne opóźnienie
   wynosi 768 us / F_CPU w MHz, przy czym gdy wartość ta jest większa
   automatycznie wywoływana jest funkcja _delay_ms */

//przyporządkowanie diodom LED, przełącznikom SW i przyciskom KEY odpowied-
nich portów mikrokontrolera
#define PORTLED      PORTB   //diody LED podłączono do portu B
#define DIRLED       DDRB    //diody LED podłączono do portu B
#define PINKEY       PINC    //przyciski KEY podłączono do portu C
#define DIRKEY       DDRC    //przyciski KEY podłączono do portu C
#define PINSWITCH    PINC    //przełączniki SW podłączono do portu C
#define DIRSWITCH    DDRC    //przełączniki SW podłączono do portu C

/*   przyporządkowanie   poszczególnym   diodom   LED,   przełącznikom   SW
i przyciskom KEY odpowiednich pinów z określonych portów mikrokontrolera */
#define LED0 PB4//dioda LED0 podłączona jest do pinu PB4
#define LED1 PB5//dioda LED1 podłączona jest do pinu PB5
#define LED2 PB6//dioda LED2 podłączona jest do pinu PB6
#define LED3 PB7//dioda LED3 podłączona jest do pinu PB7
#define KEY0 PC0//przycisk KEY0 podłączony jest do pinu PC0
#define KEY1 PC1//przycisk KEY1 podłączony jest do pinu PC1
#define KEY2 PC2//przycisk KEY2 podłączony jest do pinu PC2
#define KEY3 PC3//przycisk KEY3 podłączony jest do pinu PC3
#define SW0      PC4//przełącznik SW0 podłączony jest do pinu PC4
#define SW1      PC5//przełącznik SW1 podłączony jest do pinu PC5
#define SW2      PC6//przełącznik SW2 podłączony jest do pinu PC6
#define SW3      PC7//przełącznik SW3 podłączony jest do pinu PC7
```

8.6.5. Tworzenie i opis pliku źródłowego main.c

W pliku main.c zastosowano kompilację warunkową (jest ona zgodna ze standardem ANSI-C), która polega na zdefiniowaniu stałej, a następnie w zależności od jej wartości opcjonalnej kompilacji fragmentów kodu źródłowego.

W funkcji `init_PINs` wykorzystano bitowe operacje logiczne do selektywnego definiowania kierunkowości wykorzystywanych w projekcie wyprowadzeń. Poniżej zostanie opisany jeden z zapisów zawartych w wymienionej funkcji. W omawianym przypadku diody LED0 do LED3 zostały podłączone do wyprowadzeń portu B o numerach od 4 do 7 (rys. 8.1). Poniżej (w ramach utrwalenia wiadomości) przedstawiono sześć alternatywnych zapisów.

```
DIRLED |= ( _BV(LED0) |  _BV(LED1) |  _BV(LED2) |  _BV(LED3));
DIRLED |= (  _BV(PB4) |   _BV(PB5) |   _BV(PB6) |   _BV(PB7));
DIRLED |= (    _BV(4) |     _BV(5) |     _BV(6) |     _BV(7));
DIRLED |= (0b00010000 | 0b00100000 | 0b01000000 | 0b10000000);
DIRLED |= 0b11110000;
DIRLED |= 0xF0;
```

Zastosowanie bitowej operacji sumy logicznej z zawartością rejestru `DIRLED` (DIRB) skutkuje ustawieniem stanów wysokich tylko na najbardziej znaczących bitach w tym rejestrze (o numerach od 4 do 7). Konsekwencją omawianego fragmentu kodu jest **ustawienie kierunkowości wyprowadzeń PB4 ÷ PB7 w porcie B, jako wyjściowej i jednocześnie brak zmian kierunkowości dla pozostałych wyprowadzeń portu** (PB0 ÷ PB3). Należy zwrócić uwagę, iż zaproponowany zapis (w pliku main.c) wraz z definicjami zawartymi w pliku nagłówkowym main.h jest w pełni uniwersalny, zapewniający poprawne działanie dla dowolnego układu połączeń diod LED z wyprowadzeniami jednego portu. W przypadku zmiany układu połączeń konieczne jest tylko ponowne zdefiniowanie etykiet od LED0 do LED3 (w pliku main.h). Wartość maski bitowej `0xF0` jest wyznaczana przez preprocesor języka C, jako wynik bitowych operacji logicznych realizowanych na stałych (`_BV(LED0)| _BV(LED1) | _BV(LED2) | _BV(LED3)`). Podsumowując stosowanie analogicznych zapisów do omawianego, zapewnia uzyskanie **elastycznego kodu źródłowego** i możliwie **szybko działającego programu**.

W pierwszej wersji programu głównego zrealizowano nadpisanie portu `PORTLED` wartościami odczytywanymi z portu `PINSWITCH`. Należy zwrócić uwagę, iż w tym przypadku zrealizowano modyfikację całego bajtu w rejestrze `PORTLED`, a nie tylko żądanych bitów (w nim zawartych).

W drugiej wersji programu uzyskano sterowanie wyłącznie żądanymi bitami w porcie, wykorzystując bitowe operacje logiczne. Jednakże, podobnie jak w pierwszym rozwiązaniu, wymagane „połączenie programowe” sygnałów LED0 i SW0, LED1 i SW1, LED2 i SW2 oraz LED3 i SW3 nie zostałoby zrealizowane dla dowolnie innych układów połączeń niż przedstawiony na rys. 8.1.

W trzecim rozwiązaniu programu głównego stan każdego z przełączników testowany jest selektywnie i oddziałuje wybiórczo na włączenie/wyłączenie odpowiednich diod. Do selektywnego włączania i wyłączania poszczególnych diod wykorzystano bitowe operacje logiczne. W przypadku testowania stanu logicznego definiowanego przez poszczególne przełączniki wykorzystano makro `bit_is_set` [5]:

```
#define bit_is_set (sfr,bit)      (_SFR_BYTE(sfr) & _BV(bit)).
```

Makro to zwraca wartość zero w przypadku, gdy testowany bit w rejestrze (z przestrzeni adresowej wejścia/wyjścia) ma wartość zero i 2^{bit} (wartość różną od zera) w przypadku, gdy bit ten ma wartość jeden.

W czwartej prezentowanej wersji programu głównego wykorzystano przesuniecie bitowe w lewo, bitowe operacje logiczne i makro `bit_is_clear` [5]:

```
#define bit_is_ clear (sfr,bit)      (!(_SFR_BYTE(sfr) & _BV(bit))).
```

Makro to względem omówionego poprzednio różni się operacją negacji logicznej (!). Zwraca wartość zero w przypadku, gdy testowany bit w rejestrze (z przestrzeni adresowej wejścia/wyjścia) ma wartość jeden i jeden w przeciwnym przypadku.

Poniżej przedstawiono zawartość pliku main.c.

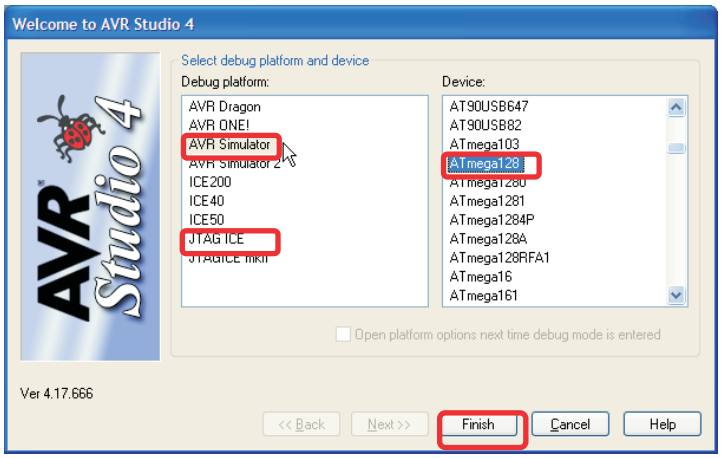
```
#include "main.h"   // dołączenie do projektu pliku nagłówkowego main.h
#define program_1_2_3_4 3 // definicja argumentu kompilacji warunkowej
int main(void);     //deklaracja funkcji main
void init_PINs(void);  //deklaracja funkcji init_PINs
//definicja funkcji init_PINs
void init_PINs(void){
    /*ustawienie odpowiednich pinów mikrokontrolera jako
    wyjść dla portu, do którego podłączono diody LED*/
    DIRLED |= (_BV(LED0)|_BV(LED1)|_BV(LED2)|_BV(LED3));
    /*ustawienie odpowiednich pinów mikrokontrolera jako
    wejść dla portu, do którego podłączono przyciski KEY*/
    DIRKEY &= ~(_BV(KEY0)|_BV(KEY1)|_BV(KEY2)|_BV(KEY3));
    /*ustawienie odpowiednich pinów mikrokontrolera jako
    wejść dla portu, do którego podłączono przełączniki SW*/
    DIRSWITCH &= ~(_BV(SW0)|_BV(SW1)|_BV(SW2)|_BV(SW3));
};
//definicja funkcji main
int main (void)
{
char temp;
    init_PINs();     //wywołanie funkcji init_PINs
    while(1)      //nieskończona pętla
    {
//realizacja kompilacji warunkowej dla wersji 1 programu głównego
#if program_1_2_3_4 == 1
       PORTLED=PINSWITCH;

//realizacja kompilacji warunkowej dla wersji 2 programu głównego
#elif program_1_2_3_4 == 2
         PORTLED&=(PINSWITCH|0x0F);
         PORTLED|=(PINSWITCH&0xF0);

//realizacja kompilacji warunkowej dla wersji 3 programu głównego
#elif program_1_2_3_4 == 3
        if(bit_is_set(PINSWITCH,SW0))    //Czy stan sygnału SW0 == 1
         {
             PORTLED|=_BV(LED0);         //włączenie diody LED0
         }
         else
         {
             PORTLED&=~_BV(LED0);        //wyłączenie diody LED0
         };
         if(bit_is_set(PINSWITCH,SW1))   //Czy stan sygnału SW1 == 1
         {
             PORTLED|=_BV(LED1);         //włączenie diody LED1
         }
         else
         {
             PORTLED&=~_BV(LED1);        //wyłączenie diody LED1
         };
         if(bit_is_set(PINSWITCH,SW2))   //Czy stan sygnału SW2 == 1
         {
             PORTLED|=_BV(LED2);         //włączenie diody LED2
         }
         else
         {
             PORTLED&=~_BV(LED2);        //wyłączenie diody LED2
         };
         if(bit_is_set(PINSWITCH,SW3))   //Czy stan sygnału SW3 == 1
         {
             PORTLED|=_BV(LED3);         //włączenie diody LED3
         }
         else
         {
             PORTLED&=~_BV(LED3);        //wyłączenie diody LED3
         };
//realizacja kompilacji warunkowej dla wersji 4 programu głównego
#else
```

```
temp=_BV(LED0)|_BV(LED1)|_BV(LED2)|_BV(LED3);
temp&=~((bit_is_clear(PINSWITCH,SW0)<<(LED0))|(bit_is_clear(PINSWITCH,SW1)<
<(LED1))|(bit_is_clear(PINSWITCH,SW2)<<(LED2))|(bit_is_clear(PINSWITCH,SW3)
<<(LED3)));
        PORTLED|=temp;
        PORTLED&=temp|~(_BV(LED0)|_BV(LED1)|_BV(LED2)|_BV(LED3));
#endif
    };
    return 0;
};
```

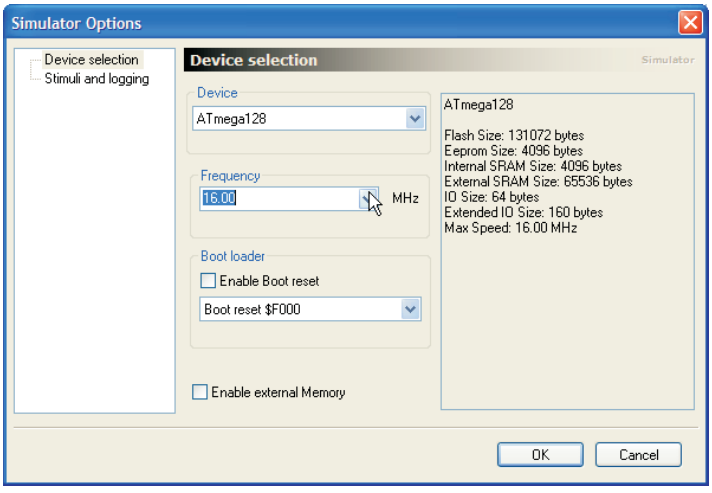
8.7. Symulacja projektu w środowisku AVRStudio

Ustawienie opcji debuggera w pliku makefile na: `DEBUG = dwarf-2` pozwala wykorzystać plik obiektowy z rozszerzeniem .elf do utworzenia projektu w środowisku AVRStudio.

Po uruchomieniu środowiska uzyskane okno dialogowe pozwala wykorzystać kreator nowego projektu lub otworzyć już istniejący. W celu skorzystania z pliku obiektowego main.elf należy wybrać opcję *Open* i wskazać ten plik. Kolejne okno dialogowe pozwala zachować tworzony projekt pod nazwą: main_elf.aps, którą zaleca się potwierdzić. Następnie należy wybrać opcję *AVR Simulator* (pozwalającą w przyszłości korzystać z dodatkowego symulatora HAPSIM) i procesor *ATmega128* (rys. 8.7) i potwierdzić wybierając *Finish*. W przyszłości w celu uruchamiania oprogramowania z wykorzystaniem płytki dydaktycznej AVR_edu należy wybrać opcję: *JTAG ICE*.



Rys. 8.7. Tworzenie projektu w środowisku AVRStudio



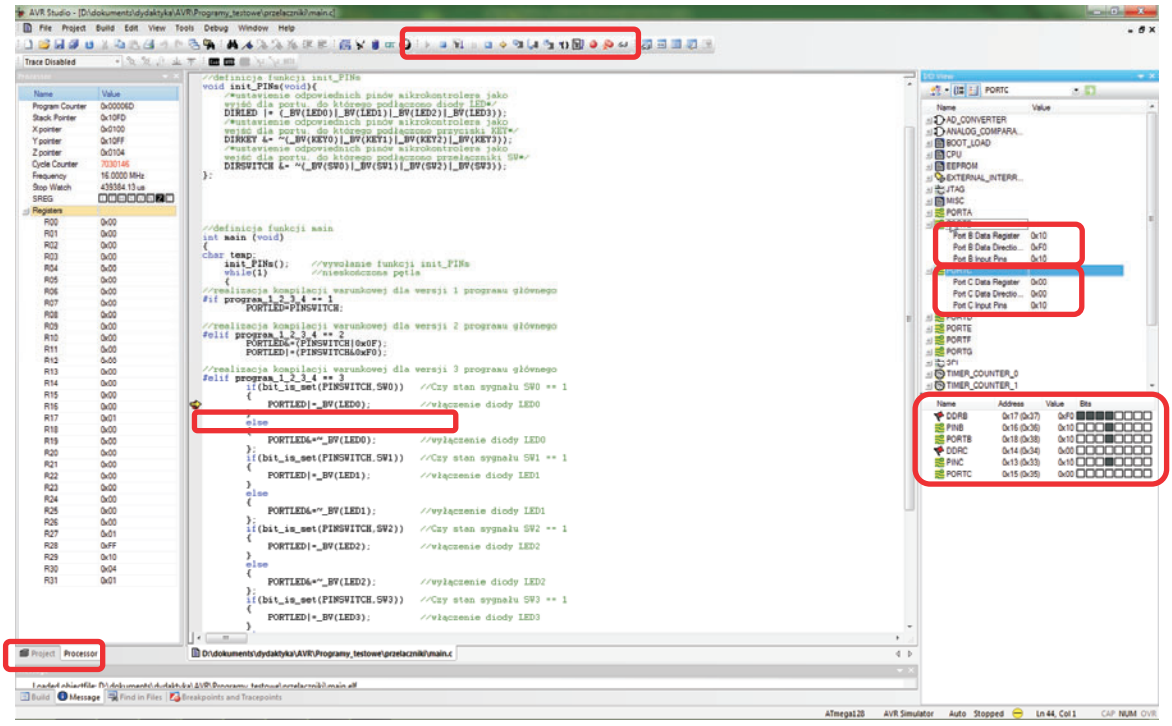
Rys. 8.8. Konfiguracja projektu w środowisku AVRStudio

Po utworzeniu projektu należy przystąpić do jego konfiguracji, wybierając skrót klawiszowy *ALT+O*. Należy zmienić częstotliwość zegara systemowego na 16 MHz (rys. 8.8).

Do obsługi symulatora przydatna jest znajomość następujących skrótów klawiszowych:

— F5 – start symulacji programu,
— Ctrl+F5 – przerwanie symulacji programu,
— Shift+F5 – reset programu,
— F9 – włączenie / wyłączenie punktu pułapki wstrzymującej działanie programu,
— ALT+F5 – start symulacji z animacją wykonywania programu,
— F10 – wykonanie jednego kroku w symulacji,
— F11 – wykonanie jednego kroku w symulacji z wejściem do ciała funkcji,
— Shift+F11 – wykonanie jednego kroku w symulacji z wyjściem z ciała funkcji,
— Alt+1 – wywołanie okna *Watch* umożliwiającego podgląd i edycję wartości zmiennych,
— Alt+4 – wywołanie okna podglądu i edycji wartości komórek pamięci.

Na rysunku 8.9 uwidoczniono wybrane zakładki i ikony dostępne w środowisku AVRStudio. W górnej części rysunku zaznaczono ikony dotyczące symulowania oprogramowania (ich skróty klawiszowe zostały w większości wymienione powyżej). W dolnej części lewej kolumny mamy do dyspozycji przełączenie się pomiędzy zakładkami *I/O View* oraz *Watch* (po wcześniejszym jej wywołaniu – *Alt+1*). W celu podglądu dwu portów jednocześnie w dolnej części lewej kolumny należy wybrać jeden z portów, a następnie wybrać drugi z jednocześnie wciśniętym przyciskiem *Ctrl*. Dla prawej kolumny dostępny jest podgląd projektu lub rejestrów procesora. W kodzie programu umieszczono pułapkę dla linii `PORTLED|=_BV(LED0)`. Stany logiczne dla wejść od PC4 do PC7 można zmieniać po zatrzymaniu symulatora lub w trybie pracy z animacją, klikając w odpowiednie pole rejestru PINC (dolna część lewej kolumny). Przydatną podczas symulacji opcją jest możliwość podglądu kodu asemblera, po wybraniu opcji z menu *View->Disassembler*. Kolejną jest wprowadzenie zawartości np. pamięci EEPROM (której wartości inicjalizujące generuje WinAVR). Wybieramy wówczas *Debug->Up/Download Memory* i wypełniamy odpowiednio uzyskane okno dialogowe.



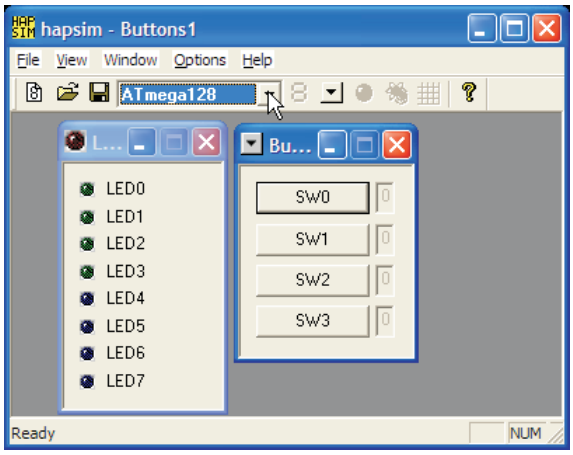
Rys. 8.9. Środowisko AVRStudio

8.8. Symulacja projektu z wykorzystaniem programu HAPSIM

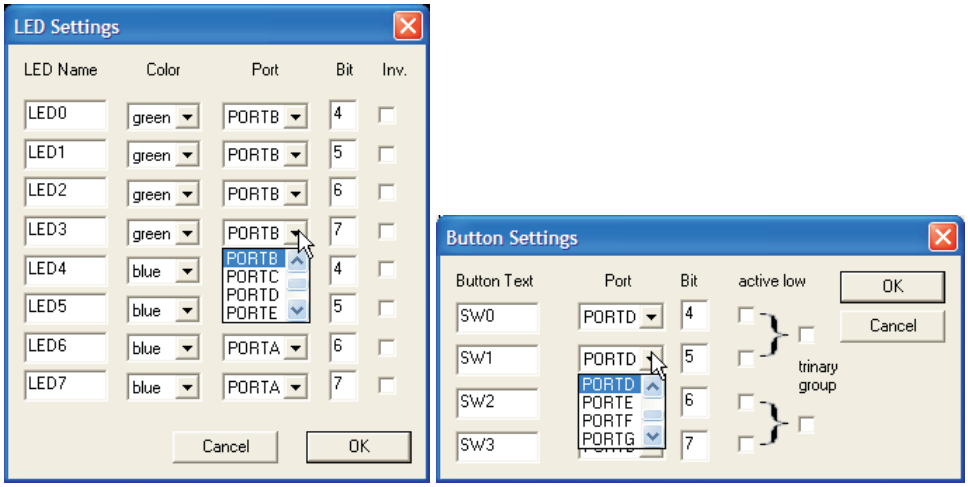
HAPSIM jest bardzo prostym do wykorzystania programem ułatwiającym symulację programów. Współpracuje on ze środowiskiem AVRStudio, w przypadku wybrania opcji *AVR Simulator* dla symulowanego projektu. Umożliwia on symulację następujących urządzeń peryferyjnych:

— 4 przycisków,
— 8 diod LED,
— wyświetlacza alfanumerycznego,
— terminala,
— klawiatury.

Konfiguracja polega na wprowadzeniu odpowiednich komponentów i wybraniu rodzaju procesora. W naszym przypadku będą to diody LED i przyciski (rys. 8.10). Następnie należy je odpowiednio skonfigurować. Należy wprowadzić odpowiednie etykiety dla diod i przycisków, wybrać kolor dla diod, port do którego są podłączone oraz numer bitu (rys. 8.11). Można także wprowadzić aktywny stan logiczny dla przełączników i diod. Wadą dostępnych przycisków jest ich wyłącznie dostępny tryb pracy jako monostabilny. Po konfiguracji peryferii w połączeniu z AVRStudio możemy przystąpić do symulacji projektu. Można zachować wprowadzoną konfigurację i w przyszłości ją przywoływać.



Rys. 8.10. Środowisko HAPSIM

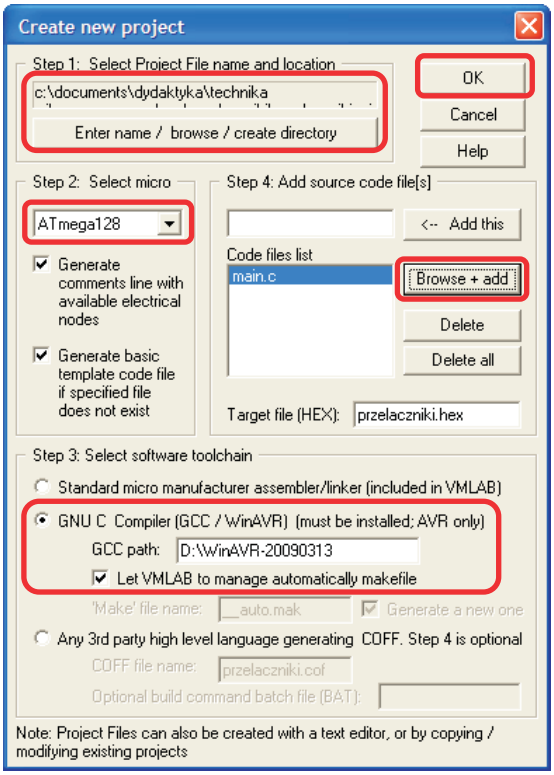


Rys. 8.11. Konfiguracja symulacji diod LED i przełączników SW w programie HAPSIM

8.9. Symulacja projektu w środowisku VMLAB

Alternatywnym (do przedstawionego powyżej) podejściem do symulacji jest wykorzystanie symulatora VMLAB (*Visual Micro Lab*). Środowisko to umożliwia, oprócz symulacji działania mikrokontrolera z oprogramowaniem, symulację całego wachlarza urządzeń peryferyjnych np.: rezystorów, kondensatorów, diod LED, klawiatury, wzmacniacza operacyjnego, komparatora, bramek logicznych, przetworników A/C i C/A, generatorów zadanych przebiegów, oscyloskopu, potencjometrów, wyświetlacza alfanumerycznego LCD itd.

W celu utworzenia projektu wybieramy *Project->New Project*. Następnie odpowiednio wypełniamy okno konfiguracyjne projektu (rys. 8.12).

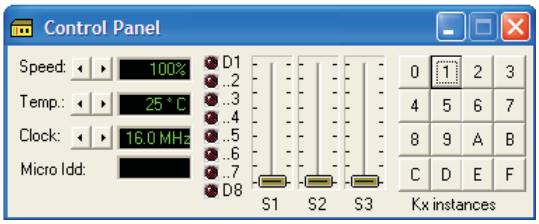


Rys. 8.12. Tworzenie projektu w środowisku VMLAB

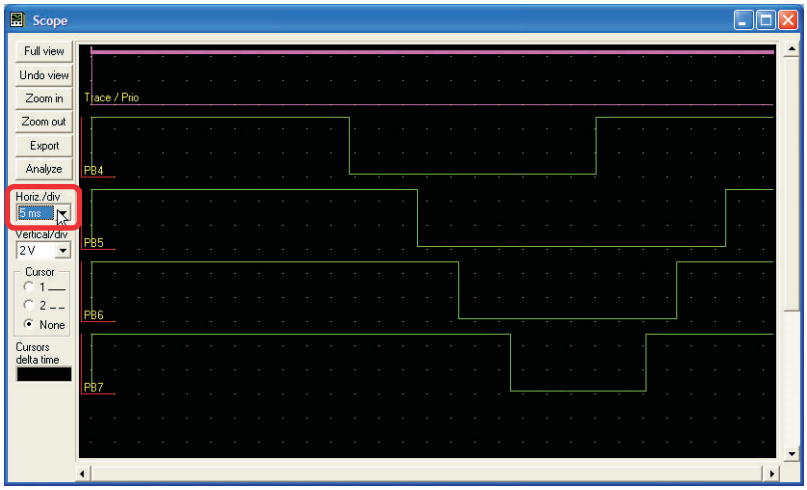
Następnie należy wybrać *View->Project File* i edytować plik projektu, aż do uzyskania analogicznej postaci do przedstawionej na rys. 8.13. Względem wygenerowanego pliku różni się on m.in. częstotliwością zegara systemowego, katalogiem środowiska WinAVR, dołączeniem definicji przełączników, przycisków oraz diod. Wprowadzono także funkcję oscyloskopu wirtualnego z podglądem wyprowadzeń wyjściowych podłączonych do diod LED. Pomoc dotyczącą dołączania do projektu poszczególnych peryferii można znaleźć w menu *Help->Contents->Hardware components*. W celu kompilacji projektu wybieramy menu *Project->Re-buid all* (skrót klawiszowy Shift+F9). Należy uaktywnić panel kontrolny (rys. 8.14) i wirtualny oscyloskop (rys. 8.15), wybierając odpowiednio menu *View->Contorl Panel* i *View->Scope*. Po kompilacji projektu i aktywacji wirtualnych urządzeń wejścia / wyjścia projekt przygotowany jest do symulacji, którą rozpoczynamy, wybierając z menu *Run->Go / Continue* (skrót klawiszowy F5).

```
; ************************************************************
; PROJECT: Przelaczniki
; AUTHOR: Artur Cichowski
; ************************************************************

; Micro + software running
; ------------------------------------------------------------
.MICRO "ATmega128"
.TOOLCHAIN "GCC"
.GCCPATH   "C:\WinAVR-20100110"
.GCCMAKE   AUTO
.TARGET    "przelaczniki.hex"
.SOURCE    "main.c"

.TRACE              ; Activate micro trace

; Following lines are optional; if not included
; Following lines are optional; if not included
; exactly these values are taken by default
; ------------------------------------------------------------
.POWER VDD=5 VSS=0  ; Power nodes
.CLOCK 16meg         ; Micro clock
.STORE 250m         ; Trace (micro+signals) storage time

; Micro nodes: RESET, AREF, PA0-PA7, PB0-PB7, PC0-PC7, PD0-PD7, PE0-PE7, PF0-PF7, PG0-PG4, TIM1OVF
; Define here the hardware around the micro
; ------------------------------------------------------------

K1 PC4 GND LATCHED  ;definicja przełącznika SW0
R1 VDD PC4 1k
K2 PC5 GND LATCHED  ;definicja przełącznika SW1
R2 VDD PC5 1k
K3 PC6 GND LATCHED  ;definicja przełącznika SW2
R3 VDD PC6 1k
K4 PC7 GND LATCHED  ;definicja przełącznika SW3
R4 VDD PC7 1k

KA PC0 GND          ;definicja przycisku KEY0
R5 VDD PC0 1k
KB PC1 GND          ;definicja przycisku KEY1
R6 VDD PC1 1k
KC PC2 GND          ;definicja przycisku KEY2
R7 VDD PC2 1k
KD PC3 GND          ;definicja przycisku KEY3
R8 VDD PC3 1k

D1 VDD net1      ;definicja diody LED0 (konieczność podłączenia anody do VCC)
R9 net1 PB4  1k
D2 VDD net2      ;definicja diody LED1 (konieczność podłączenia anody do VCC)
R10 net2 PB5 1k
D3 VDD net3      ;definicja diody LED2 (konieczność podłączenia anody do VCC)
R11 net3 PB6 1k
D4 VDD net4      ;definicja diody LED3 (konieczność podłączenia anody do VCC)
R12 net4 PB7 1k
.PLOT V(PB4)   ;właczenie przebiegu napięcia na wyjściu LED0
.PLOT V(PB5)   ;właczenie przebiegu napięcia na wyjściu LED1
.PLOT V(PB6)   ;właczenie przebiegu napięcia na wyjściu LED2
.PLOT V(PB7)   ;właczenie przebiegu napięcia na wyjściu LED3
```

Rys. 8.13. Plik projektu środowisku VMLAB



Rys. 8.14. Panel kontrolny w środowisku VMLAB



Rys. 8.15. Oscyloskop wirtualny w środowisku VMLAB

8.10. Przygotowanie do zajęć

Przygotowanie do zajęć obejmuje:

1. Przyniesienie wydrukowanej niniejszej instrukcji (jednej na grupę laboratoryjną 2–3 osobową).
2. Zainstalowanie na komputerach dostępnych poza zajęciami programów: WinAVR, AVRStudio, VMLAB i HAPSIM.
3. Zapoznanie się z operacjami bitowymi realizowanymi w języku C.
4. Zapoznanie się z obsługą portów mikrokontrolera ATmega.
5. Szczegółowe zrozumienie programu zawartego w plikach źródłowych: main.h i main.c.
6. Dla chętnych zrealizowanie samodzielne wszystkich etapów omówionych w instrukcji (gwarantuje to sprawną realizację ćwiczenia na zajęciach laboratoryjnych).

8.11. Przebieg ćwiczenia nr 8

W ramach ćwiczenia laboratoryjnego należy wykonać zadania zawarte w instrukcji z uwzględnieniem wskazówek prowadzącego.

8.12. Przebieg ćwiczenia nr 9

W opracowywanym oprogramowaniu należy zwrócić szczególną uwagę na sterowanie **wyłącznie** żądanymi wyprowadzeniami portu.

1. Opracowanie programu realizującego sekwencyjne włączanie się diod LED w kolejności: LED0, LED1, LED2, LED3, LED0, LED1 itd. Wskazówka: można wykorzystać funkcję `_delay_ms(20)` w celu generowania opóźnień po każdym włączeniu diody w trakcie testów symulacyjnych i `_delay_ms(500)` w trakcie testów z wykorzystaniem płytki dydaktycznej AVR_edu.
2. Opracowanie programu realizującego sekwencyjne włączanie się diod LED w kolejności: LED0, LED1, LED2, LED3, LED2, LED1 itd.
3. Opracowanie programu realizującego sekwencyjne włączanie się diod LED w kolejności: LED0, LED1, LED2, LED3, LED0 itd. dla niskiego stanu logicznego na wejściu SW0 oraz LED3, LED2, LED1, LED0, LED3 itd. w przeciwnym przypadku. Zmiana kierunku włączania się diod **ma być realizowana od aktualnie włączonej diody LED** (a nie zawsze od diod LED0 lub LED3).

8.13. Literatura

[1] Dokumentacja dotycząca pobierania, instalowania i konfiguracji WinAVR http://winavr.sourceforge.net/install_config_WinAVR.pdf
[2] R. Koppel: Programowanie procesorów w języku C, *Elektronika dla Wszystkich*, maj 2005 do maj 2007.
[3] A. Witkowski: Mikrokontrolery AVR programowanie w języku C – przykłady zastosowań, Katowice 2006
[4] B. W. Kernighan, D. M. Ritchie: Język ANSI C
[5] Dokumentacja do biblioteki avr-libc, katalog z WinAVR\doc\avr-libc\avr-libc-user-manual\modules.html
[6] Dokumentacja do mikrokontrolera ATmega 128 – http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf

9. System przerwań. Obsługa przycisków i wyświetlaczy siedmiosegmentowych

9.1. Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z systemem przerwań mikrokontrolera ATmega na przykładzie obsługi przycisków.

9.2. Przerwania

Podstawowe pojęcia [2, 3]:

- przerwanie – (ang. *interrupt*) przekazanie sterowania do instrukcji innej niż kolejna zapisana w programie, z zachowaniem możliwości powrotu do przerwanego programu.
- system przerwań – system przekazujący do procesora żądania przerwań (ang. *interrupt request*). Żądanie przerwania jest zgłaszane przez źródło przerwania w przypadku osiągnięcia stanu gotowości przez odpowiadające mu urządzenie.
- podprogram obsługi przerwania – kod realizujący czynności zaplanowane do wykonania po wystąpieniu odpowiedniego żądania przerwania.
- tablica wektorów przerwań – tablica zawierająca adresy podprogramów obsługi przerwań. Adres wektora przerwania przyjmuje licznik rozkazów w przypadku pojawienie się żądania przerwania.
- stos – struktura danych, w której dane zapisywane są na wierzchołek stosu i wyłącznie z wierzchołka stosu są odczytywane. Ostatnia zapisana na stos dana, jako pierwsza jest z niego odczytywana (ang. *last in first out LIFO*).

Współpraca z urządzeniami peryferyjnymi możliwa jest poprzez systematyczne (programowe) sprawdzanie stanu urządzenia lub wykorzystanie systemu przerwań. System przerwań umożliwia prawie natychmiastowe zawieszenie wykonywania bieżącego programu |i przejście do wykonania podprogramu obsługi przerwania, a po jego zrealizowaniu powrót do przerwanego programu (realizowanego wcześniej). Źródłami przerwań w mikrokontrolerze mogą być stany lub zbocza pojawiające się na dedykowanych wejściach procesora (przerwania zewnętrzne), a także zdarzenia występujące w wbudowanych modułach np. interfejsów komunikacyjnych USART, SPI, I^2C, licznikach, przetwornikach analogowo-cyfrowych.

Procesory dysponują dwoma typami rejestrów związanymi z systemem przerwań: rejestrem flag i rejestrem maski. W rejestrze flag przerwań przechowywana jest informacja o zaistnieniu przyczyn poszczególnych przerwań. W mikrokontrolerach AVR flagi te są kasowane automatycznie sprzętowo, gdy następuje wykonanie podprogramu obsługi przerwania, a także możliwe jest ich kasowanie programowe. Rejestr maski umożliwia selektywne blokowanie (maskowanie) danego przerwania.

Wyróżniamy dwa typy systemu przerwań. Jeden, dla którego poszczególnym źródłom przerwań przyporządkowano określony priorytet (na stałe lub może przyporządkować go programista). W tym przypadku obsługę przerwań o niższym priorytecie mogą przerywać wyłącznie żądania przerwań o priorytetach wyższych. W drugim przypadku realizacja każdego podprogramu obsługi przerwania (dla globalnego odblokowania przerwań) jest przerywana przez każde odblo-

kowane żądanie przerwania (w rejestrze maski). W przypadku mikrokontrolerów AVR globalne zezwolenie na przerwania umożliwia przerwanie podprogramu obsługi przerwania przez dowolne, odblokowane żądanie przerwania. Priorytetowość zawarta w dokumentacji mikrokontrolerów AVR dotyczy wyłącznie kolejności wywoływania procedur obsługi przerwań w przypadku, gdy po globalnym zezwoleniu na przerwania aktywna jest więcej niż jedna flaga. W pierwszej kolejności obsługiwane będą wówczas przerwania, których numer wektora jest najmniejszy (tab. 9.1

Tabela 9.1

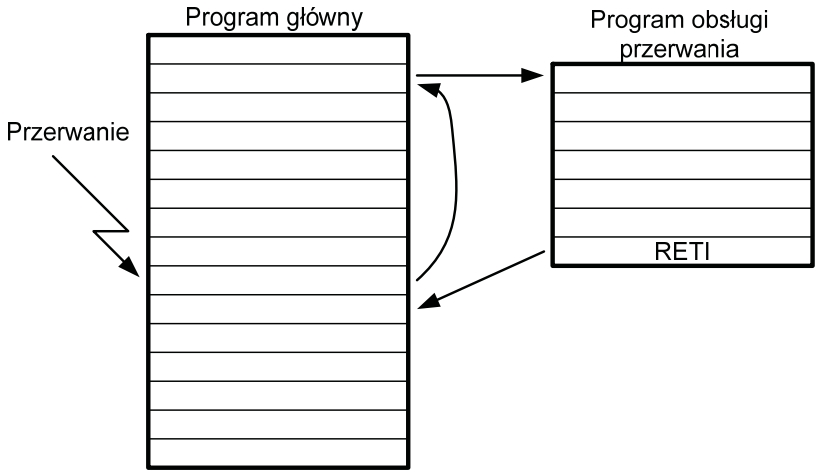
Wektory przerwań dla ATmega128 [1]

Nr wektora	Adres programu	Źródło	Definicja przerwania
1	$0000	RESET	Wszystkie źródła zerowania procesora
2	$0002	INT0	Przerwanie zewnętrzne 0
3	$0004	INT1	Przerwanie zewnętrzne 1
4	$0006	INT2	Przerwanie zewnętrzne 2
5	$0008	INT3	Przerwanie zewnętrzne 3
6	$000A	INT4	Przerwanie zewnętrzne 4
7	$000C	INT5	Przerwanie zewnętrzne 5
8	$000E	INT6	Przerwanie zewnętrzne 6
9	$0010	INT7	Przerwanie zewnętrzne 7
10	$0012	TIMER2 COMP	Licznik-stoper2 porównanie
11	$0014	TIMER2 OVF	Licznik-stoper2 przepełnienie
12	$0016	TIMER1 CAPT	Licznik-stoper1 przechwycenie
13	$0018	TIMER1 COMPA	Licznik-stoper1 porównanie A
14	$001A	TIMER1 COMPB	Licznik-stoper1 porównanie B
15	$001C	TIMER1 OVF	Licznik-stoper1 przepełnienie
16	$001E	TIMER0 COMP	Licznik-stoper0 porównanie
17	$0020	TIMER0 OVF	Licznik-stoper0 przepełnienie
18	$0022	SPI, STC	SPI ukończenie transmisji szeregowej
19	$0024	USART0, RX	USART0 zakończenie odbioru danej
20	$0026	USART0 UDRE	USART0 pusty bufor nadawczy
21	$0028	USART0 TX	USART0 zakończenie wysyłania danej
22	$002A	ADC	Zakończenie konwersji ADC
23	$002C	EE READY	EEPROM gotowy
24	$002E	ANALOG CMP	Analogowy komparator
25	$0030	TIMER1 COMPC	Licznik-stoper1 porównanie C
26	$0032	TIMER3 CAPT	Licznik-stoper3 przechwycenie
27	$0034	TIMER3 COMPA	Licznik-stoper3 porównanie A
28	$0036	TIMER3 COMPB	Licznik-stoper3 porównanie B
29	$0038	TIMER3 COMPC	Licznik-stoper3 porównanie C
30	$003A	TIMER3 OVR	Licznik-stoper3 przepełnienie
31	$003C	USART1 RX	USART1 zakończenie odbioru danej
32	$003E	USART1 UDRE	USART1 pusty bufor nadawczy
33	$0040	USART1 TX	USART1 zakończenie wysyłania danej
34	$0042	TWI	Interfejs dwuprzewodowy
35	$0044	SPM READY	Koniec zapisu pamięci flash

Skok do wektora przerwań jest realizowany, jeżeli zachodzi następująca koniunkcja: przerwania są globalnie odblokowane (ustawiona flaga **I** w rejestrze statusu **SREG**), zezwolono na przerwanie (w rejestrze maski) oraz ustawiona jest flaga danego przerwania (w rejestrze flag).

Pod odpowiednim dla danego wektora przerwań adresem w pamięci programu umieszczona jest instrukcja bezwarunkowego skoku do podprogramu obsługi danego przerwania. Kolejne adresy poszczególnych wektorów przerwań różnią się względem siebie z reguły o dwa, trzy adresy (dla ATmega128 o dwa adresy), zatem w wektorach przerwań zawarte są wyłącznie instrukcje skoków bezwarunkowych do właściwych procedur obsługi przerwań.

Po skoku do wektora przerwań w mikrokontrolerze AVR automatycznie zabronione są globalnie przerwania, na stos odkładany jest adres powrotu z przerwania (aktualna wartość licznika rozkazów PC) (zajmuje to cztery cykle zegarowe). Następnie realizowany jest skok programowy do podprogramu obsługi przerwania (kolejne trzy cykle zegarowe). Jeżeli wywołanie przerwania nastąpi podczas instrukcji realizowanej w więcej niż w jednym cyklu czas reakcji na przerwanie dodatkowo wydłuży się. W procedurze obsługi przerwania należy **programowo zapisać na stos rejestry**, których wartości mogą być zmodyfikowane podczas jej wykonywania, w szczególności należy zadbać o rejestr statusu **SREG**. Do powrotu z podprogramu obsługi przerwania (wykonywanego w czterech cyklach zegarowych) wykorzystuje się instrukcję **RETI**, powoduje ona odczyt ze stosu adresu powrotu i załadowanie go do licznika programu PC oraz globalne zezwolenie na przerwania. Instrukcję **RETI** należy poprzedzić **zdjęciem ze stosu rejestrów**, które odłożono na początku podprogramu obsługi przerwania.

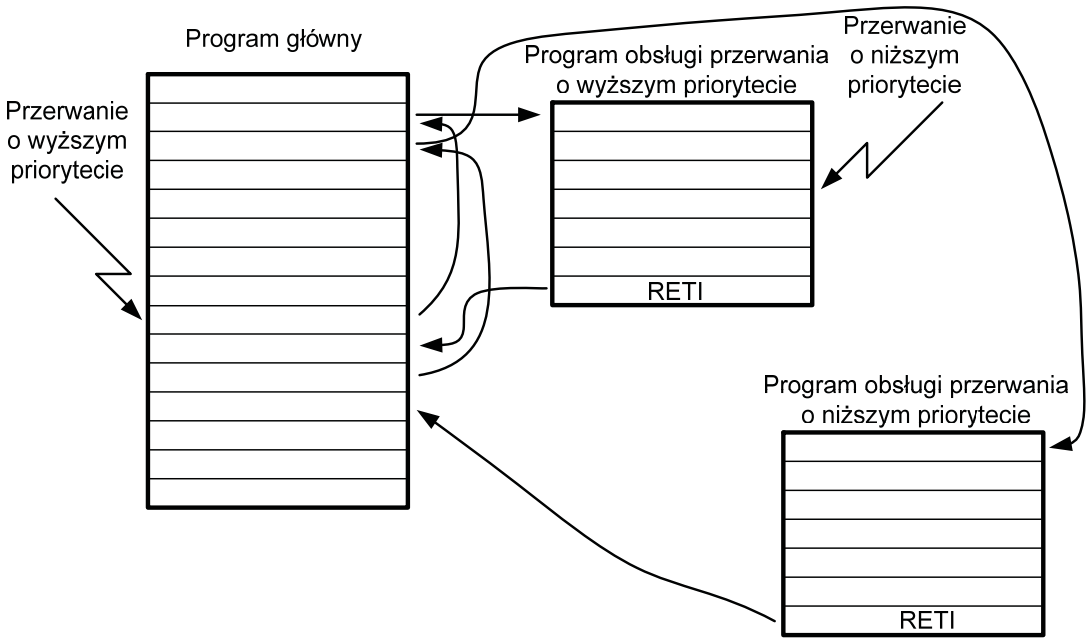


Rys. 9.1. Obsługa pojedynczego przerwania

Na rysunku 9.1 zaprezentowano ideę obsługi żądania przerwania. Procesor, wykonując program główny, realizuje kolejne instrukcje. Po wykonaniu instrukcji (innej niż instrukcja skoku) licznik rozkazów inkrementuje się o jeden, adresując kolejną komórkę w pamięci programu. W przypadku realizacji instrukcji skoku licznik rozkazów po jej wykonaniu przyjmuje wartość adresu komórki, do której realizowany jest skok. Założono, iż przerwania globalnie zostały odblokowane oraz selektywnie źródło przerwania zostało także odblokowane. Dla wymienionych uwarunkowań po zgłoszeniu żądania przerwania (ustawieniu flagi w rejestrze flag) sprzętowo realizowany jest skok do wektora przerwań odpowiedniego dla danego źródła. Ponadto dla mikrokontrolera AVR na stos sprzętowo odkładany jest adres powrotu z przerwania (aktualna wartość licznika rozkazów) oraz przerwania są globalnie zabronione. Po realizacji skoku bezwarunkowego realizowany jest program obsługi przerwania. W prologu podprogramu obsługi przerwania należy zapisać na stos rejestry ogólnego przeznaczenia, których wartości podczas jego realizacji są modyfikowane oraz rejestr statusu **SREG**. W epilogu natomiast należy odczytać ze stosu w kolejności odwrotnej odłożone wcześniej rejestry wraz z rejestrem statusu. Wykonanie ostatniej instrukcji **RETI** powoduje powrót do programu głównego (przepisanie do licznika rozkazów adresu pamięci programu wcześniej zapisanego

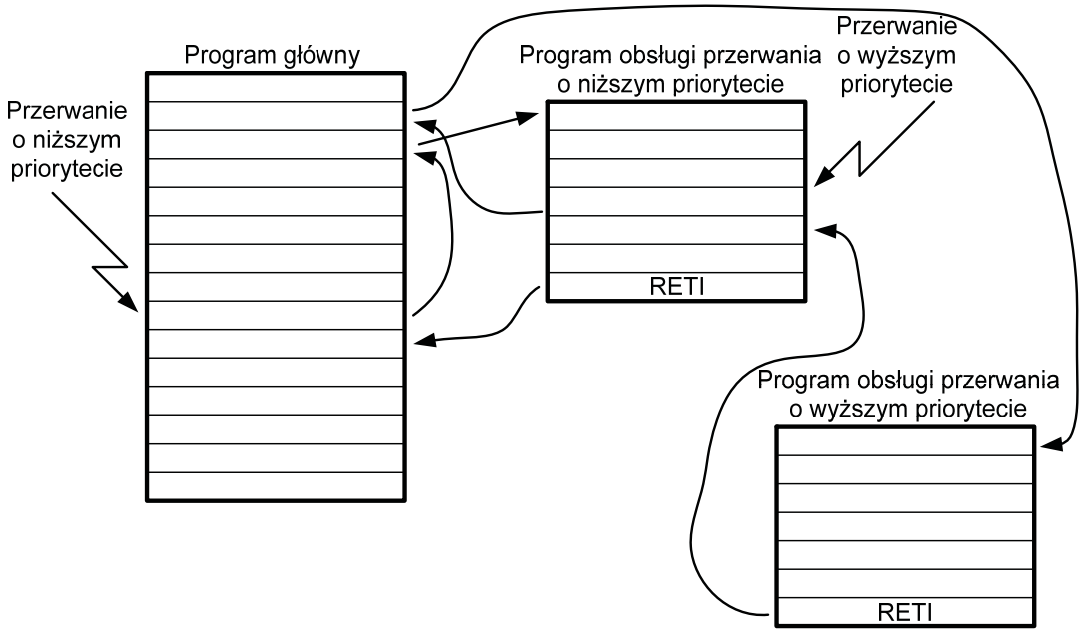
na stos) oraz globalne zezwolenie na przerwania. Należy zwrócić uwagę, iż odtworzenie modyfikowanych w podprogramie obsługi przerwania rejestrów procesora gwarantuje poprawną kontynuację wykonania programu głównego. W przypadku poprawnego ich odtworzenia realizacja programu głównego zostanie jedynie zawieszona na czas wykonania podprogramu obsługi przerwania.

Na rysunku 9.2 zaprezentowano sekwencje realizacji programów obsługi przerwań w przypadku, gdy podczas realizacji podprogramu obsługi przerwania o priorytecie wyższym zostało zgłoszone przerwanie o priorytecie niższym. Omawiany przypadek jest domyślnym dla mikrokontrolera AVR ze względu na automatyczne globalne zablokowanie przerwań podczas realizacji podprogramu obsługi przerwania. Po zrealizowaniu funkcji obsługi pierwszego przerwania następuje powrót do programu głównego wykonanie jego jednej instrukcji i skok do drugiego wektora przerwań, a następnie realizacja podprogramu obsługi przerwania o priorytecie niższym.



Rys. 9.2. Obsługa przerwań w przypadku pojawienia się żądania przerwania o priorytecie niższym w trakcie obsługi przerwania o priorytecie wyższym (dla mikrokontrolera AVR przypadek domyślny)

Kolejnym przypadkiem jest przedstawiony na rys. 9.3, gdy podczas realizacji podprogramu obsługi przerwania o priorytecie niższym zostało zgłoszone przerwanie o priorytecie wyższym. Aby omawiana sytuacja zaistniała dla mikrokontrolerów AVR należy programowo w prologu podprogramu obsługi przerwania o priorytecie niższym odblokować globalnie przerwania. Ze względu na „płaski” system przerwań dla mikrokontrolerów AVR dowolne odblokowane żądanie przerwania (nawet tego samego źródła) może przerwać wykonywanie podprogramu obsługi przerwania w przypadku globalnie odblokowanych przerwań. Priorytetowość przerwań można w pewnej mierze rozwiązać programowo za pomocą sterowania ich globalną i selektywną blokadą.



Rys. 9.3. Obsługa przerwań w przypadku pojawienia się żądania przerwania o priorytecie wyższym w trakcie obsługi przerwania o priorytecie niższym (dla mikrokontrolera AVR należy przerwania globalnie odblokować w prologu podprogramu obsługi przerwania)

9.2.1. Obsługa przerwań z wykorzystaniem biblioteki avr-libc

Programując mikrokontroler AVR z wykorzystaniem biblioteki avr-libc, programista jest zwolniony z programowej obsługi stosu w procedurach obsługi przerwań. Przerwania obsługiwane są przez funkcje **ISR(nazwa_vect)**, np. od przerwania zewnętrznego numer 0: **ISR(INT0_vect)**. Nazwy wektorów można znaleźć w dokumentacji [4], wybierając *Library Reference*, a następnie *<avr/interrupt.h>: Interrupts*. Do globalnego zezwolenia na przerwania wykorzystujemy funkcje **sei()**, natomiast do zabronienia **cli()**.

Priorytetowość procedur obsługi przerwań można zapewnić tylko w ograniczonym zakresie, ponieważ system przerwań procesorów AVR jest „płaski". Przykładowo można przyjąć jedno z przerwań jako najbardziej istotne (dla danej aplikacji), wówczas w procedurze jego obsługi nie odblokowujemy globalnie przerwań, natomiast we wszystkich pozostałych odblokowujemy. Zezwalamy wówczas na podejmowanie obsługi przerwania najistotniejszego podczas wykonywania procedur obsługi pozostałych przerwań. Należy jednak zwrócić uwagę, iż jeżeli nastąpi żądanie obsługi dowolnego odblokowanego przerwania podczas obsługi pozostałych przerwań, także zostaje przerwana obsługa bieżącego. W szczególności podprogramy obsługi przerwań mogą być wywoływane wielokrotnie tym samym źródłem przerwań i spowodować w konsekwencji na przykład przepełnienie stosu.

Chcąc modyfikować te same zmienne globalne w programie głównym i procedurach obsługi przerwań, konieczne jest nadanie im specyfikatora **volatile** (np. **volatile char zmienna1**). Wówczas kompilator nie będzie optymalizował kodu wynikowego zawierającego taką zmienną.

Niepożądane skutki mogą przynieść operacje na zmiennych zawierających więcej niż jeden bajt, gdy są one modyfikowane zarówno w programie głównym, jak i procedurach obsługi przerwań. Wystąpienie przerwania pomiędzy modyfikacją jednego a pozostałych bajtów zmiennej w programie głównym spowoduje, iż podprogram obsługi przerwania zmodyfikuje jej wartość, a po jego wykonaniu nastąpi kontynuacja jej modyfikacji w programie głównym. W omawianym przypadku uzyskana wartość jednego bajtu zmiennej jest wynikiem działania

podprogramu obsługi przerwania, natomiast pozostałych bajtów programu głównego. Analogiczne sytuacje można zaprezentować w przypadku modyfikacji zmiennej jednobajtowej, gdy przerwanie zostanie wywołane pomiędzy odczytem wartości zmiennej a jej zapisaniem. Należy wówczas (o ile jest to możliwe) tak napisać program by zmienna była modyfikowana tylko w jednej procedurze obsługi przerwania albo w programie głównym. W szczególnym przypadku, jeżeli modyfikujemy zbiór flag jednobitowych należy je tak pogrupować w bajty, aby dany bajt był modyfikowany tylko w jednym z podprogramów. Niepożądane sytuacje mogą zaistnieć także w przypadku realizacji operacji warunkowych zawierających w warunku zmienną, która jest wewnątrz jego ciała modyfikowana. Można w niektórych przypadkach ich uniknąć odpowiednio przed sprawdzeniem warunku, globalnie blokując przerwania i po modyfikacji zmiennej ponownie zezwalając globalnie na przerwania.

9.3. Ośmiobitowy licznik0 z wyjściami PWM

W niniejszej opracowaniu opisano tylko niezbędne informacje konfiguracyjne licznika0 wykorzystywane w oprogramowaniu. Inicjalizacja pracy licznika0 została zrealizowana w funkcji `init_TIMER0`.

Rejestrem umożliwiającym selektywne zezwalanie na przerwania, których źródłem jest przepełnienie, porównanie i przechwycenie dla liczników mikrokontrolera jest rejestr (ang. *Timer/Counter Interrupt MaskRegister*) TIMSK (rys.). Zezwolenie na przerwanie od przepełnienia dla licznika0 uzyskano ustawiając bit **TOIE0**: `TIMSK = _BV(TOIE0);`

Bit	7	6	5	4	3	2	1	0
TIMSK	**OCIE2**	**TOIE2**	**TICIE1**	**OCIE1A**	**OCIE1B**	**TOIE1**	**OCIE0**	**TOIE0**
Odczyt/Zapis	O/Z	O/Z	O/Z	O/Z	O/Z	O/Z	O/Z	O/Z
Wart. początkowa	0	0	0	0	0	0	0	0

Rys. 9.4. Rejestr maski dla liczników – TIMSK

Rejestrem flag dla źródeł sygnału przerwań od przepełnienia i porównania dla poszczególnych liczników mikrokontrolera jest rejestr (ang. *Timer/Counter Interrupt Flag Register*) TIFR (rys.). Poszczególne flagi są kasowane sprzętowo w przypadku wykonywania podprogramu obsługi przerwania, a ponadto można je kasować programowo, wpisując jeden do odpowiedniej flagi.

Bit	7	6	5	4	3	2	1	0
TIFR	**OCF2**	**TOV2**	**ICF1**	**OCF1A**	**OCF1B**	**TOV1**	**OCF0**	**TOV0**
Odczyt/Zapis	O/Z	O/Z	O/Z	O/Z	O/Z	O/Z	O/Z	O/Z
Wart. początkowa	0	0	0	0	0	0	0	0

Rys. 9.5. Rejestr flag dla liczników – TIFR

9.3.1. Źródła sygnału zegarowego dla licznika0

Domyślnym źródłem sygnału zegarowego dla omawianego licznika jest wewnętrzny synchroniczny sygnał zegarowy zasilający m.in. obsługę portów i liczników mikrokontrolera – sygnał $clk_{I/O}$. Opcjonalnym źródłem sygnału zegarowego może być zewnętrzny asynchroniczny rezonator kwarcowy podłączony do wyprowadzeń **TOSC1** i **TOSC2**.

Ustawienie źródła sygnału zegarowego realizowane jest w rejestrze asynchronicznego statusu **ASSR** (rys.). W przypadku, gdy bit **ASO** wynosi zero licznik0 taktowany jest we-

wnętrznym synchronicznym sygnałem zegarowym $clk_{I/O}$. W przeciwnym przypadku licznik0 będzie zasilany zewnętrznym asynchronicznym sygnałem zegarowym. W rozważanej aplikacji nie realizujemy wpisu do rejestru **ASSR,** pozostawiając w nim wartości domyślne.

Bit	7	6	5	4	3	2	1	0
ASSR	-	-	-	-	**ASO**	**TCN0UB**	**OCR0UB**	**TCR0UB**
Odczyt/Zapis	O/Z	O/Z	O/Z	O/Z	O/Z	O/Z	O/Z	O/Z
Wart. początkowa	0	0	0	0	0	0	0	0

Rys. 9.6. Rejestr asynchronicznego statusu – ASSR

9.3.2. Wstępny dzielnik częstotliwości

Dla omawianego licznika istnieje możliwość wstępnego podziału częstotliwości źródłowego sygnału zegarowego. Domyślnie sygnał zegarowy licznika0 (clk_{T0}) podłączony jest bezpośrednio do wewnętrznego sygnału zegarowego $clk_{I/O}$ (rys. 9.8). W zależności od kombinacji bitów (ang. *Clock Select*) **CS00÷02** znajdujących się w rejestrze kontrolnym (ang. *Timer/Counter Control Register*) **TCCR0** (rys. 9.7) można uzyskać podziały częstotliwości przedstawione w o trybie pracy licznika0 decyduje kombinacja bitów (ang. *waveform generation mode*) **wgm00÷01.** flagi (ang. *compare output mode*) **com00÷01** wpływają na zachowanie się wyjścia porównania.

Bit	7	6	5	4	3	2	1	0
TCCR0	**FOC0**	**WGM00**	**COM01**	**COM00**	**WGM01**	**CS02**	**CS01**	**CS00**
Odczyt/Zapis	Z	O/Z	O/Z	O/Z	O/Z	O/Z	O/Z	O/Z
Wart. początkowa	0	0	0	0	0	0	0	0

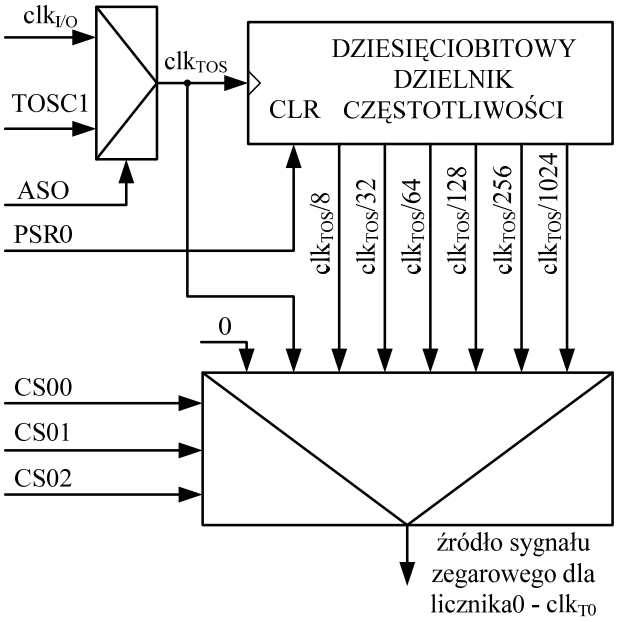
Rys. 9.7. Rejestr kontrolny trybów pracy licznika0 – TCCR0

O trybie pracy licznika0 decyduje kombinacja bitów (ang. *Waveform Generation Mode*) **WGM00÷01.** Flagi (ang. *Compare Output Mode*) **COM00÷01** wpływają na zachowanie się wyjścia porównania (tab. 9.2).

Tabela 9.2

Ustawienie podziału częstotliwości

CS02	CS01	CS00	Opis
0	0	0	Brak źródła zegarowego (Licznik zatrzymany)
0	0	1	clk_{T0S} (brak podziału)
0	1	0	$clk_{T0S}/8$
0	1	1	$clk_{T0S}/32$
1	0	0	$clk_{T0S}/64$
1	0	1	$clk_{T0S}/128$
1	1	0	$clk_{T0S}/256$
1	1	1	$clk_{T0S}/1024$



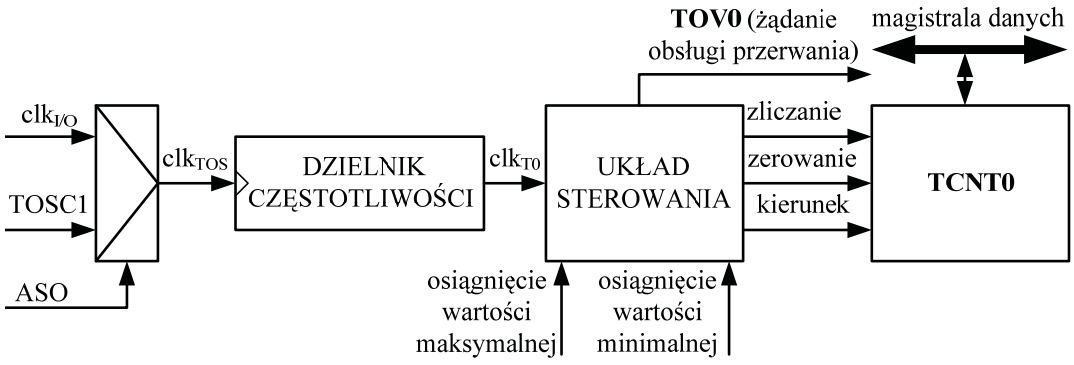
Rys. 9.8. Wstępny podział częstotliwości dla licznika0 w ATmega128 [1]

W celu ułatwienia realizacji symulacji w funkcji `init_TIMER0` zrealizowano opcjonalne dwa wpisy do rejestru **TCCR0** (wykorzystując kompilację warunkową). Symulator mikrokontrolera działa znacznie wolniej od rzeczywistego procesora, dlatego ustawiono znacznie mniejszy współczynnik podziału dla symulacji. Aby nie dzielić wstępnie częstotliwości należy ustawić parametr `HAPSIM` na 1 (`#define HAPSIM 1`). Wartość wstępnego podziału częstotliwość ustawiono na 1024 w celu eliminacji odbicia styków podczas naciskania przycisków. Szerzej zagadnienie to zostało opisane w dalszej części instrukcji.

```
#if HAPSIM
    TCCR0 = _BV(CS00); //fclk/1,
#else
    TCCR0 = _BV(CS00)|_BV(CS01)|_BV(CS02); //fclk/1024,
#endif
```

9.3.3. Moduł licznika

Zasadniczą częścią omawianego licznika jest ośmiobitowy programowalny rewersyjny moduł licznika (rys. 9.9). W zależności od wybranego trybu pracy licznik może powiększyć/zmniejszyć swoją wartość o jeden lub wyzerować w każdym takcie sygnału zegarowego clk_{T0}. Flaga przepełnienia **TOV0** może być źródłem przerwania. Jest ono zgłaszane między innymi w przypadku przepełnienia licznika (w zależności od ustawionego trybu pracy).



Rys. 9.9. Schemat blokowy licznika0 w ATmega128 [1]

9.3.4. Tryb normalny pracy licznika

W celu uzyskania najprostrzego z trybów pracy licznika – trybu normalnego, należy wpisać do obu bitów **WGM00÷01** wartość zero. W tym przypadku licznik inkrementuje wartości od 0 do 255, a po doliczeniu do 255 zeruje się i ustawia flagę **TOV0**. Flaga ta może być źródłem przerwania, przy czym kasowana jest automatycznie, gdy wykonana jest procedura obsługi przerwania lub może być skasowana programowo przez wpisanie wartości jeden w rejestrze flag. Dodatkowo do generacji przerwań można wykorzystać moduł komparatora, nie zaleca się jednak w tym trybie wykorzystywania wyprowadzenia **OC0**.

Dla omówionej konfiguracji częstotliwość przerwań od przepełnienia licznika0 można wyrazić nastepujacą zależnością:

$$f_{\mathrm{INT\_OV0}} = \frac{f_{\mathrm{clk_{I/O}}}}{N256} = \frac{16\ \mathrm{MHz}}{1024*256} \approx 61\ \mathrm{Hz} \tag{9.1}$$

gdzie: *N* oznacza wartość podziału częstotliwości przez wstępny dzielnik (1, 8, 32, 64, 128, 256 i 1024).

9.4. Opis projektu

W projekcie oprogramowano obsługę przycisków z eliminacją odbicia styków. Po przyciśnięciu jednego z przycisków (od KEY0 do KEY3) następuje inkrementacja modulo 10 jednego z czterech elementów globalnej tablicy `cyfry[4]`. Każda z cyfr jest cyklicznie wizualizowana za pomocą jednego z czterech wyświetlaczy siedmiosegmentowych. Do instrukcji dołączono projekty zrealizowane w środowiskach WinAVR, AVRStudio i HAPSIM.

9.5. Opis oprogramowania

W projekcie wykorzystano ośmiobitowy licznik0, dla którego zezwolono na przerwania od jego przepełnienia. Funkcja obsługi przerwania przekazuje do programu głównego informacje dotyczące przyciśnięcia przycisków od KEY0 do KEY3. Należy zwrócić uwagę, iż napięcie wymuszane odpowiednim stanem przycisków filtrowane jest dolnoprzepustowymi filtrami RC (rys. 8.2), a wejścia mikrokontrolera mają histerezę. Połączenie wymienionych rozwiązań sprzętowych wraz z odpowiednim oprogramowaniem funkcji obsługi przerwania pozwala skutecznie eliminować efekt odbicia styków przycisków. Wprowadzono nazwy funkcjonalne dla poszczególnych przycisków:

— KEY0 – ENTER,
— KEY1 – UP,
— KEY2 – DOWN,
— KEY3 – CANCEL.

Do obsługi przerwania wykorzystano unię `key`. Unia `key` zawiera strukturę `flags` składająca się z ośmiu pól jednobitowych (`:1`) oraz bajtu `char byte`. W czterech mniej znaczących bitach w strukturze `flags` przechowywane są stany logiczne odpowiadające poszczególnym przyciskom. W czterech bardziej znaczących bitach tej struktury zapisywane są także stany logiczne odpowiadające poszczególnym przyciskom, ale z poprzedniego wywołania podprogramu obsługi przerwania. Zastosowanie unii pozwala programiście w języku C odczytywać i modyfikować dane różnych typów zawarte w tym samym obszarze pamięci. Omawiana unia pozwala zarówno na odwołanie się w programie do całego bajtu (np.: `key.byte = (key.byte<<4)`), jak i do poszczególnych bitów (w niej zawartych) z osobna (np.: `if(key.flags.ENTER)`).

```
union {        //do obsługi przycisków -> w przerwaniu ISR
 struct {
  unsigned char ENTER:1;    //LSB
  unsigned char UP:1;
  unsigned char DOWN:1;
  unsigned char CANCEL:1;
  unsigned char ENTER_PREV:1;
  unsigned char UP_PREV:1;
  unsigned char DOWN_PREV:1;
  unsigned char CANCEL_PREV:1; //MSB
 } flags;
  char byte;
} key;
```

Druga z zastosowanych w podprogramie obsługi przerwania unia `falling_slope` zawiera słowo szesnastobitowe (`int word`) oraz strukturę składającą się z szesnastu pól jednobitowych. W najmniej znaczących czterech bitach przechowywana jest informacja o aktualnie wykrytych zboczach opadających sygnałów, których stany determinują przyciski (od KEY0 do KEY3). W kolejnych bardziej znaczących czwórkach bitów przechowywane są informacje o zboczach opadających z poprzednich wywołań podprogramu obsługi przerwania. Najstarsze informacje (sprzed trzech wywołań) przechowywane są w najbardziej znaczącej części struktury `flags`.

```
union {        //do obsługi przycisków -> w przerwaniu ISR
 struct {
  unsigned char ENTER:1;       //LSB
  unsigned char UP:1;
  unsigned char DOWN:1;
  unsigned char CANCEL:1;
  unsigned char ENTER_PREV:1;
  unsigned char UP_PREV:1;
  unsigned char DOWN_PREV:1;
  unsigned char CANCEL_PREV:1;
  unsigned char ENTER_PREV1:1;
  unsigned char UP_PREV1:1;
  unsigned char DOWN_PREV1:1;
  unsigned char CANCEL_PREV1:1;
  unsigned char ENTER_PREV2:1;
  unsigned char UP_PREV2:1;
  unsigned char DOWN_PREV2:1;
  unsigned char CANCEL_PREV2:1; //MSB
  } flags;
  int word;
} falling_slope;
```

Nazwę funkcji obsługi przerwania od przepełnienia licznika0 zastosowano zgodnie z dokumentacją do biblioteki avr-libc [4] (`ISR (TIMER0_OVF_vect)`). Funkcja ta obsługuje cztery przyciski, eliminując efekt odbicia styków. Realizację jej oparto na potrójnym potwierdzeniu niskiego stanu logicznego po wykryciu zbocza opadającego. Informacja o przyciśnięciu przez użytkownika danego przycisku jest dostępna dla programu głównego wyłącznie w przypadku odczytania na odpowiadającym mu wejściu w pierwszej kolejności domyślnego stanu wysokiego, a następnie przez cztery kolejne wywołania podprogramu obsługi przerwania stanu niskiego (niezbędny czas trwania stanu niskiego około 65,5 ms).

Początkowo w funkcji obsługi przerwania nadpisywane są wartości flag odpowiadające stanom poszczególnych przycisków z poprzedniego wywołania, a flagi odpowiadające bieżącym stanom przycisków są nadpisywane zerami: `key.byte = (key.byte<<4);`

W przypadku przyciśnięcia przycisków odpowiednie flagi nadpisywane są jedynkami (w mniej znaczącym półbajcie unii `key`). Realizowany jest odczyt stanów sygnałów wymuszanych przyciskami z jednoczesnym ich uporządkowaniem. W przypadku założenia, iż zrealizowano połączenia pomiędzy przyciskami i adresami bitów portu mikrokontrolera jak

w pakiecie dydaktycznym (KEY0 do bitu 0, KEY1 do bitu 1 itd.) odczyt wejść można zrealizować w następujący sposób:

```
key.byte |= (~PINKEY) & 0x0F;
```

Jednakże, zakładając dowolny układ połączeń pomiędzy przyciskami, a wyprowadzeniami danego portu, należy dodatkowo uporządkować kolejność flag zapisywanych w mniej znaczącym półbajcie unii key:

```
key.byte|= ((bit_is_clear(PINKEY,KEY0))|(bit_is_clear(PINKEY,KEY1)<<1)
          |(bit_is_clear(PINKEY,KEY2)<<2)|(bit_is_clear(PINKEY,KEY3)<<3));
```

Następnie zrealizowano nadpisywanie dwunastu flag przechowujących informacje o wykryciu zbocza opadającego – flagami bardziej aktualnymi (o jedno wywołanie funkcji obsługi przerwania).

```
falling_slope.word=(falling_slope.word<<4);
```

Oprogramowano wykrycie zbocza opadającego, które pojawia się podczas naciskania przycisku.

```
if(key.flags.ENTER){
    if(!key.flags.ENTER_PREV) falling_slope.flags.ENTER=1;
```

Ponadto sprawdzany jest warunek, czy przez kolejne cztery przerwania odczytany został stan niski.

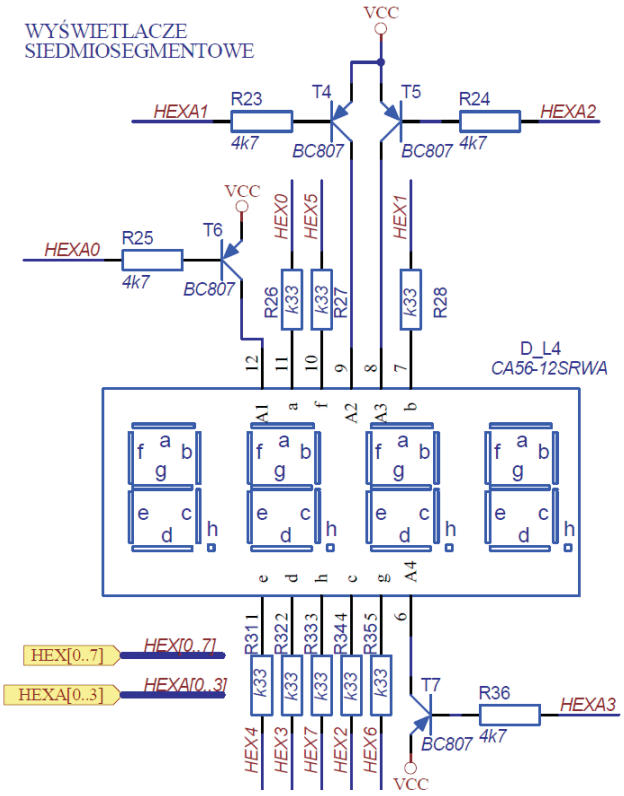
```
    if(falling_slope.flags.ENTER_PREV2) kl_ENTER = 1;
}
```

Jeżeli zostanie odczytany wysoki stan logiczny, wówczas wszystkie cztery flagi przechowujące informację o wykryciu zbocza opadającego (aktualna, opóźniona o jedno, dwa i trzy wywołania przerwania) dla danego przycisku zostają wyzerowane.

```
else{
    falling_slope.word&=0xEEEE;
};
```

Obsługa pozostałych przycisków została oprogramowana w sposób analogiczny. Należy zwrócić uwagę, iż informacja przekazywana do programu głównego to ustawienie wartości jeden dla zmiennej odpowiadającej danemu przyciskowi (np. `kl_ENTER = 1;`). W programie głównym zatem, po wykorzystaniu informacji o wciśnięciu danego przycisku, należy zmienną mu odpowiadająca wyzerować (np. `kl_ENTER = 0;`).

W układzie dydaktycznym AVR_edu zastosowano wyświetlacze siedmiosegmentowe za wspólną anodą. Sterowanie poszczególnymi anodami zrealizowano za pomocą tranzystorów bipolarnych PNP (od T4 do T7 rys. 9.10). Stanem aktywnym powodującym włączenie danego wyświetlacza jest stan niski, wówczas odpowiedni tranzystor znajduje się w stanie nasycenia. Włączenie poszczególnych segmentów następuje po wywołaniu stanu niskiego na odpowiednich katodach przy jednoczesnym wysterowaniu anody dla danego wyświetlacza. Układ połączeń poszczególnych katod i baz tranzystorów od T4 do T7 z mikrokontrolerem ATmega128 (rys. 8.1) został odpowiednio uwzględniony w pliku main.h.



Rys. 9.10. Schemat podłączenia wyświetlaczy siedmiosegmentowych

Funkcja `BCD_to_7_seg` realizuje konwersję kodu BCD na sterowanie poszczególnymi katodami wyświetlaczy siedmiosegmentowych. Dla przykładu, chcąc wizualizować cyfrę 0, należy włączyć segmenty od zerowego do piątego (od a do f), zatem wyłączone pozostaną segmenty szósty i siódmy (g i h – kropka). W omawianym przypadku dwa najbardziej znaczące bity w bajcie odpowiadające wyjściom `HEX7` i `HEX6` muszą przyjąć wartość logiczną jeden, natomiast pozostałe wartość logiczną zero (`case 0 : katody=0xC0; //0b11000000`).

```
char BCD_to_7_seg(char cyfra){
char katody = 0xFF; //domyślnie wszystkie segmenty wyłączone
    switch(cyfra){
         case 0 : katody=0xC0;  //0b11000000
                  break;
         case 1 : katody=0xF9;
                  break;
         case 2 : katody=0xA4;
                  break;
         case 3 : katody=0xB0;
                  break;
         case 4 : katody=0x99;
                  break;
         case 5 : katody=0x92;
                  break;
         case 6 : katody=0x82;
                  break;
         case 7 : katody=0xF8;
                  break;
         case 8 : katody=0x80;
                  break;
         case 9 : katody=0x90;
                  break;
    };
    return katody;
};
```

Funkcja `display_7seg` umożliwia cykliczną wizualizację czterech elementów tablicy `cyfry[4]`. Początkowo wyłączane są wszystkie wyświetlacze, a następnie modyfikowana jest zmienna `select`. Zmienna `select` jest inkrementowana przy każdym wywołaniu funkcji, a po osiągnięciu wartości pięć nadpisywana jest wartością jeden (czas, który to zajmuje wystarcza na wprowadzenie w stan zatkania tranzystorów sterujących anodami wyświetlaczy). W zależności od wartości zmiennej `select` wyświetlana jest jedna z czterech cyfr (w danym czasie). Do portu sterującego poszczególnymi katodami wpisywana jest wartość zwracana przez funkcję `BCD_to_7_seg(cyfry[selekt-1])`, a następnie zostaje włączony odpowiedni wyświetlacz poprzez wystawienie stanu niskiego i wprowadzenie odpowiedniego tranzystora (od T4 do T7) w stan nasycenia.

```
void display_7seg(void)
{
static char select = 1;     //zmienna o zakresie od 1 do 4
//wyłączenie wszystkich wyświetlaczy
    PORTHEXA |= (_BV(HEXA0)|_BV(HEXA1)|_BV(HEXA2)|_BV(HEXA3));
    select++;               //wskazanie na kolejną cyfrę
    if(select>4) select=1;
    switch(select)          //cykliczne aktywowanie wyświetlaczy 7-seg
     {
        case 1 :PORTHEX  = BCD_to_7_seg(cyfry[0]); //wizualizacja cyfry[0]
                PORTHEXA &= ~_BV(HEXA0);          //aktywacja 1 wyświetlacza
                break;
        case 2 :PORTHEX  = BCD_to_7_seg(cyfry[1]); //wizualizacja cyfry[1]
                PORTHEXA &= ~_BV(HEXA1);          //aktywacja 2 wyświetlacza
                break;
        case 3 :PORTHEX  = BCD_to_7_seg(cyfry[2]); //wizualizacja cyfry[2]
                PORTHEXA &= ~_BV(HEXA2);          //aktywacja 3 wyświetlacza
                break;
        case 4 :PORTHEX  = BCD_to_7_seg(cyfry[3]); //wizualizacja cyfry[3]
                PORTHEXA &= ~_BV(HEXA3);          //aktywacja 4 wyświetlacza
                break;
     };
};
```

Na początku programu głównego następuje inicjalizacja unii wykorzystywanych w podprogramie obsługi przerwania. Następnie następuje inicjalizacja sterowań poszczególnymi anodami i katodami wyświetlaczy powodująca ich początkowe wyłączenie. Konfigurowana jest kierunkowość wykorzystywanych w aplikacji wyprowadzeń mikrokontrolera. Realizowana jest inicjalizacja licznika0 oraz na koniec procesu inicjalizacji włączane zostało globalne zezwolenie na przerwania. Po procesie inicjalizacji wykonywana jest nieskończona pętla programu głównego. W pętli tej w zależności od przyciśnięcia danego przycisku przez użytkownika inkrementowana jest modulo 10 (cyklicznie od 0 do 9) odpowiednia cyfra zapisana w tablicy `cyfry[4]`. Dla omawianego oprogramowania, które w programie głównym realizuje wyłącznie modyfikację cyfr i ich wizualizację wywoływanie funkcji `display_7seg` jest poprawne. Jednakże w przypadku bardziej złożonego programu głównego należałoby wywoływać funkcję `display_7seg` w procedurze obsługi przerwania na przykład od kolejnego licznika (poza programem głównym). Wywoływanie funkcji `display_7seg` w procedurze obsługi przerwania wówczas zapewniłoby niezależną od stopnia skomplikowania programu głównego częstotliwość odświeżania wizualizowanych cyfr.

```
int main(void)
{
    key.byte=0x0F;                  //inicjalizacja flag stanami nieaktywnymi
    falling_slope.word=0x0000;      //inicjalizacja flag stanami nieaktywnymi
    PORTHEX = 0xFF;                 //wyłączenie wszystkich segmentów
//wyłączenie wszystkich wyświetlaczy
    PORTHEXA |= (_BV(HEXA0)|_BV(HEXA1)|_BV(HEXA2)|_BV(HEXA3));
    init_pins();                   //inicjalizacja kierunkowości wyprowadzeń
    init_TIMER0();                 //inicjalizacja Timera0
    sei();                         //globalne zezwolenie na przerwania
```

```
    while(1)                    //nieskończona pętla
    {
//modyfikacje poszczególnych cyfr pod wpływem wciśnięcia danego przycisku
        if(kl_ENTER == 1){
            cyfry[0]++;
            if (cyfry[0] > 9) cyfry[0] = 0;
            kl_ENTER = 0;
        };
        if(kl_UP == 1){
            cyfry[1]++;
            if (cyfry[1] > 9) cyfry[1] = 0;
            kl_UP =0;
        };
        if(kl_DOWN == 1){
            cyfry[2]++;
            if (cyfry[2] > 9) cyfry[2] = 0;
            kl_DOWN = 0;
        };
        if(kl_CANCEL == 1){
            cyfry[3]++;
            if (cyfry[3] > 9) cyfry[3] = 0;
            kl_CANCEL = 0;
        };
        display_7seg();  //wizualizacja wskazanej cyfry
    };
return 0;
}
```

9.6. Wytyczne dotyczące modyfikacji omówionego oprogramowania

Należy przeprogramować dołączony projekt, aby w następujący sposób reagował na przyciśnięcie przycisków:

1. Przed pierwszym i co piątym naciśnięciem przycisku ENTER program powinien zapewnić cykliczną wizualizację czterech cyfr zawartych w tablicy `cyfry[4]` i nie reagować na przyciśnięcia pozostałych przycisków.
2. Po pierwszym i co piątym naciśnięciu przycisku ENTER pierwsza cyfra powinna zacząć migać (z częstotliwością około 2 Hz), informując użytkownika o jej modyfikacji, natomiast pozostałe powinny być wizualizowane bez migania.
3. Kolejne naciśnięcia przycisku ENTER mają powodować miganie kolejnych cyfr zapewniające użytkownikowi sprzężenie zwrotne informujące go, która z cyfr jest aktualnie modyfikowana.
4. Po wielokrotności pięciokrotnego naciśnięcia przycisku ENTER następuje zatwierdzenie zmodyfikowanej nastawy (składającej się z czterech cyfr) i nadpisanie jej na cyfry zapisane w tablicy `cyfry[4]`. Po tej akcji żaden wyświetlacz nie miga i modyfikacja nastawy realizowana jest od początku (po ponownym naciśnięciu przycisku ENTER modyfikowana jest pierwsza cyfra).
5. Modyfikacja poszczególnych cyfr ma być realizowana poprzez wciśnięcie przycisku UP (inkrementacja modulo 10) i poprzez wciśniecie przycisku DOWN (dekrementacja modulo 10).
6. Jeżeli podczas modyfikacji dowolnej z cyfr zostanie naciśniety przycisk CANCEL zmodyfikowane już cyfry nie powinny zostać zapamiętane, a ich wartości powinny być przywrócone sprzed anulowanej modyfikacji. Należy zapewnić ich ponowną wizualizację.

9.7. Wskazówki dotyczące modyfikacji dołączonego oprogramowania

Poniżej zostały wymienione wskazówki dotyczące modyfikacji oprogramowania. Kolorem czerwonym w proponowanych fragmentach oprogramowania zaznaczono miejsca, które należy uzupełnić.

1. Zaleca się pisanie i testowanie oprogramowania etapowo, na przykład początkowo można pominąć miganie edytowanych cyfr.

2. Zaleca się zdefiniowanie czteroelementowej tablicy (`signed char cyfry_kopia[4]`), której elementy zostaną wykorzystane jako kopie podczas modyfikacji nastaw. Umożliwią one ponowne przywrócenie nastawy sprzed modyfikacji w przypadku naciśnięcia przez użytkownika przycisku CANCEL.
3. Zaleca się zdefiniowanie zmiennej (`char dzielnik_f`), która posłuży do programowego podzielenia częstotliwości przerwań w celu realizacji migania aktualnie edytowanych cyfr z częstotliwością 2 Hz.
4. Zaleca się zdefiniowanie zmiennej (`char miganie`), która będzie przyjmowała na przemian wartości 1 i 0 po osiągnięciu zaprogramowanej liczby przerwań (do ich odliczania proponuje się zastosowanie wyżej omówionej zmiennej). Jej wartość może zostać bezpośrednio wykorzystana w funkcji `display_7seg_kopia` do włączania i wyłączania modyfikowanej cyfry.
5. W programie obsługi przerwania zaleca się realizację programowego dzielnika częstotliwości (wykorzystując zmienne `dzielnik_f` i `miganie`):

```
dzielnik_f++;
 if(dzielnik_f==XXXX){ //częstotliwość migania modyfikowanej cyfry
                       //16e6/(1024*256*(2*XXXX)) ok. 2Hz
        dzielnik_f=0;
        miganie=!miganie;
 };
```

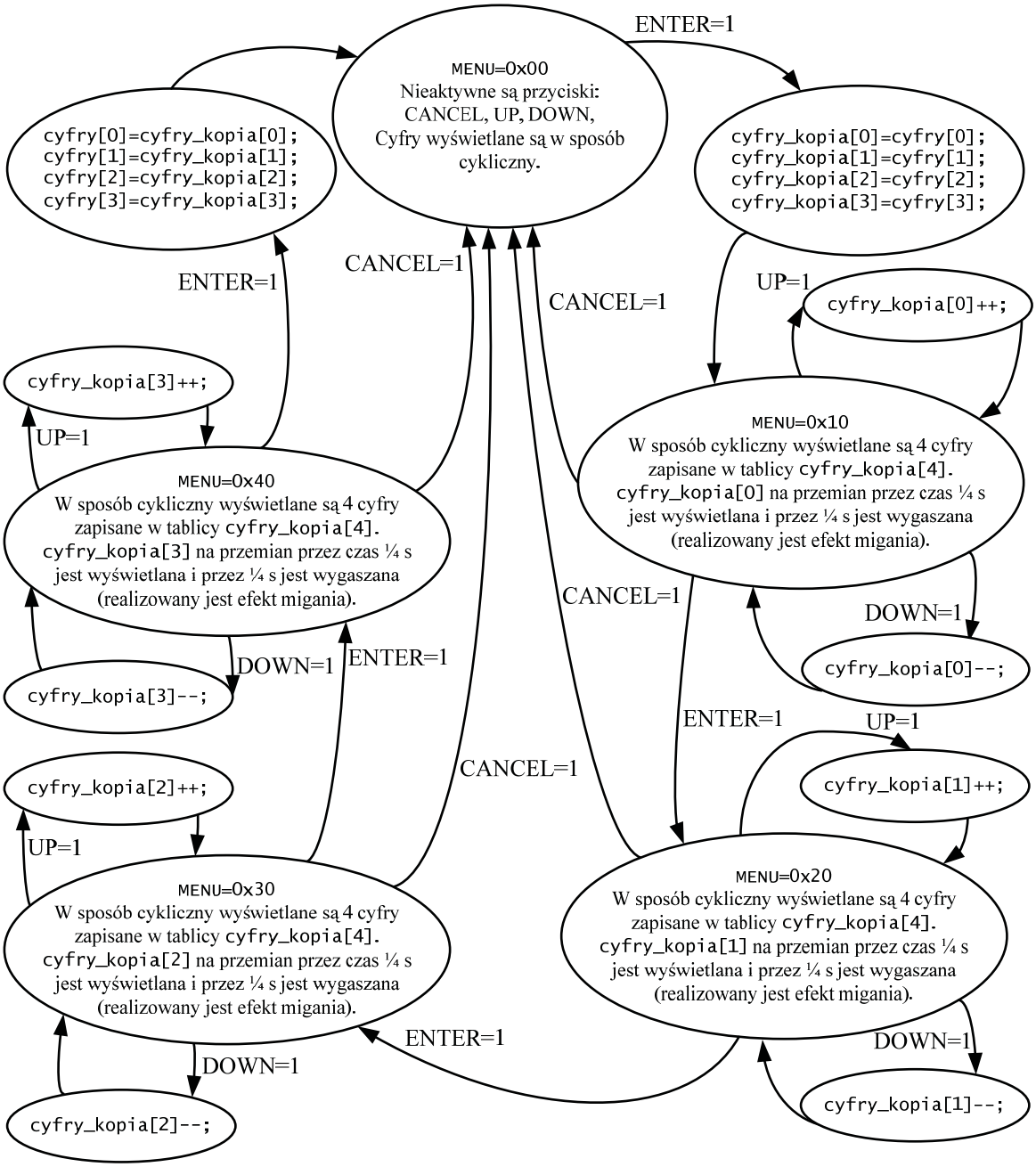
6. Zaleca się zdefiniowanie zmiennej (`char menu`), której wartość będzie modyfikowana pod wpływem naciskania przycisków ENTER i CANCEL. Zmienna ta może przechowywać informację, która z cyfr jest aktualnie modyfikowana.
7. Zaleca się utworzenie funkcji `display_7seg_kopia`, która powinna umożliwić wizualizację kopii cyfr (zapisanych w `cyfry_kopia[4]`) podczas ich modyfikacji. Ponadto w funkcji tej w celu realizacji migania edytowanej cyfry zaleca się wykorzystanie informacji zawartej w zmiennej `miganie` oraz `menu`:

```
case 1 :PORTHEX  = BCD_to_7_seg(cyfry_kopia[0]);
        if((miganie)&&(menu.....))
            PORTHEXA &= ~_BV(HEXA0);
        break;
```

8. W nieskończonej pętli programu głównego zaleca się oprogramować edycje poszczególnych cyfr zgodnie z wytycznymi zawartym w poprzednim punkcie (rys. 9.11). Do tego celu proponuje się wykorzystanie instrukcji `switch(menu)`:

```
while(1)                        //nieskończona pętla
{
//modyfikacje poszczególnych cyfr pod wpływem wciśnięcia przycisku
 switch(menu){
     case 0x00 : if(kl_ENTER == 1){
                     cyfry_kopia[0]=cyfry[...];
                     cyfry_kopia[1]=cyfry[...];
                     cyfry_kopia[2]=cyfry[...];
                     cyfry_kopia[3]=cyfry[...];
                     menu=.....;
                     kl_ENTER=0;
                 };
                 break;
     case 0x10 : if(kl_ENTER == 1){
                     menu=.....;
                     kl_ENTER=0;
                 };
                 if(kl_UP == 1){
                     cyfry_kopia[...]++;
                     if (cyfry_kopia[...] > ...) cyfry_kopia[...] = ...;
                     kl_UP = 0;
                 };
```

```
if(kl_DOWN == 1){
    cyfry_kopia[…]--;
    if (cyfry_kopia[…] < …) cyfry_kopia[…] = …;
    kl_DOWN = 0;
};
if(kl_CANCEL == 1){
    menu=……;
    kl_CANCEL = 0;
};
break;
```



Rys. 9.11. Proponowany sposób modyfikacji cyfr

9.8. Przygotowanie do zajęć

Przygotowanie do zajęć obejmuje:

1. Przyniesienie wydrukowanej niniejszej instrukcji (jednej na grupę laboratoryjną 2–3 osobową).
2. Szczegółowe zrozumienie rozwiązań programowych opisanych w instrukcji.
3. Dla chętnych zasymulowanie załączonego oprogramowania (można wykorzystać projekt main_elf_symulacja.aps dla środowiska AVRStudio i załączoną konfigurację dla programu HAPSIM). W programie HAPSIM można jedynie obserwować cykliczne aktywowanie anod i kombinacje katod odpowiadające poszczególnym cyfrom (początkowa ich wartość to cztery zera). Można także modyfikować wartości poszczególnych cyfr, aktywując przyciski (przez odpowiednio długi czas, zależny od mocy obliczeniowej komputera PC) i obserwować ich wartości w oknie podglądu zmiennych *Watch*. Należy pamiętać, by przed symulację skompilować projekt z definicją `#define HAPSIM 1`.
4. Wstępne przemyślania modyfikacji dołączonego oprogramowania pozwalające zrealizować wymienione wytyczne,
5. Dla chętnych zrealizowanie samodzielne wszystkich wymienionych modyfikacji (gwarantuje to sprawną realizację ćwiczenia na zajęciach laboratoryjnych).

9.9. Przebieg ćwiczenia

W ramach ćwiczenia laboratoryjnego należy uruchomić i przetestować dołączone oprogramowanie z wykorzystaniem układu dydaktycznej AVR_edu i uwzględnieniem wskazówek prowadzącego. Następnie należy przystąpić do modyfikacji dołączonego projektu zgodnie z wytycznymi (można wykorzystać zawarte powyżej wskazówki).

9.10. Literatura

[1] Dokumentacja mikrokontrolera ATmega 128 – http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf

[2] J. Zalewski, *Informatyka*, nr 4, 1980, http://www.aresluna.org/attached/terminology/informatyka/8004?language=pl

[3] J. Nieznański, materiały niepublikowane,

[4] Dokumentacja do biblioteki avr-libc, katalog z WinAVR\doc\avr-libc\avr-libc-user-manual\modules.html

10. Programowa realizacja zegara

10.1. Cel ćwiczenia

Celem ćwiczenia jest utrwalenie wiadomości z poprzedniego ćwiczenia i doskonalenie umiejętności programowania mikrokontrolera.

10.2. Opis projektu

W projekcie wykorzystano oprogramowanie realizowane w ramach poprzedniego ćwiczenia (wraz z uwzględnieniem wytycznych zawartych w punkcie 0). Poprzednia aplikacja została uzupełniona o obsługę przerwania, którego źródłem jest cykliczne doliczanie do zadanej wartości szesnastobitowego licznika1 z automatycznym zerowaniem. Niniejszy projekt jest uproszczoną, programową realizacją zegara. Poszczególne cyfry zawarte w tablicy `cyfry[4]` przechowują kolejno wartości: dziesiątek minut, jedności minut, dziesiątek sekund i jedności sekund.

10.3. Opis oprogramowania

W projekcie do odmierzania czasu jednej sekundy wykorzystano szesnastobitowy licznik1. Ustawiając bity kontrolne (ang. *Waveform Generation Mode*) **WGM13** i **WGM12,** uzyskano tryb kasowania licznika1 w przypadku osiągnięcia przez niego wartości zadanej (ang. *Clear Timer on Compare* CTC) [1]. Dzięki zastosowaniu tego trybu możliwe jest generowanie przerwań, z precyzyjnie nastawianym okresem. Do ustawienia przerwań, co jedną sekundę, wykorzystano wstępny dzielnik częstotliwości przebiegu zegara systemowego, ustawiony na podział przez 256 (bit kontrolny ang. *Clock Select* – **CS12**) oraz rejestr kontrolny (ang. *Input Capture Register*) **ICR1**, do którego wpisano zadaną wartość przechwycenia przez licznik1. W przypadku odliczenia przez licznik wartości zadanej (zapisanej w rejestrze **ICR1 –** 62499) następuje spełnienie przyczyny przerwania i ustawienie flagi (ang. *Input Capture Flag*) **ICF1**. Inicjalizację licznika1 zamieszczono poniżej:

```
void init_TIMER1 (void) {
#if HAPSIM
    TCCR1B = _BV(CS10)|_BV(WGM13)|_BV(WGM12); //fclk/1,
#else
    TCCR1B = _BV(CS12)|_BV(WGM13)|_BV(WGM12); //fclk/256,
#endif
    ICR1 = 62499;   // przechwycenie co 256*(62499+1)/16e6=1s
    TCNT1 = 0x0000; // zerowanie wartości zliczonej przez licznik
// selektywne zezwolenie na przerwania od doliczenia do zadanej wartości
// licznika1 - TICIE1=1
    TIMSK |= _BV(TICIE1);
};
```

W programie obsługi przerwania, którego źródłem jest przechwycenia dla licznika1 oprogramowano inkrementację elementów tablicy `cyfry[4]` odpowiednio modulo 10 albo 6.

```
ISR (TIMER1_CAPT_vect) {    //przerwanie wywoływane co 1 s
    cyfry[3]++;
    if (cyfry[3] > 9) {
        cyfry[3] = 0;
```

```
            cyfry[2]++;
            if(cyfry[2]>5) {
                cyfry[2]=0;
                cyfry[1]++;
                if(cyfry[1]>9){
                    cyfry[1]=0;
                    cyfry[0]++;
                    if (cyfry[0]>5)
                        cyfry[0]=0;
                };
            };
        };
};
```

10.4. Wytyczne dotyczące modyfikacji oprogramowania przedstawionego w poprzedniej instrukcji

1. Zmienić zakres zmienności cyfr: `cyfry[0]` i `cyfry[2]` na od 0 do 5.
2. W definicje zmiennych `cyfry[4]` należy wprowadzić specyfikator `volatile`, ponieważ są one modyfikowane zarówno w podprogramie obsługi przerwania, jak i w programie głównym.
3. Przypisać niższy priorytet dla programu obsługi przerwania od przepełnienia licznika0 w stosunku do programu obsługi przerwania od przechwycenia dla licznika1 (na początku funkcji `ISR (TIMER0_OVF_vect)`, należy globalnie zezwolić na przerwania – `sei();`).
4. Dołączyć w module main.c przedstawione wyżej funkcje: `init_TIMER1` i `ISR(TIMER1_CAPT_vect).`
5. Wywołać funkcję inicjalizującą licznik1 – `init_TIMER1` w programie głównym przed globalnym zezwoleniem na przerwania.

10.5. Przebieg ćwiczenia

W ramach ćwiczenia laboratoryjnego należy uruchomić oprogramowanie uwzględniające wytyczne zawarte w poprzedniej oraz w bieżącej instrukcji z wykorzystaniem układu dydaktycznego AVR_edu i uwzględnieniem wskazówek prowadzącego.

10.6. Przygotowanie do zajęć

Przygotowanie do zajęć obejmuje:

1. Przyniesienie wydrukowanej niniejszej instrukcji (jednej na grupę laboratoryjną 2–3 osobową).
2. Szczegółowe zrozumienie rozwiązań programowych opisanych w instrukcji.
3. Zapoznanie się z dokumentacją mikrokontrolera ATmega128 [1] w zakresie konfiguracji licznika1, ze szczególnym uwzględnieniem przyjętych w funkcji `init_TIMER1` ustawień.
4. Zrealizowanie samodzielne wszystkich modyfikacji wyszczególnionych w poprzedniej instrukcji i dla chętnych w bieżącej instrukcji.

10.7. Literatura

[1] Dokumentacja mikrokontrolera ATmega 128 – http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf

11. Obsługa wyświetlaczy alfanumerycznych LCD w języku C z wykorzystaniem mikrokontrolera AVR

11.1. Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z zasadą działania i programową obsługą sterownika HD44780 firmy Hitachi wyświetlacza ciekłokrystalicznego (ang. *Liquid Crystal Display* – LCD).

11.2. Opis sterownika HD44780 firmy Hitachi wyświetlaczy alfanumerycznych LCD

11.2.1. Informacje ogólne

Najbardziej popularnymi wyświetlaczami LCD są wyświetlacze tekstowe ze sterownikiem zgodnym z układem HD44780 firmy Hitachi. Głównymi zaletami ich stosowania są: niskie zużycie energii – szczególnie ważne w urządzeniach przenośnych, kompaktowa budowa – zajmują bardzo mało miejsca, stosunkowo niska cena, bardzo duża popularność wśród projektantów systemów mikroprocesorowych, a co się z tym wiąże, duża dostępność gotowego oprogramowania ich obsługi dla różnych procesorów. Zasadniczymi wadami są: mały zakres temperatur pracy, w którym mogą pracować oraz mała odporność na uszkodzenia mechaniczne. Wyświetlacz powinien być zabezpieczony na przykład osłoną wykonaną ze szkła akrylowego (inaczej pleksi).

W układzie dydaktycznym AVR_edu zastosowano wyświetlacz monochromatyczny z podświetleniem, o dwudziestu znakach w czterech liniach. Sterownik wyświetlacza jest zgodny ze standardem sterownika HD44780. Połączony jest on z mikrokontrolerem za pośrednictwem ośmiobitowej magistrali danych, ze sterowaniem kierunkiem przesyłu danych (podłączona linia sterująca R/W). Zastosowany sposób podłączenia wyświetlacza umożliwia testowanie wszystkich dostępnych konfiguracji połączenia go z mikrokontrolerem.

11.2.2. Opis wyprowadzeń oraz schemat połączenia wyświetlacza z mikrokontrolerem

Sterownik wyświetlacza HD44780 jest sterowany napięciami o poziomach zgodnych ze standardem TTL. Wartości napięć wejściowych dla stanu niskiego wynoszą 0 ÷ 0,6V oraz 2,2 ÷ 5V dla stanu wysokiego.

Znaczenie poszczególnych sygnałów sterujących wyświetlaczem [7]:

— Wyprowadzenia DB0 ÷ DB7 stanowią szynę danych, umożliwiającą wysyłanie i odbieranie instrukcji oraz danych przez moduł wyświetlacza. Jest to możliwe przez zastosowanie wyprowadzeń trójstanowych. Ponadto wyprowadzenie DB7 umożliwia sprawdzenie, czy sterownik wyświetlacza zrealizował poprzednią operację (ang. *busy flag*). Jeśli zostanie odczytany stan wysoki flagi zajętości, oznacza to, że wyświetlacz wykonuje operacje wewnętrzne i nie jest gotowy na przyjęcie kolejnej danej lub rozkazu. Dopiero odczyt

stanu niskiego oznacza gotowość do przyjęcia kolejnej instrukcji czy danej. Magistrala danych podłączono do portu A mikrokontrolera (rys. 8.1, rys. 11.1).

— Wyprowadzenie R/W steruje kierunkiem przesyłania danych i instrukcji. Jeżeli jest w stanie niskim, to wyświetlacz jest ustawiony w tryb zapisu danej lub instrukcji. Natomiast podanie stanu wysokiego na te wyprowadzanie spowoduje przejście wyświetlacza w tryb wysyłania informacji. Wyprowadzenie to podłączono do PE4 (rys. 8.1, rys. 11.1).

— Stan wejścia RS definiuje, czy przesyłana informacja jest daną czy instrukcją. W przypadku przesyłania danej należy wejście ustawić w stan wysoki, natomiast jeśli wyświetlacz ma odebrać rozkaz lub wystawić stan flagi zajętości, należy wymusić stan niski. Wyprowadzenie to podłączono do PE2 (rys. 8.1, rys. 11.1).

— Ustawienie stanu wysokiego na linii zezwalającej E powoduje odczyt przez sterownik stanów pozostałych linii sterujących i magistrali danych. Opadające zbocze na wejściu E powoduje zapis lub odczyt informacji z/do sterownika HD44780. Wyprowadzenie to podłączono do PE3 (rys. 8.1, rys. 11.1).

Tabela 11.1

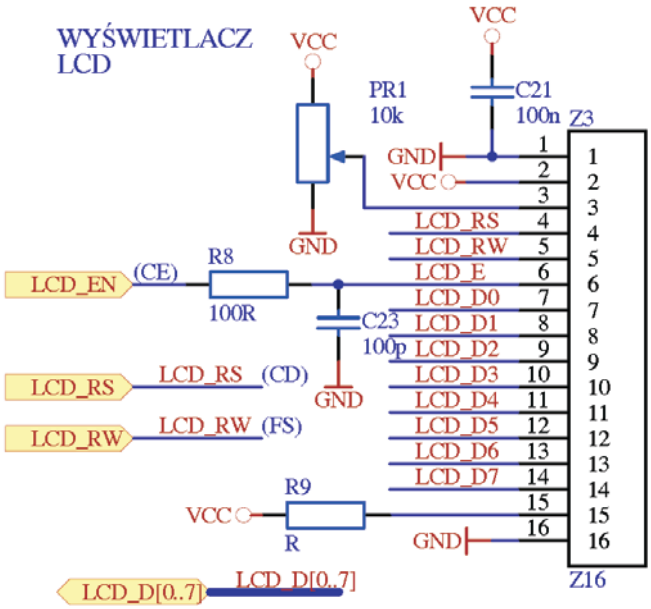
Funkcje wyświetlacza w zależność od stanów logicznych na liniach RS i R/W [7]

RS	R/W	Operacja
0	0	zapis instrukcji (np. czyszczenie wyświetlacza)
0	1	odczyt flagi zajętości (ang. *Busy Flag*)
1	0	zapis danej do DDRAM lub CGRAM
1	1	odczyt danej z DDRAM lub CGRAM

Tabela 11.2

Opis wyprowadzeń wyświetlacza

Nr wyprowadzenia	Symbol	Poziom napięcia, aktywne zbocze	Opis
1	Vss	0 V	Masa sygnałowa (ang. *ground*)
2	VDD	5.0 V (3 V)	Napięcie zasilania układu logicznego
3	VO	(zmienne)	Napięcie do regulacji kontrastu
4	RS	H/L	H: przesył danych L: przesył kodu instrukcji
5	R/W	H/L	H: Odczyt przez mikrokontroler informacji ze sterownika LCD L: Zapis przez mikrokontroler informacji do sterownika LCD
6	E	H/L, ⌐	Sygnał (ang. *enable*) zezwalający na odczyt linii sterujących i magistrali danych
7	DB0	H/L	0 bit danych
8	DB1	H/L	1 bit danych
9	DB2	H/L	2 bit danych
10	DB3	H/L	3 bit danych
11	DB4	H/L	4 bit danych
12	DB5	H/L	5 bit danych
13	DB6	H/L	6 bit danych
14	DB7	H/L	7 bit danych
15	A		Anoda diody podświetlającej LED
16	K		Katoda diody podświetlającej LED



Rys. 11.1. Schemat przedstawiający złącze wyświetlacza alfanumerycznego

11.2.3. Zależności czasowe dla wymiany informacji ze sterownikiem HD44780

W celu prawidłowego przesłania danych lub rozkazów do sterownika wyświetlacza, należy spełnić określone w dokumentacji [7] wymogi dotyczące kolejności i czasów ustawiania sygnałów na poszczególnych wejściach sterownika HD44780 (tab. 11.3 i rys. 11.2). W nawiasach podano ustawione w programie wartości generowanych opóźnień, które gwarantowały poprawną obsługę sterownika wyświetlacza. Zapis danej lub rozkazu do wyświetlacza powinien zacząć się od odpowiedniego ustawienia poziomu na linii RS i stanu niskiego na linii R/W. Jednocześnie linia E powinna pozostać w stanie niskim. Następnie po czasie minimum 40 ns należy wygenerować impuls na linii E trwający co najmniej 230 ns. Najlepiej niezwłocznie po omówionym ustawieniu linii sterujących (RS, R/W i E) należy wystawić daną (dla RS = 1) lub instrukcję (dla RS=0) na magistralę danych (linie DB0..DB7). Informacje te należy wystawić na linię danych minimum na 80 ns przed pojawieniem się opadającego zbocza na linii zezwalającej E. Podczas opadającego zbocza na linii E dana lub rozkaz zostaną zapisane do sterownika HD44780. Powinny one zostać podtrzymane przez 10 ns po wygenerowaniu aktywnego zbocza opadającego na linii E. Przed wysłaniem kolejnych informacji (danej lub instrukcji) należy odczekać czas niezbędny do wykonania przez wyświetlacz wewnętrznych operacji (związanych z przetwarzaniem bieżącego zadania lub danej) lub sprawdzać stan pinu DB7 (flagę zajętości). Dopiero po wykonaniu operacji wewnętrznych przez sterownik wyświetlacza możliwe jest rozpoczęcie kolejnego cyklu zapisu danej lub instrukcji. W przypadku interfejsu czterobitowego, po przesłaniu pierwszej połowy bajtu, można niezwłocznie przesłać drugą jego połowę. Dopiero po przesłaniu całego bajtu informacji sterownik rozpoczyna jego przetwarzanie.

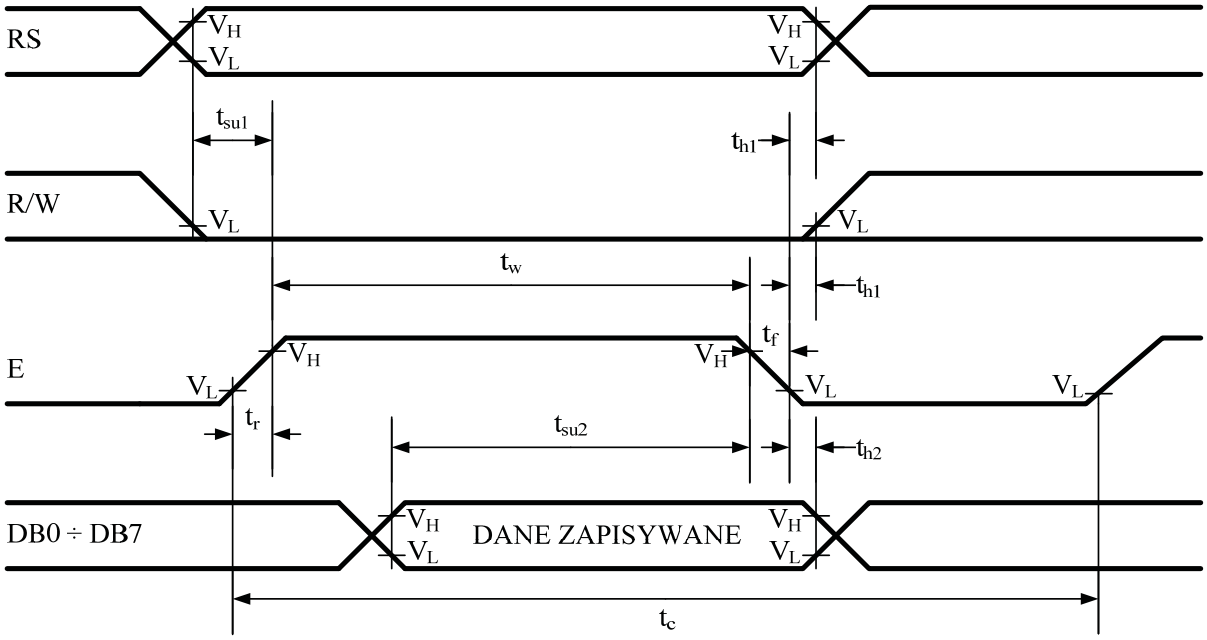
W przypadku odczytu danej lub flagi zajętości sekwencja zmian stanów sygnałów sterujących jest analogiczna (tab. 11.4 i rys. 11.3). Najpierw należy ustawić stan na linii RS i stan wysoki na linii R/W. Następnie wygenerować impuls na wejściu E. Maksymalnie po 160 ns od narastającego zbocza na linii zezwalającej E na magistrali danych (linie DB0..DB7) powinny pojawić się dane do odczytu. Należy pamiętać, iż w przypadku realizacji kolejno następujących po sobie odczytów np. stanu flagi zajętości nie można ich realizować częściej, niż co 0,5 µs.

— Tryb zapisu informacji do wyświetlacza (ang. *Write Mode*) [7]

Tabela 11.3

Minimalne czasy trwania poszczególnych stanów dla prawidłowego zapisu informacji do sterownika wyświetlacza LCD i napięcia zasilania 4,5 ÷ 5,5 V [7]

Charakterystyka	Symbol	Czas minimalny	Czas maksymalny
Czas trwania cyklu dla sygnału E	t_c	500 ns	–
Opadanie/wzrastanie napięcia dla sygnały E	t_r, t_f		20 ns
Szerokość impulsu E	t_w	230 (437) ns	
Czas ustalania sygnałów RS i R/W	t_{su1}	40 (62,5) ns	
Czas zatrzymania sygnału RS i R/W	t_{h1}	10 ns	
Czasu ustawiania danych	t_{su2}	80 ns	
Czas zatrzymania danych	t_{h2}	10 ns	



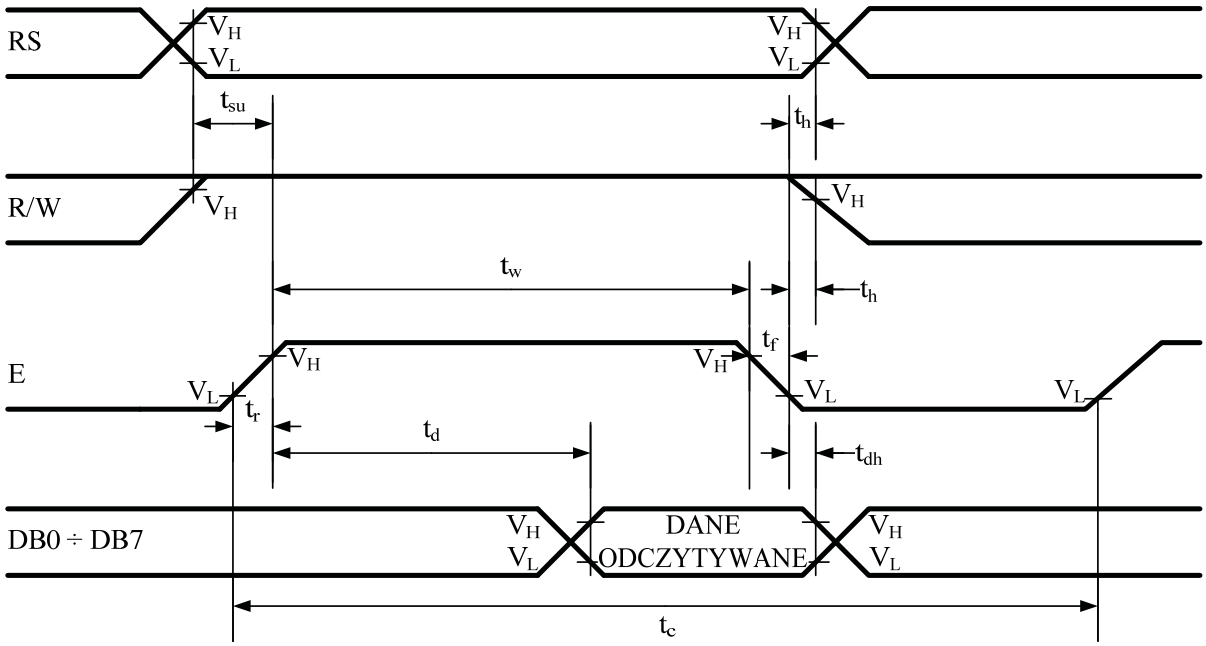
Rys. 11.2. Przebiegi sygnałów sterujących oraz magistrali danych w przypadku zapisu informacji do sterownika HD44780 [7]

— Tryb odczytu informacji z wyświetlacza (ang. *Read Mode*) [7]

Tabela 11.4

Minimalne czasy trwania poszczególnych stanów sygnałów dla prawidłowego odczytu informacji od sterownika wyświetlacza LCD [7]

Charakterystyka	Symbol	Czas minimalny	Czas maksymalny
Czas trwania cyklu dla sygnału E	t_c	500 (875) ns	–
Opadanie/wzrastanie napięcia dla sygnały E	t_r, t_f		20 ns
Szerokość impulsu E	t_w	230 (375) ns	
Czas ustalania sygnałów RS i R/W	t_{su}	40 (62,5) ns	
Czas zatrzymania sygnału RS i R/W	t_h	10 ns	
Czasu opóźnienia wysłania danych	t_d		160 ns
Czas zatrzymania danych	t_{dh}	5 ns	



Rys. 11.3. Przebiegi sygnałów sterujących oraz magistrali danych w przypadku odczytu informacji od sterownika HD44780 [7]

Zestawienie opisów flag zawartych w tab. 11.5:

— funkcja ENTRY MODE SET:
  – I/D=1: +1(zwiększanie adresu), I/D=0: –1(zmniejszanie adresu),
  – S=0: bez przesuwania, S=1: z przesuwaniem;

— funkcja DISPLAY ON/OFF:
  – D=0: wyświetlacz wygaszony, D=1: wyświetlacz włączony,
  – C=0: kursor wygaszony, C=1: kursor włączony,
  – B=0: wyłączona funkcja migania kursora lub znaku, B=1: włączona funkcja migania kursora lub znaku;

— funkcja COURSOR & DISPLAY SHIFT:
  – S/C = 0: przesuwanie kursora, S/C = 1: przesuwanie napisu,
  – R/L = 0: przesuwanie w lewo, S/C = 1: przesuwanie w prawo;

— funkcja SET:
  – DL=1: 8-bitowa szyna danych, DL=0: 4-bitowa szyna danych,
  – N=0: wyświetlanie w trybie jednoliniowym, N=1: wyświetlanie w trybie dwuliniowym,
  – F=0: matryca znaku 5×7 punktów (najczęściej stosowana), F=1: matryca znaku 5x10 punktów;

— funkcja READ BUSY FLAG:
  – BF=0: moduł wyświetlacza gotowy na następną instrukcję, BF=1: moduł wyświetlacza zajęty przetwarzaniem poprzedniej instrukcji.

Tabela 11.5

Zestawienie instrukcji sterownika wyświetlacza LCD wraz z czasami niezbędnymi na ich wykonywanie [7]

Instrukcja	Kod instrukcji										Opis	Czas wykon.
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Wyczyszczenie wyświetlacza LCD i ustawienie kursora na adresie 0	1,52 ms
Return Home	0	0	0	0	0	0	0	0	1	*	Ustawienie kursora w górnym lewym rogu i ustawienie adresu DDRAM na 0. Zawartość DDRAM pozostaje niezmieniona	1,52 ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Wskazanie kierunku przesuwu kursora oraz włączenie przesuwania napisu	37 µs
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Ustawienie wyświetlacza (D), kursora (C) i funkcji migania (B)	37 µs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	*	*	Przesunięcie kursora, przy jednoczesnym zachowaniu zawartości DDRAM	37 µs
Function Set	0	0	0	0	1	DL	N	F	*	*	Ustawienie długości interfejsu, liczby linii wyświetlacza oraz typu wyświetlanej czcionki	37 µs
Set CGRAM Adress	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Ustawienie adresu w CGRAM. Zapisy danej po tej instrukcji dotyczą CGRAM	37 µs
Set DDRAM Adress	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Ustawienie adresu w DDRAM. Zapisy danej po tej instrukcji dotyczą DDRAM	37 µs
Read Busy Flag and Adress	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Sprawdzenie, czy wyświetlacz wykonuje jakąś wewnętrzną instrukcję oraz odczytanie aktualnego adresu w pamięci	1 µs
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Zapis do aktualnie zaadresowanej pamięci RAM	37 µs
Read Data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Odczyt z aktualnie zaadresowanej pamięci RAM	37 µs

11.2.4. Zastosowane pamięci w sterowniku HD44780

Sterownik wyświetlacza wyposażony jest w trzy rodzaje pamięci wewnętrznych. Najbardziej obszerną pamięcią jest pamięć CGROM (ang. *Character Generator ROM*). Jest to wbudowana tablica ze znakami dla matrycy o rozmiarach 5 na 7 punktów i 5 na 10 punktów. Kody ASCII obejmujące litery alfabetu łacińskiego, cyfry i znaki specjalne odpowiadają adresom w pamięci CGROM. Na przykład, w celu zaadresowania cyfry 0 należy ustawić adres 0×30, który jest jej kodem ASCII (rys. 11.4, kolumna numer 3 – bardziej znaczący półbajt i wiersz numer 0 – mniej znaczący półbajt). Dane zawarte w pamięci CGROM nie ulegają wykasowaniu po wyłączeniu zasilania (są zapisane trwale w procesie produkcyjnym). Pod pierwszymi szesnastoma adresami nie wprowadza się żadnych znaków, adresy te są zarezerwowane dla znaków przechowywanych w pamięci CGRAM (ang. *Character Generator RAM*).

Rys. 11.4. Przykładowa zawartość pamięci CGROM sterownika HD44780 [7]

W pamięci CGRAM możliwe jest zdefiniowanie tablicy ośmiu własnych znaków. W odróżnieniu od CGROM, informacje zawarte w CGRAM muszą być za każdym razem definiowane po wyłączeniu i włączeniu zasilania. Pamięć CGRAM ma pojemność 64 bajtów i przeznaczona jest dla znaków zdefiniowanych przez użytkownika. Można w niej zdefiniować osiem znaków. Każdy bajt definiuje zaczernienia w jednym wierszu matrycy znaku. Znak opisuje się za pomocą kolejnych ośmiu bajtów, zaczynając od górnego wiersza. Istotnych jest tylko pięć młodszych bitów z każdego z bajtów, ponieważ matryca składa się z 5 kolumn i 7 wierszy. Ósmy bajt nie jest wliczany do opisu (jest on przeznaczony na kursor). Ułatwia to adresowanie poszczególnych znaków w obrębie tej pamięci, ponieważ do ich adre-

sowania wykorzystywane są tylko trzy najstarsze bity z adresu w pamięci CGRAM. Na rys. 11.5 przedstawiono ideę projektowania własnych znaków [7]. W celu zaczernienia danego piksela w matrycy znaku, należy wpisać wartość jeden pod odpowiadającym mu adres bitu w właściwym bajcie definiującym ten znak.

adresy CGRAM						dane CGRAM								
5	4	3	2	1	0	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	*	*	*	0	0	1	0	0	pierwszy znak
			0	0	1	*	*	*	0	1	1	1	0	
			0	1	0	*	*	*	0	1	0	1	0	
			0	1	1	*	*	*	0	1	0	1	0	
			1	0	0	*	*	*	0	1	0	1	1	
			1	0	1	*	*	*	0	1	1	1	0	
			1	1	0	*	*	*	0	0	1	0	0	
			1	1	1	*	*	*	0	0	0	0	0	kursor
0	0	1	0	0	0	*	*	*	0	0	1	0	0	drugi znak
			0	0	1	*	*	*	0	0	1	0	0	
			0	1	0	*	*	*	1	1	1	1	1	
			0	1	1	*	*	*	0	0	0	0	0	
			1	0	0	*	*	*	1	1	1	1	1	
			1	0	1	*	*	*	0	0	1	0	0	
			1	1	0	*	*	*	0	0	1	0	0	
			1	1	1	*	*	*	0	0	0	0	0	kursor

Rys. 11.5. Projektowanie znaków w pamięci CGRAM [7]

Pamięć „wyświetlania” DDRAM (ang. *Display Data RAM*) zawiera adresy znaków umieszczonych w pamięci CGROM lub CGRAM (najczęściej są to kody ASCII znaków). Jej pojemność jest większa od liczby pól odczytowych wyświetlacza. Ciąg znaków w niej zawarty jest wyświetlany tylko w pewnej części. Możliwe jest więc zapisanie do niej komunikatów, a następnie za pomocą funkcji przesuwania tekstu wyświetlenia ich. Pamięć ta może być zapisywana i odczytywana przez mikroprocesor jak zewnętrzna pamięć RAM. Sposób adresowania pól odczytowych w zastosowanym wyświetlaczu przedstawiono w tab. 11.6.

Tab. 11.6

Przypisanie adresów w kodzie szesnastkowym w pamięci DDRAM dla poszczególnych pól odczytowych zastosowanego wyświetlacza 4x20 znaków [7]

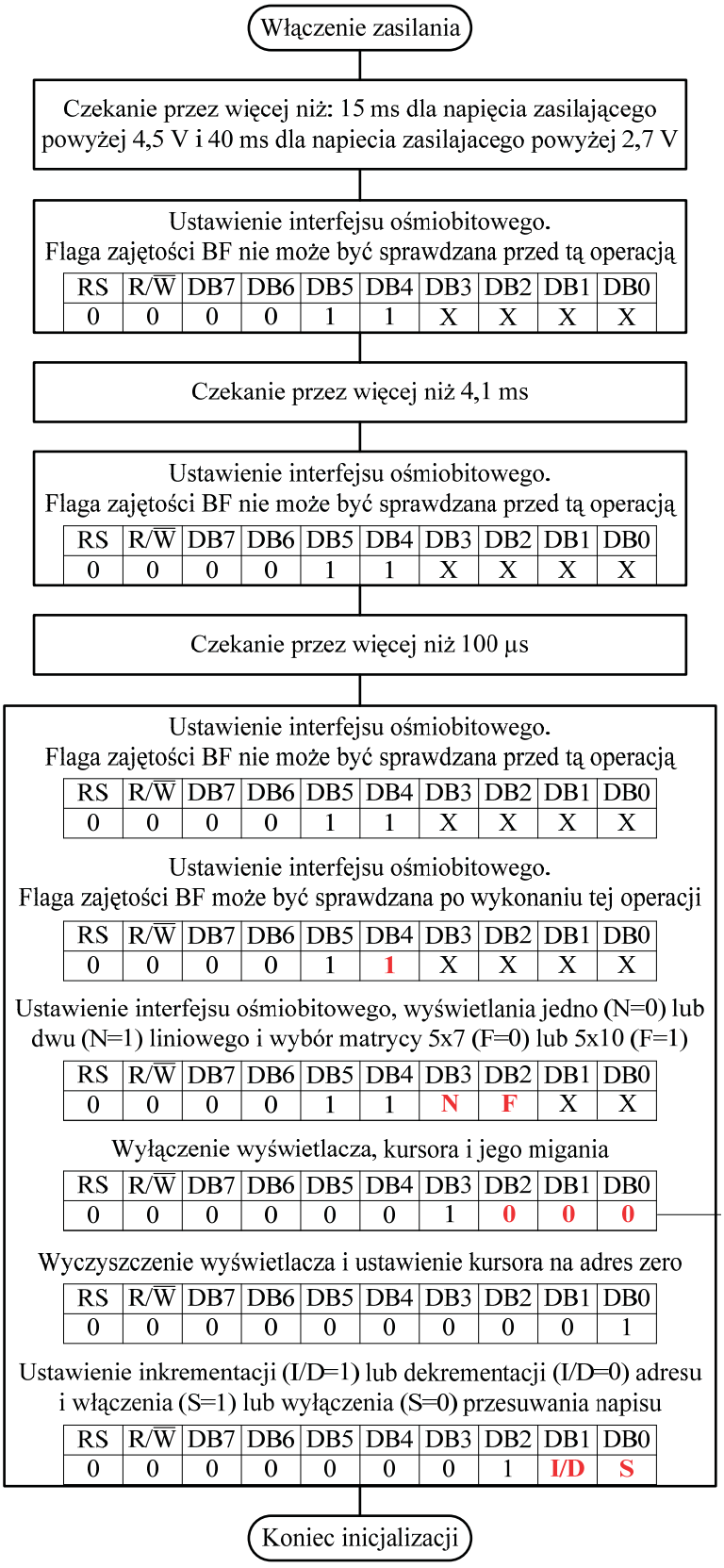
Kolumna	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Wiersz
Adres w DDRAM	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	1
	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	2
	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	3
	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67	4

11.2.5. Inicjalizacja sterownika HD44780

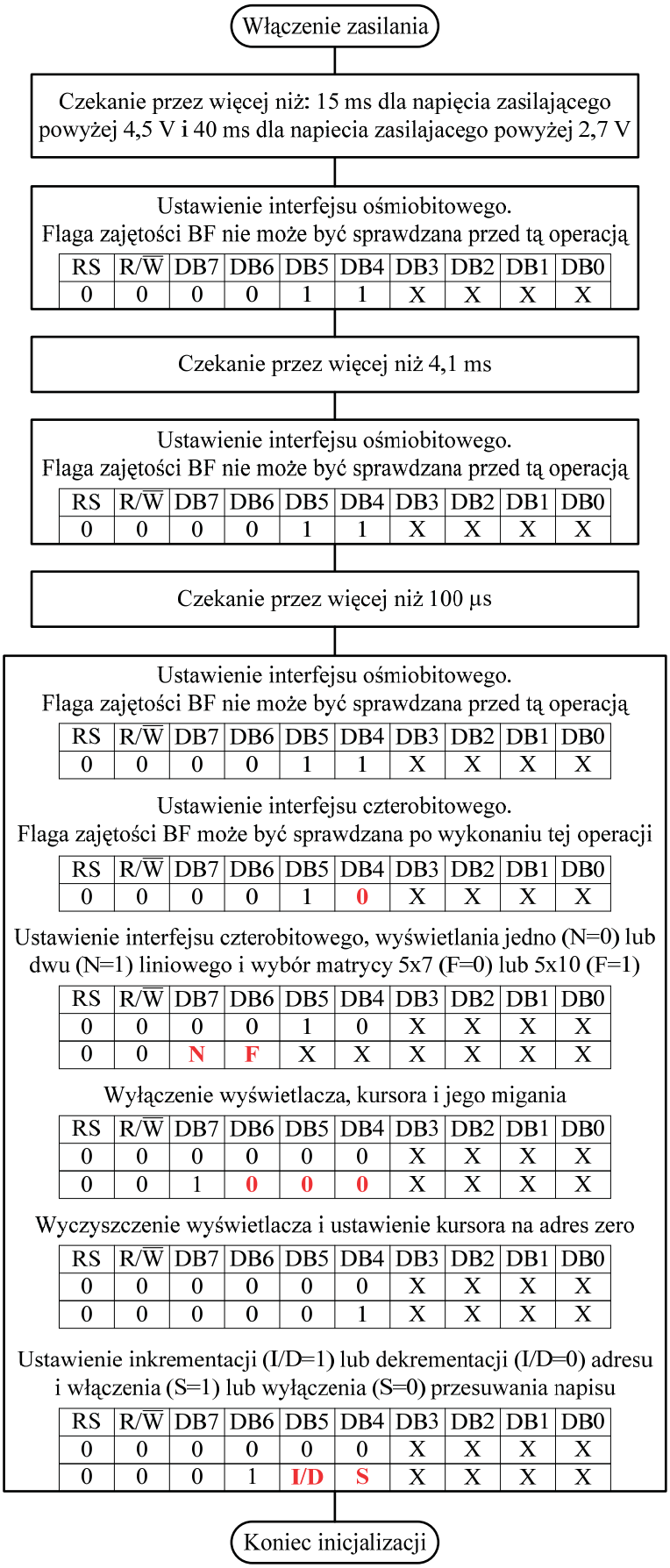
Sterownik posiada wbudowany układ generacji sygnału zerującego. Poprawna inicjalizacja układu następuje w przypadku wzrostu napięcia zasilania od 0,2 V do 4,5 V w przedziale czasu od 0,1 ms do 10 ms. Po włączeniu zasilania sterownik wyświetlacza przez 10 ms wykonuje automatyczną procedurę inicjalizacji. Podczas inicjalizacji wyświetlacz nie przyjmuje żadnych rozkazów, dlatego mikroprocesor powinien rozpocząć komunikację z wyświetlaczem po upływie czasu inicjalizacji (bezpiecznie jest przyjąć opóźnienie 15 ms dla napięcia zasilania 5V). Nie będąc pewnym, czy pojawienie się napięcia zasilania spełnia powyższe wymagania, można zrealizować programową inicjalizację modułu. Po auto-inicjalizacji wartości rejestrów wewnętrznych są skonfigurowane w następujący sposób:

- Wykonanie instrukcji DISPLAY CLEAR:
  - Pamięć wyświetlania DDRAM jest wypełniona spacjami, wskaźnik adresu DDRAM wyzerowany, pamięć CGRAM jest wypełniona przypadkowymi danymi.
- Wykonanie instrukcji FUNCTION SET:
  - DL=1: 8-bitowa szyna danych,
  - N=0: wyświetlanie w trybie 1 liniowym,
  - F=0: matryca znaku 5x7 punktów;
- Wykonanie instrukcji DISPLAY ON/OFF:
  - D=0: wyświetlacz wygaszony,
  - C=0: kursor wygaszony,
  - B=0: wyłączona funkcja migania kursora lub znaku;
- Wykonanie instrukcji ENTRY MODE SET:
  - I/D=1: inkrementacja adresu,
  - S=0: bez przesuwania.

Po inicjalizacji należy nadać rejestrom wartości pożądane dla danej aplikacji. Poniżej (rys. 11.6 i rys. 11.7) przedstawiono procedury inicjalizacji programowej modułu wyświetlacza w przypadku cztero i ośmiobitowej magistrali danych. Należy zwrócić uwagę, iż początkowo po wysłaniu instrukcji do sterownika wyświetlacza nie można programowo sprawdzać stanu flagi zajętości. W obu przypadkach trzykrotnie wysyłana jest instrukcja ustawiająca ośmiobitową magistrale danych, dopiero czwarte przesłanie tej instrukcji ustala tryb czterobitowy lub ośmiobitowy.



Rys. 11.6. Inicjalizacja programowa modułu wyświetlacza dla interfejsu ośmiobitowego [7]



Rys. 11.7. Inicjalizacja programowa modułu wyświetlacza dla interfejsu czterobitowego [7]

11.3. Opis projektu

Niniejszy projekt, podobnie jak poprzedni, jest uproszczoną, programową realizacją zegara. Poszczególne cyfry zapisane w tablicy `cyfry[4]` przechowują kolejno wartości: dziesiątek minut, jedności minut, dziesiątek sekund i jedności sekund. Po przyciśnięciu jednego z przycisków (od KEY0 do KEY3) następuje inkrementacja modulo 10 lub 6 jednej z czterech cyfr. Czas wizualizowany jest za pomocą wyświetlacza ciekłokrystalicznego.

W projekcie zawarto obsługę sterownika wyświetlacza. Oprogramowanie podzielono na moduły.

W pliku main.c zawarto między innymi program główny, w którym wywołana jest funkcja `window_1`, która umożliwia prezentację następującego komunikatu:

PIERWSZY TEKST
UZYSKANY NA
WYŚWIETLACZU
ALFANUMERYCZNYM

Następnie wywoływana jest funkcja `window_2`, która realizuje napis:

NASTAWIONY CZAS
WYNOSI:

Kolejną wywoływaną funkcją w programie głównym jest funkcja `window_3`, która odpowiada za wizualizacje poszczególnych cyfr czasu.

Funkcje inicjalizacji licznika0 i licznika1 oraz podprogramów obsługi przerwań pozostały bez zmian w stosunku do poprzedniego oprogramowania.

11.4. Opis programowania

W pliku IN_OUT.h zawarto definicje, które uwzględniają schematu połączeń zastosowany w układzie dydaktycznym AVR_edu.

W pliku IN_OUT.c zdefiniowano funkcję inicjalizującą kierunki poszczególnych wyprowadzeń mikrokontrolera:

```
#include "IN_OUT.h"
void init_pins (void) {
    /*ustawienie odpowiednich pinów mikrokontrolera jako
    wyjść dla portu, do którego podłączono diody LED*/
    DIRLED |= (_BV(LED0)|_BV(LED1)|_BV(LED2)|_BV(LED3));
    /*ustawienie odpowiednich pinów mikrokontrolera jako
    wejść dla portu, do którego podłączono przyciski KEY*/
    DIRKEY &= (~_BV(KEY0)&~_BV(KEY1)&~_BV(KEY2)&~_BV(KEY3));
    /*ustawienie odpowiednich pinów mikrokontrolera jako
    wejść dla portu, do którego podłączono przełączniki SW*/
    DIRSWITCH &= (~_BV(SW0)&~_BV(SW1)&~_BV(SW2)&~_BV(SW3));
    /*ustawienie wyjść dla portu, do którego podłączono linie danych LCD*/
    LCD_DIR_DATA = 0xFF;
    /*ustawienie odpowiednich pinów mikrokontrolera jako
    wyjść dla portu, do którego podłączono sygnały sterujące LCD*/
    LCD_DIR_CONTROL |= (_BV(LCD_RS)|_BV(LCD_EN)|_BV(LCD_R_W));
}
```

Plik LCD.h zawiera makra, w których zdefiniowano ustawienia stanów magistrali danych przy wywoływaniu instrukcji dla sterownika HD44780. Zapisano też makra, które definiują stany linii sterujących, w przypadku realizacji odczytu lub zapisu dla instrukcji lub danych. W pliku tym zadeklarowano ponadto wszystkie funkcje wykorzystywane do obsługi sterownika wyświetlacza LCD oraz zestaw polskich znaków wykorzystywany w aplikacji:

```
/* instrukcje wyświetlacza LCD */
```

```
#define LCD_MODE   0x06  // I/D=1 - inkrementacja, S=0 - brak
                         //  przesunięcia zawartości DDRAM
#define LCD_CLEAR  0x01  // czyści wyświetlacz i ustawia kursor na
                         // początku (adres=0)
#define LCD_ON     0x0c  // D=1 - włączenie wyświetlacza, C=0 - wyłączenie
                         // kursora, B=0 - wyłączenie migania kursora
#define LCD_OFF    0x08  // D=0 - wyłączenie wyświetlacza, C=0 - wyłączenie
                         // kursora, B=0 - wyłączenie migania kursora
#define LCD_8bit_data  0x38 // DL=1 - 8 linii danych, N=1 - dwu liniowy,
                            // F=0 - matryca 5x7 punktów
// ustawienie sygnałów sterujących dla rozpoczęcia zapisu instrukcji
#define LCD_WR_INST    LCD_CONTROL&=(~_BV(LCD_RS)&~_BV(LCD_R_W))
// ustawienie sygnałów sterujących dla rozpoczęcia zapisu danej
#define LCD_WR_DATA    LCD_CONTROL|=_BV(LCD_RS); LCD_CONTROL&=~_BV(LCD_R_W)
// ustawienie sygnałów sterujących dla rozpoczęcia odczytu flagi zajętości
#define LCD_RD_FLAG    LCD_CONTROL|=_BV(LCD_R_W); LCD_CONTROL&=~_BV(LCD_RS)
// ustawienie sygnałów sterujących dla rozpoczęcia odczytu danej
#define LCD_RD_DATA    LCD_CONTROL|=(_BV(LCD_RS)|_BV(LCD_R_W))
// ustawienie sygnału sterującego EN
#define LCD_EN_SET     LCD_CONTROL|=_BV(LCD_EN)
// wyzerowanie sygnału sterującego EN
#define LCD_EN_CLEAR   LCD_CONTROL&=~_BV(LCD_EN)
```

W pliku LCD.c zawarto funkcje wykorzystywane do obsługi sterownika wyświetlacza. Funkcja `check_busy_flag` odczytuje stan flagi zajętości sterownika wyświetlacza, aż do uzyskania informacji o przetworzeniu przez sterownik poprzednio wysłanej instrukcji lub danej. Wysoki stan logiczny odczytanej flagi informuje o zajętości sterownika realizacją poprzedniej instrukcji. W celu spełnienia uwarunkowań czasowych wykorzystano instrukcję asemblera `asm("nop")`, która realizuje opóźnienie o jeden okres przebiegu zegarowego (1/16 MHz = 62,5 ns).

```
void check_busy_flag (void)    {
char flag=0x80;
    LCD_RD_FLAG;         // ustawienie sygnałów sterujących dla
                         // rozpoczęcia odczytu flagi zajętości
    LCD_DIR_DATA=0x00; // ustawienie wejść dla portu, do którego podłączono
                         // linie danych LCD
    while (flag!=0){
        LCD_EN_SET;      // ustawienie linii EN
        asm("nop");      // wymagane wydłużenie impulsu do 230ns 1/16MHz <
                         // 62,5ns 6*62,5ns = 375ns
        asm("nop");      // jednocześnie wymagany odczyt po minimum
                         // 160 ns < 4*62,5ns = 250ns
        asm("nop");
        flag=LCD_DATA_IN&_BV(7); // wyzerowanie bitów poza flagą zajętości
        LCD_EN_CLEAR;            // zerowanie linii EN
        asm("nop");              // wymagane wydłużenie cyklu do
                                 // 500 ns < 14*62,5ns = 875ns
        asm("nop");
        asm("nop");
    };
    LCD_DIR_DATA=0xFF;           // ustawienie wyjść dla portu, do którego
                                 // podłączono linie danych LCD
};
```

Funkcja `send_LCDdata` realizuje przesłanie od mikrokontrolera danej do pamięci DDRAM lub CGRAM sterownika HD44780.

```
void send_LCDdata (unsigned char LCDdata) { //wysłanie bajtu do LCD
    check_busy_flag();      // sprawdzenie flagi zajętości
    LCD_WR_DATA;            // ustawienie sygnałów sterujących dla
                            // rozpoczęcia zapisu danej
// po 1 cyklu zegarowym czyli 1/16MHz=62.5us > 40us (minimalny dop. czas)
    LCD_EN_SET;             // ustawienie linii EN
    LCD_DATA_OUT = LCDdata;// przesłanie bajtu danych do LCD_DATA_OUT
    asm("nop");             // wymagane wydłużenie impulsu do
    asm("nop");         // 230 ns < 7*62,5ns = 437ns
    asm("nop");
```

```
    asm("nop");
    LCD_EN_CLEAR;       // zerowanie linii EN
};
```

Funkcja `send_LCDinstr` realizuje przesłanie instrukcji od mikrokontrolera do sterownika HD44780.

```
void send_LCDinstr (unsigned char LCDinstr) {    //wysłanie bajtu do LCD
    check_busy_flag();      // sprawdzenie flagi zajętości
    LCD_WR_INST;            // ustawienie sygnałów sterujących dla
                            // rozpoczęcia zapisu instrukcji
// po 1 cyklu zegarowym czyli 1/16MHz=62.5us > 40us (minimalny dop. czas)
    LCD_EN_SET;             // ustawienie linii EN
    LCD_DATA_OUT = LCDinstr;// przesłanie instrukcji do LCD_DATA_OUT
    asm("nop");             // wymagane wydłużenie impulsu do
    asm("nop");             // 230 ns < 7*62,5ns = 437ns
    asm("nop");
    asm("nop");
    LCD_EN_CLEAR;           // zerowanie linii EN
};
```

Funkcja `send_LCDinstr_without_check_flag` jest wykorzystywana do przesyłania przez mikrokontroler instrukcji do wyświetlacza podczas jego inicjalizacji programowej, gdy nie ma możliwości sprawdzania flagi zajętości. Różni się więc względem funkcji `send_LCDinstr` brakiem wywoływania funkcji sprawdzającej stan flagi zajętości.

Funkcja `define_char` wraz z funkcją `send_gcram` umożliwiają definicję zbioru ośmiu polskich znaków zawartych w tablicy `chtab[8]`. W pętli `for` przeszukiwana jest tablica `chtab`, w przypadku zgodności poszukiwanego znaku z zapisanym pod daną pozycją w tablicy `chtab` następuje przesłanie do pamięci CGRAM ośmiu bajtów danych definiujących znaleziony znak. W ten sposób dla poszczególnych znaków uzyskano te same wartości adresów w pamięci CGRAM i indeksów w tablicy `chtab` (wykorzystano to w kolejnej funkcji `send_LCDdata_pol`). Ideę projektowania dowolnych znaków przedstawiono na rys. 11.5. Zaczernienie piksela w matrycy następuje po wpisaniu pod odpowiadający mu adres w odpowiednim dla niego bajcie wartości logicznej „1” (rys. 11.5 i tab. 11.7).

Tabela 11.7

Projekt litery Ą

Zaczernienia	BIN	DEC
■ □ □ ■ ■	0b00001110	14
■ ■ ■ ■ ■	0b00010001	17
■ ■ ■ ■ ■	0b00010001	17
■ ■ ■ ■ ■	0b00010001	17
■ ■ ■ ■ ■	0b00011111	31
■ ■ ■ ■ ■	0b00010001	17
■ ■ ■ ■ ■	0b00010001	17
■ ■ ■ □ ■	0b00000010	2

```
void define_char(void) {          // Funkcja definiuje polskie znaki
    unsigned char i;
for (i=0; i<8; i++) {
/* Niepotrzebne definicje można wprowadzić do komentarza, aby zaoszczędzić
około 20 bajtów programu na każdej definicji) */
    send_LCDinstr (i*8+0x40);       //Ustawianie adresu dla pamięci CGRAM
    if        (chtab[i] == 'Ą') {send_gcram(14,17,17,17,31,17,17,2);  }//Ą
```

```
    else if   (chtab[i] == 'Ć') {send_gcram(2,14,21,16,16,17,14,0);   }//Ć
    else if   (chtab[i] == 'Ę') {send_gcram(31,16,16,30,16,16,31,2);  }//Ę
    else if   (chtab[i] == 'Ł') {send_gcram(16,16,18,20,24,16,31,0);  }//Ł
    else if   (chtab[i] == 'Ń') {send_gcram(2,21,25,21,19,17,17,0);   }//Ń
    else if   (chtab[i] == 'Ó') {send_gcram(2,14,21,17,17,17,14,0);   }//Ó
    else if   (chtab[i] == 'Ś') {send_gcram(2,14,16,14,1,1,30,0);     }//Ś
    else if   (chtab[i] == 'Ż') {send_gcram(31,1,2,31,8,16,31,0);     }//Ż
//  else if   (chtab[i] == 'ą') {send_gcram(0,0,14,1,15,17,15,2);     }//ą
//  else if   (chtab[i] == 'ć') {send_gcram(2,4,14,16,16,17,14,0);    }//Ćć
//  else if   (chtab[i] == 'ę') {send_gcram(0,0,14,17,31,16,14,2);    }//ę
//  else if   (chtab[i] == 'ł') {send_gcram(12,4,6,12,4,4,14,0);      }//ł
//  else if   (chtab[i] == 'ń') {send_gcram(2,4,22,25,17,17,17,0);    }//Ńń
//  else if   (chtab[i] == 'ć') {send_gcram(2,4,14,16,16,17,14,0);    }//Ćć
//  else if   (chtab[i] == 'ś') {send_gcram(2,4,15,16,14,1,30,0);     }//Śś
//  else if   (chtab[i] == 'ź') {send_gcram(2,4,31,2,4,8,31,0);       }//Źź
//  else if   (chtab[i] == 'ż') {send_gcram(4,0,31,2,4,8,31,0);       }//ż
    else     {send_gcram(21,10,21,10,21,10,21,10);}; //szachownica
};
};
```

Funkcja `send_gcram` zapisuje osiem bajtów definiujących jeden znak w pamięci CGRAM. Przesłanie danych odbywa się do pamięci CGRAM, ponieważ jej wywołanie jest poprzedzone wysłaniem instrukcji ustawienia adresu dla tej pamięci (poprzez przesłanie instrukcji *Set CGRAM Adress* – tab. 11.5). W procesie inicjalizacji ustawiono automatyczną inkrementację adresów, zatem po przesłaniu jednego bajtu danych realizowana jest generacja adresu dla kolejnego bajtu.

```
void send_gcram (unsigned char v0,unsigned char v1,unsigned char v2, un-
signed char v3,unsigned char v4,unsigned char v5,unsigned char v6, unsigned
char v7) {
send_LCDdata(v0);send_LCDdata(v1);send_LCDdata(v2);send_LCDdata(v3);
send_LCDdata(v4);send_LCDdata(v5);send_LCDdata(v6);send_LCDdata(v7);
}
```

Funkcja `send_LCDdata_pol` umożliwia wysyłanie zdefiniowanych w pamięci CGRAM polskich znaków. W przypadku wyspecyfikowania przez użytkownika kodu ASCI o wartości większej niż 127 następuje nadpisanie argumentu wejściowego rozpoznanym adresem znaku zdefiniowanym w pamięci CGRAM.

```
void send_LCDdata_pol(char LCDdata) {// wysłanie bajtu do LCD
unsigned char j;                     // z uwzględnieniem polskich znaków
    if (LCDdata>127)                 // jeśli kod ASCII znaku wyższy od 127
    {                                // oznacza to jeden z polskich znaków
        for (j=0;j<8;)
        {
            if (LCDdata == chtab [j]){// zmiana wartości argumentu LCDdata
                LCDdata = j;          // z kodu ASCII znaku na
                break;                // adres od 0 do 7 w pamięci CGRAM
            };
        j++;
        };
    };
    send_LCDdata(LCDdata);
};
```

Funkcja `init_LCD` realizuje inicjalizację programową sterownika wyświetlacza dla trybu ośmiobitowej magistrali danych (rys. 11.6).

```
void init_LCD (void) {        //inicjalizacja wyświetlacza LCD
// maksymalna wartość opóźnienia dla funkcji _delay_ms wynosi 6.5535 s z
//rozdzielczością 0.1 ms, a do (2^16-1)*4/fclk[kHz]=16.38375 ms z
//rozdzielczością 2.5e-4 ms
    _delay_ms(15);     //odmierzenie 15ms po włączeniu zasilania
//ustawia 8 bitową magistralę danych
    send_LCDinstr_without_check_flag(LCD_8bit_data);
//odmierzenie 4.1 ms po pierwszym wywołaniu instrukcji LCD_8bit_data
    _delay_ms(4.1);
```

```
//ustawia 8 bitową magistralę danych
     send_LCDinstr_without_check_flag(LCD_8bit_data);
//odmierzenie 100 us po drugim wywołaniu instrukcji LCD_8bit_data
     _delay_us(100);
//ustawia 8 bitową magistralę danych
      send_LCDinstr_without_check_flag(LCD_8bit_data);
//odmierzenie 37 us po drugim wywołaniu instrukcji LCD_8bit_data
      _delay_us(37);
//ustawia 8 bitową magistralę danych
      send_LCDinstr(LCD_8bit_data);
// D=0 - wyłączenie wyświetlacza, C=0 - wyłączenie kursora, B=0 -
// wyłączenie migania kursora
      send_LCDinstr(LCD_OFF);
//czyści wyświetlacz LCD
      send_LCDinstr(LCD_CLEAR);
// I/D=1 - inkrementacja, S=0 - brak przesunięcia zawartości DDRAM
      send_LCDinstr(LCD_MODE);
// D=1 - włączenie wyświetlacza, C=0 - wyłączenie kursora, B=0 - wyłączenie
// migania kursora
      send_LCDinstr(LCD_ON);
};
```

Funkcja `address_DDRAM` wyznacza adres komórki w pamięci DDRAM w przypadku zastosowanego wyświetlacza dla wprowadzanego numeru wiersza i kolumny definiującego pozycję znaku w polu odczytowym wyświetlacza (tab. 11.6).

```
unsigned char address_DDRAM (unsigned char x, unsigned char y){
unsigned char xy;
switch(x)
    {
            case 1:     xy=y-1;          break;
            case 2:     xy=0x40+y-1;     break;
            case 3:     xy=0x14+y-1;     break;
            case 4:     xy=0x54+y-1;     break;
            default:    xy=y-1;
    };
return xy;
};
```

Funkcja `write_string_xy` umożliwia (wykorzystując funkcję `send_LCDdata_pol`) wysyłanie do pamięci DDRAM ciągu znaków umieszczonych w pamięci programu (flash). W funkcji tej wykorzystano wskaźnik do ciągu znaków zapisanych w pamięci programu – `PGM_P text` oraz funkcję `pgm_read_byte`, która umożliwia odczyt bajtu zapisanego w pamięci programu [5]. Początkowo dla wskazanego numeru wiersza i kolumny wyliczany jest adres w pamięci DDRAM. Następnie jest on przesyłany do sterownika wyświetlacza (poprzez przesłanie instrukcji *Set DDRAM Adress* – tab. 11.5). W pętli `while` odczytywane są kolejne znaki z ciągu znaków zapisanego w pamięci programu, aż do uzyskania znaku o wartości zero (który występuje tylko jako ostatni w ciągu znaków). Odczytywane kolejno kody ASCII znaków przesyłane są jako dane do pamięci DDRAM (ponieważ przed ich wysyłaniem ustawiany był adres dla tej pamięci). Po wysłaniu danej do pamięci DDRAM, adres jest także automatycznie powiększany o jeden, wskazując kolejne pole odczytowe.

```
void write_string_xy(unsigned char x, unsigned char y, PGM_P text) {
//pisz tekst na LCD wskazywany wskaźnikiem *text
unsigned char nr = 0;
unsigned char xy;
volatile unsigned char zn;
    xy=address_DDRAM(x, y);
    send_LCDinstr((xy|0x80));    // ustalenie początku tekstu
    while ((zn = pgm_read_byte(&text[nr++])) > 0) {
        send_LCDdata_pol(zn);    // umieść wskazany znak tekstu na LCD
    };
};
```

Funkcja czas_print_xy umożliwia wizualizację minut i sekund oddzielonych znakiem „:”. W funkcji tej zrealizowano konwersje poszczególnych cyfr na ich kody ASCII poprzez dodanie do każdej z nich wartości 48.

```
void czas_print_xy (unsigned char x, unsigned char y) {
unsigned char xy;
     xy=address_DDRAM(x, y);
     send_LCDinstr((xy|0x80));      // ustalenie początku tekstu
     send_LCDdata(cyfry[0]+48);     // liczba dziesiątek minut
     send_LCDdata(cyfry[1]+48);     // liczba jedności minut
     send_LCDdata(0x3A);            // kod ASCII ':"
     send_LCDdata(cyfry[2]+48);     // liczba dziesiątek sekund
     send_LCDdata(cyfry[3]+48);     // liczba jedności sekund
};
```

Funkcje window_1 i window_2 są przykładowymi kompozycjami okien tekstowych, wizualizowanych za pomocą wyświetlacza alfanumerycznego. Wykorzystują one ciągi znaków zdefiniowane w pliku teksty.h.

```
void window_1(void) {
    send_LCDinstr(LCD_CLEAR); // czyści wyświetlacz LCD
    write_string_xy(1,4,o1_1);
    write_string_xy(2,5,o1_2);
    write_string_xy(3,5,o1_3);
    write_string_xy(4,3,o1_4);
};
void window_2(void) {
    send_LCDinstr(LCD_CLEAR); //czysc wyswietlacz LCD
    write_string_xy(1,1,o2_1);
    write_string_xy(3,1,o2_3);
    czas_print_xy(3,9);
};
```

Funkcja window_3 umożliwia wizualizację czasu w trzecim wierszu począwszy od dziewiątej kolumny:

```
void window_3(void) {
    czas_print_xy(3,9);
};
```

Zawartość pliku teksty.h przedstawiono poniżej. Zdefiniowano w nim poszczególne ciągi znaków zapisane w pamięci programu (flash). Są one wykorzystywane w funkcjach window_1 i window_2.

```
//okno1
const char o1_1[] PROGMEM = "PIERWSZY TEKST";
const char o1_2[] PROGMEM = "UZYSKANY NA";
const char o1_3[] PROGMEM = "WYŚWIETLACZU";
const char o1_4[] PROGMEM = "ALFANUMERYCZNYM";
//okno2
const char o2_1[] PROGMEM = "NASTAWIONY CZAS";
const char o2_3[] PROGMEM = "WYNOSI: ";
```

Plik deklara.h pozwala korzystać z funkcji lub zmiennych zdefiniowanych w innych niż je zadeklarowano modułach oprogramowania. Modyfikowana też jest wartość stałej HAPSIM w przypadku realizacji symulacji zaleca się ją ustawić na 1, a w przypadku testów z udziałem układu AVR_edu ustawić na 0. Zawartość pliku deklara.h przedstawiono poniżej:

```
#include <avr/pgmspace.h>
#include "IN_OUT.h"
#define HAPSIM 1
extern volatile signed char cyfry[4];   //wartości cyfr
extern void init_pins (void);
extern void init_TIMER0 (void);
extern void define_char(void);
extern void init_LCD (void);
extern void window_1(void);
```

```
extern void window_2(void);
extern void window_3(void);
```

11.5. Wytyczne dotyczące modyfikacji omówionego oprogramowania

Należy przeprogramować dołączony projekt, aby w sposób analogiczny do przedstawionego w poprzednim ćwiczeniu reagował na przyciśnięcie przycisków:

1. Przed pierwszym i co piątym naciśnięciem przycisku ENTER program powinien zapewnić wizualizację czterech cyfr zapisanych w tablicy `cyfry[4]` i nie reagować na przyciśnięcia pozostałych przycisków.
2. Po pierwszym i po piątym naciśnięciu przycisku ENTER pierwsza cyfra powinna zacząć migać (z częstotliwością około 2 Hz), informując użytkownika o jej modyfikacji, natomiast pozostałe powinny być wizualizowane bez migania..
3. Kolejne naciśnięcia przycisku ENTER mają powodować miganie kolejnych cyfr, zapewniające użytkownikowi sprzężenie zwrotne, informujące go, która z cyfr jest aktualnie modyfikowana.
4. Po wielokrotności pięciokrotnego naciśnięcia przycisku ENTER następuje zatwierdzenie zmodyfikowanej nastawy (składającej się z czterech cyfr) i nadpisanie jej na cyfry zapisanych w tablicy `cyfry[4]`. Po tej akcji żadna cyfra nie miga i modyfikacja nastawy realizowana jest od początku (po ponownym naciśnięciu przycisku ENTER modyfikowana jest pierwsza cyfra).
5. Modyfikacja poszczególnych cyfr ma być realizowana poprzez wciśnięcie przycisku UP (inkrementacja odpowiednio modulo 10 lub 6) i poprzez wciśniecie przycisku DOWN (dekrementacja odpowiednio modulo 10 lub 6).
6. Jeżeli podczas modyfikacji dowolnej z cyfr zostanie naciśniety przycisk CANCEL zmodyfikowane już cyfry nie powinny zostać zapamiętane, a wyświetlacz LCD powinien ponownie wizualizować nastawę sprzed anulowanej modyfikacji.

11.6. Wskazówki dotyczące modyfikacji dołączonego oprogramowania

Poniżej zostały wymienione wskazówki dotyczące modyfikacji oprogramowania. Kolorem czerwonym w proponowanych fragmentach oprogramowania zaznaczono miejsca, które należy uzupełnić.

1. Zaleca się pisanie i testowanie oprogramowania etapowo, na przykład początkowo można pominąć miganie edytowanych cyfr.
2. Zaleca się zdefiniowanie czteroelementowej tablicy (`volatile signed char cyfry_kopia[4]`), której elementy zostaną wykorzystane jako kopie podczas modyfikacji nastaw. Umożliwią one ponowne przywrócenie nastawy sprzed modyfikacji w przypadku naciśnięcia przez użytkownika przycisku CANCEL.
3. Zaleca się zdefiniowanie zmiennej (`char dzielnik_f`), która posłuży do programowego podzielenia częstotliwości przerwań w celu realizacji migania aktualnie edytowanych cyfr z częstotliwością 2 Hz.
4. Zaleca się zdefiniowanie zmiennej (`char miganie`), która będzie przyjmowała na przemian wartości 1 i 0 po osiągnięciu zaprogramowanej liczby przerwań (do ich odliczania proponuje się zastosowanie wyżej omówionej zmiennej). Jej wartość może zostać bezpośrednio wykorzystana w zaprojektowanej przez studentów funkcji `czas_kopia_print_xy` wywoływanej podczas modyfikacji dowolnej cyfry i realizującej miganie edytowanej cyfry. Brak wyświetlania cyfry za pomocą sterownika wyświetlacza można realizować, wpisując kod ASCII spacji.

5. Zaleca się zdefiniowanie zmiennej (`char menu`), której wartość będzie modyfikowana pod wpływem naciskania przycisków ENTER i CANCEL. Zmienna ta może przechowywać informacje, która z cyfr jest aktualnie modyfikowana.
6. W programie obsługi przerwania od przepełnienia licznika0 zaleca się realizację programowego dzielnika częstotliwości (wykorzystując zmienne `dzielnik_f` i `miganie`):

```
    dzielnik_f++;
     if(dzielnik_f==XXXX){ //częstotliwość migania modyfikowanej cyfry
                           //16e6/(1024*256*(2*XXXX)) ok. 2Hz
            dzielnik_f=0;
            miganie=!miganie;
  };
```

7. Funkcja `czas_kopia_print_xy` jest analogiczną do funkcji `czas_print_xy`, z tym, że wykorzystuje cyfry zapisane w tablicy `cyfry_kopia[4]` i realizuje miganie aktualnie edytowanej zmiennej. W funkcji tej zaleca się wykorzystanie informacji zawartych w zmiennej `miganie` oraz `menu`:

```
if((miganie)...(menu...))
    send_LCDdata(0x20);                 // kod ASCII " "
else
     send_LCDdata(cyfry_kopia[0]+48); // liczba dziesiątek minut
```

8. Funkcja `window_4` wywołuje wyżej opisaną funkcję `czas_kopia_print_xy`.
9. W nieskończonej pętli programu głównego zaleca się oprogramować edycje poszczególnych cyfr zgodnie z wytycznymi zawartym w poprzednim punkcie (rys. 9.11). Do tego celu proponuje się wykorzystanie instrukcji `switch(menu)`, której ciało może być takie samo, jak w przypadku oprogramowania z poprzedniej instrukcji.

11.7. Przebieg ćwiczenia

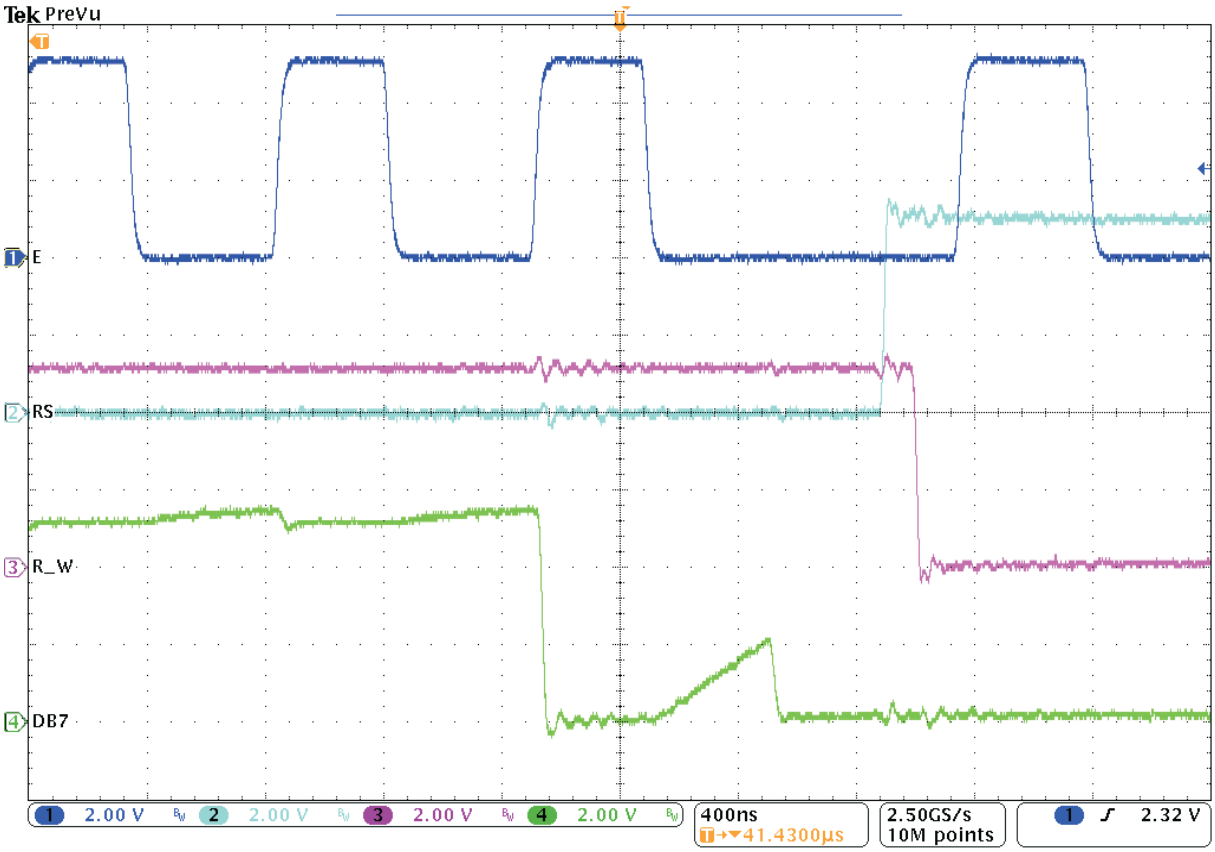
W ramach ćwiczenia laboratoryjnego należy uruchomić dołączone oprogramowanie z wykorzystaniem układu dydaktycznego AVR_edu i uwzględnieniem wskazówek prowadzącego. Następnie należy zmodyfikować dołączony projekt zgodnie z wytycznymi zawartymi powyżej (można wykorzystać zawarte powyżej wskazówki).

11.8. Przygotowanie do zajęć

Przygotowanie do zajęć obejmuje:

1. Przyniesienie wydrukowanej niniejszej instrukcji (jednej na grupę laboratoryjną 2–3 osobową).
2. Korzystając z literatury np. z [1, 2, 3, 7] i niniejszej instrukcji, zapoznać się z zasadą działania sterownika HD44780.
3. Szczegółowo zrozumieć oprogramowanie dołączone do instrukcji (można uwzględnić literaturę [1–7]).
4. Przeprowadzenie symulacji dołączonego oprogramowania (można wykorzystać projekt main_elf_symulacja.aps dla środowiska AVRStudio i gotową konfigurację dla programu HAPSIM).
5. Zaprojektowanie swojego własnego znaku użytkownika i zaprezentowanie go na wyświetlaczu LCD.
6. Wstępne przemyślenie modyfikacji dołączonego oprogramowania realizujące wyszczególnione powyżej wytyczne.
7. Przeanalizowanie zarejestrowanych oscyloskopem przebiegów sygnałów sterujących i najbardziej znaczącego bitu magistrali danych sterownika HD44780 (E, RS, R/W, DB7 –

rys. 11.8). Należy między innymi zwrócić uwagę na: okres odczytu flagi zajętości, czas trwania stanu wysokiego sygnału E, odczytywane stany logiczne flagi zajętości, kolejność ustawiania poziomów linii sterujących, zapisaną wartość logiczną na linii DB7.



Rys. 11.8. Przebiegi czasowe sygnałów sterujących i linii danych DB7 zmierzone dla przedstawionej w ćwiczeniu biblioteki obsługi sterownika HD44780

8. Dla chętnych: zrealizowanie samodzielne wszystkich modyfikacji wyszczególnionych powyżej (gwarantuje to sprawną realizację ćwiczenia na zajęciach laboratoryjnych).
9. Dla chętnych: na podstawie udostępnionego oprogramowania przebudowanie biblioteki obsługi wyświetlacza na realizującą jego obsługę za pomocą czterobitowej magistrali danych, bez sprawdzania flagi zajętości.

11.9. Literatura

[1] Elektronika dla Wszystkich, numery z 11.1997, 12.1997, 01.1998 i 03.1998

[2] R. Koppel: Programowanie procesorów w języku C, Elektronika dla Wszystkich, maj 2005 do maj 2007

[3] A. Witkowski: Mikrokontrolery AVR programowanie w języku C – przykłady zastosowań, Katowice 2006

[4] B. W. Kernighan, D. M. Ritchie: Język ANSI C

[5] Dokumentacja do biblioteki avr-libc, katalog z WinAVR\doc\avr-libc\avr-libc-user-manual\modules.html

[6] Dokumentacja mikrokontrolera ATmega 128 – http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf

[7] Dokumentacja sterownika HD44780 – HD44780.pdf

12. Obsługa przetwornika analogowo-cyfrowego i licznika1 mikrokontrolera ATmega128 umożliwiająca generowanie przebiegu prostokątnego o nastawianym współczynniku wypełnienia

12.1. Cel ćwiczenia

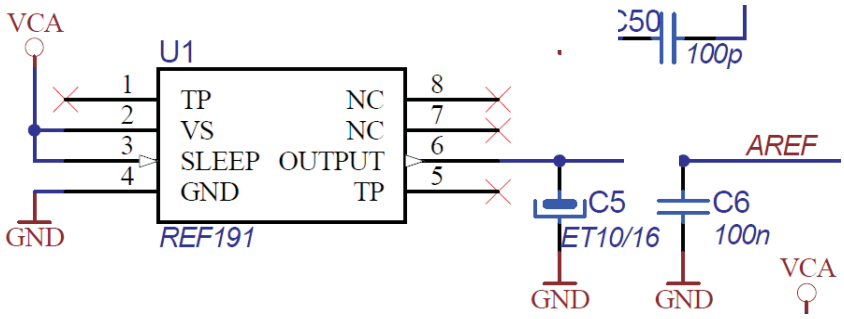
Celem ćwiczenia jest zapoznanie się z zasadą działania i programową obsługą wbudowanego przetwornika analogowo-cyfrowego oraz trybem PWM (ang. *Pulse Width Modulation*) licznika1. Ponadto istotne jest także utrwalenie i doskonalenie umiejętności Studentów związanych z programową obsługą sterownika wyświetlacza LCD – HD44780.

12.2. Przetwornik analogowo-cyfrowy

Wybrane właściwości wbudowanego przetwornika analogowo-cyfrowego [1]:

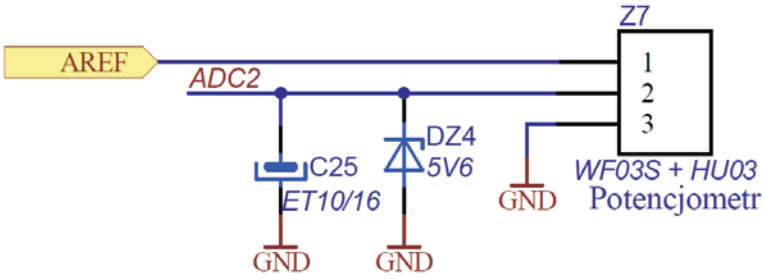
- dziesięciobitowa rozdzielczość,
- maksymalna częstotliwość próbkowania – ok. 77kHz (dla maksymalnej rozdzielczości od ok. 3,85kHz do ok. 15,3kHz),
- osiem multipleksowanych kanałów (brak możliwości jednoczesnego próbkowania kilku wejść analogowych),
- możliwość podłączenia zewnętrznego źródła napięcia odniesienia lub wybranie wbudowanego o wartości 2,56 V,
- możliwość pracy w trybie pojedynczej konwersji oraz realizowanej w sposób ciągły,
- możliwość zgłaszania żądania przerwania w przypadku końca konwersji.

W projekcie wykorzystano zewnętrzne napięcie referencyjne o wartości 2,048 V dla przetwornika analogowo – cyfrowego – układ scalony REF191 (rys. 12.1).



Rys. 12.1. Schemat ze źródłem napięcia odniesienia REF191

Napięcie nastawiane za pomocą potencjometru podłączono do kanału numer dwa przetwornika analogowo-cyfrowego (ADC2 – rys. 8.1). Potencjometr wieloobrotowy podłączono pomiędzy masę (GND – rys. 12.2), a napięcie referencyjne (AREF – rys. 12.2).



Rys. 12.2. Podłączenie potencjometru wieloobrotowego

Inicjalizację przetwornika analogowo-cyfrowego zawarto w pliku ADC.c:

```
void init_ADC (void) {
    ADMUX =_BV(MUX1);    // wewnętrzne napięcie odniesienia wyłączone,
                         // korzystamy z podłączonego do AREF
                         // uaktywniono kanał ADC2 z pomiarem napięcia
                         // zadawanego potencjometrem
    ADCSRA= _BV(ADEN)|   // włączenie przetwornika ADC
            _BV(ADSC)|   // pierwsze uruchomienie konwersji (trwa ona 25
                         // cykli zegara)
            _BV(ADIF)|   // wyzerowanie flagi przerwań od ADC
            _BV(ADIE)|   // zezwolenie na przerwania od końca konwersji ADC
            _BV(ADPS2)|  // ustawienie dzielnika częstotliwości na 128
            _BV(ADPS1)|  // częstotliwość zegara ADC wynosi
                         // 16MHz/128=125 kHz
            _BV(ADPS0);  // spełniono w ten sposób wymagania dla najwyższej
                         // dokładności ADC
                         // częstotliwość powinna zawierać się z granicach
                         // od 50 kHz do 200 kHz
};
```

Wartość cyfrowa uzyskiwana z przetwornika przechowywana jest w rejestrze **ADC**. Dla konfiguracji przetwornika realizowanej za pomocą funkcji `init_ADC` wartość cyfrową odpowiadającą przetwarzanemu napięciu można wyrazić następującą zależnością:

$$ADC = \frac{V_{IN} * 1024}{V_{REF}} = \frac{V_{IN} * 1024}{2048\ \text{mV}} = \frac{V_{IN}}{2\ \text{mV}} \qquad (12.1)$$

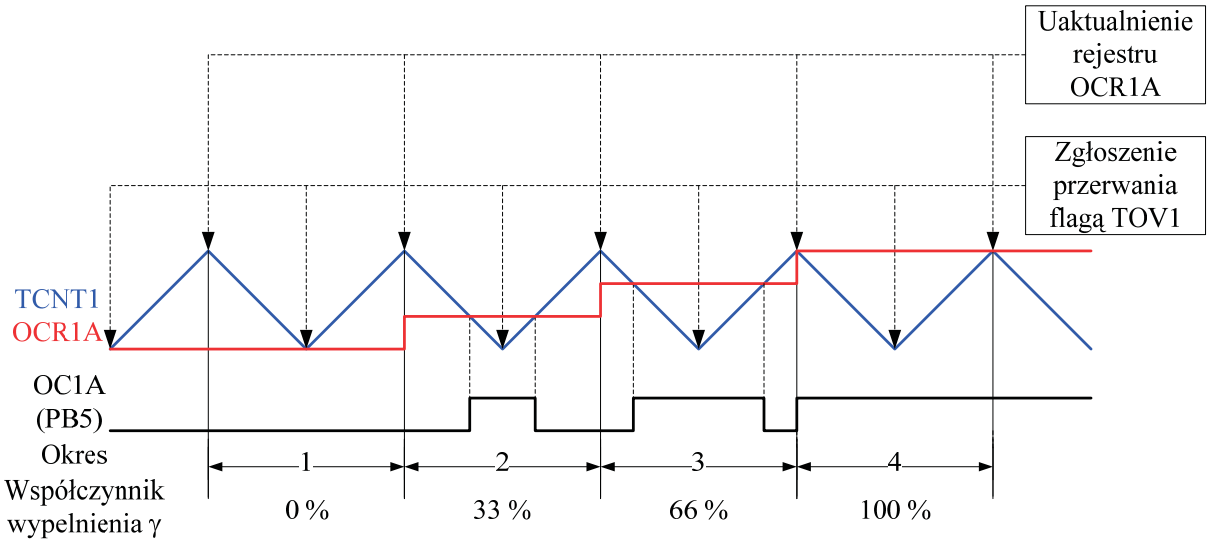
gdzie: V_{IN} – wartość napięcia wejściowego, V_{REF} – wartość napięcia referencyjnego.

Bardziej szczegółowe informacje dotyczące wbudowanego przetwornika analogowo-cyfrowego można znaleźć w dokumentacji [1].

12.3. Ustawienie dla licznika1 trybu PWM z poprawnie generowaną fazą

Poniżej opisano tryb PWM dla licznika1 ustawiony w procesie inicjalizacji za pomocą funkcji `init_TIMER1`. W omawianym trybie PWM licznik1 liczy cyklicznie od 0 w górę, a po osiągnięciu wartości 1023 (maksymalna wartość dziesięciobitowa) zmienia kierunek liczenia i liczy do 0. Współczynnik wypełnienia przebiegu prostokątnego na wyjściu OC1A (PB5) modyfikowany jest poprzez ustawianie odpowiedniej wartości w rejestrze **OCR1A** (rys. 12.3). Dla niezanegowanego trybu pracy wyjścia porównania (ustawionego w inicjalizacji) i w przypadku, gdy licznik1 liczy w górę oraz osiągnie wartość wpisaną do rejestru porównującego **OCR1A,** wyjście porównania OC1A przyjmuje stan niski. Natomiast, w przy-

padku liczenia w dół i uzyskaniu wartości wpisanej do rejestru porównania **OCR1A,** wyjście porównujące przyjmuje stan wysoki. W przypadku dziesięciobitowej rozdzielczości PWM dla wartości 1023 wpisanej do rejestru **OCR1A,** uzyskujemy współczynnik wypełnienia 100%, natomiast dla wartości 0 – 0% (rys. 12.3). W wybranym trybie pracy licznika rejestr z wartością porównywaną **OCR1A** jest buforowany. Jego wartość jest uaktualniana w przypadku osiągnięcia przez licznik wartości 1023. Buforowanie rejestru z wartością porównywaną **OCR1A** powoduje uzyskanie symetrycznego przebiegu na wyjściu porównania OC1A (PB5) względem wartości 0 licznika1 (rys. 12.3). Dla wartości zliczonej 0 generowana jest flaga przepełnienia licznika, którą wykorzystano do generowania żądania przerwania obsługiwanego za pomocą podprogramu `ISR(TIMER1_OVF_vect)`.



Rys. 12.3. Przebiegi czasowe wartości zliczanej przez licznik (TCNT1) i wyjścia PWM OC1A w przypadku poprawnie generowanej fazy przebiegu PWM [1]

Częstotliwość przerwań generowanych dla wyzerowania się licznika1, a tym samym częstotliwość nośna PWM w omawianym przypadku wynosi:

$$f_{PWM} = \frac{f_{clk_{I/O}}}{2N * TOP} = \frac{16\ \text{MHz}}{2*1*1023} \approx 7{,}82\ \text{kHz} \qquad (12.2)$$

gdzie: N – jest wartością wstępnego podziału częstotliwości przebiegu zegarowego zasilającego peryferia.

Pierwszą (jedyną w tym przypadku) instrukcją wykonywaną w podprogramie obsługi przerwania od licznika1 jest uruchomienie startu konwersji przetwornika analogowo-cyfrowego. Jest to stosunkowo prosty zabieg zapewniający synchronizację przebiegu PWM z chwilami próbkowania. Synchronizacja ta ma szczególne znaczenie w przypadku pomiaru prądu płynącego przez indukcyjności podłączone do urządzeń energoelektronicznych sterowanych impulsowo. Próbkując przebieg prądu w połowie trwania stanu wysokiego (lub niskiego) na wyjściu PWM, można z przybliżeniem założyć, iż uzyskuje się pomiar wartości średniej prądu w okresie modulacji. Przybliżenie to jest tym bardziej słuszne im wartość okresu modulacji jest relatywnie jak najmniejsza w stosunku do stałej czasowej obwodu LR.

```
ISR (TIMER1_OVF_vect) {
    ADCSRA |= _BV(ADSC); //start konwersji ADC
};
```

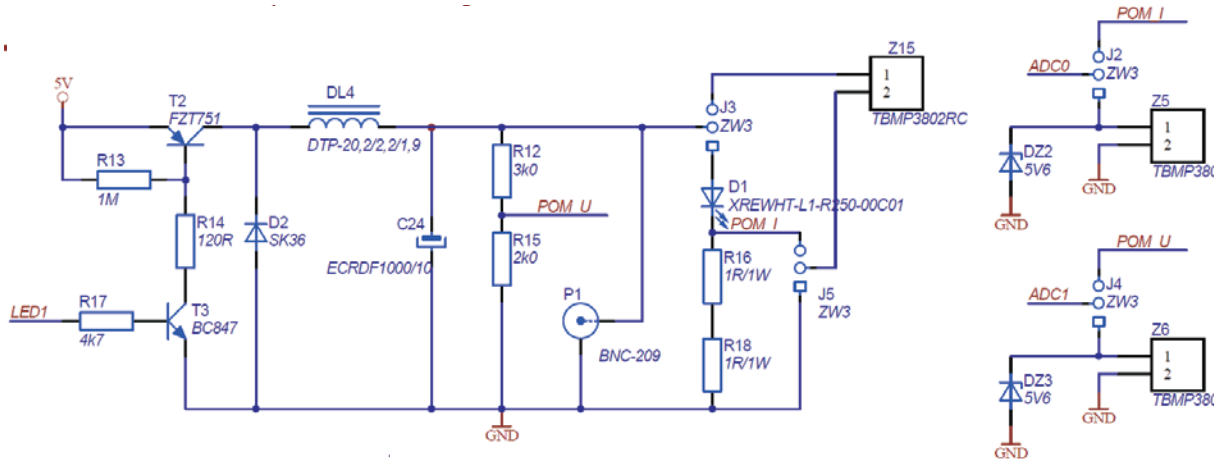
Inicjalizację licznika1 dla trybu PWM z poprawnie generowaną fazą i dziesięciobitową rozdzielczością zawarto w pliku PWM.c:

```
void init_TIMER1 (void) {
    TCCR1A = _BV(COM1A1)|    // po zrównaniu wartości zliczonej przez
                             // licznik1 w TCNT1 i wartości porównywanej
                             // zapisanej w rejestrze OCR1A
                             // następuje wyzerowanie wyjścia OC1A
    _BV(WGM11)|_BV(WGM10);   // ustawienie trybu dziesięciobitowego PWM
                             // licznika1 z poprawna fazą
    TCCR1B = _BV(CS10);      // brak podziału częstotliwości PWM -
                             // źródło zegara = clk_IO
    TCNT1=0x0000;            // wyzerowanie zliczonej wartości
    OCR1A=0x00;              // wyzerowanie rejestru z wartością porównywaną
    TIMSK |= _BV(TOIE1);     // zezwolenie na zgłaszanie przerwań przez
                             // licznik1 przy doliczeniu do 0
    TIFR = 0xFF;             // wyzerowanie flag przerwań od liczników
};
```

Szczegółowe informacje dotyczące licznika1 można znaleźć w [1].

12.4. Opis projektu

Projekt umożliwia sterowanie przetwornicą obniżającą napięcie (rys. 12.4). Współczynnik wypełnienia przebiegu sterującego przetwornicą zadawany jest za pomocą precyzyjnego, dziesięcioobrotowego potencjometru. Konfigurując odpowiednio zworki możemy podłączyć do przetwornicy diodę mocy LED (XREWHT-L1-R250-00C01 firmy CREE) (J3) i umożliwić pomiar prądu przez nią płynącego (J2).



Rys. 12.4. Schemat przetwornicy obniżającej napięcie wraz z diodą mocy LED

12.5. Opis oprogramowania

Podobnie jak w przypadku poprzedniego ćwiczenia oprogramowanie podzielono na moduły. W pliku main.c zawarto program główny. Podczas inicjalizacji wywoływane są funkcje: inicjalizujące kierunkowość wyprowadzeń, inicjalizujące sterownik wyświetlacza, definiujące zbiór polskich znaków, inicjalizujące przetwornik analogowo-cyfrowy i licznik1. Na końcu inicjalizacji globalnie zezwolono na przerwania, a następnie wywołano funkcję `window_1`, która umożliwia wyświetlenie następującego komunikatu tekstowego:

NASTAWIONA WARTOŚĆ
NAPIĘCIA: 0.000 V
WSP. WYPEŁ.: 0 %

W nieskończonej pętli programu głównego wywoływana jest funkcja window_2, która pozwala prezentować zadaną potencjometrem wartość napięcia, oraz proporcjonalny do niej zadany współczynnik wypełnienia dla przebiegu PWM. Definicje funkcji window_1 i window_2 zawarto w pliku LCD.c. Wykorzystują one ciągi znaków zdefiniowane w pliku teksty.h.

```
void window_1(void) {
    send_LCDinstr(LCD_CLEAR); // czyści wyświetlacz LCD
    write_string_xy(1,2,o1_1);
    write_string_xy(2,2,o1_2);
    print_integer(2,12,napiecie,1,3);
    write_string_xy(2,18,o1_2a);
    write_string_xy(3,2,o1_3);
    print_integer(3,15,wsp_wypelnienia,3,0);
    write_string_xy(3,19,o1_3a);
};
void window_2(void) {
    print_integer(2,12,napiecie,1,3);
    print_integer(3,15,wsp_wypelnienia,3,0);
};
```

Zawartość pliku teksty.h przedstawiono poniżej. Zdefiniowano w nim poszczególne ciągi znaków zapisane w pamięci programu (flash). Są one wykorzystywane w funkcjach window_1. Symulator VMLAB nie obsługuje pamięci CGRAM sterownika wyświetlacza. W celu wizualizacji tekstów bez polskich znaków na potrzeby symulacji w wymienionym środowisku zdefiniowano dwa zestawy ciągów znaków: pierwszy uwzględnia polskie znaki, natomiast drugi ich nie zawiera. Odpowiedni zestaw ciągów znaków wybiera się, ustawiając wartość stałej VMLAB zdefiniowanej w pliku deklara.h.

```
//okno1
#if VMLAB == 0
const char o1_1[] PROGMEM = "NASTAWIONA WARTOŚĆ";
const char o1_2[] PROGMEM = "NAPIĘCIA:";
const char o1_2a[] PROGMEM = "V";
const char o1_3[] PROGMEM = "WSP. WYPEŁ.:";
const char o1_3a[] PROGMEM = "%";
#else
const char o1_1[] PROGMEM = "NASTAWIONA WARTOSC";
const char o1_2[] PROGMEM = "NAPIECIA:";
const char o1_2a[] PROGMEM = "V";
const char o1_3[] PROGMEM = "WSP. WYPEL.:";
const char o1_3a[] PROGMEM = "%";
#endif
```

W pliku LCD.c zdefiniowano także dwie kolejne zamieszczone poniżej funkcje int_to_digits i print_integer. Funkcja int_to_digits realizuje konwersję zmiennej typu unsigned int na zbiór cyfr, dla których wskaźnik jest argumentem wejściowym funkcji.

```
void int_to_digits(unsigned int liczba, unsigned char liczba_cyfr,
                   signed char *pcyfry){
    unsigned char i;
    for(i=liczba_cyfr; i>=1;i--){ // generowanie adresów dla kolejnych
                                  // cyfr od najmniej znaczącej
        *(pcyfry+i-1)=liczba%10;  // dzielenie modulo 10
        liczba/=10;               // dzielenie przez 10
    };
};
```

Funkcja print_integer umożliwia wizualizację zmiennej typu unsigned int za pomocą cyfr traktowanych, jako część całkowita liczby (przed przecinkiem) oraz cyfr traktowanych jako ułamkowe (po przecinku). Cyfry o wartości zero od najbardziej znaczącej do przedostatniej z części całkowitej lub do pierwszej większej od zera wizualizowane są jako spacje. Najmniej znacząca cyfra z części całkowitej jest wizualizowana bez zmian.

W przypadku wizualizacji części ułamkowej cyfr, zostaje umieszczony separator dziesiętny (znak kropki), a następnie kolejne cyfry z części ułamkowej. W przypadku liczby mniejszej od zera, pierwsza wykryta cyfra różna od zera lub ostatnia z części całkowitej poprzedzona jest znakiem minus. W celu zapewnienia tej samej ilości znaków przesyłanych zarówno dla liczby dodatniej, jak i ujemnej, w przypadku liczby dodatniej zamiast znaku minus przesyłany jest znak spacji. Poniżej zamieszczono kilka przykładów prezentujących wyniki działania funkcji (tab. 12.1).

Tabela 12.1

Przykłady prezentujące wyniki działania funkcji `print_integer`

Liczba	Liczba cyfr całkowitych	Liczba cyfr ułamkowych	Wizualizacja liczby
0	3	0	0
0	3	2	0.00
20	3	0	20
–20	3	2	-0.20
12345	3	2	123.45
–12345	3	2	-123.45
–45	3	1	-4.5

```
void print_integer (unsigned char x, unsigned char y,signed int liczba,
                    unsigned char calkowite, unsigned char ulamkowe){
    unsigned char xy;
    unsigned char j;
    unsigned char znak=0;  // domyślna wartość znaku dla liczby dodatniej
    unsigned zero=1;       // domyślnie nie znaleziono cyfry różnej od zera
    signed char cyfry[5];  // tablica cyfr odpowiadających wprowadzanej
                           // liczbie
    xy=address_DDRAM(x, y);
    send_LCDinstr((xy|0x80));  //ustalenie początku tekstu
    if(liczba<0){
        liczba=-liczba;
        znak=1;                //zapamiętanie, iż liczba jest ujemna
    };
//zamiana liczby na odpowiadający jej zbiór cyfr
    int_to_digits(liczba, calkowite+ulamkowe, cyfry);
    for(j=1; j<calkowite;j++){     // pętla od najbardziej znaczącej cyfry
                                   // do przedostatniej z części całkowitej
        if((zero)&&(cyfry[j-1]>0)){
            zero=0;            // znaleziono pierwszą cyfrę większą od zera
            if(znak)
                send_LCDdata('-'); // jeżeli liczba jest ujemna przed
                                   // pierwszą cyfrą całkowitą jest
                                   // umieszczony znak "-"
            else                   // jeżeli liczba jest dodatnia przed
                                   // pierwszą cyfrą całkowitą jest
                send_LCDdata(' '); // umieszczony znak " "
            send_LCDdata(cyfry[j-1]+48);// konwersja, a następnie
                                        // wizualizacja pierwszej z części
        }                               // całkowitej cyfry
        else if(!zero)
            send_LCDdata(cyfry[j-1]+48);// wizualizacja kolejnych cyfr
                                        // części całkowitej po wykryciu
        else                            // pierwszej cyfry > 0
            send_LCDdata(' ');          // przed wykryciem pierwszej cyfry
                                        // części całkowitej > 0 jest
                                        // umieszczany znak " "
    };
    if(zero){
        if(znak)                // jeżeli nie wykryto cyfry > 0 przed
                                // ostatnią z części całkowitej jest
            send_LCDdata('-'); // umieszczony znak "-"
        else                    // jeżeli liczba jest dodatnia przed
```

```
                                // pierwszą cyfrą całkowitą jest
            send_LCDdata(' '); // umieszczony znak " "
    };
    send_LCDdata(cyfry[calkowite-1]+48); // konwersja i wizualizacja
                                         // ostatniej cyfry całkowitej
    if(ulamkowe>0){       //jeżeli funkcja ma wizualizować cyfry ułamkowe
        send_LCDdata('.');//należy przesłać po cyfrach całkowitych znak "."
        for(j=calkowite+1; j<=calkowite+ulamkowe;j++)
            send_LCDdata(cyfry[j-1]+48); //wizualizacja cyfr ułamkowych
    };
};
```

Plik deklara.h podobnie jak w poprzednio dołączonym oprogramowaniu pozwala korzystać z funkcji lub zmiennych zdefiniowanych w innych niż je zdefiniowano modułach oprogramowania. Modyfikowana jest też wartość stałej `VMLAB` w przypadku realizacji symulacji należy ją ustawić na 1.

W podprogramie obsługi przerwania `ISR(ADC_vect)` zrealizowano odczyt wartości cyfrowej uzyskiwanej z przetwornika analogowo-cyfrowego.

```
ISR (ADC_vect) {
unsigned int wartosc_cyfrowa;
        wartosc_cyfrowa=(ADC&0x03FF);
        napiecie=wartosc_cyfrowa<<1;
        wsp_wypelnienia=multiply_with_round(wartosc_cyfrowa,6406);
        OCR1A=wartosc_cyfrowa;
};
```

Na odczytywane dane z rejestru **ADC** nałożono maskę 0x03FF uwzględniającą dziesięciobitową rozdzielczość przetwornika:

```
        wartosc_cyfrowa=(ADC&0x03FF);
```

Wartość cyfrową przeliczono na wartość napięcia wyrażaną w miliwoltach przekształcając zależność (12.1):

```
napiecie=wartosc_cyfrowa<<1;
```

Wartość wpisywana do rejestru **OCR1A** ma w omawianym przypadku dokładnie taki sam zakres jak wartość cyfrowa odpowiadająca zadawanemu napięciu poprzez potencjometr (nie wymaga żadnych przeliczeń):

```
OCR1A=wartosc_cyfrowa;
```

Wyliczono zadany współczynnik wypełnienia wyrażany w procentach, który jest wprost proporcjonalny do nastawianej potencjometrem wartości napięcia.

$$\text{wsp\_wypelnienia} = \frac{ADC * \text{wsp\_wypelnienia}_{\max}}{ADC_{\max}} = \frac{ADC * 100}{1023} \approx ADC * 0{,}09775 \quad (12.3)$$

gdzie: $\text{wsp\_wypelnienia}_{\max}$– jest maksymalną wartością zadawanego współczynnika wypełnienia uzyskiwaną dla $ADC_{\max}$ – maksymalnej wartości cyfrowej z przetwornika analogowo – cyfrowego.

Operacje mnożenia liczb zmiennoprzecinkowych pochłaniają znacznie więcej czasu mikrokontrolera stałoprzecinkowego niż mnożenie liczb stałoprzecinkowych. Dlatego mnożenie liczb zmiennoprzecinkowych w równaniu (12.3) zostało zastąpione mnożeniem dwóch liczb stałoprzecinkowych. Czynnik zmiennoprzecinkowy wymnożono przez liczbę 2^{16} w celu zapewnienia wystarczającej dokładności reprezentacji stałej (po obcięciu części ułamkowej) i jednocześnie stosunkowo prostej realizacji mnożenia. Wartość stałej uzyskano na podstawie poniższej zależności:

$$\frac{2^{16} * \text{wsp\_wypelnienia}_{max}}{ADC_{max}} = \frac{2^{16} * 100}{1023} \approx 6406 \tag{12.4}$$

Przybliżając stałą zmiennoprzecinkową wartością 6406 w formacie 16.16 (format 16.16 wyjaśniono niżej), popełniono następujący błąd względny:

$$\frac{\frac{2^{16} * 100}{1023} - 6406}{\frac{2^{16} * 100}{1023}} * 100\% \approx 0{,}004\% \tag{12.5}$$

Uzyskana wartość błędu jest pomijalnie mała w stosunku do błędów wprowadzanych przez pomiar napięcia zadawanego potencjometrem.

Do mnożenia stałoprzecinkowego wykorzystano funkcję `multiply_with_round`. Funkcja ta jest wykorzystywana do mnożenia zmiennej typu `unsigned int` (szesnastobitowej) przez zmienną typu `unsigned long` (trzydziestodwubitową) z zaokrągleniem wyniku.

```
int multiply_with_round(unsigned int arg1, unsigned long arg2){
    union {
        unsigned int bits16[2];
        unsigned long bits32;
    } temp_union;
    temp_union.bits32=arg1*arg2;
    if(temp_union.bits16[0]&0x8000)
        temp_union.bits16[1]++;
    return temp_union.bits16[1];
};
```

W omawianym przypadku zmienna `arg1` zawiera tylko część całkowitą (format 16.0, czyli szesnaście bitów części całkowitej i zero bitów części ułamkowej), natomiast dla stałej przeznaczono szesnaście bitów na jej część całkowitą oraz szesnaście bitów na jej cześć ułamkową (format 16.16). Poniżej przedstawiono wagi poszczególnych bitów dla stałej bez znaku w formacie 16.16 (rys.).

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}	2^{-16}

Rys. 12.5. Wagi odpowiadające poszczególnym bitom w przypadku formatu bez znaku 16.16

W ogólnym przypadku format wyniku mnożenia binarnego dwu liczb jest następujący:

$$A.B * C.D = (A + C).(B + D) \tag{12.6}$$

gdzie: *A, B, C, D* – oznaczają odpowiednio długości części całkowitej i ułamkowej obu liczb.
W omawianym przypadku uzyskujemy następujący format wyniku :

$$16.0 * 16.16 = 32.16 \tag{12.7}$$

Wynik mnożenia w języku C ma długość dłuższego z argumentów, czyli trzydzieści dwa bity.

```
temp_union.bits32=arg1*arg2;
```

W ten sposób z wyniku mnożenia zostało odrzucone szesnaście najbardziej znaczących bitów, zatem uzyskany format wyniku to 16.16. W celu wyodrębnienia tylko części całkowitej wyniku mnożenia odrzucono programowo z wyniku szesnaście mniej znaczących bitów – `temp_union.bits16[0]`. Uzyskano zatem ostatecznie format wyniku 16.0.

```
return temp_union.bits16[1];
```

Zaokrąglenie zrealizowano, sprawdzając wartość najbardziej znaczącego bitu w odrzucanej części ułamkowej wyniku. Jeżeli wynosi on jeden, wówczas wartość wyniku jest inkrementowana.

```
if(temp_union.bits16[0]&0x8000)
        temp_union.bits16[1]++;
```

12.6. Wytyczne dotyczące modyfikacji omówionego oprogramowania

Należy opracować i uruchomić oprogramowanie, które umożliwi zadawanie współczynnika wypełnienia za pomocą czterech przycisków (w sposób analogiczny do omawianego w poprzednich ćwiczeniach). Ponadto należy oprogramować pomiar prądu płynącego przez diodę mocy LED dla zadanego współczynnika wypełnienia (z rozdzielczością 1 mA, prezentowanego w mA, ze spodziewaną wartością maksymalną ok. 500 mA). Przyciśnięcia przycisków mają zapewnić działanie programu opisane poniżej:

1. Przed pierwszym i co czwartym naciśnięciem przycisku ENTER program powinien zapewnić wizualizację współczynnika wypełnienia i prądu płynącego przez diodę mocy LED oraz nie reagować na przyciśnięcia pozostałych przycisków.
2. Po pierwszym i co czwartym naciśnięciu przycisku ENTER pierwsza cyfra współczynnika wypełnienia powinna zacząć migać (z częstotliwością około 2 Hz), informując użytkownika o jej modyfikacji.
3. Kolejne naciśnięcia przycisku ENTER mają powodować miganie kolejnych cyfr zapewniające użytkownikowi sprzężenie zwrotne informujące go, która z cyfr jest aktualnie modyfikowana. W przypadku wybrania przez użytkownika jako najbardziej znaczącej cyfry jeden, zmienna `wsp_wypelnienia` powinna przyjąć wartość sto i proces modyfikacji nastawy powinien zostać zakończony.
4. Po czterokrotnym naciśnięciu przycisku ENTER następuje zatwierdzenie zmodyfikowanej nastawy (składającej się z trzech cyfr) i nadpisanie jej na zmienną globalną `wsp_wypelnienia`. Po tej akcji żadna cyfra nie miga i modyfikacja nastawy realizowana jest od początku (po ponownym naciśnięciu przycisku ENTER modyfikowana jest pierwsza cyfra).
5. Modyfikacja poszczególnych cyfr ma być realizowana poprzez wciśnięcie przycisku UP (inkrementacja odpowiednio modulo 10 lub 2) i poprzez wciśniecie przycisku DOWN (dekrementacja odpowiednio modulo 10 lub 2).
6. Jeżeli podczas modyfikacji dowolnej z cyfr zostanie naciśniety przycisk CANCEL zmodyfikowane już cyfry nie powinny wpłynąć na zmienną `wsp_wypelnienia`, a wyświetlacz powinien ponownie wizualizować nastawę sprzed anulowanej modyfikacji.

12.7. Wskazówki dotyczące modyfikacji dołączonego oprogramowania

Poniżej zostały wymienione wskazówki dotyczące modyfikacji oprogramowania.

1. Zaleca się pisanie i testowanie oprogramowania etapami.
2. Zaleca się wykorzystanie podprogramu obsługi przerwania od przepełnienia się licznika0 (z poprzedniego ćwiczenia) w celu identyfikacji naciśnięcia przycisków. Zaleca się także wykorzystanie inicjalizacji licznika0, z tym, że zamiast stałej `HAPSIM` należy wprowadzić stałą `VMLAB`.
3. Zaleca się zdefiniować zmienną (`char dzielnik_f`), która posłuży do programowego podzielenia częstotliwości przerwań w celu realizacji migania aktualnie edytowanych cyfr.
4. Zaleca się zdefiniowanie zmiennej (`char miganie`), która będzie przyjmowała na przemian wartości 1 i 0 po osiągnięciu zaprogramowanej liczby przerwań (do ich odliczania

proponuje się zastosowanie wyżej omówionej zmiennej). Jej wartość może zostać wykorzystana w funkcji `print_three_digits` wywoływanej w funkcji `window_2`.

5. W programie obsługi przerwania zaleca się w sposób analogiczny do poprzednich zrealizować programowy dzielnik częstotliwości (wykorzystując zmienne `dzielnik_f` i `miganie`).
6. Zaleca się zdefiniowanie zmiennej (`char menu`), której wartość będzie modyfikowana pod wpływem naciskania przycisków ENTER i CANCEL. Zmienna ta może przechowywać informacje, która z cyfr jest aktualnie modyfikowana.
7. Zaleca się w sposób następujący przeprogramować funkcję `window_1`.

```
void window_1(void) {
    send_LCDinstr(LCD_CLEAR); //czyści wyświetlacz LCD
    write_string_xy(1,2,o1_1);
    write_string_xy(2,2,o1_2);
    print_integer(2,14,wsp_wypelnienia, ..., ...);
    write_string_xy(2,19,o1_2a);
    write_string_xy(3,2,o1_3);
    write_string_xy(4,4,o1_4);
    print_integer(4,10,prad, ..., ...);
    write_string_xy(4,15,o1_4a);
};
```

8. W pliku teksty.h zaleca się stworzenie następujących ciągów znaków:

```
#if VMLAB ==0
const char o1_1[] PROGMEM = "NASTAWIONA WARTOŚĆ";
const char o1_2[] PROGMEM = "WSP. WYPEŁ.:";
const char o1_2a[] PROGMEM = "%";
const char o1_3[] PROGMEM = "ZMIERZONA WARTOŚĆ";
const char o1_4[] PROGMEM = "PRĄDU:";
const char o1_4a[] PROGMEM = "mA";
#else
const char o1_1[] PROGMEM = "NASTAWIONA WARTOSC";
const char o1_2[] PROGMEM = "WSP. WYPEL.:";
const char o1_2a[] PROGMEM = "%";
const char o1_3[] PROGMEM = "ZMIERZONA WARTOSC";
const char o1_4[] PROGMEM = "PRADU:";
const char o1_4a[] PROGMEM = "mA";
#endif
```

9. Funkcja `window_2` powinna umożliwić wizualizację cyfr zawartych w tablicy `cyfry[3]` podczas ich modyfikacji. W przypadku braku modyfikacji cyfr powinien być wizualizowany nastawiony współczynnik wypełniania (zmienna `wsp_wypelnienia`). Niezależnie od tego czy modyfikacja jest realizowana, czy też nie, należy wizualizować zmierzony prąd płynący przez diodę mocy LED (zmienna `prad`). Poniżej przedstawiono propozycję jej definicji.

```
void window_2(signed char *pcyfry, char miganie, char menu) {
    if (...)
        print_three_digits(2,15, pcyfry, miganie);
    else
        print_integer(2,14,wsp_wypelnienia, ..., ...);
    print_integer(4,10,prad, ..., ...);
};
```

10. Zaleca się także napisanie funkcji `void print_three_digits (unsigned char x, unsigned char y, signed char *pcyfry, char miganie)`, która pozwoli wyświetlać poszczególne cyfry wraz z efektem migania aktualnie modyfikowanej. W funkcji tej zaleca się wykorzystanie informacji zawartej w zmiennej `miganie`. Brak wizualizacji cyfry za pomocą sterownika wyświetlacza można realizować, wpisując kod ASCII spacji. Zaleca się podczas edycji dowolnej z cyfr wywoływanie omawianej funkcji w funkcji `window_2`.

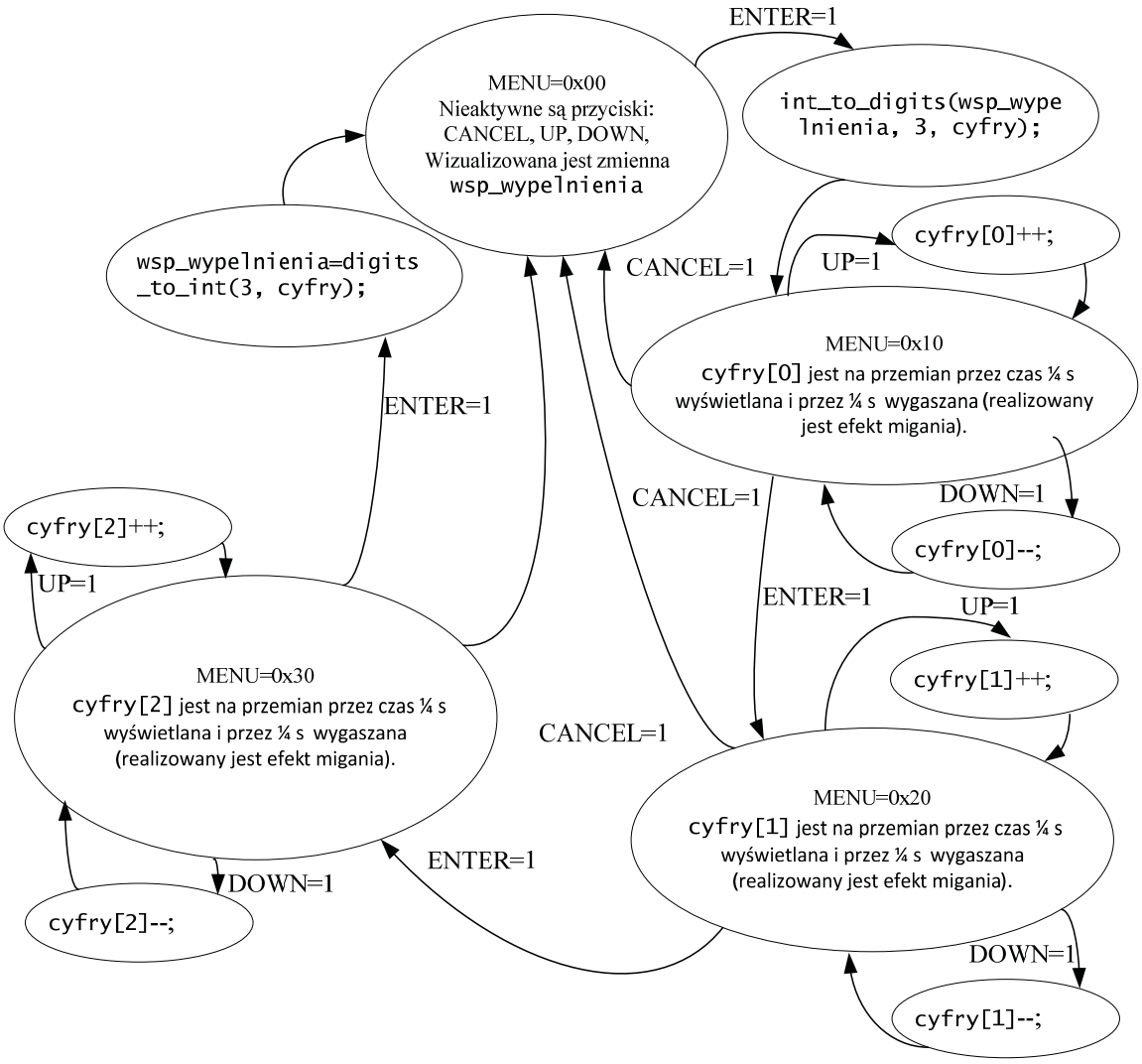
```
void print_three_digits (unsigned char x, unsigned char y,
                         signed char *pcyfry, char miganie) {
unsigned char xy;
     xy=address_DDRAM(x, y);
     send_LCDinstr((xy|0x80));    //ustalenie początku tekstu
    if((...)...(menu==...))
        send_LCDdata(' ');    //kod ASCII " "
    else
        send_LCDdata(*pcyfry+48);    //liczba setek
```

11. Zaleca się napisanie w pliku LCD.c funkcji `unsigned int digits_to_int(unsigned char liczba_cyfr, signed char *pcyfry)`, która umożliwi wyliczenie współczynnika wypełnienia dla aktualnie wprowadzonej nastawy (zawierającej trzy cyfry).
12. W funkcji inicjalizującej przetwornik analogowo-cyfrowy należy uaktywnić kanał odpowiadający pomiarowi prądu płynącego przez diodę LED, a dezaktywować kanał pomiaru napięcia zadawanego potencjometrem. Należy wykorzystać w tym przypadku informacje zawarte na schematach zamieszczonych narys. 8.1 i rys. 12.4.
13. W programie obsługi przerwania od końca konwersji przetwornika należy odpowiednio wyznaczyć wartość prądu w [mA], uwzględniając zależność (12.1) oraz informacje zawarte na schemacie zaprezentowanym na rys. 12.4.
14. Wykorzystując funkcję `multiply_with_round`, należy przeskalować zadawany w [%] współczynnik wypełnienia na wartość wpisywaną do rejestru `OCR1A` (zakres od 0 do 1023).

```
OCR1A=multiply_with_round(wsp_wypelnienia, ...);
```

15. W nieskończonej pętli programu głównego zaleca się oprogramować edycje poszczególnych cyfr zgodnie z wytycznymi zawartymi w poprzednim punkcie (rys. 12.6). Do tego celu proponuje się wykorzystanie instrukcji `switch(menu)`.

```
switch(menu){
    case 0x00 : if(kl_ENTER == 1){
                    int_to_digits(wsp_wypelnienia, 3, cyfry);
                    menu=...;
                    kl_ENTER=0;
                };
                break;
    case 0x10 : if(kl_ENTER == 1){
                    menu=...;
                    if(cyfry[0]){
                        menu=...;
                        wsp_wypelnienia=...;
                    };
                    kl_ENTER=0;
                };
                if(kl_UP == 1){
                    cyfry[0]++;
                    if (cyfry[0] > ...) cyfry[0] = ...;
                    kl_UP = 0;
                };
                if(kl_DOWN == 1){
                    cyfry[0]--;
                    if (cyfry[0] < ...) cyfry[0] = ...;
                    kl_DOWN = 0;
                };
                if(kl_CANCEL == 1){
                    menu=...;
                    kl_CANCEL = 0;
                };
                break;
```



Rys. 12.6. Proponowany sposób nastawy współczynnika wypełnienia

12.8. Przebieg ćwiczeń 13 i 14

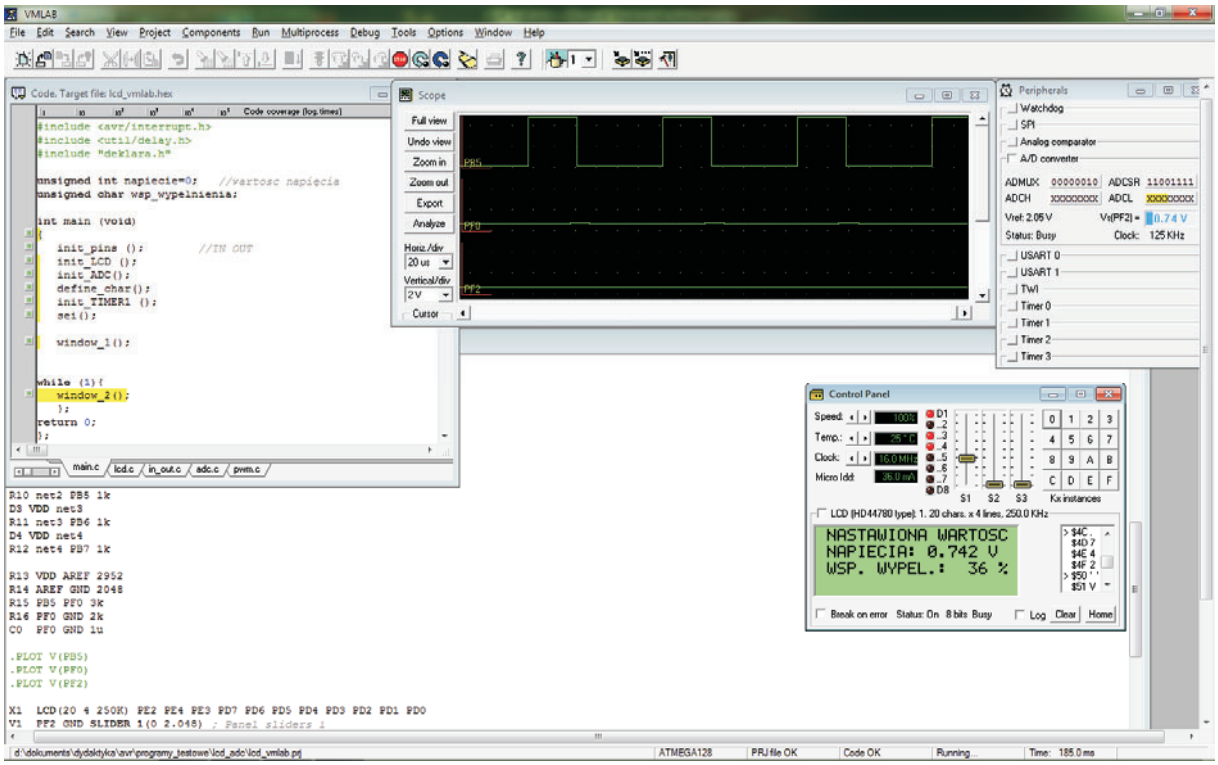
W ramach ćwiczenia laboratoryjnego należy zrealizować wytyczne zmian zawarte w poprzedniej instrukcji. Uruchomić dołączone oprogramowanie z wykorzystaniem układu dydaktycznego i z uwzględnieniem wskazówek prowadzącego. **Z uwagi na intensywność świecenia diody mocy LED, zaleca się jej przysłonięcie np. kartką przed uruchomieniem oprogramowania.** Następnie należy zmodyfikować dołączony projekt zgodnie z wytycznymi zawartymi powyżej (można wykorzystać zawarte powyżej wskazówki), a następnie uruchomić go.

12.9. Przygotowanie do zajęć numer 13

Przygotowanie do zajęć numer 13 obejmuje:

1. Zrealizowanie wszystkich wytycznych i ewentualnie wskazówek zawartych w instrukcji do poprzedniego ćwiczenia.
2. Przyniesienie wydrukowanej niniejszej instrukcji (jednej na grupę laboratoryjną 2–3 osobową),
3. Szczegółowe zapoznanie się oprogramowaniem dołączonym do instrukcji.

4. Zapoznanie się z dokumentacją dotyczącą przetwornika analogowo-cyfrowego i trybu PWM z poprawnie generowana fazą dla licznika1 [1].
5. Należy zapoznać się z zawartością pliku projektu lcd_vmlab.prj dla środowiska VMLAB (jako potencjometr został wykorzystany potencjometr S1 z panelu sterowania w środowisku VMLAB – **PF2 GND SLIDER_1(0 2.048)**) (rys. 12.7).
6. Przed uruchomieniem symulacji w środowisku VMLAB należy:
   - upewnić się, że stałą VMLAB w pliku deklara.h ustawiono na jeden,
   - otworzyć projekt – *Project -> Open project* i wybrać plik projektu: lcd_vmlab.prj
   - skompilować projekt – *Project -> Re-build all*,
   - w przypadku błędu w lokalizacji środowiska WinAVR należy odpowiednio zmodyfikować wpis w projekcie: **GCCPATH "C:\WinAVR-20100110"** i powtórzyć kompilację
   - uaktywnić zawarte poniżej narzędzia (rys. 12.7), korzystając z menu *Windows* lub *View*,
   - uruchomić symulację – skrót klawiszowy F5,
   - zmieniać wartość zadaną współczynnika modulacji potencjometrem S1 z panelu sterowania (rys. 12.7),
   - dalszą cześć symulacji można przeprowadzić, wykorzystując pułapki. Ustawiane są one po kliknięciu w jeden z zielonych kwadratów, które znajdują się po lewej stronie kodu źródłowego.
7. Przeprowadzenie symulacji dołączonego oprogramowania w środowisku VMLAB.



Rys. 12.7. Realizacja symulacji w środowisku VMLAB

8. Zastanowienie się nad modyfikacją dołączonego oprogramowania, które ma umożliwić realizację wyszczególnionych wyżej wytycznych.
9. Dla chętnych: zrealizowanie samodzielne wytycznych z ewentualnym uwzględnieniem zaleceń wyszczególnionych wyżej.

12.10. Przygotowanie do zajęć numer 14

Przygotowanie do zajęć numer 14 obejmuje:

1. Ewentualne dokończenie czynności wymienionych wyżej.
2. Zrealizowanie samodzielne wytycznych z ewentualnym uwzględnieniem zaleceń opisanych wyżej.
3. Przeprowadzenie symulacji w środowisku VMLAB projektu uwzględniającego wytyczne wymienione wyżej.

12.11. Literatura

[1] Dokumentacja mikrokontrolera ATmega128 – http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf