21 grudnia 2015

Projekt 3.2

"Symulator kombinacyjnego układu logicznego"

Autor: Jakub Jakóbczyk

Prowadzący: Krzysztof Anders

1.Opis programu

Program pozwala na zbudowanie układu logicznego, składającego się z bramek AND, OR, NOT oraz przeprowadzenie jego symulacji dla określonej kombinacji bitów wejściowych. Cały układ posiada jedno wyjście oraz nieograniczoną liczbę wejść. Program nie pozwala na dodawanie elementów w sposób powodujący sprzężenie zwrotne lub definiowanie innych niedozwolonych połączeń. W programie, do zapisu danych należało zastosować dynamiczne przydzielanie pamięci.

2.Obsługa

2.1 Obsługa programu

Program jest obsługiwany poprzez standardowe wyjście. Polecenia wprowadzane są tekstowo i zatwierdzane klawiszem *enter.* Obsługiwane polecenia to:

1)Polecenia dodające nowe elementy:

<węzeł1>=<węzeł2>and<węzeł3>

<węzeł1>=<węzeł2>or<węzeł3>

<węzeł1>=not<węzeł2>

Gdzie <węzeł1>, <węzeł2>, <węzeł3> to liczby naturalne.

2)Polecenie testowania:

test<spacja><ciag_bitów>

Gdzie <ciag_bitów> to ciąg zer i jedynek o długości liczby bitów wejściowych, poszczególne bity są przyporządkowane do bitów wejściowych według ich rosnących numerów.

3)Polecenie wyświetlania:

show

4)Polecenie wyjścia z programu:

exit

2.3 Kompilacja

Kompilację ułatwia plik makefile. Kompilacja jest możliwa jedynie w systemach Linux, zawierających kompilator gcc. Aby przeprowadzić kompilacje należy wpisać w konsoli polecenie *make*

3.Koncepcja rozwiązania

Problem podzieliłem na dwie podstawowe części:

- Pobieranie i rozpoznawanie poleceń - pozwala na wczytanie polecenia od użytkownika i rozpoznanie go.
- Operacje na danych - prowadzenie operacji na danych, takich jak zapis oraz test.

Cały program opiera się o główną pętlę, która pozwala na wczytywanie poleceń oraz ich wykonywanie aż do momentu wpisania komendy "exit"

3.1 Pobieranie i rozpoznawanie poleceń

Funkcja odpowiedzialne za wczytanie polecenia znajdują się w pliku "polecenia.c" z odpowiadojącym mu plikiem nagłówkowym "polecenia.h". Polecenie jest wczytywane od użytkownika do tablicy znaków. Maksymalna długość polecenia jest zdefiniowana w pliku "polecenia.c" przez dyrektywę preprocesora #define MAX_DLUGOSC_POLECEN Jeżeli polecenie nie jest puste, sprawdzana jest kolejno jego zgodność z poleceniami:

1) "exit" oraz "show"

2) "test" i jeśli tak sprawdza czy wprowadzony ciąg bitów odpowiedniej długości i w odpowiednim formacie. Poprawny ciąg bitów wejściowych jest zapisywany do tablicy liczb naturalnych.

3) definicja nowego elementu, jeśli tak to jakiego.

dla każdego przypadku funkcja zwraca unikalny kod każdego polecenia, lub kod błędu jeśli polecenie jest niepoprawne.

3.2 Operacje na danych

Funkcje odpowiedzialne za operacje na danych znajdują się w pliku "dane.c" z plikiem nagłówkowym "dane.h". Do zadań realizowanych przez nie należą:

1) Sprawdzanie poprawności dodawania nowych elementów oraz ich zapis do tablicy struktur zawierających dane o typie bramki, numerze węzła wyjściowego, węzłach wyjściowych oraz ich typie (pobierają dane od użytkownika albo innego elementu). Przy zapisie aktualizowane są także wejścia całego układu.

2) Testowanie - obliczanie wartości układu dla zapisanych wcześniej bitów wejściowych. Obliczenie wartości następuję rekurencyjnie, począwszy od pierwszego elementu. Po obliczeniu wartości tablica bitów wejściowych jest zwalniana z pamięci poprzez funkcję *free()*

3) Wyświetlanie wszystkich wprowadzonych dotychczas elementów oraz wejść i wyjść układu

4) Usuwanie zapisanych danych - funkcja *void usun()* zwalnia pamięć zajętą przez tablicę elementów oraz tablicę wejść poprzez funkcję *free()*, jest wywoływana raz, przy zamknięciu programu.

3.3 Sposób zapisu danych

Dane zapisywane są dynamicznie w dwóch tablicach. Pierwsza z nich jest tablicą struktur zawierająca informacje o elementach. Druga z nich zawiera informacje o wejściach. Zapis odbywa się dynamicznie, w zależności od potrzeb tablice te mogą zwiększane. Informacje o wielkościach obu tych tablic są zapisane w 2 zmiennych typu całkowitego.

4.Implementacja

4.1 Funkcja główna programu

Funkcja główna zawiera pętlę, w której najpierw wywoływana jest funkcja pobrania polecenia, która zwraca odpowiedni kod, a następnie instrukcja *switch,* wywołująca odpowiednie funkcje w zależności od polecenia

4.2 Pobieranie i rozpoznawanie poleceń

Odbywa się w funkcji *int wczytajPolecenie(int* wyjscie, int* wezel1, int* wezel2)* Rozpoznanie poleceń odbywa się poprzez szereg instrukcji warunkowych *if* sprawdzających czy zostały wywołane exit lub show oraz funkcji pomocniczych które odpowiadają za sprawdzenie zgodności z formatem pozostałych poleceń lub zwrócenie kodu błędu. Jeżeli nastąpi poprawna próba dodania nowego elementu funkcja rozpozna oznaczenia węzłów oraz zapisze je w zmiennych przekazanych przez wskaźnik.

4.3 Operacje na danych.

1)Zapis - odbywa się poprzez wywołanie funkcji void dodajBramke(int *wyjscie, int *wezel1, int *wezel2, enum typyBramek typ) Funkcja tej są przekazywane wczytane wcześniej węzły oraz typ bramki. Jeżeli typ bramki to NOT wówczas wartość *wezel2* powinna zostać ustawiona na '0' funkcja ta przede wszystkim sprawdza czy dodawana bramka jest pierwszą - wówczas traktuje to jako specjalny przypadek ponieważ musi stworzyć tablice struktur, gdzie zapisywane są elementy oraz tablicę liczb całkowitych, w której zapisywane są kody węzłów wejściowych układu. (obie tablice inicjalizowane są poprzez funkcja *calloc()*).

Zapis elementu jest realizowany przez szereg funkcji pomocniczych, których ogólny schemat to:

-Sprawdzenie czy dodanie nie wywoła sprzężenia zwrotnego - sprawdzenie to odbywa się przy pomocy rekurencyjnej funkcji *int sprawdzSprzezenie(),* która sprawdza wszystkie ścieżki prowadzące do wejść elementu, aż do napotkania wejścia układu (zwraca 0) lub wyjścia dodawanego elementu (zwraca 1)

-Sprawdzenie czy wyjście dodawanego elementu jest wejściem układu w celu uniknięcia sytuacji kiedy dwa wyjście prowadzą do jednego wejścia. W tym kroku także tablica wyjść jest aktualizowana nowe elementy są dodawane, powtarzające się są usuwane a cała tablica jest sortowana przy pomocy wbudowanej funkcji *qsort()*

-Powiększenie tablicy zapisanych elementów oraz zapisanie w niej oznaczeń węzłów wejściowych oraz wyjściowych oraz przyporządkowanie im odpowiednich typów.

2)Testowanie - funkcja *test()* odpowiada za rozpoznanie przypadku gdy nie zostały dodane żadne elementy - wówczas wyświetla wprowadzony bit jako wynik. W innym wypadku wyświetla wynik działania rekurencyjnej funkcji *obliczWynik(int wezel)*, która oblicza wynik w danym miejscu (na danym węźle) a jeżeli jest to węzeł wejściowy układu wówczas zwraca wartość wprowadzonego bitu. No koniec, tak jak zostało wspomniane w pkt 2.2 zwalnia pamięć zajmowaną przez tablicę bitów wejściowych.

3)Wyświetlanie - funkcja odpowiedzialna za wyświetlanie to *void wyswietl().* Rozpoznaje przypadek szczególny kiedy żaden element nie został dodany, wówczas wyświetla tekst informujący jedynie o jednym wejściu i wyjściu 1. W pozostałych przypadkach funkcja wyświetla najpierw wszystkie zdefiniowane elementy przy pomocy pętli *for* a następnie wszystkie wejścia. W obu przypadkach funkcja korzysta ze zmiennych informujących o rozmiarze obu tablic. Na koniec funkcja wypisuje "Wyjście: 1".

4)Usuwanie - funkcja *usun()* sprawdza czy tablicę są puste i jeśli niepuste zwalnia pamięć która jest przez nie zajmowana.