Języki programowania obiektowego

Laboratorium VI

Symulator cyfrowych układów kombinacyjnych

Prosty symulator cyfrowych układów kombinacyjnych

założenia programu

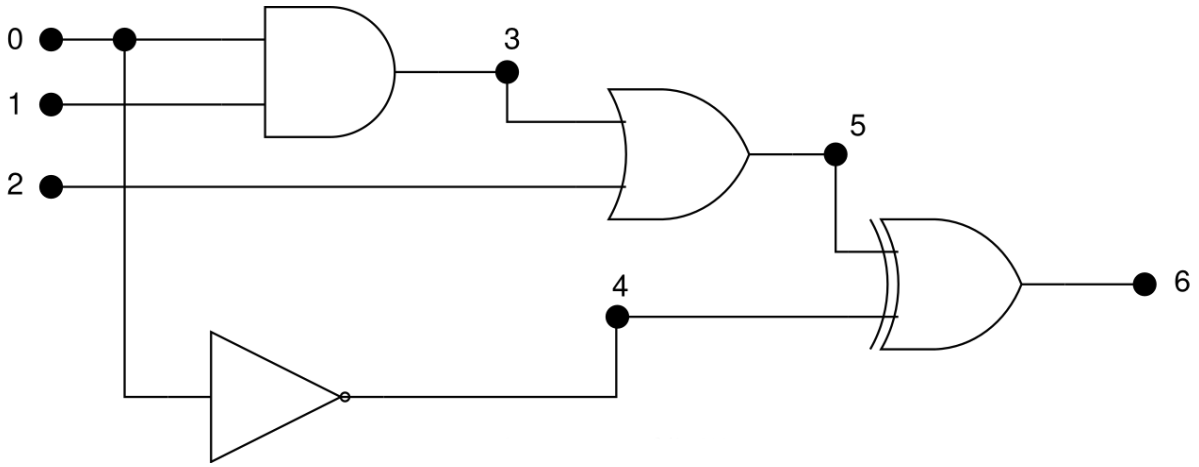
W tym ćwiczeniu stworzymy prosty symulator cyfrowych układów kombinacyjnych. Aby program nie był nadmiernie skomplikowany przyjmiemy parę założeń upraszczających. Przede wszystkim, przyjmiemy, że będziemy symulować układy cyfrowe złożone jedynie z bramek AND, OR, XOR oraz NOT. Dodatkowo przyjmiemy, że wszystkie wprowadzone przez użytkownika dane są poprawne - nie będziemy wykonywać w programie żadnej kontroli ich poprawności.

przebieg działania programu

Działanie programu będzie polegało na tym, że najpierw użytkownik zostanie poproszony o wprowadzenie konfiguracji obwodu cyfrowego a następnie wprowadza stan wejść układu. Po tym, następuje symulacja a po niej zostaje wypisany na ekranie stan wszystkich węzłów w układzie.

struktury danych w programie

Teraz zastanówmy się jakie dane musimy przechowywać w programie aby można było przeprowadzić symulację. Rozpatrzmy poniższy przykładowy schemat układu kombinacyjnego.



Aby przeprowadzić symulację potrzebne są w programie dwie struktury danych: lista zawierająca wszystkie bramki w układzie oraz lista wszystkich węzłów wraz z ich bieżącymi stanami logicznymi. Węzły będą ponumerowane liczbami całkowitymi począwszy od zera. Należy zwrócić uwagę na fakt, iż każde połączenie wyjścia bramki z wejściem kolejnej również jest węzłem (na powyższym schemacie są to węzły 3, 4 oraz 5). Z kolei, każda bramka będzie przechowywać indeksy swoich węzłów wejściowych oraz indeks węzła wyjściowego. Oprócz tego każda bramka musi mieć dostęp do listy wszystkich węzłów w układzie aby mieć możliwość wykonania odpowiedniej operacji logicznej.

Podsumowując będziemy mieć w programie następujące dwie struktury danych:

1. Lista bramek. W naszym programie przyjmiemy, że będzie to wektor wskaźników do obiektów reprezentujących konkretną bramkę.
2. Lista wszystkich węzłów. W naszym programie przyjmiemy, że będzie to mapa w której kluczem będzie indeks węzła, zaś wartością będzie jego bieżący stan logiczny. Przyjmiemy, że każdy węzeł może przyjąć jeden z trzech możliwych stanów: 0, 1 lub -1. Wartość -1 oznacza, iż bieżący stan węzła jest nieustalony. Wprowadzenie dodatkowego stanu niesutalonego ułatwi implementację algorytmu symulacji.

implementacja symulatora

Program będzie wymagał standardu języka C++ w wersji C++17 lub nowszej. W Visual Studio wybierz następujące menu: Project->Properties->Configuration Properties->General->C++ Language Standard i wybierz standard C++17 lub nowszy.

Pisanie symulatora rozpoczniemy od stworzenia odpowiedniej hierarchii klas reprezentującej bramki w programie.

Najpierw tworzymy interfejs w którym umieścimy funkcje wspólne dla wszystkich bramek. W naszym przypadku będzie to metoda, która wyznacza stan wyjścia bramki na podstawie jej wejść oraz metoda wypisująca stan bramki na ekranie.

1. Stwórz interfejs o nazwie `IGate`. Utwórz w nim metodę czysto wirtualną `int evaluate()`. Klasy reprezentujące konkretne bramki i implementujące interfejs `IGate` będą w metodzie `evaluate()` wykonywać operacje logiczne odpowiednie do swojego typu. Stwórz również metodę czytowirtualną `void print() const`. Nie zapomnij dodać do interfejsu `IGate` domyślny wirtualny destruktor.

Następnie tworzymy hierarchię klas, które reprezentują poszczególne typy bramek w układzie.

2. Stwórz klasę `Gate`, która reprezentuje bramkę dowolnego typu. Klasa `Gate` dziedziczy z interfejsu `IGate`. Ponieważ każda bramka musi wykonać operację logiczną na stanach logicznych swoich węzłów

wejściowych dlatego w sekcji `private` klasy `Gate` umieść referencję[1] do obiektu typu `map<int, int>` reprezentującego wszystkie węzły w układzie. Utwórz w klasie `Gate` konstruktor `Gate(map<int, int>& nodes)`, który zainicjuje tą referencję. Utwórz getter zwracający referencję do mapy węzłów. Utwórz również domyślny wirtualny destruktor.

3. Teraz możemy już przejść do stworzenia klas reprezentujących konkretne bramki. Stwórz klasę o nazwie `AndGate` reprezentującą bramkę AND. Klasa `AndGate` dziedziczy z klasy `Gate`. W klasie `AndGate` utwórz pola reprezentujące indeksy wejść bramki oraz indeks wyjścia. Utwórz konstruktor umożliwiający zainicjowanie indeksów wejść i wyjścia oraz referencję przechowującą wszystkie węzły w obwodzie. Następnie nadpisz metodę `evaluate()` odziedziczoną z interfejsu `IGate`. Ponieważ stan logiczny wyjścia bramki można wyznaczyć jedynie w przypadku gdy stan wszystkich wejść jest ustalony (czyli gdy wynosi 0 lub 1) dlatego metoda `evaluate()` powinna zwrócić wartość -1 gdy stan przynajmniej jednego z wejść nie jest obecnie ustalony (czyli gdy wynosi -1). W tym przypadku również nie zmieniamy stanu węzła do którego podłączone jest wyjście bramki. W przypadku gdy stan wszystkich wejść bramki jest ustalony, metoda `evaluate()` wykonuje swoje działanie logiczne na stanach węzłów wejściowych a następnie rezultat przypisuje węzłowi wyjściowemu. Na końcu rezultat ten jest zwracany przez metodę `evaluate()`. Nadpisz metodę `print()` odziedziczoną z interfejsu `IGate`. Metoda `print()` ma wypisać na ekranie napis: *AND indeks_pierwszego_wejścia indeks_drugiego_wejścia indeks_wyjścia* z bramki.

4. Podobnie do klasy `AndGate` utwórz klasy `OrGate`, `XorGate` oraz `NotGate`.

5. Utwórz funkcję `void print_nodes( const map<int, int>& nodes )`, która wypisuje na ekranie stan wszystkich węzłów w obwodzie.

6. W funkcji `main()` utwórz wektor wskaźników `vector< unique_ptr<Gate> > gates`. Wektor będzie przechowywał wszystkie bramki w układzie. Utwórz również mapę `map<int, int> nodes`, która przechowuje stany wszystkich węzłów w obwodzie. Klucz w mapie to indeks węzła, zaś wartość to jego bieżący stan logiczny. Następnie napisz kod realizujący poniższy algorytm wprowadzania danych do programu.

```
w pętli powtarzaj:

  wprowadź typ bramki

  jeśli typ bramki == "end" to wyjdź z pętli

  jeśli typ bramki == "and" lub "or" lub "xor" to wprowadź indeksy dwóch
wejść bramki

  jeśli typ bramki == "not" to wprowadź indeks wejścia bramki

  wprowadź indeks wyjścia bramki

  dodaj bramkę do listy bramek

  przypisz węzłom wejściowym i wyjściowemu wartość -1
```

Dodanie bramki do listy bramek zrealizuj za pomocą metody `emplace_back()` z kontenera `vector`. Wskaźnik do bramki utwórz za pomocą funkcji `make_unique()`, np. dla bramki AND może mieć to następująca postać: `make_unique<AndGate>( input1, input2, output, nodes )`.

7. Następnie konieczne jest wprowadzenie stanu logicznego wszystkich wejść obwodu.

```
w pętli powtarzaj:

```

[1] Referencja ta nie może być stała ponieważ, każda bramka musi mieć możliwość modyfikowania stanu węzłów.

```
wprowadź indeks węzła

jeśli indeks węzła == -1 wyjdź z pętli

wprowadź stan węzła

przypisz wprowadzony stan odpowiedniemu węzłowi
```