Przerzutniki

Prowadzący: Waldemar Grabski

Projekt wykonała: Aleksandra Majewska

Nr indeksu: 310832

- Wyjściowy problem

Zrealizować program pozwalający użytkownikowi na symulację rejestru zbudowanego z dowolnej liczby przerzutników połączonych ze sobą.

Wejściem do przerzutnika może być dowolnie wiele wyjść innych przerzutników, poddanych wybranej operacji logicznej (XOR/AND/NOR etc).

Wyjście N-tego przerzutnika jest zawsze jednym z wejść przerzutnika N+1. Wyjście ostatniego jest zawsze wejściem (jednym z) pierwszego.

W ten sposób powstaje generator ciągów bitów.

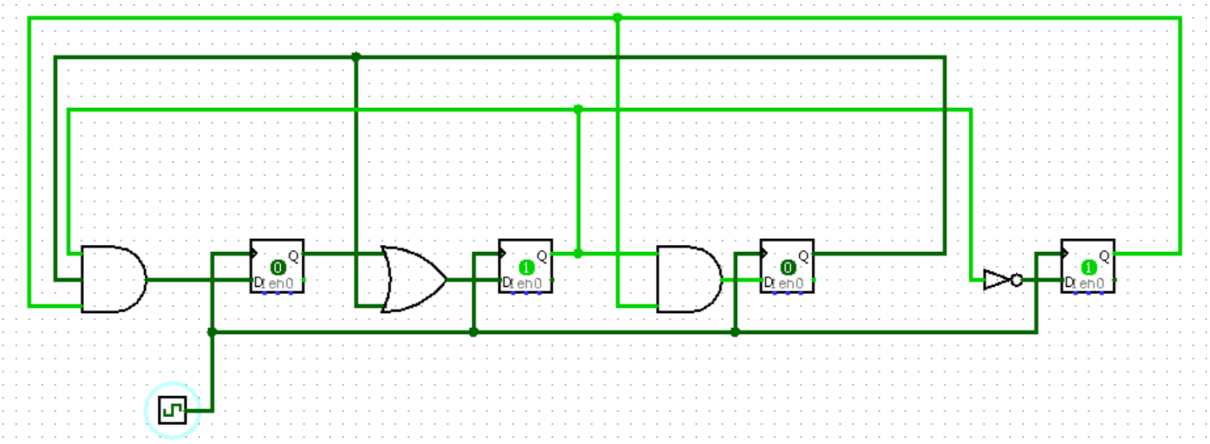
Wymagania

- Program powinien wyczytywać połączenia między przerzutnikami z pliku.
- Program powinien wygenerować ciągi dla zadanej wartości początkowej i określonej liczby kroków.
- Program powinien móc zapisywać wygenerowane ciągi.
- Program powinien mieć tryb pracy "aż się zapętli", czyli aż wróci do stanu początkowego.
- Uzyskane ciągi powinny być oceniane:
- stopień wykorzystania przestrzeni (szereg N przerzutników hipotetycznie daje 2^N kombinacji a ile zostało wygenerowanych)
- różnorodność ciągu - średnia liczba bitów różniących się pomiędzy wyrazami ciągu.

Ze względu na złożoność kombinatoryczną należy w testach podchodzić ostrożnie do liczby przerzutników (wolne obliczenia).

- Analiza problemu

Aby zobrazować, działanie programu, stworzyłam symulację w programie logisim.



Na początku doszłam do wniosku, że mój program powinien używać tylko przerzutników typu D, ponieważ taka wersja wydawała mi się najprostsza i praktyczna.

- Struktura projektu

W obecnej wersji program zawiera klasy:

Flip_Flop – klasa reprezentuje pojedynczy przerzutnik;

Gate – „klasa matka” z której dziedziczą klasy:

AND, OR, XOR, XNOR, NOR, NAND, NOT – posiadają metodę, która wykona daną operację logiczną na podanych wartościach i zwróci wynik

Block – klasa reprezentująca fragment układu składający się z bramki i przerzutnika; posiada metodę, która dla danych wartości i korzystając z odpowiedniej bramki, zmieni stan przerzutnika

Circut – klasa posiada listę bloków oraz informacje o tym, jak przerzutniki są połączone;

Simulation – klasa sterująca działaniem całego programu;

Oprócz tego istnieją również klasy Interface, Reader i Writer do komunikacji z użytkownikiem, odczytywania i zapisywania danych do pliku.

W pliku errors.py znajdują się błędy.

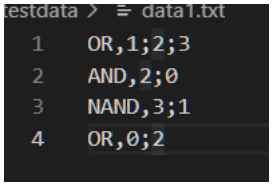
Pliki zaczynające się od test_ zawierają testy jednostkowe.

- Działanie programu

Aby przerwać program w dowolnym momencie, należy wpisać „exit”. Użytkownik jest poinformowany o tej opcji na początku. W przypadku wprowadzenia niepoprawnych danych program będzie informował, na czym polegał błąd i żądał wprowadzenia poprawnych wartości „do skutku”.

Jako pierwszą należy wprowadzić **ścieżkę do pliku**, w którym znajdują się dane obwodu.

Przykładowy plik z danymi wygląda tak:



Przerzutników jest tyle, co linijek.

W pierwszej kolumnie każdego wersu znajduje się nazwa bramki, której wyjście jest doprowadzone na wejście danego przerzutnika. Natomiast w drugiej indeksy przerzutników, których wyjścia są doprowadzone do wejść tej bramki.

Indeksy powinny być numerowane od 0 do (liczba przerzutników) – 1.

Możliwe bramki logiczne to 'AND', 'OR', 'XOR', 'XNOR', 'NOR', 'NAND', 'NOT'.

Bramki XOR i XNOR powinny mieć dwa wejścia, jednak w przypadku innej liczby wejść użytkownik dostanie jedynie ostrzeżenie. Nie przerwie to działania całego programu, a bramki zadziałają następująco:

XOR – wyprodukuje stan wysoki, jeżeli na wejściu pojawi się nieparzysta liczba stanów wysokich

XNOR – wyprodukuje stan wysoki, jeżeli na wejściu pojawi się parzysta liczba stanów wysokich

W przypadku pliku z niepoprawnymi danymi użytkownik zostanie poinformowany, gdzie wystąpił błąd i wróci do etapu wpisywania ścieżki do pliku.

Jeżeli plik zawiera poprawne dane, użytkownik zostanie poproszony o wprowadzenie **stanu początkowego**, używając 1 – jako stan wysoki, 0 – jako stan niski.

Następnie program zapyta użytkownika czy ma działać **aż się zapętli**. Użytkownik odpowiada:

1 – tak lub 2 – nie

Jeżeli **nie**, program poprosi o wprowadzenie **liczby kroków czasowych** symulacji.

Jeżeli **tak**, program przejdzie od razu do zapytania użytkownika, jaką nazwę ma mieć **plik wyjściowy**.

W repozytorium znajduje się również folder „testdata" z danymi do testowania.

Zostanie wygenerowany plik wyjściowy o podanej przez użytkownika nazwie i takiej budowie:

```
1100
1011
1111
1101
1001
Generated 5 combinations out of 16 possible combinations.
States differ on average by 1.6 bits.
```

Najpierw zostają wymienione wygenerowanie ciągi, a następnie komentarze co stopnia wykorzystania przestrzeni i różnorodności ciągów.

- Podsumowanie

Realizując projekt dużo się nauczyłam o programowaniu obiektowym. Na początku moje klasy były nieodpowiednio połączone i nie odzwierciedlały dobrze obiektów, które miały reprezentować. Przyczyną obrania nieodpowiedniego kierunku działań był brak konsultacji przed przystąpieniem do pracy.

Po otrzymaniu rad i wskazówek od prowadzącego wprowadziłam niezbędne zmiany, znacznie zmieniając strukturę projektu. W efekcie mój program przybrał mniej skomplikowaną formę i był bardziej „odporny" na błędy.

W przyszłości planuję wprowadzić możliwość używania różnych typów przerzutników, ponieważ wydaje mi się to teraz mniej skomplikowane niż na początku. Myślę również, że warto wyposażyć program w GUI, aby był bardziej przyjazny użytkownikowi, jednak na tym etapie nie uznaję tego za kluczową kwestię.

Zdecydowanie największą trudnością przy tworzeniu projektu był brak organizacji zadań, który objawiał się tym, że zaczynałam wprowadzać nowe serie poprawek przed dokończeniem starych. Ostatecznie decydowałam się na tworzenie „issues" na gitlabie i dzięki temu praca z kodem szła szybciej i wydajniej.