МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет комп'ютерних наук, фізики і математики
Кафедра інформатики, програмної інженерії та економічної кібернетики

АНАЛІЗ ТА ПОБУДОВА ІНСЕРЦІЙНОЇ МОДЕЛІ АПАРАТУРИ ІНТЕГРАЛЬНОЇ СХЕМИ

Кваліфікаційна робота (проєкт)
на здобуття ступеня вищої освіти «магістр»

Виконав: студент 2 курсу
Спеціальності 121 Інженерія програмного забезпечення
Освітньо-професійної програми «Інженерія програмного забезпечення» другого (магістерського) рівня вищої освіти
Болгарін Тимофій Олександрович
Керівник доктор фізико-математичних наук, професор
Песчаненко Володимир Сергійович
Рецензент кандидат фізико-математичних наук, професор Кузьмич Валерій Іванович

Херсон – 2020

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

IMS	Insertion Modeling System
VHDL	VHSIC (Very high speed integrated circuits) Hardware Description Language
APLAN	Algebraic Programming LANguage
ANTLR	ANother Tool for Language Recognition

ВСТУП

Актуальність дослідження.

Сьогодні майже всі системи обчислення і контролю базуються на цифрових логічних схемах. Це можуть бути не тільки прості мікросхеми з декількома логічними операціями, а й складні центральні процесори або програмовані логічні інтегральні схеми. Для проєктування складних цифрових логічних схем використовуються спеціальні мови формального опису роботи цих схем. І перед тим як втілювати їх у фізичну форму потрібно перевірити логіку роботи моделі згідно з вимогами, перевірити правильність логіки. Для підвищення надійності таким систем потрібно не лише проводити тестування систем за тестовими наборами, а й формально перевірити та довести коректність роботи системи [1,2,3] (верифікувати систему). В якості вихідної мови таких систем було обрано мову формального опису цифрових логічних схем VHDL [4,5], а систему для формальної верифікації — систему інсерційного моделювання IMS [6,7] яка була створена в Інституті кібернетики ім. В.М. Глушкова.

Інсерційне моделювання — напрям який представляє собою підхід до побудови загальної теорії взаємодії агентів та середовищ в складних розподілених багатоагентних середовищах. Останнім часом цей напрям розвивається як підхід до верифікації розподілених програм, апаратури, вимог до програмного забезпечення у тому числі й для VHDL.

Мова VHDL [8,9,10] — одна з базових мов для формального опису цифрових та змішаних логічних схем. Це один з підвидів HDL мов. Як і інші HDL мови вона створювалася, спочатку для формального опису та документації апаратних рішень. Це дозволило спростити процес проєктування, піднявшись на більш високий рівень абстракції. До того ж це дозволило, наприклад, проводити комп’ютерне моделювання цифрових схем для перевірки їх властивостей до того як їх перенесуть у реальні схеми. Для цих цілей були розроблені логічні симулятори, які

зчитують код VHDL та прогнозують поведінку цифрових схем, описаних цією мовою. Пізніше були розроблені інструменти синтезу [11,12], які компілювали код VHDL у список зв'язків вентилів і транзисторів, тобто давали на виході вже фізичну, а не абстрактну реалізацію цифровою схеми. Це дозволило ще більше спростити процес проєктування схем [13]. Зараз мову VHDL дуже часто застосовують для опису проєктів на основі яких будуть створена апаратура інтегральної схеми, або використовується для синтезу та конфігурації програмованих логічних інтегральних схем [14,15,16].

Забезпечення правильності роботи цифрових схем є надзвичайно важливим і складним завданням. Як свідчить практика, помилки у цифрових схемах масового користування або цифрових схемах призначених для приладів від яких залежать життя людей коштують дуже дорого. Як приклад, навіть у процесорах деяких великих компаній часто виявляють не тільки незначні помилки, а й критичні, які загрожують безпеці користувачів. Найбільш популярна для перевірки цифрових схем техніка — моделювання. Однак моделювання за своєю природою є неповним, і в багатьох випадках воно може допустити критичні помилки. Альтернативним методом до перевірки є використання формальних методів, які дозволяють математично довести, що реалізація цієї цифрової схеми задовольняє специфікації і немає шляхів до некоректної її роботи.

Мета дослідження – створення інсерційної семантики мови VHDL

Об'єкт дослідження – мова VHDL.

Предмет дослідження – інсерційне моделювання мови VHDL.

Методи дослідження.

Було проведено аналіз літератури, семантики мов VHDL та APLAN для синтезу інсерційної семантики мови VHDL.

Наукова новизна одержаних результатів.

Вперше описано інсерційну семантику мови VHDL, яку можна використовувати для створення інсерційних моделей з проєктів VHDL з

метою їх подальшої верифікації.

Практичне значення одержаних результатів.

Отримані в результаті роботи семантику та транслятор можна використовувати для автоматичного та напівавтоматичного створення інсерційних моделей, які потім можна аналізувати в системі інсерційного моделювання.

Завдання дослідження:

1. Визначити мету та методи для формальної верифікації
2. Проаналізувати синтаксис і семантику мови опису алгебри поведінки для системи IMS
3. Проаналізувати синтаксис і семантику мови опису апаратури інтегральних схем VHDL
4. Побудувати таблицю перетворення синтаксичних та семантичних структур мови VHDL у інсерційну модель
5. Розробити транслятор моделі VHDL у інсерційну модель

Апробація результатів.

Готується публікація в збірник “IT в освіті”. Опубліковано статтю в збірнику наукових праць “Магістерські студії”. Вихідний код реалізованого транслятора інтегровано з системою інсерційного моделювання IMS.

Структура роботи.

Робота складається зі вступу, трьох розділів, висновку та списку використаних джерел.

РОЗДІЛ 1
МЕТОДИ ФОРМАЛЬНОЇ ВЕРИФІКАЦІЇ МОДЕЛЕЙ ТА МОВА VHDL

1.1 Методи формальної верифікації моделей

Верифікація призначена для перевірки відповідності системи набору проєктних специфікацій. На етапі розробки процедури верифікації передбачають проведення тесту для моделювання або імітації частини або цілісної системи, а потім проведення аналізу результатів моделювання. На кожному етапі розробки передбачається регулярне повторення випробувань, розроблених спеціально для того, щоб впевнитись, що система відповідає технічним вимогам.

Формальна верифікація у контексті апаратних та програмних систем — це акт доведення або спростування правильності роботи алгоритмів, що лежать в основі системи щодо певної формальної специфікації або властивості, за допомогою формальних методів математики [17,18].

Формальна перевірка може встановити відповідність системи певним вибраним властивостям із 100% достовірністю. Вона може бути корисною у доведенні правильності таких систем як:

- криптографічні протоколи,
- комбінаційні схеми,
- цифрові схеми,
- програмне забезпечення.

Наприклад, буде корисним перевірити систему управління залізницею та впевнитись, що зелене світло буде ввімкнено тільки при певних умовах. На відміну від тестування, яке для великих систем навряд чи досягне повного охоплення комбінацій, формальна перевірка зможе забезпечити повне покриття системи, тому що вона використовує математичний апарат для цього.

Перевірка цих систем здійснюється шляхом надання формальних доказів на абстрактній математичній моделі системи, відповідності між математичною моделлю системи та очікуваної поведінки системи. Для моделювання систем використовують такі математичні об’єкти, як:

- машина кінцевих станів,
- розмічені транзитивні системи,
- мережі Петрі,
- алгебра процесів,
- логіка Хоара, тощо.

Доведення теореми — це формальний метод перевірки, який використовує математику та логіку: пост-умова (висновок) базується на передумові (гіпотеза). Використовуючи аксіоми та правила умовиводу, цей метод передбачає, якщо гіпотеза відповідає дійсності, то висновок також повинен бути правдивим.

На відміну від доведення теорем, перевірка моделі відвідує простір станів програми, щоб здійснити перевірку властивостей програми, сформульовану у темпоральній логіці. Перевірка моделі досліджує стан системи і має на меті довести, що заздалегідь визначена програмна модель, яка може бути або набором вимог, або імітаційною моделлю, відповідає специфікаційним властивостям.

Разом з тим, задача перевірки деяких складних систем може бути нерозв’язною. Більшість алгоритмів перевірки моделі дозволяють верифікувати тільки моделі зі скінченною кількістю станів. Однак існує велика кількість систем, для опису яких необхідно використовувати нескінченні моделі з необмеженою кількістю станів. Моделі розподілених систем такого виду складаються з однотипних взаємодіючих процесів, де кількість процесів — параметр, який може бути дуже великим, через це звичайні алгоритми розв’язування задачі перевірки моделей для кінцевих моделей програм не може гарантувати коректної перевірки параметризованих моделей програм.

1.2 Інсерційне моделювання

Інсерційне моделювання займається побудовою і вивченням агентів і середовищ. Теорія взаємодії агентів і середовищ спирається на алгебру процесів для уніфікації різних моделей взаємодій та обчислень. Функціонування агентів і середовищ забезпечується математичним апаратом, яким є поняття транзитивних систем. Даний апарат дозволяє виявити поведінку системи та еквівалентність декількох систем. Елементами транзитивних систем у загальному випадку можуть бути компоненти, програми, об’єкти, які взаємодіють один і іншим та з середовищем.

Вся модель представляє собою ієрархію середовищ і агентів, які занурені у ці середовища. Агенти і середовища еволюціонують з часом та мають певну поведінку. Занурення ж агента у середовище викликає зміну поведінки середовища і породжує нове середовище, в яке знову можуть зануритися відповідні агенти. Середовища можуть бути агентами і, відповідно, можуть занурюватися у середовища верхнього рівня.

Еволюція системи описується за допомогою історії її функціонування, яка може бути як скінченною, так і нескінченною. Історія функціонування системи включає в себе:

1. успішне завершення розрахунків в середовищі транзитивної системи,
2. тупикові стани, коли кожна з паралельних частин системи знаходиться в стані очікування події,
3. невизначені стани, які виникають, наприклад, у випадку з нескінченними циклами.

Опис поведінки представляє потік управління цієї програми. У загальному випадку поведінка може містити такі операції:

- операцію префіксінгу a.u,
- недетермінованого вибору u+v,

- послідовної композиції (u;v),
- паралельної композиції (u||v).

Тут a — дія, a u та v — поведінки.

Також поведінка може містити термінальні константи:

- Δ — успішне завершення,
- 0 — тупикова поведінка,
- $\perp$ — невідома поведінка.

Також опис поведінки допускає використання операції протиставлення дії, наприклад, *A* та *!A*. Цей принцип, наприклад, дозволяє зручно описувати поведінку структур виду “if else”, для якої поведінка буде мати вигляд (a.IF + !a.ELSE).

Базовим поняттям є “Дія”, яка трансформує стан агентів, поведінка яких, зрештою, змінюється. Для представлення дій у інсерційних моделей розподілених систем застосовуються базові протоколи [19,20]. На основі методу базових протоколів вже були розроблені методи верифікації вимог та специфікації розподілених систем в області телекомунікації, вбудованих систем та систем реального часу. Базовий протокол представляє собою вираз виду $\forall x(a \rightarrow < u > b)$, де:

- x — список параметрів,
- a — формула передумови,
- b — формула постумови,
- u — процес протоколу.

Базовий протокол може розглядатися як формула темпоральної логіки, яка виражає той факт що, якщо система задовільняє умову a, то процес u може бути ініційований та після його завершення розмітка буде задовольняти умову b.

Специфікація системи відбувається шляхом опису середовища та базових протоколів. Ця специфікація може бути доволі абстрактною та мати багато різних реалізацій. У разі конкретної інтерпретації стану системи вони можуть бути виражені у вигляді конкретних атрибутів, а

кожен базовий протокол будується таким чином щоб з стану що задовольняє передумову перейти у стан який задовольняє пост-умову. Для цього допускається використовувати присвоєння у пост-умові.

У разі абстрактної інтерпретації стан системи представляється формулами базової мови, де кожна формула представляє різні конкретні стани. Для однозначного встановлення переходів для символьних станів, які представлені формулами, до специфікації додається предикатний трансформер — функція, яка встановлює перехід від формули у передумові до формули пост-умови.

Для опису інсерційної моделі будемо використовувати мову алгебраїчного програмування APLAN (Algebraic Programming LANguage) [21,21].

Алгебраїчне програмування — програмування, яке засновано на правилах перепису [22]. Наразі мова APLAN розглядається її авторами як базова мова для системи інсерційного моделювання, роботи над якою активно ведуться в Інституті кібернетики В.М. Глушкова НАН України та Херсонському державному університеті МОН України.

Структура моделі на APLAN включає в себе:

- опис оточення,
- сигналів,
- дій на основі базових протоколів,
- поведінки системи.

У оточенні описуються:

- типи агентів,
- агенти,
- атрибути моделі,
- початкові умови.

Наступний приклад оточення описує тип агента SEnv, агента senv типа SEnv, атрибут для зберігання поточного часу a_time та початкову умову, що a_time має дорівнювати нулю:

```
environment(
   agent_types:obj(
      SEnv:obj(Nil));
   agents:obj(
      SEnv:(senv);
   attributes:(
      a_time:integer
   );
   inital_formula:obj(
      a_time==0
   );
);
```

Опис сигналів — це опис тих самих процесів протоколів. Приклад їх опису, де створюється сигнал sbool в одним параметром типу bool (boolean):

```
events(
   obj(
      sbool:obj(bool)
   );
);
```

Опис базових протоколів відбувається у блоку “Дій”. Наступний приклад описує дію з назвою “act_sbool”, де (a>b) — передумова, (PROC#a:out sbool(c) to SEnv#senv) — відправка сигналу “sbool” з параметром “c” від агента “p” типу “PROC” до агента “senv” типу “SEnv”, а (a == b) — постумова:

```
act_sbool = (
  (a>b)->
  ("PROC#p:out sbool(c) to SEnv#senv;")
  (a == b)
);
```

Останнім етапом є опис поведінки системи. Наступний приклад опису поведінки описується за допомогою таких операцій поведінки як префіксінг, недетермінованого вибору, послідовної та паралельної композиції, а діями виступають базові протоколи та інші поведінки:

```
SYS = INIT.BASE,
INIT = (A+!A)||(C+D),
BASE = ...,
```

1.3 Мова опису апаратури інтегральної схеми VHDL

Будова проєкту

На відміну від звичних нам мов програмування, таких як C,

assembler та інших, VHDL описує паралельні процеси. Вся програма описується як набір блоків, операції в яких розглядаються для одночасного виконання. До того ж створений один раз блок для розрахунку може бути використаний та налаштований під потреби у різних проєктах.

Структура проєкту VHDL складається з блоків, які з'єднуються між собою дротами. Кожен блок описується за допомогою інтерфейса ENTITY та архітектури ARCHITECTURE. ENTITY описує, які вхідні та вихідні сигнали має блок, а також різноманітні атрибути, які дозволяють конфігурувати блок при побудові кінцевої схеми. ARCHITECTURE описує паралельні процеси PROCESS та паралельні операції для опису паралельної поведінки блоку. Незважаючи на те, що мова VHDL має декілька видів структур, які описують паралельні операції, всі ці структури можна виразити як елементарні процеси. Кожен процес містить в собі послідовні оператори, тобто ті, які виконуються не паралельно, а послідовно.

Вся архітектура системи на VHDL будується на основі передачі сигналів. Сигнал — первинний об'єкт який описує апаратну систему та еквівалентний фізичним провідникам струму. Це канал зв'язку між різними паралельними операторами специфікації системи. Використовується для моделювання апаратних властивостей, таких як:

- паралелізм,
- інерціальність,
- декілька драйверів для одного сигналу та ін.

Елементарний сигнал має тип bit, який може мати два значення, але більш складні системи потребують складних типів. Це можуть бути такі типи, які використовуються для передачі:

- результатів логічних виразів (boolean),
- чисел (integer, real),
- символів (character),

- різноманітних масивів (bit_vector).

Але також існують деякі інші типи, які дозволяють більш адекватно описувати цифрові сигнали, одним з найпопулярніших є std_logic. Це спеціальний тип, який дозволяє більш повно описати сигнальний дріт. Він може приймати наступні значення:

- ‘U’ — не ініціалізовано
- ‘X’ — невідоме значення, яку формується активним виходом
- ‘0’ — логічний нуль
- ‘1’ — логічна одиниця
- ‘Z’ — “третій” стан, стан високого імпедансу
- ‘W’ — “слабке невідоме” значення
- ‘L’ — “слабкий нуль”
- ‘H’ — “слабка одиниця”
- ‘-’ — довільне значення

Такий список можливих значень враховує не тільки особливості апаратної реалізації цифрових приладів, але і особливості моделювання. Наприклад, стан “невідоме значення” означає, що система моделювання не може визначити точний логічний рівень, який має бути в даній частині системи, однак у реальному приладі цей сигнал буде однозначно визначений як “0” або “1”.

Драйвер — один з паралельних операторів який задає значення сигналу у певний момент часу. Якщо сигнал має декілька драйверів у певний момент часу, то результат може бути непередбачуваним. Для випадків, коли проєкт потрібно синтезувати, ситуації з декількома драйверами сигналів неприпустимі. Для інших випадків це допустимо, а кінцевий результат визначається за допомогою функцій розрішення, які аналізують вся значення з драйверів та визначають значення сигналу за певним алгоритмом.

Атрибути — особлива довготривала властивість предмета. У мові VHDL сигнали, змінні та інші об’єкти, окрім свого значення мають багато

атрибутів. Кожен тип об’єкта має свої передвстановлені атрибути. Користувач також може встановити ряд своїх спеціальних атрибутів. Атрибути бувають різних типів:

- тип,
- значення,
- сигнал,
- функція,
- діапазон.

Атрибути сигналу — параметри, які зберігають здебільшого інформацію про історію змін значень сигналу, його стан та інформацію для тригерів.

Для синхронізації системи використовуються оператори очікування (wait statements) та події, прив’язані до часу. Оператори очікування дозволяють симуляторам не прораховувати паралельні процеси, якщо вони цього не потребують. Наприклад, немає сенсу прораховувати значення вихідного сигналу, якщо жоден з вхідних сигналів не змінив свого значення. До подій, прив’язаних до часу, можна віднести не тільки оператори очікування, які очікують певного часу, а тільки потім продовжують виконання процесу, але і відкладені присвоєння сигналів. Відкладені присвоєння сигналів можуть бути заплановану в одному циклі симуляції, а виконані в іншому, коли пройде достатньо часу моделювання. Таким чином можуть бути змодельовані штучні затримки у передачу даних або зовнішні генератори сигналів, які будуть активувати процес кожні n одиниць часу.

Усю мову VHDL можна поділити на декілька частин: яку можна синтезувати та яку можна тільки моделювати. Не всю мову можна синтезувати, частина конструкцій, які не піддаються синтезу, може або викликати помилки під час синтезу, або просто ігноруватися інструментами синтезу, що може призвести до несподіваної поведінки системи. Тому проєкти, які будуть синтезуватися, мають бути

спроєктовані згідно певних правил, виключаючи деякі недоступні для синтезу конструкції мови [23,12].

Опис простого блоку для логічного оператора AND має наступний вигляд:

```
ENTITY binary_operator IS
  port(
            in_1:in bit;
            in_2:in bit;
       out_1:out bit;
  );
END ENTITY binary_operator;

ARCHITECTURE and_operator OF binary_operator IS
BEGIN
          PROCESS IS
                        BEGIN
                  out_1 <= in_1 and in_2;
                  WAIT ON in_1, in_2;
          END PROCESS;
END ARCHITECTURE and_operator;
```

Наведений вище блок має такий інтерфейс: два бітових входи та один бітовий вихід. Архітектура моделює логічний бінарний оператор AND. Процес в архітектурі буде виконуватися кожного раз, коли буде змінюватися один з входів цього блоку, таким чином кожного раз буде переобраховуватися значення вихідного сигналу, і, можливо, цим викликати активацію інших блоків.

Моделювання та синтез

Моделювання цифрових систем є важливим кроком під час їх розробки. Постійне підвищення складності проєктів змушує розробників витрачати багато часу на їх моделювання. Моделювання дозволяє як дослідити алгоритм роботи систем, так і верифікувати її характеристики. Під час моделювання використовується підхід, заснований на так званому стенді для випробування (testbench). Прилад, який моделюється представляє собою модель, яку можна синтезувати, а для перевірки її поведінки за різних умовах створюються тестові ситуації за допомогою структур, які може бути неможливо синтезувати, тільки промоделювати.

Ці структури дозволяють імітувати складні процеси, які неможливо реалізувати за допомогою цифрових схем. Список структур, які неможливо синтезувати, а також правила до опису VHDL моделей для синтезу, можуть відрізнятися для різних синтезаторів та чіпів.

Існуючі інструменти для формальної верифікації VHDL

Для автоматизації методів формальної верифікації потрібно використання доволі складних систем розробки, а також висококваліфікованого персоналу. Тому використання спеціалізованого програмного забезпечення, орієнтованого на формальну верифікацію, може дозволити собі не кожна організація. Розробкою і випуском інструментів для розробки, які включають в себе інструменти для формальної верифікації, сьогодні змогли реалізувати тільки такі великі фірми, як Synopsys та Cadence. Нормальне включення цих методів в розробку можуть дозволити собі тільки фірми-гіганти, такі як IBM, Motorola, HP. Також деякі розробники програмованих інтегральних схем, такі як Xilinx та Altera, включають в свої інструменти розробки засоби для інтеграції інструментів для формальної верифікації інших компаній.

РОЗДІЛ 2
ІНСЕРЦІЙНА СЕМАНТИКА МОВИ VHDL

2.1 Загальні принципи роботи мови VHDL

VHDL доволі складна мова, яка складається з різноманітних конструкцій. Її конструкції можуть породжувати нескінченні моделі на основі циклів або рекурсій. Багато конструкцій можуть виражатися у термінах більш базових конструкцій, зберігаючи ту саму семантику. Процес створення інсерційної семантики мови VHDL полягає у визначенні та побудові інсерційної семантики основної підмножини мови, а потім у поступовому розширенню цієї підмножини. Елементами вказаної підмножини є:

- Суб’єкти проєктування: entity declaration, architecture statement.
- Паралельні оператори: process statement(без sensitivity list, але з wait statements).
- Послідовні оператори: wait statement, signal assignment statement.
- Декларації: signal declaration.

У цю підмножину не було включено деякі з важливих послідовних операторів, такі як if statement, case statement, while statement, variable assignment statement та ін., тому що їх поведінка є доволі типовою для багатьох мов програмування, а тому було вирішено зосередитись на операторах з нетиповою поведінкою.

Simulation cycle

У VHDL цифрові системи моделюються як сукупність процесів обміну сигналами, синхронізації та паралельних процесів у реальному часі. Сукупність всіх можливих моделювань цих процесів і є моделлю поведінки цих систем.

Сигнали — це канали зв’язку між різними процесами, які є своєрідними сигнальними дротами, які з’єднують кілька різних частин системи. Якщо тільки один процес призначає значення для сигналу у

певний момент часу, то сигнал приймає нове значення наприкінці циклу. Якщо ж декілька процесів призначають нове значення для сигналу у один і той самий момент часу, то наприкінці циклу буде викликано функцію розрішення, яка визначить яке значення отримає сигнал. До того ж призначення сигналу може бути відкладено. У такому випадку призначення сигналу буде перенесено на наступний цикл, де значення буде присвоєно сигналу або використано у функції розрішення за необхідності.

Процеси у циклі синхронізуються за допомогою операторів очікування (wait statements), які призупиняють виконання послідовних операцій до наступної ітерації циклу симуляції. Також система моделювання вирішує, який із процесів продовжить роботу у даній ітерації на основі умов у операторі очікування. Ці умови засновані на зміні значень одного і більше сигналів, логічному виразу або часі затримки роботи процесу (наприклад, процес може активуватися кожну третю наносекунду роботи системи).

Час у VHDL дискретний. Для VHDL найменша одиниця часу — 1 фемтосекунда (10^{-15} секунд). Однак у окремому циклі симуляції час може як додаватися, так і не додаватися, в залежності від того, чи викликала робота цього циклу миттєву активацію іншого процесу у наступному циклі.

На рис. 2.1 зображена спрощена схема циклу моделювання VHDL. Спочатку модель ініціалізується: сигналам призначаються початкові значення, а процеси виконуються до першої зупинки — до першого оператора очікування. Потім в залежності від того, скільки процесів активуються у даний момент часу, час симуляції буде або додаватися або ні.

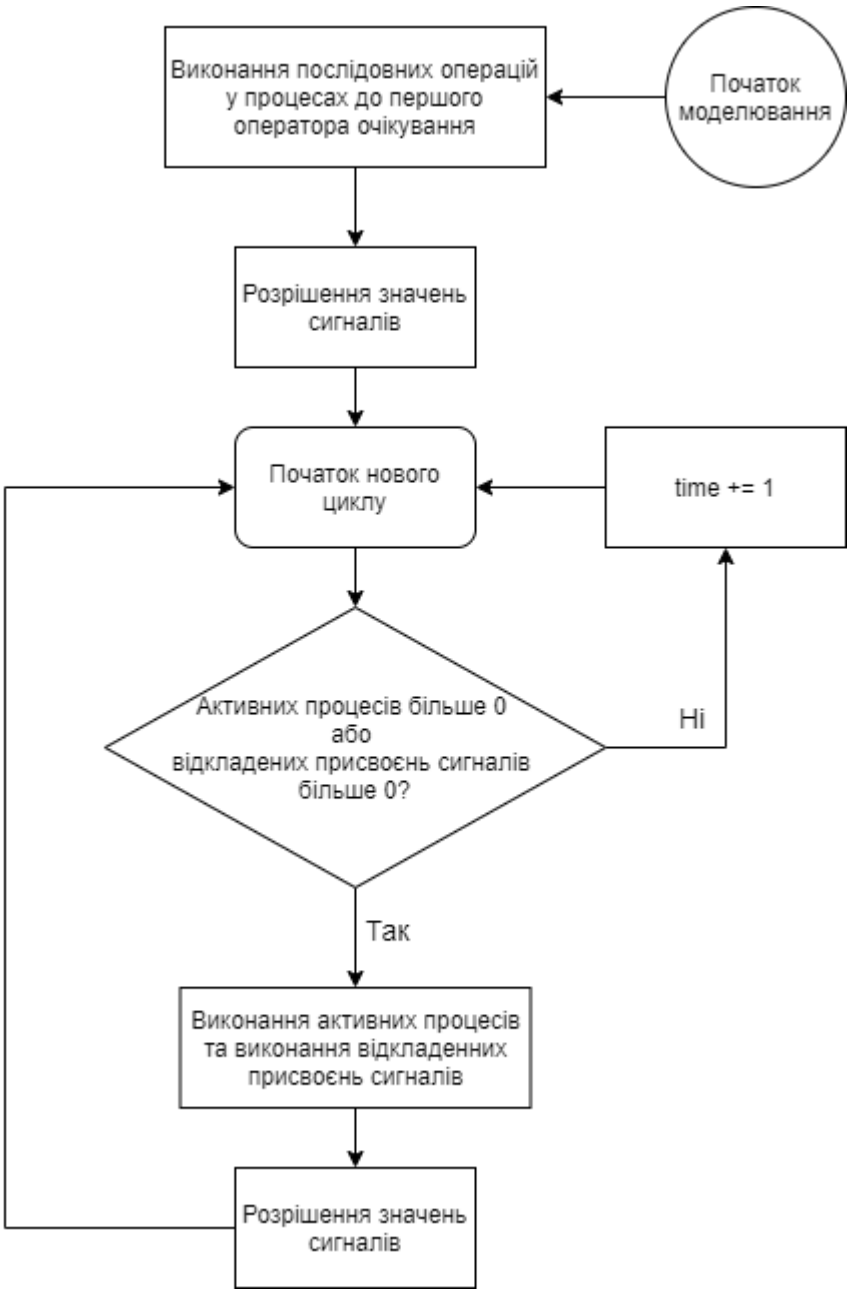

Рисунок 2.1 — Спрощена діаграма роботи циклу симуляції VHDL

Якщо у даний момент часу є хоча б один процес, який може продовжити виконання або хоча б одне заплановане присвоєння сигналу, то час симуляції не інкрементується. Таким чином моделюється майже миттєве розповсюдження сигналів по системі.

Якщо ж у даний момент часу немає активних процесів або запланованих присвоєнь сигналу, то система продовжує інкрементувати час доти, доки система не отримає стимул поновити роботу. Стимул про поновлення роботи системи може бути створений за допомогою

відкладеного присвоєння значення сигналу або за допомогою оператора очікування з паузою за часом.

У загальному вигляді поведінку моделі можна представити так:

```
SYS = INIT.SIGNAL_RESOLUTION.BASE,
BASE = (DO_PROCS.DO_SIGNALS.(
        CYCLE_ISACTIVE.SIGNAL_RESOLUTION +
        !CYCLE_ISACTIVE.ADD_TIME).BASE;
```

2.2 Інсерційна семантика визначеної множини конструкцій мови VHDL

Entity

Оголошення об’єкта вказує, як об’єкт проєкта виглядає зовні і яким чином його можна вставити у іншому об’єкті як компонент, тобто він описує зовнішній інтерфейс об’єкта. Загальне оголошення об’єкта має такий синтаксис:

entity \ідентифікатор\ **is**

[**generic**(\оголошення константи для налаштування\

{; \оголошення константи для налаштування\});]

[**port** (\оголошення порта\ {;\оголошення порта\});]

end [**entity**][\ідентифікатор об’єкта\];

Entity буде визначати атрибути та сигнали, які будуть створені для інсерційної моделі окремої архітектури проєкту VHDL.

Port

Порт представляє собою інтерфейсний сигнал об’єкта проєкту. Так само як і під час оголошення сигналу, в оголошенні порту вказуються його ідентифікатор, тип та початкове значення. Однак додатково вказується режим роботи порта:

- in — прийом,
- out — передача,
- inout — прийом та передача,
- buffer — передача й використання як сигнал-операнд

всередині об’єкта проєкту,

- link — двунапрямлене з’єднання з іншим портом с таким самим режимом.

Зважаючи на те, що порт — це сигнал, але розширений режимом роботи, у інсерційній семантиці він виглядатиме так само як і звичайний сигнал (описано нижче). Але, за потреби, можна буде внести декілька обмежень до моделі передачі сигналу, які враховуватимуть режим роботи порта.

Architecture

Архітектура об’єкта представляє собою окрему частину, де описано, яким чином реалізований об’єкт (Рис. 2.2).

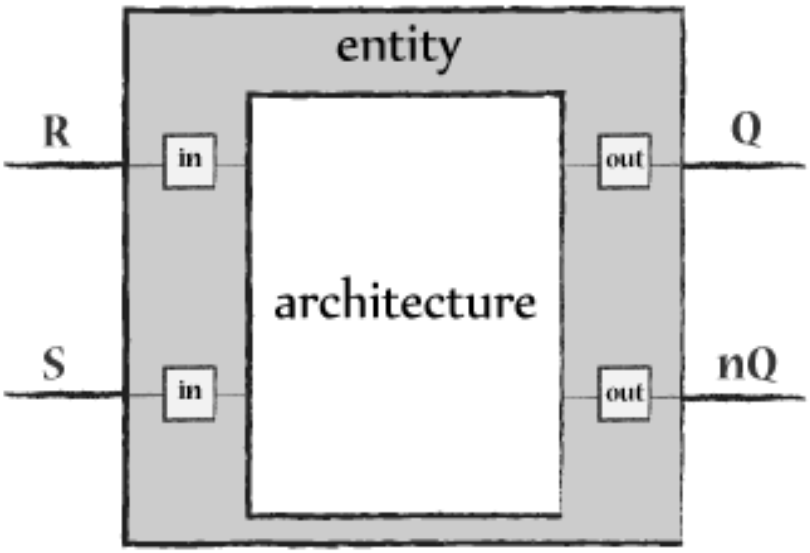

Рисунок 2.2 — Схема відношення архітектури та об’єкта

Архітектура має наступний синтаксис:

architecture \ідентифікатор\ **of** \ім’я об’єкта\ **is**

{\блокове оголошення\}

begin

{ \паралельний оператор\}

end [architecture][\ідентифікатор\];

Ідентифікатором позначається ім’я даної архітектури, а ім’я об’єкта вказує на те, який з об’єктів описує дана архітектура. Одному об’єкту може відповідати декілька архітектур, в кожній з яких описується один з варіантів реалізації об’єкта.

Блоковим оголошенням може бути:

- оголошення функцій,
- типів,
- підтипів,
- змінних,
- сигналів, тощо.

Головну частину архітектури складають паралельні оператори, такі як процес, паралельне присвоєння сигналу, тощо. Всі ці оператори виконуються паралельно. Як висновок, всі паралельні оператори з усіх архітектур також виконуються паралельно. Незважаючи на те, що окрім процесу існують ще багато різних паралельних операторів, всі їх можна описати за допомогою процесу, тому будемо розглядати тільки такий паралельний оператор.

Спираючись на те, що архітектури виконуються паралельно, і процеси в них виконуються також паралельно, можна запропонувати таку поведінку архітектур та процесів:

```
INIT = {ARCH1_INIT || ARCH2_INIT || ...},
ARCHx_INIT = {
        ARCHx_P1_INIT || ARCHx_P2_INIT || ...},
DO_PROCS = {ARCH1_PROCS || ARCH2_PROCS || ...},
ARCHxPROCS = {ARCHx_P1 || ARCHx_P2 || ...}
```

Process

Оператор процесса — паралельний оператор, який є основою мови VHDL. Його спрощений синтаксис:

process[(\лист чутливості\)] **is**

{\оголошення процеса\}

begin

{\послідовний оператор\}

end process;

Оголошеннями в процесі можуть бути: оголошення типів, констант, змінних, тощо. Те, що оголошено всередині процесу, є локальним для цього процесу.

Всі процеси в системі виконуються паралельно. Процеси обмінюються сигналами, які виконують синхронізацію процесів і переносять значення між ними. Якщо тип сигналу має функцію розрішення, то виходи декількох драйверів сигналу можуть об’єднуватись за деяким алгоритмом. Процес може включати в себе тільки послідовні оператори.

Лист чутливості — множина сигналів, при зміні одного з яких активується процес. Ця форма оператора альтернативна до оператора очікування процесу “wait on”, який стоїть останнім в ланцюгу послідовних операторів. Будь-який процес зі списком чутливості може бути перетворений в еквівалентний процес з оператором “wait on”, який ставиться останнім послідовним оператором у процесі. Але ставити оператор очікування у процес з списком чутливості не допускається.

Для моделювання логічних схем в список чутливості процеса необхідно вносити всі вхідні сигнали всередині процесу, інакше поведінка схеми може відрізнятись від очікування.

Для перетворення процесу у інсерційну модель будемо перетворювати всі процеси з списками чутливості у аналогічні з оператором очікування “wait on”. Для опису інсерційної моделі процесу потрібно визначити порядок його виконання з урахуванням:

- частина до першого оператора очікування в процесі буде блоком ініціалізації, він буде виконуватись на початку поведінки системи
- потім у кожному циклі будуть виконуватись поступово частини процесу, але тільки якщо виконується умова поточного оператора

очікування; частини процесу будуть виділятися між кожними сусідніми операторами очікування, починаючи з блоку після першого оператора очікування

- номер поточної частини буде перемикатись якщо виконується умова поточного оператора очікування, а також скидатися у початковий стан при досягненні значення кількості частин процесу.

Згідно з зазначеними принципами було отримано опис середовища, початковий стан, дії та поведінку:

```
environment(
        attributes:(a_procX_state:integer);
        initial_formula:obj(a_procX_state==0);
);
procX_ifpartY = ((a_procX_state==Y &&
        (\умова оператору очікування\))->
        (“”)(1));
procX_increment_state = (
        (a_procX_state<N)->
        (“”)(a_procX_state==a_procX_state+1));
procX_nullify_state= (
        (a_procX_state==N)->
        (“”)(a_procX_state==0));
ARCH1_Px = (
        (procX_ifpart0.PROCx_PART0)+
        (procX_ifpart1.PROCx_PART1)+
        ...
        ).PROCx_CHANGE_STATE,
PROCx_CHANGE_STATE =
        procX_increment_state.procX_nullify_state;
```

Signal declaration

Сигнал - об’єкт який переносить значення від одного процесу в інший і, разом з цим, синхронізує систему. Це первинний об’єкт, який описує аппаратурну систему та аналогічний провідникам.

Використовується для моделювання таких апаратних властивостей як паралелізм, інерціальність, декілька драйверів тощо. Оголошення сигналу має наступний синтаксис:

\оператор оголошення сигналу \ ::=

signal : \ідентифікатор\{,\ідентифікатор\} :

\тип сигналу\ := [\початкове значення\];

Ідентифікатори сигналів можуть перераховуватись через кому, якщо всі сигнали мають однаковий тип сигналу, та, за потребою, початкове значення, яке є константою. Типів сигналів у VHDL багато, тому для простоти будемо використовувати найпростіший тип BIT.

Сигнал має свій набір атрибутів, які відіграють ключову роль у активації процесів. Наш набір складатиметься з таких атрибутів, як:

- S`event — тип BOOLEAN — Для скалярного сигналу — приймає true якщо в поточному циклі симуляції відбулася зміна сигналу; для складеного типу — повертає true якщо відбулася зміна хоча б одного з складових сигналів.
- S`transaction — тип BIT — Перемикається з 1 на 0 та навпаки у кожному simulation cycle в якому сигнал стає active.
- S`last_value — тип type(S) — Повертає значення вигналу перед останнім event або поточне значення сигналу.
- S`active — тип BOOLEAN — Приймає true якщо у поточному simulation cycle є присвоєння сигналу. (Пов'язане з transaction)

У інсерційній семантиці оголошення сигналу та його атрибутів буде виглядати так:

```
environment(
        attributes:(
                a_signalX:bool,
                a_signalX__event:bool,
                a_signalX__transaction:bool,
                a_signalX__last_value:bool,
                a_signalX__active:bool));
```

До того ж, якщо під час оголошення сигналу буде вказано значення для ініціалізації, то наше оточення інсерційної моделі доповниться формулою ініціалізації:

```
initial_formula:(
        (a_signal1 == A) &&
        (a_signal2 == B) &&
        ...);
```

Wait statement

Цей оператор вже згадувався під час опису принципів роботи симуляторів. На цьому операторі зупиняється виконання процесу. У момент зупинки виконуються присвоєння сигналів і потім процеси продовжують виконуватися при появі подій, які описуються у даному операторі очікування. Синтаксис оператора очікування має наступний вигляд:

wait [**on** \лист чутливості\][**until** \логічний вираз\][**for** \вираз часу\];

Таким чином оператор очікування може мати три можливі частини, які починаються з таких ключових слів: “on”, “until”, “for”:

- Перша частина починається ключовим словом “on” за яким йде лист чутливості. Саме ця частина дозволяє зробити альтернативний опис процесу з листом чутливості. Вона описує, що процес продовжить роботу коли один із сигналів у списку змінить свій стан. Зробити перевірку, чи змінився сигнал, можна за допомогою атрибута сигналу S`event, який стає true кожен раз коли в даному циклі змінюється значення сигналу.
- Друга частина починається ключовим словом “until” за яким іде логічний вираз, виконання якого призводить до активації процесу.
- Третя частина починається ключовим словом “for”, за яким іде вираз часу. Ця частина декларує призупинення процесу на певний період часу від першого визову оператора wait з цією частиною.

Якщо в операторі вказано декілька частин, то він активується в тому

випадку, коли буде виконано всі умови з кожної з визначених частин.

У разі, якщо оператор wait вказано без якої-небудь частини, то процес, дійшовши до нього призупиняється назавжди. Якщо в процесі без списка чутливості немає жодного оператора очікування, то цей процес буде виконуватись безкінечно, не передаючи управління іншим процесам.

Таким чином, проаналізувавши поведінку оператора wait, було визначено наступний базовий протокол інсерційної семантики оператора очікування:

```
act_waitX = (
        ((a_signalA__event==1 && ...) && ...) -> (“”)(1));
```

Однак, слід зауважити, що даний протокол необхідно буде об’єднати з протоколом визначення поточної частини процесу “procX_ifpartY”, тобто будуть об’єднані передумови цих протоколів через оператор AND. До того ж, враховуючи те, що ми ввели такий атрибут як “a_waitX__act_time” для запам’ятовування часу активації оператора очікування, нам потрібно визначити, як він буде встановлюватись. Було отримано наступні протоколи та поведінку для процесу архітектури:

```
procX_ifpartY = ((a_procX_state==Y &&
                (\умова оператору очікування\))->(“”)(1));
procX_ifpartY_nactive = ((a_if_isactive==0)->(“”)
        (a_iftime=a_time && a_if_isactive=1));
act_idle = ((a_procX_state==Y &&
                !(\part wait conditions\))->(“”)(1));
act_ifreset = ((1)->(“”)(a_if_isactive=0));

ARCH1_Px = (PROCx_IFPART0 + PROCx_IFPART1 + ...),
PROCx_IFPARTy = (
        procX_ifpartY.PROCx_PARTy.procX_ifpartY_reset.
                                        PROCx_CHANGE_STATE +
        procX_ifpartY_idle.(procX_ifpartY_nactive +
                                        !procX_ifpartY_nactive));
```

Як можна побачити, дана семантика перевизначає попередньо описану для процесу інсерційну семантику. Було зроблено наступні зміни:

- об’єднано протокол для визначення частини процесу та протокол для визначення того, чи виконуються умови оператора очікування;
- якщо була обрана частина та умова оператора очікування виконалась, то виконуємо частину процесу, після чого деактивуємо поточний оператор очікування та перемикаємо стан процесу;
- якщо частина не змогла активуватися через невиконання умов оператора очікування, то один раз активуємо оператор очікування та встановлюємо час його активації.

Sequential signal assignment

Оператор послідовного присвоєння сигналів може бути вставлений тільки у процесах. Виконання оператора присвоєння сигналу тільки розраховує значення виразу та планує його для присвоєння. Фактичне ж присвоєння виконується під час зупинки процесу на операторі очікування. Також, як було зазначено раніше, якщо присвоєння відбувалося паралельно з декількох процесів, то перед присвоєнням відпрацьовує функція розрішення, яка визначає кінцеве для присвоєння значення. Тому, якщо в процесі стоїть декілька операторів присвоєння для одного сигналу, то буде враховуватись тільки останній, а всі попередні ігноруватися. Також, якщо перед зупинкою процесу виконувалось зчитування цього сигналу, то буде зчитано значення, яке було встановлено цьому сигналу на минулому циклі симуляції. Простий оператор присвоєння має наступний синтаксис:

\сигнал\ <= \вираз для присвоєння\

Під час моделювання дискретних сигналів важливе місце посідає моделювання розповсюдження сигналу з урахуванням затримок в провідниках або затримок у електронних вентилях. Мова VHDL має

декілька різних видів затримок, але ми розглянемо тільки два з них:

- transport — означає звичайну затримку передачі сигналу, тобто сигнал буде присвоєно щойно пройде час затримки;
- inertial — реалізує поведінку затримки сигналу у джерелі, який не реагує на занадто короткі вхідні імпульси; сигнал буде присвоєно після затримки тільки у тому випадку, якщо значення входу не змінювалось за час затримки; вид затримки за замовчуванням.

Розширений послідовний оператор присвоєння сигналу має наступний вигляд:

\сигнал\ <= [**transport** | **inertial**] \вираз для присвоєння\

[**after** \вираз часу затримки\]

Враховуючи вищезазначені особливості, було сформовано наступні твердження, необхідні для формування семантики:

- кінцева інсерційна модель повинна мати окремого агента, який буде розрішувати сигнали;
- для кожного з видів затримки буде створено інсерційні процеси, які будуть мати різний набір параметрів;
- якщо присвоєння сигналу має певну затримку, то подія про присвоєння буде рекурсивно пересилатися у наступний цикл, поки не закінчиться час затримки;
- для кожного окремого сигналу буде створено окремий розрішувач, в залежності від типу даних.

Для кожного з видів затримки було описано інсерційні процеси. Процес виду **transport** має наступний список параметрів:

- значення для присвоєння,
- час відправки,
- час затримки.

Для **inertial**:

- значення для присвоєння,
- вираз правої частини операції присвоєння,

- час відправки,
- час затримки.

У інсерційній семантиці це виглядає наступним чином:

```
events(obj(
        (assign_transport_<signal_name>:
          obj(<signal_type>, integer, integer)),
        (assign_inertial_<signal_name>:
          obj(<signal_type>, integer, integer, integer));
);
```

Описані вище інсерційні процеси будуть надсилатися подіями від агента процесу до агента розрішувача сигналів. Тому потрібно описати агентів для кожного з процесів VHDL моделі, а також агента для розрішувача сигналів. Опис оточення з агентами процесів та сигнальним агентом буде мати наступний вигляд:

```
environment(
   agent_types:obj(
                              PROC:obj(Nil));
      SEnv:obj(Nil));
   agents:obj(
            PROC:(p,...),
      SEnv:(senv));
);
```

Протокол, який буде надсилати інсерційні процеси від агента процесу “p” до сигнального агента “senv”, буде мати наступний вигляд для **transport** затримки присвоєння сигналу:

```
act_assign_<signal_name> = ((1)->
        (“PROC#p:out assign_transport_<signal_name>(/значення/, a_time,
/час затримки/) to SEnv#senv;”)(1));
```

Після того, як події були надіслані до агента “senv”, який займатиметься розрішенням сигналів та Їх присвоєння, ми повинні описати поведінку самого розрішувача. Розрішувач буде складатися з паралельних розрішувачів для кожного з сигналів системи. Кожен з сигналів системи буде, в залежності від виду затримки, буде аналізуватися та присвоюватись або рекурсивно відправлятися у

наступний цикл. Отримана поведінка буде мати наступний вигляд:

```
SIGNAL_RESOLUTION =
        (RESOLVE_SIGNAL1 + RESOLVE_SIGNAL2 + ...),
RESOLVE_SIGNALx = (
    get_transport.(
                transport_expired.plan_assign +
        !transport_expired.send_recursive_transport) +
    get_inertial.(expr_isequal.(
                inertial_expired.plan_assign +
        !inertial_expired.send_recursive_inertial) +
      !expr_esequal));
```

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ІНСЕРЦІЙНОЇ СЕМАНТИКИ VHDL. ОБГРУНТУВАННЯ ВИМОГ ТА ПЕРВИННА МОДЕЛЬ РОБОТИ

Оскільки ручний переклад програми мовою VHDL у інсерційну семантику займає багато часу та може призвести до появи зайвих помилок, тому було вирішено автоматизувати даний процес. До того ж, отримання інсерційної семантики дало змогу створити програмний засіб для автоматичного перекладу певних структур мови VHDL у певні структури інсерційної моделі відповідно до семантики.

Для вирішення цієї задачі було спроєктовано систему, яка проводить обробку VHDL моделі у два етапи (Рис. 3.1):

1. Зчитування та попередня обробка моделі VHDL у центральну модель. Під час даного етапу конструкції будуть об’єднуватись всередині програми у певну ієрархію, а також деякі більш складні конструкції будуть спрощуватись до більш простих, але рівнозначних за значенням.
2. Генерування конструкцій інсерційної моделі у модель на основі центральної моделі. На даному етапі генеруються конструкції кінцевої інсерційної моделі на основі шаблонів перетворення тих чи інших конструкцій згідно інсерційної семантики.

Для побудови транслятора було обрано мову Python3 [24,25]. Це досить проста у використанні, потужна та універсальна мова, що робить її гарним вибором для цієї задачі. До того ж, вона має безліч бібліотек для різного роду задач.

Рисунок 3.1 — Загальна модель роботи транслятора

Побудова парсера

Для побудови транслятора було обрано інструмент ANTLR (ANother Tool for Language Recognition) [26,27,28]. Це потужний генератор парсерів, який на основі граматики мови генерує синтаксичний аналізатор, який може будувати дерева синтаксичного аналізу, а також генерує інтерфейс відвідувача, який дозволяє легко реагувати на розпізнавання певних конструкцій чи фраз. До того ж, цей інструмент має генератор для мови Python3. Для генерування парсера було взято повну граматику мови VHDL. Уся граматика мови зберігається у файлі з розширенням *.g4 та описується як контекстно вільна граматика [29] у розширеній формі Бекуса-Наура та має такий вигляд:

```
architecture_body
 : ARCHITECTURE identifier OF identifier IS
   architecture_declarative_part
```

```
BEGIN
architecture_statement_part
END ( ARCHITECTURE )? ( identifier )? SEMI;
```

Для зберігання взятої з вхідного файла VHDL інформації про архітектури, процеси, сигнали було розроблено систему класів, де для кожного з них були описані необхідні параметри. Наприклад, для зберігання інформації про архітектуру було створено клас “Architecture”, який має наступні параметри:

- назва архітектури,
- назва об’єкта (Entity),
- список сигналів, які були об’явлені у об’єкті,
- список процесів, або інших паралельних операторів, але приведених у уніфікований вигляд.

Побудова транслятора

Після того, як центральну модель було створено на основі аналізу файлу проєкта VHDL, транслятор починає процес генерації структур інсерційної моделі. Він проходить по всім об’єктам центральної моделі та генерує опис середовища інсерційної моделі, який включає в себе атрибути сигналів, початкові формули, типи агентів та самих агентів.

Другим етапом відбувається опис базових протоколів, які будуються за шаблонами. Це базові протоколи для процесів, для їх частин, базові протоколи для послідовних операторів, а також для етапу розрішення сигналів. Також на даному етапі генеруються потрібні для моделі інсерційні процеси для кожного з сигналів.

Кінцевим етапом є побудова поведінки зі створених дій та генерація необхідних вихідних файлів.

ВИСНОВКИ

Під час роботи над даним проєктом було проаналізовано велику кількість літератури, яка описує принципи роботи інтегральних логічних схем, мову VHDL, яка використовується для їх конфігурації, методи та принципи формальної верифікації та принципи роботи систем інсерційного моделювання та її застосування у якості інструменту для формальної верифікації. До того ж, було виконано ряд задач по створенню інсерційної семантики мови опису апаратури інтегральних схем VHDL.

Було визначено, що формальна верифікація — це дуже складний, але і надзвичайно потрібний процес для створення програмного забезпечення та систем підвищеної стійкості. Формальна перевірка системи дозволяє довести, що система, наприклад, не може мати критичних станів та відповідає специфікації на 100%. Це критично необхідно для систем, від яких залежить життя людей, або просто для надто дорогих та складних систем.

Було проаналізовано синтаксис і семантику мови опису алгебри поведінки для системи IMS. Ця система заснована на принципах взаємодії агентів та середовищ. Для опису моделей використовується мова алгебраїчного програмування APLAN, яка розглядається авторами системи IMS як базова. Було визначено основні структури опису інсерційних моделей, такі як: середовище, процеси, базові протоколи та поведінка. За допомогою цих структур може бути описана система будь-якої складності.

Було проаналізовано синтаксис і семантику мови опису апаратури інтегральних схем VHDL. Це одна з базових мов для конфігурування інтегральних логічних схем. Ця мова дозволяє описувати формальні моделі роботи цифрових та змішаних логічних схем, моделювати їх роботу за певними принципами та створювати на основі цих моделей

конфігурації для програмованих логічних інтегральних схем. Однак ці системи часто мають велику кількість станів та паралельних процесів, де можуть бути заховані помилки. Для вирішення цієї проблеми було розпочато роботу зі створення інсерційної семантики та транслятора у інсерційну модель моделей VHDL.

Як результат аналізу, було створено правила з перетворення моделі VHDL у інсерційну модель — інсерційну семантику мови VHDL. Для цього було визначено принципи перекладу певних структур мови VHDL у структури опису інсерційної моделі мовою APLAN. Були розглянуті такі структури мови VHDL, як: об’єкт, архітектура, процес, сигнал, послідовний оператор присвоєння сигналу та оператор очікування.

Після того, як було створено базову інсерційну семантику для спрощення побудови інсерційних моделей систем VHDL, було розроблено транслятор у інсерційну модель. Він дозволяє автоматично створювати інсерційну модель на основі файлів проєкту мовою VHDL. Структура отриманого транслятора складається з процесу зчитування проєкту VHDL у певну центральну модель та етапу з трансляції цих структур у інсерційну модель проєкту VHDL.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Hu Y. Exploring formal verification methodology for FPGA-based digital systems. Sandia National Laboratories, New Mexico, California. 2012. 58 p.
2. Déharbe D., Shankar S., Clarke E. M. Formal verification of VHDL: the model checker CV. Proceedings of XI Brazilian Symposium on Integrated Circuit Design (Cat. No. 98EX216). IEEE, 1998. P. 95-98.
3. Zaki M. H., Tahar S., Bois G. Formal verification of analog and mixed signal designs: A survey. Microelectronics journal. 2008. Т. 39. №. 12. P. 1395-1404.
4. VHDL Online Help. URL: http://vhdl.renerta.com/mobile/index.html.
5. Subcommittee D. A. S. IEEE standard VHDL language reference manual. IEEE,1988. P. 1076-1987.
6. Letichevsky A. A., Letychevskyi O. A., Peschanenko V. S. Insertion modeling system. International Andrei Ershov Memorial Conference on Perspectives of System Informatics. – Springer, Berlin, Heidelberg, 2011. P. 262-273.
7. APS&IMS Systems. URL: http://apsystems.org.ua/.
8. Сергієнко А. М., Виноградов Ю. М., Лесик Т. М. Цифрова обробка сигналів. Комп’ютерний практикум мовою VHDL. К.: НТУУ «КПІ», 2012. 104 р.
9. Armstrong R. C. et al. Survey of existing tools for formal verification. SANDIA REPORT SAND 2014-20533. 2014. 42 p.
10. Coelho D. R. The VHDL handbook. Springer Science & Business Media, 1989. 390 p.
11. Chang K. C. Digital systems design with VHDL and synthesis. IEEE computer society press, 1999. 300 p.
12. Ott D. E., Wilderotter T. J. A designer's guide to VHDL synthesis. Springer, 2013. 306 p.

13. Ashenden P. J. The designer's guide to VHDL. Morgan Kaufmann, 2010. 936 p.
14. Maxfield, Clive. The design warrior's guide to FPGAs: devices, tools and flows. Elsevier, 2004. 542 p.
15. Skahill K. VHDL for programmable logic. Addison-Wesley Longman Publishing Co., Inc., 1996. 600 p.
16. Угрюмов Е.П. Цифровая схемотехника: учебное пособие для вузов. СПб.: БХВ-Петербург, 2007. 800 с.
17. Keller R. M. Formal verification of parallel programs. Communications of the ACM. 1976. Т. 19. №. 7. P. 371-384.
18. Drechsler, Rolf, ed. Advanced Formal Verification. Springer Science & Business Media, 2007. 250 p.
19. Letichevsky A. Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications / A. Letichevsky et al. *Computer Networks*. 2005. P. 16.
20. Systems specification by basic protocols // Letichevsky A. A. et al. Cybernetics and Systems Analysis. 2005. Т. 41. №. 4. P. 479-493.
21. Letichevsky A.A., Letichevsky A.A. Jr., Peschanenko V.S. Translation Algorithm of APLAN Code. *Control Systems and Machines*. 2010. Vol. 6. P. 40-46. URL: http://apsystems.org.ua/uploads/doc/aps/ATAC.rus.pdf
22. Meseguer J. Conditional rewriting logic as a unified model of concurrency. Theoretical computer science. 1992. Т. 96. №. 1. P. 73-155.
23. Inamdar S. L. Vhdl coding style guidelines and synthesis: A comparative approach. 2004. 85 p.
24. Reitz K., Schlusser T. The Hitchhiker's Guide to Python: Best Practices for Development. " O'Reilly Media, Inc.", 2016. 338 p.
25. Python Docs. URL: https://www.python.org/doc/.
26. Parr T. The definitive ANTLR 4 reference. Pragmatic Bookshelf, 2013. 326 p.
27. Parr T. J., Quong R. W. ANTLR: A predicated-LL (k) parser generator.

Software: Practice and Experience. 1995. T. 25. №. 7. P. 789-810.

28. What's ANTLR // Parr T. et al. 2004. 152 p.
29. Kasami T. An efficient recognition and syntax-analysis algorithm for context-free languages. Coordinated Science Laboratory Report no. R-257. 1966. 56 p.
30. Validation of Embedded Systems // J. Kapitonova et al. The Embedded Systems Handbook. CRC Press, Miami, 2005, P. 58.

ДОДАТКИ

Додаток А

**КОДЕКС АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ
ЗДОБУВАЧА ВИЩОЇ ОСВІТИ ХЕРСОНСЬКОГО
ДЕРЖАВНОГО УНІВЕРСИТЕТУ**

Я, Болгарін Тимофій Олександрович, учасник освітнього процесу Херсонського державного університету, **УСВІДОМЛЮЮ**, що академічна доброчесність – це фундаментальна етична цінність усієї академічної спільноти світу.

ЗАЯВЛЯЮ, що у своїй освітній і науковій діяльності **ЗОБОВ'ЯЗУЮСЯ**:

– дотримуватися:

- вимог законодавства України та внутрішніх нормативних документів університету, зокрема Статуту Університету;
- принципів та правил академічної доброчесності;
- нульової толерантності до академічного плагіату;
- моральних норм та правил етичної поведінки;
- толерантного ставлення до інших;
- дотримуватися високого рівня культури спілкування;

– надавати згоду на:

- безпосередню перевірку курсових, кваліфікаційних робіт тощо на ознаки наявності академічного плагіату за допомогою спеціалізованих програмних продуктів;
- оброблення, збереження й розміщення кваліфікаційних робіт у відкритому доступі в інституційному репозитарії;
- використання робіт для перевірки на ознаки наявності академічного плагіату в інших роботах виключно з метою виявлення можливих ознак академічного плагіату;

– самостійно виконувати навчальні завдання, завдання поточного й підсумкового контролю результатів навчання;

– надавати достовірну інформацію щодо результатів власної навчальної (наукової, творчої) діяльності, використаних методик досліджень та джерел інформації;

– не використовувати результати досліджень інших авторів без використання покликань на їхню роботу;

– своєю діяльністю сприяти збереженню та примноженню традицій університету, формуванню його позитивного іміджу;

– не чинити правопорушень і не сприяти їхньому скоєнню іншими особами;

– підтримувати атмосферу довіри, взаємної відповідальності та співпраці в освітньому середовищі;

– поважати честь, гідність та особисту недоторканність особи, незважаючи на її стать, вік, матеріальний стан, соціальне становище, расову належність, релігійні й політичні переконання;

– не дискримінувати людей на підставі академічного статусу, а також за національною, расовою, статевою чи іншою належністю;

– відповідально ставитися до своїх обов'язків, вчасно та сумлінно виконувати необхідні навчальні та науково-дослідницькі завдання;

– запобігати виникненню у своїй діяльності конфлікту інтересів, зокрема не використовувати службових і родинних зв'язків з метою отримання нечесної переваги в навчальній, науковій і трудовій діяльності;

– не брати участі в будь-якій діяльності, пов'язаній із обманом, нечесністю, списуванням, фабрикацією;

– не підроблювати документи;

– не поширювати неправдиву та компрометуючу інформацію про інших здобувачів вищої освіти, викладачів і співробітників;

– не отримувати і не пропонувати винагород за несправедливе отримання будь-яких переваг або здійснення впливу на зміну отриманої академічної оцінки;

– не залякувати й не проявляти агресії та насильства проти інших, сексуальні домагання;

– не завдавати шкоди матеріальним цінностям, матеріально-технічній базі університету та особистій власності інших студентів та/або працівників;

– не використовувати без дозволу ректорату (деканату) символіки університету в заходах, не пов'язаних з діяльністю університету;

– не здійснювати і не заохочувати будь-яких спроб, спрямованих на те, щоб за допомогою нечесних і негідних методів досягати власних корисних цілей;

– не завдавати загрози власному здоров'ю або безпеці іншим студентам та/або працівникам.

УСВІДОМЛЮЮ, що відповідно до чинного законодавства у разі недотримання Кодексу академічної доброчесності буду нести академічну та/або інші види відповідальності й до мене можуть бути застосовані заходи дисциплінарного характеру за порушення принципів академічної доброчесності.

05.05.2020		Болгарін Тимофій
(дата)	(підпис)	(ім'я, прізвище)