НОВІ ЗАСОБИ КІБЕРНЕТИКИ, ІНФОРМАТИКИ, ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ ТА СИСТЕМНОГО АНАЛІЗУ

УДК 004.05, 004.4[2+9], 004.94, 519.7

О.О. ЛЕТИЧЕВСЬКИЙ
Інститут кібернетики ім. В.М. Глушкова НАН України, Київ, Україна,
e-mail: *oleksandr.letychevskyi@litsoft.com.ua*.

О.М. ОДАРУЩЕНКО
Науково-виробниче підприємство «Радікс», Кропивницький, Україна,
e-mail: *odarushchenko@gmail.com*.

В.С. ПЕСЧАНЕНКО
Херсонський державний університет, Херсон, Україна,
e-mail: *volodymyr.peschanenko@litsoft.com.ua*.

В.С. ХАРЧЕНКО
Національний аерокосмічний університет ім. М.Є. Жуковського «Харківський авіаційний інститут», Харків, Україна, e-mail: *v.kharchenko@csn.khai.edu*.

В.В. МОСКАЛЕЦЬ
Науково-виробниче підприємство «Радікс», Кропивницький, Україна,
e-mail: *viktoriia.moskalets@gmail.com*.

ІНСЕРЦІЙНА СЕМАНТИКА VHDL-МОВИ ЕЛЕКТРОННОГО ДИЗАЙНУ

Анотація. Досліджено проблему інсерційної семантики специфікацій апаратного забезпечення, зокрема мови VHDL. Побудова семантики потрібна для представлення первинного коду мови VHDL у вигляді інсерційної моделі за допомогою алгебри поведінок. Це представлення дає змогу широко застосовувати формальні методи інсерційного моделювання для верифікації електронних проєктів критичних систем. У статті розглянуто основні конструкції мови VHDL, зокрема процес, архітектуру, паралельні оператори, та їхню інсерційну семантику. У вигляді поведінкових рівнянь побудовано потік керування VHDL-програми. Послідовні оператори представлено як дії алгебри поведінок. Розглянуто проблему перегонів сигналів та методів її виявлення через визначення властивості переставності (permutability).

Ключові слова: мови дизайну апаратного забезпечення, перегони сигналів, переставність, символьне моделювання, алгебра поведінок, інсерційні моделі, системи, що є критичними до безпеки.

ВСТУП

У сучасному процесі проєктування апаратного забезпечення та програмованих логічних систем проблема забезпечення надійності залишається актуальною, особливо у випадку критичних систем, для яких функціональна та інформаційна безпека є надзвичайно важливими. Пошук уразливостей, формальна верифікація та тестування є невід'ємними етапами проєктування, особливо для модельного способу розроблення [1]. Важливою складовою розроблення є визначення еквівалентності специфікацій на різних рівнях абстракції. Наприклад, під час проєктування пристроїв мікроелектроніки вимоги можуть бути визначені за допомогою мови UML [2] або в текстовому вигляді, водночас дизайн пристрою може бути створений мовою VHDL [3] або System Verilog [4] і далі втілений у бінарному коді або в макеті плати.

Є багато систем, які забезпечують формальну верифікацію, перевірку вразливостей та генерацію тестових наборів для апаратних систем, мов вимог або дизайну. В основу цих систем покладено моделювання, як з використанням імітаційного, так і символьного або алгебраїчного підходу, а також статичних методів доведення властивостей.

Найбільш відомими компаніями, що працюють у галузі комп'ютерного дизайну мікроелектроніки та розроблення систем формальної верифікації, є Synopsys (інструментарій VC Formal [5]), Xilinx (платформа Vivado Verification [6]) та Cadence (платформа JasperGold [7]).

Алгебраїчний підхід є досить популярним для використання у процесах верифікації та тестування. Що не рік, то прискорюється розвиток сучасних систем-«розв'язувачів» і машин доведення властивостей, зокрема з використанням методів машинного навчання. У цій роботі розглянуто інсерційне моделювання [8], як узагальнення алгебраїчного моделювання для множин агентів і середовищ, та інсерційну семантику мови VHDL, що представлена за допомогою алгебри поведінок [9].

Метою статті є розвиток формалізмів і демонстрація певної ефективності методів інсерційного моделювання та інсерційної семантики у разі їхнього застосування для розроблення та верифікації електронних проєктів програмованих пристроїв і апаратного забезпечення.

ІНСЕРЦІЙНЕ МОДЕЛЮВАННЯ Й АЛГЕБРА ПОВЕДІНОК

Інсерційне моделювання є узагальненням теорії транзитивних систем. Основними сутностями в інсерційному моделюванні є агенти та середовища. Кожен агент є транзитивною системою або сутністю, яка змінює свій стан і має відповідну поведінку. Агенти взаємодіють між собою через типізовані атрибути, які визначають тип агента. Зміна атрибутів відбувається шляхом обміну сповіщеннями (message passing), причому агенти нічого не знають один про одного. Сутність «середовище» знає все про агентів та забезпечує занурення (insertion) в неї агентів та їхню взаємодію. Це визначається функцією занурення, яку представляють за допомогою алгебри поведінок та мови дій [10].

Алгебра поведінок — це двосортна алгебра над двома базовими множинами, елементами яких є відповідно дії та поведінки. Основними операціями алгебри поведінок є префіксінг (операція «.») та недетермінований вибір (операція «+»). Рівняння в алгебрі поведінок є рівністю, в якій у лівій частині представлено ідентифікатор поведінки, а у правій — поведінковий вираз, що також може містити паралельну та послідовну композицію. Якщо послідовна композиція A;B двох поведінок A і B визначає, що за поведінкою A слідує поведінка B, то паралельна композиція A||B розглядає різні варіанти послідовностей компонент кожної поведінки. У загальному вигляді поведінка являє собою дерево виконання дій. Кожна дія визначається передумовою над атрибутами агента, післяумовою, що визначає зміну атрибутів агента, та ілюстрацією переходу стану агента (процесною компонентою). В описі інсерційної семантики VHDL ілюстрацією слугуватиме зміна сигналів.

Алгебра поведінок є зручним формалізмом для представлення семантики VHDL-мови. Прикладом рівняння в алгебрі поведінок є таке:

$$B0 = (a1.B1) \,||\, ((a2.B2);B3).$$

Поведінка $B0$ визначається паралельною композицією поведінок, перша з яких представлена дією $a1$, що передує поведінці $B1$, а друга визначається послідовною композицією дії $a2$, що передує поведінці $B2$, та поведінки $B3$.

СЕМАНТИКА ПОТОКУ КЕРУВАННЯ VHDL-ПРОГРАМИ

Визначимо ключові елементи семантики потоку керування. Основна структурна сутність ENTITY, яка визначає деякий електронний дизайн у VHDL на будь-якому рівні абстракції, є агентом, що взаємодіє з агентом, який представляє зовнішнє середовище. Агент ENTITY має архітектуру (у мові VHDL це структурна сутність ARCHITECTURE), що визначає його поведінку за допомогою паралельних компонент, які так само є агентами та взаємодіють між собою, а також з агентом ENTITY шляхом сприймання сигналів із відповідних портів. Визначення агента ENTITY створює інтерфейс для сигналів, що надходять із зовнішнього середовища.

Сукупність агентів ENTITY визначає електронний дизайн. Вони можуть бути як ієрархічними, так і ENTITY найвищого рівня, які можуть містити визначення ENTITY нижчих рівнів. Скористаємося поняттям «агент» для позначення деякої сутності, яка може надсилати та приймати сигнали. Для VHDL програми матимемо на увазі, що агент сприймає зміну сигналу та може змінювати значення сигналу. До того ж, агентами у програмі VHDL слугують наведені нижче елементи різного рівня.

Агент DRIVER — це сутність, яка змінює сигнали, що надходять із зовнішнього середовища. У реальному середовищі це може бути користувач апаратного забезпечення або інші електронні пристрої, що надсилають сигнали. Головним завданням агента DRIVER є генерація вхідних сигналів, які за описом є вхідними до сутності ENTITY мови VHDL.

Визначимо також агента TIMER, що здійснює відлік часових тактів певної тривалості. У VHDL-програмі є можливість задати мінімальну тривалість часового проміжку.

За один такт, створений агентом TIMER, агент DRIVER може змінити або один сигнал, або декілька сигналів (можливо, всі сигнали), що описані в декларації PORT як вхідні, або не змінити жодного сигналу.

Нехай $A1, A2, \ldots$ — послідовність вхідних сигналів до сутності як агента VHDL. Позначимо відповідно функціональними символами дію $send(X)$ та дію $receive(X)$, що означають відповідно відправлення інформації про зміну сигналу X із зовнішнього середовища, та отримання інформації агентом ENTITY. У формалізації цього процесу для ілюстрації дій у системі використовують сповіщення (message passing), що мають формат MSC (Message Sequence Chart) [11].

За допомогою формалізмів алгебри поведінок можна записати поведінкове рівняння DRIVER_BEH для агента DRIVER

$$DRIVER\_BEH = (send(A1) + nosignal);(send(A2) + nosignal);\ldots,$$

де $nosignal$ — порожня дія, що визначає відсутність відправлення сигналу агентом DRIVER у межах цього такту. Користуючись послідовною композицією та порожньою дією, можна отримувати будь-які послідовності сигналів. Це рівняння охоплює всі можливі комбінації сигналів, оскільки під час отримання сигналів вважають, що можна одержати різні послідовності згідно із семантикою дії $receive(X)$.

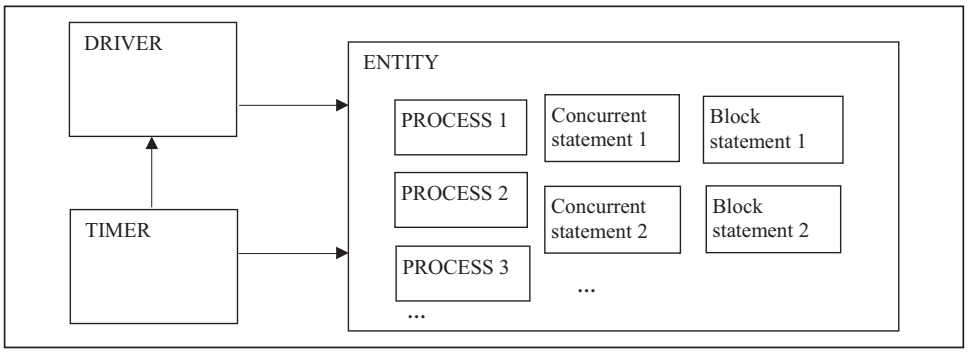

Рис. 1. Взаємодія агентів у VHDL-програмі

Визначимо агента ENTITY, що визначає поведінку VHDL програми, та містить відповідну структурну сутність ARCHITECTURE. Крім описових операторів, тіло ARCHITECTURE містить низку паралельних компонент, а саме окремі паралельні оператори, блоки та процеси. Розглянемо семантику проходження змін сигналів у програмі, яка активізується після приймання сигналу від DRIVER.

Визначимо всі процеси та паралельні оператори також як агентів, що взаємодіють у цьому середовищі. У випадку зміни в межах одного такту деякого сигналу спрацьовують усі процеси та паралельні оператори, що мають цей сигнал у списку чутливості або у правій частині оператора (чи у правій частині послідовного оператора, що міститься в тілі процесу). Зміни сигналів відбуваються по черзі в порядку здійснення зміни агентом DRIVER. Компонента BLOCK також отримує сигнал. Її семантику як агента іншого рівня розглянемо пізніше. На рис. 1 наведено схему, яка ілюструє цю семантику взаємодії агентів.

Оскільки агенти DRIVER та ENTITY поводяться синхронно відповідно до генератора такту, як загальну поведінку розглянемо їхню послідовну композицію поведінок *DRIVER_BEH* та *ENTITY_BEH* у циклі:

$$B0 = DRIVER\_BEH\,;\, ENTITY\_BEH\,;(\,nextTact.B0).$$

Послідовна композиція синхронізована дією *nextTact* для відправлення сигналів згідно з часовими тактами агента TIMER. Водночас передбачено, що один і той самий сигнал не може двічі змінитися впродовж мінімального такту. Рівняння визначає цикл за часовими тактами.

Тоді поширення сигналів до паралельних операторів від агента ENTITY представимо у вигляді такого рівняння поведінки *ENTITY_BEH*:

$$ENTITY\_BEH = (receive(A1).SEND\_ALL(A1).\ ENTITY\_BEH + receive(A2).\ SEND\_ALL\ (A2).\ ENTITY\_BEH + \ldots + no\_receive.$$

У цьому циклі будуть сприйматися тільки ті сигнали, що було відправлені агентом DRIVER, та доти, доки не будуть отримані всі ці сигнали. Параметризована поведінка $SEND\_ALL(x)$ відправляє всім паралельним конструкціям архітектури сигнал X та виконує відповідну поведінку. В загальному вигляді її визначають для кожної паралельної конструкції у такий спосіб:

$$SEND\_ALL(x) = \{(send\_in\_List\_B1(x).B1 + !send\_in\_List\_B1(x)) \,||\, (send\_in\_List\_B2(x).B2 + !send\_in\_List\_B2(x)) \,||\, \ldots\},$$

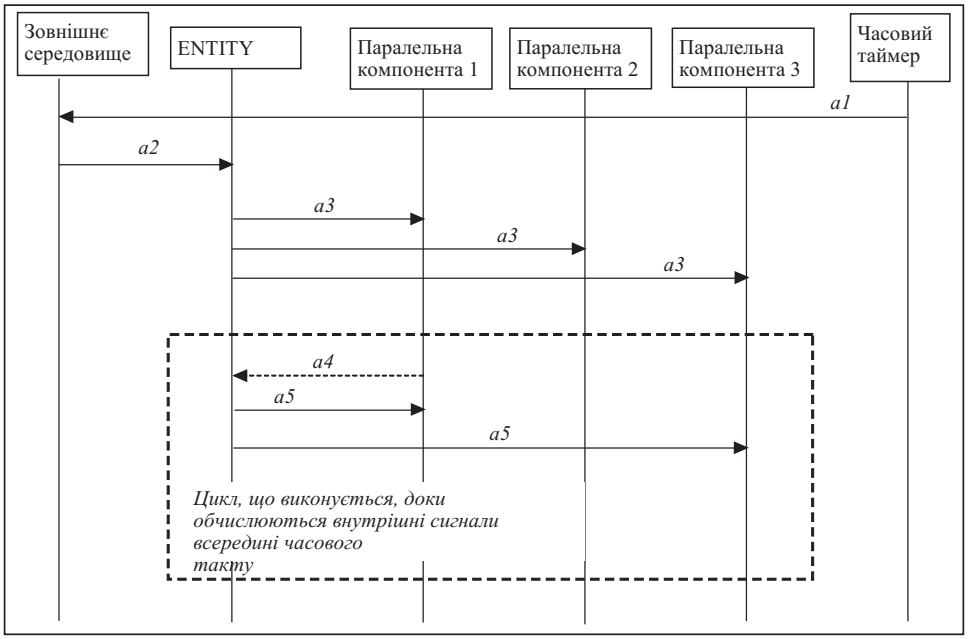

Рис. 2. Семантика отримання та відправлення сигналів агентами VHDL-програми: *a1* — сигнал синхронізації таймера; *a2* — сигнал від зовнішнього середовища; *a3* — розповсюдження сигналів паралельними компонентами; *a4* — ініціалізація внутрішнього сигналу; *a5* — розповсюдження внутрішнього сигналу

де $B1$, $B2, \ldots$ — відповідні поведінки кожної паралельної конструкції (процесу або оператора), а *send_in_List_B1(x)* — дія, що дає поведінці змогу реалізуватись, якщо сигнал є у списку чутливості.

Формалізуємо виконання паралельного оператора або процесу згідно з відповідною зміною сигналу за допомогою поведінки *PERFORM_STATEMENT (X)*. Відповідно до семантики VHDL приймемо таке правило: доки не буде оброблено сигнал, що був сприйнятий агентом ENTITY, наступний сигнал залишається в черзі:

$$PERFORM\_STATEMENT(X) = statementSequentional(N) + (initiateInternal(X)) \,||\, INTERNAL).\ PERFORM\_STATEMENT + empty.$$

Це поведінкове рівняння визначає послідовне виконання компонент у циклі паралельного оператора, зокрема надсилання внутрішніх сигналів архітектури.

Параметризована поведінка активізації внутрішнього сигналу *initiateInternal(X)* є відправленням у всі відповідні паралельні компоненти архітектури ENTITY. Відправлення вихідних сигналів архітектури входить у послідовні оператори. Згідно із семантикою мови VHDL вважаємо, що в тому разі, коли затримка сигналу не задана в явному вигляді, спрацювання всіх внутрішніх сигналів та відповідне виконання послідовних операторів має відбутися протягом часового такту.

Подібно до поведінки *ENTITY_BEH* визначаємо поведінку *INTERNAL*, в якій здійснюється отримання внутрішніх сигналів $I1$, $I2$, … та спрацьовують відповідні оператори:

$$INTERNAL = (receive(I1).PERFORM\_STATEMENT(I1)\ .\ INTERNAL + receive(I2).$$
$$PERFORM\_STATEMENT(I2)\ .\ INTERNAL + \ldots + no\_receive.$$

Поведінка *INTERNAL* виконується паралельно з ENTITY, оскільки спрацювання паралельних операторів є можливим до закінчення поведінки ENTITY і послідовність їхнього виконання може впливати на результат.

Поведінка *INTERNAL* повторюється доти, доки не будуть отримані всі внутрішні сигнали. Після цього знову повертаємося до поведінки *ENTITY_BEH*, а саме спрацювання наступного вхідного сигналу відповідно до черги від агента DRIVER.

На рис. 2 наведено діаграму MSC, яка ілюструє семантику отримання та відправлення сигналів агентами VHDL.

БАГАТОРІВНЕВІ СЕРЕДОВИЩА У ПРОГРАМІ VHDL

Як зазначено вище, VHDL-програма є мережею паралельних компонент, що взаємодіють між собою. Ця мережа може бути багаторівневою, тобто у контексті інсерційного моделювання деякі агенти можуть бути відповідно середовищами для взаємодії агентів іншого рівня. До мережі верхнього рівня входять агенти ENTITY, які взаємодіють відповідно до з'єднань, що визначаються оператором CONFIGURATION. Кожен із цих агентів можна розглядати, як середовище для паралельних компонент, що входять до його архітектури, включаючи паралельну компоненту BLOCK.

Розглянем семантику взаємодії мережі агентів ENTITY. Вони взаємодіють один з одним відповідно до зміни сигналу від зовнішнього середовища DRIVER. Семантика агента DRIVER залишається такою самою, навіть якщо сигнали можуть змінюватися на різних портах агентів ENTITY. Нехай певна порція сигналів змінилася на різних портах різних ENTITY. Тоді поведінку мережі можна описати такою системою поведінкових рівнянь:

$$NETWORK = ENTITY\_BEH\_1 \,||\, ENTITY\_BEH\_2 \,||\, \ldots\ ,$$

$$ENTITY\_BEH\_1 = receive(A11).ENT\_COMP\_BEH\_1;\ ENTITY\_BEH\_1 + receive(A12).ENT\_COMP\_BEH\_1;\ ENTITY\_BEH\_1 + \ldots + no\_receive,$$

$$ENTITY\_BEH\_2 = receive(A21)\ .ENT\_COMP\_BEH\_2;\ ENTITY\_BEH\_2 + receive(A22).ENT\_COMP\_BEH\_2;\ ENTITY\_BEH\_2 + \ldots + no\_receive,$$

де *Ai1*, *Ai2*, … — вхідні сигнали ENTITY *i*; *ENT_COMP_BEH_i* — виконання відповідних паралельних компонент в ENTITY *i* згідно з наведеною вище семантикою. Варто розглянути ще одну важливу структурну сутність VHDL — оператор BLOCK, який теж вважаємо агентом. Агент BLOCK може бути як середовищем для інших паралельних компонент, так і середовищем для окремого паралельного оператора. Отже, він працює за семантикою багаторівневих середовищ.

ТИПИ ДАНИХ ТА ПОСЛІДОВНІ ОПЕРАТОРИ

Значення сигналу у VHDL змінюється відповідно до його оголошеного типу. До того ж, у програмі логіка виконання послідовних операторів також ґрунтується на відповідних типах даних та операціях. У VHDL є скалярні типи, що за багатьма характеристиками відповідають традиційним типам даним — цілочисельні знакові типи (SIGNED, INTEGER), перелічувальний або цілочисельний беззнаковий тип (UNSIGNED), чисельний тип із рухомою комою (REAL). Ці типи в інсерційному моделюванні мають відповідні реалізації. До того ж,

встановлено спеціальні визначені типи даних, як-от фізичні, що являють собою виміри деякої фізичної величини, наприклад часу (TIME). Будь-яке значення фізичного типу є числом, кратним одиниці вимірювання для цього типу.

Іншим різновидом типів у VHDL є композитні типи даних, як-от масиви (ARRAY), які в інсерційному моделюванні представляють функціональними символами. Інший композитний тип даний, наприклад RECORD, можна представити як окремого агента з відповідними атрибутами.

Кожен тип даних має свою реалізацію базових операцій — присвоювання, арифметичні та булеві операції та відповідні предикати. Всі інші типи даних у VHDL можна звести до комбінацій наявних у системі інсерційного моделювання операцій та функціональних символів. Наприклад, тип даних доступу (ACCESS) являє собою вказівники до адреси пам'яті. В інсерційній моделі їх можна представити як цілочисельні.

Кожна паралельна компонента, крім надсилання сигналів, може виконувати набір послідовних операторів, які можна представити за допомогою мови дій в алгебрі поведінок. Розглянемо основні конструкції.

Оператор WAIT дає змогу виконати паралельну компоненту під час отримання сигналу зі списку чутливості або за деякої умови, що визначається отриманими сигналами. Цю конструкцію реалізовано в поведінкових рівняннях шляхом комбінування дій *receive(X)* та дією, що містить необхідну умову продовження виконання паралельної компоненти.

Імперативні оператори потоку керування — це оператори циклу всередині паралельної компоненти, умовні оператори, виклики процедур та повернення з неї, case-оператор, зупинка програми. Всі вони реалізуються передумовою, що дозволяє дію, та операцією «+» алгебри поведінок.

Оператори присвоювання сигналам деякої величини, яких є декілька різновидів, наприклад, умовні та безумовні присвоювання, присвоювання для вибіркових сигналів. Сигнали реалізуються за допомогою відповідної передумови та дії *send(X)* з використанням альтернативи «+».

Інші оператори мови VHDL також зводяться до представлення у вигляді мови дій та з використанням відповідного поведінкового виразу. Наведений нижче приклад представлення VHDL-програми інсерційною моделлю ілюструє семантику розглянутих конструкцій та операторів.

ТРАНСЛЯЦІЯ VHDL-ПРОГРАМИ В ІНСЕРЦІЙНУ МОДЕЛЬ

Розглянемо первинний код VHDL-програми, що реалізує роботу контролера ліфта у триповерховому будинку (рис. 3). Вхідними є сигнали, що визначають стан кнопки виклику ліфта, кнопки натискання поверху та кнопки початку руху кабіни. Крім того, на контролер надходять сигнали датчиків, що вказують на положення кабіни в поточний момент, та сигнал такту, впродовж якого програма має зреагувати. Процес спрацьовує під сигнал такту навіть тоді, коли значення вхідних сигналів не змінилися.

Програма має дві паралельні конструкції мови VHDL, а саме паралельний оператор присвоювання та процес, які змінюють внутрішні сигнали та видають відповідний вихідний сигнал, що визначає рух кабіни.

Сигнали в інсерційній моделі визначаються як булеві змінні. Вхідні сигнали мають надходити згідно з коректним сценарієм поведінки кнопок та датчиків. Поведінка контролера ліфта визначається поведінковими рівняннями згідно із семантикою потоку керування та деревом виконання умовних операторів, у яких обчислюються значення вихідних сигналів керування кабіною.

```
1  entity lift_controller is
2   port(
3   i_start_move_button : in bit := '0'; i_clk : in bit := '0';
4   i_lift_call_button : in bit := '0'; i_door_open : in bit := '0';
5   i_curr_floor_1 : in bit := '1'; i_curr_floor_2 : in bit := '0';
    i_curr_floor_3 : in bit := '0';
6   i_floor_1 : in bit := '1'; i_floor_2 : in bit := '0'; i_floor_3 : in bit := '0';
7   o_down : out bit := '0'; o_up : out bit := '0'; o_open_door : out bit := '0';
8   o_close_door : out bit := '0'; o_nomove : out bit := '1'
9   );
10 end lift_controller;
11
12 architecture arch of lift_controller is
13  signal automative_action : bit := '0';
14  signal floors_equal : bit := '0';
15 begin
16
17 floors_equal <= (i_curr_floor_1 and i_floor_1 and not(i_curr_floor_2 or i_curr_floor_3 or
   i_floor_2 or i_floor_3)) or (i_curr_floor_2 and i_floor_2 and not(i_curr_floor_1 or
   i_curr_floor_3 or i_floor_1 or i_floor_3)) or (i_curr_floor_3 and i_floor_3 and
   not(i_curr_floor_1 or i_curr_floor_2 or i_floor_1 or i_floor_2));
18
19 process(i_clk)is
20 begin
21  if (i_clk'event and i_clk='1') then
22   if (i_lift_call_button or i_start_move_button or automative_action) then
23    automative_action <= '0';
24    if (i_door_open and floors_equal) then
25     o_down <= '0'; o_up <= '0'; o_nomove <= '1'; o_open_door <= '0'; o_close_door <= '0';
26    elsif (not(i_door_open) and floors_equal) then
27     o_down <= '0'; o_up <= '0'; o_nomove <= '1'; o_open_door <= '1'; o_close_door <= '0';
28    elsif (i_door_open and not(floors_equal)) then
29     o_down <= '0'; o_up <= '0'; o_nomove <= '1'; o_open_door <= '0'; o_close_door <= '1';
30    elsif (not(i_door_open) and not(floors_equal)) then
31     o_down <= (i_curr_floor_2 and i_floor_1 and not(i_curr_floor_1 or i_curr_floor_3 or
        i_floor_2 or i_floor_3)) or (i_curr_floor_3 and i_floor_1 and not(i_curr_floor_1
        or i_curr_floor_2 or i_floor_2 or i_floor_3)) or (i_curr_floor_3 and i_floor_2 and
        not(i_curr_floor_1 or i_curr_floor_2 or i_floor_1 or i_floor_3));
32     o_up <= (i_curr_floor_1 and i_floor_2 and not(i_curr_floor_2 or i_curr_floor_3 or
        i_floor_1 or i_floor_3)) or (i_curr_floor_1 and i_floor_3 and not(i_curr_floor_2 or
        i_curr_floor_3 or i_floor_1 or i_floor_2)) or (i_curr_floor_2 and i_floor_3 and
        not(i_curr_floor_1 or i_curr_floor_3 or i_floor_1 or i_floor_2));
33     o_nomove <= '0'; o_open_door <= '0'; o_close_door <= '0';
34    else
35     o_down <= '0'; o_up <= '0'; o_nomove <= '1'; o_open_door <= '1'; o_close_door <= '0';
36    end  if;
37   elsif floors_equal = '1' then
38    automative_action <= '1';
39   end if;
40  end if;
41 end process;
42 end arch;
```

Рис. 3 Первинний код контролера ліфта

У поведінкових рівняннях з'являються параметризовані поведінки та дії, які залежать від деякого сигналу:

$$ENTITY\_BEH = receive(i\_cur\_floor\_1).SEND\_ALL(i\_cur\_floor\_1).ENTITY\_BEH +$$
$$receive(i\_cur\_floor\_2).SEND\_ALL(i\_cur\_floor\_2).ENTITY\_BEH +$$
$$receive(i\_cur\_floor\_3).SEND\_ALL(i\_cur\_floor\_3).ENTITY\_BEH + receive(i\_floor\_1).$$
$$SEND\_ALL(i\_floor\_1).ENTITY\_BEH + receive(i\_floor\_2).$$

```
receive(Y) = 1 -> ”ENTITY#lift_controller: in Y from DRIVER#env;” Y = 1,
send_in_List_B1(x) = list_B1(x)->”” 1,
send_in_List_B2(x) = list_B2(x)->”” 1,
a = (i_clk) -> “” 1,
a0 = 1 -> “” floors_equal = (i_curr_floor_1 && i_floor_1 && !(i_curr_floor_2 ||
i_curr_floor_3 || i_floor_2 || i_floor_3)) || (i_curr_floor_2 && i_floor_2 &&
!(i_curr_floor_1 || i_curr_floor_3 || i_floor_1 || i_floor_3)) || (i_curr_floor_3 &&
i_floor_3 && !(i_curr_floor_1 || i_curr_floor_2 || i_floor_1 or i_floor_2));
a1 = (i_lift_call_button || i_start_move_button || automative_action) -> “”
(automative_action = 0),
a2 = (i_door_open && floors_equal) -> “” (o_down = 0; o_up = 0; o_nomove = 1;
o_open_door = 0; o_close_door = 0),
a3 = (!(i_door_open) && floors_equal) -> “” (o_down = 0; o_up = 0; o_nomove = 1;
o_open_door = 1; o_close_door = 0),
a4 = (i_door_open && !(floors_equal)) -> “” o_down = 0; o_up = 0; o_nomove = 1;
o_open_door = 0; o_close_door = 1,
a5 = (!(i_door_open) && !(floors_equal)) -> “” o_down = (i_curr_floor_2 && i_floor_1 &&
!(i_curr_floor_1 || i_curr_floor_3 || i_floor_2 || i_floor_3)) || (i_curr_floor_3 &&
i_floor_1 && !(i_curr_floor_1 || i_curr_floor_2 || i_floor_2 || i_floor_3)) ||
(i_curr_floor_3 && i_floor_2 && !(i_curr_floor_1 || i_curr_floor_2 || i_floor_1 || i_floor_3));
o_up = (i_curr_floor_1 && i_floor_2 && !(i_curr_floor_2 || i_curr_floor_3 || i_floor_1 ||
i_floor_3)) || (i_curr_floor_1 && i_floor_3 && !(i_curr_floor_2 || i_curr_floor_3 ||
i_floor_1 || i_floor_2)) || (i_curr_floor_2 && i_floor_3 && !(i_curr_floor_1 ||
i_curr_floor_3 || i_floor_1 || i_floor_2)); o_nomove = 0; o_open_door = 0; o_close_door = 0,
a6 = (!(!(i_door_open) && !(floors_equal))) -> “” (o_down = 0; o_up = 0; o_nomove = 1;
o_open_door = 1; o_close_door = 0),
a7 = (floors_equal = 1) -> “” automative_action = 1
```

Рис. 4. Дії контролера ліфта в термінах алгебри поведінок

$$SEND\_ALL(i\_floor\_2).ENTITY\_BEH +$$

$$receive(i\_cur\_floor\_1).SEND\_ALL(i\_floor\_3).ENTITY\_BEH +$$

$$receive(cur\_clk).SEND\_ALL(cur\_clk).ENTITY\_BEH,$$

$$SEND\_ALL(x) = \{(send\_in\_List\_B1(x).B1 + !send\_in\_List\_B1(x)) \;||$$
$$(send\_in\_List\_B2(x).B2 + send\_in\_List\_B2(x))\},$$

$$B1 = a0,$$

$$B2 = a.B3 + !a,$$
$$B3 = a1.B4 \; + \; !a1.B5,$$
$$B4 = a2 + !a2.B6,$$
$$B6 = a3 + !a3.B7,$$
$$B7 = a4 + !a4.B8,$$
$$B8 = a5 \; + \; a6,$$
$$B5 = a7 + !a7.$$

Поведінка *ENTITY_BEH* визначає послідовне приймання зміни сигналів від зовнішнього середовища, що надходить від агента DRIVER, який не розглядається у формалізації. Параметризована поведінка *SEND_ALL(X)* надсилає всі зміни сигналу X у паралельні конструкції архітектури, які далі спрацьовують паралельно. Поведінка *B1* визначає поведінку паралельного оператора, а поведінка *B2* — дерево послідовних операторів процесу.

Семантика дій послідовних операторів процесу та тих, що дають змогу активізувати поведінку в поведінкових рівняннях, відповідає семантиці умов та присвоювань, яка легко транслюється у відповідні специфікації дій. Предикати

list_B1 та *list_B2* визначають, чи знаходиться отриманий сигнал у списку чутливості паралельної конструкції. Всі дії в термінах алгебри дій представлено на рис. 4.

Зауважимо, що ілюстрацію процесною компонентою використано тільки в дії *receive(Y)* за допомогою оператора прийому сигналу в мові MSC.

Трансляція з мови VHDL здійснюється автоматично. Для цього розроблено відповідний транслятор на деякій підмножині мови VHDL.

ПЕРЕГОНИ СИГНАЛІВ

Головною властивістю моделі VHDL-програми в термінах алгебри поведінок є можливість використання формальних алгебраїчних методів, що дають змогу реалізувати задачі верифікації та тестування. Однією із задач є перевірка на наявність перегонів сигналів. Її потрібно виконати, щоб перевірити коректність електронного дизайну.

Проблема полягає в тому, що для всіх паралельних компонент у VHDL-програмі порядок надходження сигналів у межах одного й того самого часового такту не має впливати на результат виконання програми. Аналогічні проблеми описано в літературі, зокрема проблему доступу до однієї й тієї самої пам'яті в багатоядерних системах. Наприклад, це стосується програм керування автомобілем, де отримання сигналів у різному порядку може спричинити колізію виконання завдань та призвести до непередбачених результатів.

Проблему пошуку можливих явищ перегонів детально розглянуто в літературі. Зокрема у роботі [12] досліджено використання методу перевірки моделей та абстрактної інтерпретації. У [13] застосовано методи евристики для виявлення потенційних місць перегонів. У роботі [14] зроблено огляд більш ранніх, але досить ефективних методів виявлення.

У наведеному прикладі колізія перегонів сигналів не може відбутися, бо список чутливості в паралельних конструкціях є різним та сигнали сприймаються від зовнішнього середовища послідовно. Отже, недетермінізму у виконанні конструкцій немає. Для паралельних конструкцій, що містять спільні сигнали, як вхідні так і внутрішні, потрібно виконати відповідну верифікацію на властивість переставності.

Властивість переставності детально розглянуто у [15]. Дві дії є переставними, якщо після будь-якої послідовності їхнього виконання стани середовища будуть еквівалентні. Переставність буває статична та динамічна. Відповідно були створені методи визначення цієї властивості. Якщо дві дії є інформаційно незалежними (або використовують різні атрибути), то в цьому випадку маємо статичну переставність.

Для визначення динамічної переставності необхідно виконувати символьне моделювання та доводити еквівалентність станів відносно відповідних залежностей атрибутів. У цьому випадку властивість переставності поширюється на поведінки. Зокрема в наведеній VHDL-програмі слід перевірити на переставність поведінку паралельного оператора присвоювання *B1* та процесу *B2*.

ВЕРИФІКАЦІЯ НА АЛГЕБРАЇЧНІЙ ВІРТУАЛЬНІЙ МАШИНІ

Представлення тексту VHDL у вигляді інсерційної моделі дає широкі можливості для застосування формальних методів, розроблених для специфікацій в алгебрі поведінок. Ці методи реалізовано у так званій «Алгебраїчній Віртуальній Машині» (АВМ) [16], розробленій на основі методів інсерційного моделювання та алгебри поведінок.

Входом в АВМ є інсерційні моделі, побудовані у середовищі програми «Створювач Моделей», де визначаються агенти, їхні атрибути, рівняння поведінок та семантика дій. Модель можна створити шляхом автоматичної трансляції з певної мови в алгебру поведінок. Окрім трансляції VHDL використовуються трансляції з мови машинних інструкцій Intel x86 та програмних мов високого рівня.

У межах АВМ реалізовано методи верифікації властивостей безпеки, якими здійснюють доведення повноти та несуперечливості специфікацій моделі, властивостей безпеки та життєздатності, пошук недетермінізмів та тупиків, а також уразливостей у специфікації. Програми в АВМ постійно оновлюються та додаються нові, що ґрунтуються на формальних методах. У набір програм АВМ також додано програми генерації тестів з використанням методів символьного моделювання.

Машина АВМ працює з різними програмами візуалізації результатів формальних методів. Крім самого факту доведення властивості, задача ілюструється можливою поведінкою, що призводить до порушення властивості. Такі сценарії представляють у вигляді MSC-діаграм та сценаріїв побудови блоків для блокчейн-задач. Для мови VHDL розробляють візуалізатор сигналів, який дасть змогу представити сценарії, що призводять до певної властивості, наприклад, перегонів сигналів.

ВИСНОВКИ

У статті розглянуто проблему інсерційної семантики специфікацій апаратного забезпечення, реалізованого на основі програмовної логіки з використанням мови VHDL. Побудова такої семантики є необхідною для представлення первинного коду мови VHDL у вигляді інсерційної моделі за допомогою алгебри поведінок.

Інсерційно-орієнтоване представлення дає змогу в майбутньому застосовувати формальні методи інсерційного моделювання для верифікації електронних FPGA-проєктів і розроблення так званих диверсних систем [17]. Це вкрай важливо для критичних систем на програмовних платформах, зокрема, для побудови програмно-технічних комплексів та систем безпеки реакторів АЕС, авіаційних систем тощо.

Подальші дослідження доцільно спрямувати на визначення відповідних метрик і кількісних оцінок верифікації, а також зниження трудомісткості побудови тест-кейсів для складних FPGA-проєктів.

СПИСОК ЛІТЕРАТУРИ

1. Kharchenko V., Letychevskyi O., Odarushchenko O., Peschanenko V., Volkov V. Modeling method for development of digital system algorithms based on programmable logic devices. *Cybernetics and Systems Analysis.* Vol. 56, N 5. P. 710–717 (2020). https://doi.org/10.1007/s10559-020-00289-8.
2. Booch G., Rumbaugh J., Jacobson I. Unified Modeling Language User Guide. Boston: Addison-Wesley, 2005, 512 p.
3. Coelho D. The VHDL Handbook. New York: Springer Science & Business Media, 1989. 406 p.
4. System Verilog Tutorial. URL: www.asic-world.com/systemverilog/tutorial.html.
5. VC Formal. URL: https://www.synopsys.com/verification/static-and-formal-verification/vc-formal.html.
6. Vivado Verification. URL: https://www.xilinx.com/products/design-tools/vivado/verification.html.

7. Jasper RTL Apps. (JasperGold platform). URL: https://www.cadence.com/ko_KR/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform.html.
8. Letichevsky A., Letychevskyi O., Peschanenko V. Insertion modeling and its applications. *Computer Science Journal of Moldova.* 2016. Vol. 24, Iss. 3. P. 357–370.
9. Letichevsky A., Gilbert D. Interaction of agents and environments. In: Recent Trends in Algebraic Development Technique. LNCS. Bert D., Choppy C. (Eds.). Berlin; Heidelberg: Springer-Verlag, 2000. Vol. 1827. P. 311–328.
10. Letichevsky A. Algebra of behavior transformations and its applications. In: Structural Theory of Automata, Semigroups, and Universal Algebra, NATO Science Series II: Mathematics, Physics and Chemistry. Kudryavtsev V.B., Rosenberg I.G. (Eds.). Dordrecht: Springer, 2005. Vol. 207. P. 241–272.
11. ITU-T Recommendation, Z.120, Message Sequence Charts (MSC). URL: https://www.itu.int/ rec/T-REC-Z.120.
12. Ehlers T., Nowotka D., Sieweck P. Finding race conditions in real-time code byusing formal software verification. 2014. URL: https://www.researchgate.net/publication/288563365_Finding_ race_conditions_in_real-time_code_by_using_formal_software_verification.
13. Didier J.-Y., Mallem M. A new approach to detect potential race conditions in component-based systems. 2014. URL: https://hal.archives-ouvertes.fr/hal-01024478.
14. Beckman N.E. A survey of methods for preventing race conditions. 2006. URL: http://www.docdatabase.net/more-a-survey-of-methods-for-preventing-race-conditions-212068.html.
15. Letichevsky A., Letychevskyi O., Peschanenko V. An interleaving reduction for reachability checking in symbolic modeling. *Proc. 11th International Conference on ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge Transfer (ICTERI 2015)* (14–16 May 2015, Lviv, Ukraine). Lviv, 2015. P. 338–353. URL: ceur-ws.org/Vol-1356/paper_74.pdf.
16. Letychevskyi O., Peschanenko V., Volkov V. Algebraic virtual machine project. *Proc. 17th International Conference on ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge Transfer (ICTERI 2021)* (28 Sep. – 02 Oct. 2021, Kherson, Ukraine). Kherson, 2021. URL: icteri.org/icteri-2021/proceedings/volume-2.
17. Kharchenko V. Independent Verification and Diversity: Two echelons of cyber physical systems safety and security assurance. *Proc. 2nd Int. Workshop on Information-Communication Technologies & Embedded Systems (ICTES 2020)* (12 November 2020, Mykolaiv, Ukraine). Mykolaiv, 2020. P. 19–29. URL: ceur-ws.org/Vol-2762/invited2.pdf.

O.O. Letychevskyi, O.M. Odarushchenko, V.S. Peschanenko, V.S. Kharchenko, V.V. Moskalets

INSERTION SEMANTICS OF VHDL AS ELECTRONIC DESIGN LANGUGE

Abstract. The paper considers the problem of insertion semantics of hardware specifications, in particular the VHDL language. The creation of semantics is necessary to represent the primary code of the VHDL language in the form of an insertion model using algebra of behaviors. This presentation allows the widespread use of formal methods of insertion modeling to verify electronic designs of safety critical systems. The main constructions of VHDL language and their insertion semantics such as process, architecture, parallel operators are considered. The control flow of the VHDL program is built in the form of behavioral equations. Consecutive operators are represented as actions of behavior algebra. The problem of signal races and methods of its detection through detection of permutability properties is considered.

Keywords: Hardware Description Language, signal races, permutability, symbolic modeling, behavior algebra, insertion model, safety critical system.

Надійшла до редакції 08.10.2021