УДК 519.713

А.С. ШКИЛЬ, Д.И. ЧЕГЛИКОВ, Д.Е. ЗИНЧЕНКО

Харьковский национальный университет радиоэлектроники, Украина

РЕАЛИЗАЦИЯ ПРОЦЕДУР ИМПЛИКАЦИИ НА ГРАФОВОЙ СТРУКТУРЕ

В данной работе были разработаны внутреннее представление и программная модель процедур прямой и обратной импликации на графовых структурах с целью верификации фрагмента VHDL-кода.

верификация, графовая структура, тест, прямая и обратная импликация, VHDL

Введение

В последние годы верификация проектов цифровых систем, описанных на языках описания аппаратуры VHDL или Verilog, становится наиболее актуальной задачей во всем цикле проектирования. Это обусловлено тем, что такие описания в результате синтезируются в цифровые системы на современной элементной базе, отладка которых требует специального оборудования и материальных затрат. С другой стороны, отладив описание на этапе, предшествующем «прошивке» FPGA или CPLD, разработчик может быть практически уверен, что создаваемое устройство заработает так, как он предполагал. Очевидно, что при обнаружении ошибки проще исправить несколько строк кода, чем заново «перепрашивать» весь кристалл.

Термин верификация появился не так давно в связи с бурным развитием языков описания аппаратуры. От диагностики его отличает объект эксперимента. В диагностике объектом является цифровое устройство с возможным физическим дефектом, от которого затем переходят к неисправностям в модели; а при верификационном процессе объектом является описывающий модель код (например, VHDL или Verilog программа), от ошибок проектирования в котором переходят к неисправностям в модели. Каждая ошибка проектирования, так или иначе, приведет к неисправности в схеме реализованной модели, к изменению закона функционирования.

Существуют два подхода верификации проекта: формальная верификация и верификация на основе подаваемых тестов.

Методы, используемые при формальной верификации, пытаются определить степень корректности проекта с помощью математических доказательств, т.е. необходимо доказать идентичность двух моделей: эталонной и рассматриваемой, применяя строго формализованные доказательства. Такой подход позволяет сделать однозначный вывод о корректности модели. Формальная верификация использует все множество состояний, которые может принимать система. Однако спецификация рассматриваемой модели должна обладать точностью и полнотой.

В отличие от формальной верификации, верификация на основе подаваемых тестов имеет дело только с ограниченным набором режимов функционирования модели. Такая верификация старается покрыть ошибки проектирования в проект путем подачи тестовых векторов (детерминированных или псевдослучайных).

Был разработан и получил широкое распространение функциональный подход к построению тестов, основанный на функциональном описании аппаратуры. Он позволяет работать с высокоуровневыми моделями цифровых устройств любой сложности, включая микропроцессорные системы с программным и микропрограммным управлением.

Метод функциональной верификации, рассматриваемый в [1, 3], основан на формировании путей активизации в поведенческой модели устройства. А алгоритмы активизации в свою очередь используют

процедуры прямой и обратной импликации, от корректности реализации которых в большой степени зависят временные характеристики алгоритмов генерации тестов при верификации [4].

Постановка задачи

Как было замечено ранее, существует необходимость в специализированном подходе к функциональной верификации моделей цифровых устройств, представленных на языке описания аппаратуры (ЯОА). Однако с другой стороны, привлекательным выглядит использование уже ранее созданных алгоритмов и методов. Подход, объединяющий эти требования, основан на алгоритмах активизации путей в описаниях на ЯОА [3]. В свою очередь, особенно сложными процедурами алгоритмов активизации являются процедуры прямой и обратной импликации, особенно обратной. От эффективности реализации процедур импликации зависят временные характеристики алгоритмов верификации.

Целью данной работы является программная реализация процедур импликации на графовых структурах для дальнейшего их использования при генерации тестов для функциональной верификации ЯОА проектов. Аналогов такой программной реализации импликации на графовых структурах в отечественных и зарубежных источниках выявлено не было.

Представление модели, описанной на ЯОА, в виде графа

Для построения тестов широко применяется метод декомпозиции устройства на так называемые однородно тестируемые сегменты на этапе его проектирования. Широко применяется метод разделения проектируемого устройства на управляющий (УА) и операционный автоматы (ОА). Существуют достаточно эффективные методы верификации управляющих автоматов, однако тестирование операционных устройств сегодня остается сложной задачей [2]. В данной работе рассматривается метод построения тестов для операционного устройства на этапе формирования его описания на ЯОА [1].

Операционные устройства (ОУ) характеризуются обработкой многоразрядных информационных слов и возможностью выделения в них управляющей части. Структура каждого из ОУ на функционально-блочном уровне описания представлена совокупностью информационных входов, информационных выходов, управляющих входов, выходов осведомительных сигналов, функциональных блоков, таких как регистры, мультиплексоры, сумматоры и др. и их схемой соединения. Типичное представление модели устройства на ЯОА содержит как минимум два типа языковых конструкций: сигналы и операторы. Сигналы делятся на входные, выходные и внутренние. Операторы подразделяются на два основных типа: выполняемые и управляющие.

Исходя из такого представления ОУ на ЯОА, предлагается модель устройства в виде двудольного ориентированного мультиграфа $G = (V, E)$, где V – конечное множество вершин, включающее два подмножества V^a и V^b, а E – множество дуг. Подмножество вершин V^a, которое назовем множеством переменных и сигналов $V = (v_1, v_2, ..., v_n)$, что внешне соответствует множеству переменных, входящих в описание микроопераций каждого функционального блока. Подмножество вершин V^b, которое назовем множеством функций $F = (f_1, f_2, ..., f_m)$, соответствует множеству микроопераций, выполняемых функциональными блоками. Дуги графа соответствуют информационным управляющим одноразрядным и многоразрядным связям между операторами и сигналами. Вершины первого типа в дальнейшем будем называть операндовыми, а вершины второго типа – функциональными или операторными. Каждому операнду ставятся в соответствие метки, определяющие его разрядность, тип и значение, а каждому оператору модели – его тип, определяемый набором операторов ЯОА.

Вершины первого типа разделим на два класса. К первому классу относятся вершины, соответствующие внешним информационным входам и выходам, внутренним сигналам и константам. Ко второму

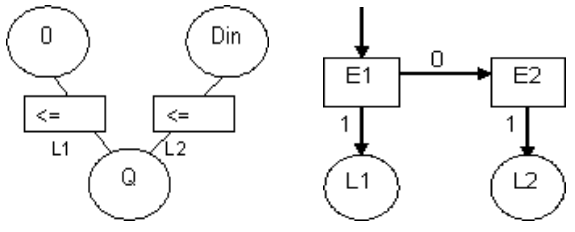
классу относятся управляющие сигналы. Такой способ разделения вершин первого типа позволяет разделить исходный граф модели на подграфы, границами которых являются вершины первого класса. Сигналы, образующие границы подграфа можно разделить на входные и выходные для данного подграфа. Продвижение информации в каждом таком подграфе осуществляется за один такт синхронизации и соответствует одной микрокоманде. Вершины второго типа разделим на два класса. К первому классу относятся вершины, соответствующие выполняемым операторам обработки данных ОУ, а ко второму классу – вершины, соответствующие управляющим операторам. С выходами управляющих операторов всегда связаны управляющие сигналы, которые могут принимать только два значения (0 или 1), и каждый из которых соответствует выполняемому оператору.

В зависимости от присутствующих в подграфах вершин, существует их 2 разновидности: информационный I-граф (операционный автомат для преобразования данных) и управляющий C-граф (для формирования управляющих сигналов). В качестве примера преобразования модели устройства, описанного на ЯОА, в предложенные модели графа рассмотрим модель синхронного D-триггера с асинхронным сбросом, описанную на ЯОА VHDL (листинг 1).

Листинг 1. Описание модели D-триггера на языке VHDL:

```
architecture flipflop of flipflop is begin
process (CLK, RESET) begin
if RESET='1' then DOUT <= '0';
elsif (CLK'event and CLK='1')
then DOUT <= DIN;
end if;  end process; end flipflop;
```

На рис. 1 показан пример представления данной модели в виде вышеописанных графов.



а б

Рис. 1. Реализация I-графа (а) и C-графа (б)

В I-графе сигналы DIN, DOUT и 0, участвующие в выполнении операций, представляются в виде операндовых вершин. Операторы параллельного назначения сигнала языка VHDL представлены как функциональные вершины I-графа. Дуги, связывающие указанные вершины, имеют метки. Например, дуга, представляющая связь между операндовыми вершинами 0 и Q имеет метку L1. Данная метка соответствует управляющему сигналу в C-графе. Сигнал L1 получит значение, равное 1 при выполнении условия E1: RESET = '1'. Если условие не выполняется, по C-графу осуществляется переход к управляющей вершине E2, соответствующей условию (CLK'event and CLK = '1'), при выполнении которого происходит переход к управляющей вершине L2. Из этого следует, что в I-графе будет активизирована дуга, имеющая метку L2, и выполнится операция назначения сигнала Din сигналу Q.

Внутренняя модель для программной реализации

Задача программного представления графа сводится к задаче реализации каждой из его вершин и созданию динамических связей между ними. Отметим, что структура рассматриваемого графа подразумевает, что каждая вершина может иметь ноль, одну и более вершин – предшественников и ноль, одну и более вершин – преемников. Таким образом, каждая вершина должна хранить списки предыдущих и последующих вершин. В предложенной модели имеется два типа вершин: операндовые и функциональные. Операндовые вершины должны характеризоваться такими параметрами как тип операнда, его разрядность, значение, принимаемое операндом. Кроме того, следует отметить, что в VHDL имеются такие объекты как сигналы и переменные, следовательно, операндовые вершины должны нести в себе информацию о том, к какому классу объектов относится данный операнд. Для функциональных вершин основным параметром является идентификатор операции, которую осуществляет данный функционал.

Применительно к программной реализации в среде программирования Visual C++ основным па-

раметром для любой вершины является ее имя – уникальный идентификатор, позволяющий однозначно детерминировать вершину в предложенном графе. Для вершин-операндов и вершин-функций имеется как общий, так и их отличающий набор методов и параметров. Это позволяет создать базовый класс для вершин I-графа, который в качестве своих данных-членов будет содержать имя объекта и списки предыдущих и последующих вершин. Операндовые и функциональные вершины будут реализованы как классы-наследники базового класса, и будут включать в себя не только унаследованные, но и лишь им присущие свойства. Иерархия вершин I-графа представлена на рис. 2.

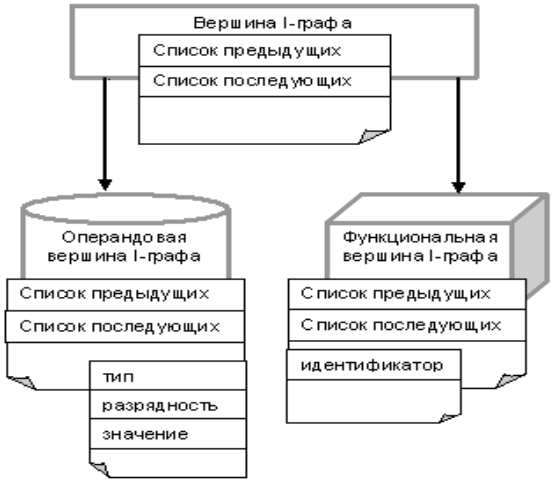


Рис. 2. Иерархия вершин I-графа

Таким образом, I-граф представляется совокупностью объектов классов «Вершина I-графа», «Операндовая вершина I-графа», «Функциональная вершина I-графа» и динамических связей между ними.

Подобная структура I-графа позволяет хранить в объектах графовых деревьев лишь списки начальных вершин (вершин, не имеющих предшественников) и конечных вершин (вершин, не имеющих преемников). Все связи внутри графа представляются неявно в виде данных-членов каждой из вершин. Но, для реализации подобной структуры необходимо выполнение следующего условия: все вершины графа должны иметь уникальные имена.

Если для вершин-операндов данная задача не актуальна, поскольку все сигналы и переменные, присутствующие в VHDL-коде, имеют оригинальные имена, то для вершин-функций существует противоречие, связанное с неоднозначностью создания связей между операндами и операторными вершинами, реализующими одинаковую функцию. Пример неоднозначности задания функциональных вершин представлен на рис. 3, а.

Данный конфликт может быть проиллюстрирован на примере:

d <= a + b; f <= b + e.

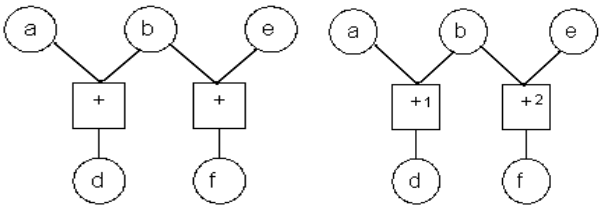


Рис. 3. Функциональные вершины:
а – неоднозначность задания; б – ранжирование

В этом случае I-граф содержит две функциональные вершины «+», одна из которых является преемником вершин «a» и «b», а другая – «e» и «b». При процедуре помещения вершин в I-граф возникнет сложность, связанная с идентификацией вершины. Решением этого конфликта является ранжирование функциональных вершин, то есть назначение функционалу определенного порядкового номера.

Пример ранжирования функциональных вершин графа представлен на рис. 3, б.

Формат представления графа

Графовая структура, заданная для формирования, представляется в виде текстового файла. Задание графа в целом заключается в последовательном перечислении всех его вершин и связей между ними. Для формирования каждой вершины необходимо указать ее параметры, такие как: имя, тип; для операндов – класс VHDL-конструкции (сигнал или переменная), тип данных языка ЯОА, количество разрядов и значение операнда при инициализации. Необходимо также указать все вершины, которые связанны с данной и являются ее предшественниками. Описание форматов представления операндовых и функциональных вершин в файле приводится на рис. 4, 5.

Рис. 4. Формат представления вершины-операнда

Рис. 5. Формат представления вершины-функционала

В том случае, если создаваемая в графе вершина является начальной, поле данных, содержащее список вершин-предшественников, отсутствует.

Разработанная структура данных была реализована посредством использования языка C++ и средств разработки программного обеспечения, предоставляемых программным продуктом Microsoft Visual Studio.NET 2003. Созданная модель графа применима для процедур импликации в соответствии со следующим алгоритмом:

1. На начальные вершины графа, список которых хранит объект класса IGraf, подаются некоторые тестовые воздействия.
2. Для каждой из начальных вершин проверяются все ее вершины-наследники, связанные с ней посредством указателей. В соответствии с идентификатором следующей вершины-функционала выбирается тип операции импликации.
3. Результат импликации записывается в поле значения следующей вершины.
4. Продвижение тестовых воздействий осуществляется до достижения конечных вершин.
5. Управляющие сигналы для всех операций передаются для доопределения в С-граф.

Выводы

Процедура обработки VHDL-кода и задания графовых структур в виде файла «вручную» неэффективна и неудобна. В данной работе была разработана и детально описана структура данных, получаемая из фрагмента VHDL-кода и пригодная для осуществления его тестирования, используя процедуры импликации.

Конечной целью данных разработок является автоматическое формирование графов из фрагментов VHDL-кода путем использования разборщика кода или парсера, выполнения процедур импликации на синтезированных структурах и получение выходных реакций на тестовые воздействия.

Данное приложение не является самостоятельным программным продуктом и рассматривается как составная утилита более масштабного и значительного проекта, целью которого является возможность проведения верификации фрагментов VHDL-кода средствами вычислительной техники.

Литература

1. Krivulya G., Shkil A., Syrevitch Y., Antipenko O. Verification Tests Generation Features for Microprocessor- based Structures // East-West Design & Test International Workshop. – 2004. – P. 96-103.

2. Ковалев Е.В. Проектирование моделей цифровых автоматов для генерации тестов в среде Active-HDL: Дисс. … канд. техн. наук. – Х., 2000.

3. Kryvulya G., Syrevitch Y., Karasyov A., Chegikov D. Test Generation for VHDL Descriptions Verification // Proceedings of IEEE East – West Design & Test Workshop. – Odessa, Ukraine. – 2005. – P. 191-195.

4. Рустинов В.А., Сыревич Е.Е., Сыревич А.В., Чегликов Д.И. Процедуры импликации на арифметической операциях при синтезе тестов верификации // АСУ и приборы автоматики, Всеукраинский межведомственный н.-т. сборник. – Х. – 2005. – Вып. 130. – С. 4-13.

Поступила в редакцию 24.01.2006

Рецензент: канд. техн. наук, проф. В.Г. Лобода, Харьковский национальный университет радиоэлектроники.