ПОСТРОЕНИЕ МОДЕЛЕЙ ПО ОПИСАНИЯМ, СОГЛАСОВАННЫМ С ПРОЦЕССНЫМ СПОСОБОМ МОДЕЛИРОВАНИЯ

Г.Л. Муравьев, Л.П. Махнист, В.И. Хвещук

Рассмотрен подход к получению процессных описаний систем и построению для них исполнимых текстов моделей применительно к специфике языка VHDL. Приводятся графовые представления процессов, пригодные для генерации исполнимых кодов и организации моделирования. Рассмотрена структура системы обработки описаний

Введение

Сложные технические системы, к числу которых относятся цифровые системы, устройства и их проекты, требуют для моделирования эффективных средств. При реализации их моделей следует ориентироваться на процессный способ описания систем и процессную организацию имитации их параллельно-последовательного функционирования [1].

В работе рассматривается подход к построению имитационных моделей, базирующийся: - на однородной промежуточной процессной модели, получаемой трансформацией произвольно-структурированных исходных описаний системы; - на графовой интерпретации процессных описаний, ориентированной на построение исполнимых моделей; - на преобразовании процессной модели в адекватный в функциональном отношении текст на языке высокого уровня с оптимизирующим транслятором для получения загрузочного кода.

Все вопросы иллюстрируются применительно к специфике языка VHDL (стандарты в области автоматизации проектирования VHDL'93 - ANSI/IEEE Std 1076-1993 и VHDL-AMS - Std 1076.1-1999) [2], который обеспечивает разную детальность и стили описания моделей, включая структурные, потоковые, поведенческие.

Процессные описания

Система описывается на VHDL такими компонентами как, например, *entity*, *architecture* в виде композиции параллельных (*concurrent*) операторов типа *block*, *process*, *procedure_call*, *signal_assignment* и других [2, 3], чувствительных к изменению состояния модели, к изменению сигналов в формирователях сигналов.

Промежуточная модель, как трансформированное исходное описание, представляет собой частный случай описания системы только операторами *process*, имеющими графовое представление, удобное для реализации моделирования.

Процессы (операторы *process)* описываются комбинациями традиционных и специальных последовательных (*sequential*) операторов *<sequential_STATEMENT>::=| <IF_statement> | <LOOP_statement> | <CASE _statement> | <WAIT_statement> | <SIGNAL_ASSIGNMENT_statement>* и другими. Это произвольные процессы и процессы-эквиваленты других

параллельных операторов. Произвольные процессы описываются пользователем и имеют любое число состояний

{ [<LABEL>] : process [(< SENSITIVITY_list>)]
begin
{ <sequential_STATEMENT> }
end process } .

Процессы-эквиваленты имеют одно состояние и основываются на эквивалентных формах параллельных операторов - содержат эквивалентный последовательный оператор и оператор ожидания *wait*

[<LABEL>]: process
begin
<sequential_ STATEMENT>; wait on < SENSITIVITY_list>;
end process .

Последний задает чувствительность исходного параллельного оператора к изменениям значений тех сигналов, которые перечислены в списке его чувствительности – конструкции *<SENSITIVITY_list>*.

У процесса-эквивалента параллельного оператора назначения сигналов (типа *conditional_signal_assignment*, *selected_signal_assignment*) вид конструкции *sequential_STATEMENT* меняется в зависимости от комбинации типа назначаемого сигнала (управляемый, неуправляемый, разрешенный *bus*, *register* или неразрешенный) и режима работы оператора назначения сигнала (защищенный, незащищенный). А к списку *SENSITIVITY_list*, включающему все сигналы из правой части оператора, добавляется неявно объявленный *GUARD*-сигнал. Последний отображает значение охранного выражения *GUARD_expression*, определяющего условия срабатывания защищенного оператора. Контроль изменения *GUARD*-сигнала выполняет атрибутивная функция *STABLE*, возвращающая значение “истина” в тот момент, когда сигнал меняется. Соответственно процесс-эквивалент для защищенного оператора назначения управляемого сигнала представляется

[<LABEL>]: process
begin ...
if (GUARD = true) then выполнить эквиваленты
{<SIGNAL_ASSIGMENT _statement>}
elseif (not GUARD'STABLE) then отключить управляемые
{<SIGNAL_ASSIGMENT_statement>}
end if;
wait on GUARD, < SENSITIVITY_list> ;
end process.

Исполнимые процессные описания

Получение модельных описаний, обеспечивающих проведение моделирования, реализуется путем преобразования процессного описания в конструкции языка программирования по заранее установленным правилам, то есть путем генерации исполнимых описаний операторов *process*.

Базируется на графовой интерпретации процессного VHDL-описания, содержащего традиционные операторы последовательных алгоритмов (в т.ч. предикатные *p*-операторы управления логикой алгоритма), *a*–операторы назначения сигналов и *w*-операторы ожидания. В [3] показано, что процессная модель может быть представлена орграфом $Gм = \{ X, U \}$, где X – множество узлов, отображающих процессы (функции и подфункции системы), а U – множество дуг, отображающих пути прохождения сигналов между процессами. Дуги размечаются сигналами с задержками на пути их передачи от процесса к процессу.

В свою очередь процессы можно интерпретировать как ориентированные мультиграфы $\{ Gi \}$ с параллельными дугами, петлями и без изолированных узлов. Операторы *w*-типа, входящие в состав процесса, образуют в его алгоритме ждущие вершины и влияют на состояние процесса [4]. С каждым *j*-м *w*-оператором связывается одно состояние *i*-го процесса *Si,j*, а число *w*-операторов определяет общее количество состояний. Допустимые переходы процесса из состояния в состояние определяются в соответствии со схемой алгоритма процесса (т.е. набором предикатных узлов, параметрами *p*-операторов), а условия перехода (сохранения текущего *Si,j* состояния) задаются параметрами соответствующего *j*-го *w*-оператора.

Соответственно процесс *i* представляется графом $Gi = \{ Xi, Ui \}$, где Xi – множество узлов, отображающих возможные состояния процесса, а Ui – множество направленных дуг, отображающих переходы (направления развития процесса). Для отображения начального состояния процесса в граф вводится дополнительный фиктивный узел (состояние *Si,o*), с которого и осуществляется запуск процесса. А для отображения специфики работы процессов (цикличности исполнения после начального запуска) от узла графа, соответствующего конечному состоянию процесса *Si,n*, добавляется дуга к узлу, соответствующему его начальному состоянию. Дуги размечаются с учетом параметров соответствующего *w*-оператора, задающим условия перехода от одного узла графа к другому. Процесс может быть представлен на языке программирования (например, на С) адекватной функцией

process_<имя_процесса> (void)
{<описание_управления_процессом> <операторная_часть_процесса>} ,

где сегмент *<описание_управления_процессом>* определяет выбор точки входа в операторную часть процесса в зависимости от его состояния, а сегмент *<операторная_часть_процесса>* реализует алгоритм обработки данных. Например, процессы с *GUARD*-выражением могут быть описаны как

```
process_<имя_процесса> ( )
{  ...  if ( State [i] = = 1)  goto M1;
M0:  <операторная_часть_процесса>  State [i] = 1;  return;
M1:  if  ( GUARD_Stable ( ) && LIST_Stable (...) )  return;  goto M0;
} .
```

Здесь *State[i]* служит для запоминания текущего состояния *i*-го процесса, а функции *GUARD_Stable()*, *LIST_Stable()* возвращают значение "истина" при изменении сигналов, к которым процесс чувствителен. Функции процессов вызываются из подсистемы, реализующей алгоритм управления моделью, а после выполнения функциональных преобразований данных, фиксации нового состояния автоматически возвращают управление.

Таким образом, процессы задаются типами, множествами состояний и условиями нахождения в них, охранными выражениями, графами алгоритмов. Состояние модели определяется: - состояниями составляющих ее процессов; - состояниями процессов, заблокированных на время; - состояниями формирователей сигналов. Основные события - изменения сигналов в формирователях, разблокирование процессов. Следующее состояние модели вычисляется после обработки ближайших событий, состоящей в попытке последовательного запуска каждого из активных процессов, начиная с его текущего состояния.

Средства обработки процессных описаний

Построение моделей выполняется путем анализа информационной базы - процессной модели системы, описанной на языке VHDL, и ее обработки по заранее установленным правилам [5]. Информационная база описаний представляет собой набор взаимосвязанных структур данных (например, типа *struct* языка С), отображающих компоненты, элементы компонентов системы в терминах проекта VHDL.

Основу базы образуют структуры данных, отображающие декларативную информацию проекта (описания интерфейсов, сигналов и т.п.), и структуры, отображающие тело архитектуры, блоков в виде составляющих их параллельных и последовательных операторов. Операторы описываются списком структур, где каждый элемент списка (описание отдельного оператора формата *<код_операции> <аргумент> <аргумент> ...*) ссылается на структуры с описаниями выражений и может содержать ссылку на последующий оператор. В качестве аргументов используются ссылки на другие операции выражения либо ссылки на описание используемых в выражении имен объектов, представленных структурами данных, отображающими их декларацию. Само выражение может быть представлено деревом, где листья - структуры, описывающие объекты, участвующие в выражении (имена функций, переменных, сечения и т.д.).

Система обработки может быть организована в виде головной программы и вызываемых ею модулей для инициирования процедур анализа и обработки текстов частей процессной модели. Соответственно система должна включать модули типа ОБРАБОТЧИК_ИНТЕРФЕЙСА, ОБРАБОТЧИК_ТЕЛА_ПРОЕКТА, ОБРАБОТЧИК_ПОДПРОГРАММ и другие. ОБРАБОТЧИК тела проекта базируется на многократном использовании модулей - ГЕНЕРАТОРОВ эквивалентных кодов параллельных и последовательных операторов. Они в свою очередь используют модули - АНАЛИЗАТОРЫ выражений, описанных в операторах.

Поскольку операторы представлены списками, отображающими деревья термов, то соответствующий текст описания исполнимых операторов генерируется модулем АНАЛИЗИРОВАТЬ_ТЕРМ, используемым для анализа дерева термов каждого выражения рекурсивно.

Выводы

Применительно к языку VHDL рассмотрен подход к получению текстов моделей, ориентированных на имитационный расчет, по их процессным описаниям. Рассмотрены исполнимые формы процессных описаний, ориентированные на реализацию средствами языков программирования высокого уровня. Обеспечивается независимость исполнимой модели от формы ее исходного представления.

Приведена структура программно-информационного обеспечения перевода произвольных описаний в процессные с последующей трансформацией в функционально-адекватный текст на языке С. Может использоваться в системах проектирования верхнего уровня в составе проектной процедуры моделирование.

Литература:

1. Максимей И.В. Имитационное моделирование на ЭВМ. - М.: Радио и связь, 1988. – 232 с.
2. Бибило П.Н. Синтез логических схем с использованием языка VHDL. - М.: “СОЛОН-Р”, 2002. – 384 с.
3. Армстронг Дж. Р. Моделирование цифровых систем на языке VHDL. - М.: Мир, 1992. – 175 с.
4. Муравьев Г.Л. Шуть В.Н. Интерпретация VHDL-описаний, согласованная с процессным способом моделирования // Вестник БГТУ. Физика, математика, информатика. – Брест: БГТУ, 2005. – № 5(35) – С. 81-84.
5. Прихожий А.А., Муравьев Г.Л. Система проектирования цифровых СБИС с языком VHDL на ПП ЭВМ // Труды международного симпозиума INFO'89 “Разработка и использование ПЭВМ”. - Мн., 1989. - т.2. – С. 136-140.

Муравьев Геннадий Леонидович, профессор кафедры интеллектуальных информационных технологий Брестского государственного технического университета, канд. техн. наук, доцент, mgl0251@mail.ru

Махнист Леонид Петрович, доцент кафедры высшей математики Брестского государственного технического университета, канд. техн. наук, доцент

Хвещук Владимир Иванович, профессор кафедры интеллектуальных информационных технологий Брестского государственного технического университета, канд. техн. наук, доцент, HVI@tut.by