рование параллельного приложения, что позволяет настраивать систему на параллельную работу с различным числом обучающих множеств.

Таким образом, разработанные программные средства и алгоритмы проектирования и обучения нейронных структур позволяют выполнять этапы инициализации, обучения, проектирования базовых архитектур нейронных сетей, предварительной обработки, анализа и прогнозирования динамических процессов в различных областях.

Автор выражает благодарность Белорусскому республиканскому фонду фундаментальных исследований Республики Беларусь за оказанную поддержку при проведении данных научно-исследовательских работ.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. N.H. Packard, J.P. Crutchfield, J.D. Farmer and R.S. Shaw, Geometry from a Time Series, Physical Review Letters 45, 1980, pp.712-716.
2. F. Takens, Detecting strange attractors in turbulence, Lecture Notes in Mathematics, Vol. 898, Springer-Verlag, Berlin, 1980, pp. 366-381; and in Dynamical System in Turbulence, Warlock, 1980, eds. D. Rand and L.S. Young.
3. A.M. Albano, J. Muench, C. Schwartz, A.I. Mees and P.E. Rapp, Syngular-Value Decomposition and the Grassberger-Procaccia Algorithm, Physical Review A 38, 1988, pp. 3017-3026.
4. M. Casdagli, S. Eubank, J.D. Farmer and J. Gibson, State space reconstruction in present of noise, Physica D 51, 1992, pp. 52-98.
5. X. Zeng, R. Eykholt and R.A. Pielke, Estimating the Lyapunov-Exponent Spectrum from shot Time Series of Low Precision, Physical Review Letter 66, 1991, pp. 3229-3232.
6. J. Holzfuss and G. Mayer-Kress, An approach to error estimation in the applications of dimensional algorithms, in Dimensions and Entropies in Chaotic Systems, editor G. Mayer-Kress, Springer-Verlag, New York, 1986, pp. 114-122.
7. M.T. Rosenstein, J.J. Colins, C.J. De Luca, Reconstruction expansion as a geometry-based framework for choosing proper delay time, Physica D 73, 1994, pp. 82-98.
8. A.M. Fraser and H.L. Swinney, Independent coordinates for strange attractor from mutual information, Physical Review A 33, 1986, pp. 1134-1140.
9. V. Golovko, Y. Savitsky, N. Maniakov. Neural Networks for Signal Processing in Measurement Analysis and Industrial Applications: the Case of Chaotic Signal Processing // chapter of NATO book “Neural networks for instrumentation, measurement and related industrial applications”. - Amsterdam: IOS Press, 2003, pp. 119-143.
10. Савицкий Ю.В., Головко В.А. Нейросетевая модель анализа хаотических сигналов на базе функции активации с настраиваемыми параметрами // Вестник Брестского государственного технического университета – физика, математика, информатика, 2004. - №5. – с.32-25.

УДК 681.3

Муравьев Г.Л., Шуть В.Н.

ИНТЕРПРЕТАЦИЯ VHDL-ОПИСАНИЙ, СОГЛАСОВАННАЯ С ПРОЦЕССНЫМ СПОСОБОМ МОДЕЛИРОВАНИЯ

Моделирование сложных технических систем и их проектов является распространенной задачей инженерной практики. К числу сложных систем относятся цифровые системы и их проекты, реализуемые, например, в виде БИС, СБИС.

Одним из инструментов спецификации цифровых систем и их моделирования с учетом временных задержек служит язык проектирования VHDL [1-4] (международные стандарты в области автоматизации проектирования 1076-1993 и 1076.1-1999). Он предлагает изобразительные средства (специфический набор языковых операторов) разной степени детализации для структурного, потокового и поведенческого описания моделей (проектов), в котором устройства задаются сигналами и алгоритмами преобразования входных сигналов в выходные либо составом и структурой аппаратных элементов и характером их взаимодействия.

Модели строятся на базе специальных программных компонентов. Это entity (проект), содержащий описание port сигналов проекта, декларации объектов (переменных, сигналов) и исполнительную часть в виде компонента architecture. Компонент architecture представляется композицией параллельных (concurrent) и последовательных (sequential) операторов. Параллельные операторы: - block (из деклараций объектов и исполнительной части в виде параллельных операторов); - process (из деклараций объектов и исполнительной части в виде последовательных операторов); - операторы типа procedure_call, function_call, assertion и др.; - параллельный оператор назначения сигнала signal_assignment, служащий для изменения его будущих значений. Сигналы передаются и обрабатываются параллельно. Операторы чувствительны к изменению состояния модели, в частности изменениям значений сигналов в соответствующих им формирователях, хранящих упорядоченные во времени списки прошлых и будущих значений сигналов.

Такие модели предназначаются [1-5] для: - оценки корректности (синтаксической, семантической) описания закона функционирования проекта, его верификации на заданных пользователем тестовых наборах; 2) поведенческого моделирования, оценки характеристик проекта, их соответствия критериям оптимизации; 3) генерации тестов для моделирования проекта на детальных уровнях описания; 4) служат основой проведения проектных процедур (кремниевой компиляции).

Здесь рассматриваются варианты преобразования исходных произвольных модельных описаний в однородную промежуточную модель, удобную для дальнейшей реализации на виртуальной машине и ориентированную, с учетом особенностей организации VHDL, на процессный способ организации квазипараллелизма [6]. Такая модель состоит из параллельных операторов одного типа, а именно – process и может быть основой для автоматической генерации функционально-адекватных исходному проекту исполнимых описаний на языках программирования общего назначения с развитыми вычислительными средствами и оптимизирующими трансляторами. Рассматривается интерпретация полученного описания [7-9], обеспечивающая обработку текущих событий, определение новых состояний и прогнозирование будущих событий модели с целью обеспечения реализации ее компью-

***Шуть Василий Николаевич**, к.т.н., доцент каф. «Интеллектуальные информационные технологии» Брестского государственного технического университета.*
Беларусь, БГТУ, 224017, г. Брест, ул. Московская, 267.
***Муравьёв Геннадий Леонидович,** к.т.н., профессор ИСЗ.*

терного моделирования.

Указанные задачи сводятся: - к замене программных компонентов проекта и входящих в него параллельных команд их последовательными эквивалентами; - к графовой интерпретации полученного описания и замене операторов назначения сигналов вызовами функций обработки формирователей.

1. Замена компонента architecture и операторов block

```
block_label_1: BLOCK[ (GUARD_expression) ]
  <декларации>
 begin
     block_label_2: BLOCK [(GUARD_expression)]
     begin …
       {concurrent_STATEMENT }
     end block [block_label_2] …
     {concurrent_ STATEMENT }
 end block [block_label_1] .
```

Выполняется, начиная с самого внешнего оператора. Блок может иметь охранное выражение <GUARD_expression> (ему соответствует неявно объявленный сигнал GUARD логического типа), определяющее его чувствительность к сигналам и условия срабатывания (GUARD = истина).

Охранные выражения внешних блоков переносятся во вложенные блоки. Охранные выражения вложенных блоков соединяются с выражениями внешних блоков конъюнкцией. Процедура повторяется, пока все блоки не будут замещены составляющими их параллельными операторами с переносом в них охранных выражений блоков.

2. Замена параллельных операторов типа procedure_call, function_call, assertion и др. функционально адекватными операторами process (тип E0), состоящими из эквивалентного последовательного оператора и оператора ожидания wait. Последний воспроизводит чувствительность исходного параллельного оператора к внешним событиям, в т.ч. wait on <SENSITIVITY_list> задает чувствительность к изменениям значений в формирователях сигналов. Здесь <SENSITIVITY_list> ::= <signal_name_list> - перечень соответствующих входных и входных-выходных сигналов. Например, оператор [label]: concurrent_procedure_call заменяется на

```
[label]: PROCESS
        begin
           sequential_PROCEDURE_CALLl_statement;
           wait on < SENSITIVITY_list>;
        end process ,
```

а оператор [label]: concurrent_assertion_statement заменяется на

```
[label]: PROCESS
        begin
           sequential_ASSERTION_statement;
           wait on < SENSITIVITY_list>;
         end process.
```

3. Замена параллельных операторов назначения сигналов операторами process (тип E.L). Операторы [label]: signal_assignment типов conditional_signal_assignment или selected_signal_assignment [1,3] описываются содержательно как

```
BLOCK [ <GUARD_expression > ]     ...
  if (not GUARD'STABLE >) then
     if ( GUARD = TRUE) then
       Выполнить все concurrent_SIGNAL_ASSIGMENT_statement
     else
       Отключить все защищенные ([guard])
       concurrent_SIGNAL_ASSIGMENT_statement
        для управляемых (bus или register) сигналов
     endif
  else if (изменились входные сигналы) then
     if ( GUARD = TRUE ) then
          Выполнить все активные
           concurrent_SIGNAL_ASSIGMENT_statement
     endif
  endif    ...
end block.
```

Здесь использована атрибутивная функция – предопределенный атрибут STABLE сигнала S, возвращающая значение истина, если сигнал S в текущем модельном времени изменился. После замены process состоит из оператора назначения сигнала и оператора ожидания wait on, воспроизводящего чувствительность исходного параллельного оператора к специальному списку сигналов <special_ SENSITIVITY_list> (включает все сигналы из правой части параллельного оператора назначения и GUARD-сигнал). Тогда для случая защищенного оператора назначения

```
[label]: PROCESS
    begin
       special_sequential_SIGNAL_ASSIGMENT_statement;
        wait on <special_SENSITIVITY_list>;
    end process.
```

Вид конструкции special_sequential_signal_assignment_statement определяется комбинацией типа назначаемого сигнала - управляемый, разрешенный (bus, register) или неразрешенный, неуправляемый и режима работы оператора назначения сигнала - защищенный (в охраняемом блоке) или незащищенный, что и учитывается при заменах.

Так процессный эквивалент (тип E3) для защищенного оператора и управляемого сигнала представляется как

```
[label]: PROCESS
     begin            ...
         if (GUARD = TRUE) then
            Выполнить все последовательные эквиваленты
             concurrent_SIGNAL_ASSIGMENT_statement
          else if (not GUARD'STABLE ) then
            Отключить все управляемые
            sequential_SIGNAL_ASSIGMENT_statement
            для управляемых сигналов
          endif;
          wait on GUARD, < SENSITIVITY_list> ;
    end process.
```

В частных случаях (когда выполняется константное назначение сигнала или в качестве охранного выражения используется фронт сигнала), охранное выражение обращается в истину только в момент изменения сигнала. Тогда процессный эквивалент

```
[label]: PROCESS
    begin            ...
      if (GUARD = TRUE) then
         Выполнить все последовательные эквиваленты
         concurrent_SIGNAL_ASSIGMENT_statement
      else
         Отключить все управляемые операторы назначения
         сигналов для управляемых сигналов
      endif;
      wait on GUARD;
```

```
end process.
```

Для защищенного оператора, неуправляемого сигнала (тип E2)

```
[label]: PROCESS
        begin ...
            if (GUARD = TRUE) then
                Выполнить все последовательные эквиваленты
                concurrent_SIGNAL_ASSIGMENT_statement
        endif;
        wait on GUARD, < SENSITIVITY_list>;
end process,
```

а для незащищенного оператора, неуправляемого сигнала (тип E1)

```
[label]: PROCESS
        begin
                sequential_SIGNAL_ASSIGMENT_statement;
                wait on < SENSITIVITY_list>;
        end.
```

4. Графовая интерпретация VHDL-описания

```
{ [label]: PROCESS [( < SENSITIVITY_list> )]
begin
        { sequential_STATEMENT }
        { sequential_SIGNAL_ASSIGMENT_statement }
end process },
```

содержащего традиционные операторы последовательных алгоритмов (в т.ч. предикатные p-операторы управления логикой алгоритма), a–операторы назначения сигналов и w-операторы ожидания. После преобразований проект может включать процессы трех типов. Это произвольные A (Arbitrary) процессы (описанные пользователем и содержащие любое число состояний) и процессы E.L (Listed) со списком чувствительности и единственным собственным состоянием. Последние относятся к числу процессов-эквивалентов E (Equivalent) параллельных операторов (E0 – для операторов типа вызов процедуры, утверждение, E1-E3 – для разных вариантов использования параллельных оператора назначения сигнала).

В [3] показано, что процессная модель может быть представлена орграфом Gм = {X, U}, где X – множество узлов, отображающих процессы (функции и подфункции проекта), а U – множество дуг, обозначающих 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. С каждым i-м w-оператором связывается одно состояние процесса Mi, а число операторов определяет их общее количество. Допустимые переходы из состояния Mi в состояние Mj определяются в соответствии со схемой алгоритма процесса (т.е. набором предикатных узлов, параметрами p-операторов), а условия перехода (сохранения текущего состояния) задаются параметрами соответствующего 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 Mo). А для отображения специфики работы процессов (цикличности исполнения после начального запуска) от узла графа, соответствующего конечному состоянию процесса Mn, 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. Запуск процесса осуществляется, начиная с состояния Mo. Процессы E-типа содержат в граф-схеме только два узла (Mo и M1).

При моделировании каждый процесс задается: номером; типом; множеством возможных состояний и условиями ожидания в них; охранным выражением; графом алгоритма. На языке программирования высокого уровня (например, на СИ) процесс может быть представлен функцией

```
process_<имя_процесса> (void)
        <описание_локальных_переменных>
        {
            <описание_управления_процессом>
            <операторная_часть_процесса>
         }.
```

Здесь конструкция <описание_управления_процессом> определяет выбор точки входа в операторную часть процесса в зависимости от его текущего состояния, а

```
<операторная_часть_процесса> :: =

Mo: <эквивалент 1-го оператора процесса>
        . . .
M1: <эквивалент wait_1>
        . . .
Mn: <эквивалент wait_n>
        . . .
goto Mo; .
```

Соответственно процессы типа E, которые содержат GUARD и всегда активны (не блокируются на время), могут быть описаны функцией

```
process_<имя_процесса> (void)
        { …
            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 процесса, а функции GUARD_Stable(), LIST_Stable() возвращают значение “истина” при изменении сигналов, к которым процесс чувствителен. Все переменные, используемые для организации процесса, должны быть объявлены как локальные и статические, а переменные, отображающие текущие значения сигналов как глобальные. Функции, описывающие процессы, легко вызываются из подсистемы, реализующей алгоритм управления моделью, а после выполнения функциональных преобразований и фиксации нового состояния автоматически возвращают управление.

В ходе моделирования фиксируются текущие характеристики процесса: состояние; статус; время и тип блокировки и т.п. В текущем модельном времени процесс может быть активным (исполняемым), заблокированным на время, пассив-

ным (ожидающим изменения сигналов). Следующее состояние модели определяется состояниями составляющих ее процессов, состояниями формирователей сигналов и состоянием списка процессов, заблокированных на время. Вычисляется после обработки ближайшего события (изменения сигналов в формирователях, разблокирования процессов), состоящей в попытке последовательного запуска каждого из активных процессов, начиная с текущего состояния.

Таким образом, результатом выполнения п.п. 1-4 для произвольного модельного описания служит однородная промежуточная модель в терминах языка VHDL, представляющая собой частный случай процессного описания проекта. А графовое представление каждого параллельного оператора process, состоящего из комбинации последовательных операторов, может служить основой для реализации моделирования. При этом процесс моделирования проекта интерпретируется как пошаговое изменение его состояний в соответствии с графами процессов и результатами вычислений в их предикатных узлах. Рассмотрены эквивалентные формы параллельных операторов, графы и варианты реализации процессов, пригодные для генерации исполнимых кодов и организации моделирования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бибило П.Н. Основы языка VHDL. - М.: “СОЛОН-Р”, 2000. – 210 с.
2. Сергиенко А.М. VHDL для проектирования вычислительных устройств. - К.: “Корнейчук”, 2003. – 208 с.
3. Армстронг Дж. Р. Моделирование цифровых систем на языке VHDL. - М.: Мир, 1992. – 175 с.
4. Поляков А.К. Языки VHDL и VERILOG в проектировании цифровой аппаратуры. - М.: “СОЛОН-Пресс”, 2003. – 320 с.
5. Прихожий А.А., Муравьев Г.Л. Система проектирования цифровых СБИС с языком VHDL на ПП ЭВМ. Разработка и использование ПЭВМ// Труды международного симпозиума INFO' 89. - Мн., 1989. - т.2, ч.1. – 6 с.
6. Максимей И.В. Имитационное моделирование на ЭВМ. - М.: Радио и связь, 1988. – 232 с.
7. Прихожий А.А., Муравьев Г.Л. и др. Система автоматизированного проектирования СБИС. Процедуры проектирования. Материалы по математическому обеспечению ЭВМ. - Мн., ИТК АН РБ, 1992. – 98 с.
8. Дудкин А.А., Головко В.А. , Муравьев Г.Л. и др. Алгоритмы и подсистемы автоматизированного логического проектирования цифровых СБИС. Материалы по математическому обеспечению ЭВМ. - Мн., ИТК АН РБ, 1994. – 120 с.
9. Муравьев Г.Л., Мухов С.В. Подход к процессной интерпретации VHDL-моделей// Сборник материалов VII международной научно-методической конференции “Наука и образование в условиях социально-экономической трансформации общества”. – Брест: ИСЗ, 2004. - Ч.2.

УДК 681.324

Савицкий Ю.В.

ПАРАЛЛЕЛЬНАЯ СИСТЕМА ОБРАБОТКИ ХАОТИЧЕСКИХ ПРОЦЕССОВ НА БАЗЕ ПРОТОКОЛА MPI

Развитие фундаментальных и прикладных наук, технологий требует применения все более мощных и эффективных методов и средств обработки информации. В качестве примера можно привести разработку реалистических математических моделей, которые часто оказываются настолько сложными, что не допускают точного аналитического их исследования. Единственная возможность исследования таких моделей, их верификации (то есть подтверждения правильности) и использования для прогноза - компьютерное моделирование, применение методов численного анализа. Другая важная проблема - обработка больших объемов информации в режиме реального времени. Все эти проблемы могут быть решены лишь на достаточно мощной аппаратной базе, с применением эффективных методов программирования [1].

Переход на качественно новый уровень производительности потребовал от разработчиков ЭВМ системных решений, значительная часть которых основана на организации параллельной обработки. Параллелизм все более широко используется как средство, позволяющее повышать производительность вычислительных систем, не прибегая к повышению быстродействия их компонентов. *Основная цель состоит в выборе такого отображения задачи в реализующую ее техническую структуру, при котором параллелизм мог бы использоваться для уменьшения времени решения задачи, не вызывая чрезмерного роста числа избыточных обрабатывающих модулей и стоимости передачи данных* [*1,2*].

Стиль программирования, основанный на параллелизме задач подразумевает, что вычислительная задача разбивается на несколько относительно самостоятельных подзадач и каждый процессор загружается своей собственной подзадачей. Компьютер при этом представляет собой MIMD - машину. Аббревиатура MIMD обозначает в известной классификации архитектур ЭВМ вычислительную систему, выполняющую одновременно множество различных операций над множеством, вообще говоря, различных и разнотипных данных. Для каждой подзадачи пишется своя собственная программа. Чем больше подзадач, тем большее число процессоров (вычислительных модулей, ВМ) можно использовать, тем большей эффективности можно добиться. Важно то, что все эти программы должны обмениваться результатами своей работы. Практически такой обмен осуществляется вызовом процедур специализированной библиотеки. Программист при этом может контролировать распределение данных между процессорами и подзадачами и обмен данными. Очевидно, что в этом случае требуется определенная работа для того, чтобы обеспечить эффективное совместное выполнение различных программ.

Примерами специализированных библиотек являются библиотеки *MPI (Message Passing Interface)* и *PVM (Parallel Virtual Machines)*. Эти библиотеки являются свободно распространяемыми и существуют в исходных кодах.

MPI – это разработанная Аргоннской Национальной Лабораторией (США) большая библиотека функций для передачи сообщений, которая позволяет преобразовать последовательную программу в программу, которая будет *полностью использовать параллельную архитектуру используемой вычислительной системы*. MPI предоставляет программисту единый механизм взаимодействия ветвей внутри параллельного приложения независимо от машинной архитектуры (однопроцессорные / многопроцессорные с общей памятью / раздельной памятью), взаимного расположения ветвей (на одном процессоре (ВМ) / на разных процессорах (ВМ)).

Для MPI необходимо создавать приложения, содержащие код всех вервей сразу. MPI-загрузчиком запускается указываемое количество экземпляров программы. Каждый экземпляр определяет свой порядковый номер в запущенном коллективе, и в зависимости от этого номера и размера коллектива выполняет