Проектирование аналоговых и аналого-цифровых ИС с использованием языка Verilog-AMS

Часть 1. Введение в язык Verilog-A

Дмитрий Осипов (Москва)

В статье определено место языка Verilog-AMS в процессе проектирования аналоговых и аналого-цифровых интегральных схем (ИС). На примере описания RC-цепи приводятся начальные сведения о синтаксисе языка.

Введение

В настоящее время традиционный подход к проектированию СБИС «снизу вверх», включающий необходимый этап моделирования всей системы на транзисторном уровне, не может обеспечивать необходимую скорость разработки [1]. Такой подход [2] предполагает последовательную разработку и верификацию отдельных функциональных блоков СБИС. Разработка каждого блока начинается с задания требований и заканчивается созданием описания транзисторного уровня. Каждый блок верифицируется как отдельный электронный компонент, а не в контексте системы в целом. Затем блоки объединяются в систему и верифицируются вместе. Таким образом, вся система оказывается представленной на транзисторном уровне.

Данный метод, хорошо зарекомендовавший себя при разработке однородных ИС, затрудняет проектирование «систем-на-кристалле» [2]. Наиболее существенными недостатками метода «снизу вверх» являются:

- при объединении блоков в систему моделирование самой системы занимает много времени (до нескольких недель), таким образом, верификация всей системы за разумное время затруднена, а зачастую невозможна [3];
- на ранних этапах разработки невозможно найти трудно исправимые ошибки в архитектуре проекта. К моменту обнаружения ошибки на уровне архитектуры уже созданы отдельные блоки проекта. Расчёт влияния параметров отдельных блоков на систему в целом;
- некоторые этапы разработки должны проводиться последовательно (верификация системы не может быть произведена, пока не готовы все блоки), что увеличивает время разработки;
- для обеспечения работоспособности системы после сборки разработчики отдельных блоков должны согласовывать свои действия между собой. Ошибки соединения блоков, вызванные недопониманием, делают всю систему неработоспособной.

Альтернативная методология проектирования «сверху вниз», свободная от указанных выше недостатков, в последнее десятилетие активно развивается компанией Cadence Design Systems [4, 5] и одним из её партнёров – компанией Designers Guide Consulting [6].

Метод проектирования «сверху вниз»

В общем виде метод проектирования «сверху вниз» состоит в последовательном переходе от системного уровня к транзисторному уровню через пошаговую детализацию отдельных блоков. Каждый из промежуточных уровней полностью разрабатывается и верифицируется перед переходом к следующему. Основными положениями данной методологии являются [7, 8]:

1. Проект ИС должен быть представлен в виде разделяемой (т.е. доступной всем разработчикам) базы данных. Каждый из разработчиков должен иметь возможность проводить моделирование как всего проекта, так и отдельных блоков. При этом блоки могут быть описаны на поведенческом, транзисторном или топологическом уровне;

2. Каждое изменение отдельного блока или архитектуры проекта должно верифицироваться в тестовом окружении всего проекта, отлаженном на предыдущем шаге разработки;

3. Процесс разработки должен начинаться с верхнего поведенческого уровня и двигаться в направлении повышения степени детализации.

Важным элементом всей методологии является единый язык Verilog-AMS [3], позволяющий унифицировать действия, необходимые для разработки и верификации системы на каждом этапе.

Использование языка Verilog-AMS предоставляет ряд преимуществ, особенно в рамках методологии «сверху вниз»:

- при разработке системы, содержащей большую цифровую часть, возможно проведение совместного моделирования только с цифровыми блоками, описанными на Verilog;
- интеграция в едином проекте описаний Verilog-AMS и SPICE подобных списков межсоединений полностью поддерживается известными производителями САПР, такими как Cadence, Mentor Graphics и Synopsys;
- при разработке на архитектурном уровне Verilog-AMS обеспечивает совместимость со средствами, используемыми на последующих этапах разработки, в отличие от таких высокоуровневых языков, как Matlab, и таких средств моделирования, как Simulink;
- верификация ИС с использованием разноуровневых моделей отдельных блоков.

Язык Verilog-A

Язык высокого уровня Verilog-AMS (Analog and Mixed Signal) позволяет проводить совместное моделирование аналоговой и цифровой частей системы и включает в себя:

- Verilog-D (традиционно обозначаемый как Verilog) – язык описания цифровой аппаратуры;
- Verilog-A – язык описания для аналоговых систем (эквивалент Verilog-D);
- расширения обоих языков для смешанного моделирования (см. рис. 1).

В популярных свободных паке-

тах программ для схемотехнического моделирования, например Geda [8] с симуляторами *ngspice* или *gnucap*, или QUCS [9], пока реализована лишь частичная поддержка языка, которая позволяет создавать модели на ограниченном подмножестве Verilog-A с последующим автоматическим переводом в код C с помощью программы ADMS [10]. Для использования полученной таким образом модели необходима совместная компиляция полученного кода C с основным кодом используемого симулятора.

Чтобы опробовать методологию «сверху вниз» не обязательно обладать лицензией на «флагманские» продукты от Cadence или Mentor Graphics. Для моделирования схем, содержащих как модели SPICE, так и описания Verilog-A, могут быть использованы программы Dolphin SMASH [11] или SymicaDE [12]. Последней программой мы воспользуемся ниже.

Несомненным достоинством SymicaDE является возможность моделирования смешанных проектов, где отдельные блоки системы могут быть описаны на любом из языков: Verilog, Verilog-AMS, VHDL, VHDL-AMS, а также списками соединений SPICE. Программа имеет версии для ОС Windows и Linux. Недостатком SymicaDE является малая информативность (если сравнивать с Cadence Spectre) вывода информации об ошибках. Кроме того, во время тестов наблюдались зависания программы, которые, однако, могут быть связаны с тем, что программа была установлена на неподдерживаемой платформе Linux Mint 16. В то время как 32-битная версия SymicaDE работает в различных ОС семейства Linux, 64-битная версия ограничена линейкой Red Hat и её производных дистрибутивов.

Знакомство с языком Verilog-A. Описание резистора и конденсатора

Наиболее общим математическим описанием резистора является $f(u, i) = 0$, где u – падение напряжения на резисторе, i – ток через резистор, f – произвольная функция двух аргументов. Такое описание подходит как нелинейным резисторам, так и другим устройствам (например, транзисторам), выполняющим функцию резистора. Под резистивностью понимают производную напряжения по току.

Простейший линейный резистор описывается выражением $u = r \times i$, где r – сопротивление резистора. Модель Verilog-A резистора, описываемого такой формулой, приведена в листинге 1.

В первой строке листинга 1 приводится описание модуля. Всю строку, начинающуюся с «//», симулятор воспринимает как комментарий. Также для создания комментариев можно использовать символы «/*» и «*/» (аналогично языку C). Текст, ограниченный указанными символами, будет считаться комментарием.

В строке 2 подключается заголовочный файл «discilplines.vams». Для этого используется директива препроцессора *include*, стандартная для многих языков программирования. Данный файл содержит описания «дисциплин» (disciplines) Verilog-AMS. Дисциплина – это набор связанных с ней физических типов сигналов (natures). Например, дисциплина *electrical* предназначена для описания электрических сигналов и состоит из двух типов сигналов – напряжения и тока.

Язык Verilog-AMS не определяет ни одной дисциплины, поэтому для нормальной работы необходимо подключение файла с описанием различных дисциплин. Кроме *electrical* стандартный файл «disciplines.vams» содержит ещё несколько дисциплин для описания других типов систем, например, механических. С этим связана популярность языка для разработки микроэлектромеханических систем (MEMS). Verilog-AMS позволяет объединять в одном цикле моделирования стандартные списки соединений SPICE или модули Verilog-AMS, в которых описана электрическая часть системы, с описанием её механической части. Более подробно дисциплины Verilog-AMS будут обсуждаться в следующих статьях.

В строке 5 листинга 1 объявляется модуль *resistance*, который является базовым «строительным» блоком Verilog-AMS. Описание модуля начинается с ключевого слова *module* и заканчивается ключевым словом *endmodule*. После слова *module* должно следовать название модуля и список портов, заканчивающийся точкой с запятой «;» (обозначает конец строки). Порты позволяют «подключиться» к модулю Verilog-A. Направление порта задаётся с помощью ключевых слов *inout* – вход-выход (*input* – вход или *output* – выход). Если порт объявлен как вход, то значение типа сигнала, связанного с данным портом, не может быть изменено внутри модуля. Если порт объявлен как выход, то значение типа сигнала, связанного с данным портом, не может быть изменено снаружи модуля. В противном случае (например, если подключить к выходу модуля идеальный источник напряжения или тока), моделирование завершится ошибкой. В листинге 1 порты резистора были объявлены как входы-выходы, т.е. значение любого типа сигнала, связанного с портами резистора, может быть изменено как снаружи, так и внутри модуля. В строке 7 задаётся дисциплина для сигналов, связанных с портами резистора. Все порты модуля обязательно должны иметь связанные с ними типы сигналов.

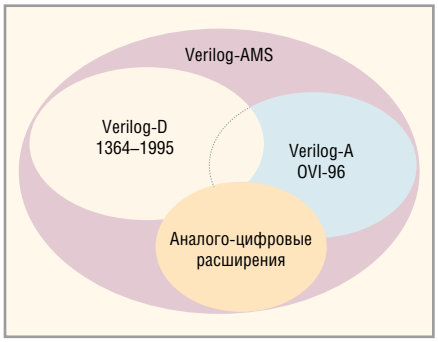

Рис. 1. Семейство языков Verilog

В строке 9 объявляется параметр модуля. В дальнейшем пользователь, заданный при объявлении экземпляра модуля, сможет задать значение этого параметра. Например, в программе графического ввода САПР Cadence IC модуль, написанный на Verilog-AMS, ничем не отличается от других элементов схемы – в одном графическом описании схемы можно использовать как модули Verilog-A, так и стандартные SPICE-модели или макромодели. Симу-

Листинг 1. Описание резистора на Verilog-A

```
01 // Простой резистор
02
03 `include "disciplines.vams"
04
05 module resistance (n1, n2);
06     inout n1, n2;
07     electrical n1,n2;
08
09     parameter real R=1.0 from
[0:inf);
10
11     analog begin
12             V(n1,n2) <+
R*I(n1,n2);
13     end
14 endmodule
```

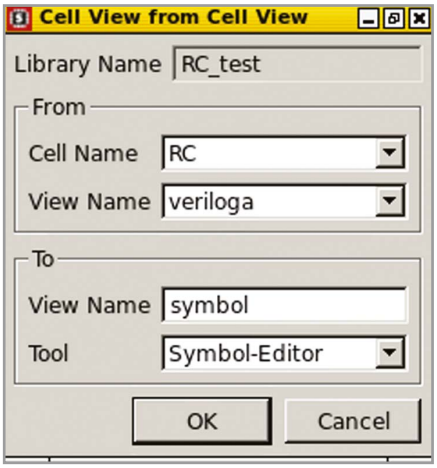

Рис. 2. Создание символа для ячейки RC-цепи

лятор Dolphin SMASH позволяет создавать как SPICE-списки соединений с использованием Verilog-AMS модулей и задавать их параметры, так и Verilog-AMS списки соединений с использованием моделей и макромоделей SPICE.

Кроме объявления имени и типа параметра модуля в строке 9 задаётся значение параметра по умолчанию: если параметр модуля не будет задан, симулятор будет использовать значение по умолчанию. Здесь же задаётся область определения параметра, чтобы предотвратить неправильное использование модуля. Так, если пользователь задаст параметр вне границ области определения, симулятор сообщит ему об ошибке.

Листинг 2. Описание конденсатора на Verilog-A

```
01 // Простой конденсатор
02
03 `include "disciplines.vams"
04
05 module capacitor (n1, n2);
06     inout n1, n2;
07     electrical n1,n2;
08
09     parameter real C=1.0e-12 ;
10
11     analog begin
12             I(n1,n2) <+
C*ddt(V(n1,n2));
13     end
14 endmodule
```

Листинг 3. Описание RC-цепи на Verilog-A

```
`include "disciplines.vams"
module RC (vin, vout, g);
parameter real r=1000;
parameter real c=1e-12;
inout vin, vout,g;
electrical vin,vout,g;
analog begin
V(vin,vout) <+ r * I(vin,vout);
I(vout,g) <+ c*ddt(V(vout,g));
end
endmodule
```

Определителем аналогового модуля на языке Verilog-AMS является ключевое слово *analog*, которое задаёт аналоговый процесс. Последний является обособленной частью моделируемой системы, контролирующей протекающие сигналы. Обычно система состоит из многих процессов. Например, в случае RC-цепи, это процессы резистора и конденсатора. Система, таким образом, представляется в виде независимых и взаимодействующих процессов. Ядро симулятора вычисляет новое состояние аналогового процесса в каждой временно́й точке (частота следования точек во времени задаётся при настройке точности симулятора). В связи с этим внутри аналогового процесса не поддерживаются такие конструкции языка Verilog, как задержки (#) и ожидание (*wait*). Однако внутри аналогового процесса можно задавать события (используя @), которые, впрочем, имеют несколько иной вид по сравнению с событиями в Verilog. Аналоговые события будут рассмотрены позднее.

Для объединения высказываний языка в единый блок применяются ключевые слова *begin* и *end*, полностью аналогичные фигурным скобкам в языке C. В строке 12 листинга 1 содержится основная управляющая конструкция модуля, задающая напряжение между портами $n1$ и $n2$ в зависимости от тока, протекающего между ними.

Описание конденсатора на Verilog-AMS приводится в листинге 2. В общем случае конденсатор может быть задан уравнением $f(u, q) = 0$, где u – падение напряжения между выводами конденсатора, q – заряд на конденсаторе, f – произвольная функция. Простой линейный конденсатор может быть описан как $q = c \times u$, или $i = c \times (du/dt)$, где c – ёмкость конденсатора.

В строке 12 используются аналоговый оператор непрерывной производной по времени dd*t*, который рассчитывает производную аргумента по времени в каждой точке моделирования.

Моделирование простой RC-цепи

Рассмотрим процесс моделирования простой RC-цепи в программе SymicaDE. Интерфейс программы очень похож на интерфейс Cadence Virtuoso, а окно настройки параметров моделирования буквально повторяет аналогичное окно ADE L в Cadence. Также как и в Virtuoso, проект организован по иерархическому принципу в библиотеках и ячейках (*cell*), при этом каждая ячейка (например, RC-цепь) может содержать несколько описаний (список соединений SPICE, Verilog-AMS, Verilog и т.д.).

После установки SymicaDE и указанных в руководстве по установке переменных окружения программа может быть запущена командой «*$ symicade*». В открывшемся окне необходимо создать библиотеку RC_test, а в ней ячейку RC с описанием Verilog-A (см. листинг 3). После этого необходимо создать символ для ячейки RC-цепи. Для этого в окне Library Manager требуется щёлкнуть правой кнопкой мыши (ПКМ) по описанию *veriloga* и в выпадающем меню выбрать New Cell View From (см. рис. 2). Появится интуитивно понятная форма для генерации символа.

Одним из преимуществ использования моделей Verilog-A является то, что для конечного пользователя модель Verilog-A ничем не отличается от модели, встроенной в симулятор, т.к. все параметры модели могут быть заданы без редактирования исходного кода. Чтобы добавить параметры, определённые в коде Verilog-A, следует выделить ячейку RC в Library manager и, щёлкнув ПКМ, выбрать Properties (Свойства). В окне Properties необходимо нажать на кнопку Load from File и выбрать VerilogA (см. рис. 3).

Для создания тестового окружения создаётся ячейка RC_test типа *schematic* (см. рис. 4). Настройки моделирования можно выполнить, открыв из главного меню программы Simulation → Environment. После настройки параметров моделирования (Transient 0,8n), можно выбрать сигналы для отображения на графике Outputs → To be plotted → Select on Schematic. Для запуска моделирования необходимо выбрать Simulation → Netlist and Run. После этого будет создан список соединений (netlist) и запущена программа *adelxm* для разбора кода Verilog-A. Она подготовит исходные коды C++ для модели RC-цепи, используемые в последующей компиляции динамической библиотеки для симулятора.

На этом этапе проявляется один из недостатков систем, использующих

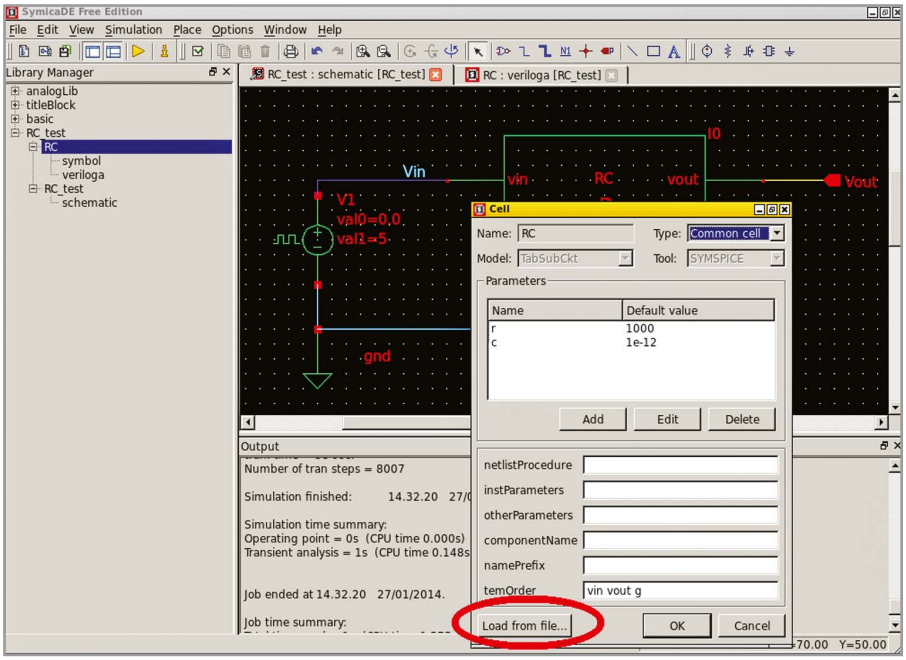

Рис. 3. Добавление параметров модели Verilog-A в символ

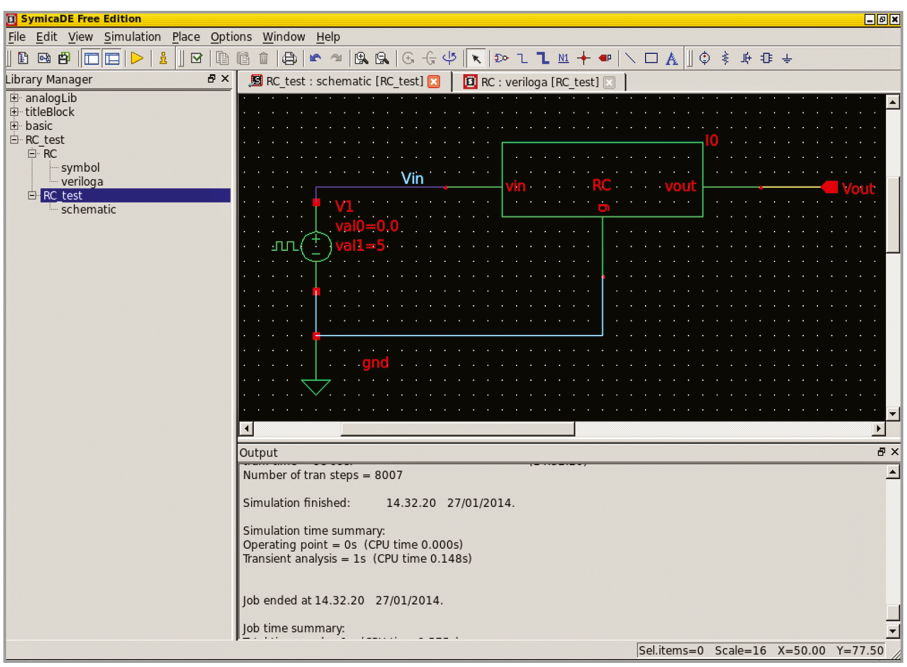

Рис. 4. Тестовое окружение для RC-цепи

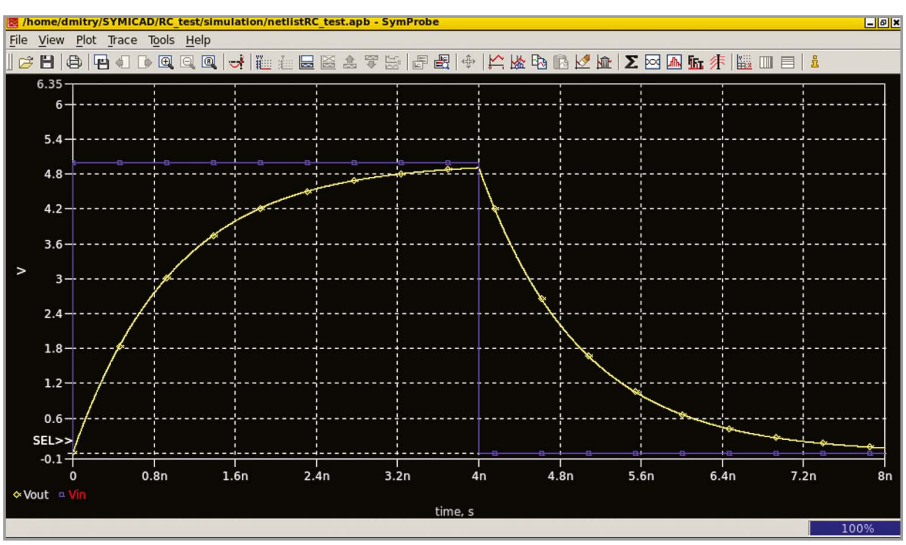

Рис. 5. Результат моделирования переходных процессов

программу *adelxm* для разбора кода: сообщения об ошибках, включая очевидные синтаксические, обычно появляются при попытке компиляции модели, и, следовательно, ссылаются на выходной файл C++ *adelxm*. Зачастую они непонятны инженеру, который разрабатывает модели на Verilog-A.

Результаты анализа переходных процессов в RC-цепи показаны на рисунке 5.

Заключение

В первой статье цикла рассмотрены базовые понятия языка аналогового моделирования Verilog-A и разобран пример моделирования простой RC-цепи в программе SymicaDE.

Литература

1. *Kundert K.* Future directions in mixed-signal behavioral modeling. Proceedings of the 2002 IEEE International Workshop on Behavioral Modeling and Simulation. BMAS. 2002. P.150-183.
2. *Kundert K., Zinke O.* The Designers Guide to Verilog-AMS. Kluwer Academic Publishers. 2004.
3. *Oliver J.A., Prieto R., Cobos J.A., Garcia O.* Hybrid Wiener-Hammerstein Structure for Grey-Box Modeling of DC-DC Converters. Twenty-Fourth Annual IEEE Applied Power Electronics Conference and Exposition. APEC. 2009. P. 280-285.
4. *Zinke O.* Mixed-signal simulation in design, verification. EE Times Asia. 2005. http://www.eetasia.com/ART_8800370732_480100_TA_8b110773.HTM.
5. *Bruggeman J.* Top-down approach brings fresh challenges to the design process. Electronics Weekly. 2010. Issue 2450. P.16. http://www.cadence.com/rl/Resources/articles/ElectronicsWeekly_JohnBruggeman_120110.pdf.
6. *Kyndert K., Chang H.* Verifying mixed-signal designs. EE Times Asia. Design. http://www.eetasia.com/ARTP_8800410568_480100.HTM.
7. *Metroka M., James F.* Top-down approach speeds mixed-signal design. EE Times. http://www.eetimes.com/document.asp?doc_id=1271279.
8. *Осипов Д., Бочаров Ю.* Пакет средств проектирования электронных устройств gEDA. Технологии в электронной промышленности. № 3. 2011.
9. http://qucs.sourceforge.net.
10. http://mot-adms.sourceforge.net.
11. http://www.dolphin.fr/medal/smash/smash_acquaintance.php.
12. http://www.symica.com.