Система высокоуровневого проектирования аппаратного обеспечения HLCCAD: тестирование

Мы продолжаем цикл статей [1–3] о средствах автоматизации сквозной совместной разработки программного и аппаратного обеспечения встроенных цифровых систем, разработанных в Гомельском государственном университете. Ранее была описана среда редактирования, симуляции и отладки аппаратного обеспечения HLCCAD [2]. В то же время рамки журнальной статьи не позволили на достаточном уровне детализации объяснить предоставляемые возможности тестирования разработок. Приведенный ниже материал призван ликвидировать пробелы в этом вопросе.

Михаил Долинский,
Вячеслав Литвинов,
Алексей Толкачев,
Алексей Корнейчук

1. Скриптовый подход к автоматизации тестирования

1.1. Состав текстовых языков поддержки тестирования

В системе HLCCAD реализовано интерактивное тестирование проектов, когда разработчик, остановив моделирование, может задать входные воздействия и проследить с помощью симуляции реакцию проекта на них. Кроме того, в системе поддерживается использование высокоуровневых компонентов (программ, написанных на языках типа Object Pascal или C++) для подачи входных воздействий и определения эталонных реакций. Однако наиболее привычным и соответствующим типичной квалификации разработчика аппаратного обеспечения является использование специальных скриптовых языков (наглядных для пользователя текстовых языков, интерпретируемых системой в процессе симуляции) для автоматизации тестирования. В системе HLCCAD реализовано три таких языка:

- **Язык описания содержимого ОЗУ/ПЗУ** используется для определения, по каким адресам ОЗУ/ПЗУ нужно записывать те или иные данные и какие именно. Поддерживаются распространенные стандартные форматы BIN и Intel Hex, а также собственный формат MEM (во многих случаях более удобный для разработчиков ввиду своей большей наглядности и компактности).
- **Язык управления тестовыми воздействиями и эталонными реакциями** (язык тестов) используется (в указанный пользователем момент модельного времени) для выполнения следующих операций: задания входных воздействий на контакты; модификации содержимого ОЗУ/ПЗУ, регистров, триггеров; установки эталонных значений на тех же объектах (контакты, ячейки ОЗУ/ПЗУ, регистры, триггеры). Эталонные значения сверяются в процессе моделирования в нужный момент времени с реальными значениями (получившимися в результате симуляции), результат сравнения, по желанию пользователя, выводится на экран или в файл протокола.
- **Язык управления пакетным тестированием** (язык сценариев) используется для описания работы по организации тестирования, выполняемой обычно пользователями интерактивно: выбор проекта для тестирования, указание нужного набора тестов, протоколирование результатов тестирования, условное исполнение процесса и т. д.

1.2. Универсальная поддержка скриптовых языков автоматизации тестирования

Сегодня можно отметить две важные тенденции в автоматизации тестирования проектов аппаратного обеспечения [4–6]: стремление к использованию скриптовых языков и отсутствие де-юре и де-факто стандарта такого языка.

Поэтому чрезвычайно важно обеспечить гибкость в поддержке подобных языков. В системах IEESD/HLCCAD/WInter обработка текстовых языков обеспечивается с помощью специально разработанного универсального синтаксического анализатора (UniSAn) [7, 8]. UniSAn обеспечивает возможность определить текстовый язык с помощью расширенных формул Бекуса—Наура; имеет средства для анализа и исправления ошибок в таких описаниях; по корректным описаниям строит сжатое автоматное представление описания языка. Затем это автоматное представление используется универсальным анализатором при обработке соответствующих текстов. Уникальной особенностью анализатора являются «функции активации», которые позволяют связывать результаты анализа текста с динамической библиотекой его интерпретации.

Таким образом, с одной стороны, обеспечено естественное и эффективное развитие наших собственных языков автоматизации тестирования, а с другой стороны, имеется высокая степень готовности и повторного использования кода при необходимости поддержать сторонние языки автоматизации тестирования.

Сам универсальный синтаксический анализатор и методику его использования предполагается описать более подробно в отдельной статье.

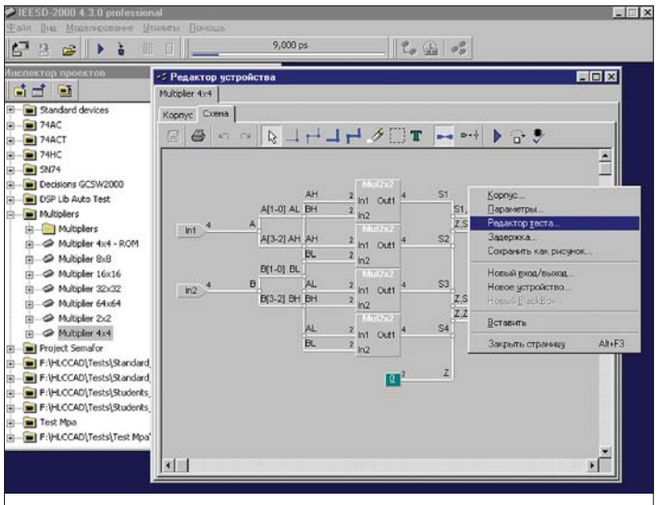

Рис. 1. Выбор пункта локального меню «Редактор теста»

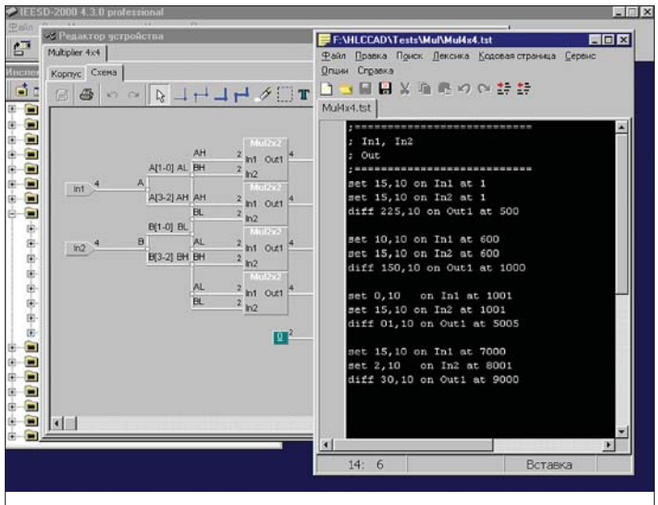

Рис. 2. Пример тестового файла для схемы «Умножитель 4×4»

2. Лингвистическое обеспечение автоматизации тестирования

2.1. Язык описания содержимого ОЗУ/ПЗУ

Система поддерживает несколько видов форматов. Среди них стандартные форматы Intel Hex и BIN. Кроме того, был разработан свой формат прошивки памяти MEM.

Файл прошивки памяти указывается в параметрах устройства (локальное меню над устройством в редакторе схемы пункт меню «Список параметров», далее имя памяти — Filename, ROM, Code memory). По расширению файла система выбирает формат файла:

- HEX — Intel HEX format;
- BIN — BIN format;
- иначе — MEM-формат.

Intel 8-bit Hex File Format — это обычный текстовый файл. Информация представлена в виде записей. Каждая запись это текстовая строка в файле. Записи могут идти в произвольном порядке. Значения представлены 2 или 4 цифрами в шестнадцатеричной системе счисления.

Формат записи:

:LLAAAARRDDDD.....DDDDCC

LL	Поле длины. Длина записи в байтах.
AAAA	Поле адреса. Адрес первого байта.
RR	Поле типа записи. 00 – данные, 01 – конец записей.
DD	Поле данных.
CC	Поле контрольной суммы. Дополнение всех данных в записи по модулю 256.

Пример:

```
:0601000001020304050 6E4
:00000001FF
```

Первая запись в примере адресует с позиции 100H числа от 1 до 6. Вторая запись информирует о последней записи в файле.

Файл прошивки памяти «BIN».

Содержимое файла в формате BIN последовательно по байтам загружается в память устройства.

Файл прошивки памяти «MEM». Это обычный текстовый файл, в котором в виде текста указано, какие значения и по каким адресам в памяти располагать. Формат предусматривает несколько команд для изменения текущих параметров:

- $DD <Size> — размерность слова в битах (по умолчанию — 8);
- $A <Address> — номер слова, с которого будут прописываться следующие команды (по умолчанию — 0);
- $AN <Notation> — система счисления, в которой будет задаваться адрес (по умолчанию — 16);
- $DN <Notation> — система счисления, в которой будут задаваться данные (по умолчанию — 16).

Все слова, начинающиеся не с символа «$», считаются данными. При записи очередного значения указатель текущего адреса увеличивается на разрядность слова.

Символ «;» считается указателем комментариев. Все символы в строке после «;» не обрабатываются.

Пример:

```
0 0
$DD 16
$A F5
A0F0 10
101A 1663
$DN 2
011001 110010 1100101
```

2.2. Язык управления тестовыми воздействиями и эталонными реакциями

С помощью этого языка разработчик имеет возможность указать тестовые воздействия, подаваемые на входы устройства, и эталонные значения для выходов. Каждая команда также определяет момент модельного времени, в которое происходит ее активизация. Любая схема разрабатываемого устройства может быть связана с собственным файлом тестовых воздействий (рис. 1.).

Файл тестовых воздействий (рис. 2) представляется в виде текстовых команд. Для анализа файла тестовых воздействий введено понятие текущего модельного времени. Эта величина определяет время активизации события в случае, если при его описании оно не определено. Например, разработчик может указать время установки значения на контакт в момент модельного времени, равного 5 пс. Далее в файле тестовых воздействий могут следовать команды, которые должны активизироваться в этот же модельный момент, но без указания величины времени. Текущее модельное время при анализе автоматически изменяется в том случае, если команда определяет время активизации.

Команды языка тестов приведены в таблице 1. В описании приведенных правил параметр, находящийся в квадратных скобках, может отсутствовать.

В случае, когда воздействия на схему производятся интерактивно, можно автоматически получить файл тестов по результатам моделирования. В процессе симуляции производится сохранение всех значений, устанавливаемых на контактах. Разработчик должен выбрать необходимый набор контактов, значения которых необходимо тестировать. Дополнительно нужно определить временной диапазон для сохранения (рис. 4).

В результате будет построен файл тестовых воздействий, который можно использовать для автоматического тестирования схемы устройства и содержимое которого может в дальнейшем модифицироваться.

2.3. Язык управления пакетным тестированием

Язык управления пакетным тестированием (язык сценария) обеспечивает построение «сценария» автоматического тестирования. Разработчику необходимо определить проект для тестирования и режимы тестирования:

- моделирование с учетом тестовых воздействий;
- моделирование сгенерированного по схеме VHDL-описания для тестирования системами сторонних фирм.

Полученный файл сценария можно использовать для пакетного тестирования проектов.

Исполнение команды файла сценария происходит в построчном режиме. Из этого следует, что описание каждой команды должно заканчиваться в той же строке, в которой и началось. Допускается написание нескольких команд в одной строке, разделенных символом «;». Каждая строка может иметь свою метку для реализации команд перехода.

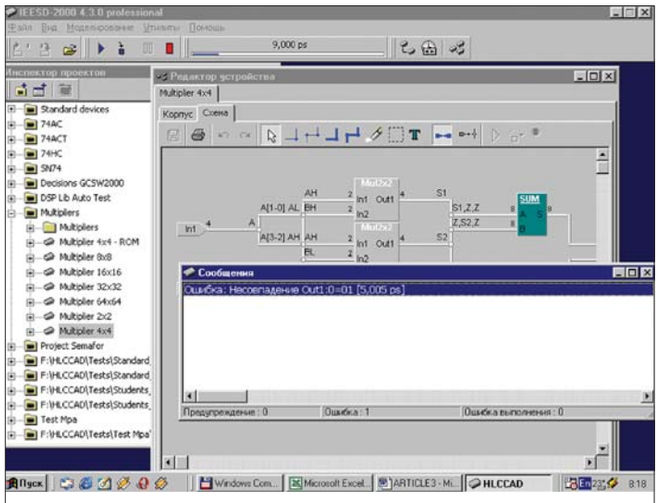

Рис. 3. Сообщение о несовпадении значений

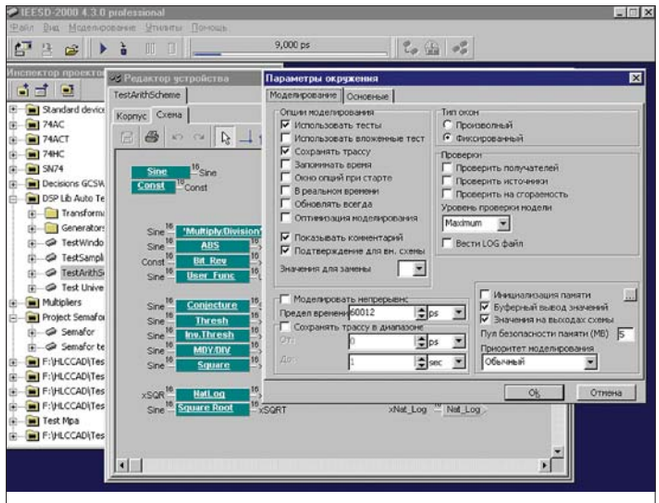

Рис. 4. Окно опций моделирования

Таблица 1. Формат языка тестов

Название	Правило	Описание
Установить значение	ItemName = ValueStr [Time] где	Устанавливает значение на контакт в указанный момент времени
	ItemName	В качестве ItemName можно указать имя контакта схемы или корпуса: «In1» или «Package1.In1». Можно устанавливать значения переменных моделей – значения памяти, регистров, битов или флагов: «Package1.R1». Если устройство содержит совпадающие имена (например, имя регистра и имя контакта), то перед именем ItemName можно дополнительно указать тип элемента в квадратных скобках – pin, mem или reg (например: [pin] In1=10[,2]).
	ValueStr	Значения ValueStr могут содержать текстовую строку со значением, где дополнительно после запятой можно указать систему счисления значения. Указанная система счисления действует на все остальные значения в командах до следующего изменения.
	Time	Параметр Time задает время активизации команды. Допустимы два варианта: • AT TimeValue – в указанный момент времени; • AFTER TimeValue – после последней выполненной команды и спустя указанный интервал времени. TimeValue представляет собой время в пикосекундах. Дополнительно можно изменить масштаб времени, указав дополнительно тип: sec, ms, us, ns или ps.
Проверить значение	ASSERT ItemName LogicOper ValueStr [Time] [REPORT String] [SEVERITY MessageType]	Если условие не выполняется, то генерируется сообщение с типом SEVERITY и текстом REPORT. Тип сообщения (MessageType) может принимать следующие значения: Warning, Message, Fatal, Error. В качестве LogicOper можно указать: • = – «Равно»; • /= – «Не равно»; • > – «Больше»; • >= – «Больше или равно»; • < – «Меньше»; • <= – «Меньше или равно».
Интервал времени	WAIT Time	Установить указатель модельного времени в указанный момент времени
	WAIT FOR Time	Увеличить указатель модельного времени на указанный интервал времени
Остановить моделирование	STOP_ Time	Остановить моделирование в указанный момент времени
	STOP_ FOR Time	Остановить моделирование после указанного интервала времени
Генератор значений	GENERATOR_ ItemName GenParams END	В качестве GenParams можно указать параметры генератора. Параметры генератора разделяются запятыми.
	а) COUNT_	• COUNT_ Value – количество событий для генерации;
	б) FROM_ TO	• [FROM_] Time TO Time – интервал активизации;
	в) FROM_ RANGE	• [FROM_] Time RANGE Time – интервал активизации;
	г) DELAY_	• DELAY_ (Time, Time, ...) – величины пауз между значениями;
	д) INTERVAL_	• INTERVAL_ Time – величина паузы между значениями;
	е) FREQUENCY_	• FREQUENCY_ Herz – частота появления значений (Hz, kHz, MHz, GHz);
	ж) DATA_	• DATA_ (ValueStr2, ValueStr2, ...) – значения;
	з) FILE_	• FILE_ FileName – имя файла со значениями;
	и) PHASE	• PHASE_ Time – задержка перед первым значением. Если не указаны значения, то величина значения будет увеличиваться на единицу каждый раз при активизации генератора. Отсчет значений начинается с нуля. Интервал генерации по умолчанию равен 1 пс.
Работа с дампом памяти	LOAD FileName ON ItemName [Time]	Загрузить содержимое дампа памяти из файла прошивки
	DIFFMEM FileName ON ItemName [Time]	Проверить содержимое дампа памяти и содержимое файла прошивки

Описание метки всегда начинается с первой позиции в строке и заканчивается символом «:».

Выполнение происходит с первой строки. После исполнения команд в строке происходит переход на следующую строку при условии, что среди выполненных команд не было команд перехода. После исполнения последней строки файла выполнение прекращается.

Во время исполнения системой ведется файл результатов действий — так называемый LOG-файл. Этот файл по умолчанию автоматически сохраняется в каталог с файлом сценария. По содержимому этого файла можно отследить весь процесс исполнения.

Языком сценария предусмотрено использование переменных для хранения промежуточных значений. Объявление переменной не требуется. При первом использовании имени переменной происходит ее инициализация. Например, при выполнении команды «Counter=10», будет выделено место для значения переменной «Counter» и записано в нее 10.

Операции над выражением имеют следующий синтаксис:

```
Value = Expression
```

В качестве Expression может быть записано любое арифметическое или логическое выражение с использованием скобок:

```
A=B+C; B=C*(A+B)-5; C=(A>B) or not (B+5<12);
```

Значения переменных хранятся в виде строк. Приведение к нужному типу происходит автоматически. Значение выражения может быть использовано в операндах команд.

Для обращения к значениям параметров, переданных после имени файла сценария, необходимо указать символ «%» и номер параметра (например, %5 или %12). Например, если были переданы параметры:

```
Test.CLD «D:\Project1.prd»
```

то в тексте сценария Test.CLD можно указать:

```
IncludeProject %1
```

В результате выполнения этой строки сценария будет осуществлено подключение проекта, расположенного в файле, имя которого передано в качестве первого параметра.

В языке сценария также присутствует оператор ветвления:

```
IF Expression THEN
```

При условии, что выражение Expression не равно 0, происходит выполнение команд, расположенных после ключевого слова

THEN. В противном случае происходит переход на следующую строку. Например:

```
IF ErrorCount>0 THEN ECHO «Incorrect device»
```

Среди команд языка сценариев можно выделить две важные группы: команды общего назначения и команды управления моделированием, описание которых отображено, соответственно, в таблицах 2 и 3.

Таблица 2. Команды общего назначения

Правило	Описание	Пример
LoadDesktop FileName	Загрузить содержимое рабочего стола из файла, определенного параметром FileName	LoadDesktop «D:\Desktop.env»
IncludeProject FileName	Загрузить в «Инспектор проектов» проектный файл, определенный параметром FileName	IncludeProject «D:\Project1.prd»
ExcludeProject FileName	Отключить в «Инспекторе проектов» проектный файл, определенный параметром FileName	ExcludeProject «D:\Project1.prd»
ExcludeProjects	Отключить в «Инспекторе проектов» все проекты	ExcludeProjects
Language = LanguageName	Изменить язык системы на LanguageName	Langauge=»Russian»
Goto LabelName	Перейти на метку LabelName	Goto «Label1»
ExecSoft FileName [, Params]	Исполнить файл FileName с параметрами Params	ExecSoft «Programator.exe» ExecSoft «Programator.exe», «Device1»
Rem	Признак комментария – текст до конца данной строки игнорируется	Rem Это комментарий
Log = FileName	Изменить имя LOG-файла	Log «D:\LogFile.log»
FileToLog FileName	Вывести содержимое текстового файла FileName в LOG-файл	FileToLog «D:\NewLogFile.log»
SaveLog	Принудительно сохранить LOG-файл	SaveLog
Exit	Прервать выполнение	Exit
Set «echo» = on или off	Включить или выключить режим записи результата исполнения команд	Set echo=on
Call FileName	Выполнить сценарий из файла FileName	Call «D:\Script2.hcl
Message MessageType TextString	Передать сообщение пользователю, где • MessageType – тип сообщения (Message, Error, Warning); • TextString – текст сообщения	Message error «Ошибка»

Таблица 3. Команды для моделирования

Правило	Описание	Пример
Check DeviceName	Проверить устройство DeviceName на наличие ошибок	CheckDevice «Device1»
Run DeviceName	Выполнить моделирование устройства DeviceName	Run «Device1»
Trace DeviceName	Выполнить шаг моделирования устройства DeviceName	Trace «Device1»
Reset	Выключить режим моделирования	Reset

Для команд языка сценария, определенных для элементов проекта, введено понятие курсора. Курсор определяет рабочий проект и элемент в дереве проекта. Для изменения рабочего проекта используется команда

```
Project FileName
```

где FileName — это имя файла проекта. Следует отметить, что команда не подразумевает подключения проекта в случае его отсутствия в «Инспекторе проектов».

С помощью команды «Folder FolderName» происходит переход курсора на папку FolderName в текущей ветви. Для перехода на уровень выше нужно указать в качестве имени две точки «..». Допускается использование нескольких параметров FolderName, разделенных символом «\».

Пример команды запуска процесса симуляции устройства (номера строк приведены для нижеследующего комментария примера):

```
1  IncludeProject «D:\i8051.prd»
2  IncludeProject  D:\HLCCAD\Projects\Standard\Standard.prd»
3  Project «i8051.prd»
4  Folder «Tests»
5  Run «Test i8051»
6  Reset
7  Folder «..»
8  Run «Tests\Test i8051»
9  Reset
```

Команды в первой и второй строке подключают проекты «i8051» и «Standard». Команда в строке 3 устанавливает курсор на проект «i8051». Команда в строке 4 перемещает курсор в папку «Tests». После выполнения команды в 5 строке производится запуск симуляции устройства «Test i8051». После останова процесса моделирования выполняется команда в строке 6, которая выключает процесс симуляции. Команда в строке 7 перемещает курсор из папки «Tests» в «корень» проекта. Команда в 8 строке запускает процесс симуляции для того же устройства, что и в строке 5. Команда в 9 строке повторно выключает процесс моделирования.

Допускается принудительная установка файла тестовых воздействий для устройства командой «MTest = Expression», также указание предела моделирования «MLimit = Expression».

Для генерации описания на VHDL предусмотрена команда:

```
BuildVHDL DeviceName VHDLOpt
```

В качестве параметра DeviceName указывается имя синтезируемого устройства, а в качестве VHDLOpt — параметры генерации, разделенные символом «;». В качестве параметра можно указать:

- Dir=FileDir — каталог, в котором будут расположены генерируемые файлы;
- Test=True или False — признак генерации тестов;
- TestType=VHDL или VEC — тип тестовых воздействий;
- Delays= True или False — признак наличия команд управления модельным временем;
- ZEmul= True или False — признак эмуляции Z состояния;
- Filename=Project или Device — способ выбора имени генерируемых файлов.

Для изменения значений параметров устройства предусмотрена команда:

```
With Device DeviceName do DeviceOperands end
```

где DeviceName — имя устройства, а DeviceOperands — команды, разделенные символом «;». В качестве DeviceOperands можно указать:

- Set Param ParamName as Expression — установить значение параметра ParamName значением выражения Expression;
- Exec Param ParamName — выполнить внешний параметр устройства.

3. Примеры практического использования пакетного тестирования

3.1. Поддержка «реинжиниринга» систем на базе СИС

Общеизвестно, что и в России, и в Белоруссии, и по всему миру, существует огромное количество реально функционирующих систем, построенных на микросхемах средней степени интеграции (СИС) на базе серий К155, К1500 и др. Перепроектирование таких систем под новую элементную базу, например, ПЛИС (получившее название «реинжиниринг») является очень трудоемкой задачей, которая, как правило, сопровождается огромным объемом работ по повторной симуляции и отладке.

Опираясь на уникальные возможности системы HLCCAD, мы предложили нетрадици-

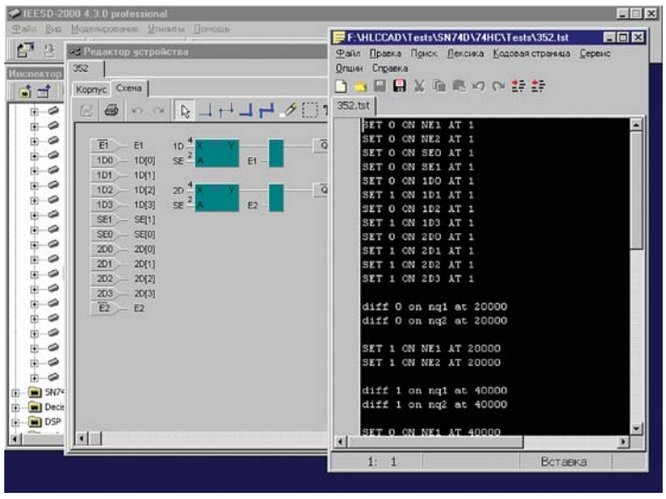

Рис. 5. Модель микросхемы 352(КП19) и фрагмент теста для нее

Таблица 4. Реализованные модели микросхем серии SN74

DEVICE	аналог	КР1533	1564	1594	КР1554
		ALS	HC	ACT	AC
OO	ЛА3	+	+	+	+
O1	ЛА8	+	+		
O2	ЛЕ1	+	+	+	+
O3	ЛА9	+	+		
O4	ЛН1	+	+	+	+
O5	ЛН2	+	+	+	
O8	ЛИ1	+	+	+	+
O9	ЛИ2	+	+		
10	ЛА4	+	+	+	+
11	ЛИ3	+	+	+	
12	ЛА10	+			
15	ЛИ4	+			
20	ЛА1	+	+		+
21	ЛИ6	+	+		+
22	ЛА7	+			
28	ЛЕ5				
30	ЛА2	+	+		
32	ЛЛ1	+	+	+	+
33	ЛЕ11	+			
34	ЛИ9				+
42	ИД6		+		
51	ЛР11		+		
74	ТМ2	+	+	+	+
75	ТМ7		+		
77	ТМ5		+		
85	СП1		+		
86	ЛП5	+	+		+
107	ТВ6		+		
109	ТВ15	+	+	+	+
112	ТВ9	+	+	+	+
113	ТВ10	+			
114	ТВ11	+			
125	ЛП8		+	+	
136	ЛП12	+			
138	ИД7		+	+	
139	ИД14	+	+	+	+
147	ИВ3		+		
148	ИВ1		+		
151	КП7	+	+	+	
152	КП5		+		
153	КП2	+	+	+	+
154	ИД3		+		
157	КП16	+	+	+	+
158	КП18	+	+	+	+
160	ИЕ9	+	+	+	
161	ИЕ10	+	+	+	+
162	ИЕ11	+	+		
163	ИЕ18	+	+	+	+
164	ИР8	+	+		
165	ИР9	+	+		
166	ИР10	+	+		
169	ИЕ17			+	
173	ИР15		+		
174	ТМ9	+	+	+	+
175	ТМ8	+	+	+	+
180	ИП2		+		
190	ИЕ12	+	+		
191	ИЕ13	+	+		
192	ИЕ6	+	+		+
193	ИЕ7	+	+		+
194	ИР11		+		
237			+		
238			+		
240	АП3	+	+	+	+
241	АП4	+	+	+	+
242	ИП6	+	+		
243	ИП7	+	+		
244	АП5	+	+	+	+

Таблица 4. Реализованные модели микросхем серии SN74 (окончание)

DEVICE	аналог	КР1533	1564	1594	КР1554
		ALS	HC	ACT	AC
245	АП6	+	+	+	+
251	КП15	+	+	+	
253	КП12	+	+	+	+
257	КП11	+	+	+	+
258	КП14	+	+	+	+
259	ИР30	+	+		
266	ЛП13		+		
273	ИР35	+			+
280	ИП5		+		
283	ИМ6		+		
298	КП13		+		
299	ИР24	+	+	+	+
323	ИР29	+		+	+
352	КП19	+	+	+	
353	КП17	+			
365	ЛП10		+		
366	ЛН6		+		
367	ЛП11		+	+	
368	ЛН7	+	+	+	
373	ИР22	+	+	+	+
374	ИР23	+	+	+	+
377	ИР27		+	+	
378	ИР18		+		
379	ИР19		+	+	
393	ИЕ19		+		
465	АП14	+			
466	АП15	+			
520				+	
521				+	
533	ИР40		+	+	+
534	ИР41		+	+	+
540			+		
541			+	+	
563			+	+	
564			+		
573	ИР33	+			
574	ИР37	+			
620			+		
623			+		
640	АП9	+	+	+	+
643	АП16	+			
645			+		
646	АП20		+	+	+
648			+		
651			+		
652			+		
664			+		
665			+		
821				+	
823				+	
825				+	
841				+	
843				+	
845				+	
873	ИР34	+			
874	ИР38	+			
1000	ЛА21	+			
1002	ЛЕ10	+			
1003	ЛА21	+			
1004	ЛН8	+			
1005	ЛН10	+			
1008	ЛИ8	+			
1010	ЛА24	+			
1011	ЛИ10	+			
1020	ЛА22	+			
1032	ЛЛ4	+			
1034	ЛП16	+			
1035	ЛП17	+			

онный подход к решению задач «реинжиниринга». Этот подход основан на предварительной подготовке моделей микросхем средней степени интеграции, использованных при первичной разработке системы. Такие модели должны обеспечивать адекватную симуляцию и VHDL-генерацию. Для микросхем семейств К155 и родственных, с использованием справочных описаний в HLCCAD созданы иерархические модели соответствующих микросхем из стандартных компонентов HLCCAD [2]. На рис. 5 представлен пример реализации модели. Полный перечень реализованных моделей представлен в таблице 4.

Система HLCCAD обеспечивает пакетную проверку адекватности справочным описаниям (посредством симуляции) и автоматическую генерацию корректных VHDL-описаний нужных микросхем.

Таким образом, корректное по построению перепроектирование старых систем заключается фактически в графическом вводе схем в HLCCAD и исправлении ошибок ввода (с помощью симуляции). При необходимости, на этом этапе можно провести и модификацию выполняемых системой функций в целях модернизации или адаптации к новым условиям функционирования. Далее HLCCAD обеспечивает автоматическую генерацию синтезируемого VHDL-описания спроектированной системы для загрузки в САПР более низкого уровня (Altera MaxPlus II, Xilinx ISE и др.).

Очевидно, что использование такого подхода резко сокращает сроки «реинжиниринга» систем, построенных на микросхемах средней степени интеграции произвольных семейств.

Ниже в качестве примера приведен фрагмент файла сценария, обеспечивающий пакетное тестирование созданной библиотеки моделей микросхем серии К155 (зарубежный аналог — серия SN74).

```
Set «echo»=off; LoadDesktop «F:\HLCCAD\testing.env»;
Set «NeedLoadModelingDesktop»=off; Set «FullOutputTesting»=on;
Language=»English»; ExcludeProjects; Log=»F:\Tester\74ac.log»;
MLimit=»None»;
glTestBAT=»D:\VHDLtester\VHDLTester.bat»;
glTestBATPath=ExtractFilePath(glTestBAT);
IncludeProject «F:\HLCCAD\Tests\SN74D\74AC\74ac.prj»
IncludeProject «F:\HLCCAD\Tests\SN74D\74HC\74hc.prj»
IncludeProject «F:\HLCCAD\Tests\SN74D\SN74\Sn74.prj»
Project «F:\HLCCAD\Tests\SN74D\74AC\74ac.prj»
echo «Testing 74ac.prj 00»; SaveLog
ClearErrors
MLimit=»55000»;  MTest=»F:\HLCCAD\Tests\SN74D\74AC\tests\
00.tst»
Echo «Modeling»; SaveLog; Run «00»; Reset;
if ErrorCount = 0 then VOutput=CurDir «\VHDL\74ac\a00»;
BuildVHDL»00»,  Dir=VOutput,  Test=True,  TestType=VEC,
Delays=False,
 ZEmul=True,FileName=Device;
 if ErrorCount = 0 then echo «External VHDL modeling»; SaveLog;
ExecSoft glTestBat, VOutput «\ a00 « glTestBATPath;
 VResult=File(VOutput «\a00.crs»);if VResult<>»Ok» then echo
VResult;
echo «Testing 74ac.prj 02»; SaveLog
ClearErrors
MLimit=»40000»; MTest=»F:\HLCCAD\Tests\SN74D\74AC\TESTS\
02.tst»
Echo «Modeling»; SaveLog; Run «02»; Reset;
if ErrorCount = 0 then VOutput=CurDir «\VHDL\74ac\a02»;
BuildVHDL  «02»,  Dir=VOutput,  Test=True,  TestType=VEC,
Delays=False,
ZEmul=True, FileName=Device;
if ErrorCount = 0 then echo «External VHDL modeling»; SaveLog;
ExecSoft glTestBat, VOutput «\ a02 « glTestBATPath;
VResult=File(VOutput «\a02.crs»);if  VResult<>»Ok»  then  echo
VResult;
```

3.2. Контроль работоспособности новых версий HLCCAD

Пакетное тестирование, обеспеченное в HLCCAD, используется нами для перманентного контроля корректности функционирования самой системы HLCCAD.

Для этого все разработанные в HLCCAD проекты снабжаются тестовыми файлами и файлами сценария, и все такие сценарии объединяются для организации последовательной загрузки, компиляции и симуляции в HLCCAD, генерации векторных тестов (в формате Max+Plus II) по результатам симуляции; компиляции и симуляции средствами Max+Plus II.

Таким образом, мы контролируем возможность появления ошибок одной из четырех следующих типов:

- ошибки компиляции в HLCCAD;
- ошибки симуляции в HLCCAD;
- ошибки компиляции сгенерированных VHDL-описаний средствами Max+Plus II;
- ошибки симуляции сгенерированных VHDL-описаний средствами Max+Plus II.

В настоящее время регрессионное (в некоторых источниках используется также название регрессивное) тестирование системы HLCCAD базируется на контроле компиляции и симуляции более 500 проектов различной степени сложности, включая проекты для семейств СИС, представленных в таблице 4.

Ниже приводится пример отображения на сайте разработчиков процесса регрессионного тестирования HLCCAD.

Тестирование: общая информация

Total devices	Всего устройств в данном проекте
All errors	Всего ошибок в данном проекте
Modeling errors	Всего ошибок моделирования в HLCCAD
VHDL compile errors	Всего ошибок компиляции в Max+Plus II
VHDL simulation errors	Всего ошибок симуляции в Max+Plus II

Последнее тестирование производилось 13 декабря 2002 года

Project	Total devices	All errors	Modeling errors	VHDL compile errors	VHDL simulation errors	time of testing
74ac.log	26	6	0	6	0	4:08
74act.log	21	4	0	4	0	3:19
74hc.log	50	7	0	7	0	8:28
sn74.log	59	10	0	10	0	9:56
dsp.log	4	0	0	–	–	0:10
gcsw2000.log	4	2	1	1	0	0:32
multipliers.log	7	6	0	6	0	1:40
proverka.log	6	0	0	0	0	1:08
students_h.log	12	1	1	–	–	0:25
students_m.log	12	4	1	3	0	2:12
test_standard_devices_h.log	159	12	12	–	–	0:54
test_standard_devices_m.log	158	27	9	18	0	30:32
mpa.log	8	8	8	–	–	0:10
Total	526	87	32	55	0	63:39

История тестирования

http://newit/projects/hlccad/testing/2002_12_13.htm
http://newit/projects/hlccad/testing/2002_10_11.htm
http://newit/projects/hlccad/testing/2002_07_08.htm
http://newit/projects/hlccad/testing/2002_01_08.htm
http://newit/projects/hlccad/testing/2001_12_19.htm
http://newit/projects/hlccad/testing/2001_12_11.htm
http://newit/projects/hlccad/testing/2001_04_27.htm
http://newit/projects/hlccad/testing/2001_04_26.htm
http://newit/projects/hlccad/testing/2001_04_21.htm
http://newit/projects/hlccad/testing/2001_04_19.htm
http://newit/projects/hlccad/testing/2001_04_05.htm
http://newit/projects/hlccad/testing/2001_03_24.htm
http://newit/projects/hlccad/testing/2001_03_22.htm
http://newit/projects/hlccad/testing/2001_03_20.htm
http://newit/projects/hlccad/testing/2001_03_03.htm
http://newit/projects/hlccad/testing/2001_02_15.htm

Заключение

По многочисленным отечественным и зарубежным источникам верификация проектов занимает от 50 до 80% временных и стоимостных ресурсов. Эффективные средства тестирования, встроенные в систему HLCCAD и описанные в данной статье, помогают существенно сократить строки и стоимость разработки аппаратного обеспечения встроенных цифровых систем. Базовые версии наших разработок и разнообразные демонстрационные примеры можно найти на сайте NewIT.gsu.unibel.by.

Литература

1. Долинский М. Концептуальные основы и компонентный состав IEESD-2000 — интегрированной среды сквозной совместной разработки аппаратного и программного обеспечения встроенных цифровых систем // Компоненты и технологии. 2002. № 8.
2. Долинский М., Литвинов В., Галатин А., Ермолаев И. HLCCAD — среда редактирования, симуляции и отладки аппаратного обеспечения // Компоненты и технологии. 2003. № 1.
3. Долинский М., Ермолаев И., Толкачев А., Гончаренко И. WInter — среда отладки программного обеспечения мультипроцессорных систем // Компоненты и технологии. 2003. № 2.
4. Долинский М. Тенденции и перспективы развития EDA-индустрии по материалам новостей специального Internet-портала DACafe.com. Январь 2001 — октябрь 2002. Часть I // Компоненты и технологии. 2002. № 9.
5. Долинский М. Тенденции и перспективы развития EDA-индустрии по материалам новостей специального Internet-портала DACafe. com. Январь 2001 — октябрь 2002. Часть II // Компоненты и технологии. 2003. № 1.
6. Долинский М. Тенденции и перспективы развития EDA-индустрии по материалам новостей специального Internet-портала DACafe.com. Ноябрь — Декабрь 2002 // Компоненты и технологии. 2003. № 2.
7. Долинский М. С., Лещенко Л. Н., Стома И. С. Настраиваемый синтаксический анализатор языков регулярных и контекстно-свободных грамматик // Автоматизация проектирования в электронике: Сб. науч. тр. АН УССР ИТК. Киев. 1993. Вып. 47.
8. Толкачев А. И. Языки программирования и описания аппаратуры: универсальный синтаксический анализатор // Труды международной конференции «Информационные технологии в бизнесе, образовании и науке». Минск. 1999.
9. Материалы сайта NewIT.gsu.unibel.by.