SystemC: единая среда для тестирования встроенных систем

Язык SystemC становится новым стандартом в области EDA. Многие разработчики начинают использовать его для моделирования сложных систем. SystemC был в основном разработан для описания абстрактных аппаратных и программных моделей, т.к. они легко могут быть использованы для быстрого макетирования (rapid prototyping). Но SystemC может применяться и для описания модулей на более высоком уровне детализации, например, на уровне регистровых передач и ассемблере. Такой подход, как будет показано ниже, очень выгоден для тестирования системы. Можно создать одну тестирующую процедуру и применять её на всех фазах проектирования, пользуясь одной и той же моделью обработки ошибок и стратегией генерации тестов. К тому же создаваемые тесты могут применяться как к программным, так и к аппаратным компонентам, что делает систему SystemC пригодной для работы со встроенными системами.

ВВЕДЕНИЕ

SystemC — это название библиотеки классов и методологии проектирования [1]. SystemC позволяет создавать эффективные и точные модели программных алгоритмов, аппаратных архитектур, интерфейсов и схем на системном уровне, т.е. практически всех компонентов встроенных систем. Такой подход имеет значительный потенциал, так как основан на однородном описании на C++ и легко позволяет моделировать, тестировать системы, рассматривать альтернативные архитектуры. Кроме того, команде проектировщиков может быть предложено развёрнутое описание процесса работы всей системы. Это описание представляет собой C++-программу, которая при исполнении ведёт себя так же, как и система. SystemC позволяет поднять уровень абстракции проекта и рассмотреть разные варианты перед непосредственным синтезом аппаратной части.

Тестирование SystemC-описаний всё ещё актуально, так как это новый язык, и до сих пор идёт поиск таких моделей ошибок, а также тестирование возможных неисправностей, которое могло бы применяться независимо к аппаратной и программной частям. Обычные программные покрытия [2] оказываются неприемлемыми для тестирования аппаратной части, как это и показано в статье [3]. Другое решение состоит в том, чтобы взять наработки по моделям ошибок и покрытиям из аппаратных описаний на уровне RTL (уровень регистровых передач) [4–6]. Также многие модели ошибок и метрики работают на абстрактном уровне [5, 7]. Специально для повышения производительности детерминистических NPG [7–9] были разработаны вероятностные генераторы тестов на основе генетических алгоритмов. Они показывают хорошие результаты при использовании на RTL-уровне аппаратной части с целью тестирования ошибок на уровне вентилей.

Предлагаемая для SystemC методика тестирования ставит своей целью охватить тестами весь процесс проектирования: от системного уровня до структурного, используя одну и ту же модель ошибок и технику генерации тестов. Основная задача такого тестирования — проверка корректности всех трансформаций системы при отображении её описания на аппаратные и программные модули на основе симуляции работы системы. Точное описание системы и производительность модели, достигаемые при использовании описания на SystemC, позволяют создать высокоэффективную процедуру тестирования.

Раздел 1 описывает использование SystemC для генерации различных вариантов реализации системы и возможность генерации тестов по однородному описанию. Модель проверки функциональности системы тестами описывается в разделе 2. В разделе 3 представлен весь цикл проектирования и генерации тестов на основе использования встроенного центрального процессора (ЦП). Последний раздел содержит выводы и очерчивает направления дальнейших работ.

1. АЛЬТЕРНАТИВНЫЕ ОПИСАНИЯ

Исследуем альтернативные версии описаний системы, доступные при использовании SystemC. Рисунок 1 показывает три различных варианта одной SystemC-модели.

Представим систему в виде взаимодействующих процессов. Их работу можно реализовать на:

- специализированном аппаратном обеспечении (ad-hoc hardware);
- программе, запускаемой на ЦП общего назначения;
- программе, запускаемой на специально разработанном встроенном ЦП;
- комбинации этих трех подходов.

Первоначальное описание используется для генерации тестов согласно модели побитового покрытия [11] и подхода функционального тестирования [7], который в данном случае применяется для представления в системе SystemC. Для всех трёх вариантов SystemC играет различную роль в процессе генерации тестов.

Специализированное аппаратное обеспечение (ad-hoc hardware)

В этом случае описание SystemC полностью отображается на аппарат-

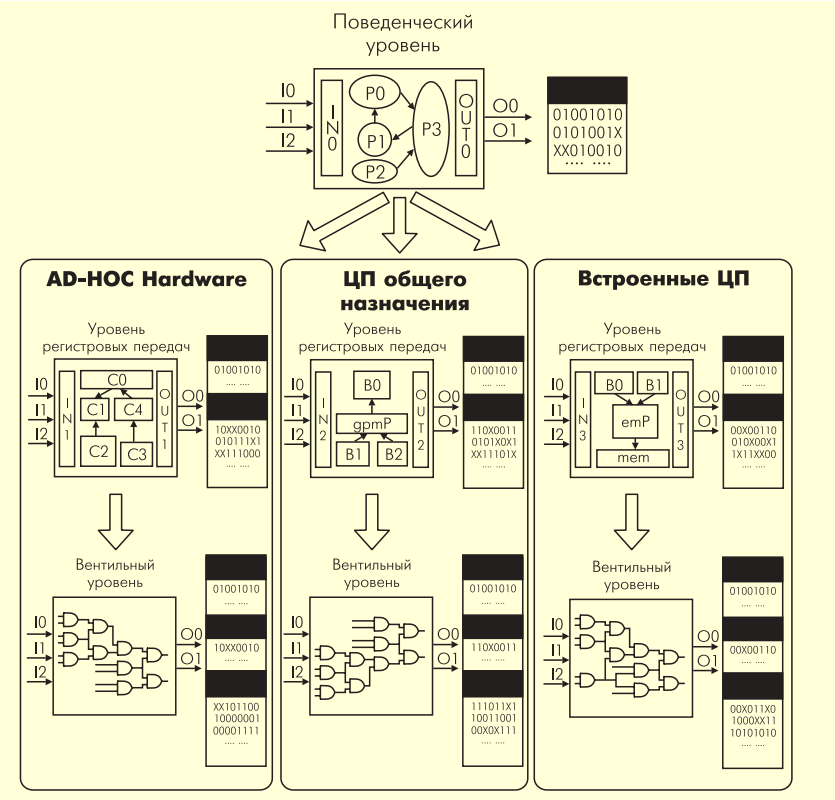

Рисунок 1 **Альтернативные версии описаний системы**

ные компоненты. Программных частей нет вообще. SystemC используется для определения начальной модели всей системы, но последующие фазы основываются на VHDL (или Verilog). Перевод SystemC в VHDL необходимо проверить. Это довольно сложная задача, т.к. она требует перевода с языка, работающего с циклами (SystemC), на язык, основанный на событиях (VHDL, Verilog). Чтобы упростить задачу, описание в SystemC следует перевести на уровень RTL. После этого перевести описание в VHDL не так сложно, это скорее задача синтаксического перевода, а не семантического [1]. Тесты, взятые с системного уровня, можно использовать для проверки нового аппаратного описания. Смена среды проектирования приводит к необходимости пересмотра техники тестирования и модели ошибок для последующих фаз, что увеличивает затраты на проектирование. С другой стороны, можно использовать универсальное SystemC-описание и универсальную технику тестирования.

ЦП общего назначения

Процесс проектирования заключается в выборе описания ЦП общего назначения, разработанного сторонним производителем, и адаптации описания SystemC на более низкий уровень. Модель на SystemC может работать как приложение на этом ЦП. Процессы SystemC-модели становятся программными модулями, запускаемыми на ЦП. Чем меньше разница между описанием SystemC-модели и средой используемого ЦП, тем меньше будет работы по переводу. Системная архитектура должна иметь память для хранения программы и какие-нибудь аппаратные блоки, работающие как интерфейсы. На SystemC могут быть спроектированы как программные, так и дополнительные аппаратные блоки. Для тестирования можно применять тесты, разработанные для SystemC. Приобретённые описания процессоров считаются протестированными, поэтому исходный набор тестов для SystemC будет тестировать программные модули и все дополнительные аппаратные блоки. Более того, разработчики могут приобрести SystemC-описания каждого дополнительного блока, а оставшиеся в модели процессы переводить только в программные модули. В этом случае исходный набор тестов под SystemC будет тестировать работу и взаимодействие программных частей.

Специально разработанный встроенный ЦП

В этом случае все условия и ограничения проекта учитываются при разработке встроенного ЦП. Типичный пример — системы-на-кристалле (Systems on Chip, SoC's). Разработчик должен определиться с каждой частью системы: ЦП, память и интерфейсы. Гибкость и возможности SystemC можно использовать для быстрого нахождения лучшей конфигурации (аппаратной и программной). Разработчик может легко разделять процессы системы между аппаратными и программными частями и сравнивать производительность полученных систем. Тесты можно взять те же, что использовались в SystemC, но только предварительно создать SystemC-описание на низком уровне. На этом этапе необходимо использовать тесты с первого шага. Количество компонент, которые нужно протестировать здесь, больше, чем в других вариантах. К тому же сложность этих компонент также достаточно высока (например, встроенный ЦП). Таким образом, разработка тестов по более простому системному описанию может упростить эту задачу. Тесты, полученные на предыдущих фазах проектирования, используются для проверки нового шага. Более того, на каждом уровне тесты улучшаются благодаря учёту специфических для этого шага моментов.

2. ТЕХНИКА ТЕСТИРОВАНИЯ

Каждый раз при переводе описания с текущего уровня абстракции на более низкий требуется проверка корректности этого перехода. Если формальные методы не позволяют этого сделать, то задача решается путём эмуляции работы системы. Все тесты, разработанные на более высоких уровнях абстракции, применяются к описанию на более низком уровне. Кроме того, на каждом новом шаге детализации набор тестов дополняется специфическими для этого уровня тестами, которые надо применить и для предыдущих уровней. При использовании общего для всех фаз проектирования языка описания такой подход к тестированию используется на всех уровнях — от системного до структурного. SystemC может служить именно таким языком. Более того, если используются компоненты сторонних разработчиков, их описания на VHDL или Verilog можно перевести в формат SystemC, как это описано в статье [10].

Предлагаемая стратегия тестирования основана на модели ошибок *побитового покрытия* [11], которая демонстрирует высокую корреляцию ошибок, смоделированных на разных уровнях абстракции. Эта модель ранее использовалась для тестирования аппаратных компонент на функциональном уровне, уровне RTL и уровне вентилей. Здесь мы применяем тот же подход для встроенных систем, тестируя и аппаратную, и программную части. Каждый бит каждой переменной, условия и портов входа/выхода устанавливается в 0 или 1, таким образом при ошибке получается SystemC-описание. Затем оно сравнивается с корректным описанием.

Для тестирования встроенного ЦП были применены две различные программы: алгоритм фильтрации FIR (Finite Input Response, конечный входной отклик) из пакета SystemC и тесто-ориентированный алгоритм. Эти два алгоритма имеют разные цели. Первый ориентирован на тестирование конкретного алгоритма, написанного под разработанный встроенный ЦП. В общем случае он не покрывает все возможные ошибки, потому что использует только ограниченный набор из доступных инструкций, и ошибки в памяти не отслеживаются, если они находятся вне сегментов кода и данных. В то же время этот алгоритм короче и требует меньше информации о ЦП. К тому же его может быть достаточно для проектирования встроенного ЦП, предназначенного для решения одной конкретной задачи. Если же стоит задача создать более универсальный ЦП и есть возможность потратить больше времени на тестирование, то надо применять тесто-ориентированный алгоритм. Он использует все доступные инструкции и располагается в различных областях памяти встроенной архитектуры, достигая благодаря этому большего покрытия ошибок.

Тесты генерируются случайным образом. Для обоих алгоритмов тест — это данные, с которыми работают алгоритмы. Чтобы достичь большего покрытия ошибок, случайный генератор тестов может быть заменён на детерминистический.

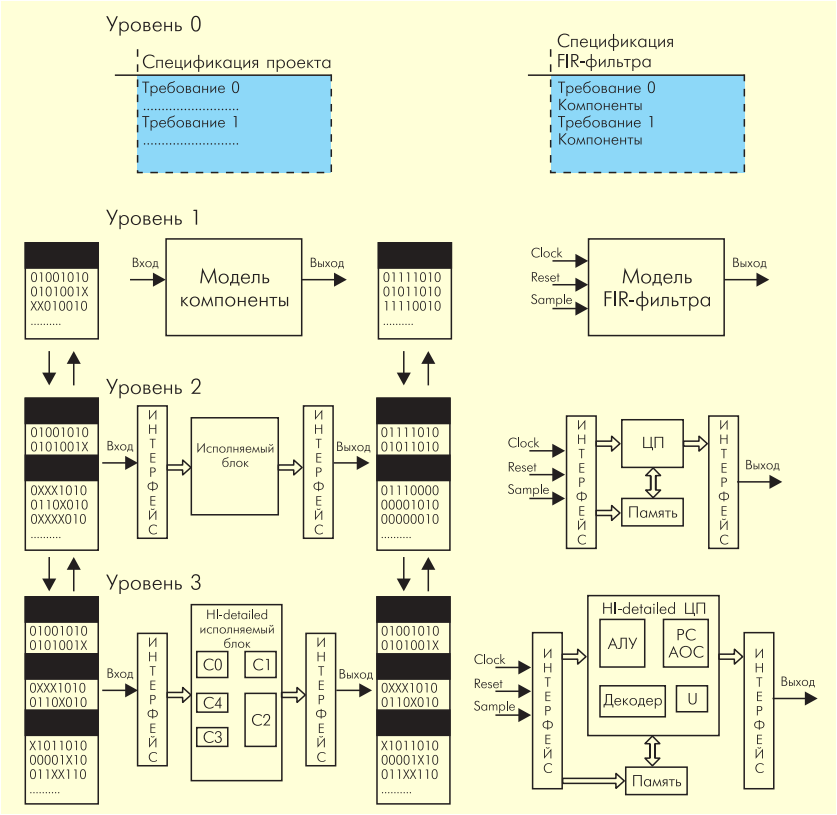

Рисунок 2 **Применение техники тестирования**

```
SC_MODULE(fir) {
   sc_in<bool>       reset;
   sc_in<bool>       input_valid;
   sc_in<int>        sample;
   sc_out<bool>      output_data_ready;
   sc_out<int>       result;
   sc_in_clk         CLK;
   …
   SC_CTOR(fir){
         SC_CTHREAD(entry, CLK.pos());
         watching(reset.delayed() == true);
         #include "fir_const.h"
   }
   …
};
```

Рисунок 3 **Частичное SystemC-описание фильтра FIR на уровне 1**

3. СПЕЦИАЛИЗИРОВАННЫЙ ВСТРОЕННЫЙ ЦП

В этой главе описывается применение предложенной методики тестирования к варианту, основанному на использовании специализированного встроенного ЦП. Рассмотрим входящий в документацию SystemC пример задачи с использованием FIR-фильтра [1].

Рисунок 2 показывает применение предложенной техники тестирования к примеру FIR. Основные преобразования модели SystemC показаны в левой части. Системная спецификация (уровень 0) реализуется встроенным ЦП. В правой части показан пример FIR-фильтра. Сначала спецификация переводится в описание (уровень 1), состоящее из взаимодействующих процессов:

- системных интерфейсов,
- системного поведения.

Принятая модель ошибок здесь используется для генерации тестов, с помощью которых проверяется корректность перехода от спецификации.

Затем для проверки ассемблерного кода, полученного из программной системы уровня 1, строится абстрактная модель ЦП на языке SystemC (уровень 2). Эта проверка осуществляется путём прогона тестов, полученных на предыдущем уровне при описании уровня 2.

```
SC_MODULE(cpu_mono) {
   sc_in<bool>             start;
   sc_in<bool>             clear;
   sc_in<sc_lv<16> >       data_in;
   sc_in<bool>             clk;

   sc_out<bool>            wr;
   sc_out<sc_lv<16> >      accout;
   sc_out<sc_lv<10> >      addr;
   …
   SC_CTOR(cpu_mono) {
         SC_METHOD(evaluate);
         sensitive_pos << clk << start << clear;
   }

   void evaluate();
};

SC_MODULE(mem) {
   sc_in<sc_lv<10> >       addr;
   sc_in<bool>             wr;
   sc_in<sc_lv<16> >       accout;
   sc_out<sc_lv<16> >      data_in;

   sc_lv<16>               memory[1024];

   void mmu();

   SC_CTOR(mem) {
   for (int I==0; I<1024; I++)
         memory[i] == "0000000000000000"
         …
         SC_METHOD(mmu);
         sensitive << addr;
         …
   }
};
```

Рисунок 4 **Частичное SystemC-описание фильтра FIR на уровне 2**

```
SC_MODULE(cpu_rtl) {
   sc_in<sc_logic>         start;
   sc_in<sc_logic>         clear;
   sc_in<sc_logic>         clk;
   sc_in<sc_lv<16> >       data_in;

   sc_out<bool>            wr;
   sc_out<sc_lv<16> >      accaut;
   sc_out<sc_lv<10> >      addr;

   // Internal Signals
   …

   // Modules pointers
   controller_ov_fsm_syn   *controller_instance;
   clock_gen_fsm_syn       *clockGenerator_instance;
   clock_gen_fsm_syn       *clockGenerator_1_instance;
   clock_gen_fsm_syn       *clockGenerator_2_instance;
   clock_gen_fsm_syn       *clockGenerator_3_instance;
   IR_stx_syn              *instruction_register_instance;
   PC_fsm_syn              *program_counter_instance;
   acc_fsm_syn             *accumulator_instance;
   decoder_sse_syn         *decoder_instance;
   mar_stx_syn             *memory_address_instance;
   out_mod1                *out_module1_instance;
   out_mod2                *out_module2_instance;
   out_mod3                *out_module3_instance;

   SC_CTOR(cpu_rtl_autl) {
   …
   // Port binding between modules
   …
   }
};
```

Рисунок 5 **Частичное SystemC-описание фильтра FIR на уровне 3**

Здесь же определяется набор новых возможных ошибок. Это делается с помощью всё той же модели ошибок, что и раньше, но уже на уровне 2. Для покрытия новых ошибок добавляются соответствующие тесты. Корректные результаты новых тестов получаются при запуске их на SystemC-описании уровня 2.

Такой анализ доходит до RTL-уровня, но может быть расширен и до уровня вентилей при использовании той же стратегии тестирования. В этом случае можно воспользоваться системой перевода описаний VHDL и Verilog в SystemC [10].

Эталонная проверка FIR (FIR benchmark)

Ниже показаны частичные представления SystemC на разных уровнях. Часть системной архитектуры уровня 1 показана на рис. 3, уровня 2 — на рис. 4, уровня 3 — на рис. 5.

Ассемблер встроенного ЦП показан в табл. 1. Каждая 16-бит инструкция состоит из 10-бит поля адреса и 6-бит поля кода инструкции. C++-программа, описывающая FIR-фильтр, компилируется в 320 ассемблерных команд, которые моделируются SystemC, как показано на рис. 5.

Таблица 2 описывает основные параметры описания: количество строк SystemC-кода и число ошибок для каждого уровня. Общие результаты применения описанной техники тестирования (число тестов, покрытие ошибок тестами предыдущего уровня и тестами, сгенерированными для нового уровня) приведены в табл. 3.

Сгенерированные тесты позволили производить корректные переводы описания системы с одного уровня на другой.

Таблица 1. Ассемблер встроенного ЦП

Инструкция	Адрес	Код
ADD	ACC + MEM[IND] → ACC	000001
SHIFT_R	SHIFT_RIGHT(ACC) → ACC	000010
LOAD	MEM[IND] → ACC	000100
STORE	ACC → MEM[IND]	001000
JMP	IND → PC	010000
JNZ	IF (ACC ≠ 0) IND → PC	010001
HALT	Halt the CPU	100000
NOP	No operation	Другие

Таблица 2. Основные параметры описания

Уровень	Количество строк	Число ошибок
1	1529	606
2	2637	750
3	3043	890

Таблица 3. Генерация функциональных тестов для встроенных и универсальных программ

Уровень	Встроенный алгоритм			Тесто-ориентированный алгоритм		
	число тестов	покрытие ошибок тестами предыдущего уровня, %	покрытие ошибок тестами нового уровня, %	число тестов	покрытие ошибок тестами предыдущего уровня, %	покрытие ошибок тестами нового уровня, %
1	50	—	98,5	—	—	—
2	177	44	63	93	—	90,3
3	321	69	71	178	78,5	88,9

ЗАКЛЮЧЕНИЕ

В этой работе анализируются три подхода при проектировании встроенной системы, исходя из её начального SystemC-описания. Во всех случаях использование SystemC как единой среды разработки даёт преимущества в тестировании системы. Одна и та же модель ошибок может быть применена на всех уровнях абстракции, позволяя тем самым на всех фазах проектирования использовать одну стратегию генерации тестов. Более того, одна стратегия может быть применена к тестированию как аппаратных, так и программных частей встроенной системы. Эта техника тестирования генерирует тесты, которые позволяют проверять корректность перевода описания системы с одного уровня на другой, от системного до структурного уровня.

Предложенная техника тестирования была показана на примере разработки встроенного ЦП и его интеграции со встроенным ПО. Во время всего цикла тестирования тесты постоянно дополняются и улучшаются с использованием одной и той же модели ошибок и техники генерации тестов. В дальнейшем планируется повысить эффективность этой методики, чтобы анализировать более сложные системы.

Литература

1. SystemC User's Guide. Synopsys, CoWare, Frontier Design. Version 1.1. 2000.
2. Myers G.J. The Art of Software Testing. Wiley–Interscience. New York. 1979.
3. Santos M.B., Gonalves F.M., Teixeira I.M. and Teixeira J.P. RTL-bases Functional Test Generation for High Defect Coverage in Digital SOCs. Proc. IEEE European Test Workshop (ETW). 2000. P. 99–104.
4. Vemuri R. and Kalyanaraman R. Generation of Design Verification Tests from Behavioral VHDL Programs Using Path Enumeration and Constraint Programming. IEEE Trans. On VLSI Systems, 3 (2): 201–214, June 1995.
5. Corno F., M. Sonza Reorda, Squillero G. High-Level Observability for Effective High-Level ATPG. Proc. IEEE VLSI Test Symp. 2000.
6. Follah F., Devadas S. and Keutzer K. OCCOM: Efficient Computation of Observability-Based Code Coverage Metrics for Functional Verification. Proc. IEEE DAC. 1998. P. 152–157.
7. Ferrandi F., Fin A., Fummi F. and Sciuto D. An Application of Genetic Algorithms and BDDs to Functional Testing. Proc. IEEE Int. Conf. on Computer Design (ICCD). 2000. P. 48–56.
8. Rudnick E.M., Patel J.H., Greenstein G.S. and Niermann T.M. A Genetic Algorithm Framework for Test Generation. IEEE Trans. Computer-Aided Design. Sept. 1997. Vol. 16. № 9. P. 1034–1044.
9. Hsiao M.S., Rudnick E.M. and Patel J.H. Application of Genetically Engineered Finite-State-Machine Sequences to Sequential Circuit ATPG. IEEE Trans. Computer-Aided Design. March 1998. Vol. 17, № 3. P. 239–254.
10. Agliada N., Fin A., Fummi F. and Martignano M. On the Reuse of VHDL Modules into SystemC Designs. Submitted.
11. Ferrandi F., Fummi F., Gerli L. and Sciuto D. Symbolic Functional Vector Generation for VHDL Specifications. Proc. IEEE Design Automation and Test in Europe Conf. (DATE). 1999. P. 442–446.