Методология проверки функционального покрытия при верификации контроля потока данных в системах на кристалле с использованием SystemVerilog и на примерах интерфейсов с протоколом AXI-Stream

М.А. Латыпов, А. А. Стародумов, Р. Ф. Ильясов

АНО ВО «Университет Иннополис», г. Иннополис m.latypov@innopolis.university,
a.starodumov@innopolis.university, r.iliasov@innopolis.ru

Аннотация — Функциональная верификация является одной из самых важных частей процесса разработки цифровых микросхем в последнее время. Причиной этому служат необходимость отслеживать внутреннюю работу системы, возможность изменить проект и, наконец, постоянно увеличивающаяся сложность цифровых устройств, что приводит к росту вероятности возникновения ошибок. В данной статье поднимается проблема методологии проверки функционального покрытия в системах на кристалле в соответствии с протоколом AXI-Stream.

Ключевые слова — функциональная верификация, функциональное покрытие, AXI-Stream, тестовая ситуация, handshake, clock, covergroup.

I. ВВЕДЕНИЕ

В настоящее время функциональная верификация необходима при дизайне микроэлектронных систем. Задача функциональной верификации состоит в том, чтобы определить, какие входные шаблоны предоставить проекту и каков ожидаемый результат правильно работающего проекта, когда он представлен этими входными шаблонами. Причем должна быть возможность изменить как входные шаблоны, так и сам проект, чтобы можно было устранить возникающие ошибки [1]. Иными словами, цель функциональной верификации в том, чтобы доказать, что проект будет работать так, как задумано [2]. Для достижения этой цели существует четыре важных момента: определить конкретную цель; понять, что делает дизайн-код; сравнить цель и дизайн-код, чтобы убедиться, что они совпадают; и оценить уровень уверенности в усилиях по верификации [3].

Функциональное покрытие кода — это технология, которая может определить, какой код был (и, что более важно, не был) выполнен в проверяемом проекте. В то время как покрытие кода измеряет, какая часть кода была реализована, функциональное покрытие измеряет, какая часть исходной спецификации проекта была реализована. Эта технология уже довольно давно используется в разработке программного обеспечения [4].

Рис. 1. Защита от неправильного толкования неоднозначного документа спецификации [5]

Проблема с дизайном, содержащим неизвестную ошибку, заключается в том, что сложно выявить недочеты и уязвимости в дизайне [6]. Невозможно с полной уверенностью сказать, что проверяемая конструкция действительно функционально верна [7].

Ошибочно полагаться на одно лишь покрытие кода, ведь даже 100-процентное покрытие кода не гарантирует отсутствие ошибок [8]. Высокое функциональное покрытие не обязательно сходится с высоким покрытием кода. В то время как покрытие кода связано с записью механики выполнения кода, функциональное покрытие связано с намерением или целью реализованной функции. Чтобы покрытие кода сообщило о пробеле, неиспользованный код должен существовать заранее. Функциональное покрытие особенно важно при тестировании, в котором большое количество случайных “сценариев”, но неизвестно, были ли покрыты все интересные “сценарии”.

II. AXI-STREAM

Протокол AXI-Stream был представлен в 2003 году и используется как стандартный интерфейс для связи компонентов, которые хотят обмениваться данными [9]. Существуют одни или более ведущих (master) устройств, которые генерируют данные, и ведомые устройства (slave), которые эти данные принимают. Протокол AXI-Stream поддерживает несколько потоков данных с использованием одного и того же набора общих шин. Это позволяет создать общее межсоединение, которое может выполнять операции downsizing, upsizing и packing. Протокол потока определяет связь между передачами и пакетами.

Протокол AXI-Stream используется для общения процессорного ядра с памятью, для обработки потока

видео. Также с помощью этого протокола можно загружать коэффициенты в блок матричного множителя нейросетей [10].

Рис. 2. Функциональная схема AXI-Stream [11]

В протоколе AXI-Stream существует базовый механизм handshake, состоящий из сигналов VALID и READY. Сигнал VALID управляется источником, чтобы сообщить приемнику, что полезная нагрузка на канале действительна и может быть считана, начиная с этого тактового цикла. Точно так же сигнал READY запускается приемником, чтобы уведомить, что он готов к приему данных.

Когда оба сигнала VALID и READY активны (равны 1) на положительном фронте тактового сигнала, полезная нагрузка данных считается переданной, и источник может либо предоставить новую полезную нагрузку данных, поддерживая VALID в положительном фронте clock, либо прекратить передачу, отменив VALID.

Для управления этими сигналами определены два основных правила:

- 1) Источник не должен ждать, пока READY будет на положительном фронте clock для подтверждения VALID.
- 2) После утверждения источник должен поддерживать VALID в положительном фронте clock до тех пор, пока не произойдет handshake.

Благодаря этому механизму и источник, и приемник могут управлять потоком данных, при необходимости регулируя пропускную способность.

Downsizing — это преобразование исходной ширины шины данных в более узкую ширину шины данных. Этот процесс обычно включает в себя создание нескольких выходных передач для одной входной передачи.

Рис. 3. Модуль Downsizing

Upsizing — это преобразование данной ширины шины данных в более широкую.

Рис. 4. Модуль Upsizing

Packing — это процесс формирования выходного пакета данных в соответствии с битами in_tkeep.

Рис. 5. Модуль Packing

III. SYSTEM VERILOG

В конце 1990-х годов Verilog Hardware Description Language (HDL) стал наиболее широко используемым языком для описания оборудования для моделирования и синтеза. В ноябре 2005 года был принят стандарт IEEE P1800-2005 для SystemVerilog. SystemVerilog — язык описания и верификации аппаратуры, являющийся расширением языка Verilog HDL [12].

Главным элементом функционального покрытия в System Verilog является cover point (точка покрытия). Группа из нескольких точек покрытия образует covergroup. Каждая точка покрытия фиксирует событие и/или комбинацию сигналов, состояние переменных или транзакций. Каждое состояние может фиксироваться либо на фронте тактового сигнала, либо в любой выбранный инженером момент времени. Конструкция bin определяет хранилище, в которое заносятся различные события. Событие происходит это значение равно одному из величин, определенных для этого хранилища, в его описании. Набор значений или переходов, связанных с точкой покрытия, можно пометить как недопустимый, указав их как illegal_bins. Все значения или переходы, связанные с недопустимыми и игнорируемыми ячейками, исключаются из покрытия. Если происходит недопустимое значение или переход, выдается ошибка времени выполнения. В конце симуляции симулятор,

например Synopsys VSC, Mentor Questa и Cadence Xcelium, делает отчет о том, какое количество точек покрытия было покрыто.

Написанием covergroups могут заниматься как инженер-верификатор, который пишет тесты, так и отдельный более опытный инженер, проверяющий первого верификатора [13]. Суть подобной организации в том, что, если инженер-верификатор написал недостаточное количество тестов, т.е. его тесты не покрывают всю функциональную спецификацию, то второй инженер независимо обнаружит это и укажет на пробелы в покрытии [14], [15].

Рис. 6. Этапы работ над RTL - проектом

Выражаем благодарность Юрию Панчулу за предоставление этой диаграммы.

В качестве примера функционального покрытия можно рассмотреть функциональное покрытие операции upsizing:

```
i_vld_lower_o_vld_o_rdy: coverpoint
{ in_tvalid, out_tready, lower_bits, out_tvalid }
{
  bins      i_vld_o_rdy_lower_o_vld_0000 = { 4'b0000 };
  bins      i_vld_o_rdy_lower_o_vld_0001 = { 4'b0001 };
  bins      i_vld_o_rdy_lower_o_vld_0010 = { 4'b0010 };
  illegal_bins i_vld_o_rdy_lower_o_vld_0011 = { 4'b0011 };

  bins      i_vld_o_rdy_lower_o_vld_0100 = { 4'b0100 };
  bins      i_vld_o_rdy_lower_o_vld_0101 = { 4'b0101 };
  bins      i_vld_o_rdy_lower_o_vld_0110 = { 4'b0110 };
  illegal_bins i_vld_o_rdy_lower_o_vld_0111 = { 4'b0111 };

  bins      i_vld_o_rdy_lower_o_vld_1000 = { 4'b1000 };
  bins      i_vld_o_rdy_lower_o_vld_1001 = { 4'b1001 };
  bins      i_vld_o_rdy_lower_o_vld_1010 = { 4'b1010 };
  illegal_bins i_vld_o_rdy_lower_o_vld_1011 = { 4'b1011 };

  bins      i_vld_o_rdy_lower_o_vld_1100 = { 4'b1100 };
  bins      i_vld_o_rdy_lower_o_vld_1101 = { 4'b1101 };
  bins      i_vld_o_rdy_lower_o_vld_1110 = { 4'b1110 };
  illegal_bins i_vld_o_rdy_lower_o_vld_1111 = { 4'b1111 };
}
```

Summary for Variable i_vld_lower_o_vld_o_rdy

Covered bins

NAME	COUNT	AT LEAST
i_vld_o_rdy_lower_o_vld_1110	43	1
i_vld_o_rdy_lower_o_vld_1101	14	1
i_vld_o_rdy_lower_o_vld_1100	52	1
i_vld_o_rdy_lower_o_vld_1010	22	1
i_vld_o_rdy_lower_o_vld_1001	4	1
i_vld_o_rdy_lower_o_vld_1000	9	1
i_vld_o_rdy_lower_o_vld_0110	88	1
i_vld_o_rdy_lower_o_vld_0101	51	1
i_vld_o_rdy_lower_o_vld_0100	468	1
i_vld_o_rdy_lower_o_vld_0010	65	1
i_vld_o_rdy_lower_o_vld_0001	29	1
i_vld_o_rdy_lower_o_vld_0000	148	1

Рис. 7. Конструкция Bins

Excluded/Illegal bins

NAME	COUNT	STATUS
i_vld_o_rdy_lower_o_vld_1111	0	Illegal
i_vld_o_rdy_lower_o_vld_1011	0	Illegal
i_vld_o_rdy_lower_o_vld_0111	0	Illegal
i_vld_o_rdy_lower_o_vld_0011	0	Illegal

Ниже представлены 6 особых сценариев, в которых прямая указывает на момент времени, в котором эти сигналы приняли свои значения:

Рис. 8. Случай i_vld_o_rdy_lower_o_vld_0000

Рис. 9. Случай i_vld_o_rdy_lower_o_vld_0001

Рис. 10. Случай i_vld_o_rdy_lower_o_vld_0010

Не существует такого момента времени, в котором были бы такие комбинации сигналов:

in_tvalid = 0, out_tready = 0, lower_bits = 1, out_tvalid = 1

in_tvalid = 0, out_tready = 1, lower_bits = 1, out_tvalid = 1
in_tvalid = 1, out_tready = 0, lower_bits = 1, out_tvalid = 1
in_tvalid = 1, out_tready = 1, lower_bits = 1, out_tvalid = 1

Рис. 11. Случай i_vld_o_rdy_lower_o_vld_0100

Рис. 12. Случай i_vld_o_rdy_lower_o_vld_0101

Рис. 13. Случай i_vld_o_rdy_lower_o_vld_0110

Как видно на картинках выше, в данные моменты времени эти сигналы приняли значения, которые были покрыты соответствующими бинами (bins).

При попытке ввести бин, который описывает сценарий, в реальности не встречающийся, бин показывается непокрытым:

```
bins i_vld_o_rdy_lower_o_vld_0011 = { 4'b0011 };
Uncovered bins
```

NAME	COUNT	AT LEAST	NUMBER
i_vld_o_rdy_lower_o_vld_0011	0	1	1

Также процент покрытия уменьшается:

Tests

Total Coverage Summary
SCORE GROUP
99.15 99.15

Бины для переходов и crosses принимают следующий вид:

```
o_rdy_transitions: coverpoint out_tready
{
  bins ordy_010 = ( 0 => 1 => 0 );
  bins ordy_101 = ( 1 => 0 => 1 );
  bins ordy_1001 = ( 1 => 0 => 0 => 1 );
  bins ordy_101_or_1001 = ( 1 => 0 => 1 ), ( 1 => 0 => 0 => 1 );
  bins ordy_10001 = ( 1 => 0 [*3] => 1 );
  bins ordy_1_from_3_5_0_1 = ( 1 => 0 [*3:5] => 1 );
}

o_vld_rdy: cross out_tvalid, out_tready;
i_vld_rdy: cross in_tvalid, in_tready;
i_vld_rdy_lower_upper: cross i_vld_rdy, lower_bits;
o_vld_o_rdy_lower_upper: cross out_tvalid, out_tready, lower_bits
{
  illegal_bins illegal
  = bins of (lower_bits) intersect { 1 }
  && bins of (out_tvalid) intersect { 1 };
}
```

IV. ЗАКЛЮЧЕНИЕ

Методы функционального покрытия используются не только при проектировании электроники, но и при проектировании ПО встроенных систем. После введения функциональной верификации в System Verilog, написание функционального покрытия стало существенно легче. Поэтому многие компании внедрили эту технологию в процесс проектирования, что является обязательным для современных чипов, в которых количество транзисторов насчитываются миллиардами. Один непокрытый сценарий может привести к ошибке, требующей изготовления чипа на фабрике заново, что стоит компаниям больших финансовых потерь в связи с задержкой выпуска чипа и производственных затрат.

БЛАГОДАРНОСТИ

Выражаем благодарность Юрию Панчулу за помощь в постановке изучения микроархитектуры цифровых схем в Иннополисе с 2018 год.

ЛИТЕРАТУРА

- [1] Никитин С.А., Николаев А.В., Путря Ф.М., Неклюдов И.А. Автоматизация маршрута функциональной верификации на основе стандарта IP-XACT // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2020. Выпуск 4. С. 90-94
- [2] Хисамбеев И.Ш. Роль стохастического тестирования в функциональной верификации микропроцессоров // Программные продукты и системы. Выпуск 3 2012. С. 107-112
- [3] Andreas Meyer. Principles of Functional Verification // Elsevier Science (USA). 2003. P. 176-178
- [4] V. Jerinic, J. Langer, U. Heinkel and D. Muller, "New Methods and Coverage Metrics for Functional Verification," Proceedings of the Design Automation & Test in Europe Conference, 2006, pp. 1-6, doi: 10.1109/DATE.2006.243901
- [5] Janick Bergeron. Writing Testbenches using SystemVerilog // Springer Science+Business Media. 2006. P. 49-50
- [6] F. Corno, E. Sanchez, M. S. Reorda and G. Squillero, "Automatic test program generation: a case study," in IEEE Design & Test of Computers, vol. 21, no. 2, pp. 102-109, March-April 2004, doi: 10.1109/MDT.2004.1277902
- [7] Andrew Piziali. Functional verification coverage measurement and analysis // Kluwer Academic Publishers. 2004. P 39-45
- [8] Хисамбеев И.Ш., Чибисов П.А. Об одном методе построения метрик функционального покрытия в тестировании микропроцессоров // Проблемы разработки перспективных микро- и наноэлектронных систем - 2014. Сборник трудов / под общ. ред. академика РАН А.Л. Стемпковского. М.: ИППМ РАН, 2014. Часть 2. С. 63-68

- [9] AXI reference guide. UG761 (v13.1) March 7, 2011
- [10] AMBA AXI and ACE Protocol Specification ARM IHI 0022D (ID102711)// 2003, 2004, 2010, 2011
- [11] Introduction to AMBA AXI. 102202 Issue 03.2021 Arm Limited.
- [12] SystemVerilog for Verification: A Guide to Learning the Testbench Language Features // Springer Science+Business Media. 2006. P. 241-250
- [13] Y.-S. Kwon and C.-M. Kyung. Functional Coverage Metric Generation from Temporal Event Relation Graph. In Design, Automation and Test in Europe (DATE) 2004, pages 670–671. IEEE Computer Society, IEEE, Feb. 2004.
- [14] O. Lachish, E. Marcus, S. Ur, and A. Ziv. Hole Analysis for Functional Coverage Data. Design Automation Conference, DAC:807–812, June 2002.
- [15] Чибисов, П. А. Встречное тестирование высокопроизводительных микропроцессоров: - Москва, 2013.- 174 с.: ил. РГБ ОД, 61 14-5/69

Functional Coverage Verification Methodology for Data Flow Control Verification in Systems on a Chip Using SystemVerilog and Examples of the Interface with the AXI-Stream Protocol

M.A. Latypov, A. A. Starodumov, R. F. Iliasov

"University Innopolis", Innopolis

m.latypov@innopolis.university, a.starodumov@innopolis.university, r.iliasov@innopolis.ru

Abstract — Functional verification is the process of demonstrating the functional correctness of a design with respect to the design specifications. Functional verification does not confirm the correctness of the design specification and assumes that the design specification is correct. Functional coverage is a technology that measures how much of the original design specification has been implemented. Functional coverage has been used in software development for quite some time. The article shows the essence of the functional coverage based on the interface AXI-Stream. Functional coverage techniques are used not only in electronics design, but also in embedded software design. Therefore, many companies have implemented this design process, which is mandatory for modern chips. One uncovered scenario could lead to an error requiring the chip to be re-fabricated at the factory, costing companies' financial losses in delayed chip release and manufacturing costs.

Keywords — functional verification, functional coverage, AXI-Stream.

REFERENCES

- [1] Nikitin S.A., Nikolaev A.V., Putrya F.M., Neklyudov I.A. Route automation of Functional Verification based on IP-XACT standard // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2020. Issue 4. P. 90-94. doi:10.31114/2078-7707-2020-4-90-94
- [2] Khisambeyev I.SH. Rol' stokhasticheskogo testirovaniya v funktsional'noy verifikatsii mikroprotssessorov // Programmnyye produkty i sistemy. Vypusk 3 2012. S.107-112.
- [3] Andreas Meyer. Principles of Functional Verification // Elsevier Science (USA). 2003. P. 176-178
- [4] V. Jerinic, J. Langer, U. Heinkel and D. Muller, "New Methods and Coverage Metrics for Functional Verification," Proceedings of the Design Automation & Test in Europe Conference, 2006, pp. 1-6, doi: 10.1109/DATE.2006.243901
- [5] Janick Bergeron. Writing Testbenches using SystemVerilog // Springer Science+Business Media. 2006. P. 49-50
- [6] F. Corno, E. Sanchez, M. S. Reorda and G. Squillero, "Automatic test program generation: a case study," in IEEE Design & Test of Computers, vol. 21, no. 2, pp. 102-109, March-April 2004, doi: 10.1109/MDT.2004.1277902
- [7] Andrew Piziali. Functional verification coverage measurement and analysis // Kluwer Academic Publishers. 2004. P 39-45
- [8] Khisambeyev I.Sh., Chibisov P.A. On one method of defining functional coverage metrics for microprocessor testing // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2014. Proceedings / edited by A. Stempkovsky, Moscow, IPPM RAS, 2014. Part 2. P. 63-68. (in Russian).
- [9] AXI reference guide. UG761 (v13.1) March 7, 2011
- [10] AMBA AXI and ACE Protocol Specification ARM IHI 0022D (ID102711)// 2003, 2004, 2010, 2011
- [11] Introduction to AMBA AXI. 102202 Issue 03.2021 Arm Limited.
- [12] SystemVerilog for Verification: A Guide to Learning the Testbench Language Features // Springer Science+Business Media. 2006. P. 241-250
- [13] Y.-S. Kwon and C.-M. Kyung. Functional Coverage Metric Generation from Temporal Event Relation Graph. In Design, Automation and Test in Europe (DATE) 2004, pages 670–671. IEEE Computer Society, IEEE, Feb. 2004.
- [14] O. Lachish, E. Marcus, S. Ur, and A. Ziv. Hole Analysis for Functional Coverage Data. Design Automation Conference, DAC:807–812, June 2002.
- [15] Chibisov, P.A. Vstrechnoye testirovaniye vysokoproizvoditel'nykh mikroprotssessorov- Moscow, 2013.- 174 P.: il. RGB OD, 61 14-5/69 (in Russian).