УДК 004.4'244

# МЕТОДИКА СИНТЕЗА СПЕЦИФИКАЦИЙ НА ЯЗЫКЕ UML ДЛЯ ВЕРИФИКАЦИИ ОБМЕНА ДАННЫМИ В АППАРАТУРЕ

***А. В. Березкин,***
*магистр техники и технологии, аспирант*
***А. С. Филиппов,***
*канд. техн. наук, доцент*
*Санкт-Петербургский государственный политехнический университет*

*Разработанная ранее авторами методика синтеза спецификаций на UML для тестирования потока управления позволяет определять поведение устройства на уровне последовательности управляющих сигналов. Созданные в соответствии с ней спецификации могут быть преобразованы в наблюдающие тесты, которые проверяют соответствие спецификаций реальному поведению устройства. В настоящей статье методика развивается и дополняется для реализации возможности специфицирования и тестирования потока данных в аппаратных системах.*

***Ключевые слова*** *— UML, моделирование аппаратуры, верификация, тестирование.*

## Введение

Унифицированный язык моделирования (Unified Modeling Language — UML) в настоящее время часто используется для проектирования и моделирования вычислительных систем. Созданный для проектирования программного обеспечения, UML успешно применяется и в других областях, в том числе в области проектирования аппаратуры.

В статье [1] предложена методика, дающая возможность с помощью UML создавать спецификации аппаратного обеспечения. Полученные спецификации являются, с одной стороны, рабочим документом проектирования, с другой — формальными описаниями, позволяющими преобразовывать их в тесты системы, которые проверяют ее функциональность на предмет соответствия ранее созданной спецификации.

Благодаря разработанной методике можно описывать поведение систем и модулей в терминах логических связей между внутренними и внешними управляющими сигналами (причина и следствие). Для создания этих описаний используется подмножество диаграмм UML: диаграмма классов и диаграмма последовательности. Эти диаграммы, представляющие собой формальные спецификации, являются материалом для автоматизированной генерации тестов. Разработан ряд правил для генерации из таких диаграмм тестов в конкретном формате — VHDL Testbench.

Задача данной работы — дополнить методику новыми правилами таким образом, чтобы можно было специфицировать и тестировать не только управляющие сигналы, но и данные.

Для того чтобы разработать правила описания потока данных в спецификациях, необходимо выявить общие признаки организации обмена данными в цифровых устройствах. Эти признаки напрямую влияют на способ описания потоков данных с помощью диаграмм UML.

## Признаки организации обмена данными в цифровых устройствах

Обработка данных в цифровых устройствах состоит в их копировании с преобразованием или без него. Копирование данных можно условно разделить на два шага: 1) выставление данных отправителем и 2) фиксация данных получателем. В различных устройствах эти действия выполняются по разным правилам. Набор правил для осуществления какого-либо обмена называют протоколом. При проектировании цифровых устройств разработчики определяют, по какому протоколу будут обмениваться данными узлы системы. Как правило, внутренние узлы обмениваются по простым протоколам (шина данных плюс строб), а внешние реализуют один из стандартных протоколов. Можно показать, что простые протоколы для внутреннего обмена — это сильно упрощенные стандартные протоколы.

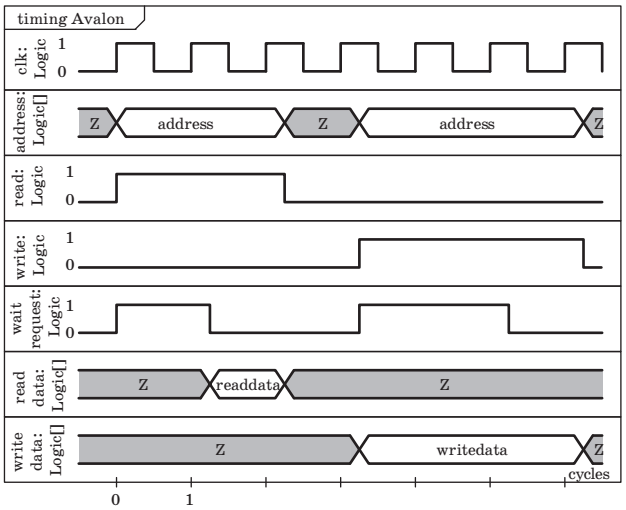
Учитывая все сказанное, можно сделать вывод, что для выделения общих признаков организации обмена данными необходимо рассмотреть несколько наиболее типичных стандартных протоколов передачи данных. Признаки, выделенные нами для них, будут актуальны и для внутренних специфических протоколов обмена данными.

Мы рассмотрим три несложных протокола — Altera Avalon, RS-232, Manchester II. Можно показать, что обмены данными в более сложных протоколах (таких как PCI, DDR, Ethernet) обладают примерно теми же признаками, однако их реализация будет существенно сложнее.

В процессе рассмотрения признаков мы будем выявлять функциональность, которой должно обладать верифицирующее устройство (верификатор) для контроля соответствующей части протокола обмена данными.

## Altera Avalon

Altera Avalon — простой параллельный интерфейс, используемый в системах на кристалле с процессором NIOS [2]. Интерфейс Avalon является шинным, он предназначен не просто для передачи данных, а для чтения и записи в адресное пространство целевого устройства. Устройства на шине Avalon могут быть ведущими и ведомыми. Первые могут только инициировать транзакции, вторые — только отвечать на запросы. Пример временной диаграммы, включающей чтение и запись, представлен на рис. 1. В примере тип `Logic` соответствует одиночному сигналу, а `Logic[]` — шине, т. е. массиву логических сигналов. Состояние Z — шина свободна (драйверы отключены), состояния `address`, `readdata` и `writedata` соответствуют передаче данных по шине.




■ ***Рис. 1.*** *Временные диаграммы чтения и записи на шине Avalon*

*Алгоритм работы ведомого устройства.*

1. Транзакция начинается в момент готовности слова для записи или в момент запроса чтения. В обоих случаях интерфейс запускается по сигналу внутреннего управляющего устройства.

2. В зависимости от вида транзакции (запись, чтение) выставляются разные стробы. В обоих случаях выставляется адрес. При записи также выставляются данные, которые читаются из внутренних регистров и передаются (возможно, с преобразованием) на внешние (т. е. управляющие драйверами линий шины).

3. При наличии сигнала `waitrequest` со стороны ведомого устройства данные удерживаются, в противном случае снимаются, линия освобождается.

*Алгоритм работы ведущего устройства.*

1. Транзакция начинается в момент получения внешнего управляющего сигнала `chipselect`.

2. Устройство захватывает адрес и в случае записи — данные, которые читаются с линии (или с буферов) и переписываются во внутренние регистры устройства (возможно, с преобразованием).

3. В случае необходимости устройство выставляет сигнал `waitrequest`.

4. В случае чтения устройство выставляет данные, т. е. переписывает их из внутренних регистров во внешние.

Выделим основные принципы выставления и захвата данных с точки зрения того, как контролировать эти процессы.

- Выставление и захват данных выполняются по управляющим сигналам. Следовательно, верификатор данных должен управляться этими же сигналами.
- Управляющие сигналы могут иметь различную длину, однако «рабочей» частью этого сигнала является только его фронт. Следовательно, верификатор должен пропускать управляющие сигналы через выявитель фронта.
- Выставление и захват данных могут выполняться как на следующий такт после прихода управляющего сигнала, так и через определенное количество тактов после этого.
- Захват данных состоит в переписывании информации с линии во внутренние регистры, выставление данных — тоже переписывание, но в другую сторону. Следовательно, проверка этого процесса состоит в сравнении значения на линии со значением на внутреннем регистре.

## RS-232

RS-232 — один из самых простых протоколов передачи данных [3]. В наиболее простом варианте интерфейс состоит из двух каналов, которые обеспечивают дуплексную передачу данных. Рас-

смотрим пример передачи одного байта по каналу. Временная диаграмма приведена на рис. 2. В примере приняты следующие параметры передачи: 8 информационных бит, контроль четности выключен, один стоповый бит. На диаграмме `tx` — это информационный сигнал, `st` — внутреннее состояние передатчика, приводится здесь для пояснения фаз передачи (ожидание, стартовый бит и т. д.).

С точки зрения *передатчика* процесс происходит в следующем порядке.

1. В момент готовности слова для передачи соответствующий управляющий сигнал инициирует передачу.

2. Если в передатчике отсутствует внешнее управление скоростью передачи, запускается таймер для отсчета интервалов времени, соответствующих передаче одного бита. В этот же момент, возможно, выставляется стартовый бит.

3. В моменты срабатывания таймера (или по внешним сигналам) на линию данных выставляется очередной бит слова данных, либо бит контроля четности, либо стоповый бит. Выбор происходит либо путем подсчета количества срабатывания сигналов, либо путем активизации другого (или вспомогательного) управляющего сигнала.

С точки зрения *приемника* прием происходит в следующем порядке.

1. Приемник ожидает перепада 1 → 0 на информационной линии.

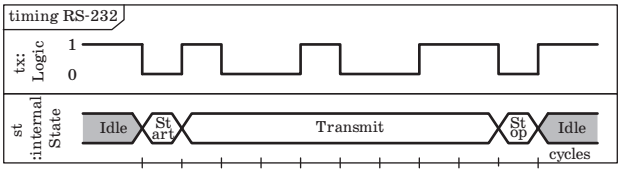
2. Запускается таймер (или настраивается внешний источник сигналов), который выдает импульсы в середине временного интервала, соответствующего очередному принимаемому биту.

3. При поступлении сигнала таймера принимается очередной бит данных, либо бит контроля четности, либо стоповый бит.

Выделим особенности передачи данных, дополняющие указанные выше.

• Значение выставляемого или принимаемого бита зависит от количества управляющих сигналов, пришедших до данного момента времени. Следовательно, проверяющие устройства должны подсчитывать пришедшие управляющие сигналы.

• В моменты начала и окончания транзакции верифицирующие устройства должны очистить все свои внутренние регистры истории.




■ ***Рис. 2.*** *Пример временной диаграмммы RS-232*

## Manchester II

Manchester II, как и RS-232, является последовательным протоколом, и канал связи в простейшем случае также состоит из трех линий [4]. Manchester II является самосинхронизирующимся протоколом, что позволяет передавать за одну транзакцию большое число бит. На рис. 3 приведен пример передачи 8 бит. Показаны следующие сигналы:

`clk` — тактовая частота передающего устройства (внутренний сигнал);

`bytes` — передаваемые байты (внутренний сигнал);

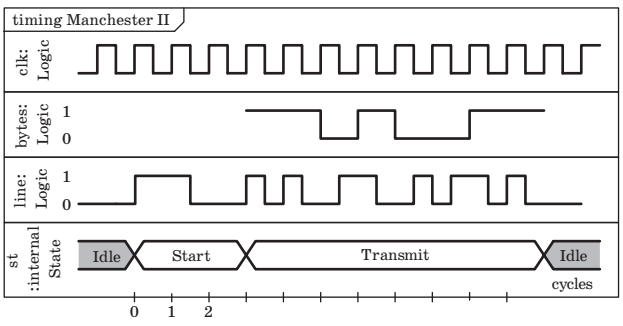
`line` — информационный сигнал (внешний сигнал, передается на шину);

`st` — состояние передатчика (внутренний сигнал).

Как правило, передача начинается со стартовой последовательности. Далее один информационный бит представляется последовательностью из двух равных по длительности сигналов («полубитов»), первый из которых соответствует информационному биту, второй — его инверсии (в некоторых случаях может использоваться обратный порядок полубитов).

Алгоритм передачи данных схож с рассмотренным в предыдущем разделе: сначала выставляется стартовая последовательность для синхронизации, затем информационные биты, затем (возможно) бит (или биты) контроля ошибок. Однако выставление данных имеет свою особенность: в середине интервала передачи бита происходит инвертирование информационного сигнала на линии (перепад синхронизации). Для этого может использоваться дополнительный таймер. То есть инверсия также выполняется под влиянием управляющего сигнала, но конечное состояние линии зависит не только от него, но и от предыдущего состояния самой линии. Таким образом:

• верифицирующее устройство должно иметь доступ к предыстории не только управляющих сигналов, но и линий данных.




■ ***Рис. 3.*** *Пример временной диаграммы Manchester II*

Прием данных также схож с алгоритмом RS-232, но обладает своей особенностью: таймер приема перезапускается (подстраивается) каждый раз при получении синхронизирующего перепада. Однако это не влияет на прием данных, которые принимаются так же, по сигналу таймера.

## Принципы верификации обменов данными

На основе анализа стандартных протоколов были выявлены признаки передачи данных. Эти признаки обуславливают также и правила верификации обменов данными. Здесь мы представим признаки в виде единого списка и рассмотрим, какие элементы должны присутствовать в спецификациях устройств для реализации их верификации (таблица).

Перечисленные принципы верификации данных реализуются и в других технологиях верификации, например таких, как языки OpenVera [5] и System Verilog [6]. Данные языки позволяют определять события (events), наступление которых запускает ту или иную проверку (assertion). Оба языка имеют достаточно богатый инструментарий для определения событий: поддерживаются простые изменения уровня сигнала, более сложные условия, операции, темпоральная логика, операции с данными. Если проводить аналогии с предлагаемым нами подходом, то определение событий и проверок в OpenVera и System Verilog эквивалентно элементам диаграмм последовательности в нашей методике. Действительно, в обоих случаях задается эталонная последовательность управляющих сигналов, выполнение которой можно проверить и которую можно использовать для проверки данных. Наличие таких аналогий позволяет говорить о том, что предлагаемое решение не противоречит имеющимся на данный момент технологиям, но вместе с тем и не копирует их.

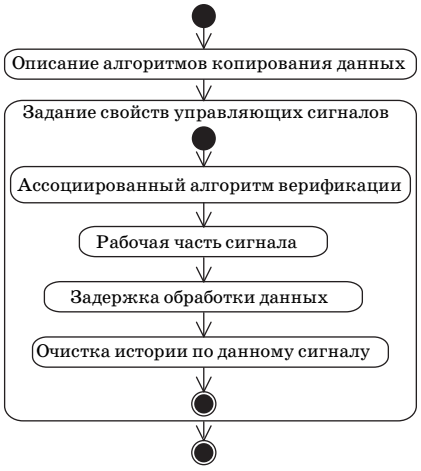
Основная особенность (и отличие) предлагаемого нами подхода состоит в том, что в нем тесты генерируются непосредственно из рабочих спецификаций системы. В случае же с OpenVera и System Verilog необходимо помимо рабочей спецификации создать дополнительный документ, описывающий поведение системы.

## Описание потоков данных с помощью элементов UML

Выше были сформулированы принципы верификации данных и каждому из них были сопоставлены элементы спецификации устройств, необходимые для реализации этих принципов при генерации тестов данных. Рассмотрим, как отражать эти элементы в спецификациях UML. Схе-

■ *Обмен данными в цифровых устройствах и их верификация*

| Признаки обменов данными | Принципы верификации | Элементы спецификации |
|---|---|---|
| 1. Захват и выставление данных состоят в их переписывании (возможно, с преобразованием) | Верификация состоит в сравнении исходных и переписанных данных | Необходимо иметь возможность задавать алгоритм копирования данных. В простейшем случае этот алгоритм будет состоять из одной операции |
| 2. Захват и выставление данных выполняются по управляющим сигналам | При получении управляющего сигнала верификатор должен запустить заданный алгоритм сравнения | Необходимо иметь возможность определять свойства сигнала: какие алгоритмы сравнения должны быть запущены при активации этого сигнала |
| 3. «Рабочая» часть управляющего сигнала может быть короче, чем сам сигнал | Алгоритмы сравнения должны запускаться только в «рабочей» фазе сигнала | Необходимо иметь возможность определять свойство сигнала — его рабочую часть. Например: фронт, спад, любой перепад, весь сигнал |
| 4. Передача данных может выполняться через несколько тактов после прихода управляющего сигнала (с задержкой, заложенной в спецификацию) | Верификатор должен задерживать необходимые управляющие сигналы | Необходимо иметь возможность определять свойство сигнала: количество тактов с момента его прихода до запуска алгоритма проверки |
| 5. Выставляемые данные могут зависеть от предыстории управляющих сигналов | 5.1. Верификатор должен вести историю (подсчет) управляющих сигналов<br>5.2. В определенные моменты историю следует очищать | 5.1. Конечный автомат ведения истории должен генерироваться на основании перечня управляющих сигналов. Верификаторы должны иметь возможность обращения к выходным сигналам автомата<br>5.2. Следует иметь возможность определять свойство сигнала: необходимо ли по нему сбрасывать историю |
| 6. Выставляемые данные могут зависеть от предыдущих состояний линии данных | Верификатор должен задерживать необходимые сигналы данных | Необходимо ввести лингвистические соглашения для обращения к задержанным данным в алгоритмах сравнения |



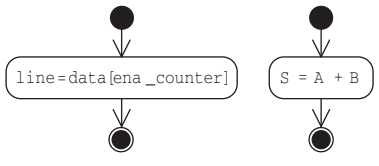

■ ***Рис. 4.*** *Схема алгоритма описания потоков данных на диаграммах UML*

ма алгоритма описания потоков данных приведена на рис. 4. Пояснения к этапам даны ниже.

**Описание алгоритма копирования данных (п. 1, 5.1, 6 из таблицы)**

Алгоритм копирования данных позволяет синтезировать ядро верифицирующего устройства, с помощью которого можно определить, как произошел обмен: правильно или с ошибкой. Ядро верификатора отслеживает шаги алгоритма, сравнивает его ожидаемые результаты с реальными и на основании этого сравнения делает вывод о правильности работы тестируемой системы. Сложность алгоритма и верификатора зависит от специфики устройства. Например, в последовательном передатчике алгоритм копирования сводится к выставлению очередного разряда исходного слова (рис. 5, слева), а верификатор, соответственно, выполняет одну операцию сравнения одноразрядных сигналов. Если проверяется более сложное устройство, такое как сумматор, алгоритм состоит из более сложной операции (однако, также единственной) (рис. 5, справа), а проверка сводится к самостоятельному суммированию операндов и сравнению результата, полученного на сумматоре, с результатом верификатора.

В UML существует средство для явного описания алгоритма — диаграмма деятельности. Алгоритм копирования данных целесообразно описать с помощью этой диаграммы.




■ ***Рис. 5.*** *Примеры диаграмм деятельности, задающие алгоритмы копирования данных*

Следует помнить, что тесты генерируются на каком-то определенном языке описания тестов (например, VHDL Testbench), поэтому на алгоритм накладываются ограничения с тем, чтобы сделать его синтезируемым в рамках выбранного языка.

• Присваивания и вообще любые операции должны выполняться с использованием тех операций, которые приняты в выбранном языке описания тестов аппаратуры.

• Необходимо обращаться только к сигналам, доступным в анализируемом устройстве.

• Допускается также обращение к специальным переменным — счетчикам управляющих сигналов и задержанным линиям данных.

Верификатор может обращаться к специальным переменным — значениям внутренних счетчиков и задержанным информационным сигналам. Данные сигналы должны добавляться в верификатор автоматически в случае их упоминания в диаграмме деятельности. Для идентификации специальных переменных следует ввести лингвистические соглашения, например:

• имя сигнала с суффиксом `_latched_`*n* (*n* — натуральное число) означает соответствующий сигнал, задержанный на *n* тактов;

• имя управляющего сигнала с суффиксом `_counter` означает счетчик управляющего сигнала, т. е. количество активаций этого сигнала с момента последнего сброса истории.

**Задание свойств управляющих сигналов (п. 2–4, 5.2 из таблицы)**

Некоторые параметры верификации можно задать с помощью опций сигналов, фигурирующих на диаграмме последовательности. Спецификация UML не поддерживает задания произвольных свойств сигнала напрямую, однако многие UML-редакторы позволяют назначать сигналам именованные значения, например Tagged Values в Microsoft Visio. Для верной интерпретации этих значений необходимо принять лингвистические соглашения относительно их имен и значений, например:

`VerificationAlg` — ассоциированный алгоритм верификации, значение — имя диаграммы деятельности;

`ActivePart` — рабочая часть сигнала, значения:

`PositiveEdge`, `NegativeEdge` — положительный и отрицательный перепады;

`Edge` — любой перепад;

`Whole` — весь сигнал;

`Latency` — количество тактов с момента активации сигнала до запуска алгоритма проверки, значение — целое неотрицательное число;

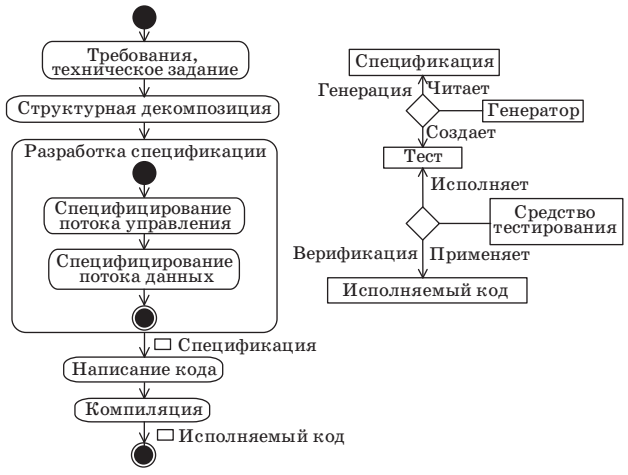
`ClearHistory` — очистить историю управляющих сигналов данной диаграммы последовательности, значения: `True` или `False` (по умолчанию).

### Применение методики

Методика применяется совместно с методикой из работы [1], расширяя ее. Методика должна применяться дважды [1]: один раз для построения тестопригодной спецификации системы, другой — для создания средства преобразования этой спецификации в тест. Применение методики для создания спецификации полностью вписывается в маршрут проектирования ([1], рис. 1); этот этап выполняется на основании материала раздела «Описание потоков данных с помощью UML».

Полная схема применения методики представлена на рис. 6. Слева отображены этапы маршрута проектирования вместе с порождаемыми ими сущностями, справа — отношения этих сущностей с другими элементами процесса разработки систем. Элементы диаграммы, непосредственно относящиеся к применению разработанной методики:

- Разработка спецификации — этап, на котором применяется разработанная методика и в результате которого порождается UML-спецификация системы.
- Генератор — техническое средство для преобразования спецификаций в тесты. Рассмотрение деталей его реализации выходит за рамки данной статьи.
- Тест — сущность, генерируемая автоматически из спецификации системы.
- Верификация — исполнение теста, которое проверяет работу устройства. Осуществляется каким-либо средством имитационного тестирования (ModelSim, Quartus и т. д.) или в реальной системе с помощью встраиваемых средств тестирования.




■ ***Рис. 6.*** *Схема применения методики*

Результат, получаемый после применения методики, — возможность автоматического тестирования системы. Полнота тестирования полностью определяется глубиной и подробностью создаваемых спецификаций. Если при составлении спецификации были учтены все детали работы устройства, можно гарантировать 100-процентное покрытие тестов при достаточном наборе входных воздействий.

### Заключение

В данной статье была предложена методика синтеза UML-спецификаций потоков данных в аппаратуре, основанная на общих признаках организации обмена данными в цифровых устройствах, позволяющая генерировать из этих спецификаций наблюдающие тесты потоков данных в аппаратных системах для проверки соответствия спроектированных устройств созданным ранее спецификациям.

Создаваемые в рамках методики спецификации основаны на спецификациях управляющих сигналов, которые описываются в соответствии с методикой, представленной в работе [1]. Применение обеих методик позволит создать тесты систем, которые проверяют как управляющие сигналы, так и данные. При высокой детализации спецификаций и достаточном наборе входных воздействий можно гарантировать 100-процентное покрытие тестов.

Применение методик полностью вписывается в маршрут проектирования цифровых устройств, становясь его частью. В данный момент разрабатывается техническое средство для реализации методик.

### Литература


1. **Березкин А. В., Филиппов А. С.** Методика синтеза тестов аппаратуры по спецификациям на языке UML // Информационно-управляющие системы. 2010. № 5. С. 24–30.
2. **Avalon** Interface Specification // Altera Website. http://www.altera.com/literature/manual/mnl_avalon_spec.pdf (дата обращения: 07.12.2010).
3. **RS-232** Bus Description and EIA232 Pinout // interfacebus.com website. http://www.interfacebus.com/Design_Connector_RS232.html (дата обращения: 07.12.2010).
4. **ГОСТ** Р 52070–2003. Интерфейс магистральный последовательный системы электронных модулей.
5. **OpenVera** Website. http://www.open-vera.com/ (дата обращения: 07.12.2010).
6. **SystemVerilog** Website. http://www.systemverilog.org/ (дата обращения: 07.12.2010).