УДК 004.91

Спецификация и валидация протоколов систем на кристалле: проблемы и пути их решения

И.С. Печенко

ЗАО «Интел А/О», ivan.pechenko@intel.com

***Аннотация* — Процесс проектирования систем на кристалле предполагает, что архитектурные спецификации создаются в основном в виде текстовых описаний на естественном языке, а пригодные для верификации модели появляются на более поздних этапах проектирования. Однако с усложнением архитектуры этих систем возникает необходимость формального анализа спецификаций по крайней мере одной из составляющих их архитектуры – протоколов взаимодействия между компонентами системы. В данной работе представлен метод создания и дальнейшего анализа спецификаций протоколов систем на кристалле в форме диаграмм последовательности операций. Целью этого метода является повышение качества спецификаций протоколов систем на кристалле и предоставление возможности их верификации на ранних стадиях проектирования.**

***Ключевые слова* — система на кристалле, процесс проектирования, архитектура, язык системного уровня, протокол, верификация, BPMN, диаграмма, система состояний и переходов.**

I. Введение

Определим для начала понятие «система на кристалле». Под системой на кристалле обычно понимают вычислительную систему, на едином кристалле которой интегрированы все основные необходимые элементы: процессор (нередко программируемый), необходимый объём памяти, а также интерфейсы к периферийным устройствам и другие функциональные устройства [1]. Важнейшей особенностью систем на кристалле является важная роль встроенного программного обеспечения в их конструкции.

С тех пор как два десятилетия назад прогресс в технологии производства микроэлектроники позволил уместить всю вычислительную систему на едином кристалле, проектировщики вычислительных систем не могли не воспользоваться возможностью интегрировать конечный продукт в единый чип, чтобы повысить производительность системы, снизить энергопотребление, размер продукта и в конечном итоге его цену. Эти преимущества особенно актуальны в области портативной электроники, мультимедиа и телекоммуникации.

Современные системы на кристалле включают в себя десятки так называемых IP-блоков – готовых сложных функциональных компонентов для проектирования микросхем.

Процесс проектирования системы на кристалле может быть формально разделен на несколько стадий: планирование, разработка спецификации архитектуры, разработка микроархитектуры, описание системы на уровне регистровых передач (RTL), а также логический и физический синтез.

Стадия планирования относится в большей мере к общей стратегии компании, нежели к конкретному проекту. На этой стадии разрабатываются и определяются основные технологии, которые будут использоваться при проектировании вычислительных систем, и решается, какие конечные продукты будут разрабатываться.

На стадии разработки спецификации архитектуры определяются основные требования к системе, её будущие характеристики и основные функциональные блоки.

На стадии проектирования микроархитектуры разрабатываются основные элементы каждого функционального блока с учётом требований к площади кристалла, энергопотреблению и прочих.

На следующем этапе создаётся RTL-код, детально описывающий потоки сигналов между аппаратными регистрами и логические операции над данными.

Последний этап включает в себя планирование топологии кристалла, логический синтез, проектирование систем питания и синхронизации, а также физический синтез – размещение элементов на схеме и трассировка связей между ними.

На данный момент большинство действий по валидации систем на кристалле проводятся уже после создания RTL-кода. Одной из главных причин этого является отсутствие полных и завершённых моделей даже отдельных аспектов структуры и поведения системы [2]. Однако описание на уровне регистровых передач является слишком низкоуровневым для того, чтобы в короткие сроки провести полную валидацию системы, содержащей сотни миллионов элементов. Поэтому в последние годы серьёзные усилия

прикладываются для того, чтобы перейти на более высокий уровень абстракции для описания и верификации систем. Требуется создать новые способы формального описания системы, причём как аппаратной её части, так и программного обеспечения, и разработать новую методологию создания и верификации такого описания на этапе создания спецификаций архитектуры и микроархитектуры системы. Это обеспечит возможность верификации архитектуры системы на самых ранних этапах её разработки, ведь, как известно, ошибки, допущенные на самых ранних этапах проектирования, «стоят» компании-разработчику дороже всего [3, 4, 5, 6].

Разработчики систем на кристалле столкнулись с проблемой выбора нового языка описания системного уровня и методологии проектирования и верификации с использованием этого языка. Любой подобный язык системного уровня должен обладать двумя основными качествами: иметь возможность объединять множество разнородных моделей и работать на высоком уровне абстракции.

В последнее время многие возможности были предложены для решения данной задачи [2]:

1) использование языков описания аппаратуры с некоторыми дополнениями, как, например, расширение языка Verilog до SystemVerilog.

2) Адаптация языков программирования высокого уровня (C/C++, Java) для описания архитектурных спецификаций систем на кристалле.

3) Создание принципиально новых языков системного уровня специально для проектирования систем на кристалле. Данный подход может привести к наилучшим результатам, но работа в этом направлении требует огромных усилий.

4) Расширение возможностей существующих языков высокоуровневого моделирования и применение их вместе с другими языками более низкого уровня для описания архитектурных спецификаций систем.

II. Недостатки процесса проектирования

На высоком уровне абстракции систему на кристалле можно представить как сеть связанных между собой аппаратных и программных компонентов (IP-блоков). Архитектурную спецификацию системы на кристалле можно условно разделить на две части: структурная спецификация и поведенческая спецификация. Первая представляет собой, во-первых, перечень компонентов системы, а во-вторых – описание базовых требований для каждого компонента. Вторая состоит из описания поведения системы в различных ситуациях и протоколов взаимодействия компонентов системы. В данной работе мы подробнее рассмотрим поведенческую спецификацию системы и возможности создания и верификации поведенческих моделей системы на кристалле.

Современные системы на кристалле чрезвычайно сложны, и к скорости их проектирования предъявляются очень жёсткие требования, особенно в мобильном сегменте. Для работы системы на кристалле требуется высокий уровень координации между аппаратным и программным обеспечением.

Большинство ключевых «инфраструктурных» протоколов в системах на кристалле включают в себя взаимодействие множества IP-блоков и подсистем и задействуют множество как аппаратных, так и программных компонент системы. Примерами таких протоколов являются протокол перезагрузки, алгоритмы управления питанием в системе, протоколы безопасности и так далее. Несмотря на то, что эти компоненты не видны конечным пользователям системы, они являются основой её успешного функционирования. Методы описания и верификации таких инфраструктурных компонентов вычислительных систем, применявшиеся ранее, теряют свои эффективность. У этой проблемы есть ряд причин [7]:

1) сложность архитектуры систем на кристалле часто ведёт к ошибкам в их спецификациях, которые бывает очень сложно исправить, особенно если их обнаруживают на поздних этапах проектирования. Из этого следует необходимость анализа архитектурных спецификаций. Однако жёсткие требования к скорости проектирования усложняют эту задачу.

2) Ключевая роль в функционировании инфраструктуры системы на кристалле, которую играет встроенное программное обеспечение, предполагает, что анализ и верификация компонентов системы должны учитывать как аппаратные ресурсы, так и программные. В современных же реалиях аппаратное и программное обеспечение системы зачастую разрабатывается разными командами с узкой специализацией, и их интеграция начинается только на поздних этапах проектирования системы. Эта ситуация ведёт к невозможности совместного анализа аппаратных и программных компонентов системы на ранних этапах проектирования.

3) Тот факт, что большинство сложных протоколов функционирования системы распределено между множеством компонентов, делает верификацию таких протоколов очень сложной. Дело не только в распределении функциональности между компонентами, но и в наличии у каждого из компонентов множества других функций. Поэтому при создании архитектурных спецификаций системы необходимо строго следить за инкапсуляцией компонентов системы и полным и строгим описанием их интерфейсов.

4) Архитектурные спецификации в большинстве своём написаны на естественных языках, и им присущи все недостатки этих языков: двусмысленность, неточность, неполнота. Это часто ведёт к отсутствию в спецификациях ключевых деталей, к различному пониманию спецификаций

различными командами, что, в свою очередь, ведёт к ошибкам в системе.

III. Существующие решения

Для решения проблем существующих методов создания и верификации архитектурных спецификаций системных протоколов предложено множество подходов.

В статье [8] описан процесс создания и верификации формальных моделей системной архитектуры. Авторы этой работы использовали модели, написанные на формальных языках, таких как TLA+, Murφ и HOL. Они работали параллельно с командами архитекторов и в тесной взаимосвязи с ними, разрабатывая формальные модели архитектуры параллельно с разработкой самой архитектуры. Такой подход привёл к значительному улучшению качества спецификации системы и устранению множества ошибок в ней, однако он потребовал значительных усилий трёх команд высококлассных экспертов в области анализа формальных моделей на этапе создания архитектуры новой системы, что не всегда возможно.

В статье [9] описан опыт применения формальных методов анализа к архитектурным компонентам различных систем, таким как протоколы когерентности кэшей, протоколы скользящего окна и так далее. Авторы приходят к выводу, что применение формальных методов на стадии создания архитектурных спецификаций имеет множество преимуществ: оно позволяет создавать спецификации, свободные от недостатков естественного языка, таких как двусмысленность, неполнота и неточность, позволяет на очень ранних этапах выявлять ошибки в спецификациях сложнейших системных протоколов и, наконец, позволяет уменьшить риск внесения ошибок при модификации этих протоколов.

В работах [2, 10] рассмотрена возможность совместного использования диаграмм UML и языков SystemC и C++ для создания спецификации системы на стадиях разработки архитектуры и микроархитектуры. В этой статье также показаны возможности по организации совместной верификации аппаратного и программного обеспечения на основе моделей аппаратных компонентов системы, описанных с помощью языков высокого уровня.

IV. Предлагаемое решение: диаграммы IFlow

В этой работе предлагается метод создания спецификаций протоколов для систем на кристалле на основе средств, предлагаемых библиотекой графических нотаций BPMN (Business Process Model and Notation) [11]. Предлагаемая в этой работе система под названием IFlow создана в рамках разработки интегрированной платформы ACES (Analyzable, Consistent, Efficient Specification) для проектирования архитектуры систем на кристалле, которая включает в себя средства создания как структурных, так и поведенческих спецификаций. Наличие такой платформы открывает широкие возможности для совместного использования таких спецификаций в целях создания более полных моделей системы, её анализа и валидации. Платформа ACES создана для решения промышленных задач и выполнена в виде расширения для Microsoft Office, являющегося основной платформой создания архитектурных спецификаций в компании Intel. Библиотека графических нотаций BPMN, на основе которой реализована система IFlow, является широко используемой библиотекой шаблонов Microsoft Visio, программы, входящей в пакет Microsoft Office, очень удобной для создания графических спецификаций.

Основной задачей нотаций BPMN является графическое описание бизнес-процессов, лёгкое в создании и понимании различными заинтересованными сторонами. Нотация BPMN является связующим звеном между проектированием бизнес-процесса и его реализацией. Аналогично этому система IFlow должна служить связующим звеном между фазой разработки архитектурной спецификации и фазами разработки и тестирования системы.

Отметим, что диаграммы IFlow несложно использовать для трансляции в формальные модели для их дальнейшей верификации.

Рис. 1. Запуск обработчика прерываний

Рассмотрим диаграмму IFlow, описывающую архитектуру протокола системы на кристалле. Можно представить её в виде ориентированного графа, чьи

вершины мы будем называть действиями и ассоциировать их с атомарными операциями, выполняемыми компонентами системы, включёнными в протокол. Такие компоненты мы будем называть агентами. Граф будет иметь два типа рёбер: рёбра типа «последовательность», соединяющие последовательные действия, выполняемые одним агентом, и рёбра типа «сообщение», соединяющие действия различных агентов и определяющие взаимодействие между агентами. Соответствие элементов диаграммы IFlow элементам нотации BPMN рассмотрено в работе [12]. На рис. 1 показан пример простейшего механизма запуска обработки прерывания.

Текст, ассоциированный с вершинами графа, будем считать псевдокодом и предположим, что он состоит из операций условного присваивания, каждое из которых имеет вид: если выполняется некое условие, описанное в виде логического выражения, то одной или нескольким переменным присваиваются определённые значения. Текст, ассоциированный с рёбрами графа, будем считать условиями перехода по этим рёбрам, представляющими собой логические выражения над переменными, описанными в других элементах диаграммы. Следует заметить, что переменные, встречающиеся в тексте каких-либо элементов диаграммы, должны быть определены, принадлежать одному из агентов и иметь некоторые заданные начальные значения. Предположим к тому же, что для каждой переменной определены права доступа к ней для каждого из агентов. Детали описания этих прав доступа могут различаться в зависимости от реализации.

Также необходимо отметить, что действия могут различаться по логике передачи управления. По умолчанию действие начинается, если любое из рёбер, входящих в него, активно и после своего завершения делает активными все исходящие рёбра. Однако существуют специальные типы вершин графа, не соблюдающие эти условия. Такими типами вершин являются блоки ветвления, передающие после завершения контроль только на одно ребро графа, и блоки объединения, становящиеся активными только после того, как становятся активными все входящие рёбра.

Рассмотрим теперь систему состояний и переходов S, которая может быть получена из такой диаграммы [13]. Определим S через набор переменных состояния, набор начальных условий и набор правил перехода, представляющих из себя пары «условие-действие», где условие – логическое выражение над переменными состояния, а действие – изменение значений переменных состояния. Переход из состояния s в состояние s` возможен тогда и только тогда, когда существует правило перехода, условие которого выполняется в s и действие которого переводит s в s`.

Набор переменных состояния S представляет собой набор переменных диаграммы вместе с:

- списком активных сообщений Q, с каждым элементом которого ассоциирован статус сообщения, принимающий одно из трёх значений: «in fabric» для сообщений, отправленных, но не дошедших до адресата, «at target» для полученных, но не обработанных сообщений, «enabled» для обработанных сообщений;

- подмножеством L всех рёбер типа «последовательность», отображающим множество всех активных на данный момент рёбер.

Начальное состояние системы характеризуется тем, что список Q и подмножество L пусты, а значения всех переменных диаграммы равны их начальным значениям.

Для каждого действия t определяем правило перехода τ системы S. Напомним, что действие представляет собой атомарную операцию вида «if Tsk_Condition then Tsk_Action», где Tsk_Condition – логическое выражение и Tsk_Action – набор операций присваивания. Условие для τ представляет собой конъюнкцию следующих логических выражений:

1. Tsk_Condition;

2. выражения, определяющего имеется ли у агента, выполняющего t, доступ ко всем переменным в Tsk_Action;

3. дизъюнкции (в случае обыкновенного действия; если же действие является блоком объединения, то вместо этого используется операция конъюнкции) условий запуска t:

3a. Sequence_Edge ϵ L для каждого из входящих в t рёбер Sequence_Edge типа «последовательность»;

3b. Q содержит сообщение m со статусом «enabled» для каждого из входящих в t рёбер типа «сообщение».

Действие для τ состоит из набора операций присваивания Tsk_Action вместе с необходимыми изменениями L и Q.

Система S имеет только один дополнительный тип правил перехода, представляющих собой изменение статуса одного из сообщений в Q с «in fabric» на «at target». Вид этих правил зависит от предполагаемого типа соединения между агентами.

V. Верификация диаграмм IFlow

После представления диаграммы в виде системы S открываются широкие возможности для дальнейшей машинной обработки, анализа и формальной верификации описанных с помощью предложенного метода протоколов.

Были разработаны два подхода к верификации диаграмм IFlow.

В первом из них [13] система состояний и переходов, описанная выше, транслировалась в формальные модели на языках Murφ и Promela

(Process Meta Language), после чего подавалась на вход системам верификации моделей Murφ и Spin (Simple Promela Interpreter) соответственно.

Покажем результат применения системы верификации моделей на примере сильно упрощённого протокола загрузки прошивки в изолированную память устройства (IM) из общей памяти (SM) (рис. 2). После перезагрузки (1) драйвер копирует прошивку в общую память (2), затем отправляет устройству сообщение с адресом прошивки, устройство копирует её в изолированную память (3), затем запрашивает у криптографического процессора аутентификацию загруженной прошивки. Криптопроцессор проверяет прошивку (4) и посылает результат проверки устройству. Если проверка пройдена (5), то устройство продолжает работу уже из изолированной памяти (6). После формальной верификации модели выяснилось, что при запуске двух экземпляров этого протокола одновременно драйвер может запросить аутентификацию одной прошивки и успеть загрузить другую во время проверки. Последовательность операций при этом выглядит так: 1, 2, 3, 4, 2`, 3`, 5, 7.

Рис. 2. Протокол загрузки прошивки

Формальная верификация настоящих системных протоколов проводилась с помощью системы Spin. Создание модели протокола могло занимать несколько дней, а работа Spin длилась по нескольку часов. Памяти системы не всегда хватало, чтобы верифицировать протоколы целиком, но авторами было получено множество контрпримеров, показывающих ошибки в протоколах.

Второй подход к верификации диаграмм IFlow [14] опирался на использование специально разработанного для верификации протоколов вида диаграмм последовательности операций под названием LSD (Live Sequence Diagrams). Этот формализм сочетает в себе плюсы сетей Петри и диаграмм последовательности сообщений. Диаграммы IFlow автоматически транслировались в диаграммы LSD, чтобы использовать последние для верификации протоколов и для проверки соответствия различных реализаций протоколов их спецификациям. Верификация с использованием этого подхода была применена к множеству системных протоколов, включая протоколы перезагрузки, алгоритмы управления питанием в системе, протоколы безопасности и так далее. Диаграммы IFlow для этих протоколов нередко насчитывали сотни вершин. Итогом этой работы стало нахождение множества ошибок в протоколах, в том числе таких, которые возникали только в случае одновременного выполнения нескольких протоколов. Такие ошибки почти невозможно поймать в процессе динамического тестирования без использования формальной верификации.

VI. Заключение

Существующий процесс создания и использования архитектурных спецификаций систем на кристалле имеет серьёзные недостатки. Метод создания спецификаций архитектурных протоколов системы на кристалле в виде диаграмм последовательности операций на основе синтаксиса нотаций BPMN, представленный в работе, является одним из возможных решений этой проблемы. Он позволяет улучшить качество спецификаций системных протоколов, избежать ошибок в их архитектуре, а также эффективно использовать эти спецификации в дальнейшем процессе проектирования системы. В работе перечислены возможности дальнейшего использования этих спецификаций: трансляция их в исполняемые модели, верификация протоколов, а также контроль соответствия реализаций системных протоколов этим спецификациям.

Авторы метода планируют продолжать работу по развитию описанной системы создания архитектурных спецификаций. Главными направлениями такой работы на сегодняшний момент являются согласование синтаксиса псевдокода, являющегося неотъемлемой частью диаграмм IFlow, определение способа задания выражений для проверки в процессе верификации протоколов, а также дальнейшее исследование возможностей проверки соответствия реализаций системных протоколов спецификациям в формате IFlow.

Литература

[1] Кравченко И. Системы на кристалле: общее представление и тенденции развития // Компоненты и технологии. 2001. № 6. С. 48-52.

[2] Riccobene E., Scandurra P., Rosti A., Bocchio S. A SoC design methodology involving a UML 2.0 profile for SystemC // Design, Automation and Test in Europe. 2005. P. 704-709.

[3] Lee Y.K., Kim N.H., Kim D., Lee D.H., In H.P. Customer Requirements Elicitation based on Social Network Service

// KSTT Transactions On Internet And Information Systems. Oct 2011. V. 5. No. 10. P. 1733-1750.
[4] Keller T. Contextual Requirements Elicitation // Seminar in Requirements Engineering. Spring 2011. Department of Informatics, University of Zurich.
[5] Dhungana D., Seyff N., Graf F. Research Preview: Supporting End-user Requirements Elicitation Using Product Line Variability Models // Requirements Engineering Foundation for Software Quality. Mar 2011. vol. 6606. P. 66-71. Springer Berlin Heidelberg.
[6] Neetu Kumari S., Pillai A.S. A survey on global requirements elicitation issues and proposed research framework // Software Engineering and Service Science (ICSESS). 2013. 4th IEEE International Conference on. IEEE, 2013. P. 554-557.
[7] Abarbanel Y., Singerman E. and Vardi M.Y. Validation of soc firmware-hardware flows: Challenges and solution directions // Design Automation Conference (DAC). 2014. 51st ACM/EDAC/IEEE. P. 1-4.
[8] Beers R. Pre-RTL formal verification: an intel experience // In Design Automation Conference. 2008. DAC 2008. 45th ACM/IEEE. P. 806-811.
[9] Azimi M., Chou C.T., Kumar A., Lee V.W., Mannava P.K., Park S. Experience with applying formal methods to protocol specification and system architecture // Formal Methods in System Design. 2003. Vol. 22(2), P.109-116.
[10] Zhu Q., Oishi R., Hasegawa T., Nakata T. System-on-chip validation using UML and CWL // Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis. 2004. P. 92-97.
[11] URL: http://www.bpmn.org/ (дата обращения: 01.01.2016)
[12] Печенко И.С., Венгер О.В. Способ автоматизированного создания диаграмм последовательности операций для сценариев поведения системы, описанных в виде текста // Информационные технологии. 2015. №1. Том 21. С. 50-56.
[13] Krstic S., Yang J., Palmer D.W., Osborne R.B., Talmor E., 2014, May. Security of SoC firmware load protocols. In Hardware-Oriented Security and Trust (HOST). 2014. IEEE International Symposium. P. 70-75
[14] Fraer R., Keren D., Khasidashvili Z., Novakovsky A., Puder A., Singerman E., Talmor E., Vardi M.Y., Yang J. From visual to logical formalisms for SoC validation // Formal Methods and Models for Codesign (MEMOCODE). 2014. Twelfth ACM/IEEE International Conference. P. 165-174.

SoC protocols specification and validation: problems and solutions

I. Pechenko, ivan.pechenko@intel.com

***Keywords* — system on chip, design process, architecture, system level language, protocol, verification, BPMN, diagram, transition system.**

I. INTRODUCTION

At first, let us define System on Chip. SoC is a complex integrated circuit that includes all the functional elements of a complete product on a single chip. The elements are usually CPU, memory, several accelerating function units and some interfaces to peripheral devices.

Today SoCs incorporate dozens of IP-blocks, complex reusable functional units.

SoC design process can be divided into five development stages: planning, architecture definition, microarchitecture design, RTL coding and convergence.

Nowadays most of system design validation activities take place only after RTL coding phase. One of the main reasons for it is the lack of complete, accurate and unambiguous architectural specifications. However, RTL is too low as an abstraction level to design and validate systems with millions of elements. In recent years, a big effort has been spent on moving from RTL level to system level in SoC design and verification. A new method is needed for formal system description and further validation on architecture or microarchitecture creation phases, both for hardware and software parts of the system. This new method will provide a possibility for system verification on the earliest stages of its design and will allow avoiding system specification errors, the most expensive errors for system designers.

II. DESIGN PROCESS CHALLENGES

At high level of abstraction System on Chip can be viewed as a network of intercommunicating hardware and software components (IP-blocks). SoC architectural specification can be divided into two parts: structural specification and behavioral specification. The first one is a map of system components and requirements specification for each of them. The second is the description of system behavior in different situations and protocols of IP-block intercommunication.

In this work, we will address the challenges in behavioral SoC architecture specifications. Most of key "infrastructure" SoC protocols include the interaction of many components, both hardware and software. Examples of these protocols are reset, power management, security and others. Previously used methods of specification and verification of such protocols are losing their effectiveness over the past years. There are several reasons for it: growing number of mistakes in SoC protocol specifications that implies the need for verification of such specifications, the key role of software in these protocols that implies the need for hardware and software co-verification etc.

III. EXISTING SOLUTIONS

Several solutions were suggested for the problems in SoC protocol specification. In [8] a process of creation and

verification of SoC architectural formal models was described. Authors used models written on several formal languages including TLA+, Murφ and HOL. They worked with architect teams and created the models in parallel with architecture creation.

In [9] authors describe their experience of applying formal methods for several complex system protocols including cache coherence protocol, sliding window protocol and other.

In [2, 10] a possibility for bringing together UML diagrams and languages like SystemC and C++ for system architecture modelling was described. Authors also consider hardware and software co-verification using their models.

IV. Proposed solution: IFlow diagrams

Here in this work a method for SoC protocols specification creation is described. Visual language called IFlow based on BPMN diagrams language is used for it. SoC protocol models described using IFlow should serve as a bridge between SoC architects, designers and validators.

Let's consider an IFlow diagram describing architecture and SoC protocol. It can be presented as an oriented graph. Its nodes are tasks performed by SoC components that will be called "agents". Edges of the graph can be of two types: control flows connecting sequential tasks performed by the same agent and messages connecting different agents.

Let's assume that text within a task is pseudocode that consists of a logical condition and a set of assignment operations on diagram variables and that text within a message or a control flow is a jump condition.

It is easy to define transition system S from this IFlow diagram. The definition of S consists of a set of state variables and a set of transition rules. The set of state variables is the set of diagram variables extended with the set of active messages with their status and the set of active control flows. The set of transition rules is inherited from the set of tasks.

V. IFlow diagrams verification

Translating the diagram into some formal model like transition system S enables wide capabilities for the protocol analysis and verification. Two teams created two different approaches for IFlow diagram verification.

One team [13] translated the transition system into Murφ and Promela models and used Murφ and Spin model checkers for their verification.

The second team [14] designed a formalism called LSD (Live Sequence Diagrams) for IFlows diagram verification. This formalism is inherited from Petri nets and message sequence charts and combines features of these models.

Both approaches bring good results: many mistakes in SoC protocol specifications were found that are very hard to catch "manually"; the quality of the architectural specifications grew dramatically.

VI. Conclusion

Existing process of SoC architectural specifications creation and usage has serious challenges. Here in this work we presented an approach for SoC architectural specifications creation using IFlow visual language derived from BPMN diagrams. Using IFlow diagrams allows creating accurate and unambiguous protocol specifications and further validating them by translation IFlow diagrams into formal models and using model checkers.

Authors plan to continue their work on the approach. Main tasks are agreement on pseudocode syntax in diagram shapes and on assertions syntax for formal verification.

References

[1] Kravchenko, "Systems on a chip: General representation and development trends," Kompon. Tekhnol. 2001. No. 6. P. 48-52.

[2] Riccobene E., Scandurra P., Rosti A., Bocchio S. A SoC design methodology involving a UML 2.0 profile for SystemC // Design, Automation and Test in Europe. 2005. P. 704-709.

[3] Lee Y.K., Kim N.H., Kim D., Lee D.H., In H.P. Customer Requirements Elicitation based on Social Network Service // KSTT Transactions On Internet And Information Systems. Oct 2011. V. 5. No. 10. P. 1733-1750.

[4] Keller T. Contextual Requirements Elicitation // Seminar in Requirements Engineering. Spring 2011. Department of Informatics, University of Zurich.

[5] Dhungana D., Seyff N., Graf F. Research Preview: Supporting End-user Requirements Elicitation Using Product Line Variability Models // Requirements Engineering Foundation for Software Quality. Mar 2011. V. 6606. P. 66-71. Springer Berlin Heidelberg.

[6] Neetu Kumari S., Pillai A.S. A survey on global requirements elicitation issues and proposed research framework // Software Engineering and Service Science (ICSESS), 2013. 4th IEEE International Conference on. IEEE, 2013. P. 554-557.

[7] Abarbanel Y., Singerman E. and Vardi M.Y. Validation of soc firmware-hardware flows: Challenges and solution directions // Design Automation Conference (DAC). 2014. 51st ACM/EDAC/IEEE. P. 1-4.

[8] Beers R. Pre-RTL formal verification: an intel experience // In Design Automation Conference. 2008. DAC 2008. 45th ACM/IEEE. P. 806-811.

[9] Azimi M., Chou C.T., Kumar A., Lee V.W., Mannava P.K., Park S. Experience with applying formal methods to protocol specification and system architecture // Formal Methods in System Design. 2003. Vol. 22(2), P.109-116.

[10] Zhu Q., Oishi R., Hasegawa T., Nakata T. System-on-chip validation using UML and CWL // Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis. 2004. P. 92-97.

[11] URL: http://www.bpmn.org/ (date of appeal: 01.01.2016)

[12] Pechenko I., Venger O. Generation of flow diagram from textual description of the flow // Information technologies. 2015. No. 1. V. 21. No. 1. P. 50-56.

[13] Krstic S., Yang J., Palmer D.W., Osborne R.B., Talmor E., 2014, May. Security of SoC firmware load protocols. In Hardware-Oriented Security and Trust (HOST). 2014. IEEE International Symposium. P. 70-75

[14] Fraer R., Keren D., Khasidashvili Z., Novakovsky A., Puder A., Singerman E., Talmor E., Vardi M.Y., Yang J. From visual to logical formalisms for SoC validation // Formal Methods and Models for Codesign (MEMOCODE). 2014. Twelfth ACM/IEEE International Conference. P. 165-174.