УДК 519.713

МУЛЬТИВЕРСНЫЙ ПАРАЛЛЕЛЬНЫЙ СИНТЕЗ ЦИФРОВЫХ СТРУКТУР НА ОСНОВЕ SYSTEMC СПЕЦИФИКАЦИИ

ОБРИЗАН В.И., СОКЛАКОВА Т.Г.

Обобщаются исследования, связанные с мультиверсным параллельным синтезом цифровых структур на основе SystemC спецификации, цель которого – существенное уменьшение времени проектирования вычислительных архитектур и повышение качества цифровых изделий.

Keywords – System-C, VHDL, Verilog, киберсистема, синтез.

1. Введение

Создание цифровых систем на кристаллах играет ключевую роль для развития компьютерных технологий, используемых в экономике, социологии, медицине, бизнесе, промышленности, образовании, которые формируют киберэкосистему планеты: распределенные центры больших данных, Internet of Things, Internet of Everything, cyber physical system, cloud-mobile computing, service computing.

Важно отметить экспоненциальный рост не только производительности компьютерных систем, но и продуктивности инженеров, благодаря созданию облачных сервисов проектирования аппаратных и программных приложений, интегрированных в понятие компьютинг (рис. 1).

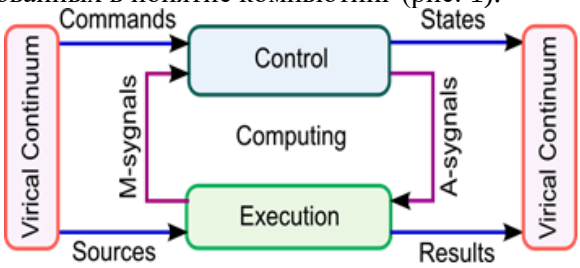

Рис. 1. Компьютинг для создания шифровых систем на кристалле

Компьютинг — процесс достижения поставленной цели путем использования ресурсов, механизмов управления и исполнения в циклически замкнутой системе с заданными отношениями и результирующей продукцией, сигналами мониторинга и актюации. Развитие облачных сервисов не имеет целью сократить рабочие места, а предоставить инженерам инструментарий для выполнения высокотехнологических работ в любой точке земного шара по созданию качественных программно-аппаратных приложений, минимизирующих время выхода годной продукции на рынок. Технологии облачных сервисов по разработке цифровых систем на кристаллах системного уровня позволяют создавать исполняемые спецификации — описание проекта на функциональном уровне, которое можно верифицировать в удаленном online-режиме. На функциональном уровне спецификация отражает вопросы «что делать?», а не «как делать», таким образом, она не содержит требований к структуре или архитектуре целевой платформы. При этом компонентами создания некоторой цифровой системы выступают уже сервисы, а не функциональные элементы. Такая замена дает существенный выигрыш во времени проектирования, которое определяется быстродействием процедур верификации и тестирования за счет варьирования параметрами системного, RT и вентильного уровней, добиваясь лучших характеристик реализации цифрового изделия на кристалле.

Цель – существенное уменьшение времени проектирования вычислительных архитектур и повышение качества цифровых изделий путем мультиверсного синтеза структуры цифрового изделия на основе заданной спецификации в среде SystemC (C++) и автоматического подбора функциональных компонентов за счет параллельного синтеза и верификации архитектурных решений системного уровня в соответствии с предложенной метрикой.

2. Методы системного и архитектурного синтеза интерфейсных структур

Модель системного уровня может не иметь однозначного интерфейса на логическом уровне. Разработанный метод определяет интерфейс модели, чтобы ее можно было однозначно представить на логическом уровне. Метод рассматривает прототипы C++-функций: количество аргументов, их типы и типы возвращаемых значений. В разделе представлено соответствие стандартных C++-типов типам языка VHDL. Следующий метод синтезирует протокол взаимодействия с модулем на логическом уровне. Для выбранного протокола взаимодействия строится граф-схема алгоритма, которая объединяется с граф-схемой проектируемого устройства. Таким образом, синтезируемое устройство сопряжено с другими устройствами, работающими на той же самой шине данных.

Обычно высокоуровневые модели не имеют понятия модельного времени: все вычисления происходят на одном условном такте работы. Модели же на логическом уровне работают по тактам. Таким образом, можно выделить две подзадачи: а) определение соответствия аргументов функции на системном уровне и сигналов на логическом уровне; б) определение протокола взаимодействия с синтезируемой функцией. Здесь протокол — это последовательность изменения информационных и управляющих сигналов схемы, которая приводит к выполнению синтезируемой функции.

Рассмотрим синтез интерфейсов для популярного стандарта шины системы на кристалле Wishbone.

Этот стандарт предполагает различные сопряжения модулей в системе: соединение «точка-точка»; конвейерное соединение (для обработки потока данных); соединение «многие ко многим» через общую шину; соединение «многие ко многим» через коммутатор; соединение в составе сети на кристалле.

Wishbone предоставляет шину данных, размер которой может быть от 8 до 64 бит, но всегда кратный одному байту. Таким образом, для передачи параметров используется последовательная загрузка значений во внутреннюю память ведомого модуля. Для этого используется специальная адресация регистров, которые расположены в интерфейсе ведомого модуля.

В том случае, если размер шины меньше 32 бит, требуется несколько циклов чтения результата из регистра R1. Можно рассчитать количество тактов, необходимых для передачи параметров. Пусть P_i – вектор параметров функции, i = 1,…,n, где n – количество параметров, передаваемых в функцию; $S(P_i)$ – размер параметра P_i в байтах; M – разрядность шины данных в байтах.

На рис. 2 показаны алгоритмы работы ведущего и ведомого интерфейсов в системе Wishbone. В состоянии a_3 показан вызов функции *f1* с параметром *par*. Согласно этой ГСА вычисление функции занимает ровно один такт работы автомата. Это может быть справедливо лишь для высокоуровневых моделей без отсутствия модельного времени. В системах на структурном уровне описания такого можно добиться только для комбинационных функций. Для автоматов, требующих большее количество состояний для завершения, необходимо изменить ГСА (а) следующим образом. На месте состояния a_3 необходимо подставить ГСА вызываемой функции *f1*.

3. Модели описания логических блоков системного уровня

На первом этапе разработки модели контейнера «вектор» системного уровня был формализован интерфейс SystemC-модуля для контейнера «вектор». На рис. 3 показана система, включающая все необходимые компоненты системного уровня. Clock — это генератор тактовых импульсов. Контейнер «вектор» — синхронное устройство, у которого все операции осуществляются по переднему фронту сигнала синхронизации.

Генератор тестов — это высокоуровневый модуль, который содержит генератор тестов для контейнера и анализатор ответных реакций. Этот генератор будет повторно использован для верификации модели на логическом уровне: H2L (high level to low level) – транзактор, который преобразует высокоуровневые вызовы функций контейнера к протоколу на уровне переключения логических сигналов; L2H (low level to high level) – транзактор, который осуществляет преобразование, обратное H2L – протокол уровня логических сигналов к высокоуровневым вызовам функций контейнера «вектор». На рис. 4 показан подробный интерфейс транзактора L2H. Этот же интерфейс будет иметь будущая аппаратная реализация.

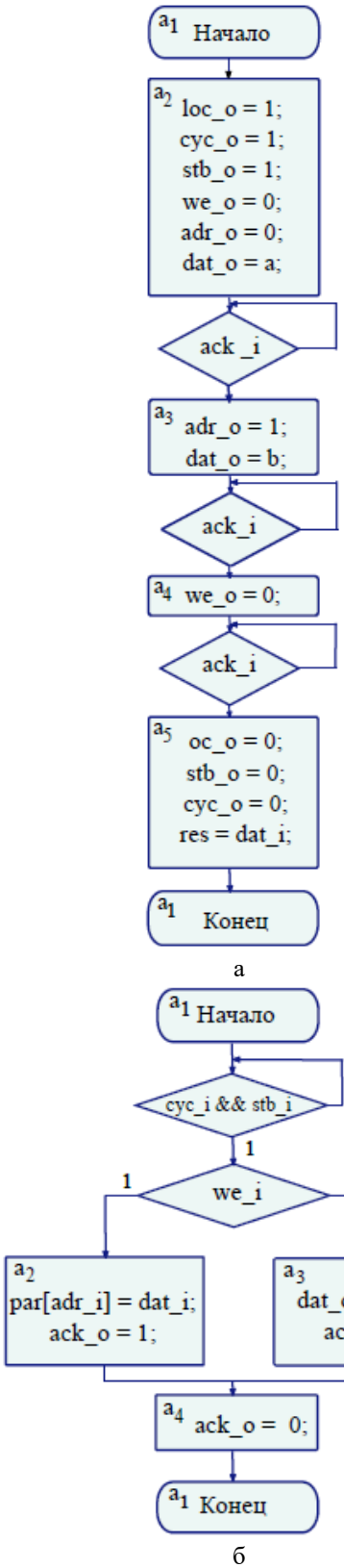

Рис. 2. Алгоритмы ведущего и ведомого интерфейсов в системе Wishbone

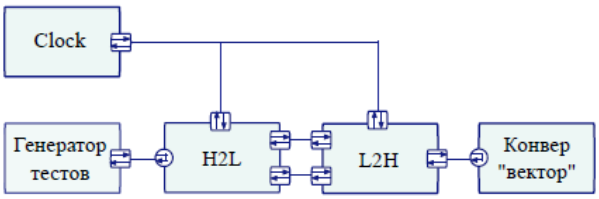

Рис. 3. Полный комплект модулей и транзакторов для разработки системной модели конвейера “вектор”

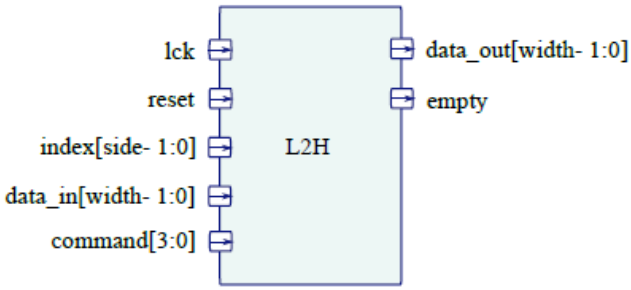

Рис. 4. Интерфейс логического уровня модели контейнера “вектор”

Логическая модель контейнера «вектор» реализована с помощью примитивов блочной памяти кристалла Xilinx Virtex-5. Один блок памяти кристалла рассчитан на 36 кбит данных и может быть сконфигурирован или как два независимых модуля памяти по 18 кбит, или как один модуль емкостью 36 кбит. Можно пользоваться следующими конфигурациями памяти: 32К по одному биту, 16К по 2, 8К по 4, 4К по 9, 2К по 18 или 1К по 36 бит.

Выбор кристаллов Xilinx Virtex определяется следующими факторами. Эти кристаллы доступны на рынке в низких партиях за приемлемые цены. Здесь они существенно выигрывают перед кристаллами жесткой логики, для которых только подготовительные затраты на изготовление маски и пробную серию могут достигать миллионов долларов. ПЛИС Virtex также имеет возможность многократного перепрограммирования, что позволяет испытывать различные конфигурации проектируемого устройства, а также исправлять логические дефекты в проекте. Кристаллы жесткой логики такой возможности не имеют.

Кристаллы Xilinx Virtex также имеют необходимые ресурсы для реализации контейнера «вектор»: элементы блочной памяти. Блочная память поддерживает синхронные операции записи/чтения, два порта доступа полностью симметричны и работают независимо друг от друга, разделяя данные в памяти.

Ограничения логической модели: контейнер занимает всю память, которую предоставляет базовый элемент блочной памяти, использованный в реализации; максимальное количество элементов вектора определяется размером одного элемента и объемом элементов памяти, использованным в реализации. Может быть вычислено по формуле $V = w \times k$, где w — размер элемента в битах, k – количество элементов, V — объем занимаемой памяти.

Для успешной реализации модели контейнера «вектор» была разработана дополнительная логика, реализующая требуемые операции для элементов памяти.

Контейнер «вектор» предоставляет полный набор конструкторов, деструкторов и операций копирования. Для реализации множества операций используется дешифратор команд, который переводит из двоичного кода инструкции в позиционный код, которые активирует управляющие сигналы контейнера. Логическая модель имеет следующие параметры: DATA_WIDTH определяет ширину шины данных. Этот параметр зависит от того, для какого типа данных создан контейнер «вектор». Параметр выбирается в зависимости от максимального количества доступных команд, число которых равно $2^{\text{CMD\_WIDHT}}$; SIZE определяет ширину шины адреса. Этот параметр влияет на максимальное количество адресуемых ячеек контейнера и равно 2^{SIZE}.

Логическая модель контейнера «вектор» имеет интерфейсные сигналы: глобальный синхроимпульс *clk*; сброс *reset*, при высоком уровне которого происходит сброс элементов памяти в нули, что соответствует очистке контейнера; шина адреса *index* размерностью [SIZE-1:0], которая используется для указания порядкового номера элемента в векторе при записи или чтении; шина данных *data_in* размерностью [DATA_WIDTH-1:0], которая используется для записи данных в вектор; шина команд *command* размерностью [CMD_WIDTH-1:0], которая используется для управления контейнером; шина данных *data_out* размерностью [DATA_WIDTH-1:0], которая используется для чтения данных из вектора; сигнал *empty*, высокий уровень которого сигнализирует о том, что вектор пуст.

Структурная модель контейнера «вектор» содержит дешифратор команд *dc*, который преобразует двоичный код в управляющие сигналы.

Блок *pointer* содержит два регистра: адрес последнего элемента (для пустого вектора он равен – 1) и адрес первого свободного элемента (для пустого вектора равен нулю). На входе блока есть три управляющих сигнала: *reset, inc, dec*. При высоком уровне сигнала inc и переднем фронте синхроимпульса внутренние регистры увеличиваются на единицу. При высоком уровне сигнала dec и переднем фронте синхроимпульса внутренние регистры уменьшаются на единицу.

Мультиплексор индекса управляется шиной из двух сигналов. В режиме чтения/записи ячейки по индексу сигналы должны быть установлены в 00, в режиме *push_back()* сигналы должны быть установлены в 01, в режиме *back()* сигналы должны быть установлены в 10, в режиме *front()* сигналы должны быть установлены в 11.

Для анализа и проектирования контейнера «список» рассмотрим типичные операции, выполняемые контейнером «вектор». Здесь X – это условие, возбуждающее ту или иную операцию, команды и описание команд: x_1_begin() – ссылка на первый элемент списка; x_2_end() – ссылка на последний элемент списка; x_bool empty() – возвращает true, если список пуст; x_3 - push_front() - добавляет новый элемент в начало; x_4_push_back() – добавляет новый элемент в конец; x_5_pop_front() – удаляет элемент из начала списка; x_6_pop_back() – удаляет элемент из конца списка; x_7_insert() – добавляет элемент перед текущим; x_8_erase() – удаляет текущий элемент; x_9_next() – ссылка на следующий элемент; x_{10}_prev() – ссылка на предыдущий элемент.

Для успешной реализации модели контейнера «список» была разработана дополнительная логика, реализующая требуемые операции.

Для верификации полученных моделей были использованы тесты, проверяющие 100% режимов работы устройства. Исходные эталоны были получены из программы на C++.

Здесь рассматриваются только функциональные тесты – проверка правильности функционирования модели, и нагрузочные тесты – замеры производительности и временных характеристик модели. На рис. 5 показана схема процесса верификации логической модели.

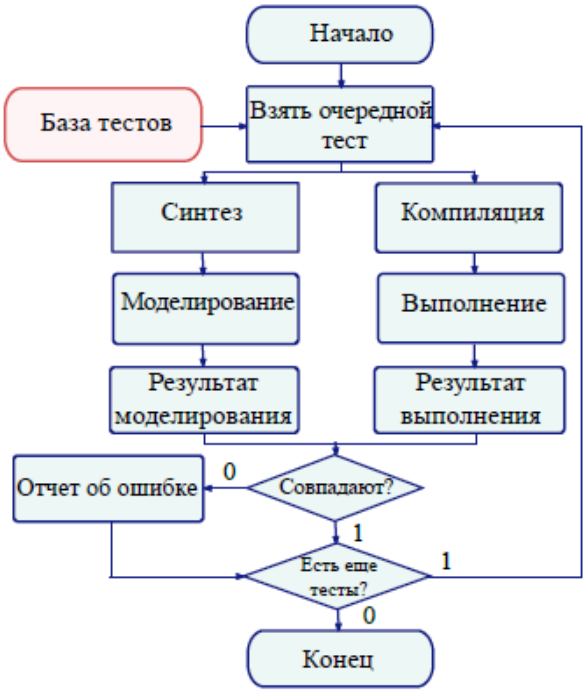

Рис. 5. Схема верификации логической модели

Создается набор тестов на C++, в которых используются контейнер «вектор». В этих тестах присутствуют вызовы всех методов, которые поддерживает контейнер. На следующем этапе тесты компилируются обычным C++ компилятором для получения исполнимого файла. При исполнении этой программы в файле сохраняются результаты работы контейнера.

С другой стороны, происходит синтез исходных текстов. В результате синтеза получается модель системного уровня, содержащая контейнер «вектор», описанный на ЯОА VHDL. Далее осуществляется моделирование с помощью программы Aldec Active-HDL для получения результатов. Эти результаты сравниваются с результатами, полученными при работе обычной программы. Логическая модель составлена верно, если результаты моделирования и выполнения совпадают.

Следующим этапом верификации является проверка соответствия функциональной модели и модели с временными задержками. В функциональной модели отсутствует информация о времени распространения сигнала от входов элементов к выходам. Подразумевается, что сигнал распространяется мгновенно или за один дельта-цикл моделирования. Программа логического синтеза, трассировки и размещения генерирует специальный файл задержек распространения сигнала для выбранной архитектуры. Этот файл может быть загружен в САПР Aldec Active-HDL для построения временной модели.

Последним этапом тестирования является проверка модели, реализованной в том или ином кристалле. В работе выбрана архитектура ПЛИС Xlinix – дешевая и высокопроизводительная микросхема с возможностью многократного перепрограммирования.

4. Программно-аппаратная реализация моделей

Представлен метод мультиверсного синтеза управляющих и операционных автоматов в заданной инфраструктуре проектирования, ориентированных на архитектурные решения в метрике, минимизирующий время выполнения функциональности за счет распараллеливания операций при ограничении на аппаратные затраты.

По своей сути программа синтеза C++ в VHDL является машиной по трансформации исходного описания в результирующее. Таких трансформаций происходит несколько. Входная модель представлена на языке C++. Это алгоритмическое описание, которое решает поставленную перед инженером задачу.

Выполнено тестирование и верификация программных модулей инфраструктуры проектирования цифровых систем на кристаллах, а также определение эффективности предложенных моделей, методов и структур данных при создании реальных компонентов цифровых изделий.

Тестирование системы состоит из нескольких этапов: разработка схемы тестирования компилятора; подготовка тестов; подготовка эталонов; исполнение тестов и анализ результатов. В работе [11] описаны этапы преобразования.

Проведено сравнение производительности разработанной аппаратной модели и программной реализации на микропроцессоре экспериментальным путем. Данные получены вследствие работы 10 инженеров над одним проектом. Оценивались следующие характеристики: временные затраты на проектирование, быстродействие, энергопотребление, площадь на кристалле. В результате исследования получили при аддитивной оценке автоматический метод лучше в 1,301292012 раз (рис. 6).

Рис. 6. Аддитивная оценка эффективности

5. Заключение

Сущность рыночно-ориентированного научно-технического исследования определена как мультиверсное проектирование архитектуры цифрового изделия на основе заданной спецификации в среде SystemC (Си++) и автоматического выбора синтезируемых функциональных структур в целях существенного уменьшения времени создания проекта и повышения выхода годной продукции за счет параллельного синтеза и верификации архитектурных решений системного уровня в соответствии с предложенной метрикой. Основная инновационная идея заключается в параллельном автоматическом синтезе квазиоптимальной архитектуры в соответствии с предложенной спецификацией и метрикой на основе подбора синтезируемых функциональных структур. Достигнута цель исследования как существенное уменьшение времени проектирования вычислительных архитектур и повышение качества цифровых изделий.

Литература:

1. *Hahanov V., Yemelyanov I., Obrizan V., Hahanov V.* "Quantum" Diagnosis and Simulation of SoC // Proceedings of XIth Intern. Conf. MEMSTECH. 2015. P. 58-60.

2. *Vladimir Obrizan, Igor Yemelyanov, Vladimir Hahanov, Eugeniya Litvinova*. Matrix-Model for Diagnosing SoC HDL-Code // Radioelektroniks and informatics. 2013. №1. P.12-19.

3. *Hahanova, I.V.; Obrizan, V.; Adamov, A.; Shcherbin, D.* Transaction level model of embedded processor for vector-logical analysis //, 2013 East-West Design & Test Symposium, pp.1-4, 27-30 Sept. 2013.

4. *Obrizan, V.* A method for automatic generation of an RTL-interface from a C++ description // East-West Design & Test Symposium (EWDTS). 17-20 Sept. 2010. pp. 186-189.

5. *Volodymyr Obrizan*. A Method of High-Level Synthesis and Verification with SystemC Language // Радиоэлектроника и информатика. 2010. №4.

6. *Technologies* for hardware simulation and verification / V. Hahanov, A. Hahanova, V. Obrizan, K. Zaharov // Proc. of the International Conference Modern Problems of Radio Engineering, Telecommunications and Computer Science – TCSET'2008, February 19–23. Slavske, Lviv, Ukraine. 2008. P. 560–564.

7. *Testing* challenges of SoC hardware-software components / V. Hahanov, V. Obrizan, S. Miroshnichenko, A. Gorobets // Proceedings of IEEE East-West Design and Test International Symposium, October 9–12, 2008. Lviv, Ukraine. P. 149–154

8. *Hardware* Simulation and Verification Technologies / V. Hahanov, A. Hahanova, V. Obrizan, W. Ghribi // Proceedings of IEEE East-West Design and Test Symposium, September 7–10, 2007, Yerevan, Armenia. P. 739–744.

9. *Parallel* Logic Simulation Using Multi-Core Workstations / V. Hahanov, V. Obrizan, A. Gavrushenko, S. Mikhtonyuk // The Experience of Designing and Application of CAD Systems in Microelectronics: proceedings of the 9th International Conference, February 19–24, 2007, Polyana, Ukraine. Lviv, 2007. P. 256–257.

10. Hierarchical Testing of Complex Digital Systems / V. Hahanov, V. Obrizan, V. Yeliseev, W. Ghribi // Proc. Of the Modern Problems of Radio Engineering, Telecommunications and Computer Science – TCSET'2006. February 28 – March 4, 2006, Slavske, Lviv, Ukraine. Lviv, 2006. P. 426–429.

11. *Обризан В.И.* Инфраструктура проектирования SOC для метода мультиверсного синтеза / В.И. Обризан // Радиоэлектроника и информатика. 2016. №2. С. 48-60.

Поступила в редколлегию 17.12.2017

Рецензент: д-р техн. наук, проф. Кривуля Г.Ф.

Обризан Владимир Игоревич, канд. техн. наук, директор Design & Test Lab. Научные интересы: компьютерная инженерия. Адрес: Украина, 61166, Харьков, ул. Бакулина, 14.

Соклакова Татьяна Игоревна, инженер кафедры АПВТ ХНУРЭ. Научные интересы: компьютерная инженерия. Адрес: Украина, 61166, Харьков, пр. Науки, 14.

Obryzan Vladimir Igorevich, PhD, director of Design & Test Lab. Scientific interests: computer engineering. Address: Ukraine, 61166, Kharkov, ul. Bakulina, 14.

Soklakova Tetiana Igorevna, Engineer, Design Automation Department. Scientific interests: computer engineering. Address: Ukraine, 61166, Kharkov, Nauki Ave, 14.