Санкт-Петербургский государственный университет

Математико-механический факультет

Кафедра системного программирования

Практическая оценка качества различных средств HLS при синтезе из SystemC

Курсовая работа студента 445 группы

Шеина Романа Евгеньевича

Научный руководитель: Салищев С. И.

Санкт-Петербург

2012

Содержание

Введение

High Level Synthesis (далее HLS) – активно развивающийся подход разработки интегральных схем[3]. Входом инструментов HLS является алгоритмический код на языке высокого уровня, обычно C/C++/SystemC, выходом является register transfer level (далее RTL) код, обычно VHDL или Verilog. Затем из RTL кода логическим синтезом получают представление схемы на уровне вентилей, что может считаться конечным продуктом разработки с использованием HLS.

В современном мире цифровой обработки сигналов очень быстро разрабатываются новые алгоритмы, и одним из ключевых факторов успеха интегральной схемы, решающей аппаратно какую-либо задачу, является минимизация задержки выхода схемы на рынок. HLS позволяет значительно сократить цикл разработки за счёт автоматизации верификации и общего ускорения разработки (написание модели на языке высокого уровня на порядок быстрее, чем написание RTL кода).

Постановка задачи

Задача HLS – получение RTL кода из кода на языке высокого уровня, инструмент HLS не обладает никакой информацией о стадии логического синтеза, такой как топография чипа, физические характеристики соединений и прочие[2]. Кроме того, логический синтез производится другим инструментом, не связанным с инструментом HLS. Таким образом, стадии получения RTL кода и представления на уровне вентилей оказываются плохо согласованы, что приводит к трудностям и ошибкам.

Инструменты HLS решают множество вычислительно сложных задач, таких как составление графика для операций, привязку операций к компонентам с разделением ресурсов и прочие. Различные инструменты применяют различные эвристические подходы для решения этих задач, что также приводит к ряду проблем при разработке.

Один из популярных подходов к разработке систем на чипе в данный момент — применение основанных на systemC виртуальных платформ[1]. Применение этого подхода с HLS для генерации аппаратной части считается одним из наиболее эффективных способов кодизайна с точки зрения скорости получения результата и простоты верификации. В данной работе рассматриваются только инструменты HLS, поддерживающие SystemC и, таким образом, пригодные для использования в разработке на базе виртуальной платформы.

Основной задачей курсовой ставится исследование проблем разработки с применением различных работающих с C/C++/SystemC коммерческих инструментов HLS на «живых» примерах при использовании техпроцесса менее 45nm и предложение практических методов их решения в рамках исследуемых инструментов.

Рассматривались следующие коммерческие средства HLS:

- Mentor Graphics Catapult [4]
- Forte Cynthesizer [5]
- Cadence C-to-Silicon [6]

Инструмент логического синтеза:

- Design Compiler Ultra [7]

Рассматривались следующие примеры:

- Быстрое преобразование Фурье
- Матричное умножение
- Кусочно-полиномиальная аппроксимация элементарных функций
- Фильтрация (FIR-filter)

1. Синтезируемое подмножество C/C++/SystemC

В современных системах на чипе, особенно для «тонких» техпроцессов, стоимость соединений и интерфейсов между модулями с точки зрения площади и мощности оказывается значительной частью общей стоимости чипа. В связи с этим хранение IP-блоков в виде неизменных «чёрных ящиков» неэффективно. Почти для каждого конкретного дизайна блоки модифицируются. Например, блок быстрого преобразования Фурье может применяться в различных протоколах связи вместе с различными блоками векторных операций. Для эффективной реализации таких протоколов необходимо разделение ресурсов между блоком Фурье и векторными операциями, в том числе разделение памяти, для чего может потребоваться модификация адресной арифметики блока Фурье.

В связи с этим эффективно хранить IP-блоки не в виде RTL кода, а в виде готового к обработке HLS исходного кода на C/C++/SystemC — внесение изменений в код на языке высокого уровня проще и быстрее, чем редактирование RTL кода, особенно если RTL сгенерирован инструментом.

HLS – активно развивающийся подход с очень динамическим и довольно непредсказуемым рынком. В связи с этим привязка к определённому производителю средств при создании библиотеки блоков на языке высокого уровня, готовых к HLS, нежелательна. В идеальном случае хочется для каждого блока иметь код, готовый к обработке любым из коммерческих инструментов.

К сожалению, на практике это оказывается весьма проблематично в связи со следующей проблемой: несмотря на то, что все рассматриваемые в данной работе инструменты работают с SystemC, синтезируемые подмножества всех трёх инструментов различны.

- Catapult поддерживает синтез из функций, C-to-Silicon и Cynthesizer работают только с модулями SystemC.
- Forte использует специальные директивы, вставляемые непосредственно в исходный код, для контроля действий инструмента на стадиях построения графика и определения ресурсов.
- Catapult не поддерживает обыкновенные каналы FIFO библиотеки systemC, для корректного соединения требуется применение специальных каналов modularIO.
- Каждый из инструментов имеет набор своих собственных типов с фиксированной точкой, хотя все три поддерживают стандартные типы sc_int, sc_uint.

В целом: арифметическая часть примерно совпадает (или приводится к общему виду с помощью препроцессора), взаимодействия между модулями поддерживаются со значительными различиями.

Попытка создать единый исходный код для блока быстрого преобразования Фурье на практике окончилась получением кода, который проходил через все 3 инструмента, однако был совершенно нечитабелен.

В качестве решения этой проблемы рекомендуется выделить общую вычислительную часть и переписывать взаимодействие между модулями под каждый инструмент отдельно.

2. Анализ потока данных

Инструменты HLS предоставляют возможность вставлять в модули память в виде массивов (при обработке кода высокого уровня инструмент запрашивает, должен ли массив быть представлен как память или распределён по регистрам). Мультибанковая память представляется двумерным массивом.

При написании in-place блока Фурье с нетривиальной адресацией на мультибанковой памяти в C-to-Silicon появлялись множественные фиктивные зависимости по памяти, связывающие банки. Вероятнее всего, это ошибка анализа потока данных.

Примерно в середины работы над курсовой Cadence была выпущена новая версия C-to-Silicon, позволяющая явно разделять на банки двумерный массив, представляющий память. На самом деле это лишь частично решает проблему, поскольку, является не до конца автоматизированным решением и не является решением вовсе, когда память делится на банке в цифре размера, отличного от степени двойки.

При работе с Catapult также появлялись фиктивные зависимости, однако из количество было мало, и с ними удалось легко справиться средствами инструмента.

В качестве решения этой проблемы рекомендуется выделять память в отдельный модуль с явной реализацией соединения с модулем, включающим память.

3. Оптимизация арифметики

При получении блока Фурье из Forte на уровне RTL в бабочке был получены умножители ширины, значительно большей, чем необходимо. Это, скорее всего, ошибка инструмента, которая будет исправлена в последующих версиях.

При полиномиальной аппроксимации глубина и ширина дерева сумматоров зависят от целевой частоты: чем выше частота, тем больше ширина и тем меньше глубина дерева. Рассматриваемые HLS инструменты либо плохо поддерживают автоматическую балансировку деревьев, либо не поддерживают вовсе.

Оценки инструментов HLS часто оказываются некорректными, поскольку они не учитывают оптимизации арифметики, проводимые инструментом логического синтеза. Например, Design Compiler Ultra балансирует деревья сумматоров и вводит carry-safe арифметику. Это приводит к различным ошибкам в оценках площади/мощности, в частности иногда оцененные лучше в HLS варианты оказываются хуже других после логического синтеза.

В качестве решения проблемы балансировки деревьев предлагается балансировка дерева вычислений «руками». Это довольно плохое решение, поскольку оно противоречит идее автоматизации, стоящей за HLS, и хорошее решение не может быть найдено без интеграции HLS с инструментами логического синтеза.

4. Перераспределение ресурсов

Инструменты HLS оптимизируют результат, проводя перераспределение и разделение ресурсов между операциями. Эта задача очень трудоёмка и потому решается эвристически, различно в каждом продукте, что приводит к тому, что результат непредсказуем и различен в зависимости от того, какой именно инструмент используется.

Чтобы получить предсказуемое разделение ресурсов, рекомендуется переписать код с явным переиспользованием ресурсов в языке высокого уровня.

Инструменты логического синтеза также могут перераспределять ресурсы, нарушая построенную HLS структуру. Это приводит к различиям между ожидаемой HLS и фактической структурой, то есть к ошибкам оценок HLS.

5. Ошибки характеризации

Обычно инструменты HLS применяются для техпроцессов в 45-90nm, однако на данный момент активно внедряются техпроцессы менее 45nm. При переходе на более «тонкие» техпроцессы штрафы, накладываемые физическими особенностями и игнорируемые средствами HLS, становятся более значимыми, и, соответственно приводят к большим ошибкам в оценках площади/мощности.

Также стоит учесть, что, поскольку HLS «не знает» ничего о логическом синтезе,

часто метод декомпозиции модуля на блоки меньшего размера, применяемый HLS, плохо соответствует результатам логического синтеза, из-за чего увеличивается площадь/мощность чипа, или же падает пиковая частота.

В качестве решения этой проблемы предлагается сперва проведение исследования задачи пропусканием модуля через HLS целиком, а затем, в случае неудовлетворительного результата, разбиение модуля на блоки меньшего размера на уровне C++ и синтеза каждого блока средствами HLS отдельно с последующим соединением результатов. Вновь, «правильным», более фундаментальным решением этой проблемы могла бы стать более плотная интеграция HLS и логического синтеза.

6. Общие рекомендации

Стандартным решением этой проблем характеризации является повышение целевой частоты в HLS без повышения целевой частоты при логическом синтезе. Надо заметить, что это решение работает далеко ни всегда и ни в коем смысле не оптимально. Поскольку внутреннее устройство средств HLS неизвестно, повышение частоты даже не всегда приводит к уменьшению критического пути на уровне вентилей. Например, при работе с блоком Фурье с Cynthesizer повышение частоты в 2 раза помогло решить проблему, однако повышение частоты ещё на 100Mhz значительно увеличило и критический путь, и площадь, приведя к результату, сравнимому с вариантом без оверклокинга по превышению критическим путём допустимого времени, и намного более худшему по площади и мощности. Для получения желаемого результат с блоком Фурье пришлось повышать частоту хотя бы в 2 раза для всех инструментов.

Часто проблема может быть решена внимательным изучением возможностей продукта - как правило, имеются опции, специально модифицирующие оценки HLS с учётом некоторых известных ошибок характеризации (например, резервирование в графике свободного места в каждом такте, обеспечивающего на логическом синтезе возможности внесения изменений без серьёзных нарушений структуры).

Инструменты находятся в разработке, есть немалые шансы получить проблемы, вызываемые ошибками в самих инструментах. Рекомендуется писать производителю при подозрениях на ошибку, поскольку обратная связь с пользователем критически важна для развития, особенно в такой специфичной области как HLS: как правило, какая-либо функциональность реализуется только по требованию пользователя.

Заключение

Легко заметить, что большинство описанных в данной работе проблем произрастают из плохой согласованности HLS с логическим синтезом. Радикальным решением является интеграция логического синтеза в цепь HLS с созданием непрерывной цепи инструментов с автоматизированной верификацией (на данный момент инструменты HLS предоставляют возможность верификации уровня вентилей, но непосредственно логический синтез осуществляется сторонней программой).

Для уменьшения ошибок характеризации требуется поддержка «тонких» библиотек и высоких частот, чего стоит ожидать по мере распространения техпроцессов менее 45nm среди пользователей HLS.

Для улучшения применимости метода желателен стандарт, определяющий унифицированное синтезируемое подмножество C/C++/systemC, однако появление такого стандарта в ближайшем будущем маловероятно в связи с большим количеством различий между синтезируемыми подмножествами различных инструментов.

В целом, если опыта разработчика хватает для преодоления проблем HLS, этот метод позволяет значительно ускорить разработку и получить результаты сравнимые с программированием в RTL по площади и мощности (а иногда и превосходящие их за счёт алгоритмических оптимизаций). С развитием инструментов ожидается распространение HLS, улучшение результатов и дальнейшее уменьшение времени разработки.

Литература

1. http://webadmin.dac.com/knowledgecenter/2010/documents/POPOVICI-VP-ABK-FINAL.pdf
2. Ferraresi, M.; Gobbo, G.; Ludovici, D.; Bertozzi, D.; , "Bringing Network-on-Chip links to 45nm," *System on Chip (SoC), 2011 International Symposium on* , vol., no., pp.122-127, Oct. 31 2011-Nov. 2 2011
3. Fingeroff, Michael. High-level synthesis : blue book. United States: Xlibris Corporation Mentor Graphics Corporation, 2010.
4. http://www.mentor.com/esl/catapult/overview
5. http://www.forteds.com/products/cynthesizer.asp
6. http://www.cadence.com/products/sd/silicon_compiler/pages/default.aspx
7. http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DCUltra/Pages/default.aspx