УДК 004.414.2

Технология построения синхронных эластичных схем и ее применение к оптимизации производительности аппаратного декодера H.264 CABAC

А. Готманов[1], М. Кишиневский[2], М. Галсеран-Омс[3]

[1]Strategic CAD Labs, Intel Corporation, Москва, alexander.gotmanov@intel.com

[2]Strategic CAD Labs, Intel Corporation, Орегон, Хилсборо (США)

[3]Политехнический университет Каталонии, Барселона (Испания)

***Аннотация* — При описании синхронных блоков обычно полагаются известными задержки связанные с обработкой и передачей данных. Это приводит к функциональной негибкости системы и к потерям быстродействия, которое определяется наихудшим из возможных поведений. Мы провели эксперимент по применению предложенной в [2] технологии проектирования синхронных схем, адаптирующихся к изменениям задержек, к оптимизации критического блока промышленного видеодекодера (H.264), повысив его производительность в 1,5 раза при неизменной площади и энергопотреблении. Для решения поставленной задачи был разработан новый метод построения блоков с переменной задержкой и прототипы средств САПР для трансляции исходного описания системы на языке Verilog в эластичную форму и его последующей оптимизации путем эквивалентных микроархитектурных преобразований.**

***Ключевые слова* — оптимизация производительности, эластичность, сжатие видео, H.264, CABAC.**

I. Введение

Корректность синхронной системы, как правило, зависит от известных значений задержек обработки и передачи данных, измеряемых в числе тактов работы. Тактовые задержки основных подблоков задаются на ранних этапах проектирования и становятся частью функционального описания системы. В результате, незапланированное изменение задержки отдельного вычисления в готовой системе часто приводит к проектированию значительной ее части с нуля.

Нечувствительные к тактовым задержкам (latency tolerant; иначе – синхронные эластичные) системы [1], [3] адаптируются к изменениям в задержках обработки и передачи данных. Функциональная правильность таких систем более устойчива к изменениям временных характеристик подблоков и окружения, а реализация может быть оптимизирована для *типичных* поведений (вместо наихудших), что открывает новые проектные решения и позволяет улучшить электронную схему по площади и производительности.

Хотя потенциальные преимущества нечувствительных к задержкам схем хорошо изучены в теории, нет убедительных примеров их использования для решения практических задач проектирования ИС. Мы провели экспериментальное исследование применимости существующих методов построения синхронных эластичных схем [2] к оптимизации существующего критического по производительности блока (декодера H.264 CABAC) разрабатываемой в Intel системы на кристалле.

Для решения поставленной задачи мы разработали новый метод оптимизации синхронных эластичных блоков и создали прототипы средств САПР для трансляции исходного описания системы на языке Verilog в синхронную эластичную форму и его последующей оптимизации путем эквивалентных микроархитектурных преобразований. Эксперимент проводился с использованием промышленных средств САПР синтеза ИС на основе 32-нм технологического процесса. В результате удалось повысить производительность блока в 1.5 раз при неизменной площади и энергопотреблении.

Далее в разделах II и III мы кратко опишем метод проектирования нечувствительных к задержкам схем с применением технологии SELF (Synchronous Elastic Flow). Раздел IV посвящен методу оптимизации синхронных эластичных схем, основанному на принципе “разделяй и властвуй”. Микроархитектура декодера H.264 CABAC представлена в разделе V. Раздел VI описывает оптимизацию блока декодера с помощью микроархитектурных преобразований. Результаты эксперимента представлены в разделе VII. В разделе VIII подведены итоги исследования.

II. Синхронные эластичные системы

Эластичность – это способность системы адаптироваться к изменениям задержек. Рис. 1 иллюстрирует идею эластичности на примере схемы сумматора. Простейший синхронный сумматор показан на рис. 1а. На каждом цикле схема интерпретирует значения на входах, как набор операндов, и вычисляет результат на выходе. Неэластичный сумматор будет производить корректную выходную последовательность, только если входные последовательности синхронизированы так, что все слагаемые появляются на входах схемы на одном и том же такте. Задача обеспечения синхронизации возлагается на внешнюю среду.

На рис. 1б показано поведение асинхронного сумматора. Входы и выходы схемы представляют собой *эластичные каналы.* Данные из входных каналов прибывают независимо в произвольные моменты времени. После операции суммирования, выполняемой с переменной задержкой, результат становится доступен в выходном канале. В силу отсутствия глобальной синхронизации, корректность передачи данных обеспечивается с помощью протокола “запрос-ответ” (handshake). Функциональная правильность асинхронного сумматора не зависит от задержек в работе внешней среды. Однако применимость асинхронных схем на практике ограничена значительными накладными расходами на предотвращение логических состязаний, которые приводят к паразитным импульсам (glitches) и некорректному поведению системы.

На рис. 1в представлен синхронный эластичный сумматор. Все передачи данных выровнены по фронту тактового сигнала, однако входные каналы допускают произвольные задержки за счет добавления *нерабочих* (пустых) циклов. Эластичная система – это особый вид синхронной системы, где данные интерпретируется как *рабочие* только при наличии единичного значения управляющего *valid*-бита, сопровождающего данные. Кроме того, каждый функциональный блок может остановить передачу данных на текущем цикле, установив управляющий *stop*-бит в единицу. Вместе биты *valid* и *stop* реализуют простой синхронный вариант протокола “запрос-ответ”, свободный от недостатков присущих асинхронным системам. Правильность работы синхронного эластичного сумматора сохраняется вне зависимости от числа тактов задержки в появлении входных данных. В дальнейшем мы рассматриваем только синхронные эластичные системы, называемые для краткости просто *эластичными*.

Работы [2]-[3] развивают технологию SELF (Synchronous Elastic Flow) систематического проектирования ИС устойчивых к изменениям тактовых задержек. Спроектированная по технологии SELF эластичная система состоит из вычислительных блоков, регистровых файлов, блоков памяти и эластичных буферов (FIFO-очередей), соединенных каналами. Канал – это набор проводов, кодирующих значение данных, вместе с управляющими проводами *valid* и *stop*. На рис. 2а показана обычная эластичная система с двумя эластичными буферами и комбинационным вычислительным блоком между ними. Эластичный буфер (ЭБ) в базовой реализации обладает емкостью для хранения двух значений данных и задержкой равной одному такту. Если останавливающий сигнал *stop* равен нулю, ЭБ работает как обычный синхронный регистр. Если же бит *stop* обращается в единицу и блокирует передачу данных на выходе буфера, ЭБ может принять и сохранить одно дополнительное значение до того, как сигнал *stop* распространится на входной канал. Эффективная реализация ЭБ может быть выполнена, как показано на рис. 2б, на основе двухступенчатого D-триггера путем добавления контроллера (C), управляющего ведущей (M) и ведомой (S) ступенями триггера и вычисляющего выходные биты *valid* и *stop* в зависимости от значений входных битов *valid* и *stop*. Задержка распространения данных и бита *valid* через ЭБ в прямом направлении, а также блокирующего бита *stop* в обратном направлении равна одному циклу тактового сигнала. Таким образом, с помощью ЭБ можно строить эластичные системы, не требующие

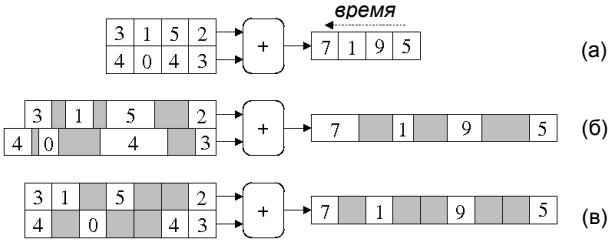

Рис. 1. Синхронный (а), асинхронный (б) и эластичный (в) сумматоры

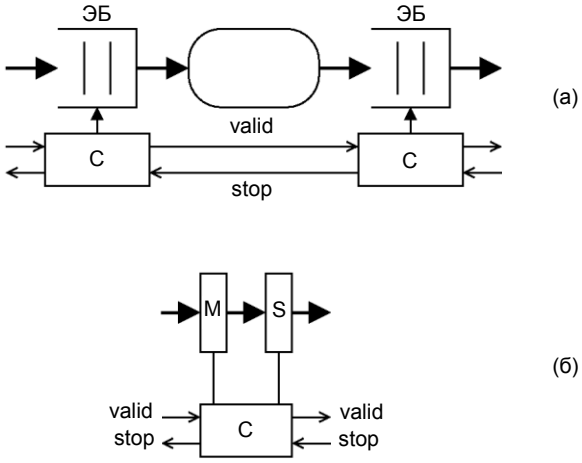

Рис. 2. Простейшая эластичная система (а) и эластичный буфер (б)

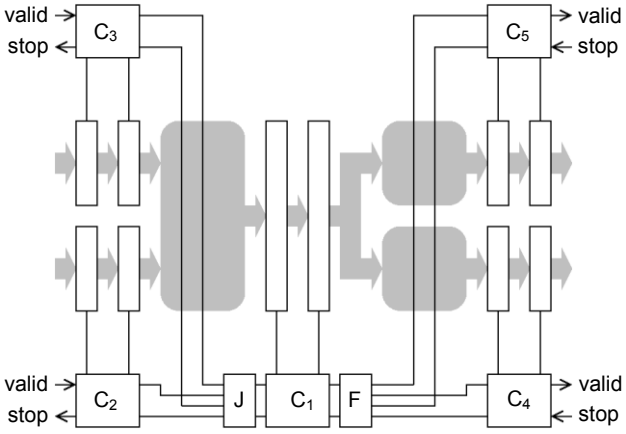

Рис. 3. Эластичный модуль с двумя входными и двумя выходными каналами

глобального распространения бита остановки в течение одного такта.

Конструкцию линейного эластичного конвейера на рис. 2а можно обобщить на случай произвольных эластичных модулей с любым числом входов и выходов. Пример такого модуля приведен на рис. 3. Контроллер C_1 синхронизируется с соседними контроллерами с помощью блоков Join (J) и Fork (F). Блок Join объединяет управляющие сигналы *valid* и *stop* контроллеров C_2 и C_3. Аналогично, блок Fork объединяет управляющие сигналы *valid* и *stop* контроллеров C_4 и C_5. В первом приближении, Join вычисляет логическое И от входных *valid*-битов, тогда как Fork вычисляет логическое ИЛИ от входных *stop*-битов. Конструкция эластичного буфера, блоков Join и Fork, а также других эластичных примитивов подробно описана в [2].

III. Раннее вычисление

Стандартная реализация блока Join обеспечивает корректность, но может приводить к потерям производительности из-за чрезмерной консервативности. Рассмотрим, например, вычисление мультиплексорной функции z = **if** s **then** a **else** b. Условие появления данных на выходе z при использовании блока Join может быть записано как *z.valid = s.valid* **and** *a.valid* **and** *b.valid*. Однако, при *s=1*, выход функции z равен a и может быть вычислен, не дожидаясь прихода значения b. Аналогично, при s=0, выход z равен b и не зависит от входа a. Вход, значение которого может быть проигнорировано в данном вычислении, называется *несущественным*. Для мультиплексорной функции вход a является несущественным при *s=0*, а вход b – при *s=1*. Используя несущественность входов, условие срабатывания блока можно сформулировать более точно, как *z.valid = s.valid* **and** ((*s=1*) **and** *a.valid* **or** (*s=0*) **and** *b.valid*). Реализация такой схемы *раннего вычисления* должна правильно обрабатывать несущественные входные значения, чтобы избежать некорректного повторного срабатывания блока из-за появления на входе данных, относящихся к уже законченному вычислению.

В [3] предложена корректная реализация раннего вычисления на основе модифицированного блока Join с использованием *отрицательных токенов* (или *анти-токенов*), распространяемых в обратном направлении по отношению к потоку данных и уничтожающих несущественные значения.

IV. Оптимизация производительности эластичных систем

Рассмотрим задачу оптимизации простого эластичного конвейера на рис. 4а. Предположим, что вычислительные блоки F и G имеют задержку в 100 единиц времени каждый. Схема может вычислять одно значение за 100 единиц времени. Проводя аналогию с микроархитектурой ЦПУ, можно сказать, что показатель TPI (Time Per Instruction, “время на инструкцию”) равен 100.

Допустим, система выполняет два различных вида вычислений (например, есть два вида инструкций). Вычисления первого типа происходят с высокой вероятностью p, а вычисления второго типа с низкой вероятностью *1-p*. Поведения системы, возникающие при выполнении вычислений первого типа, назовем *типичными* или *базовыми* (CC, от англ. “Critical Core”), а поведения, возникающие в связи с вычислениями второго типа *нетипичными* (LC, от англ. “Little Cares”). CC- и LC-компоненты могут быть выявлены в ходе имитационного моделирования системы и структурно представлены двумя независимыми конвейерами с мультиплексором на выходе, рис. 4б. Производительность после разделения остается прежней: TPI=100.

На следующем шаге CC-часть рассматривается отдельно от LC-части. Допустим, используя различные приемы оптимизации (пример декодера CABAC разобран в разделе VIII), задержку вычислительных блоков в CC-части удалось понизить с 100 до 50 единиц. Улучшить производительность только CC-компоненты часто оказывается намного проще, чем всей системы сразу, так как значительная часть поведений попадает в LC-компоненту и может быть проигнорирована. Хотя производительность базовой части и была повышена, максимальная рабочая частота по-прежнему ограничена критическими путями в нетипичной части: TPI=100.

На последнем шаге (рис. 4в) критические пути в LC-части устраняются путем вставки пустых эластичных буферов. Такое преобразование сохраняет функциональную корректность схемы и делает некритическими большое число маловероятных поведений.

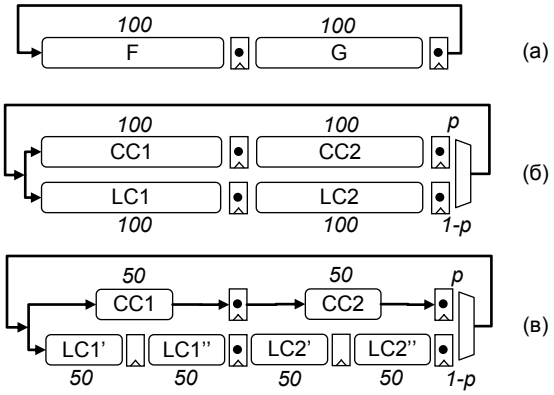

Рис. 4. Оптимизация эластичного конвейера

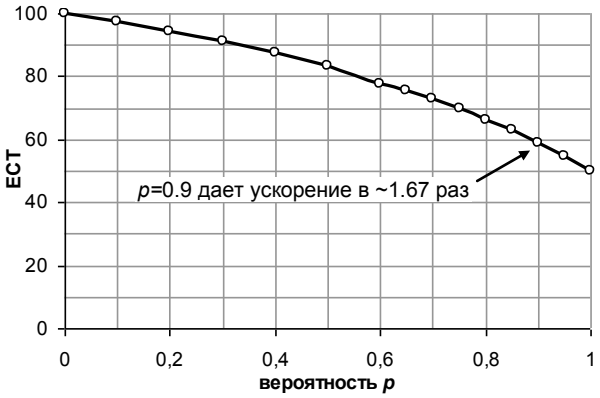

Рис. 5. Зависимость ECT конвейера на рис. 4в от вероятности *p*

Система на рис. 4в содержит два дополнительных ЭБ в LC-части, все вычислительные блоки имеют задержку в 50 единиц. Определим TPI. Допустим, что мультиплексор, объединяющий CC- и LC-части, поддерживает раннее вычисление. В ходе преобразований производительность и частота CC-части возросли вдвое. Частота LC-части также увеличилась вдвое, однако производительность осталась прежней, так как после вставки пустых буферов LC-часть производит рабочие данные только в половине циклов. Производительность всей системы, очевидно, зависит от значения вероятности *p*, т. е. от того насколько часто выход LC-части может быть проигнорирован. График зависимости *эффективного времени цикла* (ECT, effective cycle time) от вероятности *p* представлен на рис. 5. ECT – это аналог показателя TPI для эластичных систем, определяемый как среднее время, затрачиваемое системой на вычисление одного полезного значения [4].

Заметим, что использование схемы раннего вычисления при реализации мультиплексора принципиально важно. Если блок раннего вычисления заменить обычным блоком Join, ECT останется равным 100 единицам и прироста производительности не будет.

Разобранный пример иллюстрирует применение оптимизационного подхода типа “разделяй и властвуй” к синхронным эластичным системам. Подход предполагает использование имитационного моделирования для выделения множества базовых (CC) и нетипичных (LC) поведений системы с последующим разделением логики на CC- и LC- части, оптимизацию производительности CC-части, повышение частоты LC-части вставкой пустых эластичных буферов и объединение CC- и LC-частей с применением раннего вычисления.

V. Аппаратный декодер H.264 CABAC

Декодер CABAC (Context Adaptive Binary Arithmetic Code) – это критический по производительности блок видеодекодера H.264 [5], выполняющий обратное энтропийное преобразование. Как показано на рис. 4, декодер CABAC принимает на вход последовательность *битов* в адаптивном арифметическом коде и восстанавливает последовательность *бинов* (термин бин используется для обозначения бита декодированных видеоданных), которая затем преобразуется в лексемы видеопотока.

Алгоритм адаптивного арифметического кодирования основан на предсказании частоты (вероятности) появления 0 или 1 в потоке бинов. Декодер CABAC вычисляет бит *MPS* (Most Probable Symbol, наиболее вероятное значение) и вероятность *p* его появления на текущем такте, представленную описанным в стандарте H.264 6-битным кодом [5]. Эффективность сжатия алгоритмом CABAC напрямую зависит от достоверности предсказания *MPS*. Значения *MPS* и вероятности *p* корректируются на каждом шаге декодирования в зависимости от правильности сделанного предсказания. Выбор *MPS* и *p* также зависит от текущего *контекста*, т.е. от положения текущего бита в синтаксической структуре видеопотока. Алгоритм H.264 CABAC выделяет 449 различных контекстов, каждый со своей парой переменных *MPS* и *p*. В аппаратной реализации декодера текущие значения *MPS* и *p* для каждого из контекстов хранятся в контекстной памяти (регистровом файле) состоящей из 449 7-битных слов с одним портом чтения и одним портом записи.

Слово контекстной памяти для контекста с номером *i* хранит состояние (MPS_i, p_i) конечного автомата, используемого для предсказания вероятностей в части видеопотока, относящейся к данному контексту. На каждом цикле декодирования только один автомат, связанный с текущим контекстом, воспринимается декодером CABAC, как активный. Состояние (MPS_i, p_i) активного автомата используется для предсказания и получает новое значение в конце цикла, вычисляемое по таблице StateLUT, как функция текущего значения и флага достоверности предсказания. Номер текущего контекста вычисляется дебинаризаторами, осуществляющими лексический анализ потока бинов, и определяет очередной адрес для чтения из контек-

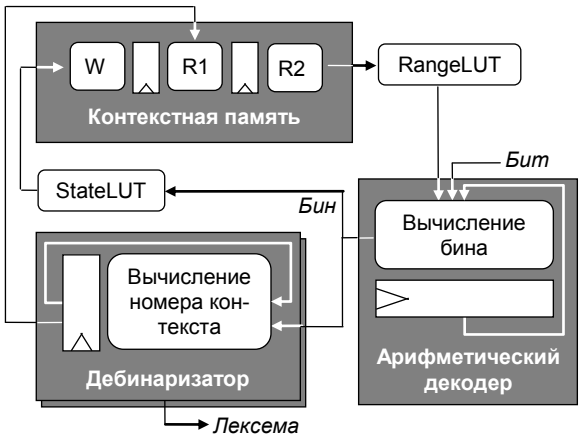

Рис. 5. Упрощенная микроархитектурная диаграмма аппаратного декодера H.264 CABAC

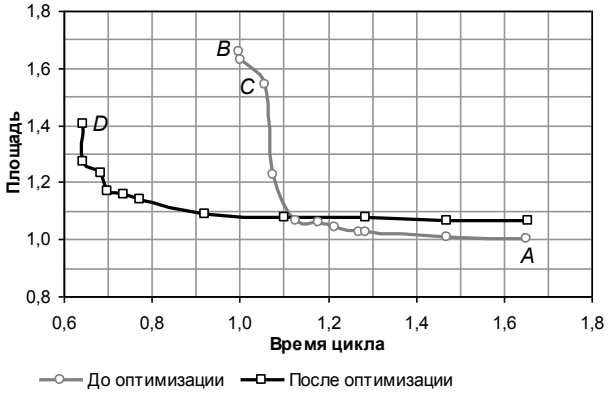

Рис. 6. Графики "быстродействие-площадь" декодера H.264 CABAC до и после оптимизации

стной памяти. Всего существует около десяти дебинаризаторов для разных типов лексем. Чтение из контекстной памяти происходит с задержкой в один цикл; прочитанное значение вероятности *p* поступает в таблицу RangeLUT и преобразуется в границу интервала для алгоритма арифметического декодирования [6].

Суммарная задержка портов записи и чтения контекстной памяти равна двум циклам, поэтому обновленное состояние, записанное на цикле *N*, может быть прочитано не раньше цикла *N+2*. Если декодирование происходит без смены контекста, значения состояния передается по обходному пути с задержкой в один цикл (не показан на рис. 5).

Логическая схема аппаратного декодера CABAC содержит несколько критических путей, проходящих через логику вычисления контекста, логику чтения и записи контекстной памяти, таблицы RangeLUT и StateLUT, а также подблок арифметического декодирования. Вычисления на последовательных циклах взаимозависимы, так как значение бина на текущем цикле определяет значение контекста на следующем. Вероятность правильно предсказать значение бина не всегда высока, поэтому оптимизация с помощью разворачивания нескольких итераций декодирования не дает ожидаемого прироста производительности и сопряжена с высокими накладными расходами. Мы предлагаем альтернативный подход, основанный на использовании эластичности и анализа типичных поведений системы.

VI. Оптимизация эластичного декодера

Прежде всего, заметим, что с применением подходов из [2], [3] и разработанных нами прототипов средств САПР, исходное неэластичное описание блока на языке Verilog можно автоматизировано преобразовать в эластичную форму. Транслятор выполняет локальную модификацию исходного RTL-кода (например, происходит замена регистров на D-триггеры эластичных буферов) и генерирует описание управляющей логики, используя граф потоков данных и управления системы [2].

Применение метода из раздела IV требует знания статистических оценок вероятностей различных поведений системы. Мы использовали стандартную функциональную модель декодера H.264 для сбора статистики на тестовых видеопотоках. Профилирование показало, что частоты различных типов лексем в видеопотоке распределены крайне неравномерно. А именно, от 90% бинов потока приходится на описание коэффициентов преобразования косинусов (residual DCT coefficients) и векторов перемещения (motion vectors). С позиции микроархитектуры в обработке этих типичных лексем задействованы только два дебинаризатора и примерно половина объема контекстной памяти. Дебинаризатор c_dec используется для коэффициентов DCT, а mv_dec – для векторов перемещения. Оставшиеся дебинаризаторы и часть регистрового файла попадают в LC-компоненту и участвуют в обработке не более 10% декодированного видеопотока.

Таким образом, микроархитектура декодера может быть преобразована по схеме на рис. 4. При этом CC-часть, включающая c_dec, mv_dec, половину контекстной памяти и арифметический декодер оптимизируется отдельно. В результате оптимизации повышается частота и показатель ECT CC-части. Затем частоты LC-части и CC-части уравнивается вставкой пустых эластичных буферов в критические пути LC-части. Для объединения частей используется мультиплексор с ранним вычислением.

Для повышения быстродействия CC-части декодера мы использовали цепочку преобразований, таких как ретайминг эластичных буферов, преобразование Шеннона для логических функций и автоматов, реструктуризация таблиц, предвычисление номера контекста, а также разделение на LC- и CC-компоненты для подблоков декодера.

Оптимизированный декодер работает на более высокой частоте, но может тратить более одного такта на вычисление выходного значения. Используя полученные ранее оценки вероятностей и имитационное моделирование управляющей логики, мы получили оценку *пропускной способности* эластичного декодера равную 0,91 значений за такт. То есть, выход декодера в среднем простаивает 9 циклов из 100.

VII. Результаты эксперимента

Все оптимизирующие преобразования были реализованы в описании декодера на языке Verilog. Для упрощения задачи все LC-дебинаризаторы были удалены из исходного кода на начальном этапе (их добавление после оптимизации не приведет к ухудшению производительности). CC-дебинаризатор mv_dec также был исключен из рассмотрения из соображений структурного подобия дебинаризатору c_dec.

После каждого преобразования выполнялся логический синтез и привязка к технологической библиотеке с помощью промышленных средств САПР. Для проверки правильности окончательной версии декодера использовалось имитационное моделирование на тестовом видеопотоке. Кривые “быстродействие-площадь” для исходного и оптимизированного блока представлены на рис. 6. Данные были получены для 32-нм технологического процесса после этапа привязки к библиотеке, но до размещения и трассировки. Каждая точка на графике соответствует нормализованным оценкам времени цикла и площади после одного запуска средства САПР. Значение площади 1,0 соответствует минимальной площади исходного блока (рис. 6, точка A), время цикла 1,0 – минимальному времени цикла (рис. 6, точка B).

Ухудшение площади после оптимизации при больших значениях времени цикла (от 1,1 до 1,7) объясняется увеличением числа логических вентилей в ходе микроархитектурных преобразований. Оптимизированный декодер обладает лучшей производительностью в диапазоне времен цикла от 0,6 до 1,1 и площадей от 1,1 до 1,4. Например, время цикла в точке D на графике примерно на 40% меньше, чем в точке C. С учетом величины пропускной способности (раздел VI) равной 0,91, уменьшение ECT составляет ~34%, а эффективная частота (число операций в единицу времени) возрастает примерно в 1,5 раза. В целом, при изменении времени цикла от 0,6 до 0,9 получается выигрыш в производительности в 1,25-1,5 раза.

Для некоторых точек на рис. 6 были также проведены эксперименты по размещению и трассировке. Предварительные данные позволяют сделать вывод о возможности эффективной физической реализации оптимизированной схемы декодера.

VIII. Заключение

Мы провели экспериментальное исследование применимости методов проектирования синхронных эластичных схем к оптимизации промышленного аппаратного декодера CABAC H.264. Используя новый метод повышения быстродействия синхронной эластичной системы, основанный на характеризации множества ее типичных поведений, мы достигли прироста производительности декодера в 1,25-1,5 раза (в диапазоне высоких частот). Мы показали правильность работы декодера после оптимизации средствами имитационного моделирования. Для проверки возможности эффективной реализации схемы на кристалле были проведены предварительные эксперименты по размещению и трассировке.

На основании собранных данных, мы пришли к выводу, что методы построения схем нечувствительных к тактовым задержкам дают значительное преимущество при проектировании блоков промышленных ИС.

Благодарности

Мы благодарим проф. Джорди Кортаделлу (UPC, Барселона, Испания) за сотрудничество в разработке технологии SELF и Алексея Поспелова за вклад в создание прототипов средств САПР синхронных эластичных схем. Мы благодарим сотрудников Intel Corporation: Дмитрия Лукьянова за предложение использовать декодер H.264 CABAC в эксперименте, Майка Коултера, Джей Эполайт и Илью Брайловского за полезные обсуждения, Андрея Аюпова, Леонида Крагинского, Антона Сорокина и Николая Рыженко за помощь в настройке и использовании промышленных средств САПР.

Литература

[1] L. Carloni, A. Sangiovanni-Vincentelli. Coping with latency in SoC design. // IEEE Micro. - 2002. - V. 22. - № 5.

[2] J. Cortadella, M. Kishinevsky, and B. Grundmann. Synthesis of synchronous elastic architectures. // In Proc. ACM/IEEE Design Automation Conference. - 2006.

[3] J. Cortadella, M. Kishinevsky. Synchronous elastic circuits with early evaluation and token counterflow. // In Proc. ACM/IEEE Design Automation Conference. - 2007.

[4] J. Júlvez, J. Cortadella, M. Kishinevsky. On the performance evaluation of multi-guarded marked graphs with single-server semantics. // Discrete Event Dynamic Systems. - 2010. - V. 20. - № 3.

[5] H.264. Advanced video coding for generic audiovisual services. // ITU-T Recommendation. - 2005.

[6] D. Marpe, H. Schwarz, T. Wiegand. Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard. // IEEE Trans. on Circuits and Systems for Video Technology. - 2003. - V. 13. - № 7.

[7] W.J. Dally, B. Towles. Principles and Practices of Interconnection Networks. - Morgan Kaufmann, 2003.