УДК 004.032.6; 004.272.3

Ал.А. Петровский, А.В. Станкевич, А.А. Петровский

КОНВЕЙЕРНАЯ АРХИТЕКТУРА ДЕКОДЕРА CABAC СТАНДАРТА H.264/AVC ДЛЯ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

Описывается архитектура декодера CABAC для мобильных приложений с разрешением до 625SD с трехступенчатым конвейером, позволяющая обеспечить декодирование одного бина за такт. Декодер совместим с профилями high profile, high 10 profile, high 4:2:2 profile, поддерживает режим MBAFF и блоки 8 × 8, а также масштабируем как по разрешению, так и по поддерживаемым инструментам декодирования, описанным в стандарте H.264. Выполняется сравнение с известными реализациями прототипа декодера CABAC на FPGA фирмы Xilinx.

Введение

В настоящее время наблюдается значительный рост спроса на высококачественное видео для мобильных устройств. При передаче и хранении видеоданных одной из ключевых проблем является компрессия этих данных при сохранении требуемого качества изображения. Для цифрового телевидения высокого разрешения (HDTV), мобильных телефонов, смартфонов, Blu-ray-дисков, потокового видео в Интернете и других приложений цифрового видео ITU-T и ISO/IEC рекомендован стандарт H.264/AVC (Advanced Video Coding: MPEG-4 Part 10) [1–6]. Высокая степень сжатия видеоданных в данном стандарте обеспечивается за счет большой вычислительной сложности, что требует высокой производительности аппаратуры при декодировании в реальном масштабе времени. Например, для видео с разрешением HDTV требуются производительность процессора порядка 83 гигаинструкций в секунду (GIPS) и пропускная способность подсистемы памяти 70 ГБ/с [7]. Реализация видеодекодеров на универсальных микропроцессорах с такими требованиями производительности вызывает серьезные затруднения, особенно для мобильных приложений, из-за высокого энергопотребления. Более предпочтительной является разработка специализированных вычислительных устройств на базе параллельно-поточной архитектуры [8].

Среди средств стандарта H.264, обеспечивающих высокую степень сжатия видеопотока, большую роль играет энтропийное кодирование синтаксических единиц и особенно контекстно-зависимое адаптивное двоичное арифметическое кодирование CABAC (context-adaptive binary arithmetic coding). Данный вид кодирования обеспечивает более высокую степень сжатия, чем контекстно-зависимое адаптивное кодирование с переменной длиной кодового слова CAVLC (context-adaptive variable-length coding), и на 9–14 % снижает требования по скорости передачи данных [9].

Известен ряд реализаций декодера CABAC как специализированных вычислительных устройств [10–13]. Реализации [10–12] не поддерживают такие важные инструменты стандарта H.264 для высокого профиля (high profile), как кодирование MBAFF (macroblock adaptive frame field) и кодирование с использованием блоков 8 x 8. Реализация [13] ориентирована на приложения HDTV и содержит избыточные аппаратные средства для получения требуемой высокой производительности. Так, для повышения производительности используются два вычислительных ядра для процесса DecodeDecision и четыре ядра для процесса DecodeBypass с соответствующими схемами управления. Избыточные аппаратные средства приводят к повышению энергопотребления, что критично для мобильных приложений.

Целью работы являлась разработка архитектуры и реализация прототипа декодера CABAC стандарта H.264 для мобильных приложений на базе FPGA с разрешением до 625SD (720x576 отсчетов яркости), частотой смены кадров 30 кадров/с и поддержкой профилей high profile, high 10 profile, high 4:2:2 profile, включая кодирование MBAFF и использование блоков 8 x 8.

1. Процесс декодирования стандарта H.264

Вследствие неоднозначного перевода на русский язык в различных источниках терминов стандарта H.264 далее будем использовать англоязычные названия процессов декодирования и обозначения переменных в соответствии со стандартом. Русскоязычные термины будут поясняться в скобках англоязычными оригинальными терминами.

Восстановление отдельных кадров видеосигнала из закодированного видеопотока по стандарту H.264/AVC MPEG-4 Part 10 выполняется в соответствии с блок-схемой (рис. 1). Здесь входной видеопоток представляет собой бинарную последовательность (bitstream) закодированных кадров изображения, которые разбиты на секции (в частном случае размер секции равен кадру), состоящие из цепочки синтаксических единиц (СЕ) макроблоков размером 16x16 отсчетов яркости и соответствующих им отсчетов цветности. Входной видеопоток декодируется одним из декодеров: декодером экспоненциальных кодов Голомба, декодером CAVLC или декодером CABAC – в соответствии с типом СЕ. Полученный результат декодирования – это значения отдельных СЕ и блоков остаточных данных (residual). Блок остаточных данных представляет собой набор разностных отсчетов яркости и цветности между ссылочным и текущим кадрами, закодированных дискретным косинусным преобразованием. Таким образом, формирование разностного кадра получается путем применения обратного косинусного преобразования к блоку восстановленных остаточных данных после деквантования. Восстановленный кадр получается сложением результирующего разностного кадра с кадром-прогнозом, который в режиме intra строится из ранее декодированных отсчетов текущего кадра или в режиме inter по одному или двум ранее восстановленным ссылочным кадрам, полученным с помощью векторов из блока компенсации движения (motion compensation). Для устранения артефактов на границах макроблоков восстановленный кадр фильтруется (deblocking filtering).

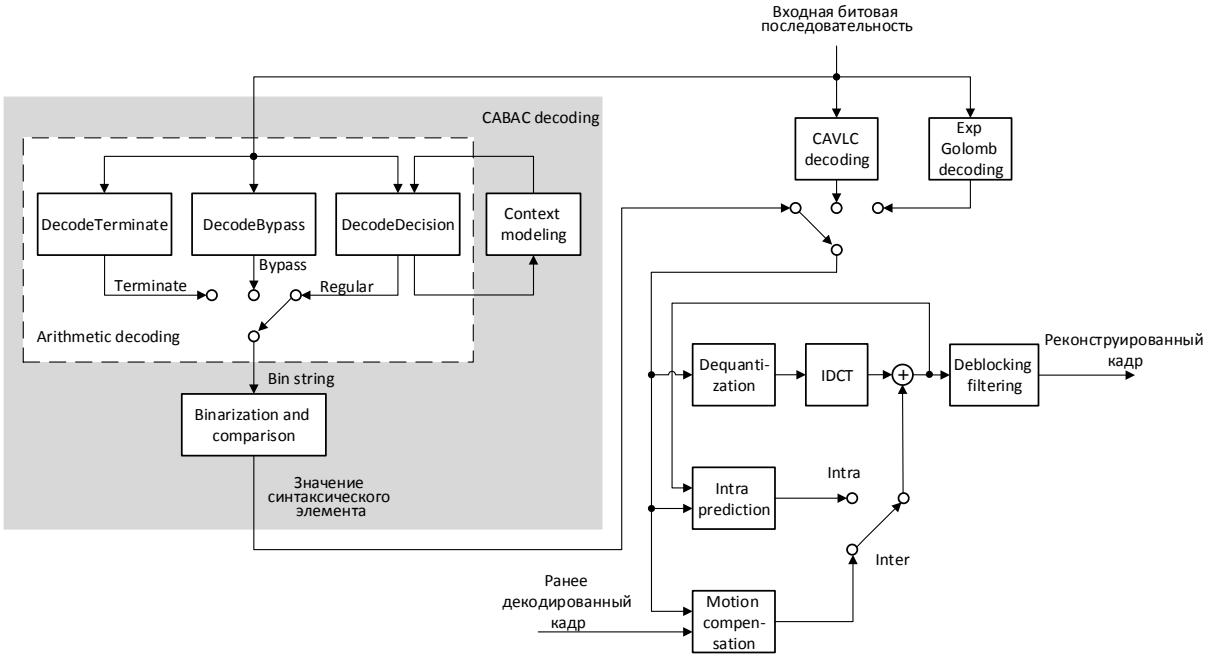


Рис. 1. Процесс декодирования стандарта H.264

В декодере CABAC входная кодированная битовая последовательность декодируется в последовательность бинов (bin string), которой впоследствии с помощью процесса бинаризации ставится в соответствие образец последовательности бинов для данной СЕ. При совпадении декодированной последовательности бинов с образцом бинаризации определяется искомое значение анализируемой СЕ.

Двоичное арифметическое декодирование использует только два символа. В CABAC они именуются как наиболее вероятный символ MPS и наименее вероятный символ LPS. Символы MPS и LPS могут быть как нулем, так и единицей.

Для каждого декодируемого бина осуществляется выбор контекстной модели, количественно характеризующейся двумя контекстными переменными *pStateIdx* и *valMPS*. Переменная *pStateIdx* является индексом вероятностного состояния и позволяет определить часть кодового интервала, соответствующего вероятности наименее вероятного символа LPS. Переменная *valMPS* хранит значение наиболее вероятного двоичного символа. Контекстная модель может быть выбрана из некоторого конечного множества доступных моделей в зависимости от ранее декодированных значений этой же СЕ для соседних макроблоков и ранее декодированных значений бинов текущей СЕ. Этот процесс называется контекстным моделированием. Таким образом, каждый бин декодируется согласно выбранной вероятностной модели. По результатам декодирования бина выбранная контекстная модель обновляется [14].

Арифметическое кодирование основывается на делении кодового интервала на подинтервалы в зависимости от вероятности появления символов. В стандарте H.264 данный процесс называется DecodeDecision. Исходный кодовый интервал *codIRange* делится на две части: *codIRangeLPS* и *codIRangeMPS* – в соответствии с выражениями

$$codIRangeLPS = codIRange \cdot PLPS, \quad codIRangeMPS = codIRange - codIRangeLPS,$$

где *PLPS* – вероятность LPS.

Далее проводится сравнение текущего значения переменной *codIOffset* со значением *codIRangeMPS* и в зависимости от результата сравнения формируются значение текущего бина и новая контекстная модель. Переменная *codIOffset* заполняется битами входной кодированной последовательности.

При декодировании некоторых синтаксических единиц или частей синтаксических единиц может использоваться процесс DecodeBypass, имеющий меньшую вычислительную сложность, чем рассмотренный процесс декодирования DecodeDecision. Характерной особенностью процесса DecodeBypass является отсутствие изменений значений переменной *codIRange*.

При декодировании синтаксической единицы *end_of_slice_flag*, а также при декодировании бина синтаксической единицы *mb_type*, указывающего режим I_PCM, должен реализовываться процесс DecodeTerminate стандарта H.264. В стандарте отмечается, что процесс DecodeTerminate может быть реализован как DecodeDecision при значениях переменных контекстной модели *pStateIdx* = 63 и *valMPS* = 0 (значение *pStateIdx* = 63 для других контекстных моделей встретиться не может). Однако в стандарте не отмечается то важное обстоятельство, что процесс ренормализации для DecodeTerminate по сравнению с DecodeDecision выполняется только для случая, когда $codIOffset \geq codIRange$.

2. Декодер CABAC

2.1. Организация вычислительного процесса декодера CABAC

Проведем анализ алгоритма декодирования стандарта H.264 (рис. 2).

Шаги алгоритма для декодирования одного бина нельзя выполнить за один такт, и их прямая последовательная реализация потребует не менее трех тактов. Рассмотрим возможности распараллеливания вычислительного процесса алгоритма и организации конвейерных вычислений.

Из приведенной на рис. 2 последовательности шагов алгоритма параллельно в одном такте могут выполняться шаги 4 и 5, поскольку они будут реализовываться разными блоками декодера.

Третий шаг не может быть совмещен в одном такте ни с одним из других шагов, поскольку вычисление значения контекстного индекса *ctxIdx* и получение текущей контекстной модели должны осуществляться перед декодированием очередного бина. Полученное значение бина используется для процесса бинаризации и вычисления контекстного индекса для следующего бина.

Для того чтобы арифметический декодер, реализующий вычисления на шаге 3, без простоев проводил вычисления в каждом такте, необходимо совмещение в одном такте шагов 1 и 2. Тогда можно будет построить трехступенчатый конвейер со следующим распределением шагов алгоритма: первая ступень – шаги 1 и 2, вторая ступень – шаг 3, третья ступень – шаги 4 и 5. При этом следует учитывать то обстоятельство, что в процессе декодирования возможно возникновение ситуа-

ции, когда следующий декодируемый бин использует то же значение *ctxIdx*, что и предыдущий. Выполнение шага 3 на такт позже, чем шагов 1 и 2, приведет в этом случае к получению неправильной контекстной модели для шага 3, поскольку она будет корректно обновлена только в следующем такте. Модификация алгоритма декодирования одного бина, позволяющая совместить в одном такте шаги 1 и 2 и выполнить их параллельно шагу 3, показана на рис. 3.

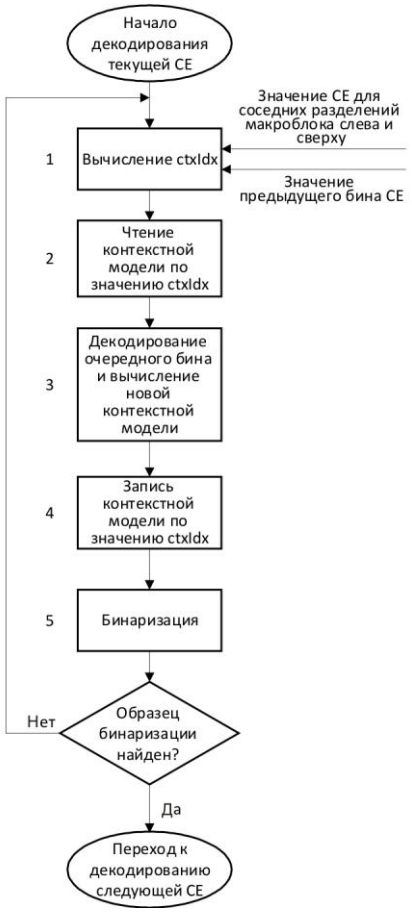


Рис. 2. Алгоритм декодирования одной СЕ

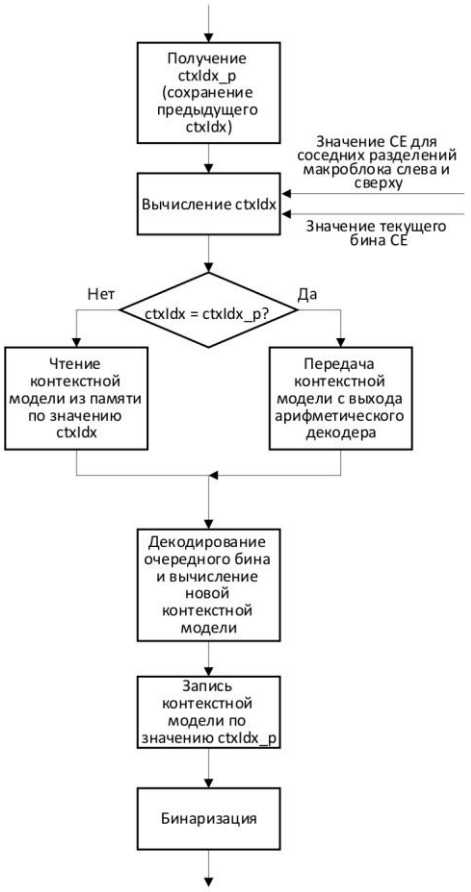


Рис. 3. Модифицированный алгоритм декодирования

Предлагаемая модификация алгоритма предполагает задержку на такт значения *ctxIdx* (на рис. 3 задержанное значение обозначено как *ctxIdx_p*). Если значения *ctxIdx* и *ctxIdx_p* совпадают, то в качестве контекстной модели для декодирования очередного бина должна быть выбрана модель с выхода арифметического декодера, а не прочитанная из памяти контекстных моделей. Значение *ctxIdx_p* также понадобится при обновлении контекстной модели. Поскольку арифметическое декодирование будет происходить на такт позже, чем расчет *ctxIdx*, при вычислении *ctxIdx* необходимо использовать значение текущего декодированного бина данной СЕ.

Реализация вычислительного процесса декодирования с учетом сделанных замечаний представлена в табл. 1.

2.2. Архитектура декодера CABAC

Для реализации поточных вычислений одного бина за такт работы декодера (рис. 4) был организован трехступенчатый конвейер в соответствии с табл. 1. Такт работы конвейера равен такту сигнала частоты синхронизации. В табл. 2 приведены блоки декодера, входящие в соответствующие ступени конвейера, и указаны операции, выполняемые ими. Следует иметь в виду, что вычислительные операции выполняются в течение такта, а фиксация результатов происходит по нарастающему фронту сигнала синхронизации. Все блоки декодера работают параллельно, однако обрабатывают информацию для различных бинов в соответствии с номером ступени конвейера. После завершения процесса декодирования текущего бина контекстная модель для следую-

щего бина фиксируется во входных регистрах второй ступени конвейера, новая контекстная модель сохраняется в памяти контекстных моделей по значению индекса *ctxIdx_p*, а декодированный бин заносится во входной сдвиговый регистр процессора образца бинаризации.

Таблица 1

Поточный вычислительный процесс декодирования

Ступень конвейера	Операции процесса декодирования	
1	Вычисление *ctxIdx* с учетом декодированного значения бина, полученного во второй ступени	
	Передача на вход арифметического декодера контекстной модели с выхода арифметического декодера в случае *ctxIdx* = *ctxIdx_p*, в противном случае – передача на вход арифметического декодера контекстной модели, прочитанной из памяти контекстных моделей по *ctxIdx*	
2	Декодирование очередного бина. Вычисление и сохранение новой контекстной модели по значению *ctxIdx_p*	Получение *ctxIdx_p* (сохранение предыдущего *ctxIdx*)
3	Бинаризация	

В начале каждой секции осуществляется инициализация памяти контекстных моделей. Для этого из памяти контекстных таблиц считываются пары значений переменных стандарта H.264 *m* и *n* и с помощью процессоров переменных контекстной модели рассчитываются значения *pStateIdx* и *valMPS*. Для ускорения процесса инициализации память контекстных таблиц располагает широкой 256-разрядной выходной шиной данных и в декодере имеется 16 идентичных параллельно работающих процессоров переменных контекстной модели. Начальные контекстные модели обновляются в процессе работы декодера.

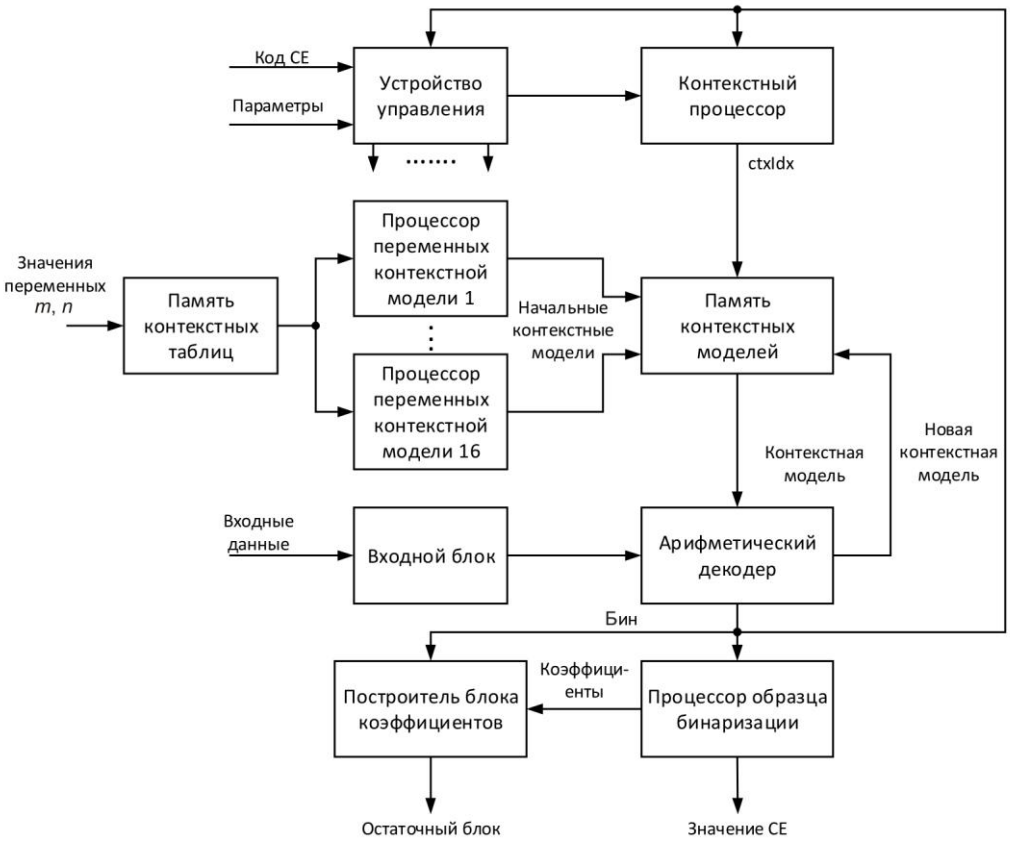


Рис. 4. Структура декодера CABAC

Таблица 2

Поточный вычислительный процесс декодера CABAC

Ступень конвейера	Блок декодера	Такт работы конвейера		
		$k-1$	k	$k+1$
1	Контекстный процессор. Память контекстных моделей	Вычисление $ctxIdx_i$ с учетом bin_{i-1}, выбор контекстной модели для $ctxIdx_i$	Вычисление $ctxIdx_{i+1}$ с учетом bin_i, выбор контекстной модели для $ctxIdx_{i+1}$	Вычисление $ctxIdx_{i+2}$ с учетом bin_{i+1}, выбор контекстной модели для $ctxIdx_{i+2}$
2	Арифметический декодер. Память контекстных моделей	Вычисление bin_{i-1}, вычисление и сохранение новой контекстной модели для $ctxIdx_{i-1}$	Вычисление bin_i, вычисление и сохранение новой контекстной модели для $ctxIdx_i$	Вычисление bin_{i+1}, вычисление и сохранение новой контекстной модели для $ctxIdx_{i+1}$
3	Процессор образца бинаризации	Вычисление образца бинаризации для bin_{i-2}	Вычисление образца бинаризации для bin_{i-1}	Вычисление образца бинаризации для bin_i

Для хранения полных контекстных таблиц стандарта H.264 необходима память общим объемом 8 Кбайт (8192 значения в диапазоне от –128 до +127). Полная память контекстных таблиц реализуется как массив из 1024 64-разрядных слов. Поскольку часть контекстных моделей может не использоваться для каких-то конкретных профилей стандарта H.264, объем памяти контекстных таблиц может быть сокращен.

Полная память контекстных моделей имеет объем 1024 7-разрядных слова. Каждое слово хранит пару переменных контекстной модели *pStateIdx* и *valMPS*.

Входные закодированные данные поступают в декодер по 32-разрядной шине данных (входная битовая последовательность «нарезана» 32-разрядными словами). Разрядность шины данных выбрана исходя из разрядности современных микропроцессоров. Входной блок обеспечивает формирование 7-разрядного окна, смещающегося над 32-разрядным словом входных данных. Смещение задается числом бит, потребленных арифметическим декодером при декодировании. Необходимость формирования окна связана с тем обстоятельством, что при декодировании очередного бина может быть потреблено заранее неизвестное число бит входной закодированной последовательности. Окно будет всегда выравниваться по биту входного слова, соответствующему первому биту для очередного декодируемого бина. По мере декодирования бинов окно продвигается по входному 32-разрядному слову, начиная с первого еще не использованного при декодировании бита входного слова. Размер окна определяется максимальным числом бит, которое может быть потреблено декодером при операции ренормализации.

По значению контекстного индекса *ctxIdx* из памяти контекстных моделей извлекается соответствующая контекстная модель, которая используется при декодировании текущего бина. Значение контекстного индекса *ctxIdx* формируется с помощью контекстного процессора в соответствии с кодом декодируемой СЕ, текущими значениями параметров (типов секции и соседних макроблоков, индексов разделений и т. п) и значениями этой же СЕ для соседних слева и сверху макроблоков или разделений макроблоков. Диапазон возможных значений *ctxIdx* для каждого типа СЕ ограничен стандартом, поэтому для исключения операции сложения при расчете *ctxIdx* проведены предварительные вычисления с целью замены сумматоров мультиплексорами. Это позволяет уменьшить критическую задержку за счет удаления схем межразрядного переноса.

Для декодирования ряда СЕ (*mb_skip_flag, mb_field_decoding_flag, coded_block_pattern, mb_type, ref_idx, mvd, intra_chroma_pred_mode, coded_block_flag, transform_size_8x8_flag*) требуется информация о соседних слева и сверху значениях этой же СЕ для соседних разделений макроблока или соседних макроблоков с адресами mbAddrA и mbAddrB [14]. Для этого контекстный процессор содержит регистровые файлы для хранения значений СЕ для верхней строки макроблоков секции или значений СЕ для нижних макроблоков пары верхней строки

пар макроблоков при наличии такого деления. Запись в указанные регистры происходит по завершении декодирования макроблока. Кроме того, имеются регистровые файлы для хранения значений СЕ для верхних макроблоков пары макроблоков, которые используются для случая полевых пар макроблоков для кадров MBAFF.

В соответствии с выбранной контекстной моделью в арифметическом декодере осуществляется декодирование очередного бина. Декодированный бин поступает в процессор образца бинаризации для сравнения с образцами последовательности бинов для данной СЕ. Декодированное значение СЕ определяется при совпадении декодированной последовательности бинов с образцом бинаризации.

Декодированное значение СЕ или остаточный блок сопровождается сигналом готовности. Одновременно с сигналом готовности формируется сигнал, стробирующий значение числа бит входной последовательности, использованных для декодированной текущей синтаксической единицы (сигнал числа потребленных бит и сигнал готовности на рис. 4 не показан).

Все СЕ, кодируемые CABAC, разделены на две группы: одиночные СЕ и СЕ блока остаточных данных. К одиночным СЕ относятся: *mb_skip_flag*, *mb_field_decoding_flag*, *end_of_slice_flag*, *mb_type*, *transform_size_8x8_flag*, *coded_block_pattern*, *mb_qp_delta*, *prev_intra4x4_pred_mode_flag*, *rem_intra4x4_pred_mode*, *prev_intra8x8_pred_mode_flag*, *rem_intra8x8_pred_mode*, *intra_chroma_pred_mode*, *ref_idx_l0*, *ref_idx_l1*, *mvd_l0*, *mvd_l1*, *sub_mb_type*. К СЕ блока остаточных данных относятся: *coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, coeff_abs_level_minus1, coeff_sign_flag*.

Одиночные СЕ декодируются поодиночке. Все СЕ блока остаточных данных обрабатываются последовательно друг за другом с помощью построителя блока коэффициентов до тех пор, пока не будут декодированы все уровни коэффициентов дискретного косинусного преобразования соответствующего блока остаточных коэффициентов (до 64 коэффициентов). Обработка полного блока коэффициентов как единой СЕ позволяет сократить накладные расходы, связанные с простаиванием декодера CABAC в паузах между завершением обработки предыдущей СЕ и началом обработки следующей.

В процессоре образца бинаризации осуществляется преобразование последовательного кода (бинов от арифметического декодера) в параллельный код для подбора образца бинаризации. Подбор образца осуществляется в зависимости от кода типа бинаризации, поступающего от устройства управления.

При декодировании СЕ, и особенно блоков остаточных данных, возможна ситуация, когда 32 разрядов входного слова будет недостаточно для кодирования значения текущей СЕ. В этом случае декодер CABAC установит сигнал cabac_shift_en на один период сигнала синхронизации (рис. 5), а на выход числа потребленных битов будет установлено значение 32 (при декодировании потреблено 32 бита). Сигнал готовности cabac_ready установлен не будет. Работа декодера будет приостановлена. Ведущее устройство (декодер H.264) должно в течение такта установить на вход cabac_data_input новое 32-разрядное слово данных, после чего в следующем такте процесс декодирования продолжится. На рис. 5 приняты следующие обозначения сигналов: start_cabac – сигнал разрешения работы декодера; clk – сигнал синхронизации; gclk_cabac – клапонированный сигнал синхронизации; cabac_consumed_bits_len – число бит, потребленных при декодировании; cabac_decoding_output – выход процессора образца (не используется при декодировании остаточного блока); coefflevel_i – уровень *i*-го коэффициента дискретного косинусного преобразования. Декодирование осуществляется в порядке, обратном порядку построения карты значащих коэффициентов остаточного блока [14]. В данном примере понадобились дополнительные биты закодированной последовательности для декодирования coefflevel_0.

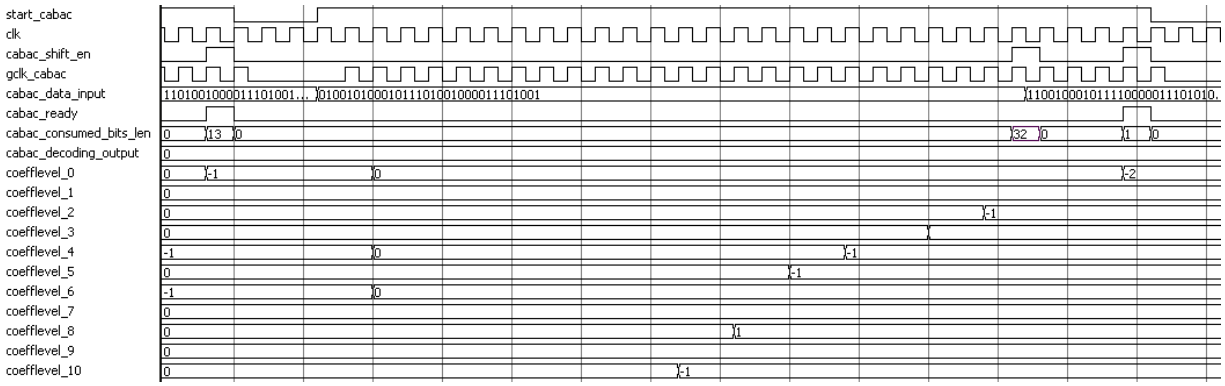
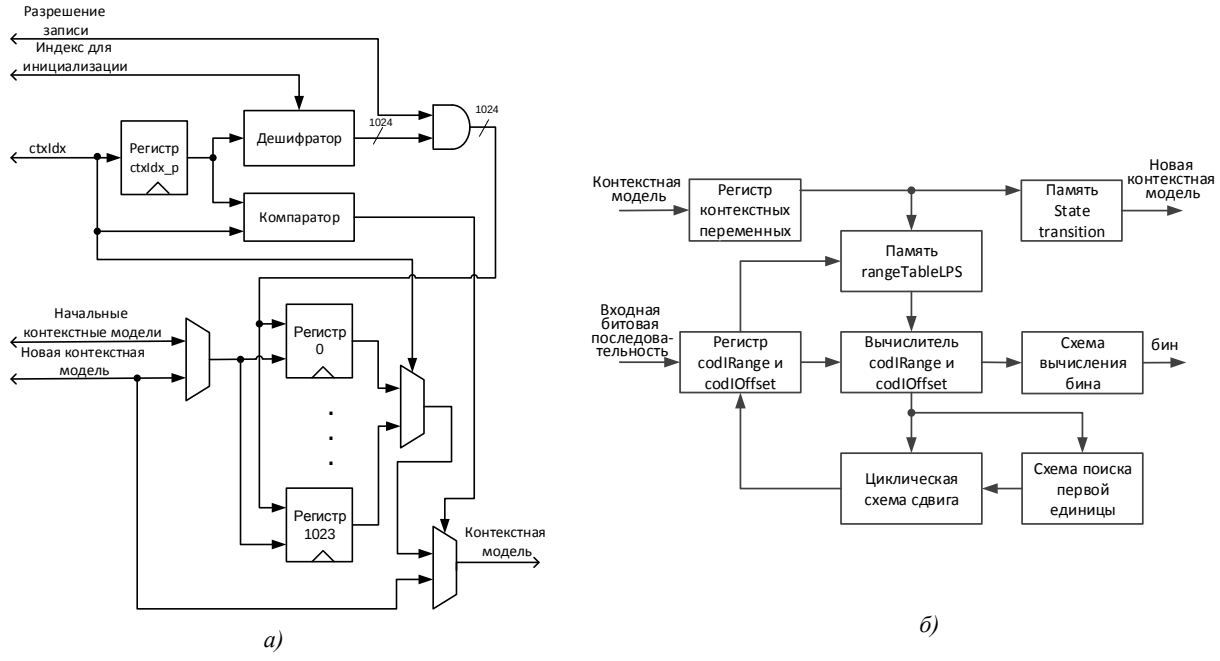


Рис. 5. Временная диаграмма декодирования блока остаточных данных с подкачкой очередного входного слова из внешнего буфера

2.3. Память контекстных моделей

Контекстные модели должны храниться в загружаемой памяти. Текущее значение индекса *ctxIdx* адресует конкретную контекстную модель. Память должна быть многопортовой, поскольку в нее необходимо одновременно записывать новое значение контекстной модели по индексу *ctxIdx_p* (синхронная запись) и читать контекстную модель по индексу *ctxIdx* (асинхронное чтение). Кроме того, должна быть предусмотрена запись начальных моделей в начале каждой секции через третий порт либо требуется использовать мультиплексор данных для входного порта. Из приведенных требований следует, что синхронную память использовать в данном случае нельзя, поскольку для такой памяти результат чтения появится на выходе с задержкой на такт относительно появления текущего индекса *ctxIdx*. Память с одним или двумя синхронными портами для записи и асинхронным портом для чтения достаточно сложно реализуется в заказной микросхеме. Поэтому для того чтобы чтение из памяти контекстных моделей происходило в том же такте, что и вычисление индекса *ctxIdx*, память была реализована на регистрах (рис. 6, *а*).

Количество 7-разрядных регистров на рис. 6, *а* приведено для случая хранения полного числа контекстных моделей стандарта H.264.



а) *б)*

Рис. 6. Структура блоков декодера: *а)* память контекстных моделей; *б)* арифметический декодер

В начале каждой секции по индексу для инициализации памяти и сигналу разрешения записи, поступающих от устройства управления, происходит занесение в память начальных контекстных моделей. В процессе декодирования чтение текущих контекстных моделей организуется путем мультиплексирования выходов регистров в зависимости от значения индекса *ctxIdx*. Компаратор на равенство обеспечивает управление выходным мультиплексором контекстных моделей для обработки ситуации, когда *ctxIdx = ctxIdx_p*.

2.4. Арифметический декодер CABAC

Арифметический декодер (рис. 6, *б*) позволяет декодировать один бин за такт работы декодера.

Разрядность всех операционных устройств – девять разрядов (минимально оговоренная стандартом H.264).

По сигналу инициализации происходит занесение начальных значений переменных *codIRange* и *codIOffset* в соответствующие регистры. В каждом такте работы арифметического декодера значения *codIRange* и *codIOffset* обновляются в соответствии с алгоритмом. Значения контекстных переменных *pStateIdx* и *valMPS* текущей вероятностной модели в каждом такте принимаются во входные регистры.

Таблицы арифметического декодера хранятся в ПЗУ. При реализации алгоритма DecodeDecision используются таблицы значений rangeTabLPS и таблицы для определения новых значений pStateIdx. Табличная реализация алгоритма арифметического декодирования предназначена для исключения операции умножения при подинтервальном разбиении исходного диапазона. Таблицы имеют следующий объем и разрядность: rangeTableLPS memory – 256 8-разрядных слов, state transition LPS memory и state transition MPS memory – 64 6-разрядных слова.

Из алгоритма арифметического декодирования следует, что необходимо осуществлять сравнение значений переменных *codIRange* и *codIOffset*. Аппаратные затраты на реализацию цифровых компараторов для проверки условия неравенства являются достаточно большими. С учетом необходимости реализации последующей операции вычитания *codIOffset* – *codIRange* такую проверку можно реализовать путем анализа знака операции вычитания. Выбор декодированных значений и значений переменных для различных процессов DecodeDecision, DecodeBypass и DecodeTerminate реализуется с помощью соответствующих мультиплексоров.

При арифметическом декодировании возможно возникновение необходимости проведения ренормализации. Для реализации процедуры ренормализации за один такт используются многоразрядные циклические сдвигающие устройства на базе мультиплексоров (Barrel Shifter). Число необходимых сдвигов формируется схемой обнаружения старшей двоичной единицы в текущем значении *codIRange*. Число разрядов, потребленных из входной закодированной последовательности, передается во входной блок (этот выход на структуре не показан).

В соответствии со стандартом H.264 процессы DecodeDecision, DecodeBypass и DecodeTerminate не могут выполняться параллельно; следовательно, для экономии аппаратных ресурсов их можно выполнить на одной аппаратуре.

Помимо декодированного значения бина арифметический декодер формирует новую контекстную модель для текущего контекстного индекса *ctxIdx*.

2.5. Реализация декодера CABAC на FPGA

Для проверки работоспособности предложенных архитектурных решений была выполнена реализация декодера CABAC на FPGA Xilinx.

Для сравнения характеристик арифметического декодера (см. разд 2.3) с известными реализациями [15, 16] было выбрано семейство Viretex 4. Аппаратные затраты и производительность предлагаемого арифметического декодера, а также сравнение их с известными реализациями на базе FPGA приведены в табл. 3.

Таблица 3

Сравнение известных реализаций арифметического декодера на FPGA

Характеристика декодера	Источник		Предлагаемый декодер
	[15]	[16]	
Тип FPGA	90 nm FPGA семейства Virtex 4	Altera Stratix II S60	Virtex 4 XC4VLX15
Затраты аппаратных ресурсов FPGA	302 slice	389 Adaptive Logic Module	273 slice
Максимальная тактовая частота, МГц	100	105	105
Количество тактов для декодирования одного бина	2	1	1

Семейство Altera Stratix II является приблизительным аналогом Xilinx Virtex 4, выпускаемым по такой же 90 nm технологии. При сопоставлении ресурсов следует иметь в виду, что один ALM (Adaptive Logic Module) Stratix II приблизительно соответствует одной секции Virtex 4, поскольку они содержат по два LUT, два триггера, а также некоторые другие аналогичные друг другу ресурсы для реализации арифметических операций.

В реализации [15] декодирование одного бина осуществляется за два такта. В течение первого такта из таблицы содержимое извлекается *Range LPS* и рассчитываются количественные характеристики контекстной модели. В течение второго такта вычисляется декодированное значение бина. Реализация [16] так же, как и предлагаемая, обеспечивает получение одного декодированного бина за такт частоты синхронизации.

Результаты реализации арифметического декодера на базе FPGA показывают, что при сопоставимой с реализацией [16] производительности предлагаемый декодер имеет меньшие затраты аппаратных ресурсов кристалла FPGA, а по сравнению с реализацией [15] имеется выигрыш в два раза по производительности при меньших аппаратных затратах.

Для реализации полного декодера CABAC были выбраны более производительные семейства Virtex 5 и Virtex 6. Характеристики реализации декодера CABAC после процедуры синтеза проекта с помощью ISE 12.2 приведены в табл. 4 и 5.

Блочная память применяется для реализации памяти контекстных таблиц, блоки DSP48E используются в качестве умножителей в процессорах переменных контекстной модели.

Из приведенных результатов следует, что основные аппаратные затраты приходятся на два блока: контекстный процессор и память контекстных моделей.

Основным блоком, вносящим комбинационную задержку, является контекстный процессор. Арифметический декодер по отчету процедуры синтеза может работать для XC6VLX195T на тактовой частоте 206 МГц.

Таблица 4

Аппаратные затраты ресурсов FPGA и производительность декодера CABAC

Характеристика	Virtex 5	Virtex 6
Тип FPGA	XC5VLX155T	XC6VLX195T
Количество триггеров	23446	23073
Количество LUT	15225	12473
Количество блоков памяти	4	4
Количество блоков DSP48E	16	16
Максимальная тактовая частота, МГц	75	94

Таблица 5

Распределение аппаратных затрат по блокам декодера CABAC

Блок декодера	Доля затрат для триггеров, %	Доля затрат для LUT, %
Контекстный процессор	61,0	39,3
Устройство управления	0,7	1,8
Входной блок	0,2	0,5
Арифметический декодер	0,1	1,2
Процессор образца бинаризации	0,2	1,8
Построитель блока коэффициентов	4,7	1,9
Память контекстных таблиц	0,1	0,2
Процессор контекстных переменных	2,0	3,1
Память контекстных моделей	31,0	50,2

Следует отметить, что поддержка возможности кодирования MBAFF значительно увеличивает затраты аппаратных ресурсов и снижает максимальную тактовую частоту декодера. Так, для кристалла XC6VLX195T исключение поддержки кодирования MBAFF снижает аппаратные затраты по триггерам на 31 %, просмотровым таблицам (LUT) на 30 % и повышает максимальную тактовую частоту на 4 % по сравнению с данными табл. 4.

Провести сравнение полученных характеристик с реализациями [10–13] затруднительно, поскольку данные реализации выполнены на ASIC.

Для тестирования декодера было разработано программное обеспечение на основе Reference Software JM [17]. Результаты тестирования считались правильными при полном совпадении результатов декодирования предложенным декодером и тестовым обеспечением на базе JM. Тестирование подтвердило работоспособность декодера с учетом режимов MBAFF и I_PCM.

Заключение

В работе предложена архитектура конвейерного декодера CABAC для мобильных приложений с разрешением до 625SD и поддержкой профилей high profile, high 10 profile, high 4:2:2 profile, включая кодирование MBAFF и использование блоков 8 x 8. Декодер обеспечивает декодирование одного бина за такт и имеет трехступенчатый конвейер.

Реализован прототип декодера на FPGA Xilinx семейств Virtex 5 и Virtex 6 с максимальными тактовыми частотами 75 и 94 МГц соответственно.

Анализ известных источников по реализациям CABAC показывает, что при декодировании одного бина за такт достаточно частоты синхронизации 100 – 160 МГц для обработки разрешений HD (105 МГц для 1920x1088@30 frames/s по сведениям [13], 160 МГц для 1920x1088@30 frames/s по сведениям [18]). Приведенные сведения позволяют сделать вывод о достаточной производительности разработанного декодера для разрешений 720x576 даже при его реализации на FPGA семейства Virtex 6.

Декодер является масштабируемым как по разрешению, так и по поддерживаемым инструментам кодирования стандарта H.264. Основное ограничение на дальнейшее расширение возможностей предложенного декодера CABAC связано с обрабатываемыми типами остаточных блоков. Так, декодер поддерживает следующие типы остаточных блоков: block of luma DC transform coefficient levels, block of luma AC transform coefficient levels, block of 16 luma transform coefficient levels, block of chroma DC transform coefficient levels when ChromaArrayType is equal to 1 or 2, block of chroma AC transform coefficient levels when ChromaArrayType is equal to 1 or 2, block of 64 luma transform coefficient levels.

Список литературы

1. ITU-T. ISO/IEC. ITU-T Rec. H.264 Advanced video coding for generic audiovisual services / ISO/IEC 14496-10 MPEG-4 AVC. ITU. Geneva [Electronic resource]. – Mode of access : http://www.itu.int/rec/T-REC-H.264. – Date of access : 31.05.2013.

2. Overview of the H.264/AVC video coding standard / T. Wiegand [et al.] // IEEE Transaction on Circuits and Systems for Video Technology. – 2003. – Vol.13, no. 7. – P. 560–576.

3. Marpe, D. The H.264/MPEG4 Advanced Video Coding standard and its applications / D. Marpe, T. Wiegand, G.J. Sullivan // IEEE Communications Magazine. – 2006. – Vol. 44, no. 8. – P. 134–143.

4. Kwon, S.-K. Overview of H.264/MPEG-4 part 10 / S.-K. Kwon, A. Tamhankar, K.R. Rao // Journal of Visual Communication and Image Representatin. – 2006. – Vol. 2, no. 17. – P. 186–216.

5. Schäfer, R. The emerging H.264/AVC standard / R. Schäfer, T. Wiegand, H. Schwarz // EBU Technical Review. – 2003, no. 293. – P. 1–12.

6. Richardson, I.E.G. H.264 and MPEG-4 Video Compression / I.E.G. Richardson. – Chichester, UK : Wiley, 2003. – 305 p.

7. Furht, B. Handbook of Mobile Broadcasting: DVB-H, DMB, ISDB-T, and mediaflo / B. Furht, S. Ahson. – NW, USA : CRC Press, 2008. – 726 p.

8. H.264/AVC Decoder Prototype Using a Platform Based SoC Design Methodology / Al. Petrovsky [et al.] // In Proc. of the 10th International Conference Pattern Recognition and Information Processing (PRIP)'2009. – Minsk, 2009. – P. 165–170.

9. Marpe, D. Context-Based Adaptive Binary Arithmetic Codingin the H.264/AVC Video Compression Standard / D. Marpe, H. Schwarz, T. Wiegand // IEEE Transactions on circuits and systems for video technology. – 2003. – Vol. 13, no. 7. – P. 620–636.

10. Chen, J.W. A hardware accelerator for context-based adaptive binary arithmetic decoding in H.264/AVC / J.W. Chen, C.R. Chang, Y.L. Lin // In Proc. IEEE ISCAS. – Kobe, Japan, 2005. – P. 4525–4528.

11. Yu, W. A high performance CABAC decoding architecture / W. Yu, Y. He // IEEE Transaction Consumer Electronics. – 2005. – Vol. 51, no 4. – P. 1352-1359.

12. Kim, C.H. High speed decoding of context-based adaptive binary arithmetic codes using most probable symbol prediction / C.H. Kim, I.C. Park // In Proc. IEEE ISCAS. – Island of Kos, Greece, 2006. – P. 1707–1710.

13. Yang, Y.-C. High-Throughput H.264/AVC High-Profile CABAC Decoder for HDTV Applications / Y.-C. Yang, J.-I. Guo // IEEE Transaction on Circuits and Systems for Video Technology. – 2009. – Vol. 19, no 9. – P. 1395–1399.

14. Recommendation ITU-T H.264 [Electronic resource]. – Mode of access : http://www.itu.int/rec/T-REC-H.264-200903-S/en. – Date of access : 31.05.2013.

15. Hardware Assisted Rate Distortion Optimization with Embedded CABAC Accelerator for the H.264 Advanced Video Codec / J. L. Nunez-Yanez [et al.] // IEEE Transactions on Consumer Electronics. – 2006. – Vol. 52, no. 2. – P. 590–597.

16. Optimizing the critical loop in the H.264/AVC CABAC decoder / H. Eeckhaut [et al.] // In Proc. 2006 IEEE International Conference on Field Programmable Technology (FPT2006). – Bangkok, Thailand, 2006. – P. 113–118.

17. H.264/MPEG-4 AVC Reference Software [Electronic resource]. – Mode of access : http://iphome.hhi.de/suehring/tml. – Date of access : 31.05.2013.

18. Novel Pipeline Design for H.264 CABAC Decoding / J. Zheng [et al.] // Pacific Rim Conference on Multimedia 2007. – Hong Kong, China, 2007. – P. 559–568.

Поступила 25.03.2013

Белорусский государственный университет
информатики и радиоэлектроники,
Минск, П. Бровки, 6
e-mail: {petrovsky, stankevich, palex}@bsuir.by

Al.A. Petrovsky, A.V. Stankevich, A.A. Petrovsky

PIPELINE ARCHITECTURE OF H.264/AVC STANDARD CABAC DECODER FOR MOBILE APPLICATIONS

The paper describes a three-stage pipeline architecture implementation of the CABAC decoder for mobile applications, with image resolution up to 625SD. The decoder architecture is suggested for pipeline calculations with the decoding performance of one bin per clock cycle. The decoder is compatible with profiles high profile, high 10 profile and high 4:2:2 profile and supports regime MBAFF and 8×8 blocks. It is scalable both in the resolution and in the supported decoding tools described in standard H.264. A comparison of our implementation with implementations of a prototype CABAC decoder on FPGA from the company Xilinx is given.