Продолжение. Начало в № 1`2008

Методы и программные продукты для повышения производительности проектов на базе ПЛИС Xilinx

Илья ТАРАСОВ,
к. т. н.
tile@kc.ru

Использование выделенных тактовых ресурсов

Программируемые трассировочные линии внутри ПЛИС далеко не равнозначны. Поскольку практически нереально обеспечить максимальную производительность для всех соединений, наиболее важные сигналы, и прежде всего тактовый, распространяются при помощи выделенных трассировочных ресурсов. Вполне возможен переход с этих линий на ресурсы общего назначения, в том числе и получаемый неявно, путем неаккуратного описания схемы. В этом случае задержка распространения тактового сигнала обычно существенно возрастает, что снижает производительность схемы. Кроме того, момент прихода фронта на вход синхронных устройств оказывается разным для разных частей кристалла, что приводит к нестабильной работе. «Нерекомендуемые» способы формирования тактового сигнала в англоязычной документации называются gated, divided и derived clock. Рассмотрим их подробнее.

Gated clock при дословном переводе означает «тактовый сигнал, пропущенный через логический вентиль» (gate). В цифровой электронике таким образом можно отключать тактовый сигнал — например, если на один из входов элемента 2И подать собственно тактовый сигнал, то другой вход можно использовать как вход разрешения. Однако внутри FPGA такое решение приведет к использованию логической ячейки и автоматическому переходу к более медленным линиям общего назначения. Как правило, gated clock формируется при синтезе конструкции if clk'event and clk = '1 and ce = '1', хотя, как было упомянуто выше, для самых простых случаев формируется вполне корректная конструкция, в которой сигнал ce подается на аналогичный аппаратный вход триггера, а не подмешивается к тактовому сигналу. Несмотря на это, настоятельно рекомендуется не надеяться на проведение средствами синтеза серьезной оптимизации схемы и явно задавать тактовый сигнал в виде:

```
if clk'event and clk = '1' then
  if ce = '1' then
```

Следующий способ формирования — divided clock (деленная тактовая частота) также относится к широко известным цифровым узлам. Делитель частоты легко может быть построен на базе D-триггера, если проинвертированный выход подать на вход данных. Такая схема показана на рис. 8, где использован также специальный буфер BUFG, предназначенный для формирования крутых фронтов тактового сигнала. На первый взгляд, проблемы решены и полученное решение достаточно эффективно. Однако сигнал Clk2 будет отставать по фазе от исходного сигнала Clk1, что может в конечном итоге привести к негативным эффектам. Корректная схема также показана на рис. 8, где сигнал с выхода триггера служит не тактовым сигналом, а сигналом разрешения счета. Поскольку этот сигнал появляется каждый второй такт, очевидно, что полученная схема также представляет собой делитель частоты на 2, однако при этом оба триггера используют одну и ту же глобальную тактовую линию с минимальными задержками распространения и хорошим фронтом.

При использовании счетчиков для организации делителей частоты может образоваться «производный» (точная калька с английского — «полученный», derived) тактовый сигнал. Такой сигнал формируется комбинаторной логикой при достижении счетчиком определенного состояния и формально обеспечивает деление входной частоты на нужный коэффициент. Однако с учетом возможных различий трассировки может возникнуть ситуация, показанная на рис. 9. Рассчитывая, что комбинация 1111 будет возникать один раз за каждые 16 тактов, разработчик предполагает, что полученная схема представляет собой делитель на 16. Однако в силу того, что старший разряд распространяется до логического вентиля быстрее, при переходе от 0111 к 1000 может возникнуть ситуация, когда на входе вентиля будут

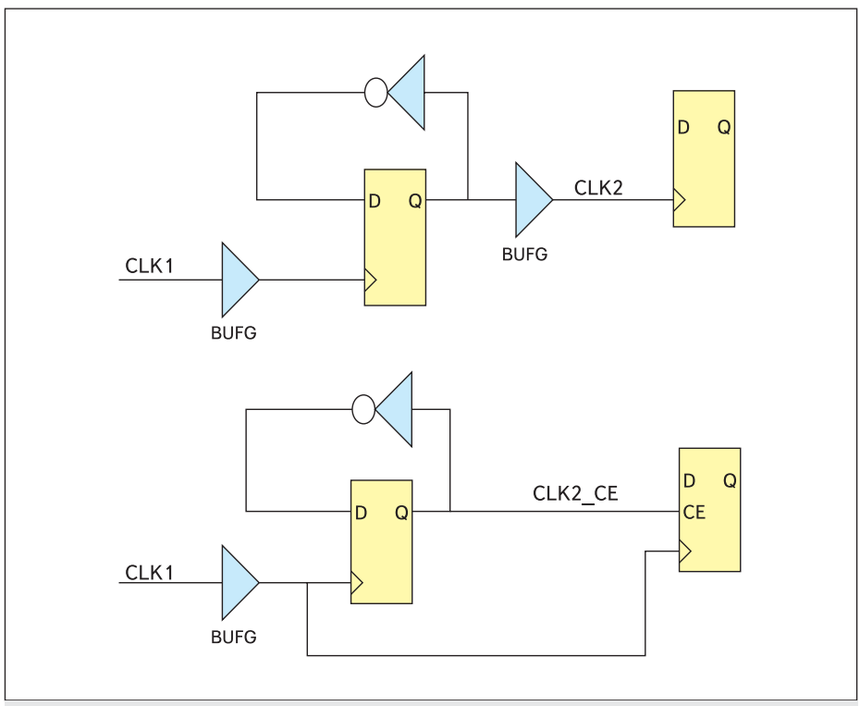

Рис. 8. Некорректное и корректное построение делителя частоты для FPGA

присутствовать старые значения младших разрядов и новое значение старшего разряда, т. е. комбинация 1111. Таким образом, 1111 будет образовываться два раза за 16 тактов, заставляя выходной триггер срабатывать в 2 раза чаще, чем ожидалось. Такой эффект является оборотной стороной увеличения производительности логических ячеек, которые теперь становятся чувствительными даже к коротким импульсам, ранее срабатывания не вызывавшим. К прочим негативным последствиям такого решения можно отнести:

- нестабильность эффекта как в пределах одной версии проекта (длительность «иголки» может быть на грани срабатывания триггера и коэффициент деления входной частоты будет непостоянным), так и при повторных трансляциях (трассировочные линии могут быть проложены по-другому, что изменит соотношение задержек распространения сигналов);
- трудность отладки и идентификации, усугубляемая тем, что введение в проект отладочных ресурсов и «тестовых точек» само по себе изменяет структуру схемы и трассировку проводников; в результате схема, корректно работающая при наличии отладочных линий, может оказаться неработоспособной при их удалении;
- ухудшение времени нарастания и спада тактового сигнала, сформированного комбинаторной логикой;
- сдвиг фазы тактового сигнала по отношению к глобальному тактовому сигналу.

На рис. 9 показана также корректная схема формирования сигнала с помощью счетчика, в которой комбинаторная логика используется для управления входом разрешения счета, а тактовый сигнал в обоих счетчиках подается от глобальной тактовой сети.

Таким образом, общим требованием к реализации синхронных схем является широкое использование сигналов «разрешение счета», которые в триггерах логических ячеек реализованы аппаратно. В то же время тактовый сигнал должен постоянно подаваться на все синхронные ресурсы ПЛИС — триггеры, блочную память, аппаратные умножители.

В FPGA Xilinx имеются аппаратные модули формирования тактового сигнала — DCM (Digital Clock Manager). Ввиду важности использования именно глобальных тактовых сетей в САПР ISE введен мастер, облегчающий подключение к проекту и настройку этого модуля. Внешний вид мастера показан на рис. 10. Как можно видеть, DCM обладает достаточно широкими возможностями, включая:

- подстройку фазы выходного сигнала с помощью Delay-Locked Loop (DLL);
- формирование квадратурных сигналов (сдвинутых на 0, 90, 180 и 270 градусов);
- деление частоты;
- удвоение частоты;
- преобразование частоты (*A/B, где A и B — произвольные целые числа до 32);

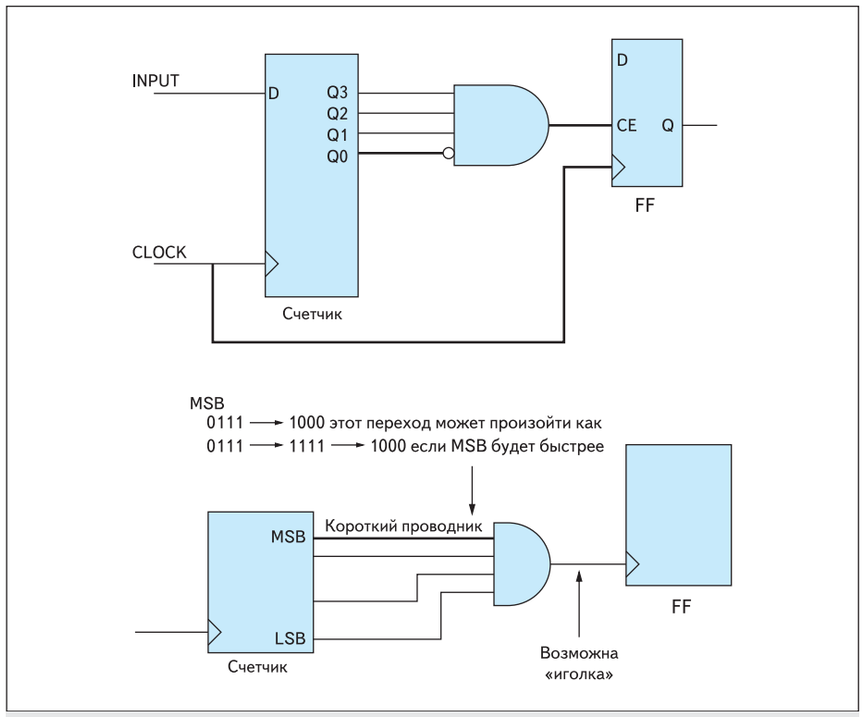

Рис. 9. Некорректное и корректное построение делителя частоты с использованием счетчика

- коррекцию коэффициента заполнения сигнала (Duty Cycle Correction).

Блоки DCM в семействе Virtex также допускают динамическую реконфигурацию и точное управление фазой выходного сигнала. Представленный мастер позволяет быстро и наглядно задать желаемые характеристики тактовой частоты, что существенно облегчает правильное включение DCM в проекты. Можно еще раз отметить, что использование DCM настоятельно рекомендуется не только для получения высоких рабочих частот, но и вообще для достижения стабильных характеристик устройства.

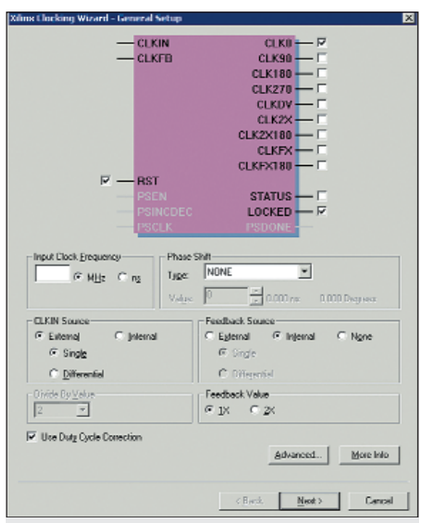

Рис. 10. Мастер настройки модуля DCM

В FPGA существует также большое количество локальных тактовых подсетей, которые тоже обладают хорошими характеристиками в плане производительности. Однако главным мероприятием, которое следует выполнить при разработке проекта, является все же формирование основного тактового сигнала с помощью глобальной тактовой сети.

Триггеры вместо защелок

Xilinx рекомендует использовать в цифровых узлах триггеры (flip-flop) вместо защелок (latch). Защелка представляет собой устройство, которое постоянно пропускает на выход входной сигнал при наличии сигнала разрешения. В момент перехода сигнала разрешения в неактивное состояние на выходе запоминается последнее значение входного сигнала. Защелка может быть создана из триггера и мультиплексора (который будет пропускать на выход либо копию входного сигнала, либо значение на выходе триггера), и в ПЛИС она имеет несколько худшие характеристики, чем собственно триггер, который реализован аппаратно. Xilinx рекомендует обращать внимание на ликвидацию неочевидных защелок, которые формируются средствами синтеза из некоторых фрагментов кода на HDL, содержащего неоднозначное поведение синхронного узла. Например, в приведенном фрагменте не во всех ветках условного оператора определено значение выхода. Следовательно, средства синтеза сформируют защелку, которая будет хранить на выходе последнее записанное туда значе-

ние. В ряде случаев несколько странно выглядящая конструкция вида **reg <= reg**, находящаяся внутри синхронного процесса, в действительности заполняет «пробелы» в возможных вариантах условий, определяя, что если ни одно из явно указанных условий не выполняется, то в регистр следует переписать его старое значение. Также в примере, приведенном ниже, предположим, что строки, выделенные жирным шрифтом, отсутствуют. В этом случае для веток оператора case RST и JUMP задано поведение только сигнала SIG1, а для LOAD и others — только SIG2. Соответственно, для обоих сигналов были бы сгенерированы защелки, запоминающие для всех неуказанных вариантов их поведения последнее состояние:

```
process ( OP_STATE, DATA_ADDR, DATA_STORED…)
begin
    SIG1 <= «0000» ;
    SIG2 <= «00000000» ;
    case (OP_STATE) is
      when RST => SIG1 <= «0000» ;
      when JUMP => SIG1 <= DATA_ADDR ;
      when LOAD => SIG2 <= DATA_STORED ;
      when others => SIG2 <= NEW_DATA ;
    end case ;
end process;
```

Case вместо if-then

Целый ряд логических условий может быть эквивалентно задан с помощью конструкции case либо с помощью вложенных операторов if-then. С точки зрения классического программирования, ситуация не вполне однозначная, и иногда для программиста вложенные условные операторы представляются более предпочтительными. Однако с учетом особенностей архитектуры FPGA оказывается, что более рациональным является использование конструкции case, поскольку в этом случае средства синтеза генерируют мультиплексор, имеющий эффективную реализацию в современных FPGA. В то же время вложенные if-then генерируют цепочку последовательных выражений комбинаторной логики, где для активизации последующей стадии необходимо получить информацию о невыполнении предыдущего условия. При этом задержка распространения сигнала возрастает.

Необходимым условием для использования конструкции case является наличие переменной-селектора. Если условие для выбора одной из альтернатив более сложное, включающее в себя операции сравнения или существенно различающиеся логические выражения в отдельных ветвлениях, использование if-then является единственно доступным вариантом.

В то же время можно упомянуть практический прием, который позволяет свести проверку различных сигналов к переменной-селектору. Например, если требуется определение приоритета действий по последовательным проверкам переменных a, b, c и d, то можно создать для них копию в виде сигнала-селектора **sel(3 downto 0)**;

Далее для этого сигнала выполняется асинхронное присваивание отдельных разрядов:

```
sel(3) <= a;
sel(2) <= b;
sel(1) <= c;
sel(0) <= d;
```

Теперь вместо вложенных условных операторов вида

```
if a = '1' then
   elsif b = '1' then
      elsif c = '1' then
         elsif d = '1' then
            else
```

можно записать:

```
case sel(3 downto 0) is
   when 8 to 15 => -- выполняется, когда a = '1', при любых комбинациях остальных сигналов
   when 4 to 7 => -- выполняется, когда a <> '1' и b = '1'
   when 2 to 3 => -- a = '0', b = '0', c = '1'
   when 1 => -- a = '0', b = '0', c = '0', d = '1'
   when others => -- все сигналы равны '0'
end case;
```

Группировка комбинаторных выражений

Известный из математики принцип «от перемены мест слагаемых сумма не изменяется» имеет интересные тонкости при реализации комбинаторной логики в FPGA. Каким бы странным ни могло показаться такое поведение ПЛИС, выражения A + B + C + D и (A + B) + (C + D) могут дать логический узел с (безусловно) одинаковым поведением, но разной производительностью. Это происходит потому, что синтез выражений происходит обычным порядком, с учетом приоритета операторов (на который влияют и скобки). Следовательно, первый вариант записи сформирует «ступенчатую» конструкцию, в которой первый сумматор выполнит сложение A + B, затем второй добавит к этой сумме C, а третий — D (рис. 11). В итоге сигналы A и B пройдут через три сумматора, образуя тем самым самую длинную цепь. Если эти сигналы по каким-то причинам формируются позже, чем C и D, в итоге получается неэффективное решение, содержащее излишние задержки. В то же время второй вариант выполнит сложения A + B и C + D параллельно, а результаты будут поданы на входы третьего сумматора. При этом каждый из входных сигналов пройдет только через два сумматора. Приведенные рассуждения являются достаточно схематичными и нуждаются в массе оговорок и индивидуальном рассмотрении каждого случая. Например, если четыре входных сигнала будут поданы на входы одного и того же LUT, то ни о каких путях прохождения сигналов по кристаллу не может идти речи — все они будут обработаны в пределах одной и той же таблицы истинности, вне зависимости от реализуемой функции. Однако смысл рекомендации состоит в том, чтобы обратить внимание разработчика на неочевидную возможность улучшения характеристик комбинаторной логики.

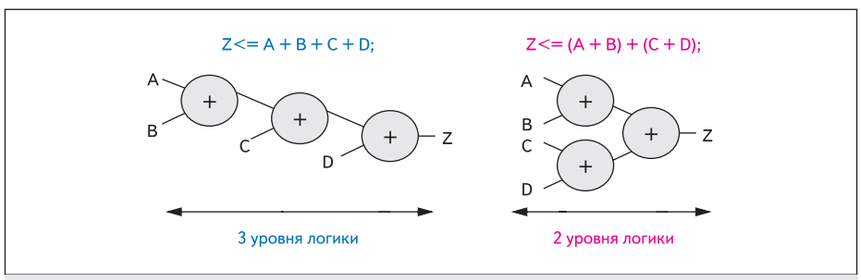

Рис. 11. Синтез комбинаторной логики при бесскобочной записи выражения и при явном указании приоритета операций

Для high-end ПЛИС Virtex-4 и Virtex-5 (а также DSP-ориентированных Spartan-3A DSP) необходимо дать уточнение к этому, и так несколько запутывающему читателя, материалу. Дело в том, что блоки DSP48, содержащие встроенные 48-битные аккумуляторы, могут функционировать в качестве сумматоров, каскадируясь с помощью выделенных линий ускоренного переноса. Такое каскадирование гораздо эффективнее, чем «лестничная» передача частичных сумм с уровня на уровень логики, поскольку в данном случае сложение выполняется выделенным аппаратным блоком. В этом случае для получения высокой разрядности следует «вытянуть в линию» все сумматоры.

Синхронизация при переходе между тактовыми доменами

В ряде случаев различные части ПЛИС должны тактироваться разными сигналами по условиям функционирования устройства. Для надежной передачи данных между отдельными частями кристалла необходимо решить проблему защиты от метастабильности тех регистров, данные в которые записываются одним тактовым сигналом, а читаются частями схемы, тактируемыми другим сигналом. При этом может возникнуть ситуация, когда чтение будет происходить непосредственно в процессе смены состояния отдельных триггеров, что и является наиболее опасным вариантом. Кроме рассмотренных выше схем синхронизации, можно также рассмотреть возможность организации FIFO

(очереди) как буфера данных между отдельными тактовыми подсетями. При использовании FIFO часть ПЛИС, производящая чтение, может дожидаться накопления определенного объема данных, с тем чтобы появилась гарантия, что читаемая в данный момент ячейка уже была записана ранее и не будет изменяться. Построению FIFO способствуют и архитектурные особенности FPGA Xilinx:

- наличие распределенной памяти внутри отдельных секций (распределенная память 16×1 или сдвиговый регистр на 16 разрядов — штатные режимы работы LUT);
- дополнительные аппаратные ресурсы в блочной памяти, способствующие построению эффективных очередей — каждый блок имеет аппаратные счетчики, а также флаги «FIFO пуст», «FIFO полон», «FIFO почти пуст», «FIFO почти полон» с порогами срабатывания, настраиваемыми индивидуально для каждой очереди (в двух последних случаях).

Использование конечных автоматов

Конечный автомат (КА, также FSM — Finite State Machine) представляет собой синхронное устройство, способное находиться в ограниченном числе устойчивых состояний. Каждое состояние однозначно характеризуется набором переменных состояния, в качестве которых удобно использовать триггеры. Переход от одного состояния к другому происходит по тактовому сигналу в зависимости от предыдущего состояния и значений входных сигналов. Пример диаграммы КА показан на рис. 12. На ней окружностями показаны состояния с соответствующими номерами, а дугами — условия перехода от одного состояния к другому. В силу единообразия работы в различных состояниях конечный автомат допускает простую и удобную реализацию в FPGA, обеспечивая там достаточно высокую производительность и простоту трансляции. На протяжении развития последних версий САПР ПЛИС алгоритмы трансляции КА получали все большее развитие, и в настоящий момент можно настоятельно рекомендовать решение в подобном стиле как высокопроизводительное и надежное. Разумеется, параметры производительности проекта на базе ПЛИС зависят от множества факторов, в том числе и от таких предельных параметров, как суммарная задержка в «обязательных» компонентах преобразования сигнала. Иными словами, конечные автоматы не могут обеспечить превышение теоретического предела производительности, но простота и единообразие порядка проектирования помогает приблизиться к этому пределу.

Для эффективной реализации КА следует придерживаться ряда рекомендаций. Прежде всего, необходимо прокомментировать порядок нумерации состояний автомата. Наиболее наглядной является обычная последовательная нумерация, при которой каждому состоянию присваивается номер, представляемый в ПЛИС двоичным числом, хранящимся в регистрах. Однако двоичное кодирование — не самое эффективное по производительности, поскольку увеличение номера состояния на единицу сопряжено с распространением бита переноса по всему счетчику. Например, при переходе от 01111111 к 10000000 бит переноса будет распространен от самого младшего к самому старшему разряду. Большей частоты можно добиться, используя one-hot кодирование, которое подразумевает выделение одного бита для кодирования каждого состояния. При этом для перехода к новому состоянию достаточно сбросить один бит (соответствующий предыдущему состоянию) и установить один бит для нового состояния. Сходными свойствами обладают коды Грея (для перехода к следующему коду достаточно изменить один бит), счетчики на базе сдвиговых регистров (LFSR — Linear Feedback Shift Register) и коды Джонсона. Однако, сравнивая затраты триггеров на представление необходимого числа состояний, легко заметить, что двоичное кодирование наиболее компактно — 8 триггеров достаточно для представления $2^8 = 256$ комбинаций, тогда как one-hot требует одного бита для каждого состояния, и 8 триггеров способны представить только 8 состояний (а для 256 потребуется 256-разрядное one-hot кодирование!). Таким образом, в зависимости от числа состояний КА оптимальным оказывается тот или иной метод кодирования состояний, что и может выбрать ISE в режиме Auto. Однако для этого необходимо использовать не явное назначение номеров состояниям (что ликвидирует свободу средств синтеза), а использовать символические, перечислимые типы (например, RESET, LOAD, STORE, STOP) для идентификации состояний, чтобы средства синтеза могли назначить им действительные двоичные представления наилучшим образом.

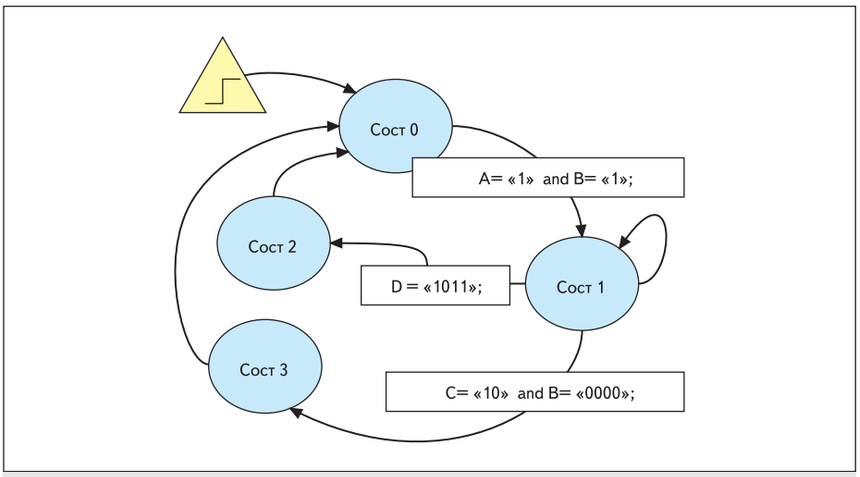

Рис. 12. Пример диаграммы состояний конечного автомата

Конечный автомат достаточно просто реализуется с помощью оператора case, в отдельных ветках которого описаны условия переходов к другим состояниям:

```
process(clk)
begin
    if rising_edge(clk) then
    case state is
        when STATE1 => if <условие> then state <= STATE2; end if;
        when …
        when others => null;
    end case;
    end if;
end process;
```

В показанном описании необходимо тщательно фиксировать реакцию автомата на входные воздействия в каждом состоянии и проверять наличие недостижимых состояний.

В настройках синтеза имеется также пункт Safe Implementation, который при выборе Yes генерирует схему, обеспечивающую реакцию по умолчанию для каждого из состояний, которое теоретически может быть получено на имеющемся наборе регистров, но не используется среди перечисленных в КА (например, для автомата на 129 состояний при двоичном кодировании требуется 8-разрядный регистр, могущий представить 256 возможных комбинаций; при этом 256–129 = 127 состояний будут недостижимыми, поскольку ни одна из последовательностей действий КА не приводит к загрузке этих номеров). Однако в силу каких-либо причин, например аппаратного сбоя или прямой загрузки состояния с неверным номером через внешние интерфейсы (если такое предусмотрено), этот недостижимый номер все же может появиться в регистрах и поведение КА будет неопределенным. Для обеспечения реакции по умолчанию и используется параметр Safe Implementation → Yes, хотя для достижения максимальной производительности его следует отключить.

Кроме того, в рамках общего подхода к использованию синхронных схем, инициализацию КА рекомендуется также выполнять синхронно, т. е. проверять состояние входа сброса по фронту тактового сигнала (а не асинхронно, в любой момент). Ниже показаны листинги VHDL, реализующие такие схемы. Можно видеть, что приоритет сброса в первом случае определяется его первым упоминанием среди проверок сигналов, во втором же он оказывает влияние на схему только в момент прихода фронта тактового сигнала:

```
-- асинхронный сброс
process(clk, reset)
begin
   if reset = '1' then <сброс>
      elsif rising_edge(clk) then
         …
end process;

-- синхронный сброс
process(clk)
begin
   if rising_edge(clk) then
      if reset = '1' then <сброс>
         …
end process;
```

Использование IP-ядер

Входящий в САПР ISE генератор стандартных цифровых узлов Core Generator представляет собой библиотеку параметризуемых модулей с заранее настроенными параметрами относительного размещения ресурсов на кристалле. Это означает, что использование такого модуля дает в распоряжение разработчика не просто некий шаблон исходного текста, но и указания САПР к оптимальному размещению отдельных логических ячеек для достижения наилучших результатов. Термин «параметризуемый» означает, что разработчик имеет возможность настраивать параметры таких ядер (в соответствии с теми пунктами, которые предоставляет то или иное ядро). Как правило, настройке доступна разрядность обрабатываемых данных, наличие сигналов разрешения работы, сброса, количество тактов конвейера и т. п. Core Generator обычно предлагает создание RPM — Relationally Placed Macro, который представляет собой «оттиск» цифрового узла, жестко помещаемый на кристалл относительно некой «точки привязки». Таким образом, САПР получает готовое решение по размещению ячеек, и нюансы настроек конкретного проекта уже не могут ухудшить параметры такого модуля.

В версии ISE 9.2 (при наличии соответствующих обновлений библиотеки IP-ядер) доступно приложение Memory Interface Generator (MIG), которое производит автоматическую генерацию контроллеров современных модулей памяти: SDR и DDR SDRAM, RLDRAM. Весьма важным является тот факт, что для надежной реализации такого контроллера требуется его тщательное расположение относительно внешних выводов, а также настройка модуля DCM и задержек в блоках ввода/вывода. Кроме того, MIG использует тонкую подстройку сдвига фазы тактового сигнала, что позволяет рассчитывать на подключение современных высокопроизводительных микросхем памяти большого объема.

Организационные вопросы — стиль оформления кода, комментирование, документирование

Организационные вопросы ведения работ хотя и не имеют непосредственного влияния на технические параметры, также важны для эффективной разработки, модификации и сопровождения проекта. При использовании модульного проектирования адекватное и полное описание характеристик отдельных модулей является крайне важным для успешной интеграции этих модулей в проект. Кроме того, подробное комментирование кода, использование стандартных заголовков и единообразных правил комментирования, четкое структурирование кода позволяют вести коллективную разработку, в том числе разделять задачи между отдельными участниками проекта в соответствии с их квалификацией и имеющимися предпочтениями. Рекомендуется ведение общего репозитория рабочей группы.

В языке VHDL имеется возможность создания пакетов (package), в которых обычно размещают функции, процедуры, константы и прочие объекты. Такой пакет может быть использован в качестве основы для рабочей группы, что позволит использовать проверенные алгоритмы преобразования и компоненты, а также исключить разночтения при задании глобальных констант.

Настройка программы синтеза

САПР обладает возможностью регулировать в некоторых пределах характеристики получаемого проекта, применяя те или иные алгоритмы синтеза. Если поставленная задача характеризуется как «получение максимальной тактовой частоты», необходимо выполнить соответствующие настройки.

На вкладке Synthesis options внимание необходимо уделить следующим пунктам.

Optimization Goal — имеет значения Speed и Area (оптимизация по тактовой частоте и занимаемой площади соответственно).

Optimization Effort — данный параметр влияет на интенсивность расчетов САПР при попытках получить оптимальное решение. Данный параметр имеет смысл устанавливать в High, поскольку прочие значения при уменьшенном времени трансляции проекта могут ухудшить получаемые характеристики, и их имеет смысл устанавливать только для получения промежуточных, отладочно-экспериментальных конфигураций.

Keep Hierarchy — включает или отключает сохранение иерархии проекта в том виде, в котором она установлена разработчиком, путем разбиения на субмодули. Необходимо обращать внимания на данный параметр, поскольку его отключение может свести на нет усилия по выделению и отдельной оптимизации модулей проекта.

На вкладке HDL Options: пункты FSM Emcoding Algorithm и Safe Implementation относятся к настройке способа реализации декодеров состояния конечных автоматов. Алгоритмы автоматического выбора производят анализ числа состояний КА и выбирают оптимальный способ реализации в соответствии со структурой используемой ПЛИС. При необходимости можно принудительно задать желаемый способ реализации декодеров. Аналогичные соображения применимы и к остальным пунктам данной вкладки — САПР выбирает специализированные аппаратные ресурсы автоматически, если они применимы к решению задачи, описанной разработчиком в HDL.

На вкладке Xilinx Specific options расположены специфичные для архитектуры ПЛИС Xilinx настройки. Пунктом, который может оказать явное влияние на производительность, является Register Duplication (дублирование регистров). Данная настройка разрешает алгоритмам синтеза создавать копии сигналов в различных частях кристалла, обеспечивая таким образом тиражирование критичных цепей с целью минимизации общей задержки распространения важного для проекта сигнала по всему кристаллу. В данном случае повышение производительности происходит за счет использования дополнительных ресурсов, однако затраты весьма несущественны по сравнению с получаемой выгодой.

При дублировании триггеров особенно важным является устранение метастабильности. Если рассматривать ситуацию, когда два триггера дублируют захват одного и того же сигнала и один из них (или оба) попадают в метастабильное состояние, легко представить, что одна часть кристалла работает с сигналом, представляющим собой логический ноль, а для другой части кристалла тот же сигнал воспринимается как логическая единица. Нетрудно понять, что возникающие проблемы могут иметь широчайший спектр проявлений. Достаточно привести пример, когда сигнал представляет собой направление передачи для двунаправленного буфера и передается на внешний вывод с помощью дублирующего триггера для управления двунаправленной шины внешней микросхемы. В этом случае может возникнуть ситуация, когда из-за «разночтений» обе двунаправленные шины окажутся активизированными на вывод, что приведет как минимум к электрическому конфликту, а в худшем случае — к повреждению ПЛИС.

В целом настройки программы синтеза не являются абсолютно однозначными, и максимальная производительность в ряде случаев может быть достигнута при комбиниро-

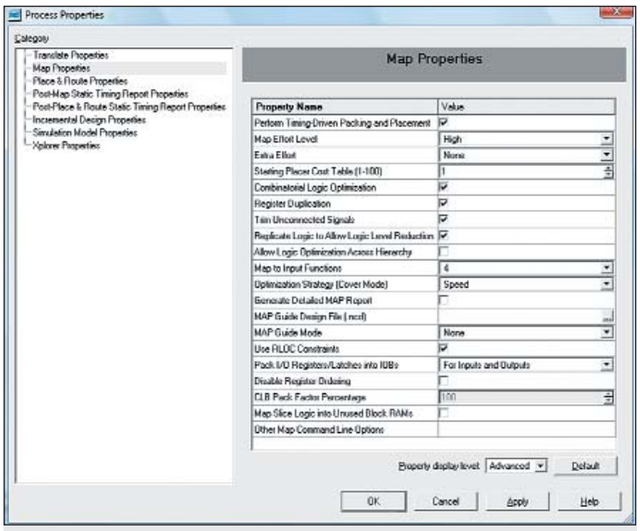

Рис. 13. Диалоговое окно настроек процесса Map

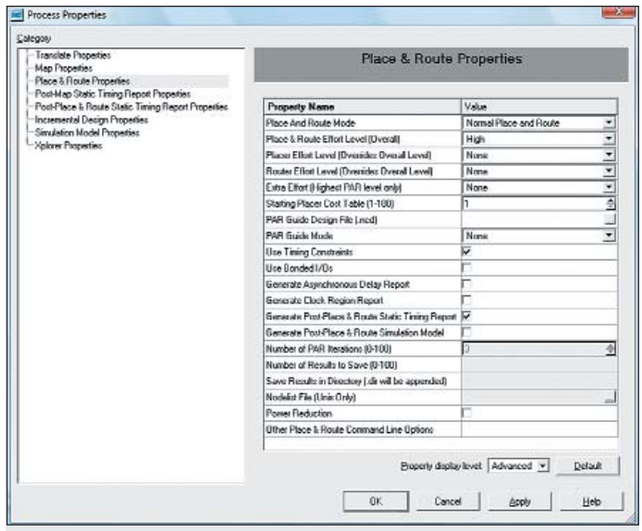

Рис. 14. Настройки процесса Place & Route

вании перечисленных приемов и рассмотрении различных вариантов. Можно отметить, что разработка программного обеспечения для синтеза списка связей проекта по HDL-описанию представляет собой достаточно обширный рынок, где активно работают различные фирмы. В отличие от последующих этапов, которые доступны исключительно производителю микросхем, синтез может быть выполнен и с помощью программ сторонних разработчиков.

Настройка размещения и трассировки

После создания списка соединений САПР выполняет следующие действия:

- объединение отдельных модулей проекта (Translate);
- связывание абстрактных компонентов с определенными физическими ресурсами кристалла (Map);
- размещение компонентов в логические ячейки (Place);
- соединение компонентов внутренними трассировочными ресурсами (Route).

Основные проблемы при достижении высокой тактовой частоты решают два последних этапа (в САПР они объединены в один процесс Place&Route). Тем не менее на этапе Map также есть возможности эффективного управления алгоритмами САПР.

Рассмотрим наиболее важные настройки процесса Map. Пункт Perform Timing-Driven Packing and Placement может оказать влияние на производительность проекта на уровне 3–5% (типичное значение, крайне зависящее от конкретных условий), однако это производится ценой многократного увеличения длительности этапа Map. Для FPGA большого объема рост времени в абсолютном выражении может составлять часы, хотя можно повторить, что параметры сугубо индивидуальны для каждого конкретного случая. Установка этого флажка включает процесс оценки времени распространения сигнала при отображении каждого компонента на физические ресурсы кристалла. Разработчик должен иметь в виду, что существует такой резерв повышения рабочей частоты, хотя для предварительных стадий работы увеличение времени трансляции может оказаться весьма большим, а получаемый результат — не таким уж необходимым, поскольку в этот момент разработчиком скорее всего будут решаться задачи отработки архитектуры проекта и основных алгоритмов, а не повышения частоты до предела. С другой стороны, для проектов объемом до миллиона вентилей увеличение времени работы несущественно, поэтому ПЛИС малого и среднего объема вполне могут транслироваться с установленным флажком Timing-Driven Map.

Параметр Map Effort Level аналогичен по назначению параметру Optimization Effort на этапе синтеза и регулирует время, выделяемое САПР на данный этап работы. Совершенно аналогично финальная трансляция должна быть проведена на уровне High, а понижение уровня может быть вызвано необходимостью ускорить процесс работы САПР на этапах отработки общей идеи.

Пункт Combinatorial Logic Optimization, как следует из его перевода, разрешает оптимизацию комбинаторной логики. Рекомендуется его установка. Аналогично рекомендуется установка пункта Trim Unconnected Signals, удаляющего из проекта неподключенные сигналы.

Параметры Register Duplication и Replicate Logic to Allow Logic Level Reduction дают схожий эффект — небольшое повышение производительности путем создания точных копий сигналов в разных частях кристалла. Параметры несколько пересекаются с аналогичными настройками этапа синтеза, однако здесь следует помнить, что синтез может выполняться не только с помощью программного обеспечения, разработанного Xilinx, и его настройки могут оказаться иными. Аналогично, пункт Allow Logic Optimization Across Hierarchy по смыслу пересекается с настройкой Keep Hierarchy. На рис. 13 видно, что флажок с этого пункта снят, поскольку предполагается, что разбиение проекта на иерархические блоки выполнено разработчиком сознательно, с учетом функционального назначения каждого блока, и достаточно грамотно, ввиду чего «сквозная» оптимизация скорее могла бы дать ухудшение параметров.

Очевидной настройкой является Strategy Mode, установленная в состояние Speed. Также очевиден пункт Pack I/O Registers/Latches into IOBs, разрешенный для входов и выходов, так как данная настройка явно разрешает использование аппаратных регистров в блоках ввода/вывода. В таком режиме задержка распространения между собственно выводами ПЛИС и регистрами является минимальной, и появляется возможность записи входных сигналов в триггер за минимальное время (а также распространение сигнала до выхода за минимальное время). Как было упомянуто выше, важным методом повышения производительности является использование специализированных аппаратных ресурсов ПЛИС, и регистры блоков ввода/вывода относятся именно к таким ресурсам.

Интерес представляет также пункт Map Slice Logic into Unused Block RAMs, разрешающий использование блочной памяти в качестве мощных логических генераторов. Такие генераторы обладают неплохими характеристиками, с учетом того, что при организации 1024×18 каждый блок легко способен реализовать функцию от 10 входов (по сравнению с 4 входами LUT), однако блочная память представляет собой доста-

точно дефицитный ресурс, и использовать такой параметр необходимо с осторожностью, планируя распределение блоков памяти по задачам и выполняемым функциям.

На рис. 14 показаны настройки процесса Place & Route.

В данном случае мы также можем наблюдать возможность регулирования времени, выделяемого САПР на работу над этим этапом (Effort Level). Можно отдельно задать интенсивность расчетов для этапов Place (размещение) и Route (трассировка). Для получения высокой производительности может оказаться необходимым как повторить трассировку при неизменном размещении (за это отвечает режим Reentrant Route, выбираемый в пункте Place And Route Mode), так и полностью повторить размещение и трассировку с несколько измененными параметрами (Multi Pass Place And Route). В этом случае можно задать количество итераций (Number of PAR Iterations) и количество вариантов, которые будут сохранены (Number of Results to Save). Поскольку выбираются варианты с наилучшими характеристиками получаемой схемы, один из этих вариантов может служить основой для последующей ручной трассировки с помощью FPGA Editor.

Как несложно догадаться, речь идет уже о финальных стадиях разработки конфигурации ПЛИС, когда проделаны все мероприятия «стратегического характера», т. е. проект полностью синхронный, по возможности использует конечные автоматы, в текстах нет слишком сложных комбинаторных выражений, которые могли бы сформировать многоуровневую логику, использованы правильные настройки синтеза. Далее процесс может идти следующим образом.

1. Попытка трассировки проекта с настройками PAR по умолчанию.
2. Если результат несущественно хуже желаемого, можно попробовать изменить Effort Level, дав возможность САПР исправить ситуацию автоматически.
3. Если изменение Effort Level не помогает или полученные по умолчанию характеристики существенно хуже желаемых (что практически не позволяет надеяться на автоматическое доведение до нужного уровня средствами САПР), необходимо включение режима Multi Pass Place and Route, в котором размещение и трассировка выполняются многократно для различных начальных вариантов.

Можно некоторым образом повлиять на процесс выбора порядка размещения компонентов с помощью параметра Starting Placer Cost Table. Он изменяется в пределах 1–100, однако прямой количественной связи между этим параметром и каким-либо «индексом» или «производительностью» на самом деле не существует. В действительности наилучший результат может быть получен для совершенно непредсказуемого значения, и главным фактором здесь является количество итераций PAR, которые готов проверить разработчик. В итоговом отчете в списке лучших могут оказаться, к примеру, 5, 17, 23 и 46 проходы, т. е. какой-либо монотонности или прямой зависимости в виде «больше номер — больше времени на итерацию — лучше результаты», как правило, здесь не наблюдается. Следует иметь в виду, что 20–30 итераций PAR для FPGA класса Virtex-4/5 среднего объема обычно занимают много времени (до нескольких суток).

Если результаты размещения признаны разработчиком удачными и проблема, на его взгляд, кроется в неоптимальном выборе трассировочных ресурсов для реализации нужных соединений, можно ограничиться режимом Reentrant Route, который повторяет только процесс трассировки, оставляя неизменным размещение ячеек.

Перечисленные методы соответствовали ситуации, когда разработчик влияет на результаты трассировки опосредованно, путем выдерживания определенного стиля кодирования и настройки САПР с помощью предусмотренных для этого диалоговых окон. Однако можно добиться и более глубокого контроля над проектом при использовании низкоуровневых средств проектирования на уровне топологии кристалла, которые и будут рассмотрены в заключительной части статьи.

Окончание следует