Использование проектных ограничений формата xdc в САПР Vivado для работы с ПЛИС Xilinx

Илья ТАРАСОВ,
к. т. н.
ilya_e_tarasov@mail.ru

Одним из нововведений САПР Vivado, предназначенной для разработки проектов на базе ПЛИС Xilinx серии 7 и последующих поколений, является переход к формату xdc (Xilinx Design Constraints) для описания проектных ограничений. Этот формат полностью заменил использовавшийся ранее ucf (User Constraints File), следовательно, для полноценной работы с Vivado необходимо освоить xdc. Возможности этого формата, основанного на языке tcl и близкого к промышленному стандарту sdc, обширны, поэтому применение его в практике проектирования может помочь разработчикам получать более качественные и воспроизводимые результаты.

Введение

Переход к формату xdc обусловлен возрастающей сложностью проектов и необходимостью более полного контроля процессов синтеза, реализации, установки проектных ограничений, формирования отчетов и пр. При работе с ПЛИС большого объема часто бывает удобно выполнять все стадии проектирования путем запуска единственного сценария. При этом разработчику более не требуется повторять все действия с интерфейсом САПР в правильном порядке, поскольку порядок действий определяется сценарием. Также за счет системы проектных ограничений результаты работы становятся более предсказуемыми. Можно представить процесс сборки проекта для ПЛИС серии 7 с сотнями тысяч и миллионами ячеек, отдельные стадии которого могут занимать несколько часов. Если разработчик будет вынужден следить за надлежащим выполнением каждой стадии, проверять результаты и вручную запускать следующий этап, производительность его работы заметно упадет. Кроме того, последовательный запуск синтеза и реализации (implement) с разными наборами настроек (что может потребоваться для проектов со сложной структурой и жесткими временными ограничениями) неудобно выполнять вручную. Именно для автоматизации подобных действий используется встроенный интерпретатор языка tcl, к синтаксису которого и приведен формат проектных ограничений xdc.

В Vivado больше не поддерживаются проектные ограничения в формате ucf, который использовался в САПР ISE и PlanAhead на протяжении последнего десятилетия. Поэтому переход к xdc является не просто рекомендацией, а вопросом практической необходимости.

Организация САПР Vivado и интеграция языка tcl для управления проектом

Формат xdc является расширением языка tcl (Tool Command Language), который широко используется в современных САПР. Отдаленным аналогом могут служить пакетные файлы MS-DOS (*bat*-файлы), назначением которых была организация последовательного запуска нескольких программ. Однако возможности tcl гораздо шире и включают в себя объявление переменных и структуры управления, работу с файлами и визуальными компонентами и т. п. Важно, что tcl ориентирован на реализацию расширений, с тем чтобы конечный пользователь системы кроме собственно базового синтаксиса tcl использовал специфичные для своей области команды. Такие команды, предназначенные для управления САПР ПЛИС, и были добавлены программистами Xilinx для Vivado.

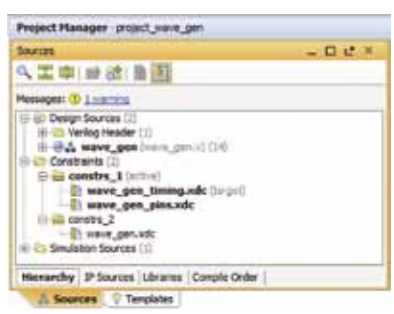

Рис. 1. Управление набором проектных ограничений в Vivado

Для управления проектом можно использовать два режима. Они соответствуют работе в GUI и пакетному режиму, в котором графическая среда разработки Vivado не запускается, а сборка проекта ведется в консольном режиме (рис. 1).

Команды tcl для запуска процессов в Vivado приведены в таблице 1.

При работе Vivado поддерживает базу данных компонентов проекта. Все используемые ресурсы ПЛИС (рис. 2) подразделяются на:

- внешние порты проекта, то есть внешние выводы ПЛИС, участвующие в реализации проекта;
- ячейки — все аппаратные компоненты, включая логические ячейки, аппаратные примитивы, иерархические модули;
- выводы аппаратных компонентов всех уровней (на рис. 2 можно видеть, что в качестве выводов рассматривается не только вход аппаратного примитива FDCE, но и вход иерархического модуля meta_harden, который внутри подключается к этому входу);

Таблица 1. Команды tcl для запуска процессов в Vivado

	Работа в GUI	Консольный режим
Создание проекта	create_project add_files import_files	read_verilog read_vhdl
Синтез	launch_runs *synth1* synth_design	
Реализация проекта (implementation)	launch_runs *impl1*	opt_design power_opt_design place_design phys_opt_design route_design

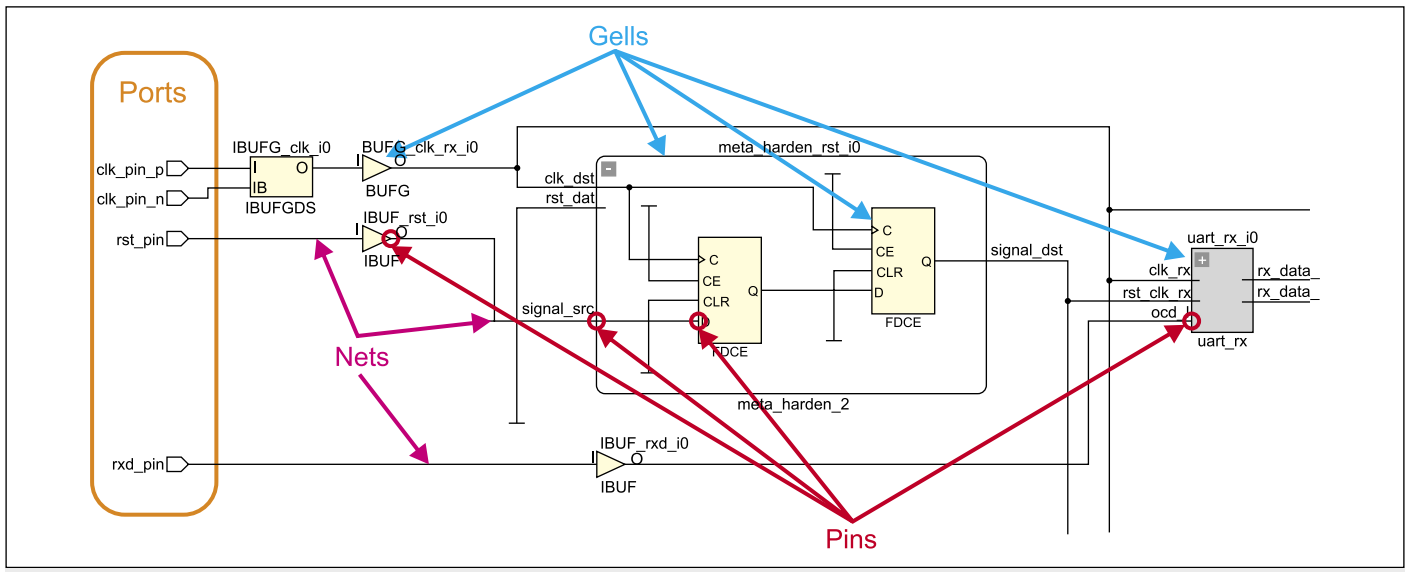

Рис. 2. Элементы базы данных проекта, которыми оперирует Vivado

- соединения между ports/pins в любых сочетаниях.

При задании проектных ограничений можно воспользоваться возможностями tcl по созданию списков из элементов, обладающих определенными свойствами. Например, команда:

```
get_cells -hier -filter {LIB_CELL == FDCE}
```

создаст список аппаратных примитивов типа FDCE, использованных в проекте. Перечень команд tcl, формирующих списки компонентов, приведен в таблице 2.

Все команды, перечисленные в таблице 2, могут комбинироваться с фильтрами. Например, команда:

```
get_ports -filter {DIRECTION == in && NAME =~ *clk*}
```

создаст список внешних выводов проекта, являющихся входами (условие ***DIRECTION == in***) и одновременно не имеющих в своем имени фрагмента ***clk*** (условие ***NAME =~ *clk****).

Создание списков компонентов важно для последующего задания временных или топологических ограничений. Использование регулярных выражений и команд разных типов позволяет более гибко формировать условия фильтрации объектов. Например, вместо указания цепей в виде «цепи с именами nnn» можно выполнить запрос на создание списка в стиле «все цепи, подключенные к входам nnn» или «все цепи, выходящие из группы примитивов nnn и приходящие на входы mmm».

Таблица 2. Команды tcl, формирующие списки компонентов проекта

Команда	Описание
get_cells	Создает список ячеек проекта
get_pins	Создает список внутренних выводов
get_nets	Создает список цепей
get_ports	Создает список внешних выводов
all_inputs	Создает список из всех входов проекта
all_outputs	Создает список из всех выходов проекта
all_ffs	Создает список из всех триггеров (Flip-Flop, FF) проекта
all_latches	Создает список из всех защелок (latch) проекта
all_dsps	Создает список из всех модулей DSP48 проекта
all_rams	Создает список из всех модулей памяти проекта

Проектные ограничения для закрепления выводов ПЛИС

Печатная плата, на которой установлена ПЛИС, разведена вполне определенным образом, так что внешние компоненты должны подключаться к конкретным выводам, определяемым топологией этой платы. Поэтому каждый проект должен сопровождаться списком выводов ПЛИС, к которым подключаются те или иные сигналы схемы. С отказом от формата ucf этот список должен быть представлен в файле *xdc*.

Описание свойств сигналов производится с помощью команды ***set_property***. Основными свойствами, подлежащими определению, являются название вывода и электрический стандарт, который должен быть для него установлен. Пример:

```
set_property PACKAGE_PIN V20 [get_ports wbClk]
set_property IOSTANDARD LVCMOS18 [get_ports wbClk]
```

Необходимость обязательного определения электрического стандарта является новой по сравнению с ucf. Технически каждый банк ввода/вывода в FPGA имеет собственное питание, которое подается на все блоки такого банка. Указание напряжения в электрическом стандарте использовалось для оценки уровня шумов, а также для проверки того, что разработчик не разместил в одном банке выводы, для которых он предполагает разное напряжение питания. При работе FPGA напряжение будет подано «по факту», в зависимости от трассировки печатной платы, и указание в ucf LVCMOS18 при действительном напряжении в 2,5 В не приведет к выходу из строя микросхемы, хотя реальные показатели помехоустойчивости и потребления мощности могут не совпадать с моделью, которая была основана на другом напряжении питания. Отсутствие явного указания электрического стандарта приводило к использованию стандарта «по умолчанию» — LVCMOS, при этом САПР не могла никак проконтролировать возможные ошибки разработчика.

В файле *xdc* разработчик обязан указать, какие электрические стандарты он собирается использовать для каждого блока ввода/вывода. САПР контролирует совместимость стандартов для каждого банка, в частности запрещая трассировку проекта, если в одном банке используются два и более значений напряжения питания.

Кроме обязательных, в xdc могут быть установлены следующие параметры:

- DRIVE — установка выходного тока (в мА). Параметр доступен для некоторых электрических стандартов.
- SLEW — установка скорости нарастания выходного сигнала. Установка пониженной скорости нарастания улучшает помехоустойчивость проекта.
- IN_TERM — установка входного терминатора.
- OUT_TERM — установка выходного терминатора.
- DIFF_TERM — установка дифференциального терминатора.
- KEEPER — установка схемы удержания последнего состояния.
- PULLDOWN — установка «подтягивающего» резистора для обеспечения логического нуля.

- PULLUP — установка «подтягивающего» резистора для обеспечения логической единицы.
- DCI_VALUE — определение поведенческой модели для IOB при создании IBIS-модели.
- DCI_CASCADE — определение набора master и slave банков ввода/вывода для образования цепочки опорного напряжения Vref.
- INTERNAL_VREF — установка использования внутреннего опорного напряжения вместо его подачи с входов Vref.
- IODELAY_GROUP — группировка компонентов IDELAY и IODELAY для совместной работы с DELAYCTRL.
- IOBDELAY — установка задержки компонентов IDELAY и IODELAY.
- IOB — разрешение использования триггеров в блоках ввода/вывода.

Описание временных ограничений

Временные ограничения являются практически обязательной частью любого сколько-нибудь сложного проекта на базе ПЛИС. Для многих проектов оказывается достаточно указать следующие глобальные параметры:

- период тактовой частоты;
- задержка распространения входного сигнала;
- задержка распространения выходного сигнала.

Тактовый сигнал описывается с помощью команды ***create_clock***:

```
create_clock –period 5 –name clk_pin_p [get_ports clk_pin_p]
```

Цепи проекта, которые будут проанализированы после указания такого ограничения, показаны на рис. 3. Можно видеть, что указание периода тактового сигнала позволяет САПР проверить все цепи, соединяющие тактируемые этим сигналом компоненты.

Тактовые сигналы автоматически распространяются по проекту. Это означает, что описание сигнала, приходящего на вход компонента MMCM или PLL, автоматически приведет к появлению корректных описаний выходных сигналов от этих блоков. При этом будут учтены такие факторы, как изменение номинального значения частоты, сдвиг фазы, изменение коэффициента заполнения (Duty Cycle), вносимые джиттер и время нарастания фронта. Рекомендуется задавать проектные ограничения именно для входного тактового сигнала, а не для выходных сигналов, получаемых с помощью MMCM/PLL, поскольку в библиотеках Xilinx параметры MMCM и PLL описаны более точно, чем это может сделать разработчик, руководствуясь своими соображениями. В отчете по тактовым сетям выходные сигналы, параметры которых автоматически определены САПР, помечаются символом «P» (Propagated — распространенные).

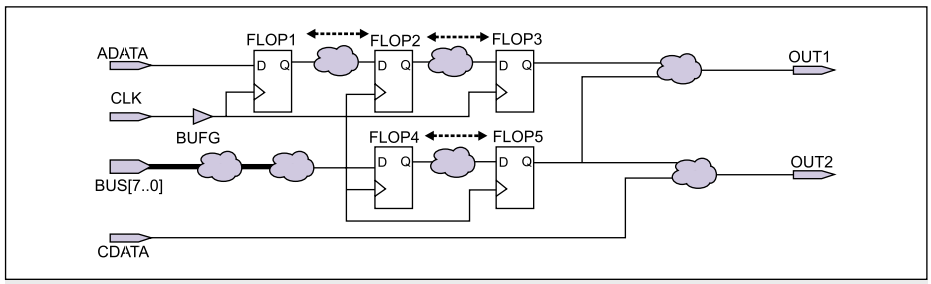

Рис. 3. Действие проектного ограничения create_clock

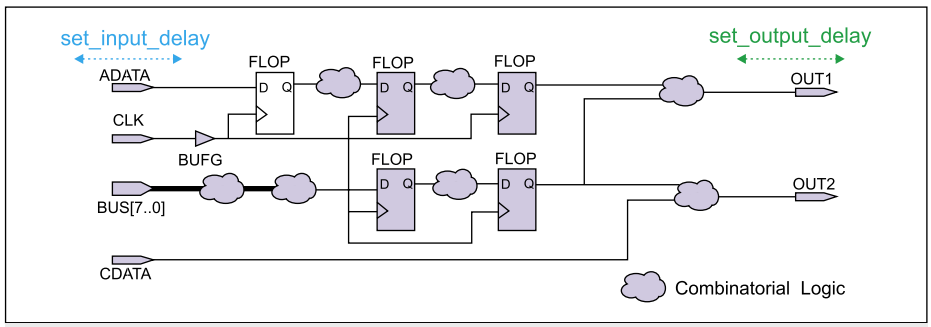

Рис. 4. Действие проектных ограничений set_input_delay и set_output_delay

На рис. 3 также можно видеть, что задержка распространения сигнала от входов ПЛИС до триггеров и от выходов триггеров до выходов ПЛИС не учитывается при проверке ограничения по периоду тактового сигнала. Для задания этих ограничений используются команды ***set_input_delay*** и ***set_output_delay***. (В ucf они задавали проектные ограничения OFFSET IN и OFFSET OUT соответственно.) Действие проектных ограничений ***set_input_delay*** и ***set_output_delay*** показано на рис. 4.

Указанные ограничения могут быть описаны в формате:

```
set_input_delay 2 –clock clk [all_inputs]
set_output_delay 3 –clock clk [all_outputs]
```

Значения задержек, необходимых для приведенных команд, невозможно определить, пользуясь лишь «внутренней» информацией о проекте. Внешняя по отношению к ПЛИС составляющая задержки может быть определена только на основе анализа печатной платы. При этом недостаточно ориентироваться на длину проводников, поскольку в зависимости от импеданса источника, дорожки и приемника величина задержки распространения сигнала может существенно меняться.

Можно указать, что наилучшие характеристики для ограничений ***input_delay*** и ***output_delay*** могут быть достигнуты при установке в проект триггеров, расположенных непосредственно в блоках ввода/вывода.

Для облегчения ввода проектных ограничений существует диалоговое окно Timing Constraints, показанное на рис. 5. Проектные ограничения сгруппированы на панели слева, выбор одного из пунктов открывает соответствующий список, показанный на рис. 5 в правой части окна. При редактировании параметров проектного ограничения в консоли (в нижней части окна) появляется соответствующая строка в формате xdc, которая в действительности и будет добавлена в файл проектных ограничений. С помощью показанного инструмента можно освоить xdc и при соответствующем навыке использовать только текстовое представление.

Как уже было сказано, основными проектными ограничениями для ПЛИС FPGA являются: период тактового сигнала и задержка распространения входных и выходных сигналов. Эти параметры могут быть описаны на xdc с помощью графического интерфейса. На рис. 6 показан пример диалогового окна для настройки параметров тактового сигнала. Можно обратить внимание, что по мере заполнения отдельных полей в строке **Command** (в нижней части окна) формируется строка в формате xdc, которая в действительности будет записана в файл проектных ограничений. Аналогичное окно для ввода параметров задержки входных сигналов показано на рис. 7.

Специалисты Xilinx отмечают, что использование формата xdc имеет ряд преимуществ по сравнению с ucf. На рис. 8 показан пример схемы, для которой требуется сохранить цепь с именем netA. Для ucf необходимо указать имя сохраняемой цепи в явном виде. Однако при дальнейшей работе со схемой это имя может изменяться, так как синтезатор назначает многим внутренним цепям автоматические имена. В то же время для xdc запись может трактоваться как «сохранить цепь, которая подключена к входу D компонента mREG». Компонент имеет больше шансов остаться неизменным в проекте, а его имя обычно соответствует RTL-представлению.

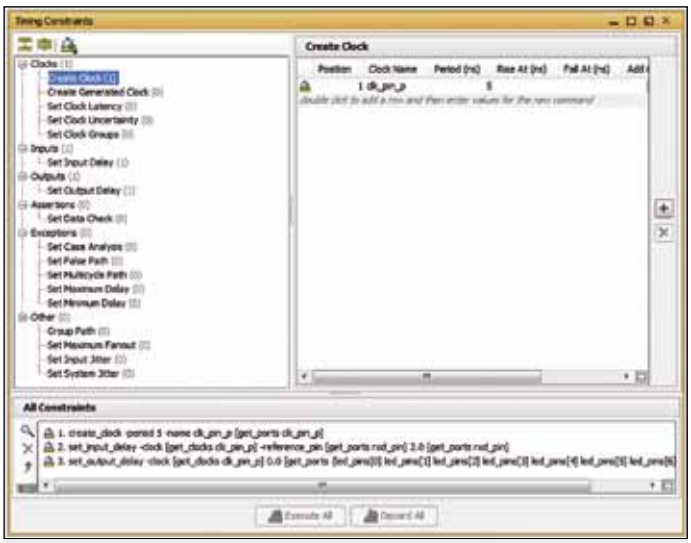

Рис. 5. Диалоговое окно для настройки проектных ограничений

Рис. 7. Диалоговое окно настройки параметров входной задержки

UCF:
NET netA KEEP;

XDC:
set_property DONT_TOUCH 1 [get_nets –of [get_pins mREG/D]]

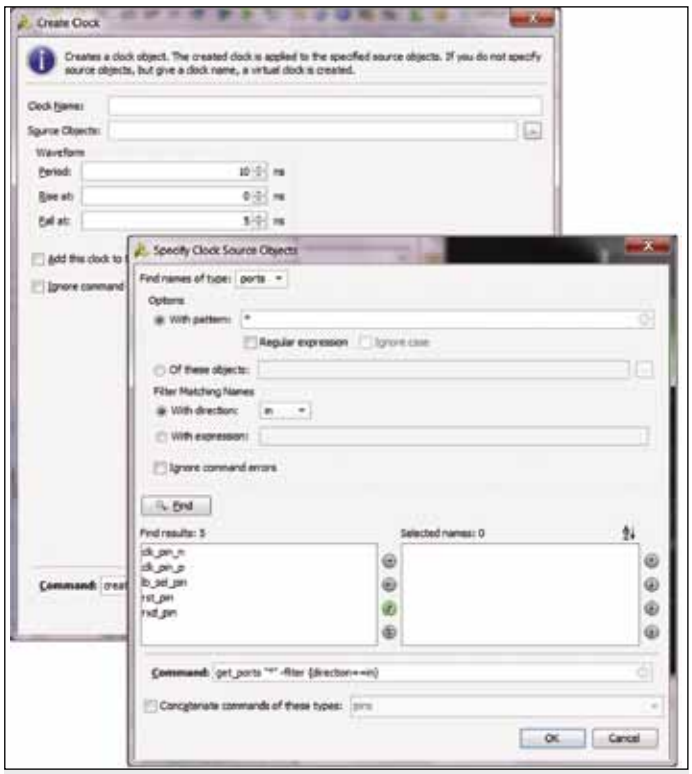

Рис. 6. Диалоговое окно настройки параметров тактового сигнала

Наличие общей базы данных для всех стадий работы с проектом позволяет выполнять сложные запросы для поиска элементов с определенными свойствами. На рис. 9 показан пример схемы, для которой выполняется ряд запросов, приведенных ниже.

Можно применить следующие команды tcl/xdc для доступа к базе данных проекта: ***get_cells***, ***get_nets***, ***get_ports***, ***get_pins*** и т. д. Например:

- вернет триггеры из блока meta_harden:

```
get_cells meta_harden_rst_i0/*reg
```

- вернет все компоненты, имя которых заканчивается на reg:

```
get_cells –hierarchical *reg
```

Можно создавать сложные запросы:

```
get_cells -hier -filter {NAME =~ *meta_harden* && \
IS_PRIMITIVE && (REF_NAME == FDRE || REF_NAME == FDCE)}
```

Этот запрос вернет все аппаратные примитивы типа FDRE или FDCE, которые входят в состав блоков, содержащих в имени ***meta_***

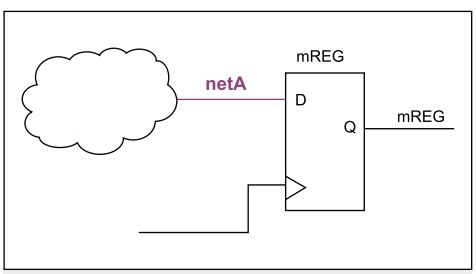

Рис. 8. Пример схемы для выполнения запроса на сохранение внутренней цепи

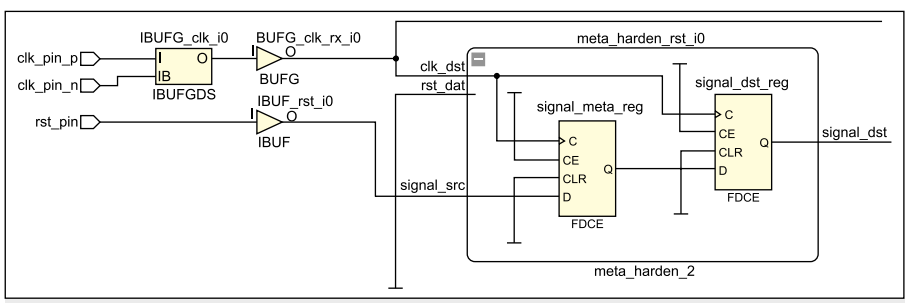

Рис. 9. Пример схемы для выполнения запросов в формате xdc

harden (здесь символ «\» означает продолжение команды на следующей строке).

На рис. 10 показан пример сложного запроса, результатом которого является компонент i2. При этом, как можно видеть, имя этого компонента в запросе не фигурирует, а обнаружить его можно в результате прохождения сложной цепочки: «Найти компонент i1 → найти его выводы → найти подключенные к этим выводам цепи → найти выводы, подключенные к этим цепям → найти компоненты, которым принадлежат эти выводы». В итоге запрос останется работоспособным даже в том случае, если разработчик заменит источник сигнала *x* на какой-то другой компонент.

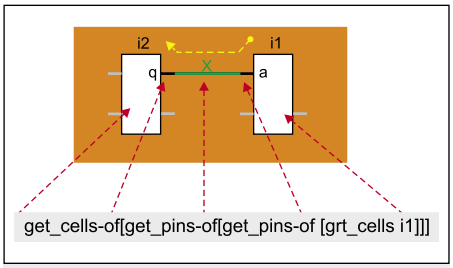

Рис. 10. Пример схемы для создания сложного запроса на xdc

При выполнении запросов можно использовать фильтры и дополнительные ключи. Например, ключ ***–leaf*** (дословно — «лист») управляет возможностью перехода через границы иерархических модулей. Если представить проект в виде дерева, то его «листами» окажутся конечные примитивы, находящиеся на самом глубоком уровне иерархии. На рис. 11 показан пример схемы, для которой выполняются запросы с ключом ***–leaf*** и без него. Можно видеть, что без указания ключа будет найдена ближайшая точка, являющаяся источником сигнала для входа a: выход модуля, обозначенный как *c*. При указании ключа ***–leaf*** запрос будет продолжен вглубь этого модуля, и результатом запроса будет выход *q* компонента i2, который иерархически принадлежит этому модулю.

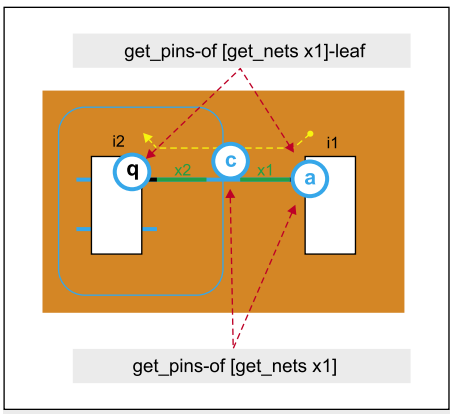

Рис. 11. Пример схемы для выполнения запросов с учетом и без учета иерархии проекта

Описание топологических ограничений

При создании сложных проектов может оказаться необходимым указать регион, в котором следует выполнять размещение конкретного модуля или непосредственно закрепить часть компонентов на кристалле. Понятие «модуль» может относиться как к группе компонентов, получаемой из одного файла, так и к произвольно создаваемой группе, выбираемой пользователем из всей иерархии проекта. Такая группа существует в рамках IDE Vivado и называется P-блоком (P-block). На рис. 12 показано размещение компонентов в проекте, представляемом в качестве примера. На этом рисунке зеленым цветом показаны соединения между модулями и внешними выводами ПЛИС, а толстые красные линии отмечают связи между модулями внутри кристалла.

Однозначное решение по этому рисунку принять сложно, с учетом того, что для показанных блоков могут существовать различные требования к временным задержкам. Однако можно заметить, что блок в левом верхнем углу находится близко к выводам ПЛИС, поэтому для него можно ожидать хороших показателей для ограничений input_delay и output_delay. В то же время блок внизу справа соединен в основном с выводами ПЛИС, находящимися в центре противоположной стороны кристалла. Видимо, хорошим решением будет перемещение этого блока ближе к центру левой части кристалла.

Блок, выделенный на рис. 12 желтым цветом, имеет интенсивные связи с тремя другими блоками. Можно попытаться разместить его на приблизительно равном удалении от них, но также возможно, что он содержит так называемую glue logic (что можно почти дословно перевести как «клей», это выражение технически эквивалентно также словосочетанию «связывающая логика»). Иными словами, в этот блок по каким-то причинам были собраны вспомогательные компоненты, которые по логике работы системы могут и не располагаться компактно. Более того, удаление связанных с этим P-блоком топологических ограничений (то есть разрешение размещать входящие в него ячейки в любых частях кристалла) может улучшить временные характеристики проекта в целом.

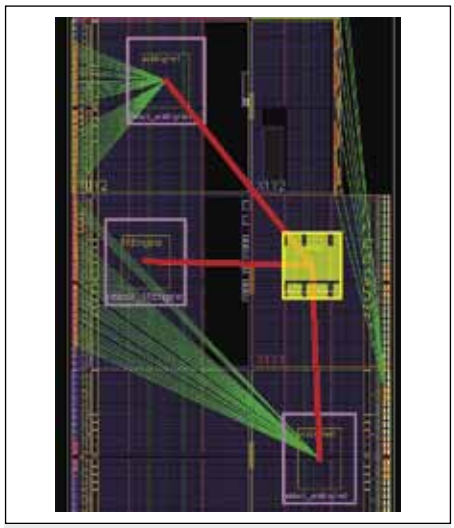

Рис. 12. Пример размещения компонентов в тестовом проекте

Кроме стратегического управления компонентами, можно выполнять анализ критических цепей. После выполнения стадии implement и анализа задержек будет получен список цепей с наихудшими задержками («критические цепи»). Разработчик может применять разные подходы для исправления ситуации, если такая цепь находится внутри одного иерархического модуля или же соединяет разные модули. Например, на рис. 13 показана цепь, соединяющая две логические ячейки внутри одного модуля.

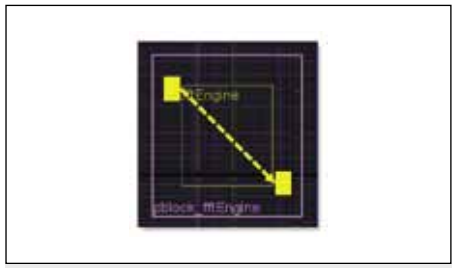

Рис. 13. Размещение критической цепи внутри иерархического модуля

В показанном примере разработчик может выполнить следующее:

- изменить размеры модуля;
- указать для модуля дополнительный P-блок, разместив критическую цепь внутри этого блока;
- установить топологические ограничения для привязки компонентов критической цепи к абсолютным координатам на кристалле или к относительным координатам блока.

На рис. 14 показан пример, в котором критическая цепь соединяет компоненты из разных модулей. Очевидно, что наилуч-

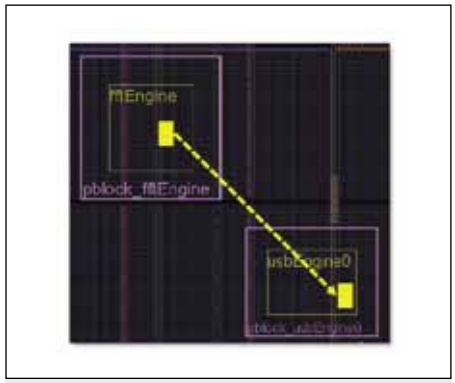

Рис. 14. Размещение критической цепи между модулями

шим вариантом было бы помещение двух компонентов в соответствующие углы своих P-блоков, ближайшие друг к другу. Однако для P-блоков выделены большие прямоугольники, что формально разрешает предварительное размещение компонентов в любых частях выделенных областей. В этом случае разработчику остается надеяться, что при последующей оптимизации алгоритмы Place&Route смогут автоматически найти правильное положение этих компонентов. Естественно, эту неопределенность можно устранить с помощью дополнительных мероприятий:

- перенос критической цепи в один из P-блоков;
- создание дополнительного блока, содержащего оба компонента, формирующих критическую цепь, и выделение компактной области для этого компонента, гарантирующей достижение требуемых характеристик;
- жесткая фиксация компонентов внутри P-блоков.

Говоря о фиксации размещения (fix placement), следует отметить, что она трудоемка и при этом не гарантирует улучшения характеристик проекта. Корректнее было бы говорить о том, что при фиксации размещения временные характеристики определяются квалификацией разработчика, его уровнем понимания архитектуры ПЛИС и вниманием, проявленным к деталям конкретного проекта. При разработке больших схем чрезвычайно трудно уделить внимание всем аспектам размещения, поэтому прямое управление топологией следует рассматривать не как обязательный признак качественного проекта, а скорее как вынужденную меру, на которую разработчик идет в качестве «последнего средства» для достижения требуемых характеристик.

Для фиксации размещения компонентов используются следующие команды xdc (с примерами):

- Команда устанавливает положение (location, LOC) для компонента ram0, входящего в блок u_ctrl0:

```
set_property LOC RAMB18 X0Y10 [get_cells u_ctrl0/ram0]
```

- Команда устанавливает для компонента конкретный базовый элемент логики (base element of logic, BEL). В этом примере указано, что компонент lut0 в блоке u_ctrl0 следует разместить в таблице истинности C5 (5-входовый режим LUT, третья LUT в секции согласно обозначениям A, B, C, D):

```
set_property BEL C5LUT [get_cells u_ctrl0/lut0]
```

- Команда предписывает размещение регистров, содержащих в имени mData_reg, в блоках ввода/вывода:

```
set_property IOB TRUE [get_cells mData_reg*]
```

- Команда размещает две функции в одной физической LUT. Это возможно для некоторых логических выражений при условии, что часть входов у них общая и суммарное количество независимых входов не превышает шести:

```
set_property LUTNM L0 [get_cells {u_ctrl0/dmux0 u_ctrl0/dmux1}]
```

- Команда запрещает (prohibit) использование ресурсов в указанных координатах. Это может быть полезно, если эти координаты необходимо зарезервировать для гарантированного размещения в них других компонентов:

```
set_property PROHIBIT TRUE [get_sites {RAMB18_X0Y* RAMB36_X0Y*}]
```

Описание проектных ограничений в формате xdc, по сути, представляет собой программирование на расширении языка tcl. Поэтому при конфликте команд последующая команда просто перекрывает действие предыдущей. Это свойство следует иметь в виду при анализе результатов действия сложных сценариев, в разных частях которых могут оказываться команды, дублирующие или отменяющие ранее установленные ограничения.

Заключение

В результате перехода к формату xdc, основанному на промышленном стандарте в области проектирования интегральных микросхем, Xilinx предлагает разработчикам мощную систему управления проектом на базе текстовых сценариев. Добавление к проекту таких сценариев ускоряет размещение и трассировку, а также обеспечивает лучшее воспроизведение топологических и временных характеристик.

Литература

1. Vivado Design Suite User Guide. Using Constraints. http://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_2/ug903-vivado-using-constraints.pdf