Продолжение. Начало в № 3`2009

Изучаем Active-HDL 7.1. Урок 13. Методы построения эффективных SDE-проектов

Александр ШАЛАГИНОВ
shalag@vt.cs.nstu.ru

Графический редактор **State Diagram Editor** предлагает два основных метода построения компактных, наглядных и эффективных диаграмм состояний цифровых автоматов:

- поведенческий стиль представления диаграмм состояний;
- иерархическое описание диаграмм состояний.

Поведенческое описание цифровых автоматов

С поведенческим стилем описания цифровых автоматов мы познакомились в конце прошлого урока. Правда, рассмотренный там пример был фактически забракован из-за многочисленных недостатков.

Рассмотрим более привлекательный вариант той же самой диаграммы (рис. 1), реализующей работу двоичного реверсивного счетчика.

Главная ее особенность заключается в том, что в графическую модель счетчика добавлен внутренний сигнал **Q_INT** (**INT**, от слова **internal** — внутренний). Он задается командой **signal\variable**, которая вызывается из пункта **FSM** основного меню. На инструментальной панели **FSM VHDL Diagram Toolbar** данная команда дублируется соответствующей кнопкой.

Будьте внимательны, данный объект надо разместить за пределами прямоугольной рамки автомата (рис. 1), иначе будет создан не внутренний сигнал, а локальная переменная.

Внутренний сигнал имеет одно несомненное достоинство: его разрешается использовать как в левых, так и в правых частях аналитических выражений. Теперь все необходимые действия выполняются именно с этим сигналом, а полученные результаты передаются на выход операцией параллельного назначения **Q<=Q_INT**. Удобнее всего данное выражение поместить в специальный графический элемент **diagram ACTION**, как это сделано в нашем примере.

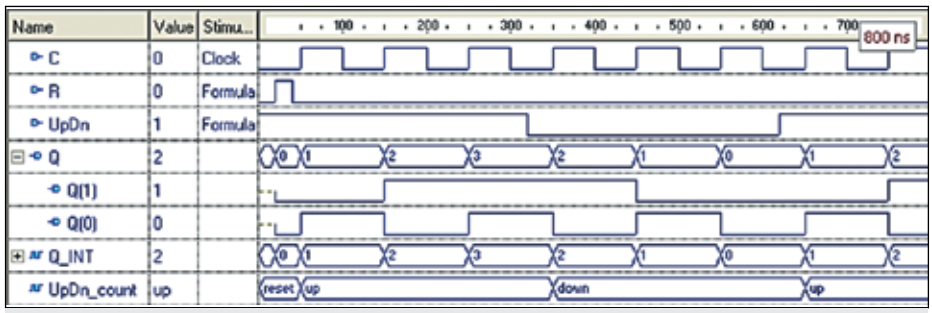

Рис. 2. Результаты моделирования реверсивного счетчика, заданного поведенческим описанием цифрового автомата (предпочтительный вариант)

Обратите внимание, теперь выход **Q** имеет статус выходного, а не двунаправленного порта, как в предыдущем случае. Кроме того, выход имеет тип **Combinatorial** («Комбинационный»), а это основной (базовый) тип выходного сигнала, устанавливаемый системой по умолчанию.

Результаты моделирования, показанные на рис. 2, подтверждают правильность работы реверсивного счетчика.

Рассмотренный пример не очень показателен с точки зрения экономии числа состояний автомата: сейчас мы имеем три состояния вместо четырех для потокового стиля описания (урок 12, рис. 19). Однако в случае использования 4-разрядного счетчика выигрыш станет более очевидным (рис. 3).

Чтобы усовершенствовать графическую модель счетчика, наделив ее свойством «настраиваемой разрядности», внесем в предыдущую диаграмму состояний некоторые изменения. В «шапке» диаграммы объявим новый элемент — **generic**. Это и есть настраиваемый параметр. Он подобен константе, но, в отличие от нее, позволяет передавать из внешней среды в модель не только статическую информацию, но и оперативно изменять параметры настройки, в данном случае разрядность счетчика.

Чтобы все работало, нам необходимо сделать сброс счетчика нечувствительным к числу его разрядов. С этой целью заменим для состояния **Reset** выражение **Q_INT<="00"**; на более подходящее: **Q_INT<= (others => '0');**. Потребуется

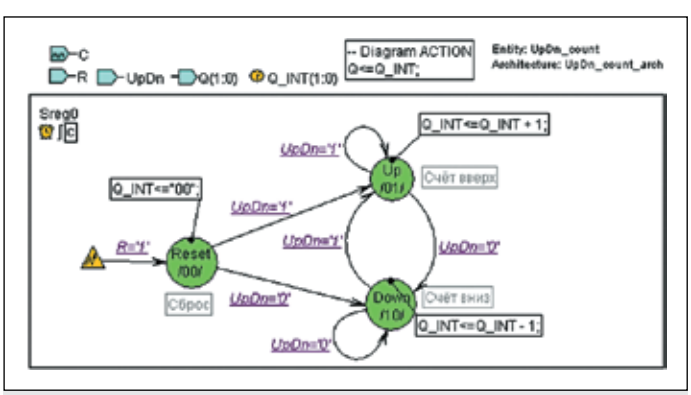

Рис. 1. Предпочтительный вариант поведенческого описания цифрового автомата

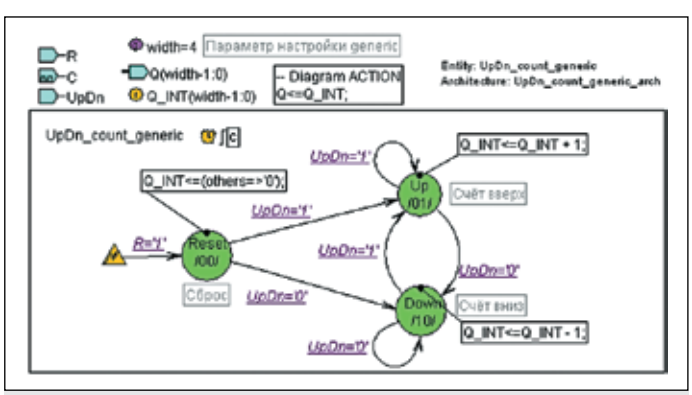

Рис. 3. Рекомендуемый стиль поведенческого описания цифровых автоматов

также отредактировать ширину шины выходного **Q** и внутреннего **Q_INT** сигналов.

Графическое представление модели счетчика больше не зависит от его разрядности, а число состояний в нем по-прежнему остается равным трем. Например, чтобы промоделировать 8-разрядный счетчик, достаточно изменить параметр настройки **generic** с 4 на 8. А ведь при потоковом описании проекта в диаграмме состояний 8-разрядного счетчика появилось бы 256 состояний.

Обратите внимание еще на один элемент диаграммы — это комментарий (**Comment**). Он представляет собой дополнительный текст, который вы можете присоединить к любому типу объектов диаграммы. Важно заметить, что сделанный комментарий по умолчанию автоматически вставляется в генерируемый системой **VHDL**-код на этапе конвертации диаграммы в текст.

Чтобы задать комментарий, достаточно вызвать диалоговую панель «Свойства объекта», например, командой **Properties** из всплывающего меню, активизировать закладку **Comment** и ввести требуемый текст.

Иерархическое описание цифровых автоматов

Основное достоинство графического описания проекта — его наглядность — во многом теряется, если проект разрастается до значительных размеров. Не спасает даже поведенческий стиль представления диаграммы состояний. В этой ситуации на помощь приходит иерархическое описание проекта, при котором внутри одного иерархического состояния верхнего уровня можно спрятать целый подграф дочернего уровня.

Иерархическое состояние изображается на диаграмме кружком синего цвета с тройной обводкой. Однако прямой команды для размещения такого состояния на диаграмме автомата вы не найдете ни на инструментальных панелях, ни в главном, ни во всплывающем меню.

Для получения иерархического состояния придется сначала поместить на чертеж графическое изображение обычного (стандартного) состояния, а затем превратить его в иерархическое.

Это можно сделать, по крайней мере, тремя способами:

- Дважды щелкнуть на стандартном состоянии и установить на открывшейся диалоговой панели **State Property** флажок **Hierarchical**.
- Вызвать контекстное меню и выполнить команду **Hierarchical State**.
- Выполнить команду из главного меню: **FSM/Hierarchy/Convert to Hierarchical State**. Заметим, что последняя команда дублируется иконкой на инструментальной панели **FSM Hierarchy Toolbar**.

Первые два способа применимы только к одному (выделенному) состоянию, третьим способом вы можете преобразовать в иерархическое состояние целую подсхему, содержащую много состояний и переходов.

Попробуем упростить диаграмму состояний реверсивного счетчика, которую мы получили в результате предыдущих экспериментов (рис. 3). Одно за другим заменим стандартные состояния **Reset**, **Up** и **Down** иерархическими, используя любой из двух первых способов. В результате получится диаграмма, содержащая только иерархические состояния (рис. 4). Командой **Push Hierarchical State** можно понизить уровень иерархии и посмотреть содержимое иерархических состояний (они показаны на рис. 4).

Все, что удалось убрать с верхнего уровня и «погрузить» внутрь иерархических состояний,— это графический объект **Reset** и выражения, описывающие действия на входах в состояния.

Обратите внимание, названия состояний верхнего уровня не изменились, и комментарии к ним остались на прежнем уровне.

Более впечатляющие результаты получаются, если использовать программу преобразования фрагментов диаграммы в иерархические состояния (третий способ). Вернемся к первоначальному описанию автомата, реализующего функцию реверсивного счетчика. Для этого достаточно выполнить «откатку», многократно повторяя команду **Undo** или нажимая горячие клавиши **Ctrl+Z**.

Выделим первое состояние **Reset** и нажмем на кнопку **Convert to Hierarchical State**. Проделаем эту операцию для двух оставшихся

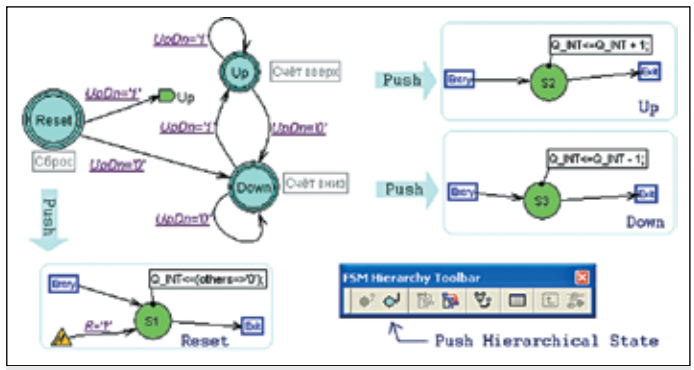

Рис. 4. Замена стандартных состояний иерархическими упрощает описание цифрового автомата

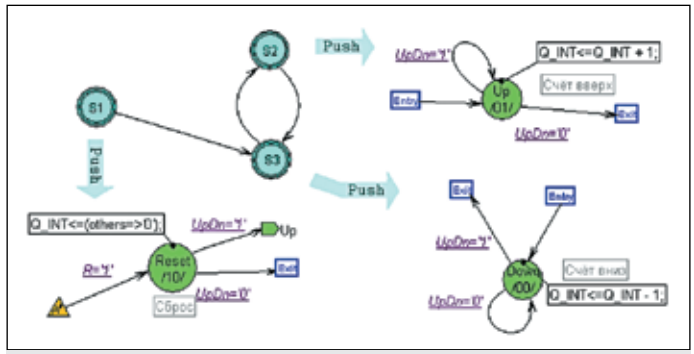

Рис. 5. Диаграмма состояний реверсивного счетчика после автоматической конвертации ее фрагментов в иерархические состояния

состояний **Up** и **Down**. В результате вы увидите иерархическую диаграмму, показанную на рис. 5.

Обратите внимание, на дочерние листы перенесена практически вся сопутствующая диаграмме информация: объект **Reset**, выражения на входах в состояния, комментарии, условия на переходах, переходы в форме петли и переходы, заканчивающиеся ссылкой **Link**. Да и старые названия состояний оказались на более низком уровне. Их заменили формальные имена **S1**, **S2** и **S3**.

У вас появились опасения, что разработчики программы **Convert to…** перестарались и «вместе с водой выплеснули из ванны и ребенка»? Автору тоже показалось, что диаграмма верхнего уровня стала практически не информативна. Впрочем, у нас всегда остается возможность вручную подправить полученную диаграмму.

Проведем еще один показательный эксперимент. Используя «откатку», вновь придадим диаграмме реверсивного счетчика первоначальный вид. Затем выделим состояние **Reset** и конвертируем его. Впрочем, так мы уже действовали.

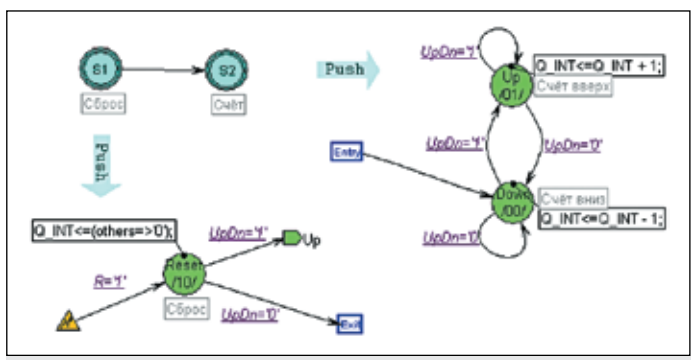

Рис. 6. Программа конвертации позволяет «упрятать» в одно иерархическое состояние (S2) целый функциональный блок (состояния Up — счет вверх и Down — счет вниз)

А вот далее поступим по-другому. Выделим оба оставшихся состояния **Up** и **Down** (удерживая нажатой клавишу **Ctrl**) и преобразуем их совместно в одно иерархическое состояние (рис. 6). Теперь у нас на диаграмме только два состояния: **S1** — «Сброс» и **S2** — «Счет».

Все детали поведения счетчика не видны на диаграмме верхнего уровня иерархии и обнаруживают себя только в том случае, если раскрыто содержимое иерархического состояния.

Понятно, что в текущий момент вы можете видеть дочерний лист только одного иерархического состояния.

Сравнивая иерархические состояния с обычными состояниями, можно заметить несколько особенностей, в частности, в иерархическое состояние нельзя задать переход от объекта **Reset** или назначить для него действия.

Наверное, вы уже обратили внимание, что стандартные состояния автомата, размещенные на дочерних листах графа переходов, имеют составные имена, а их формат зависит от способа, которым получены иерархические состояния.

В сгенерированном системой **HDL**-коде они будут представлены в виде:

<hierarchical_state_name>_<state_name> или <state_name>_<hierarchical_state_name>.

Например, для диаграммы состояний, изображенной на рис. 5, названия имен получаются такими: `Reset_S1, Up_S2, Down_S3`, а для схемы на рис. 6 (полученной с помощью конвертации) — все наоборот: `S1_Reset, S2_Up, S2_Down`.

Самое неприятное в иерархических диаграммах — это реальная опасность потерять графическую информацию, размещенную на дочерних листах. Если вы передумали и захотели вернуться от иерархического состояния к обычному, то можете потерять вложенные в иерархию описания. Впрочем, система предупредит о такой неприятности, выдав на экран монитора **Warning** (рис. 7).

Поэтому лучше возвращаться к стандартному состоянию, выполняя простую «откатку» командой **Undo** (**Ctrl**+**Z**).

Рис. 7. Предупреждение системы о возможности потерять графическую информацию, размещенную на вложенных уровнях

Анализируя фрагменты диаграмм состояний, показанные на рис. 4–6, легко заметить, что связь между объектами, размещенными на различных уровнях иерархии, осуществляется с помощью объектов **Entry** и **Exit**. Исключение составляет только элемент **Link**, обеспечивающий прямую связь между конкретными состояниями (рис. 6). Заметим, что связываемые объектом **Link** состояния могут находиться как на одном, так и на различных уровнях иерархии.

Однако вернемся к элементам **Entry** и **Exit**. Если иерархическое состояние имеет много входящих в него переходов, то все они как бы объединяются в один узел и проникают на дочерний лист через одну точку — элемент **Entry** («Ввод»).

То же самое справедливо и для элемента **Exit**, то есть он связан со всеми переходами, соединяющими это иерархическое состояние с другими состояниями на листе верхней иерархии.

Такая структурная организация напоминает по входу мультиплексор (коммутатор), а по выходу — демультиплексор (распределитель). Соединяя последовательно названные устройства, можно обеспечить режим коммутации «каждый с каждым». Это свойство широко используется для сокращения числа переходов на диаграмме состояний цифрового автомата.

Способы компактного описания цифровых автоматов

На рис. 8 показано, как иерархическое состояние **HS** исполняет роль коммутатора состояний **S1… S6** (**junction state**).

В данном случае экономятся три перехода и повышается наглядность диаграммы. При большем числе коммутируемых состояний выигрыш становится гораздо ощутимее.

Приведем еще один пример, подтверждающий эффективность данного метода. На рис. 9 изображен цифровой автомат, реализующий функцию 2-разрядного регистра с тактируемой параллельной загрузкой. Автомат имеет всего четыре состояния, но из-за большого числа переходов даже такой простой граф теряет наглядность.

На рис. 10а изображена диаграмма состояний того же самого автомата, но теперь она легко читается, так как не перегружена большим числом переходов.

Здесь мы применили два приема компактного описания графа переходов: глобальный переход с условием **L='1'** и иерархическое состояние **HS1** в качестве коммутатора состояний.

Обратите внимание, проще становится не только диаграмма, но и условия выполнения переходов, так как сложное выражение типа **L='1' and D="10"** заменяется двумя простыми **L='1'** (на глобальном переходе) и **D="10"** на выходе из иерархического состояния **HS1**.

Хотя графический объект **HS1** и имеет статус иерархического состояния, в действительности он таковым не является, потому что автомат в нем «не застревает» даже на один такт, то есть проходит его насквозь без задержки.

Вероятно для того, чтобы не вводить пользователя в заблуждение, разработчики системы Active-HDL 7.1 добавили в редактор диаграмм еще один объект — **Junction** (соединитель).

Замените иерархический объект **HS1** соединителем **J1** (имя **J1** задается по умолчанию) и вы убедитесь в том, что их функции одинаковы (рис. 10б). Однако соединитель **Junction** не является иерархическим объектом, а потому значительно удобнее в работе. Более того, в сгенерированном системой **VHDL**-коде вы вообще не обнаружите следов объекта, называемого соединителем.

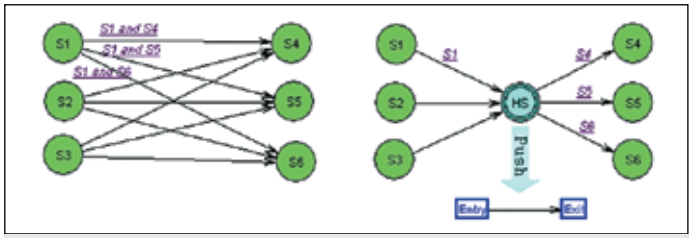

Рис. 8. Иерархическое состояние HS играет роль коммутатора состояний junction state

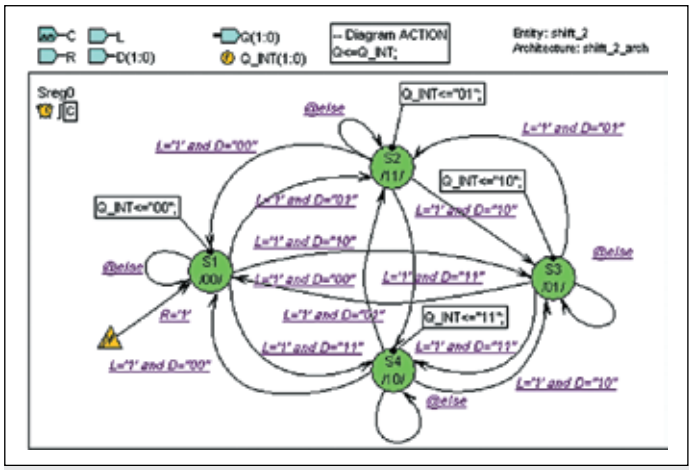

Рис. 9. Простой цифровой автомат 2-разрядного регистра, перегруженный большим числом переходов (режим коммутации «каждый с каждым»)

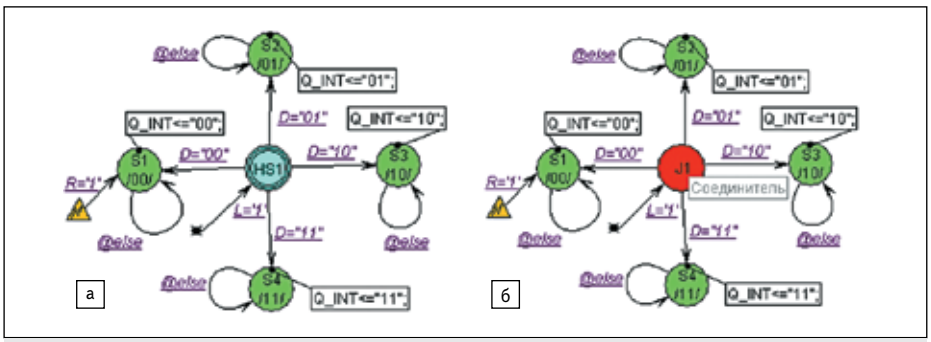

Рис. 10. Диаграмма состояний 2-разрядного регистра становится гораздо проще и нагляднее при использовании:
а) глобального перехода L='1' и иерархического состояния HS1 в качестве коммутатора состояний или
б) соединителя Junction

Для размещения соединителя следует выполнить команду главного меню **FSM/Junction** или нажать пиктограмму с тем же названием на инструментальной панели **FSM/VHDL Diagram Toolbar**.

Заслуживает внимания еще один показательный пример — диаграмма состояний универсального регистра с асинхронной параллельной загрузкой, показанная на рис. 11.

Первая особенность, которая бросается в глаза, — на рис. 11 не один, а два объекта типа **Reset**. Функция одного оправдывает свое название и асинхронно устанавливает регистр в исходное состояние. Оно так и называется: **Reset**. При этом все триггеры регистра обнуляются: **Q_INT <= (others=>'0');**.

Второй элемент **Reset** связан с состоянием **Load**, в котором в регистр загружаются данные с входной шины **D**. Загрузка выполняется при условии, что на управляющем входе **L** (от слова **Load**) присутствует сигнал высокого уровня. Если в этом режиме меняется информация на входе **D**, то должно измениться и содержимое регистра, это происходит до прихода тактирующего сигнала **C**.

Чтобы реализовать эту особенность асинхронного управления, в условие на переходе (рис. 11) добавлено выражение: **L='1' and D'event**. Загадочная запись **D'event** на языке **VHDL** означает изменение сигнала **D** (от слова **event** — событие, изменение, переключение).

Другими словами, переход в состояние **Load** (параллельная загрузка) будет выполняться не только в тот момент, когда на входе **L** появится высокий уровень, но и всякий раз, когда в данном режиме (при **L='1'**) будут обновляться данные на входе **D**.

Возможна ситуация, когда одновременно выполняются оба условия, то есть сигналы **R** и **L** активны на перекрывающемся интервале времени. В реальных микросхемах (например, 555ИР11 или 155ИР13) команда сброса является более приоритетной, и сброс должен преодолеть параллельную загрузку.

Чтобы разрешить описанную коллизию, следует задать различные приоритеты на переходах. Графический редактор **SDE** позволяет это сделать. Приоритет перехода задается целым числом в диапазоне 0–4095. Визуально оно отображается на стрелке перехода . Самый высокий приоритет имеет переход с порядковым номером 0. По умолчанию всем переходам присваивается такое значение, и обычно (если не установлена опция **Transition Auto Priority**) они не видны на диаграмме состояний.

Чтобы задать переходу конкретное (отличное от нуля) значение, надо вызвать для него диалоговую панель **Transition Properties** и в разделе **Priority** задать желаемое значение. В нашем случае более низкий приоритет должен иметь переход, переводящий автомат в режим параллельной загрузки, поэтому на рис. 11 ему присвоено значение 1.

При **L='0'** регистр переходит в режим сдвига (вход в соединитель **J1**). Однако предстоит еще выяснить направление сдвига. При **L_R='1'** должен выполняться сдвиг влево (в сторону старших разрядов), а в освободившийся при сдвиге младший разряд **Q0** заноситься бит с входа **DL**. При **L_R='0'** выполняется сдвиг вправо, а в освободившийся старший разряд **Q3** записывается бит с входа **DR**.

Так как условия на выходе из **Junction** (соединителя) взаимоисключающие, то задавать разные приоритеты на выходящих из него переходах не требуется.

На временных диаграммах, отражающих результаты моделирования универсального регистра (рис. 12), курсорами выделен временной интервал 320–400 нс, на котором одновременно активны команды сброса и асинхронной параллельной загрузки. Видно, что автомат реагирует на более приоритетную команду сброса и игнорирует команду параллельной загрузки.

Разговор о поведенческом стиле представления диаграмм состояний будет неполным, если мы обойдем вниманием графический объект, называемый переменной (**variable**). Переменная Variable1, как и сигнал Signal1, вы-

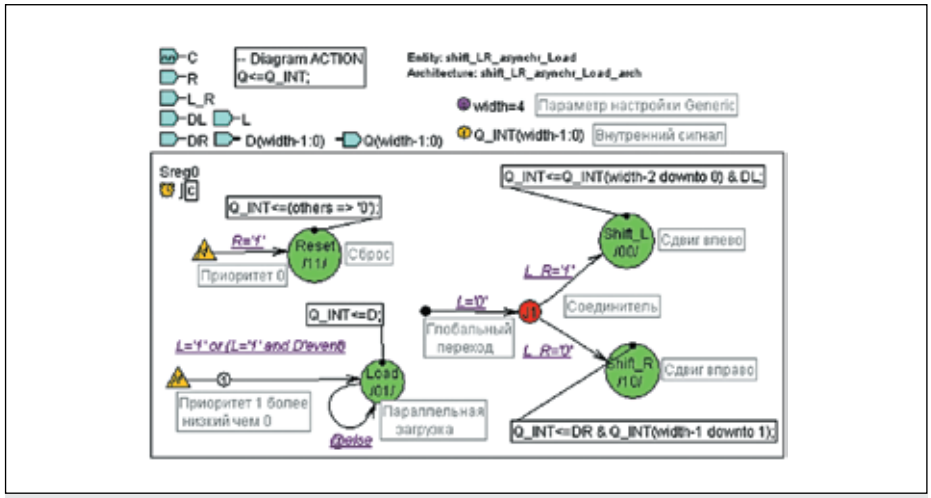

Рис. 11. Диаграмма состояний универсального регистра сдвига с асинхронной параллельной загрузкой

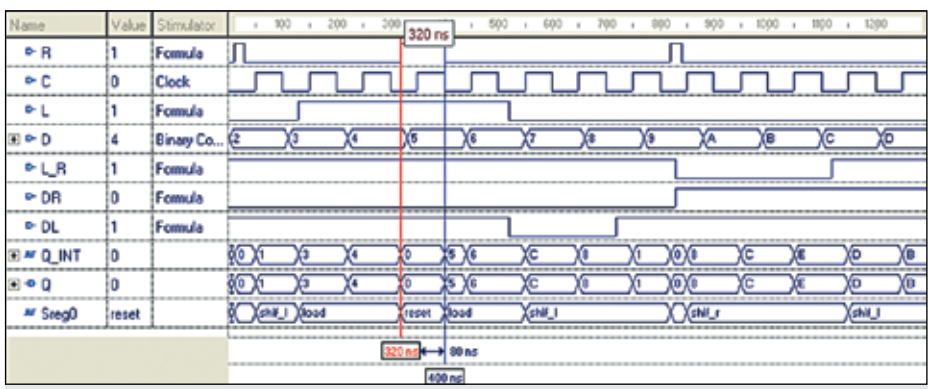

Рис. 12. Результаты моделирования универсального регистра сдвига с асинхронной параллельной загрузкой

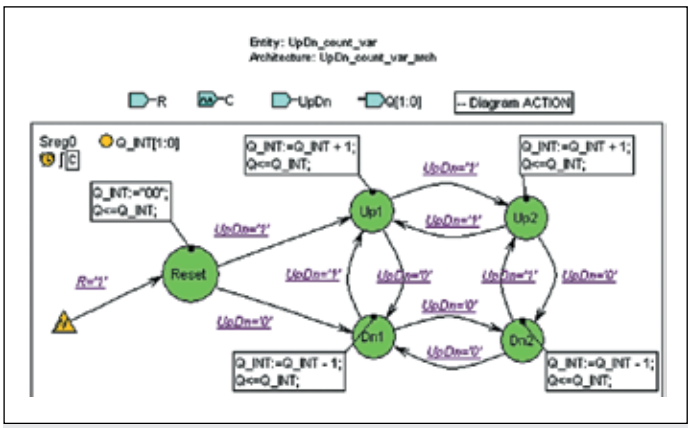

Рис. 13. Поведенческий стиль описания цифрового автомата с заменой внутреннего сигнала Q_INT переменной Q_INT

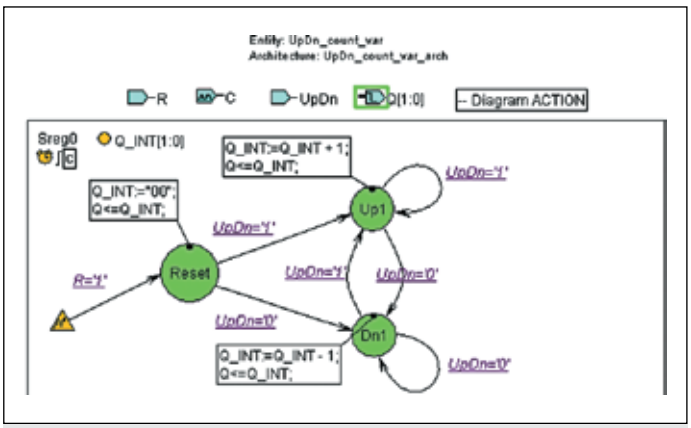

Рис. 14. Предпочтительная реализация цифрового автомата, выполняющего функцию реверсивного счетчика

зываются одной и той же командой **signal/variable** из меню **FSM**, и этот факт может навести на мысль, что и функции у них схожие.

Внутренний сигнал **Q_INT** мы неоднократно использовали для хранения текущего состояния цифрового автомата. Нельзя ли использовать для этих же целей и переменную?

Попробуем найти ответ, используя в качестве объекта исследования построенную ранее диаграмму состояний реверсивного счетчика (рис. 1).

Заменить внутренний сигнал **Q_INT [1:0]** переменной с тем же именем не составляет труда (рис. 13). Для этого достаточно отбуксировать его внутрь прямоугольной рамки, задающей габаритные размеры автомата. Что же еще предстоит сделать?

Прежде всего, придется заменить все операторы назначения сигнала, задающие действия на входах в состояния (Q_INT<="00"; Q_INT<=Q_INT + 1; Q_INT<=Q_INT - 1;), на соответствующие операции присвоения переменной (Q_INT:="00"; Q_INT:=Q_INT + 1; Q_INT:=Q_INT - 1;).

Но вот беда, операция параллельного назначения Q<=Q_INT;, помещенная в специальный графический объект **Diagram ACTION** (рис. 1), перестает работать. А все дело в том, что в языке **VHDL** параллельный оператор назначения <= активизируется только при изменении сигналов в его правой части, а там теперь сигнала нет, только переменная **Q_INT**, на изменения которой оператор не реагирует.

Ничего не поделаешь, придется тиражировать оператор назначения во все состояния автомата и удалить его из объекта **Diagram ACTION**. Однако и это не решает всех проблем. Попытка откомпилировать проект дает ошибку **Unknown identifier Q_INT** («Неизвестный идентификатор **Q_INT**»).

Ошибка легко обнаруживается, если посмотреть автоматически созданный **VHDL**-код. Любопытно, что компилятор сам сделал эту ошибку, поместив переменную **Q_INT** за пределами процесса, где она не видна.

Чтобы устранить дефект, достаточно расширить область видимости переменной **Q_INT**, изменив ее статус: вместо локальной переменной объявить глобальную. Выполним команду **FSM/Code Generation Settings** и на открывшейся панели с тем же названием установим флажок Use SHARED variables.

Теперь ошибок компиляции нет, но выходы по-прежнему вычисляются неправильно. Причину мы уже знаем: оператор условного назначения не реагирует на изменения переменной **Q_INT** и начинает действовать только при смене состояний автомата.

Чтобы автомат «не застревал» в одном состоянии на несколько тактов, пойдем на небольшую уловку. Продублируем состояния **Up** и **Down** (рис. 1) и организуем процесс так, чтобы на каждом такте автомат менял свои состояния (рис. 13).

Последнее замечание касается состояния **Reset**. Чтобы при сбросе (условие **R='1'**) выходные сигналы обнулялись, нужно, чтобы состояние **Reset** не являлось состоянием по умолчанию, в которое устанавливается автомат при инициализации моделирования. Другими словами, имя **Reset** не должно занимать левую позицию в перечне возможных состояний автомата (**VHDL**-код).

Чтобы достичь желаемого, достаточно активизировать команду **View/Sort Object** из меню **FSM** и на закладке **State** отбуксировать состояние **Reset** с верхней позиции в любое другое место. Впрочем, подобными операциями мы уже занимались, обсуждая работу цифрового автомата с функцией Т-триггера (урок 12, рис. 8).

```
-- signal assignment statements for combinatorial outputs
Q_assignment:
Q <= Q_INT when (SregO = Reset and UpDn='0') else
     Q_INT when (SregO = Reset and (UpDn='1' and not (UpDn='0'))) else
     Q_INT when (SregO = Reset) else
     Q_INT when (SregO = Dn1 and UpDn='0') else
     Q_INT when (SregO = Dn1 and (UpDn='1' and not (UpDn='0'))) else
     Q_INT when (SregO = Up1 and UpDn='1') else
     Q_INT when (SregO = Up1 and (UpDn='0' and not (UpDn='1'))) else
     Q_INT when (SregO = S1 and UpDn='0') else
     Q_INT when (SregO = S1 and (UpDn='1' and not (UpDn='0'))) else
     Q_INT when (SregO = S2 and UpDn='0') else
     Q_INT when (SregO = S2 and (UpDn='1' and not (UpDn='0'))) else
     Q_INT;
```

Рис. 15. Программная реализация на языке VHDL (фрагмент): для комбинационного выходного сигнала используется оператор условного назначения, который помещается в конце кода

```
SregO_machine: process (C, r)
-- machine variables declarations
variable Q_INT: STD_LOGIC_VECTOR (1 downto 0);

begin
    if R='1' then
        SregO <= Reset;
        -- Set default values for outputs, signals and variables
        -- ...
        Q_INT := "00";
        Q <= Q_INT;
    elsif C'event and C = '1' then
        -- Set default values for outputs, signals and variables
        -- ...
        case SregO is
            when Reset =>
                if UpDn='0' then
                    SregO <= Dn1;
                    Q_INT := Q_INT - 1;
                    Q <= Q_INT;
                elsif UpDn='1' then
                    SregO <= Up1;
                    Q_INT := Q_INT + 1;
                    Q <= Q_INT;
                end if;
```

Рис. 16. Программная реализация на языке VHDL (фрагмент): для регистрового выходного сигнала используется оператор назначения, встраиваемый в тело процесса

Ну вот, наконец-то получен полноценный результат, но дорога к нему изобиловала многими неочевидными решениями. Думаем, что убедили вас в том, что рассмотренный пример нельзя назвать образцом решения подобного сорта задач.

В принципе имелась более простая возможность достичь желаемого результата, но до сих пор мы не интересовались, каким образом операторы действия помещаются в сгенерированный системой **VHDL**-код. Оказывается, стоит изменить тип выходного сигнала, и ситуация поменяется коренным образом.

На рис. 14 показана более простая и естественная реализация цифрового автомата, выполняющего функцию реверсивного счетчика. Назначение выходному порту Q [1:0] регистрового типа **Registered** снимает все проблемы.

Как и в предыдущем примере, используется переменная Q_INT[1:0], но выходной сигнал теперь имеет тип **Registered** Q[1:0]. Об этом говорит срез импульса на условном графическом изображении порта. Все! Больше никаких хитростей не требуется.

Чтобы сменить тип порта, достаточно дважды щелкнуть на его изображении и на открывшейся диалоговой панели **Port Properties** установить переключатель в положение **Registered** (вместо **Combinatorial**).

Сравнивая сгенерированный **VHDL**-код двух последних примеров, легко заметить, что в первом случае (рис. 13) назначение выходному сигналу выполняется за пределами автоматного тела **process**. Для этого используется специальный оператор условного назначения, который помещается в самом конце текста (рис. 15).

Напомним, что вне процесса переменная **Q_INT** может быть использована только в том случае, если она объявлена как глобальная, то есть с установленным флажком **shared** на диалоговой панели **Code Generation Settings**.

В последней реализации автомата (рис. 14), когда выходной порт объявлен как регистровый (**Registered**), операторы действия встроены в тело процесса, что не противоречит синтаксису языка **VHDL**. Текущие значения переменной **Q_INT**, определяющие состояние автомата, передаются выходному сигналу **Q** внутри процесса (рис. 16), и поэтому не требуют никаких ухищрений.

Рассмотренные примеры показывают, что переменные в принципе можно использовать для тех же целей, что и внутренние сигналы, но делать это следует с большой осторожностью. Сказанное, прежде всего, касается разделяемых (**shared**) переменных, не поддерживаемых некоторыми системами синтеза.

Продолжение следует