

Продолжение. Начало в № 3`2009

## Изучаем Active-HDL 7.1. Урок 12. Проектирование цифровых автоматов

Александр ШАЛАГИНОВ  
shalag@vt.cs.nstu.ru

Не секрет, что графические методы описания цифровой аппаратуры уступают текстовым по многим показателям и постепенно сдают им свои позиции. Однако, несмотря на это, они остаются неизменным атрибутом современных САПР.

Главное достоинство графического представления проекта — его наглядность, удобство восприятия человеком. Но для компьютера — это лишнее звено в маршруте проектирования: схему все равно приходится конвертировать в текстовый формат, например в VHDL- или Verilog-код.

Заметим, что наглядность графического изображения проявляет себя в полной мере только для простых проектов. Когда же описание превращается в толстый «Альбом схем» с многочисленными межстраничными ссылками, даже разработчики старой закалки теряют энтузиазм.

Конечно, схема в какой-то мере защищает пользователя от необходимости изучать языки описания аппаратуры (Hardware Description Language, HDL). Но противиться новым технологиям автоматизированного проектирования просто неразумно. Время, когда можно было вручную нарисовать принципиальную схему или составить карту прошивки, безвозвратно уходит.

Сказанное ни в коей мере не означает, что данный урок можно пропустить. Вероятнее всего, правильное решение, как всегда, лежит посередине: нужно овладеть всеми возможными инструментами описания аппаратуры и использовать их совместно. Например, поведенческое (языковое) описание компонентов нижнего уровня удачно сочетается с графическим представлением структуры проекта на верхнем уровне.

### Знакомство с редактором State Diagram Editor

Со схемным редактором Block Diagram Editor мы познакомились на седьмом и восьмом уроках. Теперь нам предстоит встреча еще с одним графическим редактором — State Diagram Editor (SDE). Он предназначен для создания диаграмм состояний цифровых автоматов.

Запустим Active-HDL 7.1 и создадим в рабочем пространстве Lessons новый рабочий



Рис. 1. Фрагмент закладки менеджера маршрута проектирования Design Flow


проект Lesson\_12. Активируем закладку Design Flow (рис. 1) и щелкнем мышкой по иконке FSM (аббревиатура от слов Finite State Machine — «конечный автомат»).

Появится «старый знакомый» — «мастер» New Source File Wizard. Но теперь он намерен создать asf-файл, содержанием которого будет диаграмма состояний автомата.

Следуя указаниям «мастера», выполним все, что он от нас потребует: укажем язык проектирования VHDL, имя файла, например T\_trigger («счетный триггер»), и зададим внешние сигналы (порты):

- R — асинхронный сброс;
- C — счетный вход;
- Q — прямой выход;
- NQ — инверсный выход.

На вопрос «Хотите ли вы добавить тактовый вход с именем 'CLK'?» ответим «Нет». Получив нужную информацию, «мастер» закончит свою работу и передаст управление редактору диаграмм (State Diagram Editor). В основном окне среды проектирования Active-HDL 7.1 появится новая закладка с именем открытого файла — T\_trigger.asf (рис. 2).

Нам предстоит построить элементарный автомат, описывающий работу счетного триггера. Это синхронный автомат, и «мастер» неспроста предлагал добавить тактовый вход. Но мы отказались, потому что роль тактового входа будет играть счетный вход C. Двойным щелчком по графическому обозначению порта  откроем диалоговую панель Port Properties («свойства

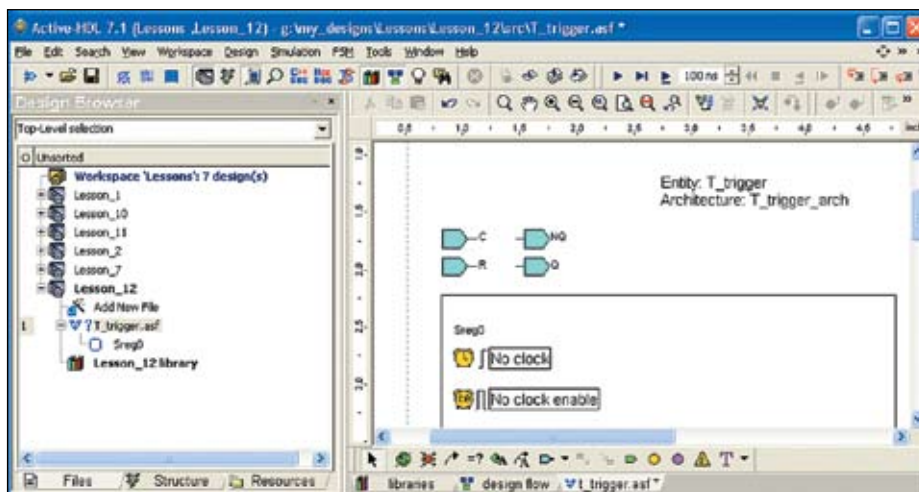


Рис. 2. В редактор State Diagram Editor загружен файл T\_trigger.asf

порта») и установим флажок **Clock** («тактовый импульс»). Сигнал С получит статус тактового входа, о чем говорят изменения в его обозначении

Если вы забудете проделать эту операцию, то при компиляции проекта в окне **Console** получите сообщение об отсутствии тактового сигнала для автомата **Sreg0 (Missing CLOCK signal for the Sreg0 machine)**. Кстати, заданное по умолчанию имя автомата **Sreg0** всегда можно изменить (например, на **T\_trigger**), щелкнув по нему правой кнопкой мыши и выбрав в контекстном меню команду **Properties**.

Как видно, у нас есть два способа изменять свойства объекта:

- напрямую — дважды щелкнув на редактируемом объекте;
- косвенно — вызвав сначала контекстное меню, а затем активизировав команду **Properties**.

Первый способ немного быстрее, второй дает большие возможности.

Для создания и редактирования диаграмм состояний цифровых автоматов нам потребуются специальные инструменты. Их можно найти в выпадающем меню **FSM**, которое появляется только в том случае, когда активна закладка редактора диаграмм **SDE**. Меню содержит полный набор возможных операций.

Однако удобнее пользоваться пиктограммами инструментальных панелей. В редакторе **SDE** их три:

- **FSM VHDL (или Verilog) Diagram Toolbar**;
- **FSM Edit Toolbar**;
- **FSM Hierarchy Toolbar**.

Сейчас нас интересует первая панель, предназначенная для рисования диаграмм состояний конечного автомата (рис. 3).

Фундаментальными понятиями в теории автоматов являются два объекта: состояние (**State**) и переход (**Transition**). Они-то и понадобятся нам в первую очередь. Наждем на пиктограмму **State** и поместим в рабочей области диаграммы (она выделена сплошной рамкой) два состояния — **S1** и **S2**. Затем с помощью кнопки **Transition** соединим их переходами (рис. 4).

Если вы хотите сразу увидеть дугу, а не прямую линию, поставьте на предполагаемой траектории хотя бы одну промежуточную точку. И наоборот, дугу всегда можно выпрямить командой **Straight Transition** («прямой переход») из контекстного меню.

В принципе наш проект уже можно моделировать. Откомпилируем файл **T\_trigger.asf**

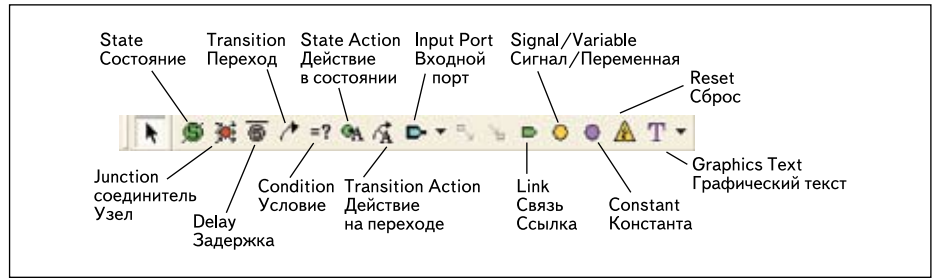


Рис. 3. Инструментальная панель SDE-редактора для проектирования диаграмм состояний цифровых автоматов

(пиктограмма или горячая клавиша **F11**), убедимся, что в окне **Console** нет ошибок (**Compile success 0 Errors 0 Warnings**), и выполним пробный модельный эксперимент (рис. 5). Обратите внимание, триггер изменяет свое состояние на каждом периоде тактового сигнала С, то есть так, как и должен работать Т-триггер.

Но почему работа началась с состояния **S1**, а переходы происходят на фронтах сигнала С? Возможно, вы уже знаете ответ. Состояние **S1** на диаграмме появилось первым. И если посмотреть **VHDL**-код (файл **T\_trigger.vhd**), автоматически сгенерированный при компиляции проекта, то мы увидим строку:

```
type T_trigger_type is (S1, S2);
```

В объявленном перечислимом типе значение **S1** занимает левую позицию, а по умолчанию именно эта позиция используется при инициализации моделирования. Поменяйте местами имена состояний **S1** и **S2** и вы увидите, что автомат начинает работу с состояния **S2** (после эксперимента восстановите все, как было).

Чтобы гарантировать старт с любого желаемого состояния, в редакторе **SDE** есть специальный символ **Reset** (рис. 3). Сброс (иначе — установка исходного состояния) может привязываться к тактовому сигналу или выполняться асинхронно . По внешнему виду их легко отличить.

Добавим объект **Reset** в наш проект, подведя переход к состоянию **S2**. По умолчанию мы получили синхронный сброс. Щелкнем по значку правой кнопкой и установим переключатель в положение **Asynchronous** ().

Зададим условие, при выполнении которого будет происходить сброс, то есть выполняться переход в исходное состояние **S2**. Мы уже знаем, что на панели инструментов (рис. 3) для этих целей имеется подходящий значок **Condition** («условие») .

Активизируем данную операцию, подведем небольшой прямоугольник , привязанный к курсору мыши, так, чтобы он пересек дугу перехода , и снова щелкнем левой кнопкой мыши.

Откроется небольшое окно, в которое введем желаемое условие перехода, например **R='1'**. Написанное условие означает, что в момент времени, когда сигнал сброса **R** примет уровень «логической единицы», автомат должен перейти в **S2**, каким бы ни было его текущее состояние.

Повторим модельный эксперимент с новой версией автомата (рис. 6).

Прежде всего, отметим, что сброс действительно выполняется асинхронно и переводит автомат в состояние **S2** (момент времени 20 нс). На третьем такте (250 нс) автомат должен был перейти в состояние **S1**, но ему «помешал» сигнал сброса, имеющий более высокий приоритет.

Чтобы «заставить» Т-триггер реагировать на срез (**Falling**) тактового сигнала С, надо

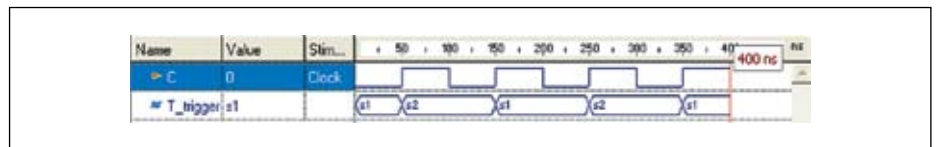


Рис. 5. Результаты моделирования «полуфабриката» Т-триггера

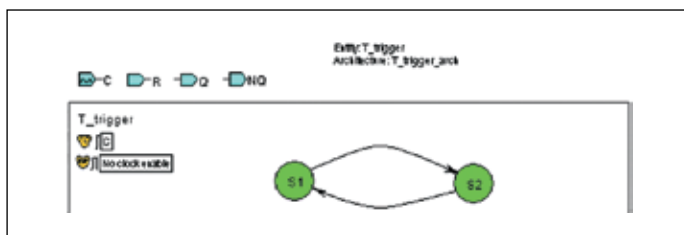


Рис. 4. Элементарный автомат Т-триггера

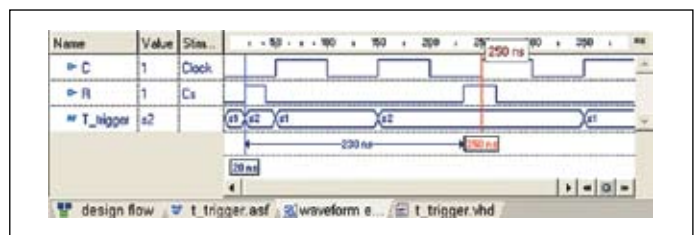


Рис. 6. Сигнал сброса R исправно выполняет свои функции

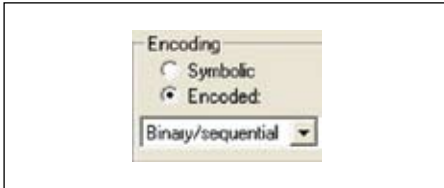


Рис. 7. Выбор значения Encoded в панели Encoding

щелкнуть мышью на свободной области автомата (внутри прямоугольной рамки), выбрать в контекстном меню опцию **Properties** и в разделе **Clock** диалоговой панели **Machine Properties** («свойства автомата») установить переключатель в положение **Falling**.

Раз уж мы оказались на панели **Machine Properties**, то сделаем еще одну вещь: в разделе **Encoding** («кодирование») заменим символическое кодирование (**Symbolic**) на альтернативное значение **Encoded** и выберем двоичное кодирование состояний автомата (**Binary/sequential**), как самое экономное (рис. 7).

Строго говоря, эта информация используется для последующего синтеза автомата, и нам она пока что не нужна. Однако заданное по умолчанию инструментальное средство синтеза **Synopsys FPGA Express** не поддерживает атрибут **ENUM\_ENCODING** в режиме символического кодирования. Поэтому каждый раз при генерации **VHDL**-кода появляется неприятное предупреждение (**Warning**): **Attribute ENUM\_ENCODING is not supported for Symbolic encoding**. Больше вы его не увидите, а вот под именем каждого состояния появится его двоичный кодовый эквивалент: /0/ и /1/.

Однако мы отвлеклись. Выходные сигналы **Q** и **NQ** до сих пор не определены. Присвоение (назначение) им конкретных значений называется действием. На панели инструментов (рис. 3) для этих целей имеются две пиктограммы: — **State Action** («действие в состоянии») и — **Transition Action** («действие на переходе»). Но какую выбрать? Оказывается, все просто: первая иконка используется для автоматов Мура, вторая — для автоматов Мили.

В схемотехнике преобладают автоматы Мура, и наш триггер не является исключением. В автоматах Мура выходные сигналы зависят только от его текущего состояния и не могут

измениться, если состояние остается прежним. Значит, действие следует «вложить» в состояние.

Нажмем на кнопку **State Action**, на курсоре мыши «повиснет» специальный значок , левым краем которого надо указать на редактируемое состояние. Особенно целиться не имеет смысла: действие в любом случае окажется привязанным к правой крайней точке кружка, изображающего состояние. Подведем курсор к состоянию **S1** и зафиксируем позицию левой кнопкой мыши. В появившееся окно введем текст: **Q<='0'; NQ<='1'**; и снова щелкнем мышью на свободной области. Напомним, что символ **<=** в языке **VHDL** называется назначением (**assignment**) или присвоением сигналу конкретного значения.

Другой способ, на наш взгляд, более удобный, задает действия с помощью диалоговой панели **State Properties**. Мы уже знаем, что панель свойств любого элемента диаграммы вызывается двойным щелчком левой кнопкой мыши или одинарным правой с последующим выполнением команды **Properties**.

Зададим действия для состояния **S2**. Дважды щелкнем на нем, активизируем закладку **Actions** и в среднее поле **State** введем нужный текст: **Q<='1'; NQ<='0'**. Обратите внимание, действия можно задавать не только в состоянии, но и на входе (**Entry Action**) и выходе (**Exit Action**) из него.

Законченный проект будет выглядеть так, как показано на рис. 8 (сброс перенесен в состояние **S1**), и функционировать он должен в соответствии с рис. 9.

Вам не нравится, что первый сброс не произвел никакого действия? Тогда выполните в выпадающем меню **FSM** команду **View/Sort Objects** и на закладке **State** отбуксируйте состояние **S2** на первую позицию. Теперь перечислимый тип будет начинаться с состояния **S2**:

```
type T_trigger_type is (S2, S1);
```

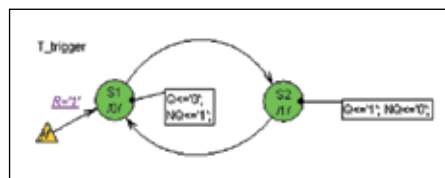


Рис. 8. Цифровой автомат, реализующий функцию T-триггера

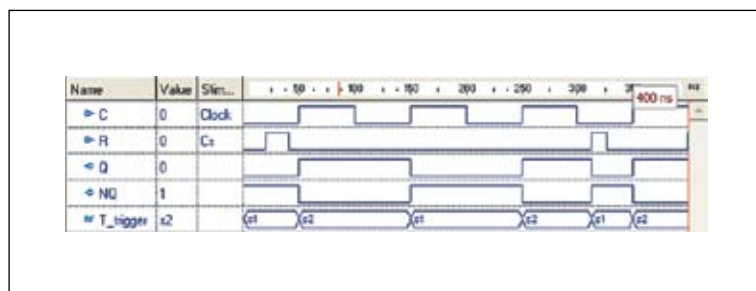


Рис. 9. Результаты моделирования элементарного автомата

По умолчанию оно и будет приниматься за исходное. При этом двоичные коды состояний тоже изменятся на противоположные, но не ранее, чем вы выполните повторную компиляцию.

## Построение асинхронных цифровых автоматов

Графический редактор **SDE** позволяет строить не только синхронные, но и асинхронные автоматы. Мы рассмотрим эту возможность на простейшем **RS**-триггере, часто называемом защелкой (**latch**).

Создадим новый **asf**-файл в том же самом рабочем проекте **Lesson\_12**. Назовем его **test\_RS\_latch**. Введем входные **S**, **R** и выходные **Q**, **NQ** порты.

На диалоговой панели **Machine Properties** в разделе **Machine** установим переключатель в положение **Asynchronous** («асинхронный») и в поле **Propagation Delay** («задержка распространения») введем абстрактное, но близкое к реальному значение задержки, например 10 нс. Заменим предлагаемое по умолчанию имя автомата **Sreg0** на **RS\_latch**.

Нарисуем диаграмму состояний, как показано на рис. 10. **S1** и **S2** — это рабочие состояния триггера. Состояние **Ban** («запрет») добавлено в диаграмму для того, чтобы «отлавливать» запрещенную комбинацию входных сигналов: **S='1' и R='1'**. Обратите внимание на косую штриховку символа. В редакторе **SDE** таким способом помечается особое состояние, называемое состоянием по умолчанию (**Default State**).

Автомат переходит в состояние по умолчанию во всех случаях, когда ни одно из условий, назначенных на выходящие из текущего состояния переходы, не выполняется. В нашем примере таким способом имитируется реакция автомата на запрещенный набор сигналов: **S='1' и R='1'**.

Чтобы состояние получило статус **Default State**, дважды щелкните на нем левой кнопкой мыши и установите флажок **Default** на закладке **General**.

Состояние по умолчанию особенно полезно при отладке цифровых автоматов, когда у вас остаются сомнения относительно полноты его переходов.

Обратите внимание еще на одну вещь. Символ **Reset** , используемый в синхрон-

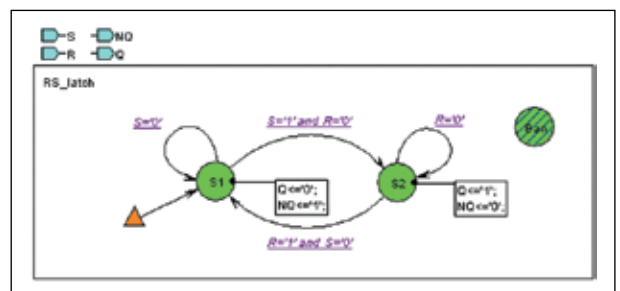


Рис. 10. Цифровой автомат асинхронного RS триггера-защелки

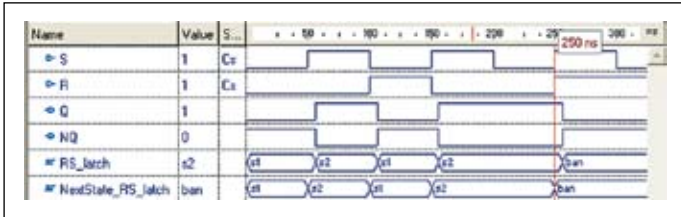


Рис. 11. Результаты моделирования RS триггера-защелки

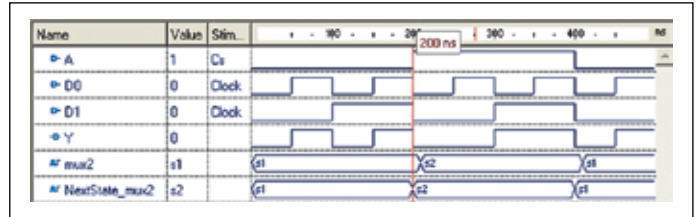


Рис. 14. Результаты моделирования асинхронного цифрового автомата mux2

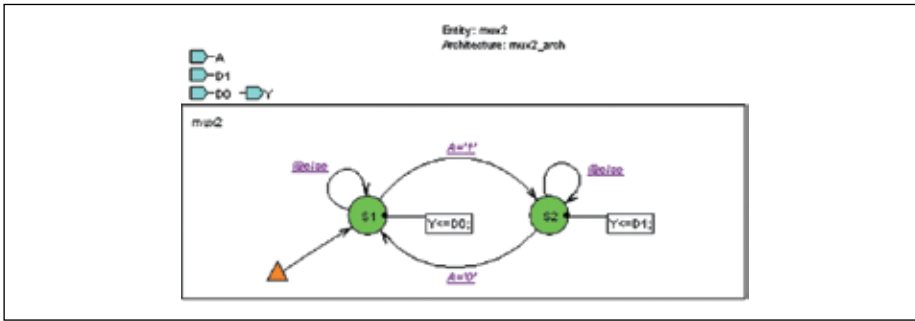


Рис. 12. Мультиплексор mux2, описанный в терминах цифрового автомата

```

-- ...
case mux2 is
  when S1 =>
    Y<=D0;
    if A='1' then
      NextState_mux2 <= S2;
    else -- красный цвет
      NextState_mux2 <= S1;
    end if;
end case;
    
```

Рис. 13. Конструкция @else диаграммы состояний замещается в условном операторе VHDL-кода предложением else

ных автоматах, заменен здесь другим объектом ▲, который называется индикатором инициализированного состояния (**Initial State Indicator**). Назначение у него прежнее — указать начальное состояние, в котором автомат должен перейти в момент его инициализации.

Но есть и разница. В синхронных автоматах вы можете многократно возвращать схему в исходное состояние на интервале одного модельного эксперимента. В асинхронных автоматах такой возможности нет. Поэтому на дуге, соединяющей индикатор с инициализируемым состоянием, не задается никакого условия перехода.

Добавим, что SDE-редактор автоматически заменяет символ **Reset** ▲ на символ **Initial State Indicator** ▲, как только вы захотите сменить тип синхронного автомата на асинхронный. При этом будет удалено и потеряно условие перехода к инициализируемому состоянию.

Кроме того, если в синхронных автоматах разрешено использовать несколько символов **Reset**, то в асинхронных — всего один индикатор. Как только вы разместите на чертеже первую копию индикатора, его пиктограмма на панели инструментов (рис. 3) станет недоступной для использования.

Не стоит забывать еще про одну особенность асинхронных автоматов. Дело в том, что в режиме **One Process** («один процесс»), который установлен по умолчанию, программа конвертации не может сгенерировать шаблон для асинхронного автомата, и в окне **Console** будет выдано соответствующее сообщение:

Cannot generate code for asynchronous machine in 1 process template. Please choose 2/3 process template.

Поэтому лучше сразу открыть выпадающее меню **FSM** и выполнить команду **Code**

**Generation Setting** («установка генерации кода»). В разделе **HDL-style** надо поставить переключатель в положение **Two Processes** («два процесса») или **Three Processes** («три процесса»).

Результаты моделирования RS триггера-защелки показаны на рис. 11. В момент времени 250 нс на триггер поступает запрещенный набор сигналов, и автомат переходит в состояние **Ban**, информируя разработчика проекта о некорректном наборе входных сигналов.

Конечно, модель можно улучшить, добавив в нее запрещенное состояние и другие аномальные состояния триггера (метастабильные и колебательные). Но стоит ли овчинка выделки? Проще убедиться, что схема не попадает в такие состояния.

Асинхронные автоматы вполне годятся и для описания поведения комбинационных схем. Возможно, этот тезис звучит странно, ведь комбинационная схема не является автоматом с памятью, а ее выходные сигналы не зависят от предыдущего состояния. Тем не менее все работает.

В подтверждение этих слов рассмотрим двухвходовой мультиплексор **mux2**, работающий как асинхронный автомат (рис. 12). Процесс создания цифрового автомата **mux2** мы не станем подробно комментировать, предполагая, что он уже хорошо усвоен.

Мультиплексор может находиться в одном из двух состояний. В состоянии **S1** на выход **Y** передается сигнал с входа **D0**, в состоянии **S2** на выход коммутируется сигнал **D1**. Переход из одного состояния в другое определяется значением сигнала на селекторном входе **A**.

Условие перехода — булевское выражение, например **A='1'** или **A='0'**. Однако вместо явного выражения мы можем использовать конструкцию **@else**, которая считается ис-

тинной, если никакие другие условия для выходящих из состояния переходов не принимают значение **True** («истинно»). Другими словами, условие **@else** читается как «во всех остальных случаях».

На рис. 13 показан фрагмент сгенерированного по диаграмме состояний VHDL-кода для мультиплексора **mux2**. Конструкция **@else** представлена в условном операторе **IF** предложением **else**, выделенным красным цветом.

Рассмотренная конструкция **@else** особенно полезна при проектировании сложных автоматов, когда трудно заранее предусмотреть все возможные переходы. Временные диаграммы, показанные на рис. 14, подтверждают правильность работы мультиплексора.

Мы уже отмечали, что новое состояние автомата для комбинационных схем не зависит от предыстории, а это значит, что переход в новое состояние возможен из любого предыдущего состояния.

Возникают опасения, что для сложных схем число переходов окажется настолько большим, что сделает неэффективным автоматный способ описания комбинационных схем. Однако на деле этого не происходит, выручают так называемые глобальные переходы.

На рис. 15 показана диаграмма состояний мультиплексора, имеющего четыре информационных (**D0, D1, D2, D3**) и два селекторных (**A0, A1**) входа. Но вместо 12 обычных переходов, которыми пришлось бы соединить все возможные сочетания состояний, на рисунке всего 4 глобальных перехода.

Как видно, глобальный переход не имеет стартового состояния. Предполагается, что он может начинаться «ниоткуда», точнее из любого состояния. Другими словами, если условие перехода выполняется, то происходит переход в указанное им состояние независимо от прежнего состояния автомата.



Рис. 15. Глобальные переходы позволяют создавать очень компактные автоматные описания комбинационных схем

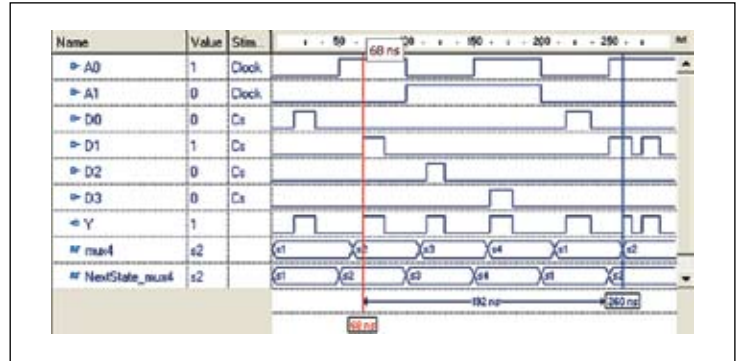


Рис. 16. Результаты моделирования мультиплексора mux4

Глобальный переход создается так. Сначала вы рисуете обычный переход. Затем щелкаете на нем правой кнопкой мыши и выполняете команду **Global**. На исходящем конце перехода появляется жирная точка — признак глобального перехода. Попробуйте отбуксировать ее на какое-нибудь состояние, и, в отличие от обычного перехода, «соединения» не произойдет.

Результаты работы мультиплексора **mux4** показаны на рис. 16. Обратите внимание, сигналы с информационных входов **D0...D3** передаются на выход без задержки (момент времени 68 нс), а смена состояний запаздывает на указанную в параметре **Propagation Delay** величину в 10 нс (момент времени 260 нс). Поэтому на выход **Y** проникает только часть сигнала с входа **D1**.

**Способы описания цифровых автоматов**

Создавая диаграммы состояний цифровых автоматов, вы можете использовать несколько стилей описания: низкоуровневое (структурное), потоковое и высокоуровневое (поведенческое). Отличаются они уровнем абстракции и числом рабочих состояний.

При низкоуровневом (структурном) описании проект представляется взаимосвязанными параллельными автоматами (рис. 17), каждый из которых имитирует работу одного или нескольких входящих в систему элементов.

На рис. 17 показана диаграмма состояний трехразрядного двоичного счетчика с параллельным переносом. Каждый из трех автоматов фактически описывает работу одного разряда счетчика (триггера и соответствующей логики формирования переносов).

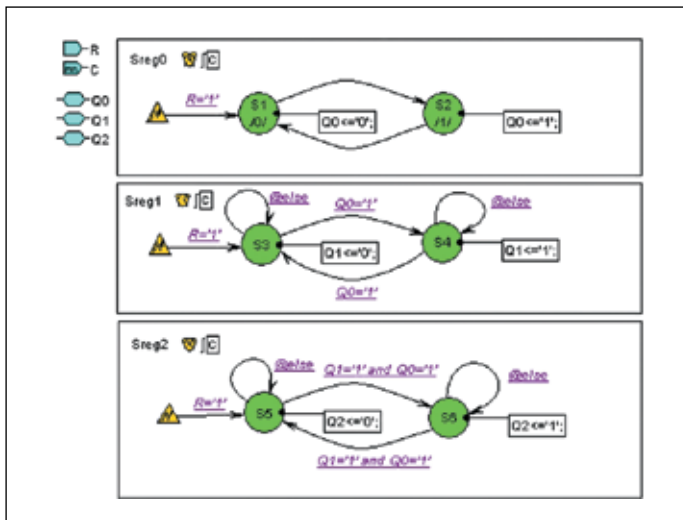


Рис. 17. Диаграмма состояний 3-разрядного двоичного счетчика, представленная тремя взаимосвязанными автоматами

Для всех трех автоматов общими являются сигнал сброса **R** и тактирования **C**. Триггер **Q1** будет переключаться только в том случае, если младший разряд счетчика **Q0** находится в '1' (см. автомат **Sreg1**, условие  $Q0=1$ ).

Для переключения старшего разряда **Q2** должно выполняться аналогичное условие — оба предыдущих разряда установлены в '1':  $Q1=1$  and  $Q0=1$ .

Обратите внимание, выходы счетчика **Q2...Q0** объявлены двунаправленными (**Inout**). Это сделано по необходимости, чтобы их можно было использовать в булевых выражениях, задающих условия на переходах.

Результаты моделирования счетчика показаны на рис. 18.

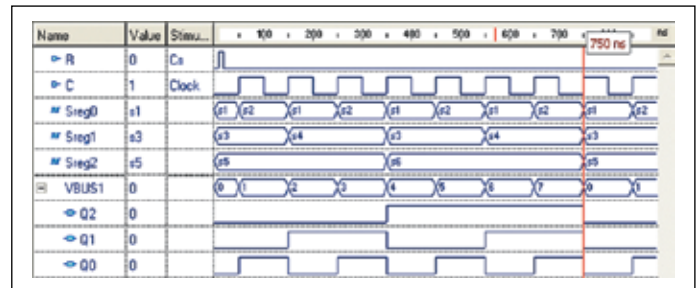


Рис. 18. Результаты моделирования двоичного суммирующего счетчика

Потоковый стиль описания реализуется с помощью одного цифрового автомата. Отличительным признаком этого стиля является полное соответствие числа состояний реального объекта и его диаграммы состояний. Например, 2-разрядный двоичный реверсивный счетчик имеет четыре состояния, столько же состояний должно быть и на его диаграмме (рис. 19).

Направление счета задается сигналом **UpDn**. При **UpDn=1** счетчик работает на сложение, и автомат последовательно проходит состояния: **S1-S2-S3-S4** (рис. 20). При **UpDn=0** счетчик работает в обратном направлении: **S4-S3-S2-S1**.

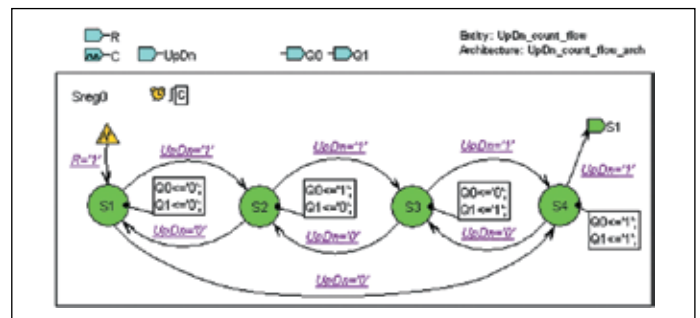


Рис. 19. Потоковый стиль описания работы 2-разрядного реверсивного счетчика

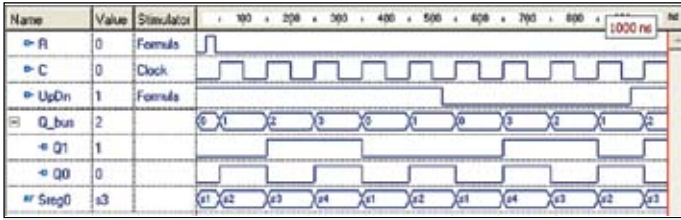



Рис. 20. Результаты моделирования реверсивного счетчика

Обратите внимание на один новый графический элемент , с которым мы встречаемся впервые (на рис. 19 он расположен в правом верхнем углу). Он называется **link** («связь») и применяется в тех случаях, когда возникают трудности с непосредственным проведением переходов.


В нашем примере при переполнении счетчика автомат должен перейти из состояния **S4** в состояние **S1**. Чтобы не тянуть длинную дугу через весь рисунок, можно создать короткий переход, заканчивающийся на объекте **link**, заменив заданное по умолчанию имя **StateName** на имя состояния **S1**, которым должен заканчиваться данный переход.

Добавим, что этот механизм работает и при подключении состояний, расположенных на различных иерархических уровнях диаграммы. Подробнее об этом мы поговорим позднее.

Потоковый стиль описания неудобен тем, что для схем с большим числом состояний диаграмма получается очень сложной. Действительно, для 3-разрядного счетчика потребуется уже 8 состояний, для 4-разрядного — 16, для 8-разрядного — 256 состояний.

Этот недостаток отсутствует в поведенческом стиле описания цифрового автомата. Показанная на рис. 21 диаграмма состояний 2-разрядного реверсивного счетчика выполнена именно в таком стиле. Она строится на основании логической таблицы, описывающей работу (поведение) проектируемого объекта. Отсюда и название — «поведенческий стиль описания автомата».

В нашем примере счетчик имеет три режима работы: сброс **Reset**, счет вверх **Up** и счет вниз **Down**. На графе переходов автомата им соответствуют три состояния с теми же самыми именами.

Исходное состояние счетчика задается асинхронно  высоким уровнем на входе **R** ( $R=1$ ). Направление счета определяется значением сигнала на входе **UpDn**. При  $UpDn=1$  счетчик работает на сложение, при  $UpDn=0$  — на вычитание (рис. 22).

Для того чтобы заставить данную реализацию правильно работать, пришлось пойти на определенные уловки. Прежде всего, это касается выходного сигнала **Q**. Его нельзя использовать в правых частях выра-

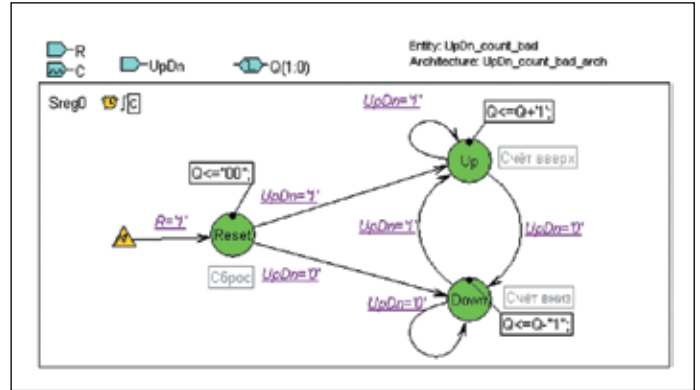


Рис. 21. Поведенческий стиль описания цифрового автомата

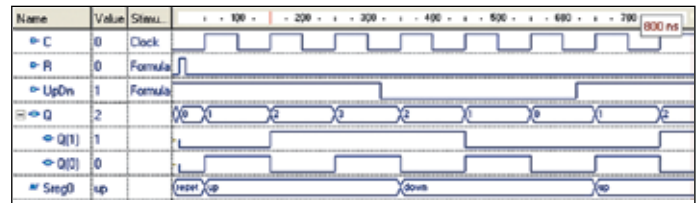



Рис. 22. Результаты моделирования реверсивного счетчика при использовании поведенческого стиля описания его диаграммы состояний

жений типа  $Q \leq Q+1$  или  $Q \leq Q-01$  (рис. 21). Поэтому пришлось заменить выходной порт **Q** на двунаправленный , что нельзя назвать изящным решением.

Кроме того, потребовалось сделать этот порт регистровым (**Registered**) и использовать команду **FSM/Action/Entry** («действия на входах») вместо рекомендуемой по умолчанию команды **FSM/Action/State** («действия в состояниях»).

Итак, для получения верного результата потребовались весьма неочевидные манипуляции. Поэтому в название объекта проекта **UpDn\_count\_bad** включено слово **bad** («плохой»). И мы не рекомендуем действовать подобным образом.

Способы построения эффективных, компактных и наглядных диаграмм состояний цифровых автоматов будут рассмотрены на следующем уроке. ■

*Продолжение следует*