Министерство образования Российской Федерации

Тамбовский государственный технический университет

И. А. Д Ь Я К О В

МОДЕЛИРОВАНИЕ ЦИФРОВЫХ И МИКРОПРОЦЕССОРНЫХ СИСТЕМ. ЯЗЫК VHDL

Утверждено Ученым советом в качестве
учебного пособия

Тамбов 2001
Издательство ТГТУ

ББК 32.97
УДК 621.396
Д92

Рецензент
кандидат технических наук, доцент
А. Е. Бояринов

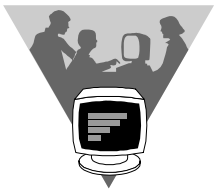
Д9 Дьяков И. А.
Моделирование цифровых и микропроцессорных систем. Язык VHDL: Учеб. пособие.Тамбов: Изд-во Тамб. гос. техн. ун-та, 2001. 68 с.

ISBN 5-8265-0071-9

В учебном пособии рассмотрены основные вопросы моделирования цифровых и микропроцессорных систем. Показаны традиционные методы моделирования и автоматизированного проектирования с использованием процедурных языков программирования и специализированного языка VHDL. Изложено описание основных компонентов языка VHDL и методов моделирования для различных уровней представления схем, от логических элементов до систем, состоящих из набора БИС. Приведены примеры схем и представление их на языке VHDL.

Предназначено для студентов дневного отделения 4 курса специальности 2203.

ББК 32.97
УДК 621.396



ВВЕДЕНИЕ

Применение в современных цифровых системах БИС и СБИС при-водит к необходимости структурирования процесса проектирования, поэ-тому особое внимание уделяется созданию соответствующих нстру-ментальных средств проектирования. Конкретным примером подобных инструментальных средств являются языки описания аппаратуры, CDL, ISP и AHPL. Однако эти языки не позволяют создать адекватную модель с учетом временных параметров. Такие языки, как, например, HHDL, ISP и VHDL, предусматривают построение универсальных моделей с временными параметрами и не ориентированы на конкретные аппаратные структуры. Преимущество этих языков состоит в том, что они докумен-тируют проекты и осуществляют моделирование. Хорошая документация необходима для обеспечения транспортабельности проекта.

Все широко распространенные языки описания аппаратуры под-держиваются соответствующими системами моделирования. Следо-вательно, макетирование систем заменяется моделированием, и процесс проектирования отличается от традиционного.

Разработчики подготавливают свои проекты при помощи рабочих станций (автоматизированных рабочих мест - АРМ), где проект вводится как исходный файл на языке описания аппаратуры в виде принципиальной электрической схемы. Параллельно проводится разработка программного обеспечения (если это необходимо) для проектируемой системы. После изготовления опытного или экспериментального образца прототипа системы проводится автономное тестирование программной и технической части проекта. На завершающем этапе проектирования производится комплексное тестирование системы с использованием внутрисхемных эмуляторов и логических анализаторов.

1 ОСНОВНЫЕ ВОПРОСЫ МОДЕЛИРОВАНИЯ ЦИФРОВЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

В системах автоматизированного проектирования цифровых вы-числительных систем необходимо предсказывать реальное поведение элементов и узлов отдельных устройств разрабатываемой системы. Для предварительной оценки поведения в большинстве случаев оказывается достаточным моделирование на логическом уровне, где оперируют с логическими

значениями "1" и "0". В некоторых случаях дополнительно рассматриваются переходные процессы и задержки сигналов для более точного прогнозирования работы схемы [1 - 5].

При логическом моделировании на ЭВМ эмулируется работа функциональной схемы по принципу прохождения информации в виде логических состояний сигналов. Процесс моделирования выполняется в три этапа: подача на вход схемы тестовой комбинации; вычисление состояний элементов схемы по соответствующим логическим моделям последовательно от входа до выхода всей схемы, анализ реакции схемы на полученное возмущение.

Для моделирования схем необходимо иметь специальное описание схемы, библиотеку математических (логических) моделей элементов, набор методов моделирования, тестовые последовательности и известные или прогнозируемые реакции схемы.

Моделирование систем отличается степенью детализации схем. В качестве элементов могут использоваться модули цифровых систем, включающие десятки БИС высокой степени интеграции, и даже целые вычислительные системы в виде офисных и промышленных компьютеров, серверов и т.д. Модели таких систем очень сложны и могут представлять собой функциональное объектно-ориентированное описание.

1.1 Логическое моделирование функциональных узлов

Описание функциональных схем. Одной из важнейших задач в моделировании является разработка и использование в процессе функ-ционально-логического проектирования языка описания функциональных схем. Язык должен быть понятным, достаточно простым и иметь удобный человеко-машинный интерфейс. Такие требования приводят к специали-зированной ориентации на различные типы схем (цифровые, аналоговые) и уровни детализации моделей (логические элементы, БИС, СБИС, программируемые логические контроллеры). Особенности языков прояв-ляются в описании переменных типа "сигнал", в указании задержки рас-пространения сигналов, а также в использовании процессов для моде-лирования параллельного прохождения сигналов в схемах. В качестве при-мера можно привести такие моделирующие системы, как ЛЕМП, ЯЛМ, МОЛК, VHDL и др. [6].

Для работы моделирующих программ необходима информация и сведения о топологии схемы, входных и выходных узлах, модели элементов, модели неисправностей элементов и различная дополнительная информация.

Ранжирование элементов функциональной схемы. Большое зна-чение при моделировании имеет порядок рассмотрения описаний схемы. При поиске значений выходов элементов необходимо, чтобы значения входов этих элементов были уже определены.

Процессу логического моделирования предшествует процедура ран-жирования описаний элементов функциональной схемы. Элементы дол-жны размещаться в таком порядке, в каком происходит последовательное переключение элементов схемы при входном воздействии $X(\tau)$. Элемент схемы имеет ранг $p + 1$, если его входы подключены к элементу, имеющему максимальный ранг p. Элементу, на входы которого поступают сигналы от внешних источников, присваивается ранг $p = 1$.

Ранжирование комбинационных схем выполняется по следующему алгоритму:

1 Определить элементы, все входы которых подключены к источ-нику внешних сигналов $X(\tau)$, и присвоить ранг $p = 1$.

2 Определить элементы, все входы или более половины входов которых подключены к выходам элементов, имеющих ранг.

3 Увеличить ранг на единицу и присвоить выбранным элементам.

4 Проверить, есть ли еще свободные, без ранга, элементы. Если есть, то осуществить переход к п. 2, иначе выполнить следующий шаг.

5 Если все элементы отранжированы и наивысший ранг имеют эле-менты, выходы которых являются выходами схемы, то завершить процесс ранжирования. В противном случае вывод сообщения "Схема не комбинационная" или "Ошибка в описании схемы".

Приведенный алгоритм может быть модифицирован и улучшен. В нем обозначены только основные шаги ранжирования и самые простые условия выбора элементов. Проверим его работоспособность на примере несложной схемы (рис. 1.1). Имеем описание связей функциональной схе-мы формирования импульса, задержанного и укороченного относительно входного.

Пусть ранг входного сигнала $X_1(\tau)$ равен нулю. Тогда получаем, что ранг элемента 1 равен единице, элементов 2 - 4 соответственно два, три и четыре. Результат очевиден и не нарушает последовательного прохож-дения сигнала.

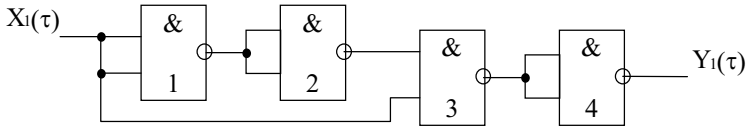


Рис. 1.1. Функциональная схема формирователя импульса

Рис. 1.1 Функциональная схема формирования импульса

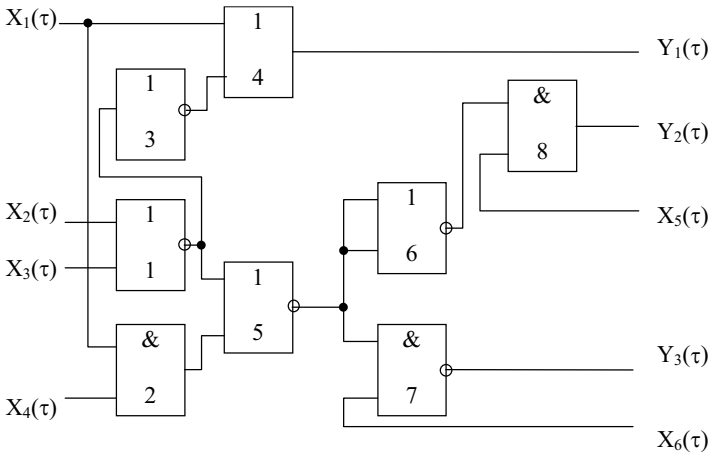


Рис. 1.2 Пример функциональной схемы

В более сложном случае (рис. 1.2) результат не столь очевиден. Выполним процедуру ранжирования. Ранг элементов 1, 2 равен 1 ($p_{1,2} = 1$). Ранг остальных элементов также определяется по вышеописанному алгоритму и равен $p_{3,5} = 2$; $p_{4,6,7} = 3$; $p_8 = 4$.

Если в схеме имеются обратные связи, то сначала ранжируют ее ком-бинационную часть описанным выше способом, а затем прибегают к условному ранжированию, выбирают из списка обратных связей одну и считают ее условно ранжированной (обратную связь условно размыкают). Если становится возможным продолжить процесс прямого ранжирования, то элемент с условно разомкнутой обратной связью получает очередной ранг и процесс ранжирования продолжается. В противном случае вы-бирают другую обратную связь, размыкают и пытаются продолжить ран-жирование. Перед выполнением ранжирования полагают, что описание связей элементов уже выполнено, т.е. определено множество $G = (G^1, G^2)$ компонентов схемы, где G^1 - множество логических элементов; G^2 - множество связей.

Таким образом, алгоритм ранжирования последовательностных схем и схем с обратными связями имеет следующий вид:

1 Определить элементы множества G^1, все входы которых подклю-чены к источникам внешних сигналов $X(\tau)$ и присвоить $p = 1$.

2 Определить элементы $g_i \in G^1$, все входы или более половины вхо-дов которых подключены к выходам элементов, имеющих ранг.

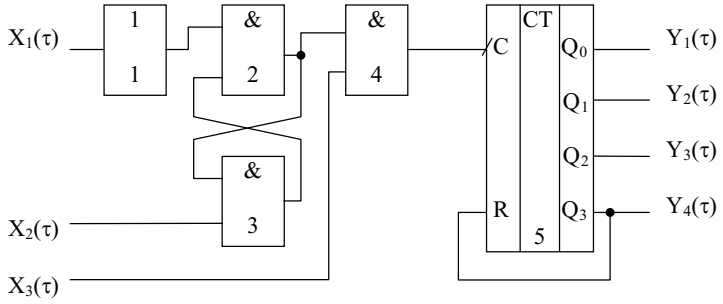


Рис. 1.3 Схема с обратными связями

Рис.1.3. Схема с обратными связями

3 Если оставшиеся входы элементов g_i подключены к выходам эле-ментов, еще не имеющих ранг, т.е. найдены связи $s_i \in G^2$, то будет переход к п. 4, иначе увеличить ранг $p = p + 1$ и присвоить элементам g_i. Временно исключить элементы g_i из списка G^1.

4 Выбрать из s_i связь s_j с наименее удаленным не ранжированным элементом, исключить ее из рассмотрения и присвоить элементу $p = p + 1$. Исключить связь s_j из s_i, а также из списка обратных связей G^2 на время обработки.

5 Если есть еще обратные связи в G^2, то обработка повторяется с п. 3.

6 Если есть еще элементы без ранга в G^1, то процесс повторяется с п. 2.

7 Если все элементы отранжированы и наивысший ранг имеют элемен-ты, выходы которых являются выходами схемы, то ранжирование завершить. В противном случае выдать сообщение "Ошибка в описании схемы".

Проведем ранжирование схемы, показанной на рис. 1.3, по опи-санному выше алгоритму. Элементы схемы будут иметь следующий ранг: $p_1 = 1$; $p_{2,3} = 2$; $p_4 = 3$; $p_5 = 4$. Ранжированные элементы заносятся в список в порядке возрастания рангов. Элементы, имеющие одинаковый ранг, могут располагаться в списке в произвольном порядке.

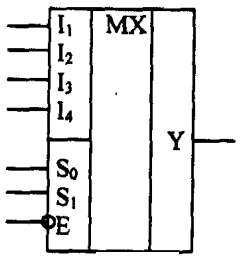
При моделировании список элементов обрабатывается в соответствии с установленными рангами. Процесс начинается с элементов, имеющих $p = 1$, и продолжается в порядке возрастания. Порядок записи элементов и последовательность обработки совпадают, что позволяет значительно экономить время.

Логическая модель. Задача логического моделирования сводится к решению логических уравнений для отдельных элементов:

$$O_i(\tau) = M_i[I_i(\tau)],$$

где M_i - логическая модель (функция); O_i - вектор логических состояний выходов; I_i - вектор логических состояний входов i-го элемента.

Например, для логического элемента 2И-НЕ логическая модель - $O(\tau) = \overline{(I_1(\tau) \& I_2(\tau))}$. Для переключателя-мультиплексора К155КП2 условное графическое обозначение и логическая функция пока-заны на рис. 1.4. В соответствии с функцией логическое значение сигнала с одного из входов I_1 - I_4 передается на выход Y. Номер коммутируемого входа определяется двоичным значением кода S_0S_1. Вход E разрешает или запрещает передачу данных.



а)

$$Y = \overline{E}(I_1\overline{S}_1\overline{S}_0 + I_2S_1S_0 + I_3S_1\overline{S}_0 + I_4S_1S_0)$$

б)

Рис. 1.4 Пример моделируемой схемы:
а - условное графическое обозначение мультиплексора; *б* - логическая функция

Логическая функция в не-которых случаях может быть задана в виде таблицы. Это может объясняться сложностью функци-онирования схемы, например, пос-тоянно запоминающего устрой-ства и невозможностью составить функцию через логические операции. Таблицы применяют для

упрощения описания работы элементов, имеющих сложные и громоздкие логические выражения. Пример таблицы показан ниже.

Входы элемента					Выходы элемента		
I_1	I_2	I_3	…	I_N	O_1	…	O_M
0	1	0	x	1	1	x	0
…	…	…	…	…	…	…	…
1	1	1	x	1	0	x	1

Таблица условно разделена на два поля для входных значений сиг-налов и выходных. В столбцах записываются логические значения соот-ветствующих входов и выходов. Каждая строка показывает соответствие входной и выходной комбинаций.

Для упрощения моделирования иногда применяют поведенческие модели схем. Такие модели не показывают принцип организации схемы на логическом уровне, а иллюстрируют поведение схемы на высоком уровне абстракции. Например, проводится моделирование накапливающего счетчика. Декомпозиция схемы приводит к триггерам и связям между ними. Каждый триггер можно представить набором элементарных логических элементов. Получим достаточно сложную схему. Но если требования к моделированию несколько ниже, то вполне достаточным может оказаться модель вида i = i + 1, это и есть поведенческая модель.

В процессе работы с моделями и функциями требуется ввести некоторые соответствия. Если на входы элементов поступает воздействие $X(\tau)$, то вектор входов $I(\tau)$ для элементов i схемы принимает значения этого воздействия $I_i(\tau) = X(\tau)$. В схеме входы элементов, подключенные к выходам других элементов, имеют значения $I_i(\tau) = O_j(\tau)$ для $i \neq j$. При-сутствие обратных связей в i-м элементе означает, что i = j:

Процесс однопроходного логического моделирования производится последовательной обработкой моделей элементов в порядке возрастания рангов. Учитывая наличие нескольких элементов в схеме и существующие связи, логические модели объединяются в систему уравнений:

$$\begin{cases} O_1^p = M_1\left[O_1^{p-1}(\Delta\tau), O_2^{p-1}(\Delta\tau), \ ..., \ O_{n-1}^{p-1}(\Delta\tau), O_n^{p-1}(\Delta\tau), \mathbf{X}(\tau)\right]; \\ O_2^p = M_2\left[O_1^p(\tau), O_2^{p-1}(\Delta\tau), \ ..., \ O_{n-1}^{p-1}(\Delta\tau), O_n^{p-1}(\Delta\tau), \mathbf{X}(\tau)\right]; \\ \qquad\qquad\qquad\qquad .... \\ O_{n\text{-}1}^p = M_{n\text{-}1}\left[O_1^p(\tau), O_2^p(\tau), \ ..., \ O_{n-1}^{p-1}(\Delta\tau), O_n^{p-1}(\Delta\tau), \mathbf{X}(\tau)\right]; \\ O_n^p = M_1\left[O_1^p(\tau), O_2^p(\tau), \ ..., \ O_{n-1}^p(\tau), O_n^{p-1}(\Delta\tau), \mathbf{X}(\tau)\right]. \end{cases}$$

Система уравнений может быть решена известными методами ре-шения систем линейных уравнений. В комбинационных схемах после од-нократного прохода, при условии правильности описания, решение будет най-дено. Схема в этом случае получит окончательное установившееся состояние. Для последовательностных схем с обратными связями решение будет найдено только за несколько проходов.

Логические модели, в зависимости от сложности описания элементов, можно разделить на несколько типов: булевы, троичные, учитывающие задержки сигналов и многозначные.

Булевы модели работают с двумя логическими состояниями "1" или "0". Основные операции с переменными - это логическое сложение (ИЛИ), умножение (И), инверсия (НЕ), исключающее ИЛИ (=1). При решении используется математический аппарат двоичной алгебры.

В троичных моделях дополнительно к двум логическим состояниям элемента используют третье состояние неопределенности. Это состояние обозначается символом Z. Троичные модели применяются при описании элементов, состояния которых в начальные моменты времени не опре-делены или оперируют с Z-состоянием.

Моделирование с учетом временных задержек элементов дает более достоверные результаты по сравнению с бинарным моделированием. При этом учитывается очередность поступления сигналов и одновременно логическое состояние этих сигналов. Такой способ моделирования дает детальную картину обработки информации и позволяет определить несоответствия, вызывающие состязания или гонки сигналов.

При многозначном моделировании, дополнительно к описанным вы-ше, учитываются статические и динамические характеристики элементов, из которых состоит логический элемент. Такое моделирование дает информацию о протекании реальных физических процессов в схеме.

Результатом моделирования цифровых систем являются временные диаграммы, которые строятся автоматически по известным уста-новившимся значениям сигналов на входах и выходах элементов. Вре-менные диаграммы в большинстве случаев строятся по двум осям. На го-ризонтальной оси размечаются временные интервалы моделирования, по вертикальной оси откладываются логические состояния сигнала. Таким образом, получается развернутое во времени изменение логических состояний, иллюстрирующих работу схемы. В некоторых случаях на диаграмме стрелками показывают фронты или спады сигналов, влияющих на поведение других сигналов. Сигналы на временных диаграммах изо-бражают в виде прямоугольников или трапеций. Это связано с необ-ходимостью показать время фронтов и спадов. Пример временной диаг-раммы работы БИС параллельного ввода-вывода в режиме 0 показан на рис. 1.5. Информация передается с шины данных микропроцессорной сис-тема на внешние устройства через порты A, B или C. В нашем примере БИС работает в режиме вывода по всем портам, хотя может иметь и дру-гие настройки. Комбинация сигналов A0 и A1 определяет порт записи, а $\overline{CS}$ определяет доступ к БИС. Непосредственно запись в порт происходит по сигналу $\overline{WR}$. На диаграмме отмечены основные временные параметры: $\tau_{\overline{WR}}$ - длительность сигнала записи, не менее 450 нс; $\tau_{A\overline{WR}}$ - время сдвига сигнала записи относительно сигналов адреса, не менее 50 нс; $\tau_{\overline{WR}A}$ - вре-мя сохранения сигналов адреса после сигнала записи, не менее 50 нс; τ_{OUT} - вре-мя выборки записи, не более 550 нс. Все эти параметры должны учи-тываться в процессе моделирования [13, 14].

Анализ временных диаграмм позволяет проверить корректность схемы, оценить эффективность функционирования, частотные свойства. Особые трудности возникают при анализе цифровых устройств на элементах с различными задержками, например, для устранения рисков сбоя, гонок и состязаний сигналов. Для устройств, выполненных на эле-ментах, имеющих одну и ту же задержку, пути прохождения сигнала в схеме могут быть различными. Это приводит к неодновременному при-ходу сигналов на входы отдельных элементов. Такой эффект называется гонками сигналов. В результате появляются всплески ложных сигналов, т.е. возникают риски сбоя. Риски сбоя могут возникнуть из-за наличия задержки при отработке входных сигналов, а также для некоторых элементов с прямыми и инверсными выходами.

Иногда схема с обратными связями может быть неработоспособна из-за состязания сигналов. Состязание сигналов - это появление кратко-временных импульсов во время переключения схемы из одного состояния в другое. Возникает это явление вследствие наложения сигналов с соизмеримыми величинами задержек.

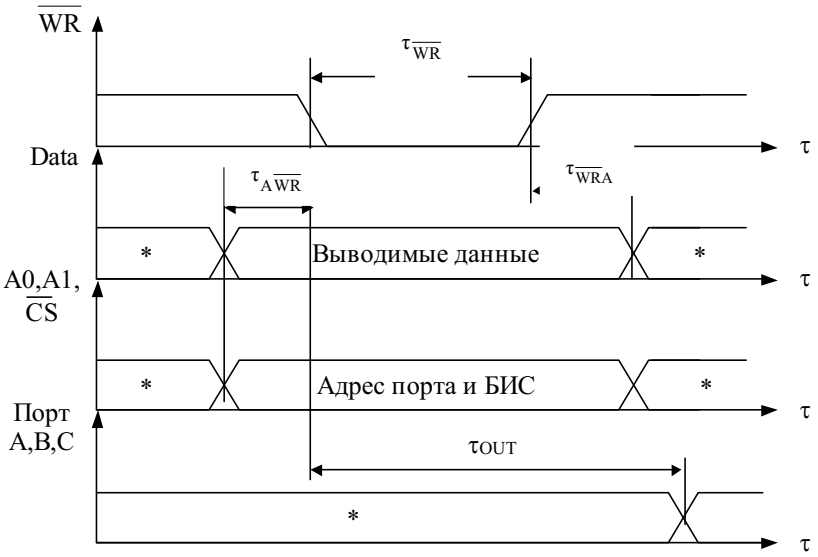


Рис. 1.5 Временная диаграмма работы БИС 8255 на вывод в режиме 0

1.2 Способы логического моделирования

В зависимости от способа организации вычислительного процесса различают два основных способа логического моделирования цифровых устройств: потактовый (синхронный и асинхронный) и событийный.

Потактовый способ моделирования. По этому способу ось времени разбивается на равные дискретные интервалы (такты). Поведение функ-циональной схемы устройства рассматривается на этой тактированной оси времени. Внутри каждого такта любой элемент схемы находится только в одном из двух состояний: нулевом или единичном. Элементы, образующие функциональную схему, заносятся в упорядоченный или произвольный список с указанием его номера и типа. Состояние каждого элемента фиксируется, как функция времени в памяти моделирующей системы.

В процессе моделирования осуществляется программное скани-рование списка в определенном порядке. В одном такте выполняется нес-колько итераций, на каждой из которых формируется вектор логических состояний элементов схемы. Каждая следующая итерация характеризует состояние схемы при измененном состоянии элементов в текущем такте.

Многократный циклический просмотр списка производится до тех пор, пока две соседние итерации не совпадут. Полученное совпадение является признаком окончания моделирования на данном такте. Затем аналогичные действия производятся на следующем такте. Моделирование завершается по исчерпании входных воздействии и временного интервала моделирования.

Потактовый способ имеет некоторые особенности, которые необхо-димо учитывать при моделировании систем. Первая особенность - это выбор длительности такта. Длительность такта влияет на общее время моделирования. Следовательно, чем меньше длительность, тем эффектив-нее работа моделирующей программы. Однако необходимое время такта должно быть по возможности наибольшим общим, кратным для всех значений временных параметров. Это зависит от учета длительностей фронтов сигналов на входах и выходах элементов схемы, времен их задержки, интервалов между изменениями входных сигналов и т.д. Таким образом, длительность такта должна быть такой, чтобы полученная временная диаграмма была корректна при возможно меньшем времени моделирования.

Вторая особенность состоит в принципе построения списка элементов и последовательности сканирования. Возможно последовательное или произвольное построение списка, но вне зависимости от этого список всегда будет просматриваться полностью. Следовательно, предпоч-

тительнее использовать последовательный принцип или такие алгоритмы, которые дают наименьшее время моделирования.

Синхронное потактовое моделирование. При таком способе моде-лирования полагают, что для всех элементов комбинационной части схемы задержки равны нулю. Задержки всех элементов памяти одинаковы и равны или кратны длительности такта работы схемы. В этом случае эму-лируется только логика работы схемы.

Моделирование осуществляется методом простых итераций по формуле

$$O^{(k)} = F[O^{(k-1)}, X(\tau)],$$

где $O^{(k)}$ - значение вектора O на k-й итерации. Если $O^{(k)} = O^{(k-1)}$, то решение найдено, иначе выполняется следующая итерация. Неустойчивое сос-тояние схемы (генерирование) или ошибки, допущенные при про-ектировании, характеризуются отсутствием сходимости итерационного процесса. Критерий выхода из итераций - большое количество повторя-ющихся тактов или превышение временного интервала моделирования.

Асинхронное потактовое моделирование. В данном способе учи-тывают, что элементы схемы имеют различное время срабатывания. Время срабатывания определяется задержкой τ прохождения сигнала с входа на выход элемента. Каждый такт работы схемы разбивается на ряд микротактов с длительностью, соответствующей задержке τ. Модели-рование производится по микротактам итерационно. Для такого модели-рования необходимо ввести два массива логических значений выходов элементов: $MO = [MO_1, MO_2, \dots, MO_n]$ - на предыдущем микротакте и $MN = [MN_1, MN_2, \dots, MN_n]$ - на последующем микротакте. Массивы имеют одинаковые размерности, где n - количество логических элементов. В процессе моделирования, по завершении каждой итерации, происходит сохранение значений массива MN_i в MO_i. На следующей итерации по ис-ходным данным массива MO_i решается система булевых уравнений и ре-зультат записывается в массив MN_i. Решение считается найденным, когда значения массивов на предыдущем и текущем микротакте совпадают, итерационный процесс заканчивается.

Асинхронный способ моделирования можно использовать как для анализа на уровне интегральных микросхем, так и на уровне дискретных элементов, входящих в состав логических элементов (диоды, транзисторы, резисторы).

Событийный способ моделирования. Основная идея состоит в вычислении уравнений только активизированных элементов, т.е. таких, у которых хотя бы на одном входе произошло событие (изменилась входная переменная).

Алгоритм, реализующий событийный способ, на каждом шаге устанавливает, какие элементы являются активизированными, и вызывает модели только этих элементов. Такой подход позволяет существенно сократить затраты машинного времени на анализ схем.

Событием считается любое изменение состояния схемы, например, переход ее из одного логического состояния в другое. Если сигналы, поданные на входы элемента, приводят к изменению его состояния, то это фиксируется как событие. Как при потактовом способе, здесь составляется список, в который заносится информация о всех элементах.

Процесс моделирования состоит в следующем. Среди всех элементов находят те, которые должны работать первыми, например, от входных сигналов. Каждый такой элемент изменением своего состояния может вызвать новые события. Это определяется существующими связями между элементами. Следует заметить, что все события можно разделить на две группы: мгновенные и события в будущем. Мгновенные события происходят без задержки во времени. События в будущем представляют собой множество мгновенных событий. Каждая реализация мгновенного события пополняет множество событий в будущем. Когда все события в множестве мгновенных реализованы, то выбирается такое событие из множества будущего, которое ближе всего по времени расположено к текущему моменту. Множество будущих событий переносится в множество мгновенных, а время моделирования приобретает значение времени этого будущего события. Этот процесс повторяется циклически.

Моделирование можно считать завершенным, если полностью реализована группа мгновенных событий и ни одно событие из множества мгновенных не вызывает события в будущем.

Алгоритм событийного асинхронного моделирования рассмотрим на примере абстрактной схемы (рис. 1.6).

Перед выполнением моделирования должны быть определены элементы $g_i \in G^1$, связи $s_i \in G^2$, векторы $\mathbf{I_i}$, $\mathbf{O_i}$ входов и выходов элементов g_i. Для каждого элемента должна существовать логическая модель или описание набора логических функций M_i.

В процессе моделирования выполняются следующие шаги.

1 На входах схемы устанавливаются начальные значения тестовой последовательности $\mathbf{X}(\tau) = \mathbf{X}(\tau_0)$.

2 Проверяется список элементов G^1 и формируется список текущих событий элементов g_i, на выходах которых произошло изменение состояния сигналов в связи с воздействием $\mathbf{X}(\tau)$.

3 Из списка текущих событий выбираются для элементов g_i соответствующие им модели M_i. По модели вычисляется реакция выходов $\mathbf{O}_i$ с задержкой $\Delta\tau$. Формируется список элементов g_j, входы которых соединены с выходами элементов g_i. Затем выбираются M_j и производятся вычис-

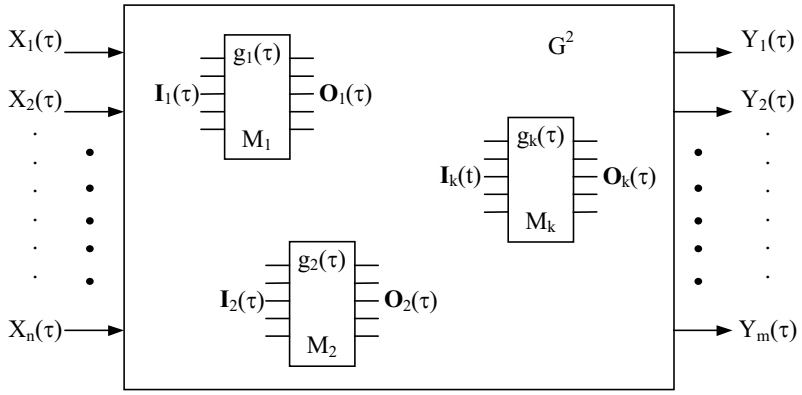


Рис. 1.6 Абстрактная схема цифровой системы

ления, определяются новые элементы g_k и т.д. до тех пор, пока перестанет выполняться условие распространения воздействия $\mathbf{X}(\tau)$. Распространение воздействия завершено, если нет больше элементов, на входах которых произошло изменение состояний.

4 После просмотра списка текущих событий время моделирования увеличивается на один такт $\tau = \tau + T$.

2 ОПИСАНИЕ ЯЗЫКА VHDL

2.1 Структура VHDL и описание разработки

Язык VHDL - это язык описания аппаратных средств. Он содержит признаки общепринятого языка программирования, классического PLD языка программирования, а также весь список общепринятых признаков для разработки административной системы.

Основное описание и организация структуры VHDL показаны на рис. 2.1 [7 - 12].

Оператор пакета является необязательным для раздела описаний. Оператор entity содержит описание входного и выходного интерфейса проекта, так как каждая микросхема имеет входы или выходы. Оператор

architecture содержит описание функционирования проекта. Проект может содержать любое количество операторов пакетов, объектов и архитектур. Из структурной схемы языка видно, что архитектура содержит параллельные операторы, такие как netlists и классические последовательные операторы. Параллельные операторы выполняются не зависимо от порядка, в

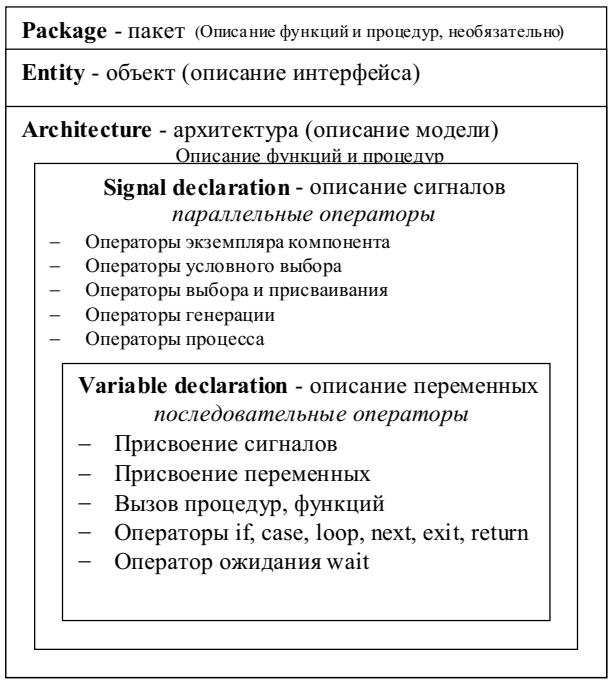


Рис. 2.1 Структура проектов на языке VHDL языке VHDL

котором они написаны. Параметры, передаваемые по значению, передаются между операторами с помощью сигналов, присвоение сигнала осуществляется драйвером. Сигнал можно сравнить физически с проводом или набором проводников.

Наиболее мощные конструкции языка встречаются внутри последовательных операторов. Такие конструкции размещаются вместо параллельных операторов, например, оператора процесса, функций или процедур. Последовательные операторы похожи на операторы языков программирования, они выполняются в том же порядке, в котором пишутся. Значения содержатся в переменных или константах. Сигналы используются для передачи входных и выходных значений процесса или от других параллельных операторов. Это очень важно для понимания концепции языка. В следующих разделах рассмотрим различные уровни абстракции на некоторых примерах, но сначала определим лексические элементы языка. Комментарии в программе (проекте) могут идти отдельной строкой или в конце строки и отделяются двумя знаками тире. Символы - от a до Z, включая стандартные спецсимволы. Строки заключаются в двойные кавычки, например, "строка". Битовые строки похожи на обычные, но в них можно указать систему счисления для данных, например, b"0101", o"05", x"5".

2.2 Описание интерфейса

В языке VHDL логическая схема представляется как объект проекта. Представляемая логическая схема может быть достаточно сложной, как, например, микропроцессор или весьма простой, как логический вентиль 2И-НЕ. Каждый объект проекта в свою очередь имеет два различных типа описаний: описание интерфейса и одно или более архитектурные тела. Проиллюстрируем эти понятия на примере. Описание интерфейса для схемы дешифратора, которая преобразует входной четырехразрядный код в напряжение низкого уровня на одном из десяти выходов.

```
entity Id_1 is
port(A: in bit_vector(0 to 3); Y: out bit_vector(0 to 9));
end Id_1;
```

Как видно, в описании интерфейса приводится имя объекта и характеризуются его входы и выходы. Для сигналов в описании интерфейса указывается вид сигнала (например, входной или выходной) и его тип. В данном случае bit_vector(0 to 3) и bit_vector(0 to 9) - это битовые векторы длиной 4 и 10 бит.

В описании интерфейса может содержаться также информация относительно природы объекта, необходимая для документирования. Например, можно переписать указанное описание интерфейса и включить в него в качестве комментария таблицу истинности объекта:

```
entity Id_1 is
```

```
port(A: in bit_vector(0 to 3); Y: out bit_vector(0 to 9));
------Таблица истинности
-----------------------------------------------------------------------------------
------A3   A2  A1  A0      Номер выхода с активным низким уровнем
------ 0    0   0   0              0
------ 0    0   0   1              1
------ 0    0   1   0              2
------ 0    0   1   1              3
------ 0    1   0   0              4
------ 0    1   0   1              5
------ 0    1   1   0              6
------ 0    1   1   1              7
------ 1    0   0   0              8
------ 1    0   0   1              9
end Id_1
```

Отметим, что здесь в описании интерфейса таблица истинности вставлена как комментарий, так как любая строка, начинающаяся с двух знаков тире, интерпретируется как комментарий.

2.3 Архитектурные тела

Описание интерфейса по существу определяет только входы и выходы объекта проекта. Кроме этого необходимо иметь средства описания поведения объекта. В языке VHDL для этой цели служит так называемое архитектурное тело. Это тело может определять поведение объекта непосредственно (т.е. быть телом-примитивом) или представлять структурную декомпозицию на более простые компоненты. Приступая к процессу проектирования, разработчики обычно уже имеют алгоритм или принципиальную схему, которые необходимо реализовать. Вначале, однако, им хотелось бы проверить их правильность.

Первое архитектурное тело, которое будет описано - чисто поведенческое тело. Ниже показано подобное поведенческое архитектурное тело для дешифратора Id_1:

```
architecture DC of  Id_1 is
begin
   process(A)
   begin     Y<= "1111111111";
      case A(0 to 3) is
           when "0000" => Y  <= "0111111111";
           when "1000" => Y  <= "1011111111";
           when "0100" => Y  <= "1101111111";
           when "1100" => Y  <= "1110111111";
           when "0010" => Y  <= "1111011111";
           when "1010" => Y  <= "1111101111";
           when "0110" => Y  <= "1111110111";
           when "1110" => Y  <= "1111111011";
           when "0001" => Y  <= "1111111101";
           when "1001" => Y  <= "1111111110";
    end case;
  end process;
 end DC;
```

В архитектурном теле отсутствует программный цикл, осуществляющий сканирование входов от A(0) до A(3), вместо этого применен оператор выбора case, в котором управляющее выражение - битовый вектор. В зависимости от значений входного вектора A, определяется значение выходного вектора Y. Нумерация разрядов в векторе осуществляется слева направо.

Это поведенческое тело идеально описывает работу алгоритма, однако практически не отражает построения и функционирования реальной аппаратуры, так как не отвечает на вопросы: какой логике соответствует конструкция цикла и какова величина задержки для описанной схемы?

Продолжая рассмотрение данного примера, можно установить логические функции для элементов выходного вектора:

$$Y0 = \overline{A0} \cdot \overline{A1} \cdot \overline{A2} \cdot \overline{A3};\ Y1 = A0 \cdot \overline{A1} \cdot \overline{A2} \cdot \overline{A3};\ Y2 = \overline{A0} \cdot A1 \cdot \overline{A2} \cdot \overline{A3};$$
$$Y3 = A0 \cdot A1 \cdot \overline{A2} \cdot \overline{A3};\ Y4 = \overline{A0} \cdot \overline{A1} \cdot A2 \cdot \overline{A3};\ Y5 = A0 \cdot \overline{A1} \cdot A2 \cdot \overline{A3};$$
$$Y6 = \overline{A0} \cdot A1 \cdot A2 \cdot \overline{A3};\ Y7 = A0 \cdot A1 \cdot A2 \cdot \overline{A3};\ Y8 = \overline{A0} \cdot \overline{A1} \cdot \overline{A2} \cdot A3;$$
$$Y0 = A0 \cdot \overline{A1} \cdot \overline{A2} \cdot A3.$$

Можно заменить чисто поведенческое тело на архитектурное:

```
architecture Level_Mech of Id_1 is
  begin
     Y(0)<= not A(0) and not A(1) and not A(2) and not A(3);
     Y(1)<= A(0) and not A(1) and not A(2) and not A(3);
     Y(2)<= not A(0) and A(1) and not A(2) and not A(3);
     Y(3)<= A(0) and A(1) and not A(2) and not A(3);
     Y(4)<= not A(0) and not A(1) and A(2) and not A(3);
     Y(5)<= A(0) and not A(1) and A(2) and not A(3);
     Y(6)<= not A(0) and A(1) and A(2) and not A(3);
     Y(7)<= A(0) and A(1) and A(2) and not A(3);
     Y(8)<= not A(0) and not A(1) and not A(2) and A(3);
     Y(9)<= A(0) and not A(1) and not A(2) and A(3);
   end Level_Mech;
```

Многоуровневый "механизм" представления логики подразумевает использование стандартной структуры и применение логических элементов таких как, И, ИЛИ, НЕ, исключающее ИЛИ, что ближе к реальным схемам.

2.4 Операторы блоков

Базовым элементом описания на языке VHDL служит блок, т.е. ограниченный фрагмент текста, содержащий раздел описания и исполняемый раздел. Следовательно, архитектурное тело представляет собой блок. Однако в рамках архитектурного тела могут существовать внутренние блоки. Рассмотрим пример. Блоки A и B вложены во внешний блок архитектурного тела. Возможно любое число уровней вложенности, например, блоки A и B также могут быть разбиты на подблоки. Использование такой структуры позволяет иерархически представить структуру объекта и записать условия "защиты". Если условие будет иметь значение "истина", это разрешит выполнение определенных типов внутренних операторов блока. Охранные конструкции полезны для моделирования последовательностной логики.

Наличие метки перед оператором блока обязательно, в дальнейшем ее можно рассматривать как имя блока. Имя метки, например, L01 заканчивается двоеточием и может стоять перед оператором блока или на предыдущей строке. Главное, чтобы между меткой и оператором блока не было других операторов.

```
architecture Block_Structured  of  System is
  ----------------   Раздел описаний типов внешнего блока
  begin
     -----------     Выполнимые операторы внешнего блока
  A: block
    -----------     Раздел описаний типов  внутреннего блока A
  begin
    -----------     Выполнимые операторы внутреннего блока A
  end block A;
  B: block
    -----------     Раздел описаний типов внутреннего блока B
     begin
   -----------     Выполнимые операторы внутреннего блока B
  end block B;
end Block_Structured;
```

Оператор блока может иметь "охранное" выражение логического типа. Оператор охраняемой конструкции будет выполняться, когда охранное выражение есть истина и сигнал правой части охраняемого оператора меняется или меняется значение охранного выражения из ложного в истинное.

```
A: block (Clock='1' or Reset='1')
    begin
Y<= guarded '0' when Reset = '1';   -- Reset=1
    else X when Clock='1';  -- Reset=0 и Clock=1
    else Y;
end block A;
```

Когда охранное выражение есть ложь, сигнал, указанный в левой части, сохраняет свое прежнее значение. Для приведенного примера при Reset = 0 и Clock = 1 выход Y следует за входом X. Когда синхросигнал Clock переключится в 0, то последнее значение выхода Y зафиксируется. Сброс выхода в 0 произойдет при условии, что Reset = 1. Таким образом смоделировано поведение триггера-защелки.

2.5 Процессы

Наиболее значимым элементом для моделирования на языке VHDL является оператор процесса. Оператор содержит последовательные операторы и позволяет разработчику описывать схемы на поведенческом уровне абстракции. Например:

```
process (Insig)
   variable v1: Integer;     -- Описание переменных
begin
  v1 := Insig;                -- Присвоение переменной
  v1 := function_name(v1+1);   -- Вызов функции
end process;
```

При проектировании аппаратных средств оператор процесса применяется в двух случаях: для комбинационных и последовательностных схем. Для комбинационных схем оператор выглядит так:

```
process (signal1, signal2, …, signalN)
begin
  …
end process;
```

а для последовательностной несколько иначе:

```
process ( X_signal)
begin
  if X_signal and X_signal'event then
   …
  end if;
end process;
```

Для комбинационных схем определяется весь список вводимых в процесс сигналов. Это так называемый список чувствительности или сигналов запуска процесса. Если какой-либо сигнал из этого списка меняет свое значение, то процесс активизируется и выполняются операторы блока этого процесса.

Для последовательностных схем есть список чувствительности, вклю-чающий в себя операторы времени и условия запуска с атрибутами. Если такого списка нет, то должен быть оператор времени wait. В языке VHDL требование одновременного выполнения действий элементами схемы реализуется при помощи механизма процессов. Каждый процесс представляет некоторый логический блок моделируемой схемы, причем все процессы выполняются параллельно. Если система моделирования работает на од- ном процессоре, то, естественно, фактически они будут выполняться последовательно, однако, с точки зрения моделирования это будет выглядеть так, как если бы они выполнялись параллельно.

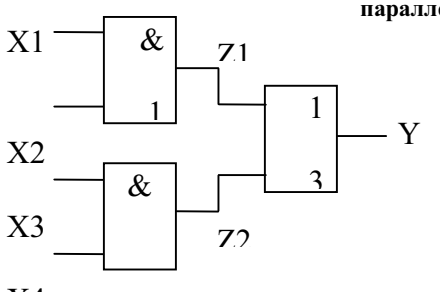


Рис. 2.2 Пример логической схемы

Пусть схема состоит из трех логических блоков (рис. 2.2). Если предположить, что входные наборы X1, X2 элемента 1 и X3, X4 элемента 2

активизируются одновременно, то логические элементы (блоки) также должны срабатывать одновременно. Логический блок 3 станет активным, как только изменяются состояния или на выходе элемента 1 (Z1), или на выходе элемента 2 (Z2). Следовательно, поток сигналов может проходить через все блоки одновременно и также должен обрабатываться.

В качестве конкретного примера укажем, что логические блоки могут быть представлены процессами. Здесь тело каждого процесса содержит вначале раздел описаний, где объявляется переменная локальная для данного процесса. Выполняемый раздел между ключевыми словами begin и end содержит оператор присваивания переменной, который вычисляет промежуточное значение Zi и за которым следует оператор назначения сигнала, учитывающий задержку распространения сигнала в блоке:

```
Lb1:  process(X1,X2)
variable Zi : bit;
begin
    Zi := X1 and X2;
    Z1<= Zi after 20ns;
end process Lb1;
Lb2:  process(X3,X4)
variable Zi : bit;
begin
     Zi := X3 and X4;
     Z2<= Zi after 20ns;
end process Lb2;
Lb3:  process(Z1,Z2)
variable Zi : bit;
begin
    Zi := Z1 or Z2;
    Y<= Zi after 20ns;
end process Lb3;
```

Концепция процесса может быть также реализована в базовых операторах регистровых передач. Рассмотрим следующие два оператора регистровых передач:

$$Y1 = X1 * X2;$$

$$Y2 = Y1 + X3.$$

Так как эти операторы записываются последовательно, то естественно сказать, что и выполняться они будут последовательно. Значение Y1 используется как входное для оператора Y2. Новое значение Y1 представляет собой фактический параметр для Y2.

Однако операторы Y1 и Y2 можно рассматривать с точки зрения процессов. Список сигналов запуска для Y1 содержит X1 и X2, а для Y2 - Y1 и X3.

Поскольку процесс представляет физическую систему, время его выполнения не может быть нулевым. Пример на рис. 2.2 показывает, каким образом процессу может быть присвоено значение задержки распространения. При этом, если такое значение не указано, считают, что процесс выполняется за время $\Delta\tau$, бесконечно малое, но большее нуля ($\Delta\tau \rightarrow 0$). Предположим, что значения X1 и X3 меняются одновременно в момент времени τ. При этом активизируются оба процесса. Значение Y1 относится к моменту времени τ и определяется значениями X1, X2, известными в момент времени $\tau - \Delta\tau$. Следовательно, новое значение Y1 при активизации процесса по X1 станет известным в момент времени $\tau + \Delta\tau$. Как только изменится значение Y1, то активизируется процесс для Y2. Однако процесс для Y2 уже был активен по изменению X3, но значение Y1 было не известно.

В табл. 2.1 и 2.2 приведены примеры параллельного и последовательного выполнения операторов. В примере будем считать, что действия выполняются над целыми числами в десятичном виде.

Таблица 2.1

Обозначение	Значения операторов в заданные моменты времени

операторов	τ - Δτ	τ	τ + Δτ	τ + 2Δτ
X1	1	4	6	7
X2	2	2	2	2
Y1	x	2	8	12
X3	3	3	5	5
Y2	x	x	5	13
x - обозначает неопределенное состояние				

Таблица 2.2

Обозначение операторов	Значения операторов в заданные моменты времени			
	τ - Δτ	τ	τ + Δτ	τ + 2Δτ
X1	1	4	6	7
X2	2	2	2	2
Y1	2	8	12	14
X3	3	3	5	5
Y2	5	11	17	19

В первом примере (табл. 2.1) действия выполняются с учетом времени Δτ. Значение оператора в какой-то момент времени может быть неопределенным. При выполнении действий происходит переход значений из одного столбца таблицы в следующий. Во втором примере (табл. 2.2) неопределенных значений нет. Действие выполняется как бы одновременно по всему столбцу сверху вниз.

2.6 Типы данных

Во всех языках программирования уделяется большое внимание типам данных и операциям, производимым над этими типами. Традиционные типы данных: целый, вещественный, символьный, логический - используются и в языке VHDL. Но поскольку этот язык используется для представления аппаратных проектов в самых разных вариантах, средства типизации данных приобретают здесь особенно важное значение. Например, они дают разработчику возможность представлять группу линий шины как массив битов, целое число или мнемонический код. В языке VHDL реализована строгая типизация; это означает, что непредумышленное смешение различных типов в одной операции будет восприниматься как ошибка. Средства строгого контроля типов дают возможность уточнять намерения разработчика.

Определим тип как поименованное множество значений с некоторыми общими характеристиками. Подтип - это подмножество значений данного типа.

Типы данных удобнее классифицировать в виде схемы, показанной на рис. 2.3. Типы данных делятся на скалярные (одномерные) и составные (многомерные).

Скалярный включает перечислимый, числовой и физический типы. В перечислимом типе просто перечисляются все члены. Например, для

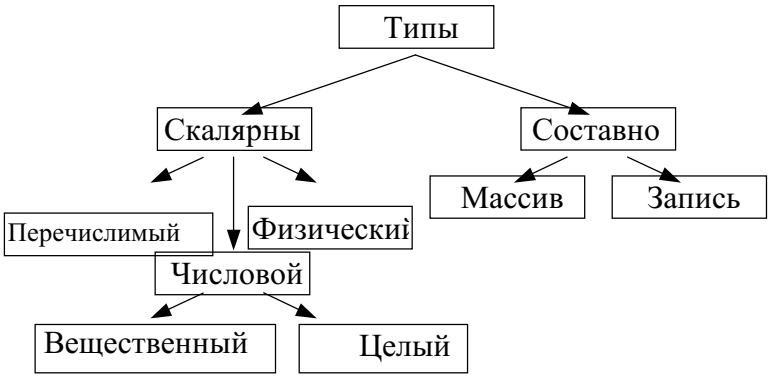


Рис. 2.3 Схема классификации типов данных

двоичной системы членов типа всего два - 0 и 1, для трехзначной логики три - 0, 1, Z (состояние неопределенности). Чтобы задать перечислимый тип, надо записать type BinL is (′0′, ′1′) для десятичной логики type DecL is (′0′, ′1′, ′2′, ′3′, ′4′, ′5′, ′6′, ′7′, ′8′, ′9′). Перечисление производится слева направо, от младшего бита к старшему. В языке Паскаль аналогично определяется множественный тип. Boolean, Bit, Character в языке VHDL - это перечислимые типы. Тип Boolean состоит из значений True (истина) и False (ложь). Все операторы if (если) в языке должны проверять объекты или выражения этого типа. Тип bit состоит из значений 0 и 1.

Числовой тип - либо целый (Integer), либо вещественный (Real). Они являются стандартными, как и в других языках программирования. Тип Real очень полезен для высокоуровневых поведенческих описаний, например, аналого-цифрового интерфейса или алгоритма обработки сигналов.

Натуральные числа (т.е. все неотрицательные целые числа) составляют подтип Natural типа Integer. Диапазон подтипов ограничен; например, члены подтипа Natural не могут быть отрицательными целыми. Подтип может наследовать или не наследовать все алгебраические свойства родительского типа. Например, для подтипа Natural операции сложения и умножения определены, однако, закон симметрии или обращения, не действует. Подтип Positive включает все целые, которые больше нуля.

Описание целого и вещественного типов с заданным диапазоном изменения выглядит, например, так:

```
type Index is range 0 to 9;    -- целый тип
type VOLTAGE is range 0.0 to 10.0  -- вещественный тип.
```

Базовый тип, целый или вещественный, неявно определяется значениями диапазона. Вещественный тип представляется значениями с плавающей запятой.

Как наиболее распространенные физические типы в языках описания аппаратуры используются напряжение, сила тока, электрическая мощность, частота.

Для языка VHDL стандартным физическим типом является тип Time:

```
type Time is range 0 to 1E20
units
   fs;
   ps = 1000 fs;
   ns = 1000 ps;
   us = 1000 ns;
   ms = 1000 us;
   s = 1000 ms;
   min = 60 s;
   hr = 60 min;
end units;
```

В качестве базовой единицы выбрана фемтосекунда (10^{-15} с) и взят диапазон до 27,7 ч. Базовую единицу времени и диапазон может выбрать пользователь, однако, диапазон ограничивается длиной слова инстру-ментальной машины.

Составные типы - это массив (индексируемый тип) и запись (структурный тип). Тип bit_vector есть массив, который неявно описывается следующим образом:

```
type bit_vector is array (Natural range < >) of  bit;
```

Bit - это базовый тип элементов массива. Выражение "Natural range" - стандартный способ указать, что длина массива будет задаваться диапазоном натуральных чисел. Пользователь должен задавать конкретный диапазон при применении такого типа, например, bit_vector (0 to 3), возрастающий диапазон, или bit_vector (7 downto 0), убывающий диапазон. Возможны массивы с другими базовыми типами.

Запись - составной тип, состоящий из ряда полей. Например, IMS можно описать как запись следующим образом:

```
type IMS is  record
```

```
Num : Integer range 1000 to 9900;
Chip: Integer range 8 to 200;
Func: Func_IMS;
end record;
```

Здесь Func_IMS - перечислимый тип, содержащий набор функций для каждой микросхемы. Тип Num определяет порядковый номер микросхемы, а Chip определяет корпус и количество выводов.

Язык VHDL относится к языкам со строгим контролем типов. Поэтому компилятор будет проверять не только синтаксис исходного текста, но и квалифицировать попытки смешения различных типов (подтипов) как ошибку. Например, если определить следующие подтипы и объявить переменные:

```
subtype Address is bit_vector (0 to 15);
subtype Data is bit_vector (0 to 7);
variable Port16: Address;
variable Port8 : Data;
```

то выполнение операторов Port16:= Port8; или Port8:= Port16; будет неверным, так как Port16 и Port8 относятся к различным подтипам. Можно определить короткий тип, как целое число от 0 до 255. Но тогда надо знать, что результаты операций над разными подтипами будут соответствовать типу объекта и возможно усечение или дополнение нулями:

```
subtype Shorter is short range 0 to 31;
subtype Shortest is short range 0 to 15;
signal X1,X2,Y1 : Shortest;
signal Y2 : Shorter;
signal Y3 : short
  begin
  Y1 < = X1 + X2; --сумма будет усечена
  Y2 < = X1 + X2; --результат не имеет усечения
  Y3 < = X1 + X2; --результат имеет дополнение нулями.
```

2.7 Операции

Язык VHDL имеет полный набор операций для работы с предусмотренными типами данных. Перечень операций приведен в табл. 2.3.

Логические операции определены для типов bit, Boolean и одномерных массивов, в которых каждый элемент имеет тип bit или Boolean; таким образом, они определены и для типа bit_vector.

Все операции сравнения возвращают результат типа Boolean, т.е. значение True или False. Все операции сравнения имеют левые и правые операнды, например, A /= B. Обе операции контроля эквивалентности (=, /=) в качестве левого и правого операндов могут иметь любой, но один и тот же тип. Если тип составной (например, массив), проверка выполняется по его элементам. Оба операнда составного типа считаются равными, если равны все их соответствующие элементы.

Таблица 2.3

Классы операций	Члены класса
Логические	not (НЕ); and (И); or (ИЛИ); nand (И-НЕ); nor (ИЛИ-НЕ); xor (исключающее ИЛИ)
Сравнения	=; /=; <; >; <=; >=
Сложения	+; -; & (конкатенация)
Присвоения знака	+; -
Умножение	*; /; mod (по модулю); rem (остаток)
Смешанные	** (возведение в степень); abs (абсолютное значение)

Операции отношения (<, < =, >, > =) определены для любого скалярного типа; например, перечислимый тип

type MVL **is** ('0', '1', 'Z')

подразумевает упорядочение слева направо, так что имеет место отношение '1' < 'Z'. Операции отношения определяются также для одномерных массивов, в которых каждый элемент - это дискретный тип. Предположим, например, что мы описали трехэлементный многозначный тип массив следующим образом:

type MVL **is** ('0','1','Z');

type THREE_bit_MVL **is array** (0 to 2) **of** MVL;

В этом случае операции отношения могут выполняться над трехэлементными векторами. Сравнение начинается с минимального индексного значения и продолжается до максимального. Так, например, значение выражения (′0′, ′1′, ′1′) < = (′0′, ′Z′, ′1′) есть истина.

Арифметические операции класса сложения (+, -) определены для типов Integer и Real. Для типа bit_vector они не определены, так что пользователь должен сам предусматривать подпрограммы для этой цели. Операция конкатенации (&) выполняет свою обычную функцию, т.е. объединяет два одномерных массива с образованием единого одномерного, длина которого есть сумма длин обоих массивов-операндов. Так, например:

(′0′, ′1′, ′1′) & (′0′, ′Z′, ′1′) = (′0′, ′1′, ′1′, ′0′, ′Z′, ′1′).

Операции присвоения знака (+, -) - это унарные операции, при помощи которых перед одиночными объектами типа Integer или Real ставится арифметический знак, обозначающий положительное или отрицательное значение. Для битовых векторов формирование дополнительного или обратного кода должно производиться специальными подпрограммами пользователя. Для типов Integer и Real определены операции умножения и деления (*, /). Остальные операции, указанные в таблице (операции по модулю mod, нахождения остатка rem, абсолютного значения abs и возведения в степень **), имеют обычные функции, как в стандартных языках программирования.

2.8 Классы объектов

В языке VHDL имеются три класса объектов: константы, переменные и сигналы. Объект создается, когда дается его описание. По мере рассмотрения всех этих трех классов объектов приведем примеры описаний.

Константа - это объект, значение которого не может изменяться. В качестве примеров описания констант приведем следующие:

constant TIEOFF : MVL := '1',
constant OVFL MSG : STRING(1 **to** 20) := "Переполнение аккумулятора";
constant INT_VECTOR : bit_vector (0 **to** 7):= "00001000";
constant Pi : Real :=3 .14;

В описании каждой константы указывается ее имя, тип и значение.

Переменные - это объекты, значения которых могут меняться. Когда по описанию создается некоторая переменная, вместе с ней создается и так называемый "контейнер" для этого объекта. Изменение значений переменных осуществляется путем выполнения оператора присваивания для переменной, например, A:=B + C;. Операторы присваивания для переменных не имеют временного параметра, т.е. их результат сказывается немедленно. В связи с этим переменные не имеют прямого отображения в аппаратных средствах, но полезны для алгоритмических представлений. Приведем некоторые примеры описаний переменных:

variable RUN: Boolean:= False;
variable COUNT: Integer := 0;
variable ADDR: bit_vector (0 **to** 11);

Таким образом, в описаниях переменных указываются имя, тип и, необязательно, начальное значение данной переменной. Переменные, используемые в блоке процесса, считаются статическими, т.е. система моделирования сохраняет значение такой статической переменной до тех пор, пока оно не изменится оператором присваивания. В качестве примера рассмотрим блок процесса, используемый для моделирования микросхемы оперативного запоминающего устройства (RAM):

```
RAM: process (ADDR,RW,CS)
      type MEMORY is array(0 to 1023) of Bit_vector(0 to 7)
      variable MEM : MEMORY;
       begin
       ----------   Чтение из памяти MEM и запись в память MEM
       ----------
end process RAM;
```

Поскольку переменная MEM является статической, записанные в MEM значения будут сохраняться до момента изменения ее оператором присваивания.

Сигналы - это объекты, значения которых могут меняться и которые имеют временные параметры. Значения сигналов меняются операторами назначения сигналов, например, A< = B + C **after** 50 ns или D<=E nor C. Отметим, что в операторе назначения сигнала используется символ < =, чтобы отличить его от оператора присваивания для переменной. В первом примере оператора назначения сигнала указано "через 50 нс". Это означает, что сигнал A может в принципе принимать свое новое значение спустя 50 нс после текущего момента времени моделирования. В рамках модели может быть более одного оператора назначения, действующего для сигнала A. Во втором примере нет фразы "after" (через), что эквивалентно "через 0 нс". Однако операторы назначения сигналов применяются для представления событий в реальных электрических схемах, так что, если даже нет фразы "after" или если в результате вычисления оказывается, что фактический параметр этой фразы равняется нулю, предполагается, что данный сигнал принимает свое новое значение с некоторой дельта-задержкой. Здесь дельта - произвольно малое значение времени, большее нуля.

Еще одно отличие сигналов от переменных состоит в том, что сигналы могут иметь драйверы, причем значение данного сигнала есть функция всех его драйверов. Пример подобной ситуации приведен на рис. 2.4. Здесь показаны два процесса A и B, каждый из которых содержит оператор назначения для сигнала X. Для каждого процесса, присваивающего значение сигналу X, создается драйвер, фиксирующий результат такого присваивания. В данном примере драйверы имеют метки Dax и Dbx. Значение сигнала X вычисляется при помощи функции разрешения шины (здесь F). Функции разрешения шины определяет пользователь: они вычисляются, когда один из драйверов данного сигнала получает новое значение. После этого значение сигнала меняется на значение, вычисленное по функции разрешения шины.

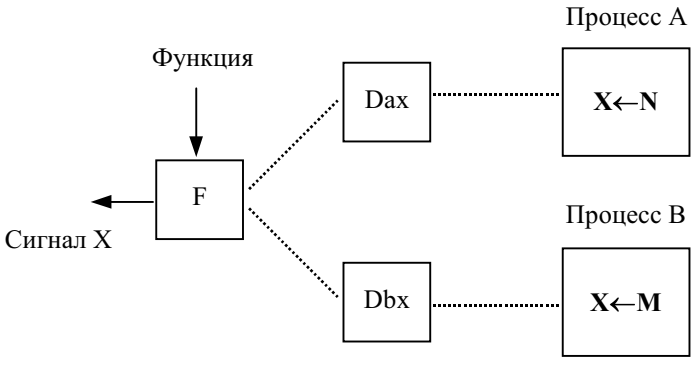


Рис. 2.4 Пример схемы с многими драйверами сигнала и функцией разрешения шины

Оператор присваивания сигнала представляет собой параллельный оператор (a<=b **and** c или m<=In1 **when** a1 **else** In2). Присваивание на этом уровне расценивается как описание потока данных. Но в некоторых случаях рассматривается как описание моделирования на уровне регистровых операций.

В комбинационных схемах поток данных описывается с помощью оператора присваивания сигнала и это не регистровая операция. В последовательностных схемах в оператор присваивания включают дополнительные условия.

2.9 Атрибуты

Атрибуты - это значения, связанные с поименованным элементом - объектом в языке VHDL. Для построения моделей и моделирования особенно важную роль играют атрибуты. Есть несколько групп атрибутов, например, для типов, массивов, сигналов и пр. Рассмотрим группу атрибутов, определенную для сигналов:

1 S'last_value (прошлое значение S) - предыдущее значение, которое сигнал S имел непосредственно перед последним изменением S. Относится к тому же самому типу, что и S. Может использоваться для того, чтобы проверять, не изменился ли данный сигнал. Например, оператор

if S/=S'last_value позволяет контролировать изменение сигнала S.

2 S'stable(T) - тип Boolean. Атрибут имеет истинное значение, если сигнал S стабилен в течение последних T единиц времени. Если T = 0, атрибут записывается как S'stable.

3 S'delayed(T) есть значение, которое сигнал S имел на T временных единиц ранее. Относится к тому же самому типу, что и S.

4 S'event, тип Boolean, принимает истинное значение, если только что произошло изменение сигнала S.

Эти атрибуты полезны для обнаружения изменений сигналов и детального временного моделирования.

Еще один полезный набор атрибутов относится к массивам. Предположим, например, что переменная массив была определена следующим образом:

variable A: bit_vector (0 **to** 15).

В этом случае атрибуты имеют значения, приведенные в табл. 2.4:

Таблица 2.4

Атрибут	Значение
A'range (диапазон переменной A)	от 0 до 15
A'left (левая граница A)	0
A'right (правая граница A)	15

2.10 Функции и процедуры

В языке VHDL описание функции осуществляется как в обычных языках программирования. Указывается имя функции и, если необходимо, входные параметры. Функция имеет оператор возврата с определенным типом возвращаемого параметра.

Функция может быть описана, например, следующим образом:

```
function Is_zero(n : Integer) return Boolean is
  -- описание типов, переменных, констант, подпрограммы
begin
   -- последовательные операторы
   if n=0 then return True;
   else return False;
   end if;
end;
```

Возвращаемое значение может быть определено в виде выражения либо вычислено через соответствующую последовательность операторов. Как отмечалось выше, могут создаваться локальные описания, однако, в данном простом случае это не требуется.

Язык VHDL предусматривает также описание процедур. Процедура может иметь как внутренние, так и внешние параметры. В остальном описание процедуры аналогично описанию функций. Ниже показан пример описания процедуры:

```
procedure Count (Incr : Boolean; big : out Bit; num : inout Integer) is
  -- описание типов, переменных, констант, подпрограмм
begin
  if Incr then
   num := num+1;
  end if;
  if num >101 then
big := '1';
  else
   big := '0';
  end if;
end;
```

Отметим, что в списке параметров указываются и входные, и выходные параметры. За списком параметров следует раздел описания. Алгоритм, реализуемый данной процедурой, - это последовательность операторов после ключевого слова begin.

2.11 Пакеты

Для того, чтобы повысить эффективность программирования, в языке VHDL предусмотрен механизм пакетов для часто используемых описаний. Пакету присваивается имя. Описания, содержащиеся в пакете, можно сделать видимыми, давая ссылку на данный пакет. Вначале, однако, пакет нужно описать. Пример подобного описания приведен ниже. Отметим, что в описании пакета могут располагаться подтипы и интерфейс для функции.

```
-- описание пакета
package Pack1 is
  -- описание типов, констант, сигналов, подпрограмм
end Pack1;
```

Код функции содержится в теле пакета. Если в пакете нет подпрограмм, тела пакета не требуется.

```
-- описание тела пакета
package body Pack1 is
  -- формирование подпрограммы
end Pack1;
  -- описание интерфейса
entity ID_1 is
  port(A: in bit_vector(0 to 3); Y: out bit_vector(0 to 9));
end ID_1;
  -- объявление архитектуры
architecture DC of ID_1 is
-- описание типов, сигналов, констант, подпрограмм
begin
-- параллельные операторы блоки, процессы
end DC;

-- объявление конфигурации
configuration Example of  ID_1 is
-- конфигурация
end Example;

-- подключение библиотеки
-- содержимое библиотеки не делается видимым
```

```
library utils;

-- предложение использования в начале проекта
-- содержимое библиотеки становится видимым
use utils.all;
use utils.utils_pkg.all;
```

Имея описание пакета, предположим, что нужно организовать доступ к этому пакету со стороны объекта под именем LogSys. Это можно сделать внутри проекта, поместив фразу **use** (использовать) перед описанием интерфейса объекта LogSys:

```
use Pack1.all;
entity LogSys is
   port(A: in bit_vector(0 to 3); Y: out bit_vector(0 to 9));
end LogSys;
```

В этом случае все (all) описания, содержащиеся в пакете Pack1, будут "видимыми" для объекта LogSys, включая все его архитектурные тела.

Пакеты - очень полезная особенность языка описания аппаратуры. Группы разработчиков могут использовать стандартные пакеты, содержащие описания типов и подпрограммы, относящиеся к проектируемой ими системе. Эти описания и подпрограммы имеют значительный объем кода. Механизм пакетов освобождает разработчика модели от необходимости многократно вводить этот код. Кроме того, если в группе разработчиков код коллективно используют несколько специалистов, то обеспечивается непротиворечивость данных для всего проекта.

В языке VHDL предусмотрен пакет standard (стандарт), который может использоваться всеми объектами. В числе прочих вещей этот пакет содержит описания типов Bit, Bit_vector, Boolean, Integer, Real, Character, String и Time, а также подтипы Positive и Natural. В зависимости от версии, используемой для разработки проекта, могут подключаться пакеты std.vhd (стандартный пакет IEEE 1076), ieee. vhd (пакет стандартной логики IEEE 1164), num_bit.mm0 (пакет числовой и битовый IEEE 1076.3), num_std.mm0 (пакет стандартный числовой IEEE 1076.3). Для ввода или вывода текстовой информации в проекте надо подключить библиотеку Vector (library vector;), а затем записать строку use std.textio.all, vector.functions.all.

2.12 Операторы управления

Среди операторов управления, используемых в языке VHDL, можно выделить стандартные и специальные, характерные только для данного языка, как, например, wait. Рассмотрим основные и наиболее часто используемые операторы управления.

```
-- Оператор if (если)
if increment and not decrement then
   count := count +1;
  elsif not increment and decrement then
   count := count -1;
  elsif increment and decrement then
   count := 0;
 else
 count := count;
end if;
```

Условия в операторе должны быть типа Boolean. Возможно наличие любого числа фраз elsif. Фразы elsif и else являются необязательными.

Оператор case (выбор) проиллюстрируем несколькими примерами.

Пример 1

```
case X(0 to 1) is
 when "00" =>  Z <=  0;
 when "01" =>  Z <=  1;
 when "10" =>  Z <=  not 2;
 when "11" =>  Z <=   Z;
```

Пример 2

```
case N is
 when 0   => Z  <=  0;
 when 1   => Z  <=  1;
 when 2   => Z <=   not 2;
 when 3   => Z <=   2;
```

```
    end case;                                    end case;
```

Пример *3*

```
case day is
when Saturday to Sunday =>
  work := False;
  work_out := False;
when Monday | Wednesday | Friday =>
  work := True;
 work_out := True;
when others =>
 work := True;
 work_out := False;
end case;
```

Оператор выбора case осуществляет декодирование на основе значения управляющего выражения, а затем выполняет выбранный оператор (или группу операторов). В *примере* 1 управляющее выражение - битовый вектор, *примере* 2 - целое, *примере* 3 - символьное. В общем случае в качестве управляющего выражения может использоваться любой дискретный тип, как в показанных примерах, либо одномерный массив символов.

В языке VHDL существует оператор назначения сигнала с выбором, позволяющий сделать более сжатую запись:

```
with X1&X2&X3 select
f <= '0' after 20 ns when "000",
'1' after 20 ns when "001",
'0' after 20 ns when "010",
'1' after 20 ns when "011",
'0' after 20 ns when "100",
'1' after 20 ns when "101",
'1' after 20 ns when "110",
'1' after 20 ns when "111";
```

Операторы цикла реализуют те же функции, что и во всех языках программирования. Приведем четыре примера с использованием операторов управления for, while, loop, next и exit:

Пример 1

```
for i in 0 to 3 loop
  A(i):= 2** i
 end loop;
```

Пример 2

```
Sum: =0;
i:= 1;
Sum_Int: while i<= n loop
    Sum:= Sum + 1;
    i := i + 1;
end loop Sum_Int;
```

Пример 3

```
Sum:= 0;
i:= 0;
Sum_Int: while i <= N loop
   i:= i+1;
   next Sum_Int when i=3;
   Sum:= Sum + 1;
end loop Sum_Int;
```

Пример 4

```
Sum:= 1;
loop
        VAL(X);
        exit when X <0;
        Sum:= Sum + X;
        end loop;
```

Пример 1 иллюстрирует базовый цикл for, а *пример* 2 - цикл while и использование метки цикла. Цикл *примера* 4 вычисляет сумму первых n целых. В *примере* 3 производится вычисление той же самой суммы, но без числа 3, поскольку эта итерация пропускается при помощи оператора next (следующий). *Пример* 4 показывает потенциально бесконечный цикл, который прибавляет к накопленной сумме значения, поступающие из под-программы. Выход из этого цикла будет

происходить в случае, когда подпрограмма VAL(X) выдаст отрицательное значение. Оператор for может работать в обратном направлении:

```
L1 : for i in 0 to 9 loop
L2 : for j in 0 to 9 loop
for k in 4 downto 2 loop   -- дополнительная метка цикла
   if k = i next L2;       -- следующий переход на метку L2
end loop;
exit L1 when j =8;         -- выход на метку L1
end loop;
end loop;
```

Два последних оператора управления - return (возврат) и wait (ожидание). Оператор return используется в процедурах и функциях.

Оператор ожидания WAIT используется для приостановки процесса на определенный период времени или до момента наступления определенного события, например, **wait until** clk. Полная конструкция оператора ожидания имеет вид:

wait on Sensitivity_list **until** Condition_clause **for** Time_out.

Указанный оператор приостанавливает процесс до момента, пока не изменится некоторый сигнал в списке чувствительности сигналов (Sensitivity_list). В это время вычислится условие (Condition_clause), если результат имеет значение "истина", то выполнение процесса возобновляется. Максимальное значение времени ожидания, после которого процесс возобновит работу, указывается значением Time_out. Некоторые условия могут быть пропущены, тогда оператор имеет неполный вид:

```
wait on X1,X2;       -- продолжить работу, когда изменится X1 или X2
wait until (X3=0);   -- продолжить работу, когда изменится X3 из 1 в 0
wait for 100 ns;     -- продолжить работу через 100 нс.
```

Вариант оператора процесса без списка сигналов запуска предполагает использование оператора ожидания с этими сигналами запуска, записываемого последней строкой в операторе процесса.

2.13 Задержки сигналов

В цифровых системах сигналы передаются не мгновенно, а с некоторой задержкой во времени. При моделировании на языке различают два типа задержек сигналов - инерционную и транспортную. Каждый логический элемент, если рассмотреть временную диаграмму его работы, получая на входе значение сигнала, выдает на выход обработанное значение через определенное время. Стандартный логический элемент, выполненный по технологии ТТЛ, имеет задержку 20 нс. Более сложные элементы, например, ПЗУ, имеют время выбора по установленному адресу 200 - 400 нс.

Для моделирования такого типа задержек в распространении сигнала и используется оператор инерционной задержки after. Например, запись Y<= X1&X2 **after** 20 ns, означает, что значение сигнала Y изменится через 20 нс после изменения одного из сигналов X.

Транспортная задержка предполагает, что все изменения на входе будут передаваться на выход независимо от времени существования входного сигнала. В этом состоит главное отличие ее от инерционной, работающей по принципу фильтра. Фактически при быстром изменении сигнала на входе логической схемой будет обнаружено только то значение, которое будет присутствовать по истечении времени задержки. Иными словами, часть входного сигнала в течение времени задержки будет "вырезана" из временной диаграммы работы. При транспортной задержке временная диаграмма "вход-выход" будет более полной, но со сдвигом по времени на значение задержки. В качестве физического носителя информации можно провести аналогию с проводником, соединяющим

микросхемы. Оператор транспортной задержки записывается следующим образом: Y<= **transport** X **after** 3 ns.

3 МОДЕЛИРОВАНИЕ ЛОГИЧЕСКИХ СХЕМ

Требования, предъявляемые к моделированию логических схем, включают параллельное выполнение операторов, использование задержек сигналов и разделение методов моделирования, применяемых к схемам комбинационным и последовательностным.

3.1 Моделирование комбинационных схем

Схема считается комбинационной, если в ней нет обратных связей или элементов памяти. Модели комбинационных схем в языке VHDL могут быть описаны различными командами: логическими, арифметическими, с использованием операций отношения и операторов одновременного утверждения, условных переходов и вызовов процедур. Рассмотрим некоторые варианты схем и различные способы их представления.

Описание логической схемы, показанной на рис. 3.1, можно представить логическими операторами: and, or, nand, nor, xor, nxor, not. Задержка сигнала в каждом элементе составляет 20 нс.

```
entity logsh_1 is
   port (x1, x2, x3, x4: in bit;  y: out bit);
end logsh_1;
architecture p1 of logsh_1 is
   signal e1,e2: bit;
begin
e1 <= x1 and x2 after 20 ns; -- параллельное выполнение
e2 <= x3 xor x4 after 20 ns;
y <= e1 or e2 after 20 ns;
end p1;
```

Если несколько элементов работают параллельно (рис. 3.2), то в описании можно использовать вектор:

```
entity logsh_2 is
  port (a, b: in bit_vector (0 to 3);  y: out bit_vector (0 to 3));
end logsh_2
architecture p2 of logsh_2 is
begin
  y <= a and b after 20 ns;
end p2;
```

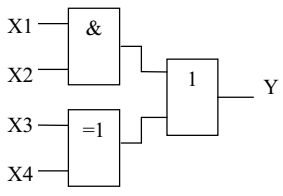


Рис. 3.1 Последовательная обработка

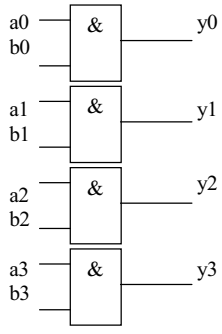


Рис. 3.2 Параллельная

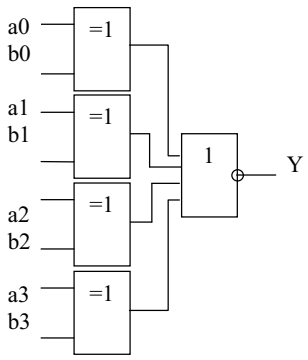


Рис. 3.3 Схема компаратора

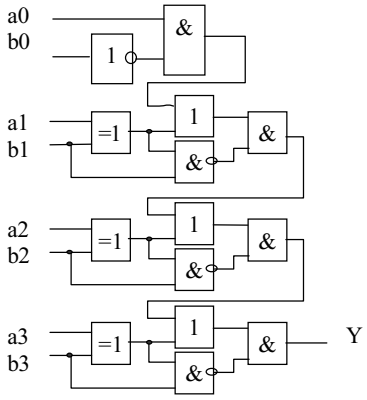


Рис. 3.4 Схема компаратора

В этих примерах происходит прямая трансляция логической схемы в операторы языка. Чтобы получить реальную картину распространения сигналов, учитываются задержки во всех элементах схемы. Последовательность "трансляции" схемы в операторы программы совпадает с последовательностью прохождения сигналов через элементы.

Схемы, предназначенные для сравнения сигналов, рациональнее моделировать операторами отношений или упорядочения. На рис. 3.3 показана схема компаратора, определяющего поразрядное сравнение сигналов A и B. Если хотя бы один разряд $a_i \neq b_i$, то на выходе схемы - уровень логической единицы, иначе уровень логического нуля.

Схема, показанная на рис. 3.4, выполняет сравнение a ≥ b. Фрагмент проекта для компараторов может быть записан в виде:

```
entity relat_1 is       -- отношение a ≠ b
   port (a, b: in bit_vector (0 to 3);  y: out boolean);
end relat_1;
architecture cmp of relat_1 is
begin
   y <= a = b after 40 ns;
end cmp;

entity relat_2 is       -- отношение a ≥ b
   port (a, b: in integer range 0 to 3;  y: out boolean);
end relat_2;
architecture cmp of relat_2 is
begin
   y <= a >= b after140 ns;
end cmp;
```

Недостаток такого описания состоит в том, что приходится указывать приблизительную задержку сигналов, равную сумме задержек каждого элемента. В первой схеме компаратора каждый сигнал проходит два элемента, следовательно, суммарное значение 40 нс (по 20 ns на каждый элемент). Во второй схеме в максимальном случае сигнал проходит семь элементов.

Логические схемы сумматора или вычитателя, а тем более умножения, деления, возведения в степень очень сложны в логических элементах принципиальной схемы. Кроме сумматора, другие схемы применяются крайне редко. Транслятор VHDL делает специальную оптимизацию, но можно применить ее на этапе разработки проекта.

Проиллюстрируем логику сумматора (рис. 3.5) фрагментом программы с использованием пакета оператора печати объявлений:

```
package add_vect is
  type small_int is range 0 to 2;
end add_vect ;
use add_vect.all;
entity arithmetic is
   port (a, b: in small_int;  c: out small_int);
end arithmetic;
```

```
architecture exmpl of arithmetic is
signal c: bit;
begin
    c <= a + b after 80 ns;
end exmpl;
```

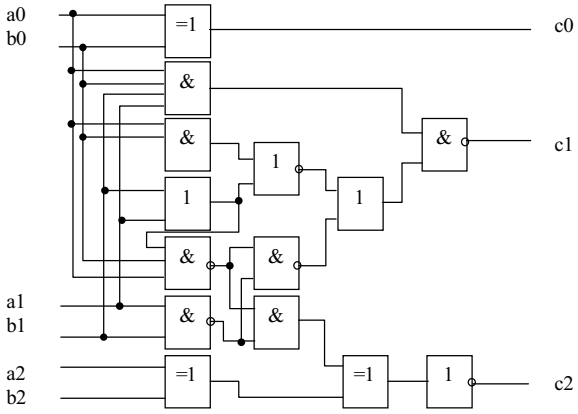


Рис.3.5. Логическая схема сумматора **Рис. 3.5 Логическая схема сумматора**

Как и в ранее рассмотренном примере, недостаток описания состоит в моделировании задержки.

VHDL обеспечивает одновременные утверждения для создания условной логики, мультиплексоров, демультиплексоров, схем выбора. Схема, показанная на рис. 3.6, легко представляется оператором **if**, а на рис. 3.7 - оператором выбора **case**. В приведенных фрагментах программ реализованы последовательное и одновременное утверждения в соответствии с логическими схемами.

```
entity control is
  port (a, b, c: in boolean;  y: out boolean);
end control;
architecture p of control is
begin
   y <= b when a else  c;
end p;
```

Для выбранного значения сигнала a, b, c, d по коду sel разрешается прохождение на выход y:

```
entity controls is
   port (sel: bit_vector (0 to 1);  a,b,c,d : bit;  y: out bit);
end controls;
architecture p of controls is
begin
  with sel select
     y <= c when b"00",
     y <= d when b"01",
     y <= a when b"10",
     y <= b when others;
end p;
```

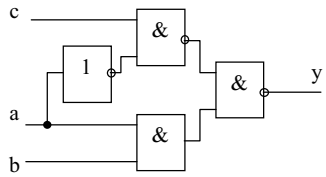


Рис. 3.6 Логическая схема для if

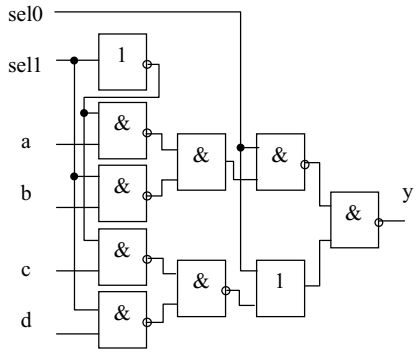


Рис. 3.7 Логическая схема выбора case

ия схема выбора case

Те же функции могут быть выполнены с использованием последовательных утверждений и будут происходить внутри процесса. Условие в условном операторе должно быть обязательно булева типа.

Следующий пример иллюстрирует решение той же задачи, но с использованием условного оператора:

```
entity control is
  port (a, b, c:  boolean;  y: out boolean );
end control;
architecture exmpl of control is
begin
 process (a, b, c)
 variable n: boolean;
  begin
    if a then
         n := b;
    else
         n := c;
    end if;
   y <= n after 60 ns;
end process;
end exmpl;
```

Использование оператора выбора (или выбранное назначение сигнала) компилируется быстрее и отрабатывает логику с меньшей задержкой распространения, чем использование вложенных условных операторов.

VHDL требует, чтобы все возможные условия были представлены в операторе выбора. Чтобы гарантировать это, используйте предложение "другие" в конце оператора выбора, чтобы исключить любые неопределенные условия.

Следующий пример иллюстрирует такой вариант:

```
entity control is
  port (sel: bit_vector(0 to 1);  a,b,c,d : bit;  y: out bit);
end control;
architecture p of control is
begin
process (sel,a,b,c,d)
begin
case sel is
  when b"00"  => y <= c;
  when b"01"  => y <= d;
  when b"10"  => y <= a;
  when others => y <= b;
end case;
end process;
end p;
```

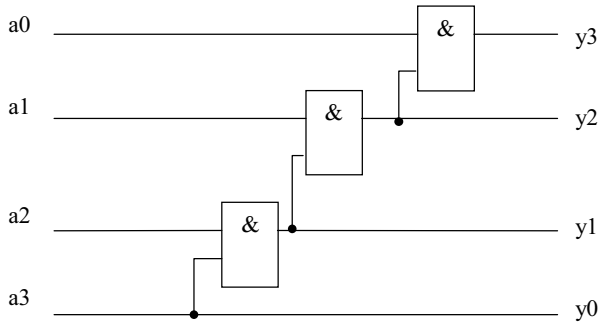


Рис. 3.8 Каскадное включение ное включение

Если есть повторяющиеся фрагменты схемы или каскадное включение одинаковых элементов, то достаточно организовать цикл или вызов подпрограмм, имея в таком случае описание только одного элемента.

Если возможно, сначала для цикла генерируют диапазоны. Иначе логика внутри цикла может скопироваться для всех возможных значений диапазонов цикла. Это может иметь сложную реализацию в элементах схем.

Рассмотрим пример (рис. 3.8) каскадного включения элементов 2И. Фрагмент программы для такой схемы будет выглядеть следующим образом:

```
entity loop_sh is
   port (a: bit_vector(0 to 3);  y: out bit_vector(0 to 3));
end loop_sh;
     architecture p of loop_sh is
   begin
   process (a)
   variable b:bit;
   begin
     b := '1';
     for i in 0 to 3 loop
       b := a(3-i) and b;
       y(i) <= b;
       report "Loop number = " &  integer'image(i);
     end loop;
    end process;
end p;
```

Количество итераций должно быть ограничено; если оно будет определяться состоянием какого-либо сигнала, то можно получить бесконечный цикл. Внутри цикла можно расположить генератор отчета и видеть на экране каждую итерацию цикла.

Старшие версии языка VHDL имеют в своем составе команды сдвига, позволяющие работать непосредственно с битами: sll, srl, sla, sra, rol, ror.

Левая часть операнда может быть одномерным массивом типа bit или boolean, а правая часть - типа "целое". Если правая часть операнда - константа или выражение, то никаких действий не выполняется. Ниже приведен фрагмент программы с использованием команд сдвига влево, вправо и циклического сдвига влево.

```
     entity sr_1 is
        port (a, b,c: in bit_vector (5 downto 0 );
        ctl : integer range 0 to 2**5 -1;
         w,x,y: out bit_vector (5 downto 0));
     end sr_1
     architecture p of  sr_1 is
     begin
       w <= a sll ctl;   -- сдвиг влево и заполнение '0'
       x <= a sra ctl;   -- сдвиг вправо и запись  a'left [ a(5) ]
       y <= a rol ctl;    -- циклический сдвиг влево
     end p;
```

Отрицательный правый параметр означает сдвиг или чередование во встречном направлении.

В некоторых случаях при моделировании необходимо учитывать третье состояние элемента. Для этого предусмотрено подключение стандартных пакетов std_logic, определенных в ieee. std_logic_1164, или использование пустого оператора null, чтобы выключить драйвер. Первый метод применяется только к типу std_logic, второй применяется к любому типу. Первый метод обычно используется чаще.

```
library ieee;
use ieee.std_logic_1164.all;
entity tbuf is
  port (enable : boolean;
        a : std_logic_vector(0 to 4);
        m : out std_logic_vector(0 to 4));
end tbuf;
architecture p of tbuf is
begin
  process (enable, a)
     if enable then
       m <= a;
    else
       m <= 'Z';
     end if;
 end process;
end p;
```

Если надо описать внутреннюю шину, то можно воспользоваться вторым методом.

```
package p_bus is
  subtype bundle is bit_vector (0 to 4);
end p_bus;
use work.p_bus.all;
entity tbuf is
  port (enable: boolean;  a: bundle;  m: out bundle bus);
end tbuf;
architecture p of tbuf is
begin
  process (enable, a)
  begin
    if enable then
      m <= a;
    else
      m <= null;
    end if;
  end process;
end p;
```

Применение пустого оператора к сигналу вида шины выключает драйвер. Когда в условном операторе выполнится условие, будет сформирован буфер с тремя состояниями.

3.2 Моделирование последовательностных схем

Для моделирования последовательностной логики в языке VHDL нет специальных ресурсов, однако, правильное использование операторов задержек, ожидания, охранных конструкций в блоках и процессов позволяет достаточно хорошо описывать схемы такого типа. Динамика изменения сигналов (фронты и срезы) регистрируется атрибутами языка. Очень хорошо составлять проекты, применяя при описании схем их представления в виде конечных автоматов. В практике программирования сложились определенные подходы к описанию тактирования схем и записи-чтения данных, сброса и установки состояний. Далее будут рассмотрены некоторые варианты этих подходов, несомненно имеющих возможность модификации и улучшения.

Базовые компоненты последовательностной схемы - это один или несколько блоков комбинационных схем и некоторое множество обратных связей. Задержки сигналов в таких схемах могут быть тактированными (синхронными) или зависящими от времени прохождения сигнала (асинхронные).

Состояние синхронных схем может отображаться переменными, например, накопление значения в счетчике. Но в зависимости от состояния синхронизирующих сигналов значение

переменной присваивается с определенной задержкой выходному сигналу. В асинхронных схемах механизм обратных связей моделируется аналогично синхронным схемам, меняется только семантика переменной обратной связи. Для синхронных схем переменная обратной связи представляет период тактовой частоты, а для асинхронной схемы - задержку распространения.

Одним из часто используемых в последовательностных схемах элементом является тактируемый триггер. Такой триггер имеет вход данных Data, вход тактового сигнала Clk, вход начальной установки Reset или Set и выход хранимых данных Q. Триггер передает данные с входа на выход только по фронту (уровню или срезу) тактового сигнала. Моделирование таких элементов целесообразно проводить с использованием охранных конструкций в блоках. Для описанного выше триггера фрагмент программы может быть, например, следующим:

```
D: block ((Clk='1' and not Clk'Stable) or Reset='1')
    begin
    Q<=guarded '0' when Reset='1'; else
    Data when Clk='1' and not Clk'Stable; else
    Q;
end block D;
```

В этом примере данные записываются по фронту Clk. Если Reset имеет высокий уровень, то значение триггера сбрасывается в нуль. Любые изменения на входе и отсутствие перехода из нуля в единицу на тактовом входе не изменяют хранимого в триггере значения.

Следующий пример иллюстрирует запись информации по сигналу Clk, поступающему на вход схемы. Одновременно выполняется некоторая обработка входных данных. Во всех случаях сигнал "y" сохраняет текущее значение, если нет изменений уровня тактового сигнала.

```
process (Clk, a, b)    -- список всех сигналов, используемых в процессе
begin
  if Clk then
    y <= a and b;
  end if;
end process;
```

Указанный фрагмент программы можно разместить внутри функции или процедуры и вызывать при каждой записи фиксированных данных. Такой подход удобен и тогда, когда необходимо моделирование параллельной обработки сигналов, например, одновременной фиксации данных в регистре, построенном на нескольких триггерах.

```
procedure my_latch (signal Clk, a, b :  boolean; signal y : out boolean)
begin
  if Clk then
    y <= a and b;
  end if;
end;
```

Обращение к процедурам, например, с именем proc_A внутри про-цесса может быть записано так:

```
L1: proc_A ( clock, input1, input2, outputA );
L2: proc_A ( clock, input3, input4, outputB  );
```

При назначении сигнала можно использовать условные выражения. Здесь следует обратить внимание на то, что "y" используется в двух направлениях:

```
y <= a and b when Clk else y;
y <= a and b when Clk;
```

Если триггер должен быть чувствителен к фронту тактового сигнала Clk, то условие записи дополняется атрибутом, например:

```
process (Clk)     -- список сигналов чувствительности
begin
  if Clk and Clk'event then   -- определение среза сигнала
```

```
      y <= a and b;
   end if;
end process;
```

Здесь сигнал "y" сохраняет значение, если Clk не изменяется (нет перехода от нуля к единице).

В операторе процесса может отсутствовать список сигналов чувствительности, тогда применяют оператор ожидания выполнения условия:

```
process     -- нет списка сигналов
begin
  wait until not Clk ;  -- ожидание среза сигнала
     y <= a and b;
end process;
```

Если необходима параллельная обработка, то, как и в предыдущем примере, объявляем процедуру и вызываем ее при необходимости:

```
procedure my_ff (signal Clk, a, b :  boolean; signal y : out boolean)
begin
  if not  Clk  and Clk'event then   -- фронт тактового сигнала
     y <= a and b;
  end if;
end;
```

Как и в предыдущем примере, "y" используется двунаправленно:

```
y <= a and b when Clk and Clk 'event  else y;
y <= a and b when Clk and Clk 'event;
```

Вход сброса или установки значений триггера по изменению сигнала или по его уровню иногда описывают в виде функции:

```
function rising_edge (signal s : bit ) return boolean is
begin
  return s = '1' and s'event;
end;
```

При использовании этой функции D-триггер может быть написан как:

```
q <= d when rising_edge(Clk);
```

Условие проверки тактового сигнала, как правило, простое и может включать элементарные логические операции, например, когда важно не только изменение Clk, но и дополнительный сигнал разрешения. Тот же результат достигается составным условным оператором. Но в таких случаях существует вероятность пропуска тактового сигнала при временных ограничениях.

Начальное значение триггера можно сбросить или установить внешними входными сигналами Reset или Set. Такое действие может быть синхронным или асинхронным. В текст программы добавляется еще одна переменная или константа. Ниже показан пример синхронной установки триггера по сигналу Set.

```
process (Clk)
begin
  if Clk and Clk'event then       -- фронт сигнала
   if Set then
     y <= true;               -- y типа boolean
   else
     y <= a and b;
   end if;
  end if;
end process;
```

Пример сброса и установки группы бит по сигналу init:

```
process
```

```
begin
  wait until Clk                    -- фронт Clk
    if init then
y <= 7;                             -- y типа integer
    else
y <= a + b;
    end if;
end process;
```

Поведение асинхронного сброса или установки описывается независимо от управления по тактовому сигналу. Просто добавляется еще одно условие и действие по этому условию. Рассмотрим пример сброса значения для использования в параллельной обработке по сигналу Reset:

```
y <= false when Reset else a when Clk and Clk 'event  else y;
```

Полное описание триггера с асинхронным сбросом или асинхронной установкой, где обработка сигналов независима и параллельна, можно представить так, как показано ниже:

```
process (Clk, Reset)
begin
  if Reset then
    q <= false;              -- y типа boolean
  else
    if Clk and Clk'event then    -- фронт clock
      q <= d;
    end if;
  end if;
end process;
procedure ff_async_set (signal Clk, a, Set:  boolean;
            signal q : out boolean)
begin
  if Set then
    q <= true;
  elsif Clk  and Clk'event then  -- фронт clock
    q <= a;                      -- ввод данных
  end if;
end;
```

Чтобы описывать поведение и асинхронной установки и асинхронного сброса, мы просто добавляем второй условный оператор:

```
q <= false when Reset else
  true when Set else
 d when Clk and Clk 'event;
-- Reset и Set в параллельной обработке
process (Clk, Reset, Set)
begin
  if Reset then
    q <= false;      -- q типа boolean
  elsif  Set then
    q <= true
  else
    if Clk and Clk'event then    -- фронт clock
      q <= d;
    end if;
          end if;
end process;
```

Чтобы описывать поведение асинхронной загрузки, надо заменить переменную Set или Reset сигналом или выражением.

Загрузка данных может выполняться с использованием последовательных операторов:

```
process (Clk, load_ctl,load_data)
begin
```

```
  if load_ctl = '1' then
    q <= load_data;
  elsif rising_edge(Clk) then
    q <= d;
  end if;
end process;
```

Таким образом, описание триггера различно по загрузке данных, сбросу и установке исходных значений. В процессе моделирования необходимо иметь четкое представление не только о логике работы схемы, но и по какому принципу: синхронному или асинхронному выполняются действия.

4 МОДЕЛИРОВАНИЕ НА УРОВНЕ МИКРОСХЕМ

При составлении проектов на элементной базе, состоящей из микросхем высокой степени интеграции, например, ИС, БИС или СБИС, необходимо пользоваться моделями того же уровня. Модель уровня интегральной схемы - это поведенческая модель логического блока. Точное описание задержек сигналов в таких моделях осуществляется без использования описаний более низкого уровня.

Основные требования к моделям уровня интегральных схем формулируются следующим образом.

С целью получения правильно функционирующей схемы временные параметры входов и выходов должны быть точно смоделированными. Исходя из предположения, что схема сама является базовым примитивом, не допускается ее декомпозиция на более простые структурные примитивы. Модель схемы реализуется в виде последовательности микроопераций, описывающих логику работы схемы, без описаний вентильного или регистрового уровня. Разработчик модели может использовать текстовые описания, структурные схемы, спецификации и временные диаграммы, а также таблицы состояний и диаграммы состояний.

Для того, чтобы иметь точное представление временных параметров схемы, модель должна иметь достаточную степень детализации по времени.

Модель схемы можно условно разделить на три части: входную, внутреннюю и выходную. Входная часть содержит входные спецификации на количество входов, время предустановки и удержания сигналов, а также минимальные длительности импульсов. Внутренняя часть содержит спецификации на возможные микрооперации обработки сигналов (программируемые схемы), память, задержки сигналов в основных внутренних цепях. Выходная часть содержит спецификации на количество выходных сигналов и их временные параметры. Структура модели схемы имеет три важных составляющих: виды задержек сигналов, минимальная энергия и функциональное разбиение схемы.

Среди задержек сигналов можно выделить: простую, обратной связи и в цепях с точкой принятия решения. Простая задержка моделируется путем задержанного назначения одной переменной или сигнала другому сигналу. Задержанное назначение реализуется оператором after. Задержка обратной связи представляется сочетанием оператора задержанного присваивания и конструкции процесса. Выходной параметр при этом используется как входной в списке сигналов запуска процесса. Задержка в цепях с точкой принятия решения - более сложная модель задержки. Для ее моделирования применяется механизм процессов. Прохождение сигналов в таких схемах рассматривается в несколько этапов. На первом и третьем этапах сигнал задерживается в входных и выходных цепях. Второй этап соответствует "центральной" части схемы, где принимается решение пропустить сигнал далее или блокировать его. Возможна обработка сигнала микропрограммой схемы. Примером такой схемы может быть регистр с трехстабильными выходами и сигналами стробирования и разрешения записи данных. Примеры с микропрограммной обработкой: БИСы параллельного или последовательного ввода-вывода.

Модель схемы структурно может быть представлена в виде графа, а общие функции разбиты на подфункции. Каждый узел графа реализуется в отдельном процессе и выполняет одну подфункцию. Дуги графа обозначают пути прохождения сигналов между узлами-процессами. Кроме того, каждый узел графа может иметь свое функциональное разбиение. Другими словами, подфункция

имеет несколько входящих в нее функций. Например, подфункция обрабатывает сигналы X, Y, Z. Но действия выполняются отдельно над сигналами X и Y, X и Z, Y и Z. В этом случае узел графа следует разделить на три части, а подфункцию реализовать в виде трех отдельных процессов с соответствующими списками сигналов чувствительности.

Временные параметры входных и выходных сигналов представляются инерционной или транспортной задержкой. Тем самым достигается временное моделирование. Применение инерционных задержек можно контролировать с помощью операторов assert и report, так как возникает вероятность пропустить импульсы малой длительности. Это, в первую очередь, связано с временами предустановки и удержания, а также с нарушением требований к наименьшей длительности сигнала. Параметром оператора assert является выражение булева типа. В нем также можно использовать атрибуты сигналов. Параметром оператора report является любое сообщение, заключенное в двойные кавычки. Обычно оба оператора используются вместе и записываются последовательно друг за другом. Операторы контроля временных параметров могут включаться в описание интерфейса или архитектурное тело объекта проекта. Область действия в первом варианте глобальная, во втором - локальная, только в рамках одного архитектурного тела.

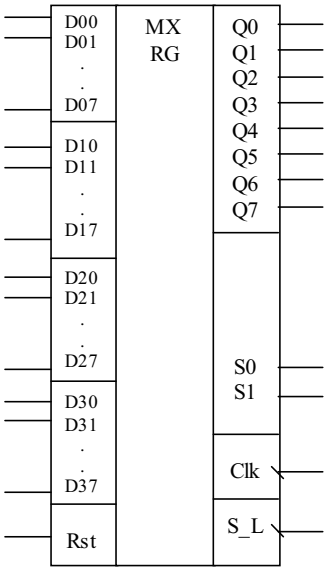


Рис. 4.1 Условное графическое обозначение БИС бозначение БИС

Рассмотрим пример разработки проекта. Пусть имеем БИС, включающую мультиплексор и сдвиговый регистр (рис. 4.1). Принцип работы БИС следующий. Четыре независимых восьмибитовых входа (D00-D07, D10-D17, D20-D27, D30-D37) коммутируются на восьмибитовый выход (Q0-Q7) по сигналам выбора S0 и S1.

Комбинации сигналов выбора определяют соответствующий им вход, например, если S0 = 0 и S1 = 0, то Qi = D0i; S0 = 1 и S1 = 0, то Qi = D1i; S0 = 0 и S1 = 1, то Qi=D2i; S0 = 1 и S1 = 1, то Qi = D3i. Данные с выбранного таким образом входа записываются в выходной регистр Qi по фронту сигнала Clk. Срез и уровень этого сигнала не влияют на записанные данные. Сдвиг зафиксированных данных влево происходит по фронту сигнала S_L. Высокий уровень сигнала Rst сбрасывает в нулевое значение содержимое регистра Qi.

Для моделирования теперь известен интерфейс БИС, и на основе описания функционирования можно построить поведенческую модель. Проведем функциональное разбиение микросхемы и построим граф модели процессов. На основе анализа функций БИС можно выделить следующие ее части: входные каналы данных и мультиплексор, сдвиговый регистр и выходные каналы данных. В каждой из частей сигнал проходит соответствующую обработку и задерживается на некоторое время. Один из возможных вариантов графа модели показан на рис. 4.2.

Процесс Mux соответствует коммутации одного из входных каналов на вход регистра с задержкой распространения Del_sel. Процесс Load отрабатывает функцию загрузки данных в регистр по сигналу Clk. Время записи данных Del_load. Сдвиговый регистр функционально разбит на схему

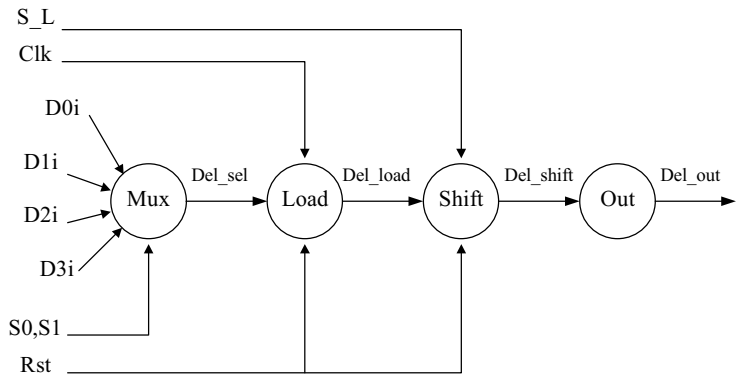


Рис. 4.2 Граф модели процессов БИС

фиксации данных и сдвига данных, так как операции записи и сдвига независимы друг от друга и активизируются различными сигналами. Время сдвига данных Del_shift. Спецификация выходных каналов необязательно приводит к отдельному процессу. Но следуя общей схеме, можно ввести еще один узел Out и соответствующий ему процесс. Время прохождения сигнала от сдвигового регистра до выхода схемы через выходные ключи равно Del_out.

По построенному графу теперь можно составить проект схемы на языке VHDL. Наиболее простой вариант проекта, где нет временных спецификаций сигналов и все процессы объединены в один, может выглядеть так:

```
package   typedef   is
   subtype byte   is bit_vector (7  downto 0);
 end;
use work.typedef.all;
entity Bis_data   is
   port    (clk,rst,s_l : boolean;
        s0, s1 : bit;
        d0, d1 ,d2, d3 : byte;
        q :  out byte);
end Bis_data;
architecture BIS  of Bis_data   is
    constant Del_sel, Del_load, Del_shift, Del_out : time := 10 ns;
    signal reg,shft : byte;
begin
    process (clk,rst,s_l)
    begin
        if rst  then                    -- асинхронный сброс
                reg <= x"00";
                shft <= x"00";
elsif clk  and clk'event  then    -- такт загрузки
    case s0 & s1   is            -- мультиплексирование
    when b"00" => reg <= d0 after Del_sel;
    when b"10" => reg <= d1 after Del_sel;
    when b"01" => reg <= d2 after Del_sel;
    when b"11" => reg <= d3 after Del_sel;
    end  case;

    if s_l and s_l'event then                -- такт сдвига
          shft <= shft rol 1 after Del_shift;
    else
         shft <= reg after Del_load;
    end  if;
end  if;
```

```
  end process;
  q <= shft after Del_out;
end BIS;
```

Следует отметить, что рассмотренный выше пример хорош для понимания логики работы БИС и не должен применяться для моделирования работы. Чтобы правильно выполнить моделирование, надо в соответствии с графом определить три процесса. Если потребуется контроль временных параметров, то сначала необходимо привести временную диаграмму с указанными в ней временными характеристиками сигналов (рис. 4.3) и "встроить" ее в проект.

На временной диаграмме показаны значения основных временных характеристик, одновременно диаграмма иллюстрирует принцип работы микросхемы. В начальный момент времени на вход асинхронного сброса микросхемы поступает сигнал Rst, имеющий наименьшую длительность τ_{Rst}. Через некоторое время, не показанное на диаграмме, на входы D поступают данные. Наименьшее время удержания данных обозначено как τ_{DATA}. За это время должна выполниться загрузка данных в регистр, через мультиплексор. После того, как данные на входе будут иметь устойчивое значение, интервал τ_{DS}, по сигналам S0 и S1 происходит выбор канала, коммутируемого на вход регистра. В течение времени τ_{Sel} (времени удержания), при неизменных значениях S0 и S1 по фронту сигнала Clk происходит фиксация данных в регистре.

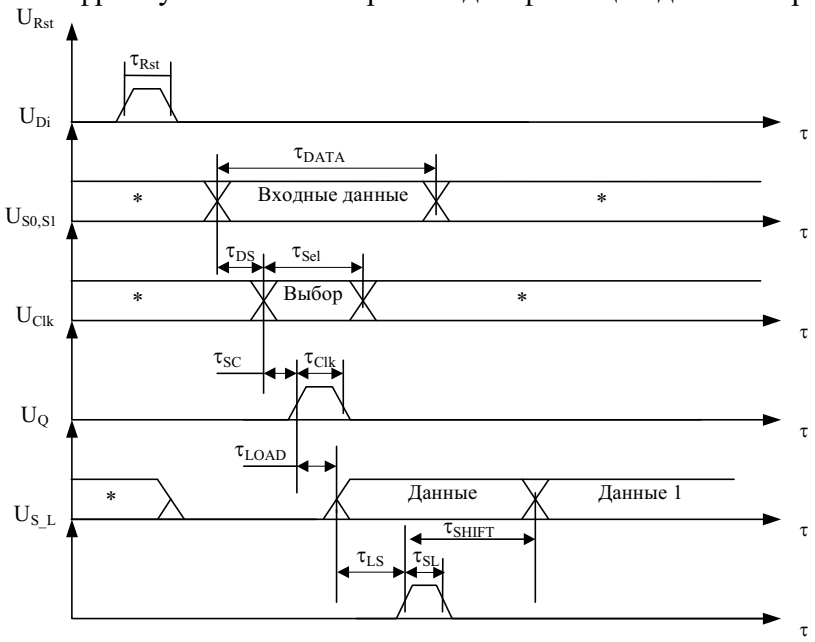


Рис. 4.3 Временная диаграмма работы БИС ма работы БИС

Наименьшее время между поступлением сигналов S0 и S1 фронтом Clk обозначено как τ_{SC}. Длительность импульса записи должна быть не меньше τ_{Clk}. В течение времени τ_{LOAD} данные записываются в регистр. Через интервал времени, наименьшее значение которого равно τ_{LS}, допускается сдвиг данных по фронту S_L. Операция сдвига может быть проведена не обязательно только после записи новых данных. Минимальная длительность сигнала сдвига - τ_{SL}. Обработанные таким образом данные появятся на выходах микросхемы не менее, чем через время τ_{SHIFT}.

Проект на языке VHDL в соответствии с графом модели и проверкой некоторых временных параметров показан ниже.

```
package typedef is
   subtype byte is bit_vector (7 downto 0);
end;
use work.typedef.all;

entity Bis_data is
```

```
        generic(Trst,Tsel,Tclk,Tsl : time);
        port    (clk,rst,s_l : boolean;
            s0, s1 : bit;
            d0, d1 ,d2, d3 : byte;
            q :  out byte);
    end Bis_data;
    architecture BIS  of Bis_data   is
        constant Del_sel, Del_load, Del_shift, Del_out : time := 10 ns;
        variable F : integer;
        signal reg,shft : byte;
    begin F:=0;
    Reset: process(rst)
    assert rst'stable(Trst)   report "Error Trst"
    if rst  then                    -- асинхронный сброс
        reg <= x"00";
        shft <= x"00";
    end if;
    end process Reset;
  Load:   process (clk)
     begin
      if clk'delay(Tclk)='1'  and             --Clk=1 в течение Tclk
        not clk'delay(Tclk)'stable  then --и перед этим не стабилен, то фронт
        assert s0'stable(Tsel) and          --если сигналы выбора стабильны, то
          s1'stable(Tsel)               --загрузка, иначе вывод сообщения
       report "Error Tsel"
      if  s0'stable(Tsel) and s1'stable(Tsel) then
        case s0 & s1   is                           -- мультиплексирование
        when b"00" => reg <= d0 after (Del_sel+Del_load);
        when b"10" => reg <= d1 after (Del_sel+Del_load);
        when b"01" => reg <= d2 after (Del_sel+Del_load);
        when b"11" => reg <= d3 after (Del_sel+Del_load);
        end  case; F:=1;
      end if;
    end if;
end process  Load;

    Shift:   process (s_l)
        if s_l='1' and s_l'stable(Tsl) then             -- такт сдвига
            shft <= shft rol 1 after Del_shift;
        end  if;
       F:=3;
    end  process Shift;

    Out: process
            if F=3 then  q <= shft after Del_out;
    else q <= reg after Del_out;
            end if;
            F:=0;
```

```
    end process Out;
end BIS;
```

Так как нет промежуточного хранения данных после мультиплексора, то две функции мультиплексирования и загрузки объединены в одну и оформлены в проекте в одном процессе. Отдельно записан процесс асинхронного сброса. Процесс Out не имеет сигналов запуска и поэтому всегда активен. В процессе моделирования проверяются только наиболее значимые временные характеристики. Это длительности импульсов и стабильность сигналов коммутации.

5 МОДЕЛИРОВАНИЕ СИСТЕМ

Моделирование систем имеет два основных аспекта решения задачи. Во-первых, все модели представлены на более высоком поведенческом уровне. Это соответствует таким компонентам, как микропроцессоры, микросхемы памяти, интерфейсные БИС и т.д. Во-вторых, все компоненты системы объединены схемой соединений, локальными и общими шинами, системными магистралями и отдельными линиями связи. Таким образом, при моделировании системы должна учитываться специфика компонентов и целостность всей системы.

5.1 Моделирование схемных соединений

В языке VHDL модель соединений компонентов для передачи и приема информации представляется в виде архитектурного тела с описанием интерфейса компонентов. Рассмотрим пример системы (рис. 5.1), состоящей из двух компонентов A и B, соединенных локальной шиной L_{AB}. Пусть локальная шина имеет восемь проводников. Система имеет восемь входных линий и 16 выходных.

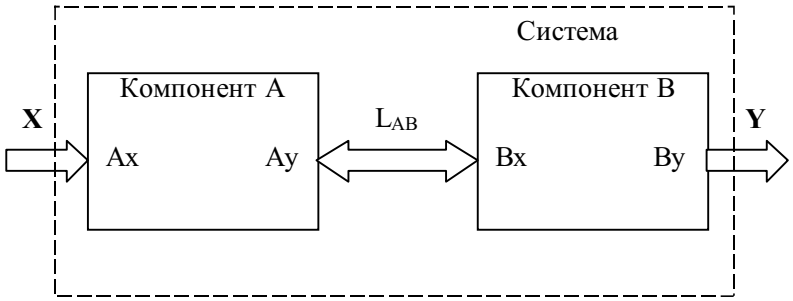


Рис. 5.1 Структурная схема системы из двух компонентов

Тогда интерфейс компонентов A и B имеет следующее описание:

```
entity A is
   port (Ax : in bit_vector(0 to 7); Ay : out bit_vector(0 to 7));
end A;
entity B is
   port (Bx : in bit_vector(0 to 7); By : out bit_vector(0 to 15));
end B;
```

Описание системы включает ее интерфейс и архитектурное тело, в котором записывается интерфейс компонента и конкретизация связей между компонентами. Для рассматриваемой в примере системы можно записать:

```
entity System is
   port (X : in bit_vector(0 to 7); Y : out bit_vector(0 to 7));
end System;
architecture System_AB of System is
  signal Lab : bit_vector(0 to 7);  ----описание линий связи
component  A is                     ---- описание компонента A
   port (Ax : in bit_vector(0 to 7); Ay : out bit_vector(0 to 7));
```

```
    end component;
    component  B is                    ---- описание компонента B
      port (Bx : in bit_vector(0 to 7); By : out bit_vector(0 to 15));
    end component;
    begin
      ----  выполнение процессов, блоков
      A port map(X,Lab);    ---- конкретизация компонента системы
      B port map(Lab,Y);    ----  конкретизация компонента системы
end System_AB;
```

Таким образом, объединение компонентов в систему моделируется описанием интерфейсов и передачей соответствующих значений сигналов компонентам.

Если для соединения используются отдельные единичные линии, то соответствующие им переменные могут иметь битовый тип. Все дальнейшие действия аналогичны. Но следует заметить, что групповое представление сигналов, когда оно соответствует схеме, выгоднее, чем битовое. Это связано с тем, что вместо одного события при одновременном изменении нескольких сигналов в группе придется обрабатывать отдельно все N событий.

В тексте программы операторы port map записываются внутри архитектурного тела за операторами процессов. Приведенная в примере запись оператора определяется позиционным сопоставлением элементов отображения портов в описании компонентов. То же самое можно сделать при помощи ключевого соответствия, например:

S1: A **port map**(Ay=>Lab, Ax=>X);

S2: B **port map**(By=>Y, Bx=>Lab);

Позиции здесь не играют роли, так как соответствие обеспечивается сопоставлением имен.

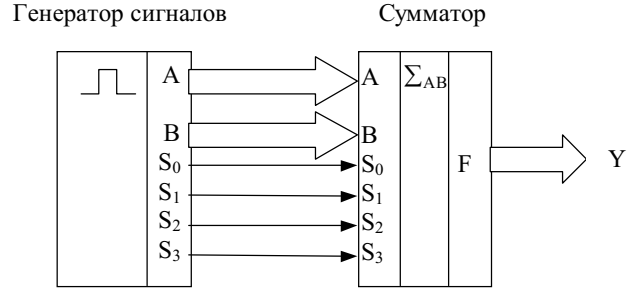


Рис. 5.2 Пример системы генератор-сумматор ератор-сумматор

При моделировании схемных связей на высоком уровне для представления информации можно использовать другие типы данных. Например, использовать перечислимый тип:

type control **is** (mov, lda, ani,…, hlt);
signal data_bus : control;

Управляющая информация, таким образом, представляется в мнемоническом виде. Такой подход облегчает моделирование и освобождает от контроля низкоуровневых деталей. В некоторых случаях применяют числовой тип, например, subtype control is integer range 0 to 31. Это удобнее по сравнению с целым типом из-за упрощения контроля ошибок.

Рассмотрим пример моделирования системы, состоящей из двух компонентов: сумматора и генератора сигналов. Принцип работы системы достаточно прост (рис. 5.2).

Генератор формирует значения для операндов A и B с сигналами управления функцией работы сумматора S0-S3. На выходе системы имеем результат Y. Компонент "сумматор" назван так условно и это видно из таблицы истинности 5.1.

Таблица 5.1

S_3	S_2	S_1	S_0	Функция F
0	0	0	0	$\overline{\mathbf{A}}$
0	0	0	1	$\overline{\mathbf{A}+\mathbf{B}}$
0	0	1	0	$\overline{\mathbf{A}}\,\mathbf{B}$

0	0	1	1	0
0	1	0	0	$\overline{\mathbf{AB}}$
0	1	0	1	$\overline{\mathbf{B}}$
0	1	1	0	$\mathbf{A} \oplus \mathbf{B}$

ПРОДОЛЖЕНИЕ ТАБЛ. 5.1

S_3	S_2	S_1	S_0	Функция F
0	1	1	1	$\mathbf{A}\overline{\mathbf{B}}$
1	0	0	0	$\overline{\mathbf{A}} + \mathbf{B}$
1	0	0	1	$\overline{\mathbf{A} + \mathbf{B}}$
1	0	1	0	**B**
1	0	1	1	**AB**
1	1	0	0	1
1	1	0	1	$\mathbf{A} + \overline{\mathbf{B}}$
1	1	1	0	**A + B**
1	1	1	1	**A**

Задержка прохождения сигналов через схему составляет 57 нс. Проект системы на языке может быть записан в виде:

```
library vector;
use std.textio.all,vector.functions.all;
entity MFChip is
   port(s3:in bit;s2:in bit;s1:in bit;s0:in bit;
        A:in bit_vector(1 to 8);B:in bit_vector(1 to 8);
        Fun:out bit_vector(1 to 8));
end MFChip;
architecture Chip of MFChip is
 begin
  Proc: process(s3,s2,s1,s0,A,B)
     begin
       if (s3='0') and (s2='0') and (s1='0') and (s0='0') then Fun<=not(A) after      57 ns;end if;
if (s3='0') and (s2='0') and (s1='0') and (s0='1') then Fun<=not(A or B) after 57 ns;end if;
if (s3='0') and (s2='0') and (s1='1') and (s0='0') then Fun<=not(A) and B after 57 ns;end if;
if (s3='0') and (s2='0') and (s1='1') and (s0='1') then Fun<="00000000" after 57 ns;end if;
if (s3='0') and (s2='1') and (s1='0') and (s0='0') then Fun<=not(A and B) after 57 ns;end if;
if (s3='0') and (s2='1') and (s1='0') and (s0='1') then Fun<=not(B) after 57 ns;end if;
if (s3='0') and (s2='1') and (s1='1') and (s0='0') then Fun<=A xor B after 57 ns;end if;
if (s3='0') and (s2='1') and (s1='1') and (s0='1') then Fun<=A and not(B) after 57 ns;end if;
if (s3='1') and (s2='0') and (s1='0') and (s0='0') then Fun<=not(A) or B after 57 ns;end if;
if (s3='1') and (s2='0') and (s1='0') and (s0='1') then Fun<=not(A or B) after 57 ns;end if;
if (s3='1') and (s2='0') and (s1='1') and (s0='0') then Fun<=B after 57 ns;end if;
if (s3='1') and (s2='0') and (s1='1') and (s0='1') then Fun<=A and B after 57 ns;end if;
if (s3='1') and (s2='1') and (s1='0') and (s0='0') then Fun<="00000001" after 57 ns;end if;
if (s3='1') and (s2='1') and (s1='0') and (s0='1') then Fun<=A or not(B) after 57 ns;end if;
if (s3='1') and (s2='1') and (s1='1') and (s0='0') then Fun<=A or B after 57 ns;end if;
if (s3='1') and (s2='1') and (s1='1') and (s0='1') then Fun<=A after 57 ns;end if;
        end process Proc;
end Chip;

library vector;
use std.textio.all,vector.functions.all;
entity Generator is
     port (O1,O2:out Bit_Vector(1 to 8);os0,os1,os2,os3:out bit);
end Generator;
architecture Gen of Generator is
   signal O11: bit_vector(1 to 8):="10101010";
   signal O21: bit_vector(1 to 8):="01010101";
   signal os01:bit:='1';signal os11:bit:='0';signal os21:bit:='1';signal os31:bit:='0';
begin
    process begin
          O11<=O11;O21<=O21;
```

```
        os01<=os01;os11<=os11;os21<=os21;os31<=os31;
        wait for 100 ns;
    end process;
    O1<=O11;O2<=O21;
    os0<=os01;os1<=os11;os2<=os21;os3<=os31;
end Gen;
entity shema is end shema;
architecture Sh of shema is
  component MFChip
    Port(s3:in bit;s2:in bit;s1:in bit;s0:in Bit;
    A,B,Fun:in bit_vector(1 to 8));
  end component;
  component Generator
    port (O1,O2:out bit_vector(1 to 8);os0,os1,os2,os3:out bit);
  end component;
  signal A,B,O1,O2,Fun:bit_vector(1 to 8);
  signal s1,s2,s3,s0,os0,os1,os2,os3:bit;
  begin
  process (O1,O2,os0,os1,os2,os3) begin
     A<=transport O1 after 5 ns;B<=transport O2 after 5 ns;
     s0<=transport os0 after 5 ns;s1<=transport os1 after 5 ns;
     s2<=transport os2 after 5 ns;s3<=transport os3 after 5 ns;
  end process;
  Generator port map(O1,O2,os0,os1,os2,os3);
  MFChip port map(s3,s2,s1,s0,A,B,Fun);
end Sh;
```

Сигналы Si представлены как физические проводники с временем распространения сигналов от одного компонента до другого 5 нс. Следует заметить, что описание схемных связей имеет определенные трудности, особенно при моделировании многозначной логики.

5.2 Моделирование многозначной логики

Моделирование реальных логических схем зачастую требует использования более, чем двух значений состояния. Большинство элементов имеют буферные схемы, переключающие выходы в высокоимпедансное состояние. Трехзначную систему {0, 1, Z} можно считать базовой в моделировании большинства компонентов и шин. В поведенческой модели для Z-состояния должно быть соответствующее действие, иначе моделирование становится невозможным. В некоторых случаях, исходя из технологических соображений, считают, что Z - логическая единица.

Состояние неопределенности имеет стандартное обозначение X и занимает в многозначной системе место перед логическим нулем. Для состояния X также должна быть определена функция обработки.

Стандартный тип логики IEEE 1164, применяемый в моделировании, имеет следующее описание:

```
type std_ulogic is ( 'U',  -- не инициализирован
     'X',  -- установить неопределенное состояние
     '0',  -- установить нулевое состояние
     '1',  -- установить единичное состояние
     'Z',  -- установить состояние высокого импеданса
     'W',  -- сбросить состояние неопределенности
     'L',  -- сбросить состояние нуля
     'H',  -- сбросить состояние единицы
     ' - ', --  состояние безразличия
);
```

В стандартном или собственном пакете или проекте описывается работа функций с состояниями многозначной логики. Только после этого допускается моделирование. Если описание функции осуществляется в пакете, то в теле пакета записывают ее представление, например:

```
function "and" (l: std_ulogic, r: std_ulogic) return UX01 is
  begin
  return(and_table(l,r));
  end "and";
```

Результат работы функции and_table записывается также в теле пакета как константа с несколькими значениями, представленными в виде таблицы истинности:

```
constant and_table : stdlogic_table := (
-----------------------------------------------------------
--  U    X    0    1    Z    W    L    H    -
-----------------------------------------------------------
  ( 'U', 'U', '0', 'U', 'U', 'U', '0', 'U', 'U'),  -- U
  ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X'),  -- X
  ( '0', '0', '0', '0', '0', '0', '0', '0', '0'),  -- 0
  ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X'),  -- 1
  ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X'),  -- Z
  ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X'),  -- W
  ( '0', '0', '0', '0', '0', '0', '0', '0', '0'),  -- L
  ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X'),  -- H
  ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X')   --  -
);
```

Тогда при выполнении команды s := a and b, где a = X и b = 1, переменная s примет значение X. Аналогично определяются таблицы истинности других логических и арифметических функций.

СПИСОК ЛИТЕРАТУРЫ

1 Советов Б. Я., Яковлев С. А. Моделирование систем. М.: Высш. шк., 1998. 319 с.

2 Альянх И. Н. Моделирование вычислительных систем. Л.: Машиностроение, 1988. 223 с.

3 Иванников А. Д. Моделирование микропроцессорных систем. М.: Энергоатомиздат, 1990. 143 с.

4 Бубенников А. Н. Моделирование интегральных микротехнологий, приборов и схем. М.: Высш. шк., 1989. 320 с.

5 Преснухин Л. Н., Воробьев И. В., Шишкевич А. А. Расчет элементов цифровых устройств. М.: Высш. шк., 1991. 526 с.

6 Системы автоматизированного проектирования в радиоэлектронике: Справочник / Под ред. И. П. Норенкова. М.: Радио и связь, 1986.

7 Армстронг Дж. Р. Моделирование цифровых систем на языке VHDL. М.: Мир, 1992. 175 с.

8 J. Armstrong. Chip Level Modelling in VHDL. Prentice Hall, 1988. P. 148.

9 Louis Baker. VHDL Programming with Advanced Topics, John Wiley & Sons. New York, 1993.

10 Gunther Lehmann, Bernhard Wunder. Schaltungsdesign mit VHDL. Manfred Selz, Poing, 1994. 317 s.

11 Abstrakte Modellierung digitaler Schaltungen (VHDL vom funktionalen Modell bis zur Gatterebene) // K. ten Hagen. Springer, 1995.

12 Peter J. Ashenden. University of Adelaide, South Australia: ftp: // ftp.cs.adelaide.edu.au / pub / VHDL-Cookbook (Mac, PC, PS); ftp: // bears.ece.ucsb.edu pub / VHDL; ftp: // du9ds4.fb9dv.uni-duisburg.de/pub/cad.

13 Щелкунов Н. Н. Микропроцессорные средства и системы. М.: Радио и связь, 1989. 288 с.

СОДЕРЖАНИЕ