БИБЛИОТЕКА СОВРЕМЕННОЙ ЭЛЕКТРОНИКИ

Джон Ф. Уэйкерли

Проектирование цифровых устройств

том I

Перевод с английского Е.В. Воронова, А.Л. Ларина

Библиотека БГУИР

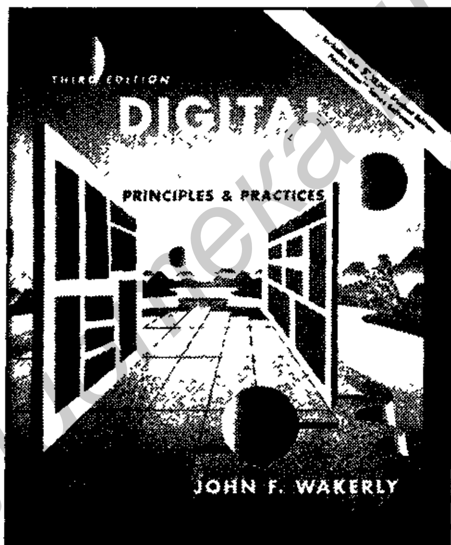
ПОСТМАРКЕТ
МОСКВА
2002

Дж. Ф. Уэйкерли
Проектирование цифровых устройств, том 1.
Москва:
Постмаркет, 2002. – 544с.

Основополагающий учебник, в котором рассмотрены все направления современной цифровой электроники. Особое внимание уделено программируемым логическим интегральным схемам (ПЛИС).

На двух прилагаемых лазерных дисках размещено программное обеспечение фирмы Xilins.

Предназначен для студентов, аспирантов и преподавателей ВУЗов, разработчиков аппаратуры.



© 2000, 1994, 1990 by Prentice Hall
© 2002, перевод на русский язык
ЗАО «Предприятие Постмаркет»

ISBN 5-901095-12-X

Библиотека БГУИР

Оглавление

3.3. КМОП-логика 114
3.3.1. Логические уровни КМОП-схем 114
3.3.2. МОП-транзисторы 115
3.3.3. Базовая схема КМОП-инвертора 116
3.3.4. КМОП-схемы И-НЕ и ИЛИ-НЕ 119
3.3.5. Коэффициент объединения по входу 120

3.3.6. Неинвертирующие вентили 122
3.3.7. КМОП-схемы И–ИЛИ-НЕ и ИЛИ–И-НЕ 123

3.10. Транзисторно-транзисторная логика 191
3.10.1. Базовый ТТЛ-вентиль И-НЕ 191
3.10.2. Логические уровни и запас помехоустойчивости 195
3.10.3. Коэффициент разветвления по выходу 196
3.10.4. Неиспользуемые входы 199
3.10.5. ТТЛ-схемы других типов 201

3.14. Эмиттерно-связанная логика 215
3.14.1. Базовая схема ЭСЛ 216
3.14.2. Семейства ЭСЛ-схем 10K/10H 219
3.14.3. Семейство ЭСЛ-схем 100K 222
3.14.4. ЭСЛ-схемы с положительным напряжением питания 222

4.7. Язык описания схем VHDL 314
4.7.1. Ход выполнения проекта 315
4.7.2. Структура программы 319
4.7.3. Типы и константы 323
4.7.4. Функции и процедуры 329
4.7.5. Библиотеки и пакеты 333
4.7.6. Элементы структурного проектирования 336
4.7.7. Элементы потокового проектирования 341
4.7.8. Элементы поведенческого проектирования 344
4.7.9. Отсчет времени и моделирование 351
4.7.10. Синтез 354

5.2. Временные соотношения в схеме 389
5.2.1. Временные диаграммы 390
5.2.2. Задержка распространения 392
5.2.3. Временные параметры 392
5.2.4. Временнуй анализ 396
5.2.5. Программные средства временнуго анализа 397
5.3. Комбинационные программируемые логические устройства 397
5.3.1. Программируемые логические матрицы 397
5.3.2. Программируемые матричные логические устройства 401
5.3.3. Универсальные матричные логические устройства 405
5.3.4. Схемы биполярных ПЛУ 407
5.3.5. Схемы ПЛУ на основе КМОП-логики 408
5.3.6. Программирование и тестирование микросхем 411

3.3. КМОП-логика

Функциональное поведение логической КМОП-схемы понять довольно просто, даже если ваши знания аналоговой электроники не особенно глубоки. Главным элементом в структуре логических КМОП-схем являются описываемые ниже МОП-транзисторы; чаще всего логические КМОП-схемы только из них и состоят. Но до рассмотрения МОП-транзисторов и логических КМОП-схем, мы должны поговорить о логических уровнях.

3.3.1. Логические уровни КМОП-схем

Абстрактные логические элементы оперируют двоичными цифрами 0 и 1. Однако реальные логические схемы имеют дело с электрическими сигналами в виде уровней напряжения. В любой логической схеме имеется диапазон напряжений (или другие состояния схемы), соответствующий логическому 0, и другой, не перекрывающийся с ним диапазон напряжений, соответствующий логической 1.

Типичная логическая КМОП-схема работает от 5-вольтового источника питания. Такая схема может интерпретировать любое напряжение в диапазоне 0–1.5 В как логический 0 и напряжение в диапазоне 3.5–5.0 В – как логическую 1. Таким образом определяются низкий уровень и высокий уровень для 5-вольтовой КМОП-логики (рис. 3.6). Не предполагается, что напряжение окажется в промежуточной области (1.5–3.5 В), кроме интервалов времени, когда сигнал переходит от одного уровня к другому; в противном случае логические значения будут не определены

Библиотека БГУИР

(то есть схема может интерпретировать их и как 0, и как 1). У КМОП-схем с другими напряжениями питания – например, 3.3 или 2.7 вольта – имеется аналогичное разделение диапазонов напряжений.

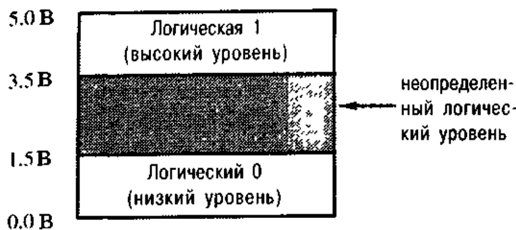


Рис. 3.6. Логические уровни для типичных логических КМОП-схем

3.3.2. МОП-транзисторы

МОП-транзистор можно представить как устройство с 3 выводами, которое действует подобно управляемому напряжением резистору. Как изображено на рис. 3.7, напряжение, приложенное ко входу, изменяет сопротивление между нижним и верхним выводами. В логических схемах МОП-транзистор работает так, что его сопротивление всегда либо очень велико (при этом транзистор «закрыт»), либо очень мало (при этом транзистор «открыт»).

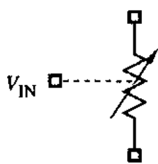
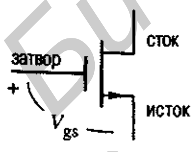


Рис. 3.7. Представление МОП-транзистора в виде резистора, сопротивление которого зависит от управляющего напряжения

Существуют два типа МОП-транзисторов: с *n*-каналом и с *p*-каналом; названия определяются типом полупроводникового материала, использованного в качестве управляемого резистора. Условное обозначение *МОП-транзистора с каналом n-типа* [*nМОП-транзистор*; *n-channel MOS* (*NMOS*) *transistor*] приведено на рис. 3.8. Выводы имеют следующие названия: *затвор* (*gate*), *исток* (*source*) и *сток* (*drain*). Глядя на условное обозначение транзистора, можно догадаться, что в нормальных условиях потенциал стока выше потенциала истока.



Резистор, управляемый напряжением: с увеличением V_{gs} значение R_{ds} уменьшается

Примечание: обычно $V_{gs} \geq 0$

Рис. 3.8. Условное обозначение МОП-транзистора с каналом *n*-типа

Напряжение между затвором и истоком (V_{gs}) у МОП-транзистора с каналом *n*-типа обычно равно нулю или положительно. Если $V_{gs} = 0$, то сопротивление между стоком и истоком (R_{ds}) очень велико и составляет, по крайней мере, мегаом (10^6 Ом) или больше. По мере увеличения V_{gs} (то есть с увеличением напряжения на затворе) R_{ds} уменьшается до очень малого значения порядка 10 Ом, а у некоторых транзисторов и меньше.

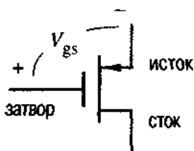
Условное обозначение *МОП-транзистора с каналом p-типа* [*pМОП-транзистор*; *p-channel MOS* (*PMOS*) *transistor*] приведено на рис. 3.9. Его функциониро-

Библиотека БГУИР

вание аналогично работе МОП-транзистора с каналом *n*-типа, за исключением того, что обычно исток имеет более высокий потенциал, чем сток, а V_{gs} равно нулю или отрицательно. Если V_{gs} равно нулю, то сопротивление между истоком и стоком (R_{ds}) очень велико. С уменьшением V_{gs} (когда напряжение на затворе становится все более отрицательным) R_{ds} уменьшается, принимая в конце концов очень малое значение.

Затвор МОП-транзистора называют изолированным, поскольку он отделен от истока и стока изолирующим материалом, имеющим очень большое сопротивление. Тем не менее, напряжение на затворе создает электрическое поле, которое увеличивает или уменьшает ток, текущий от истока к стоку. Этот «полевой эффект» дал название транзистору – «полевой транзистор».

По этой причине, независимо от напряжения на затворе, никакой ток практически не течет от затвора к истоку или от затвора к стоку. Сопротивления между затвором и другими выводами очень велики, намного больше мегаома. Ток, протекающий по этим сопротивлениям, очень мал, обычно меньше одного микроампера (мкА, 10^{-6} А), и называется *током утечки* (*leakage current*).



Резистор, управляемый напряжением: с уменьшение V_{gs} значение R_{ds} уменьшается

Примечание: обычно $V_{gs} \le 0$

Рис. 3.9. Условное обозначение МОП-транзистора с каналом *p*-типа

Само условное обозначение МОП-транзистора напоминает нам, что между затвором и двумя другими выводами нет никакого соединения. Однако изображение МОП-транзистора наводит на мысль, что затвор имеет емкостную связь с истоком и стоком. В быстродействующих схемах мощность, расходуемая при заряде и разряде этих емкостей при каждом изменении входного сигнала, составляет заметную долю потребляемой схемой мощности.

3.3.3. Базовая схема КМОП-инвертора

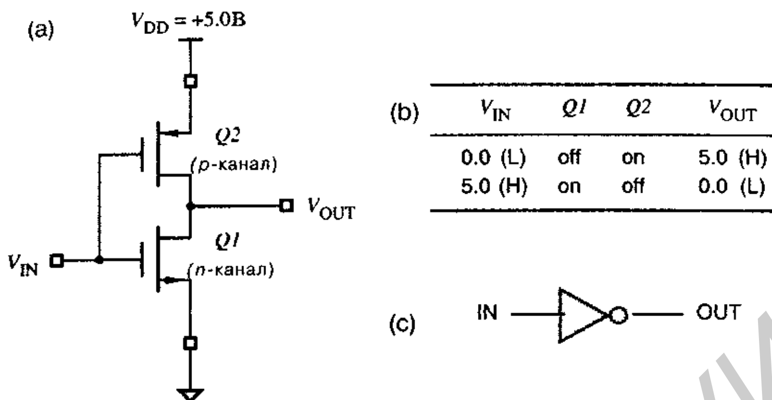
Схемы *КМОП-логики* (*CMOS logic*) образуются в результате совместного использования дополняющих друг друга *n*МОП- и *p*МОП-транзисторов. Самой простой КМОП-схемой является логический инвертор, для которого необходимо по одному транзистору каждого типа, соединенных как показано на рис. 3.10(a). Напряжение питания V_{DD} обычно может быть в диапазоне 2–6 В и наиболее часто выбирается равным 5.0 В для совместимости с ТТЛ-схемами.

В идеальном случае поведение схемы КМОП-инвертора можно описать всего лишь двумя строками таблицы, приведенной на рис. 3.10(b):

ИМПЕДАНС И СОПРОТИВЛЕНИЕ

Формально между терминами «импеданс» и «сопротивление» имеется различие, но инженеры-электрики часто используют эти термины как равнозначные. Так же поступаем в этой книге и мы.

Библиотека БГУИР



V_{IN}	Q1	Q2	V_{OUT}
0.0 (L)	off	on	5.0 (H)
5.0 (H)	on	off	0.0 (L)

Рис. 3.10. КМОП-инвертор: (a) принципиальная схема; (b) таблица, описывающая работу схемы (L – низкий уровень, H –высокий уровень, on – открыт, off – закрыт); (c) условное обозначение

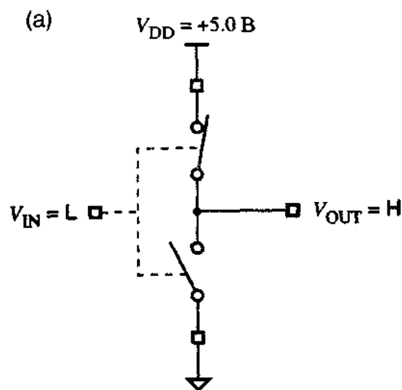
1. V_{IN} равно 0.0 В. В этом случае нижний, *n*-канальный транзистор *Q1* закрыт, так как у него напряжение V_{gs} равно 0 В, а верхний, *p*-канальный транзистор *Q2* открыт, так как у него напряжение V_{gs} имеет большое по величине отрицательное значение (–5.0 В). Поэтому сопротивление транзистора *Q2*, включенного между шиной питания (V_{DD}, + 5.0 В) и выходом (V_{OUT}), мало, и выходное напряжение равно 5.0 В.
2. V_{IN} равно 5.0 В. При этом транзистор *Q1* открыт, поскольку у него напряжение V_{gs} равно +5.0 В, а транзистор *Q2* закрыт, так как у него V_{gs} равно 0. Таким образом, транзистор *Q1* представляет собой малое сопротивление между выходом схемы и землей, и выходное напряжение равно 0 В.

Из сказанного ясно, как ведет себя логический инвертор: при напряжении 0 вольт на входе выходное напряжение равно 5 вольтам, и наоборот.

Другой способ наглядного представления работы КМОП-схемы состоит в изображении транзисторов в виде ключей. Как показано на рис. 3.11(a), *n*-канальный (нижний) транзистор заменяется ключом с нормально разомкнутым контактом, а *p*-канальный (верхний) транзистор – нормально замкнутым ключом. При подаче на вход высокого напряжения состояние каждого из ключей изменяется на состояние, противоположное первоначальному, как показано на рис. 3.11(b).

Модель с ключами позволяет нагляднее представить работу КМОП-инвертора. Как показано на рис. 3.12, для *p*- и *n*-канальных транзисторов используются различные условные обозначения, чтобы отразить логику их работы. Когда к затвору *n*-канального транзистора (*Q1*) приложено напряжение высокого уровня, он находится в «открытом» состоянии и ток течет от стока к истоку; это кажется достаточно естественным. Противоположная ситуация наблюдается в отношении *p*-канального транзистора (*Q2*). Он находится в «открытом» состоянии, когда к его затвору приложено напряжение низкого уровня; кружок на затворе указывает на инвертирование.

Библиотека БГУИР



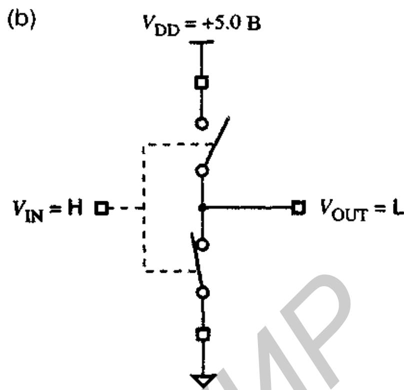


Рис. 3.11. Модель КМОП-инвертора с использованием ключей: (а) низкое входное напряжение; (b) высокое входное напряжение (L – низкий уровень, H – высокий уровень)

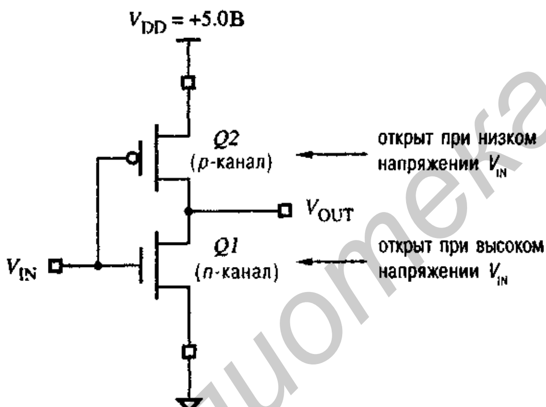


Рис. 3.12. Логика работы КМОП-инвертора

Библиотека БГУИР

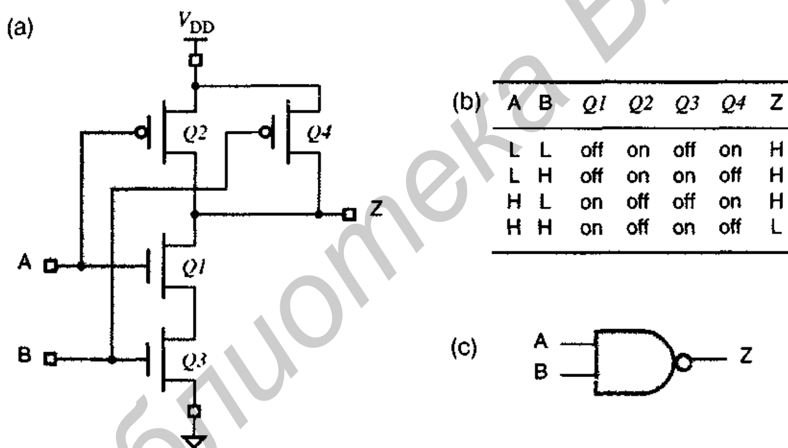
ЧТО ЗАКЛЮЧЕНО В ОБОЗНАЧЕНИЯХ?

Индекс "DD" в обозначении "V_{DD}" относится к выводам *стока* МОП-транзисторов. Это может показаться странным, так как в КМОП-инверторе напряжение источника питания V_{DD} фактически соединено с выводом *истока* *p*МОП-транзистора. Однако логические КМОП-схемы ведут свое происхождение от логических *n*МОП-схем, где источник питания *был* соединен со стоком nМОП-транзистора через резистор нагрузки, и обозначение "V_{DD}" осталось. Заметьте также, что в КМОП- и *n*МОП-схемах землю иногда обозначают как "V_{SS}". Некоторые авторы и большинство производителей схем используют обозначение "V_{CC}" для напряжения питания КМОП-схем, так как оно используется в ТТЛ-схемах, которые исторически предшествовали КМОП-схемам. Имея в виду, что можно использовать оба обозначения, начиная с параграфа 3.4 мы будем применять обозначение "V_{CC}".

3.3.4. КМОП-схемы И-НЕ и ИЛИ-НЕ

Используя *n*МОП- и *p*МОП-транзисторы можно создать схемы И-НЕ и ИЛИ-НЕ. Чтобы образовать вентиль с *k*-входами, необходимо *k p*-канальных и *k n*-канальных транзисторов. На рис. 3.13 показана 2-входовая КМОП-схема И-НЕ. Если хотя бы на одном из входов сигнал имеет низкий уровень, то выход Z через малое сопротивление «открытого» *p*-канального транзистора подключен к шине питания V_{DD}, а цепь на землю разорвана «закрытым» *n*-канальным транзистором. Если на обоих входах присутствует сигнал высокого уровеня, то цепь в сторону шины питания V_{DD} разорвана и выход Z через малое сопротивление подключен к земле. Рис. 3.14 иллюстрирует работу схемы И-НЕ на модели с ключами.

На рис. 3.15 приведена схема ИЛИ-НЕ в КМОП-исполнении. Если сигналы на обоих входах имеют низкий уровень, то выход Z через малые сопротивления «открытых» *p*-канальных транзисторов подключен к шине питания V_{DD}, а цепь на землю разорвана «закрытыми» *n*-канальными транзисторами. Если сигнал на любом из входов принимает высокий уровень, то цепь в сторону шины питания V_{DD} разорвана и выход Z через малое сопротивление подключен к земле.



A	B	Q1	Q2	Q3	Q4	Z
L	L	off	on	off	on	H
L	H	off	on	on	off	H
H	L	on	off	off	on	H
H	H	on	off	on	off	L

Рис. 3.13. Схема 2-входового КМОП-вентиля И-НЕ: (a) принципиальная схема; (b) таблица, описывающая работу схемы (L – низкий уровень, H – высокий уровень, off – закрыт, on – открыт); (c) условное обозначение

СРАВНЕНИЕ СХЕМ И-НЕ и ИЛИ-НЕ

КМОП-схемы И-НЕ и ИЛИ-НЕ имеют различные характеристики. При одной и той же площади кремниевого кристалла, транзистор с каналом *n*-типа имеет меньшее сопротивление в «открытом» состоянии, чем транзистор с каналом *p*-типа. Поэтому у последовательно включенных *k* транзисторов с *n*-каналом сопротивление в «открытом» состоянии меньше, чем у *k* транзисторов с *p*-каналом. В результате быстродействие схемы И-НЕ с *k* входами обычно выше и предпочтительнее, чем у *k*-входовой схемы ИЛИ-НЕ, и поэтому схемы И-НЕ предпочтительнее.

Библиотека БГУИР

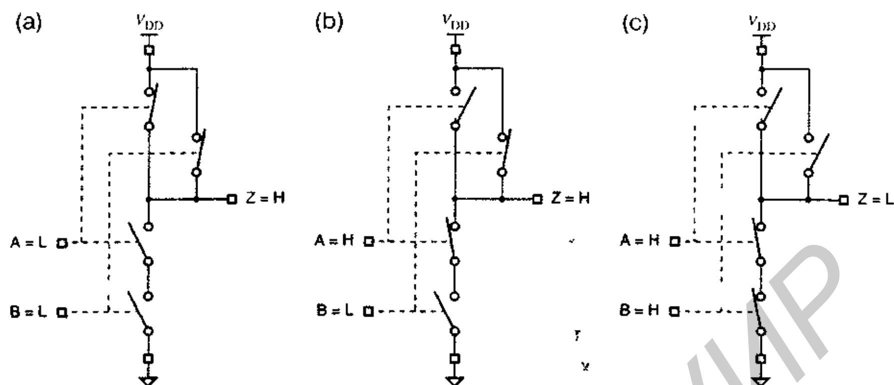
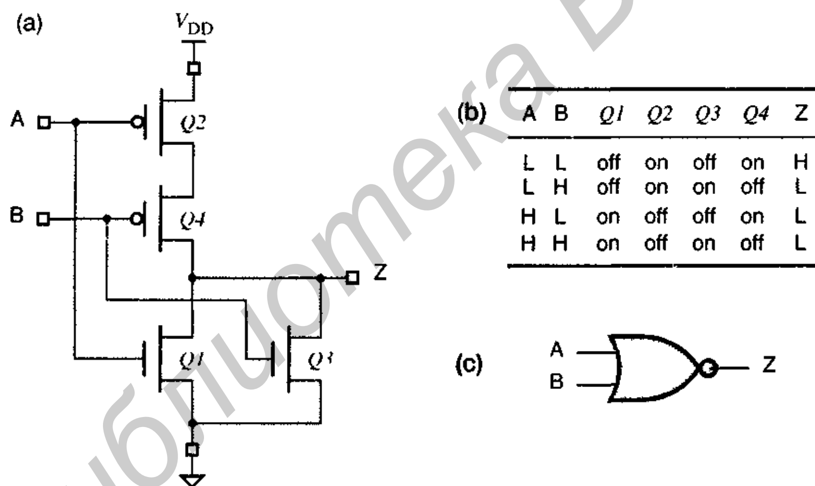


Рис. 3.14. Модель 2-входовой КМОП-схемы И-НЕ на основе ключей (a) сигналы на обоих входах имеют низкий уровень (L), (b) сигнал на одном входе имеет высокий уровень (H), (c) сигналы на обоих входах имеют высокий уровень



(b)

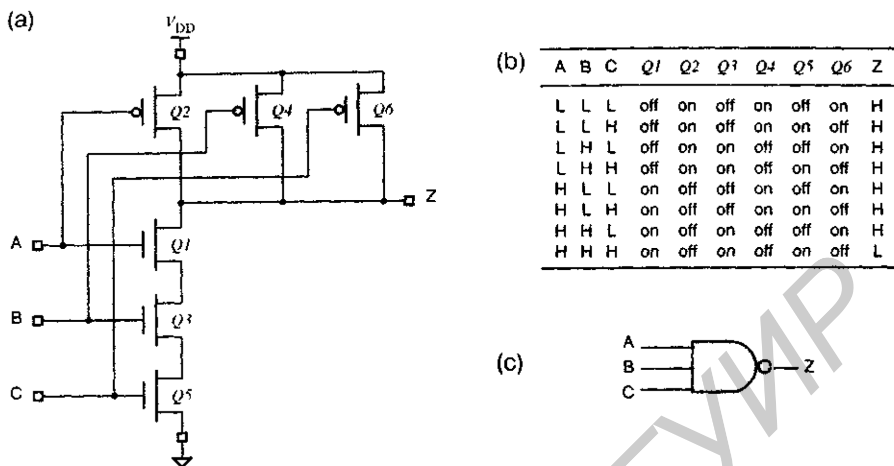
A	B	*Q1*	*Q2*	*Q3*	*Q4*	Z
L	L	off	on	off	on	H
L	H	off	on	on	off	L
H	L	on	off	off	on	L
H	H	on	off	on	off	L

Рис. 3.15. 2-входовая КМОП-схема ИЛИ-НЕ· (a) принципиальная схема, (b) таблица, описывающая работу схемы (L – низкий уровень, H – высокий уровень, off – закрыт, on – открыт), (c) условное обозначение

3.3.5. Коэффициент объединения по входу

Число входов, которые может иметь вентиль в конкретном логическом семействе, называется *коэффициентом объединения по входу* (*fan-in*). КМОП-схемы с числом входов больше двух можно очевидным способом получить путем последовательно-параллельного расширения схем, представленных на рис. 3.13 и 3.15. Например, на рис. 3 16 показана 3-входовая КМОП-схема И-НЕ.

Библиотека БГУИР



A	B	C	Q1	Q2	Q3	Q4	Q5	Q6	Z
L	L	L	off	on	off	on	off	on	H
L	L	H	off	on	off	on	on	off	H
L	H	L	off	on	on	off	off	on	H
L	H	H	off	on	on	off	on	off	H
H	L	L	on	off	off	on	off	on	H
H	L	H	on	off	off	on	on	off	H
H	H	L	on	off	on	off	off	on	H
H	H	H	on	off	on	off	on	off	L

Рис. 3.16. 3-входовая КМОП-схема И-НЕ: (a) принципиальная схема, (b) таблица, описывающая работу схемы (L – низкий уровень, H – высокий уровень, off – закрыт, on – открыт), (c) условное обозначение

В принципе можно создавать КМОП-схемы И-НЕ и ИЛИ-НЕ с очень большим числом входов. Однако на практике сопротивление последовательно включенных «открытых» транзисторов обычно ограничивает коэффициент объединения по входу у КМОП-схем числом 4 для вентилей ИЛИ-НЕ и числом 6 для вентилей И-НЕ.

По мере увеличения числа входов разработчики КМОП-схем могут увеличивать размеры последовательно включенных транзисторов для уменьшения их сопротивления и соответствующей задержки переключения. Однако с некоторого момента этот способ становится неэффективным или непрактичным. Вентиль с большим числом входов можно сделать более быстрым и меньших размеров путем последовательного включения схем с меньшим числом входов. На рис. 3.17 показана логическая структура 8-входового КМОП-элемента И-НЕ. Полная задержка прохождения сигнала через 4-входовую схему И-НЕ, 2-входовую схему ИЛИ-НЕ и инвертор обычно меньше, чем задержка в одноуровневой 8-входовой схеме И-НЕ.

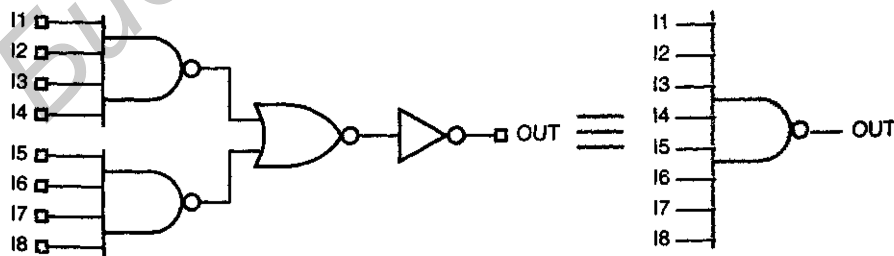


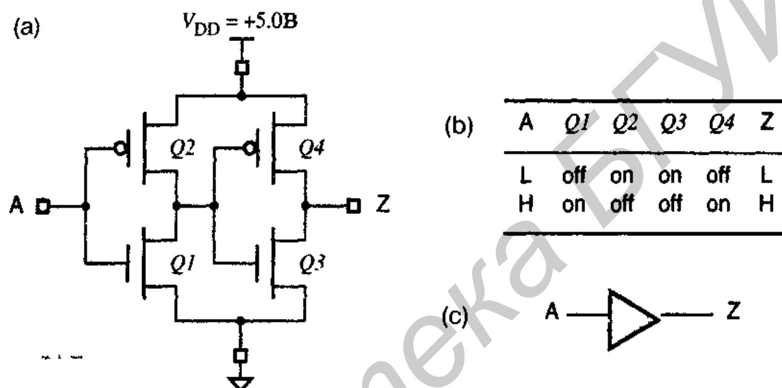
Рис. 3.17. Схемный эквивалент внутренней структуры 8-входового КМОП-элемента И-НЕ

Библиотека БГУИР

3.3.6. Неинвертирующие вентили

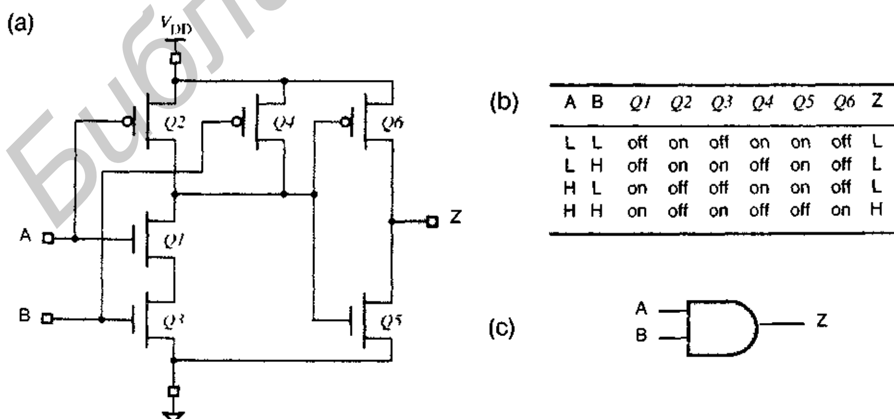
В КМОП-логике и в большинстве других логических семейств простейшими являются схемы инверторов, а следом за ними идут элементы И-НЕ и ИЛИ-НЕ. Инверсия получается «бесплатно» и обычно невозможно создать неинвертирующий вентиль с меньшим числом транзисторов, чем в простом инверторе.

Неинвертирующий КМОП-буфер, а также логические схемы И и ИЛИ получаются в результате подключения инвертора к выходу соответствующего инвертирующего элемента. Реализованные таким образом неинвертирующий буфер и логический элемент И показаны на рис. 3.18 и рис. 3.19 соответственно. Комбинация схемы, приведенной на рис. 3.15(a), с инвертором дает логический элемент ИЛИ.



A	Q1	Q2	Q3	Q4	Z
L	off	on	on	off	L
H	on	off	off	on	H

Рис. 3.18. Неинвертирующий КМОП-буфер: (a) принципиальная схема; (b) таблица, описывающая работу схемы (L – низкий уровень, H – высокий уровень, off – закрыт, on – открыт); (c) условное обозначение



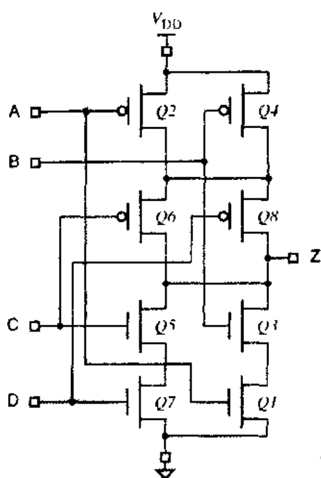
A	B	Q1	Q2	Q3	Q4	Q5	Q6	Z
L	L	off	on	off	on	on	off	L
L	H	off	on	on	off	on	off	L
H	L	on	off	off	on	on	off	L
H	H	on	off	on	off	off	on	H

Рис. 3.19. 2-входовая КМОП-схема И: (a) принципиальная схема; (b) таблица, описывающая работу схемы (L – низкий уровень, H – высокий уровень, off – закрыт, on – открыт); (c) условное обозначение

Библиотека БГУИР

3.3.7. КМОП-схемы И–ИЛИ-НЕ и ИЛИ–И-НЕ

По КМОП-технологии всего лишь на одном «слое» транзисторов можно реализовать двухуровневую логику, то есть последовательное выполнение двух логических операций. Например, на рис. 3.20(a) представлена двухвходовая КМОП-*схема И–ИЛИ-НЕ* (*AND-OR-INVERT, AOI gate*) с двухкратным объединением по ИЛИ. Таблица, описывающая работу этой схемы, приведена на рис. 3.20(b), а схема, реализующая соответствующую логическую функцию с помощью вентилей И и ИЛИ-НЕ, показана на рис. 3.21. Добавляя или удаляя транзисторы в схеме на рис. 3.20(a), можно получить функцию И–ИЛИ-НЕ с другим числом вентилей И и с другим числом входов у этих вентилей.



A	B	C	D	*Q1*	*Q2*	*Q3*	*Q4*	*Q5*	*Q6*	*Q7*	*Q8*	Z
L	L	L	L	off	on	off	on	off	on	off	on	H
L	L	L	H	off	on	off	on	off	on	on	off	H
L	L	H	L	off	on	off	on	on	off	off	on	H
L	L	H	H	off	on	off	on	on	off	on	off	L
L	H	L	L	off	on	on	off	off	on	off	on	H
L	H	L	H	off	on	on	off	off	on	on	off	H
L	H	H	L	off	on	on	off	on	off	off	on	H
L	H	H	H	off	on	on	off	on	off	on	off	L
H	L	L	L	on	off	off	on	off	on	off	on	H
H	L	L	H	on	off	off	on	off	on	on	off	H
H	L	H	L	on	off	off	on	on	off	off	on	H
H	L	H	H	on	off	off	on	on	off	on	off	L
H	H	L	L	on	off	on	off	off	on	off	on	L
H	H	L	H	on	off	on	off	off	on	on	off	L
H	H	H	L	on	off	on	off	on	off	off	on	L
H	H	H	H	on	off	on	off	on	off	on	off	L

Рис. 3.20. КМОП-схема И–ИЛИ-НЕ: (a) принципиальная схема; (b) таблица, описывающая работу схемы (L – низкий уровень, H – высокий уровень, off – закрыт, on – открыт)

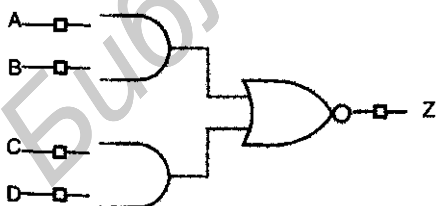
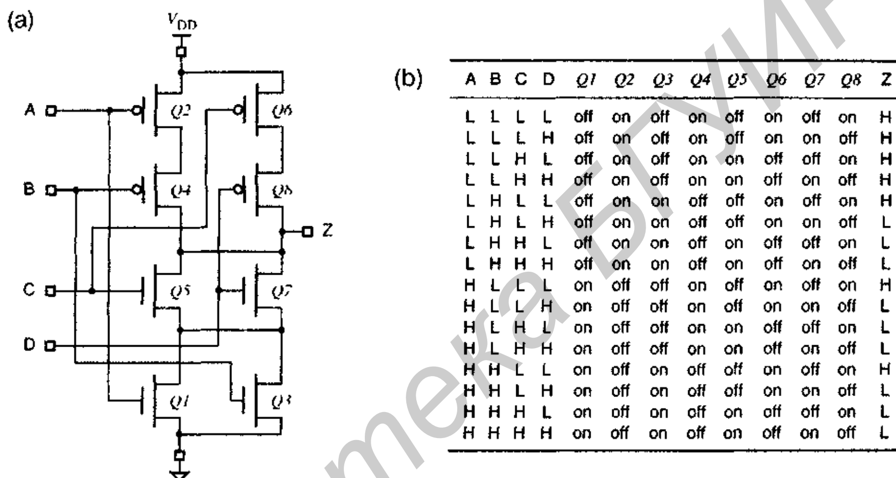


Рис. 3.21. Логическая схема КМОП-вентиля И–ИЛИ-НЕ

Содержание каждого из столбцов *Q1–Q8* в таблице на рис. 3.20(b) зависит только от входного сигнала, поданного на затвор соответствующего транзистора. Последний столбец заполняется путем проверки для каждой входной комбинации, оказывается ли выход Z соединенным через «открытый» транзистор с шиной питания V_{DD} или он соединен с землей. Обратите внимание, что при любой комбинации входных сигналов выход никогда не бывает соединен *одновременно* с шиной питания V_{DD} и с землей; в этом случае напряжение на выходе было бы где-то посередине между

низким и высоким уровнями и не соответствовало бы ни одному из логических уровней, а выходная цепь потребляла бы чрезмерно большую мощность из-за малого сопротивления между шиной питания V_{DD} и землей.

Можно также разработать схему, реализующую функцию ИЛИ–И-НЕ. Например, на рис. 3.22(a) приведена двухвходовая КМОП-*схема ИЛИ–И-НЕ* (*OR-AND-INVERT, OAI gate*), а на рис. 3.22(b) – таблица, описывающая работу этой схемы; состояния транзисторов и значения сигналов в каждом столбце определены так же, как это было сделано для КМОП-схемы И–ИЛИ-НЕ. Схема, реализующая соответствующую логическую функцию с помощью вентилей ИЛИ и И-НЕ, показана на рис. 3.23.



A	B	C	D	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Z
L	L	L	L	off	on	off	on	off	on	off	on	H
L	L	L	H	off	on	off	on	off	on	on	off	H
L	L	H	L	off	on	off	on	on	off	off	on	H
L	L	H	H	off	on	off	on	on	off	on	off	H
L	H	L	L	off	on	on	off	off	on	off	on	H
L	H	L	H	off	on	on	off	off	on	on	off	L
L	H	H	L	off	on	on	off	on	off	off	on	L
L	H	H	H	off	on	on	off	on	off	on	off	L
H	L	L	L	on	off	off	on	off	on	off	on	H
H	L	L	H	on	off	off	on	off	on	on	off	L
H	L	H	L	on	off	off	on	on	off	off	on	L
H	L	H	H	on	off	off	on	on	off	on	off	L
H	H	L	L	on	off	on	off	off	on	off	on	H
H	H	L	H	on	off	on	off	off	on	on	off	L
H	H	H	L	on	off	on	off	on	off	off	on	L
H	H	H	H	on	off	on	off	on	off	on	off	L

Рис. 3.22. КМОП-схема ИЛИ–И-НЕ: (a) принципиальная схема; (b) таблица, описывающая работу схемы (L – низкий уровень, H – высокий уровень, off – закрыт, on – открыт)

Рис. 3.23. Логическая схема КМОП-вентиля ИЛИ–И-НЕ

Быстродействие и другие электрические характеристики КМОП-схем И–ИЛИ-НЕ и ИЛИ–И-НЕ очень близки к параметрам одиночных КМОП-схем И-НЕ или ИЛИ-НЕ. В результате, эти схемы очень привлекательны, потому что могут выполнять двухуровневую логическую операцию (И–ИЛИ либо ИЛИ–И) с задержкой, соответствующей одному уровню. Большинство конструкторов цифровых устройств не затрудняют себя применением схем И–ИЛИ-НЕ в своих разработках. Однако в составе СБИС, выполненных по КМОП-технологии, эти схемы используют часто, так как многие языки описания схем могут автоматически преобразовывать схемы логики И/ИЛИ в схемы И–ИЛИ-НЕ, когда это целесообразно.

Библиотека БГУИР

3.10. Транзисторно-транзисторная логика

Наиболее часто применяемыми биполярными логическими схемами являются схемы транзисторно-транзисторной логики (ТТЛ). В действительности, существует много различных семейств ТТЛ, различающихся быстродействием, потребляемой мощностью и другими характеристиками. В качестве примера в этом параграфе выбран типичный представитель ТТЛ-семейств – схемы с диодами Шоттки и малой потребляемой мощностью серии LS (семейство LS-TTL).

У схем ТТЛ логические уровни, в основном, те же самые, что и у ТТЛ-совместимых КМОП-схем, о которых шла речь в предыдущих параграфах. При рассмотрении поведения ТТЛ-схем мы воспользуемся следующими определениями низкого и высокого уровней:

низкий уровень (LOW) 0 – 0.8 вольт,
высокий уровень (HIGH) 2.0 – 5.0 вольт.

3.10.1. Базовый ТТЛ-вентиль И-НЕ

На рис. 3.75 приведена принципиальная схема двухвходового ТТЛ-вентиля И-НЕ 74LS00 серии LS. Функция И-НЕ реализуется путем объединения диодной схемы И с инвертирующим буферным усилителем. Чтобы лучше понять работу схемы, разделим ее на три части, как показано на рисунке:

Библиотека БГУИР

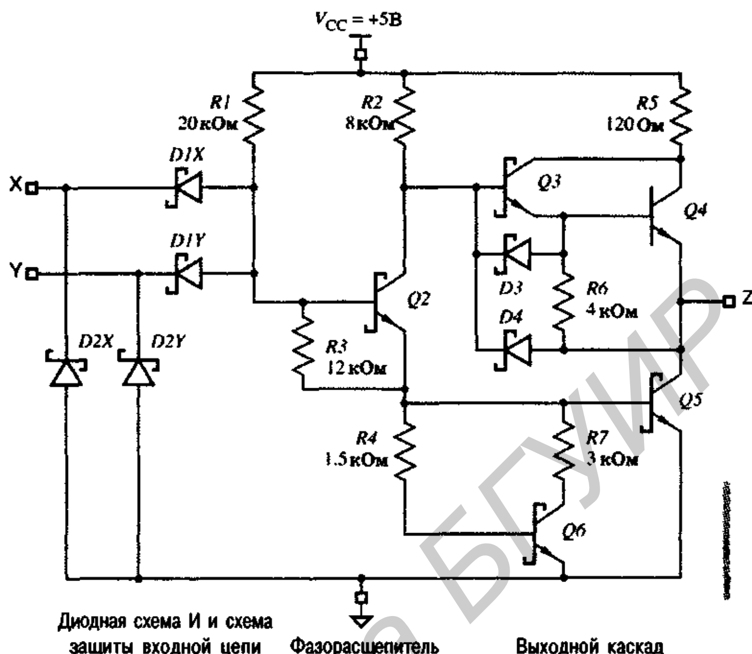


Рис. 3.75. Принципиальная схема двухвходового ТТЛ-вентиля И-НЕ серии LS

- диодная схема И и схема защиты входной цепи,
- фазорасщепитель,
- выходной каскад,

и рассмотрим их работу порознь.

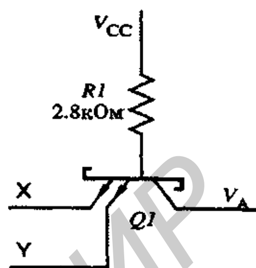
Диоды *D1X*, *D1Y* и резистор *R1* на рис. 3.75 образуют точно такую же *диодную схему И* (*diode AND gate*), что и в разделе 3.9.2. *Фиксирующие диоды* (*clamp diodes*) *D2X* и *D2Y* при нормальной работе не оказывают никакого влияния, но ограничивают нежелательные отрицательные выбросы на входах величиной, равной падению напряжения на открытом диоде. Такие отрицательные выбросы могут появляться при изменении входного напряжения от высокого уровня до низкого в результате эффектов, возникающих в линиях передачи, которые рассматриваются в параграфе 11.4.

Транзистор *Q2* и связанные с ним резисторы образуют *фазорасщепитель* (*phase splitter*), сигналы с выходов которого поступают на выходной каскад. В зависимости от того, низкий или высокий уровень напряжения V_A на выходе диодной схемы И, транзистор *Q2* закрыт или открыт.

Выходной каскад (*output stage*) содержит два транзистора *Q4* и *Q5*, один из которых в любой момент времени открыт, а другой закрыт. Выходной каскад ТТЛ-схемы иногда называют *двухтактным выходным каскадом* (*totem-pole output* или *push-pull output*). Подобно транзисторам с *p*-каналом и *n*-каналом в КМОП-схемах, транзисторы *Q4* и *Q5* осуществляют подтягивание выходного напряжения к напряжению питания или к потенциалу земли, обеспечивая на выходе сигналы высокого и низкого уровня соответственно.

ГДЕ ЖЕ ТРАНЗИСТОР *Q1*?

Обратите внимание, что в схеме на рис. 3.75 нет транзистора *Q1*, а другие транзисторы обозначены традиционным образом; в некоторых ТТЛ-схемах реально присутствует транзистор, обозначаемый *Q1*. В этих схемах для реализации логики вместо диодов *D1X* и *D1Y* применяется многоэмиттерный транзистор *Q1*. В этом транзисторе на каждый логический вход приходится один эмиттер, как показано на рисунке справа. Достаточно хотя бы на одном из эмиттеров иметь низкий уровень, чтобы транзистор был открыт, и в этом случае напряжение V_A будет соответствовать низкому уровню.



Функционирование ТТЛ-вентиля И-НЕ отражено на рис. 3.76(a). Вентиль дей ствительно реализует функцию И-НЕ. На рис. 3.76(b) и (c) приведены таблица ис тинности и условное обозначение этой схемы. Путем простого изменения числ: диодов в диодной схеме И, можно создать ТТЛ-вентиль И-НЕ с любым желаемым числом входов. Имеющиеся в продаже ТТЛ-вентили И-НЕ имеют до 13 входов ТТЛ-инвертор, представляет собой 1-входовой элемент И-НЕ, у которого отсут ствуют указанные на рис. 3.75 диоды *D1Y* и *D2Y*.

(a)

X	Y	V_A	*Q2*	*Q3*	*Q4*	*Q5*	*Q6*	V_Z	Z
L	L	≤1.05	off	on	on	off	off	2.7	H
L	H	≤1.05	off	on	on	off	off	2.7	H
H	L	≤1.05	off	on	on	off	off	2.7	H
H	H	1.2	on	off	off	on	on	≤0.35	L

(b)

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

(c)

Рис. 3.76. Функционирование ТТЛ-вентиля И-НЕ с двумя входами: (a) таблица, описывающая работу схемы [L – низкий уровень (LOW), H – высокий уровень (HIGH); on – открыт, off – закрыт]; (b) таблица истинности; (c) условное обозначение

Поскольку обычно выходные транзисторы *Q4* и *Q5* находятся в противоположных состояниях и один из них открыт (ON), а другой закрыт (OFF), у вас может возникнуть вопрос о роли резистора *R5* с сопротивлением 120 Ом в выходном каскаде. Ведь отсутствие этого резистора позволило бы схеме отдавать больший ток при высоком уровне на выходе. Это, конечно, верно с точки зрения постоянно-

Библиотека БГУИР

го тока. Однако при изменении сигнала на выходе ТТЛ-схемы от высокого уровня до низкого или наоборот имеется короткий интервал времени, когда оба транзистора могут быть открыты. Резистор *R5* служит для ограничения тока, протекающего в это время от шины питания V_{CC} к земле. Когда происходит переключение выходного каскада ТТЛ-схемы, то даже при наличии резистора с сопротивлением 120 Ом наблюдаются так называемые броски тока, в пределах которых величина тока превышает нормальные значения. Это явление подобно броскам тока, появляющимся при переключении быстродействующих КМОП-схем.

До сих пор мы считали, что сигналы поступают на вход ТТЛ-вентиля от идеального источника напряжения. На рис. 3.77 показана ситуация, когда на вход ТТЛ-схемы поступает сигнал низкого уровня с выхода другой ТТЛ-схемы. Транзистор *Q5A* в левой схеме, являющейся источником сигнала, открыт (ON), и по нему течет ток, вытекающий через диод *D1XB* из правой схемы, на вход которой подан данный сигнал. Если ток течет *в* выходную цепь ТТЛ-схемы при низком уровне сигнала на выходе, как это имеет место в нашем случае, то говорят, что *ток втекает* (*sinking current*) в эту схему со стороны ее выхода.

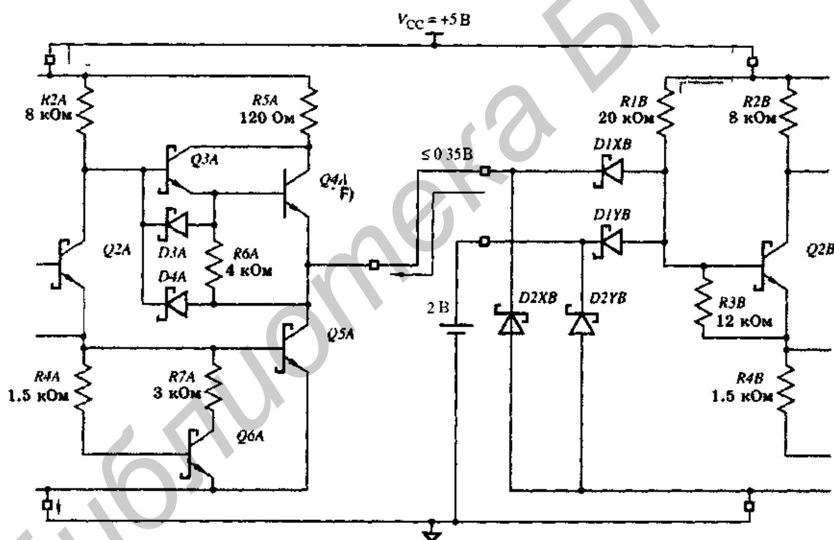


Рис. 3.77. ТТЛ-вентиль с низким уровнем сигнала на выходе, нагруженный входом другого ТТЛ-вентиля (ON – открыт, OFF – закрыт)

СНОВА БРОСКИ ТОКА

Броски тока могут проявляться в ТТЛ- и КМОП-схемах в виде помехи на шине питания и земляной шине, особенно в тех случаях, когда происходит одновременное переключение на нескольких выходах. По этой причине для надежной работы схемы требуются развязывающие конденсаторы, включаемые между шиной питания V_{CC} и землей. Эти конденсаторы должны быть распределены по всей схеме, по крайней мере, по одному в пределах дюйма или у каждой микросхемы, чтобы служить источниками тока во время переходов.

Библиотека БГУИР

На рис. 3.78 показан тот же случай с высоким уровнем сигнала, подаваемого с выхода одной схемы на вход другой. Теперь в схеме, служащей источником сигнала, транзистор *Q4A* открыт и по нему течет небольшой ток утечки смещенных в обратном направлении диодов *D1XB* и *D2XB* в схеме, на вход которой подан данный сигнал. Когда ток течет *из* ТТЛ-схемы со стороны ее выхода при высоком уровне сигнала, этот *ток* называют *вытекающим* (*sourcing current*).

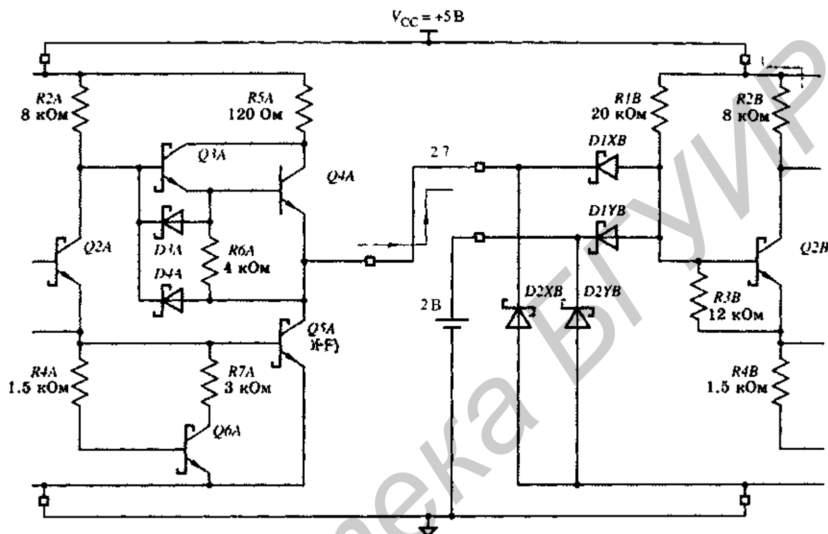


Рис. 3.78. ТТЛ-вентиль с высоким уровнем сигнала на выходе, нагруженный входом другого ТТЛ-вентиля (ON – открыт, OFF – закрыт)

3.10.2. Логические уровни и запас помехоустойчивости

В начале этого параграфа мы условились, что применительно к ТТЛ-схемам речь идет о сигналах низкого уровня при напряжениях между 0 В и 0.8 В и о сигналах высокого уровне при напряжениях между 2.0 В и 5.0 В. В действительности, уровни входных и выходных ТТЛ-сигналов можно задать более точно так же, как это было сделано для сигналов КМОП-схем:

V_{OHmin} – минимальное выходное напряжение высокого уровне (HIGH), равное 2.7 В для большинства ТТЛ-семейств;

V_{IHmin} – минимальное входное напряжение, гарантированно воспринимаемое как высокий уровень (HIGH); для всех ТТЛ-семейств оно равно 2.0 В.

V_{ILmax} – максимальное входное напряжение гарантированно воспринимаемое как низкий уровень (LOW), равное 0.8 В для большинства ТТЛ-семейств

V_{OLmax} – максимальное выходное напряжение низкого уровня (LOW), равное 0.5 В для большинства ТТЛ-семейств.

На рис. 3.79 показаны границы уровней и запас помехоустойчивости.

Согласно техническим условиям, у большинства ТТЛ-схем напряжение V_{OHmin} больше напряжения V_{IHmin} на 0.7 В, так что при высоком уровне *запас помехоус-*

Библиотека БГУИР

тойчивости по постоянному току (*DC noise margin*) ТТЛ-схем составляет 0.7 В. Это означает, что в наихудшем случае выходной высокий уровень не будет надежно восприниматься на входе как высокий уровень, когда помеха, по меньшей мере, равна 0.7 В. Однако при низком уровне напряжение V_{ILmax} превышает напряжение V_{OLmax} только на 0.3 В, так что запас помехоустойчивости по постоянному току в этом случае составляет всего лишь 0.3 В. Таким образом, ТТЛ-схемы и ТТЛ-совместимые схемы более чувствительны к шуму при низком уровне сигнала, чем при высоком.

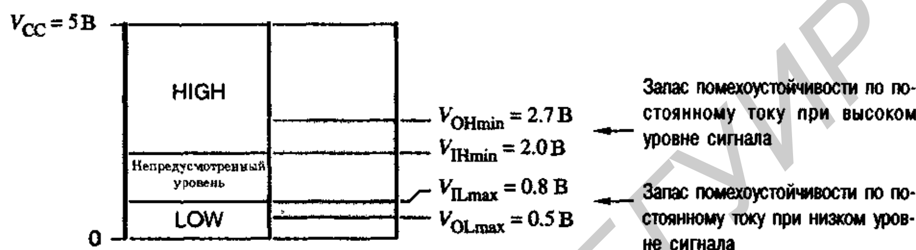


Рис. 3.79. Границы уровней и запас помехоустойчивости популярных ТТЛ-схем (семейств 74LS, 74S, 74ALS, 74AS, 74F)

3.10.3. Коэффициент разветвления по выходу

Как было сказано ранее в разделе 3.5.4, *коэффициент разветвления по выходу* (*fanout*), по определению, равен числу входов логических схем, которые могут быть подключены к одному выходу. Как было показано там же, у КМОП-схем, нагруженных входами КМОП-схем, коэффициент разветвления по выходу по постоянному току по существу не ограничен, потому что входы КМОП-схем практически не потребляют ток как при низком, так и при высоком уровне сигнала. С входами ТТЛ-схем дело обстоит иначе. В результате для ТТЛ- и КМОП-схем имеются предельные значения коэффициентов разветвления по выходу в случае их нагрузки входами ТТЛ-схем, о чем сейчас и пойдет речь.

Как и в случае КМОП-схем, ток, текущий во входной или выходной вывод ТТЛ-схемы считается положительным, если он действительно *втекает* в схему по этому выводу, и отрицательным, если *вытекает* из схемы по этому выводу. В результате, когда выход соединен с одним или большим числом входов, алгебраическая сумма всех входных токов и выходного тока равна 0.

Величина тока, протекающего через ТТЛ-вход, зависит от того, каков уровень входного напряжения – высокий или низкий, – и определяется двумя параметрами:

I_{ILmax} – максимальное значение входного тока, необходимого для получения на входе низкого уровня; вспомним схему на рис. 3.77: ток действительно течет от шины питания V_{CC} по резистору *R1B*, через диод *D1XB*, по выводу входа и через выходной транзистор *Q5A* схемы, служащей источником сигнала, на землю; так как при низком уровне ток вытека-

Библиотека БГУИР

ет из ТТЛ-входа, величина I_{ILmax} отрицательна; у большинства входов ТТЛ-схем LS-серии $I_{ILmax} = -0.4$ мА; это значение иногда называют *единичной нагрузкой при низком уровне* (*LOW-state unit load*) ТТЛ-схем серии LS;

I_{IHmax} – максимальное значение тока, необходимого для того, чтобы сигнал на входе был сигналом высокого уровня; как показано на рис. 3.78, ток течет от шины питания V_{CC} по резистору *R5A* и транзистору *Q4A* схемы, служащей источником сигнала, и *втекает* во входную цепь схемы, подключенной к данному выходу, где течет на землю через смещенные в обратном направлении диоды *D1XB* и *D2XB*; так как при высоком уровне на входе ток *втекает* по входному выводу ТТЛ-схемы, величина I_{IHmax} положительна; у большинства ТТЛ-схем серии LS $I_{IHmax} = 20$ мкА и это значение иногда называют *единичной нагрузкой при высоком уровне* (*HIGH-state unit load*).

Подобно выходам КМОП-схем, ТТЛ-схема со стороны ее выхода может быть источником тока или приемником тока некоторой величины в зависимости от того, каков уровень сигнала на выходе – высокий или низкий:

I_{OLmax} – максимальное значение тока, который может потреблять выходная цепь схемы при низком уровне выходного напряжения и при условии, что это напряжение не превосходит V_{OLmax}; поскольку ток втекает в выходную цепь, величина I_{OLmax} положительна и ее значение для большинства ТТЛ-схем серии LS равно 8 мА;

I_{OHmax} – максимальное значение тока, который может выдать выходная цепь схемы при высоком уровне выходного напряжения и при условии, что это напряжение не менее V_{OHmin}; так как ток вытекает из схемы, величина I_{OHmax} отрицательна и ее значение для большинства ТТЛ-схем серии LS равно –400 мкА.

Обратите внимание, что значение I_{OLmax} для типичных ТТЛ-схем серии LS ровно в 20 раз больше абсолютного значения I_{ILmax}. Поэтому говорят, что у ТТЛ-схем серии LS *коэффициент разветвления по выходу при низком уровне* выходного напряжения (*LOW-state fanout*) равен 20, поскольку при низком уровне сигнала выход можно нагрузить 20 входами. Точно так же абсолютное значение I_{OHmax} ровно в 20 раз больше значения I_{IHmax}, так что считается, что *коэффициент разветвления по выходу* ТТЛ-схем серии LS *при высоком уровне* выходного напряжения (*HIGH-state fanout*) также равен 20. *Результирующий коэффициент разветвления по выходу* (*overall fanout*) равен меньшему из коэффициентов разветвления при низком и при высоком уровне выходного напряжения.

Если нагрузка ТТЛ-схемы превышает номинальный коэффициент разветвления по выходу, то возможны те же опасные последствия, какие были отмечены в отношении КМОП-устройств в разделе 3.5.5, а именно может сократиться или вовсе отсутствовать запас помехоустойчивости по постоянному току, могут увеличиться время переходного процесса и задержка, а схема может перегреться.

Библиотека БГУИР

АСИММЕТРИЯ ВЫХОДА ТТЛ-СХЕМ

Хотя коэффициент разветвления по выходу у TTL-схем серии LS при высоком и при низком уровне выходного напряжения одинаков, семейство LS-TTL и другие ТТЛ-семейства имеют четко выраженную асимметрию, проявляющуюся в разных значениях втекающего и вытекающего токов: при низком уровне в выходную цепь ТТЛ-схемы серии LS может втекать ток 8 мА, а при высоком уровне из схемы может вытекать только ток, не превосходящий 400 мкА.

Из-за этой асимметрии не возникает никаких проблем, когда сигнал с выхода данной ТТЛ-схемы подается на входы других ТТЛ-схем, поскольку асимметричными являются требования в отношении входного тока ТТЛ-схем (ток I_{ILmax} велик, в то время как ток I_{IHmax} мал). Однако эта асимметрия становится ограничением, когда ТТЛ-схемы используются для включения и выключения светодиодов, реле, соленоидов или других устройств, которым требуются большие токи, составляющие часто десятки миллиампер. Схемы, в которых применяются такие устройства следует разрабатывать так, чтобы ток через управляемое устройство протекал (и оно при этом было «включено»), когда напряжение на выходе ТТЛ-схемы имеет низкий уровень, а при высоком уровне ток либо не протекал, либо был мал. Специальные ТТЛ-схемы, предназначенные для того, чтобы выполнять функции буфера или драйвера, устроены так, что при низком уровне напряжения на выходе втекающий ток может достигать 60 мА, но вытекающий ток при высоком уровне напряжения все же довольно мал (2.4 мА).

ОБОЖЖЕННЫЕ ПАЛЬЦЫ

Если втекающий ток ТТЛ- или КМОП-схемы много больше, чем I_{OLmax}, то устройство может быть повреждено, особенно если такой ток течет дольше секунды. Предположим, например, что выход ТТЛ-схемы с низким уровнем напряжения замкнут накоротко с шиной питания 5 В. Сопротивление $R_{CE(sat)}$ открытого и находящегося в режиме насыщения транзистора *Q5* в типичном выходном каскаде ТТЛ-схемы не больше 10 Ом. Таким образом, на транзисторе *Q5* должна рассеиваться мощность, равная примерно $5^2/10$ или 2.5 ватта. Не пытайтесь убедиться в этом, если вы не готовы к последствиям! Выделяющегося тепла достаточно, чтобы за очень короткое время испортить устройство (и обжечь ваш палец).

В общем случае для того, чтобы убедиться, что схема не перегружена по выходу, необходимо проверить выполнение следующих двух условий:

при высоком уровне напряжения на выходе сумма значений I_{IHmax} для всех входов, подключенных к данному выходу, не должна превышать абсолютного значения тока I_{OHmax} для схемы, являющейся источником сигнала;

при низком уровне напряжения на выходе сумма значений I_{ILmax} для всех вх дов, подключенных к данному выходу, не должна превышать абсолютно значения тока I_{OLmax} для схемы, являющейся источником сигнала.

Предположим, например, что вы спроектировали систему, в которой к выхо какой-то ТТЛ-схемы серии LS подключены десять входов ТТЛ-схем серии LS и т входа ТТЛ-схем серии S. При высоком уровне напряжения на выходе данн схемы потребуется суммарный ток, равный $10 \cdot 20$ мкА + $3 \cdot 50$ мкА = 350 мкА. З величина находится в пределах нагрузочной способности ТТЛ-схемы серии при высоком уровне, поскольку максимальный ток, отдаваемый схемой, равен 4 мкА. Но при низком уровне напряжения на выходе данной схемы требуется су марный ток $10 \cdot 0.4$ мА + $3 \cdot 2.0$ мА = 10.0 мА. Это больше, чем максимальный т втекающий в ТТЛ-схему серии LS со стороны ее выхода при низком уровне, р ный 8 мА, так что схема перегружена.

3.10.4. Неиспользуемые входы

С неиспользуемыми входами ТТЛ-схем можно поступить так же, как с входами КМО схем (см. раздел 3.5.6), а именно: неиспользуемые входы можно соединить с исполь емым входом, либо подать на них напряжение, соответствующее высокому или низ му уровню, в зависимости от реализуемой логической функции.

Сопротивление резистора, с помощью которого вход соединяется с шиной п тания или с землей, у ТТЛ-схем более критично, чем у КМОП-схем, потому ч входные токи ТТЛ-схем существенно больше, особенно при низком уровне нап жения на входе. Если сопротивление резистора слишком велико, то падение нап жения на нем может привести к тому, что потенциал входа окажется вне стандар ных диапазонов, соответствующих низкому и высокому уровням.

Рассмотрим, например, случай, когда резистор соединяет входы с землей, к показано на рис. 3.80. Через этот резистор от каждого из подключенных к не неиспользуемых входов ТТЛ-схем серии LS должен протекать ток 0.4 мА. Несм ря на это, падение напряжения на резисторе не должно превышать 0.5 В, что входное напряжение, соответствовало низкому уровню и было не больше нап жения, поступающего с выхода нормальной схемы. Если к резистору подключе n входов ТТЛ-схем серии LS, то должно выполняться неравенство:

$$n \cdot 0.4\,\text{мА} \cdot R_{pd} < 0.5\,\text{В}.$$

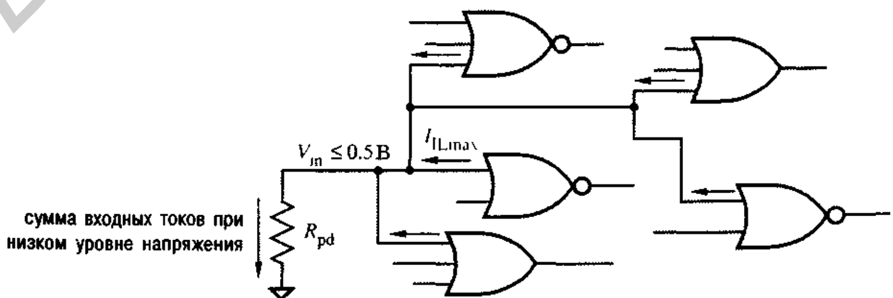


Рис. 3.80. Резистор, соединяющий входы ТТЛ-схем с землей

Библиотека БГУИР

ПЛАВАЮЩИЕ ВХОДЫ ТТЛ

Рассмотрение входных цепей ТТЛ-схем показывает, что неиспользуемые входы, оставленные не подключенными (или *плавающими*), ведут себя так, как будто на них подано напряжение, соответствующее высокому уровню: потенциал такого входа подтягивается до высокого уровня благодаря базовому резистору *R1* (см. рис. 3.75). Однако резистор *R1* осуществляет такое подтягивание менее эффективно, чем это происходит в случае, когда данный вход подключен к выходу какой-либо ТТЛ-схемы. В результате небольшой шум в цепи, создаваемый другими схемами при переключении, может оказаться достаточным, чтобы потенциал плавающего входа был ошибочно воспринят как низкий уровень на входе. Поэтому, ради надежности, неиспользуемые входы ТТЛ-схем следует подключать к стабильному источнику напряжения, соответствующего высокому или низкому уровню.

Таким образом, если резистор должен обеспечить низкий уровень для 10 входов ТТЛ-схем серии LS, то его сопротивление должно удовлетворять неравенству: $R_{pd} < 0.5 / (10 \cdot 4 \cdot 10^{-4})$ или $R_{pd} < 125$ Ом.

Точно так же оценивается сопротивление резистора, который соединяет входы с шиной питания, как показано на рис. 3.81. Этот резистор должен обеспечить протекание тока 20 мкА в каждом из неиспользуемых входов, причем входное напряжение должно быть не меньше 2.7 В, то есть не меньше напряжения высокого уровня, поступающего с выхода нормальной схемы. Поэтому, падение напряжения на этом резисторе не должно превышать 2.3 В; если к резистору подключено *n* входов ТТЛ-схем серии LS, то должно выполняться неравенство:

$$n \cdot 20\text{мкА} \cdot R_{pu} < 2.3\ \text{В}$$

Таким образом, если к резистору R_{pu} подключены 10 входов ТТЛ-схем серии LS, то $R_{pu} < 2.3 / (10 \cdot 20 \cdot 10^{-6})$ или $R_{pu} < 11.5$ кОм.

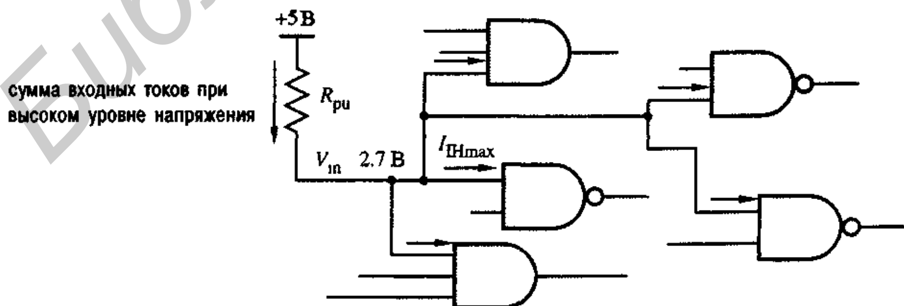


Рис. 3.81. Резистор, соединяющий входы ТТЛ-схем с шиной питания

Библиотека БГУИР

ПОЧЕМУ ПРИМЕНЯЕТСЯ РЕЗИСТОР?

Вы можете спросить: «Почему для подключения входов к шине питания или к земле применяется резистор, если непосредственное соединение с шиной питания или с землей обеспечивает хороший высокий или низкий уровень?»

Ну, непосредственное подключение к шине питания 5 В с целью поддержания на входе высокого уровня не рекомендуется потому, что возможное повышение напряжения на входе вследствие переходного процесса до значений, превосходящих 5.5 В, может повредить некоторые из ТТЛ-схем, в частности те, у которых во входной цепи включен многоэмиттерный транзистор. В этом случае резистор ограничивает ток и предотвращает повреждение схемы.

Низкий уровень в большинстве случаев прекрасно достигается путем непосредственного соединения входа с землей, без включения резистора. Повсюду в этой книге вы увидите много примеров такого рода включения. Однако в некоторых случаях резистор все же желателен для того, чтобы можно было преодолеть «постоянный» низкий уровень, обеспечиваемый этим резистором, подавая высокий уровень для целей проверки системы (см. раздел 11.2.2).

3.10.5. ТТЛ-схемы других типов

Вентиль И-НЕ является базовым элементом ТТЛ-семейства, но по той же само общей схеме можно построить вентили и других типов.

На рис. 3.82 показана принципиальная схема ИЛИ-НЕ ТТЛ-семейства серии L: Если на какой-либо из входов X или Y подать сигнал высокого уровеня, то соотве ствующий транзистор фазорасщепителя *Q2X* или *Q2Y* будет открыт, вследстви чего окажутся закрытыми транзисторы *Q3* и *Q4*, а транзисторы *Q5* и *Q6* – откры тыми, так что на выходе установится низкий уровень. Если на обоих входах имеет сигнал низкого уровня, то оба транзистора фазорасщепителя закрыты, что приве дит к появлению на выходе высокого уровня. Работа этой схемы отражена в табл це, приведенной на рис. 3.83(а).

Входные цепи, фазорасщепитель и выходной каскад ТТЛ-схемы ИЛИ-НЕ с рии LS почти совпадают с аналогичными элементами ТТЛ-схемы И-НЕ этой с рии. Отличие состоит в том, что в схеме И-НЕ семейства LS-TTL для реализац функции И применяются диоды, в то время как в схеме ИЛИ-НЕ этого семейст функция ИЛИ реализуется в фазорасщепителе параллельно включенными тра зисторами.

Входные и выходные параметры ТТЛ-схемы ИЛИ-НЕ, а также ее быстроде ствие сравнимы с аналогичными характеристиками ТТЛ-схемы И-НЕ. Однако в входовом вентиле ИЛИ-НЕ используется большее число транзисторов и резист ров, чем в *n*-входовом вентиле И-НЕ, и поэтому они дороже из-за большей площад занимаемой ими на поверхности кристалла кремния. Кроме того, внутренний т утечки ограничивает число транзисторов *Q2*, которые можно включить пара

лельно, поэтому схемы ИЛИ-НЕ имеют меньший коэффициент объединения по входу. (Наибольший коэффициент объединения по входу ТТЛ-микросхем ИЛИ-НЕ равен всего лишь 5, в то время как микросхемы И-НЕ могут иметь 13 входов.) Вследствие этого в устройствах, создаваемых на основе ТТЛ-схем, вентили ИЛИ-НЕ применяются реже, чем вентили И-НЕ.

Входные диоды и защита входной цепи

Реализация функции ИЛИ и фазорасщепитель

Выходной каскад

Рис. 3.82. Принципиальная схема двухвходового ТТЛ-вентиля ИЛИ-НЕ серии LS

(a)

X	Y	V_{AX}	Q2X	V_{AY}	Q2Y	Q3	Q4	Q5	Q6	V_Z	Z
L	L	≤ 1.05	off	≤ 1.05	off	on	on	off	off	≥ 2.7	H
L	H	≤ 1.05	off	1.2	on	off	off	on	on	≤ 0.35	L
H	L	1.2	on	≤ 1.05	off	off	off	on	on	≤ 0.35	L
H	H	1.2	on	1.2	on	off	off	on	on	≤ 0.35	L

(b)

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

(c)

Рис. 3.83. Двухвходовой ТТЛ-вентиль ИЛИ-НЕ серии LS: (a) таблица, описывающая работу схемы [L – низкий уровень (LOW), H – высокий уровень (HIGH); on – открыт, off – закрыт}; (b) таблица истинности; (c) условное обозначение

Библиотека БГУИР

Для ТТЛ-схем наиболее «естественными» являются инвертирующие вентили такие как И-НЕ и ИЛИ-НЕ. Неинвертирующие ТТЛ-схемы имеют дополнительны инвертирующий каскад, находящийся, как правило, между входным каскадом фазорасщепителем. В результате неинвертирующие ТТЛ-схемы обычно содержа большее число элементов и функционируют медленнее, чем инвертирующие схе мы, на базе которых они созданы.

Подобно КМОП-схемам у ТТЛ-схем может быть выход с тремя состояниями Такие схемы имеют вход «разрешение выхода» или «запрещение выхода»: сигна действующий на этом входе, позволяет перевести выход в высокоомное состоя ние, когда оба выходных транзистора закрыты.

Некоторые ТТЛ-схемы имеют *выход с открытым коллектором* (*open-collecto output*). В таких схемах полностью отсутствует верхняя половина выходного каскада изображенного на рис. 3.75, так что подтягивание потенциала выхода до высоког уровня обеспечивается только с помощью внешнего резистора. Применения ТТЛ схем с открытым коллектором и необходимые в этом случае расчеты подобны тем что были приведены для КМОП-схем с открытым стоком.

*3.14. Эмиттерно-связанная логика

Ключом к уменьшению задержки распространения в биполярных логических схе
мах является предотвращение насыщения находящихся в них транзисторов. В раз
деле 3.9.5 уже было объяснено, что диоды Шоттки предотвращают насыщени
транзисторов в ТТЛ-схемах. Однако насыщение можно также предотвратить, ис
пользуя совершенно другой принцип, а именно – с помощью схем, называемы
логическими схемами на переключателях тока (*current-mode logic, CML*) ил
схемами *эмиттерно-связанной логики* (*ЭСЛ*; *emitter-coupled logic, ECL*).

В отличие от других логических схем, рассматриваемых в этой главе, ЭСЛ
схемы не обеспечивают большого различия напряжений между низким и высо
ким уровнями. Эти схемы дают небольшой перепад напряжения, меньше вольта,
внутри у них происходит переключение тока между двумя возможными путями
зависимости от значения сигнала на выходе.

Первое семейство логических схем ЭСЛ было представлено фирмой Genera
Electric в 1961 году. Идея вскоре получила развитие и фирмой Motorola и другим
были созданы популярные до настоящего времени семейства ЭСЛ-схем 10K
100K. Эти схемы очень быстры; задержка распространения у них менее 1 нс.
последнего семейства ЭСЛ-схем ECLinPS (буквально: пикосекундная ЭСЛ; ECL i
picoseconds) максимальные задержки меньше 0.5 нс (500 пс), включая задержк
сигнала при включении и выключении ИС. На всем протяжении развития техноло
гии цифровых схем ЭСЛ-схемы того или иного типа всегда оказывались самым
быстродействующими среди логических схем, выпускавшихся в дискретном ис
полнении.

Однако сегодня ЭСЛ-схемы не столь популярны, как КМОП- и ТТЛ-схемы, глав
ным образом потому, что они потребляют намного большую мощность. Факти
чески из-за большой потребляемой мощности создание супер-ЭВМ на схемах ЭС
типа Cray-1 и Cray-2 представляло собой столь же сложную задачу для техник
охлаждения, что и для цифровой электроники. Кроме того, у ЭСЛ-схем велик
значение такого параметра, как произведение задержки на потребляемую мощ
ность, для них не удается обеспечить большую степень интеграции, а сигнал

Библиотека БГУИР

имеют настолько короткие фронты, что в большинстве случаев требуется учитывать эффекты, возникающие в линии передачи; наконец, схемы ЭСЛ непосредственно не совместимы с ТТЛ- и КМОП-схемами. Тем не менее, ЭСЛ-схемы попрежнему находят свое место в качестве логических и интерфейсных элементов в сверхбыстродействующих устройствах связи, включая волоконно-оптические интерфейсы приемопередатчиков для гигабитной сети Ethernet и сетей с асинхронной передачей данных (ATM).

*3.14.1. Базовая схема ЭСЛ

Основная идея применения токового переключателя в логической схеме показана на рис. 3.88 на примере схемы инвертора/буфера. У этой схемы два выхода: инвертирующий выход (OUT1) и неинвертирующий выход (OUT2). Два транзистора образуют *дифференциальный усилитель* (*differential amplifier*) с общим эмиттерным резистором. Постоянные напряжения в этом примере имеют следующие значения: $V_{CC} = 5.0$ В, $V_{BB} = 4.0$ В и $V_{EE} = 0$ В; низкий и высокий уровни на входе, по определению, равны 3.6 В и 4.4 В. В действительности данная схема дает на выходе в качестве низкого и высокого уровней напряжения, которые на 0.6 В выше (4.2 В и 5.0 В соответственно), но в реальных ЭСЛ-схемах это скорректировано.

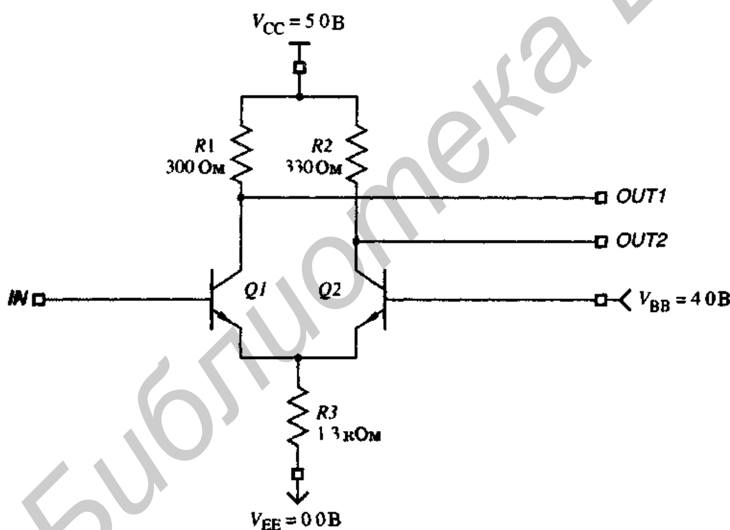


Рис. 3.88. Базовая схема инвертора/буфера ЭСЛ при высоком уровне сигнала на входе (LOW – низкий уровень, HIGH – высокий уровень; on – открыт, OFF – закрыт)

Пусть напряжение V_{IN} соответствует высокому уровню, как показано на рисунке; тогда транзистор *Q1* открыт, но не насыщен, а транзистор *Q2* закрыт. Это достигается путем аккуратного выбора сопротивлений резисторов и уровней напряжения. Благодаря наличию резистора *R2* напряжение V_{OUT2} поднимается до 5.0 В (высокий уровень). Можно показать, что падение напряжения на резисторе

Библиотека БГУИР

R1 составляет около 0.8 В, так что напряжение V_{OUT1} равно примерно 4.2 В (низки уровень).

Когда напряжение V_{IN} соответствует низкому уровню, как показано на ри 3.89, транзистор *Q2* открыт, но не насыщен, а транзистор *Q1* закрыт. При это напряжение V_{OUT1} благодаря наличию резистора *R1* поднимается до 5.0 В, а напр жение V_{OUT2}, как нетрудно показать, равно примерно 4.2 В.

Выходы этого инвертора называют *дифференциальными выходами* (*differenti outputs*), потому что они всегда комплементарны, и значение выходного сигнала мож но определить по разности между выходными напряжениями ($V_{OUT1} - V_{OUT2}$), а не п их абсолютным значениям. Другими словами, можно считать, что выходной сигна равен 1, если $(V_{OUT1} - V_{OUT2}) > 0$, и равен 0, если $(V_{OUT1} - V_{OUT2}) < 0$. Входную цеп можно сделать такой, чтобы на каждый логический вход сигнал поступал по дву проводам и его логическое значение определялось по указанному правилу; таки входы называются *дифференциальными входами* (*differential inputs*).

Дифференциальные сигналы используются в большинстве устройств на осно ве ЭСЛ-схем, выполняющих функции «интерфейса» и «размножителя тактовы сигналов» из-за их малой асимметрии и высокой помехоустойчивости. Малая асим метрия обусловлена тем, что время перехода от 0 к 1 или от 1 к 0 слабо зависит о порогових напряжений, которые могут изменяться с температурой или от схемы схеме: момент перехода зависит только от того, когда одно напряжение измени знак относительно другого. Точно так же, «относительность» определения 0 и 1 обеспечивает превосходную помехоустойчивость, так как шум, создаваемый ко лебаниями напряжения питания или поступающий от внешних источников, как правило, является *синфазным сигналом* (*common-mode signal*), который действуе на обоих дифференциальных входах одновременно, и при этом значение разности остается неизменным.

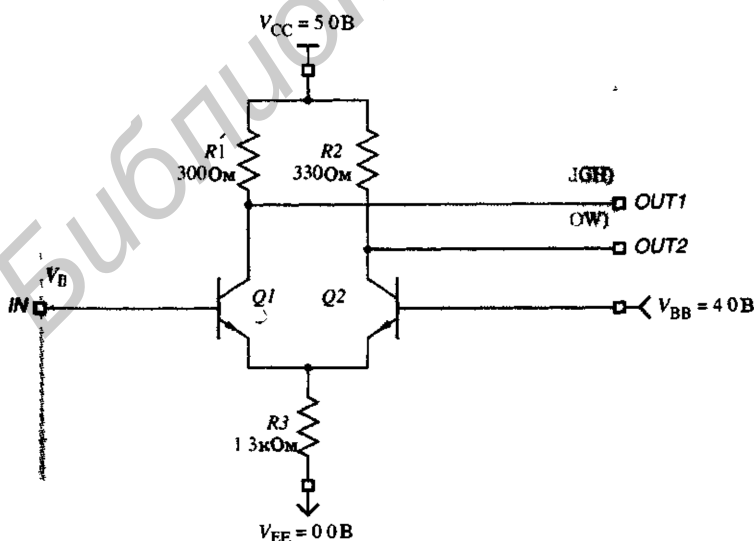
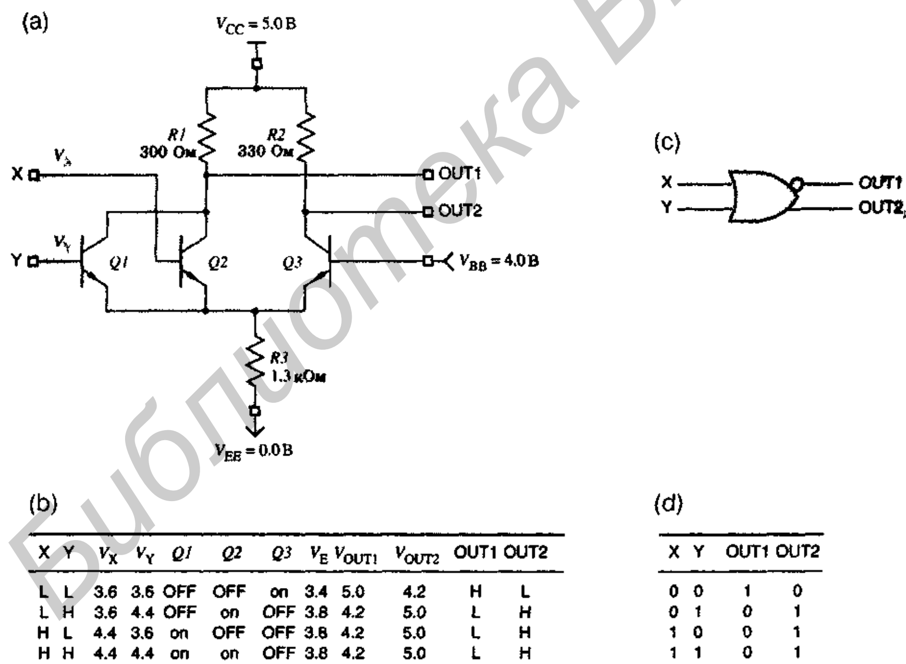


Рис. 3.89. Базовая схема инвертора/буфера ЭСЛ при низком уровне сигнала на входе (LOW – низкий уровень, HIGH – высокий уровень, on –открыт, OFF – закрыт)

Библиотека БГУИР

Можно, конечно, определять логическое значение, учитывая абсолютный уровень сигнала на одном входе, который в этом случае называется *несимметричным входом* (*single-ended input*). Чтобы избежать очевидного удвоения числа сигнальных линий, в большинстве «логических» приложений ЭСЛ-схем сигналы передаются по одному проводу на несимметричный вход. Базовая схема ЭСЛ-инвертора, изображенная на рис. 3.89, имеет несимметричный вход. Кроме того, у нее всегда имеются оба «выхода», поэтому фактически схема является инвертирующим или неинвертирующим буфером, в зависимости от того, каким выходом мы пользуемся: OUT1 или OUT2.

Чтобы реализовать ту или иную логику на основе базовой схемы (рис. 3.89), параллельно с транзистором *Q1* включаются дополнительные транзисторы, подобно тому, как это делается в ТТЛ-схеме ИЛИ-НЕ. На рис. 3.90 в качестве примера приведена 2-входовая ЭСЛ-схема ИЛИ/ИЛИ-НЕ. Если на каком-либо входе имеется высокий уровень, то соответствующий входной транзистор открыт и напряжение V_{OUT1} соответствует низкому уровню (выход ИЛИ-НЕ). В то же самое время транзистор *Q3* закрыт и напряжение V_{OUT2} соответствует высокому уровню (выход ИЛИ).



(b)

X	Y	V_X	V_Y	*Q1*	*Q2*	*Q3*	V_E	V_{OUT1}	V_{OUT2}	OUT1	OUT2
L	L	3.6	3.6	OFF	OFF	on	3.4	5.0	4.2	H	L
L	H	3.6	4.4	OFF	on	OFF	3.8	4.2	5.0	L	H
H	L	4.4	3.6	on	OFF	OFF	3.8	4.2	5.0	L	H
H	H	4.4	4.4	on	on	OFF	3.8	4.2	5.0	L	H

(d)

X	Y	OUT1	OUT2
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	1

Рис. 3.90. Схема 2-входового ЭСЛ-вентиля ИЛИ/ИЛИ-НЕ: (a) принципиальная схема; (b) таблица, описывающая работу схемы (L – низкий уровень, H – высокий уровень; on – открыт, OFF – закрыт); (c) условное обозначение; (d) таблица истинности

Вспомним, что напряжения входных уровней для инвертора/буфера, по определению, составляют 3.6 В и 4.4 В, в то время как получаемые на выходе напряжения равны 4.2 В и 5.0 В. Очевидно, что это вызывает затруднения. Для того, чтобы

Библиотека БГУИР

выходные напряжения соответствовали входным уровням, можно было бы посл- довательно с каждым выходом включить диоды, понизив тем самым напряжени на 0.6 В, но при этом остается другая проблема – мал коэффициент разветвлени по выходу. При высоком уровне на выходе данной схемы в базы входных транзи- торов схем, подключенных к этому выходу, течет ток, что приводит к дополнитель- ному падению напряжения на резисторах *R1* или *R2* и уменьшению выходног напряжения (и у нас нет достаточного запаса помехоустойчивости, чтобы преодо- леть это). Перечисленные проблемы решены в серийных ЭСЛ-схемах, таких ка 10K, которые описаны ниже.

*3.14.2. Семейства ЭСЛ-схем 10K/10H

Микросхемы самого популярного сегодня ЭСЛ-семейства имеют обозначени состоящее из 5 цифр в виде «10xxx» (например, 10102, 10181, 10209), поэтому ег обычно называют *ЭСЛ-семейством 10K* (*ECL 10K*). У схем этого семейства име- ются некоторые усовершенствования по сравнению с базовой ЭСЛ-схемой, опи- санной выше:

- Эмиттерные повторители, включенные на каждом из выходов, сдвигают выход- ные напряжения настолько, что они соответствуют входным уровням и обеспе- чивают большую нагрузочную способность по току, до 50 мА на выход.
- Напряжение смещения V_{BB} обеспечивается внутренней цепью, поэтому от- дельного, внешнего источника питания не требуется.
- Семейство разработано так, чтобы оно работало с $V_{CC} = 0$ В (земля) и $V_{EE} = -5.2$ В Как правило, при наблюдении сигналов относительно земли уровень шума оказывается меньшим, чем в случае, когда уровни сигналов отсчитываются относительно шины питания. В ЭСЛ-схемах уровни логических сигналов оп- ределяются относительно напряжения V_{CC} на шине питания, поэтому разра- ботчики семейства решили сделать это напряжение равным 0 В («чистая» земля) и использовать в качестве V_{EE} отрицательное напряжение. Помехи, воз- никающие в цепи питания и появляющиеся на шине V_{EE}, являются «синфаз- ным сигналом», который ослабляется в дифференциальном усилителе благо- даря конфигурации его входной цепи.
- Микросхемы с префиксом 10H (*ЭСЛ-семейство 10H*; *ECL 10H family*) явля- ются полностью скорректированными по напряжению, поэтому они будут нормально работать при напряжениях питания V_{EE}, отличающихся от –5.2 В; этот режим работы будет рассмотрен в разделе 3.14.4.

На рис. 3.91 показано, как для ЭСЛ-семейства 10K определены низкий и высо- кий уровни. Обратите внимание: несмотря на то, что напряжение питания отрица- тельно, названия уровней «низкий» (LOW) и «высокий» (HIGH) приняты для ЭСЛ- схем в соответствии с *алгебраически* низким и высоким напряжениями соответственно.

Запас помехоустойчивости по постоянному току у схем ЭСЛ-семейства 10K намного меньше, чем у КМОП- и ТТЛ-схем, всего лишь 0.155 В при низком уровне и 0.125 В при высоком уровне. Однако ЭСЛ-схемы не нуждаются в таком же боль- шом запасе помехоустойчивости, как схемы КМОП- и ТТЛ-семейств. В отличие от КМОП- и ТТЛ-схем, ЭСЛ-схемы при переключении создают очень малые помехи

Библиотека БГУИР

на шине питания и на шине земли; токи, потребляемые ЭСЛ-схемами, остаются неизменными, так как в них просто происходит переключение тока с одного пути на другой. К тому же в ЭСЛ-схемах выходные сопротивления эмиттерных повторителей в любом состоянии очень малы, и внешнему источнику трудно создать помеху в сигнальной линии, подключенной к такому выходу.

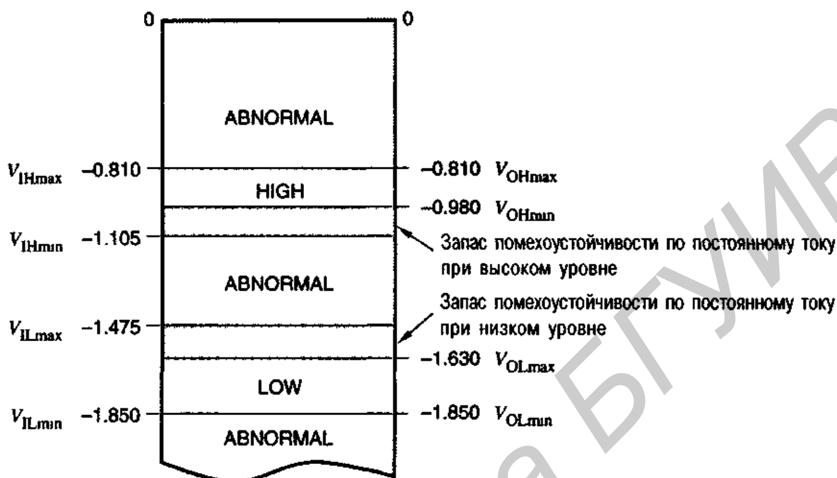


Рис. 3.91. Логические уровни ЭСЛ-семейства 10K (ABNORMAL – непредусмотренный уровень, HIGH – высокий уровень, LOW – низкий уровень)

На рис. 3.92(a) приведена схема одного из четырех вентилей **ИЛИ/ИЛИ-НЕ** в микросхеме 10102. С помощью резисторов *R1* и *R2* на входах обеспечивается присутствие на любом из них низкого уровня, если этот вход остается ни к чему не подключенным. Номиналы компонентов в цепи смещения выбраны так, чтобы получить напряжение $V_{BB} = -1.29$ В, необходимое для правильной работы дифференциального усилителя. У выходных транзисторов, включенных по схеме эмиттерного повторителя, потенциал каждого эмиттера ниже потенциала соответствующей базы на величину напряжения на открытом диоде; этим достигается требуемый сдвиг выходного уровня. Таблица на рис. 3.92(b) описывает работу схемы.

Выходы эмиттерных повторителей в схемах ЭСЛ-семейства 10K, требуют подключения внешних резисторов, как показано на рисунке. Применение в данном случае внешних резисторов, а не внутренних имеет серьезные основания. Скорость нарастания и спада сигналов на выходе ЭСЛ-схем при переключении настолько велика (типичное значение времени переключения составляет 2 нс), что любое соединение длиной в несколько дюймов необходимо рассматривать как длинную линию, которая должна иметь на концах согласованные нагрузки (см. параграф 11.4). Дело здесь даже не в мощности, которая бесполезно рассеивалась бы на внутреннем резисторе: применяя ЭСЛ схемы семейства 10K, разработчик имеет возможность выбрать внешний резистор так, чтобы одновременно обеспечить нагрузку для эмиттерного повторителя и согласовать длинную линию на конце. В простейшем случае при коротких соединениях подходящей нагрузкой является резистор с сопротивлением от 270 Ом до 2 кОм, включенный между каждым из выходов и шиной питания V_{EE}.

Библиотека БГУИР

(b)

X	Y	V_X	V_Y	Q1	Q2	Q3	V_E	V_{C2}	V_{C3}	V_{OUT1}	V_{OUT2}	OUT1	OUT2
L	L	–1.8	–1.8	OFF	OFF	on	–1.9	–0.2	–1.2	–0.9	–1.8	H	L
L	H	–1.8	–0.9	OFF	on	OFF	–1.5	–1.2	–0.2	–1.8	–0.9	L	H
H	L	–0.9	–1.8	on	OFF	OFF	–1.5	–1.2	–0.2	–1.8	–0.9	L	H
H	H	–0.9	–0.9	on	on	OFF	–1.5	–1.2	–0.2	–1.8	–0.9	L	H

(c)

X	Y	OUT1	OUT2
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	1

(d)

Рис. 3.92. Двухвходовая ЭСЛ-схема ИЛИ/ИЛИ-НЕ серии 10K: (a) принципиальная схема; (b) таблица, описывающая работу схемы (L – низкий уровень, H – высокий уровень; OFF – закрыт, on – открыт); (c) таблица истинности; (d) условное обозначение

У типичного вентиля ЭСЛ-семейства 10K задержка распространения равна 2 нс и сопоставима с задержкой ТТЛ-схем 74AS. Если выходы вентиля семейства 10K оставить ни к чему не подключенными, то он потребляет мощность около 26 мВт, что также сравнимо с мощностью, потребляемой ТТЛ-схемой 74AS, которая составляет примерно 20 мВт. Однако оконечная нагрузка, необходимая для ЭСЛ-схемы 10K, также потребляет мощность от 10 до 150 мВт на каждый выход в зависимости от типа нагрузки, тогда как выход ТТЛ-схемы 74AS может требовать, а может и не требовать оконечной нагрузки, потребляющей мощность, в зависимости от физических характеристик приложения.

Библиотека БГУИР

*3.14.3. Семейство ЭСЛ-схем 100K

Микросхемы *ЭСЛ-семейства 100K* (*ECL 100K family*) имеют обозначение, состоящее из 6 цифр в виде «100xxx» (например, 100101, 100117, 100170), но в общем случае реализуемые ими функции отличаются от функций, выполняемых схемами 10K с теми же номерами. Семейство 100K имеет следующие существенные отличия от семейства 10K:

- Меньшее напряжение питания $V_{EE} = -4.5$ В.
- Другие логические уровни, как следствие другого напряжения питания.
- Меньшие значения задержки распространения: типичное значение 0.75 нс.
- Меньшие значения времени перехода: типичное значение 0.70 нс.
- Бóльшая потребляемая мощность: типичное значение 40 мВт на вентиль.

*3.14.4. ЭСЛ-схемы с положительным напряжением питания

Мы уже говорили о достоинствах ЭСЛ-схем с отрицательным напряжением питания ($V_{EE} = -5.2$ В или -4.5 В), которые заключаются в их нечувствительности к шумам, но с другой стороны схемам с отрицательным напряжением питания присущ большой недостаток: в самых популярных сегодня КМОП- и ТТЛ-схемах, в специализированных интегральных схемах и в микропроцессорах применяется положительное напряжение питания, обычно равное +5.0 В и имеющее тенденцию к понижению до +3.3 В. Поэтому для систем, содержащих как ЭСЛ-схемы, так и КМОП/ТТЛ-схемы, требуются два источника питания. Кроме того, сопряжение обычных ЭСЛ-схем 10K или 100K с отрицательными логическими уровнями и КМОП/ТТЛ-схем с положительными уровнями требует наличия специальных схем преобразования уровней, на которые нужно подавать оба напряжения питания.

В *ЭСЛ-схемах с положительным напряжением питания* (*positive ECL*; *PECL*, произносится "peckle") используется стандартное напряжение питания +5.0 В. Обратите внимание, что в ЭСЛ-схеме 10K, изображенной на рис. 3.92, нет ничего, что требовало бы обязательного заземления шины V_{CC} и соединения с источником питания -5.2 В шины V_{EE}. Схема будет работать точно так же, если заземлить шину V_{EE}, а шину V_{CC} соединить с источником напряжения +5.2 В.

Таким образом, ЭСЛ-схемы с положительным напряжением питания представляют собой не что иное, как обычные ЭСЛ-схемы с заземленной шиной V_{EE} и поданым на шину V_{CC} напряжением +5.0 В. При этом напряжение между шинами V_{EE} и V_{CC} немного меньше, чем у обычной ЭСЛ-схемы 10K, и больше, чем у обычной ЭСЛ-схемы 100K, но схемы серий 10H и 100K будут хорошо работать с немного большими или немного меньшими напряжениями питания.

Так же, как и у ЭСЛ-схем, логические уровни сигналов ЭСЛ-схем с положительным напряжением питания привязаны к потенциалу шины V_{CC}, так что высокому уровню сигналов в этих схемах соответствует напряжение, примерно равное $V_{CC} - 0.9$ В, а низкому уровню – напряжение, примерно равное $V_{CC} - 1.7$ В, или около 4.1 В и 3.3 В соответственно при номинальном напряжении питания $V_{CC} = 5$ В. Так как эти уровни привязаны к значению V_{CC}, они смещаются вверх и вниз при любых изменениях V_{CC}. Таким образом, при проектировании устройств на ЭСЛ-

Библиотека БГУИР

схемах с положительным напряжением питания особенно пристального внимания требует разводка питания, чтобы предотвратить появление помех на шине V_{CC}, искажающих логические уровни сигналов, передаваемых и получаемых этими схемами.

Вспомним теперь, что с выходов ЭСЛ-схем можно снимать дифференциальные сигналы, а у самих схем могут быть дифференциальные входы. Дифференциальный вход относительно слабо реагирует на напряжение, одновременно действующее на обоих входах, и чувствителен только к разности напряжений на них. Поэтому при использовании ЭСЛ-схем с положительным напряжением питания для ослабления влияния помех, упомянутых в предыдущем абзаце, довольно эффективно можно применять дифференциальные сигналы.

Вполне естественно обеспечить КМОП-схемы дифференциальными входами и выходами, совместимыми с ЭСЛ-схемами с положительным питанием, позволяя тем самым осуществлять прямое сопряжение между КМОП-схемами и такими устройствами, как волоконно-оптический приемопередатчик, которому требуются уровни сигналов ЭСЛ-схем с положительным или отрицательным напряжением питания. Сегодня, когда КМОП-схемы переходят на 3.3-вольтовое питание, стало возможным создание ЭСЛ-подобных дифференциальных входов и выходов, у которых логические уровни привязаны к напряжению питания 3.3 В, а не к 5 В.

.7. Язык описания схем VHDL

середине 80-х годов Министерство обороны США (U.S. Department of Defense, D) и Институт инженеров по электротехнике и электронике (Institute of Electrical d Electronic Engineers, IEEE) поддержали разработку довольно мощного язы- описания схем VHDL. С самого начала и по настоящее время отличительными обенностями этого языка является следующее:

- Проектируемые устройства можно иерархически разбивать на составные элементы.
- Каждый элемент устройства имеет ясно очерченный интерфейс (для соединения его с другими элементами) и точное функциональное описание (для его моделирования).

- Функциональное описание может быть основано на алгоритме, либо на реальной конструкции, которыми определяется работа элемента. Например, первоначально можно описать работу элемента посредством алгоритма, и это сделает возможной верификацию элементов более высокого уровня, в которых используется данный элемент; позднее алгоритмическое определение можно заменить структурной схемой.
- Все можно моделировать: параллелизм, временные соотношения и синхронизацию тактовыми сигналами. На языке VHDL можно описать как асинхронные, так и синхронные последовательные структуры.
- Можно моделировать выполняемые устройством в целом логические действия и его временные характеристики.

Таким образом, с самого начала VHDL является языком документации и моделирования, позволяющим точно задавать и имитировать поведение цифровых систем.

Хотя язык VHDL и его среда моделирования сами по себе были важными нововведениями, квантовый скачок полезности и популярности языка VHDL произошел с появлением коммерческих *программных средств синтеза на основе VHDL* (*VHDL synthesis tools*). Применяя эти средства, можно строить логические схемы непосредственно из описания их работы на языке VHDL. С помощью VHDL разрабатывается, моделируется и синтезируется все, что угодно, от простой комбинационной схемы до законченной микропроцессорной системы в одном кристалле.

В 1987 году Институтом инженеров по электротехнике и электронике был принят стандарт языка VHDL (*VHDL-87*), а в 1993 году этот стандарт был расширен (*VHDL-93*). В этом параграфе речь пойдет о таких правилах, которые действуют в обеих версиях языка. Другие особенности языка VHDL будут рассмотрены в параграфе 7.12 применительно к проектированию последовательностных логических схем.

ЧТО ТАКОЕ VHDL?

VHDL означает “VHSIC Hardware Description Language” («язык описания схем на основе VHSIC»). В свою очередь, VHSIC (Very High Speed Integrated Circuit, интегральная схема с очень высоким быстродействием) было названием программы поддержки Министерством обороны США исследований в области высокоэффективной интегральной электроники.

4.7.1. Ход выполнения проекта

Прежде чем обратиться к самому языку, полезно составить представление об окружающей среде, в которой развивается VHDL-проект. Процесс проектирования на основе языка VHDL, или *ход выполнения проекта* (*desigh flow*), состоит из ряда этапов. Через эти этапы бывает необходимо пройти при разработке устройств на основе любого языка описания схем; в общих чертах они представлены на рис. 4.50.

Библиотека БГУИР

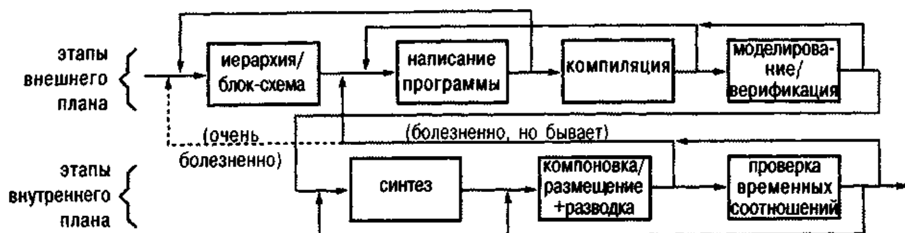


Рис. 4.50. Этапы в ходе выполнения проекта на основе языка VHDL или любого другого языка описания схем

Действия так называемого *«внешнего плана»* (*"front-end"*) начинаются с осознания основного подхода и функций отдельных блоков на уровне блок-схемы. Большие логические проекты типа программного обеспечения обычно являются иерархическими, и язык VHDL служит хорошей основой как для определения модулей и их интерфейсов, так и для детализации в дальнейшем.

ЯЗЫКИ VERILOG И VHDL

Примерно в то же время, когда разрабатывался язык VHDL, на сцене появился другой язык описания схем. Язык *Verilog HDL*, или просто *Verilog*, был предложен в 1984 году фирмой Gateway Design Automation в качестве собственного языка описания схем и средства моделирования. Когда в 1988 году появились программные средства синтеза на основе языка Verilog, выпущенные только-только оперившейся фирмой Synopsys, а фирма Gateway была в 1989 году приобретена фирмой Cadence Design Systems, такое сочетание стало решающим фактором, приведшим к повсеместному распространению этого языка.

Сегодня оба языка – VHDL и Verilog – широко применяются и делят рынок логического синтеза примерно поровну. Синтаксис языка Verilog берет свое начало в языке С и, в некотором отношении, ему легче научиться и им легче пользоваться, тогда как язык VHDL больше похож на язык Ada (язык программирования, поддерживаемый Министерством обороны США) и в большей степени пригоден для больших проектов.

Рассматривая вопрос о том, с изучения какого из этих языков следует начинать, и сравнивая с этой точки зрения их относительные достоинства и недостатки, лучше всего, по-видимому, положиться на заключение Дэвида Пеллерина (David Pellerin) и Дугласа Тейлора (Douglas Taylor), сделанное ими в их книге *С VHDL теперь все просто!* (*VHDL Made Easy*. Prentice Hall, 1997):

Оба языка легко выучить и обоими языками трудно овладеть.
После того как вы изучили один из этих языков,
у вас не будет затруднений при переходе к другому.

Библиотека БГУИР

ЧТО ЗНАЧИТ VERILOG?

Слово "Verilog" не является акронимом, но мне кажется, что оно вполне могло бы быть сокращением от "VERIfy LOGic" («проверяй логику»).

Следующий шаг состоит в фактическом описании на языке VHDL модулей, их интерфейсов и деталей их внутреннего устройства. В этой части проекта вы можете, в принципе, воспользоваться любым текстовым редактором, поскольку на языке VHDL пишется текст. Однако в большинстве случаев среда, в которой осуществляется проектирование, включает специализированный *текстовый редактор VHDL* (*VHDL text editor*), который облегчает работу. Такие редакторы обычно содержат автоматическое высвечивание ключевых слов языка VHDL, автоматический отступ от начала строки, встроенные шаблоны часто используемых программных структур, встроенную проверку синтаксиса и упрощенный доступ к компилятору.

Написав некоторую программу, вы, безусловно, захотите ее оттранслировать. *Компилятор языка VHDL* (*VHDL compiler*) проанализирует ваш текст на отсутствие в нем синтаксических ошибок и проверит совместимость вашей программы с другими модулями, на которые имеются ссылки. Компилятор подготавливает также внутреннюю информацию, которая понадобится моделирующей программе на следующем этапе вашего проекта. При программировании часто бывает так, что вам хочется приступить к компиляции еще до того, как программа будет написана до конца. Компилирование по частям, поможет предотвратить размножение синтаксических ошибок, появление несовместимых имен и так далее, и, наверняка, позволит вам испытать столь необходимое чувство движения вперед на стадии, когда конца проекта еще не видно!

Самым впечатляющим этапом, по-видимому, является моделирование. *Моделирующая программа VHDL* (*VHDL simulator*) позволяет задавать входные сигналы и подавать их на входы разрабатываемой конструкции, а также наблюдать выходные сигналы, не собирая схему физически. При выполнении небольших проектов типа домашнего задания по курсу цифровой электроники входные сигналы можно задать вручную и визуально наблюдать выходные сигналы. Но в случае больших проектов язык VHDL дает возможность осуществлять тестирование, создавая *программные средства тестирования* (*«испытательные стенды»*, "*test benches*"), в которых входные сигналы подаются автоматически, а выходные сигналы сравниваются с ожидаемыми.

На самом деле, моделирование является одной из ступеней более крупного этапа *верификации* (*verification*). Наблюдение того, как на выходах моделируемой схемы возникают сигналы, действительно доставляет большое удовлетворение, но цель моделирования шире: она состоит в том, чтобы проверить схему и убедиться, что она работает так, как хочется. В типичном большом проекте существенные усилия затрачиваются как на стадии написания программы, так и после этого, когда бывает необходимо задать достаточно широкий диапазон ус-

Библиотека БГУИР

ловий тестирования схемы для проверки правильности реализуемых ею логических действий. Обнаружение ошибок в проекте на этой стадии очень ценно; если ошибки обнаружатся позднее, то чаще всего все так называемые этапы «*внутреннего плана*» ("*back-end*") придется повторить.

Заметьте, что существует, по крайней мере, два аспекта верификации. При *функциональной верификации* (*functional verification*) логика работы схемы изучается независимо от временных соображений; задержки в вентилях и другие временные параметры считаются равными нулю. При *проверке временны́х соотношений* (*timing verification*) работа схемы исследуется с учетом предполагаемых задержек; мы проверяем при этом, в частности, удовлетворяются ли требования по времени установления сигналов и их удержания в случае последовательностных устройств типа триггеров. Принято осуществлять *функциональную* верификацию до перехода к выполнению этапов внутреннего плана. Что касается проведения *проверки временны́х соотношений*, то на этой стадии наши возможности ограничены, так как требуемые временные соотношения в очень сильной степени зависят от результатов синтеза и подгонки. Можно выполнить предварительную проверку временных соотношений, чтобы приобрести бóльшую уверенность в правильности самого подхода к проекту в целом, но проведение детальной проверки временных соотношений необходимо отложить до самого конца.

После верификации мы готовы перейти к стадии «внутреннего плана». Характер действий на этом этапе и используемые средства довольно сильно зависят от технологии, по которой выполнен кристалл, выбранный для данного проекта, но существуют три основных этапа. Первый из них – это *синтез* (*synthesis*), то есть преобразование описания на языке VHDL в набор примитивов или компонентов, которые можно будет образовать в выбранном кристалле. Например, в случае ПЛУ или ИС типа CPLD программа синтеза может выдать равенства, ориентированные на реализацию двухуровневыми схемами выражений вида «сумма произведений». В случае специализированных ИС результатом действия программы синтеза могут быть список вентилей и *список соединений* (*netlist*), которым определяются необходимые соединения вентилей между собой. Разработчик может «помочь» программе синтеза, задав *ограничения* (*constraints*), характерные для выбранной технологии, такие как максимальное число логических уровней или нагрузочная способность логических буферов.

На этапе *компоновки* (*fitting*) *программа компоновки* (*fitter*) отображает синтезированные примитивы и компоненты на имеющиеся в микросхеме ресурсы. В случае ПЛУ и ИС типа CPLD это может означать приписывание равенств тем или иным элементам И–ИЛИ. В случае специализированных ИС на этом этапе происходит раскладка отдельных вентилей в нужной конфигурации и нахождение путей для их соединения с учетом физических ограничений в кристалле данной ИС; эту процедуру называют *размещением и разводкой* (*place and route*). На этой стадии разработчик, как правило, имеет возможность ввести дополнительные ограничения, такие как размещение модулей в кристалле или назначение выводов для внешних входов и выходов.

«Последний» этап заключается в проверке временных соотношений в схеме с учетом ее размещения в кристалле. Только на этой стадии можно с разумной

Библиотека БГУИР

точностью найти реальные задержки в схеме, обусловленные длиной соединений, величиной нагрузки и другими факторами. Обычно на этом этапе используются те же самые условия тестирования, что и при функциональной верификации, только на данном шаге в эти условия помещается схема, которая в действительности будет построена.

Как и в любом другом творческом процессе может случиться так, что после продвижения на два шага вперед, вам приходится делать шаг назад (если не хуже!). Как отмечено на рисунке, при написании программы могут встретиться проблемы, которые заставят вернуться назад и пересмотреть иерархию, и почти наверняка в процессе компиляции и моделирования обнаружатся ошибки, из-за которых вам придется переписать часть программы.

Самыми болезненными в ходе выполнения проекта бывают те проблемы, с которыми вы сталкиваетесь при выполнении этапов внутреннего плана. Если, например, результат синтеза не влезает в имеющийся кристалл FPGA или не удовлетворяет временным требованиям, может случиться так, что вы должны будете вернуться назад и пересмотреть сам подход к проекту в целом. Стоит напомнить, что прекрасные программные средства не заменяют все же тщательного продумывания проекта в самом начале.

4.7.2. Структура программы

При создании языка VHDL имелось в виду воплотить в нем принципы структурного программирования с заимствованием идей у языков программирования Паскаль и Ада. Ключевая идея состояла в том, чтобы задать интерфейс схемного модуля, а его внутреннее устройство скрыть. Таким образом, *объект* (*entity*) в языке VHDL – это просто объявление входов и выходов модуля, а *архитектура* (*architecture*) – подробное описание внутренней структуры модуля или его поведения.

РАБОТАЕТ!?

Как разработчик и системщик с многолетним опытом, я всегда думал, что знаю, чтó имеется в виду, когда кто-то говорит о своей схеме: «Она работает!». На мой взгляд, это означает, что вы можете пойти в лабораторию, подать питание на макет (и при этом не пойдет дым), нажать кнопку «Пуск» и с помощью осциллографа или логического анализатора наблюдать, как ваш макет последовательно выполняет необходимые действия.

Но с годами значение слов «она работает» изменилось, по крайней мере, для некоторых людей. Когда несколько лет назад я перешел на новую работу, я порадовался, услышав, что несколько основных специализированных ИС для нового важного продукта «работают». Но позднее (спустя совсем немного времени) я понял, что эти ИС работают только при моделировании и что коллективу, занятому в этом проекте, предстоит еще не раз возвращаться назад и понадобится несколько месяцев напряженного труда для синтеза, подгонки и проверки временных соотношений, прежде чем они смогут заказать опытные образцы. Да, верно: «Она работает!». Только это напоминает мне, как мои дети говорят о домашнем задании: «Готово!».

Библиотека БГУИР

Рис. 4.51(а) иллюстрирует этот принцип. Многие разработчики склонны считать объявление объекта в языке VHDL «оболочкой» архитектуры, скрывающей детали того, что находиться внутри, но обеспечивающей «зацепки» для других модулей, использующих данный модуль. Эта идея служит основой иерархического подхода к проектированию систем: архитектура верхнего уровня может использовать (или «обрабатывать») другие объекты, оставляя архитектурные детали объектов нижнего уровня скрытыми от объектов более высокого уровня. Как показано на рис. 4.51(b), архитектура более высокого уровня может использовать объекты более низкого уровня многократно, и несколько архитектур верхнего уровня могут использовать один и тот же объект более низкого уровня. На рисунке архитектуры B, E и F являются самостоятельными; они не используют никакие другие объекты.

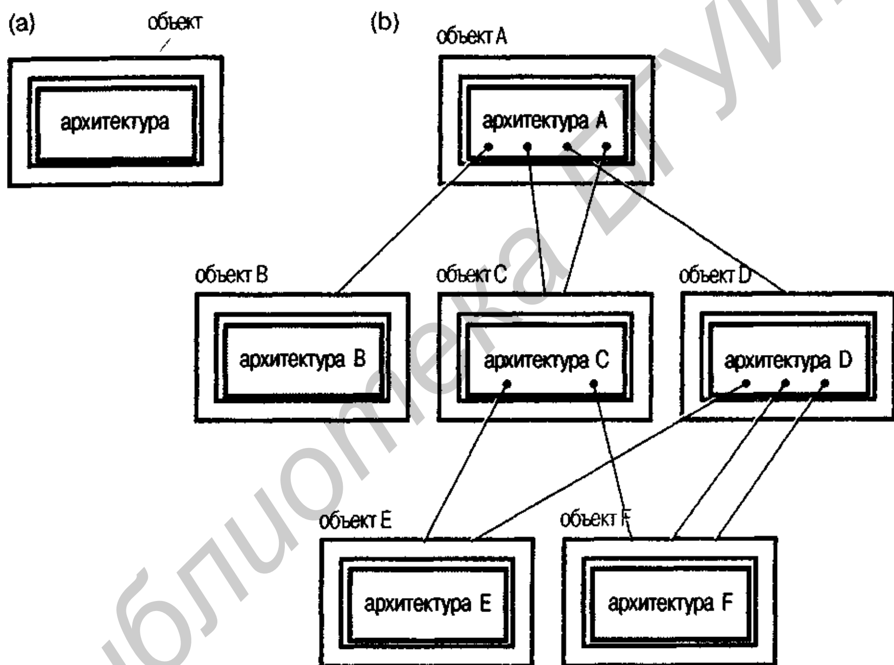


Рис. 4.51. Объекты и архитектуры языка VHDL: (а) идея «оболочки»; (b) иерархическое использование

ПИШИТЕ ВСЕ, ЧТО ХОТИТЕ!

В действительности, язык VHDL позволяет определять несколько архитектур, объединенных в одном объекте, и предоставляет средства управления конфигурацией, посредством которых можно указывать, какая именно архитектура будет использоваться при очередном запуске компилятора или программы синтеза. Это позволяет опробовать различные архитектурные подходы, не выкидывая и не пряча другие варианты. Однако в нашем учебнике мы не воспользуемся этой возможностью и не станем ее обсуждать подробнее.

Библиотека БГУИР

В текстовом файле на языке VHDL *объявление объекта* (*entity declaration*) и *определение архитектуры* (*architecture definition*) разделены, как это сделано на рис. 4.52. В табл. 4.26 в качестве примера приведена очень простая программа на языке VHDL для 2-входового вентиля «запрета». В больших проектах объекты и архитектуры иногда бывают помещены в отдельные файлы, связь между которыми компилятор обнаруживает по их объявленным именам.

текстовый файл (например, mydesign vhd)

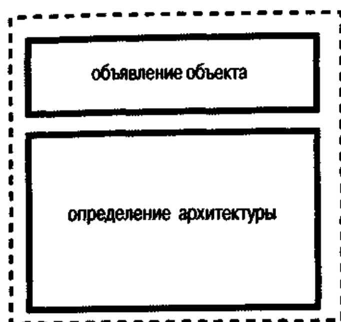


Рис. 4.52. Общий вид файла программы на языке VHDL

Табл. 4.26. Программа на языке VHDL для вентиля «запрета»

```
entity Inhibit is        -- also known as 'BUT-NOT'
  port (X,Y: in BIT;      --  as in 'X but not Y'
        Z:   out BIT);    --  (see [Klir, 1972])
end Inhibit;

architecture Inhibit_arch of Inhibit is
begin
  Z <= '1' when X='1' and Y='0' else '0';
end Inhibit_arch;
```

Как и в других языках программирования, в языке VHDL пробелы и переходы с одной строки на другую в общем случае игнорируются, и для удобства чтения их можно вставлять как угодно. *Комментарии* (*comments*) начинаются с двух дефисов (--) и заканчиваются концом строки.

В языке VHDL определено много специальных строк символов, называемых *зарезервированными словами* (*reserved words*) или *ключевыми словами* (*keywords*). В приведенном примере имеется несколько ключевых слов: `entity`, `port`, `is`, `in`, `out`, `end`, `architecture`, `begin`, `when`, `else` и `not`. Определяемые пользователем *идентификаторы* (*identifiers*) начинаются с буквы и содержат буквы, цифры и подчеркивания. (Символ подчеркивания не может следовать за другим символом подчеркивания и не может быть последним символом идентификатора.) В данном примере идентификаторами являются `Inhibit`, `X`, `Y`, `BIT`, `Z` и `Inhibit_arch`. "`BIT`" – это встроенный идентификатор предопределенного типа; он не считается зарезервированным словом, так как его можно переопределять. Зарезервированные слова и идентификаторы не чувствительны к регистру.

Библиотека БГУИР

В табл. 4.27 представлен синтаксис объявления объекта. Целью объявления объекта, помимо присвоения объекту имени, является определение сигналов внешнего интерфейса или *портов* (*ports*) в части объявления объекта, которая называется *объявлением портов* (*port declaration*). Кроме ключевых слов `entity`, `is`, `port` и `end`, объявление объекта содержит следующие элементы:

entity-name выбираемое пользователем имя объекта;

signal-names список выбираемых пользователем имен сигналов внешнего интерфейса, состоящий из одного имени или из большего числа имен, разделенных запятой;

mode одно из четырех зарезервированных слов, определяющих направление передачи сигнала:

- `in` сигнал на входе объекта;
- `out` сигнал на выходе объекта; заметьте, что значение такого сигнала нельзя «прочитать» внутри структуры объекта; он доступен только объектам, использующим данный объект;
- `buffer` сигнал на выходе объекта, такой что его значение можно читать также внутри структуры данного объекта;
- `inout` сигнал, который может быть входным или выходным для данного объекта; обычно этот режим используется применительно к входам/выходам ПЛУ с тремя состояниями;

signal-type встроенный или определенный пользователем тип сигнала; в следующих разделах мы будем много говорить об этом.

Обратите внимание, что после заключительного *signal-type* нет точки с запятой; изменение порядка следования закрывающей скобки и точки с запятой после нее – типичная синтаксическая ошибка программиста, начинающего писать на языке VHDL.

Табл. 4.27. Синтаксис объявления объекта на языке VHDL

```
entity entity-name is
  port (signal-names : mode signal-type;
        signal-names : mode signal-type;
        ...
        signal-names : mode signal-type);
end entity-name;
```

Порты объекта, а также направление передачи и типы сигналов – это все, что видят другие модули, использующие данный модуль. Внутренняя работа объекта задается его *определением архитектуры* (*architecture definition*), синтаксис которого в общем случае имеет вид, указанный в табл. 4.28. *Имя объекта* (*entity-name*) в этом определении должно быть таким же, какое раньше было присвоено объекту в объявлении объекта. *Имя архитектуры* (*architecture-name*) – это выбираемый пользователем идентификатор, обычно так или иначе связанный с именем объекта; при желании имя архитектуры может быть тем же самым, что и имя объекта.

Библиотека БГУИР

Табл. 4.28. Синтаксис определения архитектуры на языке VHDL

```
architecture architecture-name of entity-name is
   type declarations
   signal declarations
   constant declarations
   function definitions
   procedure definitions
   component declarations
begin
   concurrent-statement
   ...
   concurrent-statement
end architecture-name;
```

Сигналы внешнего интерфейса архитектуры (порты) наследуются от той части объявления соответствующего объекта, где объявляются порты. У архитектуры могут быть также сигналы и другие объявления, являющиеся для нее локальными, подобно тому как это имеет место в других языках высокого уровня. В отдельном «пакете», используемом несколькими объектами, можно сделать объявления, общие для этих объектов, о чем будет сказано позднее.

Объявления в табл. 4.28 могут располагаться в произвольном порядке. В свое время мы рассмотрим много различных способов записи объявлений и операторов в определении архитектуры. Начать легче всего с *объявления сигнала* (*signal declaration*), которое сообщает ту же самую информацию о сигнале, какую содержит объявление порта, за исключением того, что вид сигнала не задается:

```
signal signal-names : signal-type;
```

В архитектуре может быть объявлено любое число сигналов, начиная с нуля, и они приблизительно соответствуют поименованным соединениям в принципиальной схеме. Их можно считывать и записывать внутри определения архитектуры и, подобно другим локальным элементам, на них можно ссылаться только в пределах данного определении архитектуры.

Переменные (*variables*) в языке VHDL похожи на сигналы, за исключением того, что, как правило, они не имеют никакого физического смысла в схеме. Действительно, обратите внимание, что, согласно табл. 4.28, в определении архитектуры не предусмотрено «объявление переменных». Переменные используются в функциях, процедурах и процессах языка VHDL. Каждый из этих элементов программы мы рассмотрим позднее. Вот у них внутри имеются *объявления переменных* (*variable definitions*), и эти объявления в точности подобны объявлениям сигналов, за исключением того, что употребляется ключевое слово variable:

```
variable variable-names : variable-type;
```

4.7.3. Типы и константы

Каждому сигналу, переменной и константе в программе на языке VHDL необходимо поставить в соответствие *тип* (*type*). Типом определяется множество или диа-

Библиотека БГУИР

пазон значений, которые может принимать данный элемент, и обычно имеется набор операторов (таких как сложение, логическое И и т.д.), связываемых с данным типом.

В языке VHDL есть всего лишь несколько *предопределенных типов* (*predefined types*); они перечислены в табл. 4.29. В дальнейшем в этой книге будут использованы только следующие предопределенные типы: `integer`, `character` и `boolean`. Вы можете подумать, что при цифровом проектировании большую роль должны играть имена "`bit`" и "`bit_vector`", но оказывается, что более полезны определяемые пользователем варианты этих типов, как это вскоре будет объяснено.

Табл. 4.29. Предопределенные типы языка VHDL

bit	character	severity_level
bit_vector	integer	string
boolean	real	time

Типом *integer* определяется диапазон значений целых чисел, который, как минимум, простирается от –2147483647 до +2147483647 (от $-2^{31}+1$ до $+2^{31}-1$); в некоторых реализациях языка VHDL этот диапазон может быть и шире. Типом *boolean* предусматриваются два значения: *true* и *false*. Тип *character* содержит все символы 8-битового набора ISO, из которых первые 128 являются символами стандарта ASCII. Встроенные операторы для типов `integer` и `boolean` приведены в табл. 4.30.

Табл. 4.30. Предопределенные операторы для типов `integer` и `boolean` в языке VHDL

Операторы для типа *integer*		***Операторы для типа*** *boolean*	
+	сложение	and	И
-	вычитание	or	ИЛИ
*	умножение	nand	И-НЕ
/	деление	nor	ИЛИ-НЕ
mod	деление по модулю	xor	ИСКЛЮЧАЮЩЕЕ ИЛИ
rem	остаток от деления по модулю	xnor	ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ
abs	абсолютное значение	not	дополнение (инверсия)
**	возведение в степень		

Чаще всего в типичных программах на языке VHDL используются *определяемые пользователем типы* (*user-defined types*), а из них самыми употребительными являются *перечислимые типы* (*enumerated types*), которые определяются путем перечисления их значений. Предопределяемые типы `boolean` и `character` – это перечислимые типы. Формат объявления типа в случае перечислимого типа указан в первой строке табл. 4.31. Здесь *value-list* представляет собой список (перечисление) всех возможных значений этого типа, разделяемых запятыми. Значе-

Библиотека БГУИР

ниями могут быть определяемые пользователем идентификаторы или символы (где под «символом» понимается символ ISO, заключенный в одинарные кавычки). Идентификаторы чаще всего применяются для обозначения альтернатив или состояний конечного автомата, например:

```
type traffic_light_state is (reset, stop, wait, go);
```

Символы используются в очень важном случае стандартного определяемого пользователем логического типа *std_logic* (см. табл. 4.32), являющегося частью стандартного пакета IEEE 1164, рассматриваемого в разделе 4.7.5. Этот тип включает не только «0» и «1», но также и семь других значений, которые оказываются полезными при моделировании логического сигнала (бита) в реальной логической схеме, как это детально объясняется в разделе 5.6.4.

Табл. 4.31. Синтаксис объявления типов и констант в языке VHDL

```
type type-name is (value-list);

subtype subtype-name is type-name start to end;
subtype subtype-name is type-name start downto end;

constant constant-name: type-name := value;
```

Табл. 4.32. Определение типа std_logic в языке VHDL (о значении "resolved" см. раздел 5.6.4)

```
type STD_ULOGIC is ( 'U',  -- Uninitialized
                     'X',  -- Forcing  Unknown
                     'O',  -- Forcing  0
                     '1',  -- Forcing  1
                     'Z',  -- High Impedance
                     'W',  -- Weak     Unknown
                     'L',  -- Weak     0
                     'H',  -- Weak     1
                     '-'   -- Don't care
                   );
subtype STD_LOGIC is resolved STD_ULOGIC;
```

Язык VHDL позволяет пользователю создавать также *подтипы* (*subtypes*) согласно синтаксису, указанному в табл. 4.31. Значения подтипа должны быть слитным подмножеством значений, предусмотренных основным типом, начиная со *start* и кончая *end*. Для перечислимого типа «слитность» означает расположение на соседних позициях в исходном списке значений *value-list*. Вот несколько примеров определения подтипов:

```
subtype twoval_logic is std_logic range '0' to '1';
subtype fourval_logic is std_logic range 'X' to 'Z';
subtype negint is integer range -2147483647 to -1;
subtype bitnum is integer range 31 downto 0;
```

Библиотека БГУИР

СТРОГОЕ СОБЛЮДЕНИЕ ТИПОВ

VHDL, как и C, является языком со строгим контролем типов. Это означает, что компилятор не позволит вам присвоить сигналу или переменной значение, которое в точности не согласуется с объявленным типом этого сигнала или переменной.

Требование строго соблюдать типы – это одновременно и благо, и бедствие. Это делает вашу программу более надежной и ее легче отлаживать, поскольку оказывается трудным делать «глупые ошибки», присваивая значения неправильного типа или не подходящей величины. С другой стороны, иногда это может раздражать. Даже при выполнении простых действий может потребоваться обращение к функции преобразования типов в явном виде, например, когда необходимо 2-битовый сигнал интерпретировать как целое число, чтобы перейти к одному из возможных случаев в операторе "case".

Заметьте, что порядок следования значений в указываемом диапазоне может быть в сторону возрастания или в сторону убывания в зависимости от того, какое из ключевых слов *to* или *downto* употреблено. Из-за некоторых особенностей подтипов это различие может быть существенным, но мы не будем использовать эти особенности в нашей книге и поэтому ограничимся уже сказанным.

В языке VHDL есть два предопределенных подтипа `integer`:

```
subtype natural is integer range 0 to highest-integer;
subtype positive is integer range 1 to highest-integer;
```

Константы (*constants*) способствуют удобству чтения программ, возможности их поддержания и сопровождения, а также переносу на какой-либо другой язык. Синтаксис *объявления констант* (*constant declaration*) в языке VHDL указан в последней строке в табл. 4.31; его можно проиллюстрировать следующими примерами:

```
constant BUS_SIZE: integer := 32;      -- width of component
constant MSB: integer := BUS_SIZE-1;  -- bit number of MSB
constant Z: character := 'Z';          -- synonym for Hi-Z value
```

ЧТО ЗА СИМВОЛ?

Вы можете удивиться тому, что в типе `std_logic` значения задаются символами, а не однобуквенными идентификаторами. Ясно, что "U", "X" и т.д. было бы легче набрать, чем "'U'", "'X'" и т.д. Ну, прежде всего, тогда потребовался бы другой идентификатор, не "–", для безразличных значений, но это еще полбеды. Главная причина употребления символов в одинарных кавычках состоит в том, что нельзя воспользоваться «нулем» и «единицей» – "0" и "1", – поскольку уже принято, что их следует распознавать как постоянные целые числа. Это возвращает нас снова к строгому следованию типам в языке VHDL; едва ли было целесообразно позволить компилятору осуществлять автоматическое преобразование типов в зависимости от контекста.

Библиотека БГУИР

НЕНАТУРАЛЬНЫЕ ДЕЙСТВИЯ

В языке VHDL подтип "natural" определяется как множество неотрицательных целых чисел, начиная с 0, но большинство математиков считают, что, по определению, натуральные числа начинаются с 1. Действительно в своей ранней истории человечество начинало счет с 1; понятие "0" появилось много позднее. Однако полемика на эту тему продолжается, особенно в наш компьютерный век, когда большинству из нас приходиться начинать счет с 0. Последние соображения по данной проблеме можно найти в Интернете, осуществив поиск по словам "natural numbers".

Обратите внимание, что значение константы может быть задано простым выражением. Константы можно использовать повсюду, где встречаются соответствующие значения, и они особенно полезны при определении типов, как будет вскоре показано.

Другую очень важную группу определяемых пользователем типов образуют *типы массивов* (*array types*). Как и в других языках, в языке VHDL *массив* (*array*), по определению, – это упорядоченный набор элементов одного и того же типа, отдельные компоненты которого выбираются с помощью *индекса массива* (*array index*). Возможны несколько вариантов синтаксиса объявления массива в языке VHDL; они представлены в табл. 4.33. В первых двух вариантах *start* и *end* являются целыми числами, которыми задается возможный диапазон изменения индекса массива и, следовательно, полное число элементов массива. В последних трех вариантах диапазоном изменения индекса массива являются все значения указанного типа (*range-type*) или подмножество этих значений.

Табл. 4.33. Синтаксис объявления массивов в языке VHDL

```
type type-name is array (start to end) of element-type;
type type-name is array (start downto end) of element-type;
type type-name is array (range-type) of element-type;
type type-name is array (range-type range start to end) of element-type;
type type-name is array (range-type range start downto end) of element-type;
```

В табл. 4.34 приведены примеры объявления массивов. Первые два примера совсем обычны и демонстрируют задание диапазона изменения индекса в сторону возрастания и в сторону убывания. Следующий пример показывает, как можно воспользоваться константой WORD_LEN при объявлении массива; отсюда видно также, что границу диапазона можно задать простым выражением. Из третьего примера следует, что сам элемент массива может быть массивом; таким образом создается двумерный массив. Последний пример показывает, что множество возможных значений элементов массива можно задать, указав перечислимый тип (или подтип); в этом примере массив состоит из четырех элементов согласно данному нами чуть раньше определению типа traffic_light_state.

Библиотека БГУИР

Табл. 4.34. Примеры объявления массивов в языке VHDL

```
type monthly_count is array (1 to 12) of integer;
type byte is array (7 downto 0) of STD_LOGIC;

constant WORD_LEN: integer := 32;
type word is array (WORD_LEN-1 downto 0) of STD_LOGIC;

constant NUM_REGS: integer := 8;
type reg_file is array (1 to NUM_REGS) of word;

type statecount is array (traffic_light_state) of integer;
```

Элементы массива считаются упорядоченными слева направо в том же направлении, в каком индекс пробегает свои значения. Таким образом, индексы самых левых элементов массивов типов `monthly_count`, `byte`, `word`, `reg_file` и `statecount` в табл. 4.34 равны 1, 7, 31, 1 и `reset` соответственно.

Обращение к отдельным элементам массивов в операторах программы на языке VHDL осуществляется путем указания имени массива и индекса элемента в круглых скобках. Если, например, `M`, `B`, `W`, `R` и `S` – сигналы или переменные тех пяти типов массивов, которые приведены в табл. 4.34, то любая из записей `M(11)`, `B(5)`, `W(WORD_LEN-5)`, `R(0,0)`, `R(0)` и `S(reset)` является правильным указанием элемента.

Массивы-литералы (*array literals*) можно задать, перечисляя в скобках значения элементов. Например, переменной `B` типа `byte` можно задать значение, состоящее из одних единиц, оператором

```
B:=('1','1','1','1','1','1','1','1');
```

В языке VHDL возможно и в более сжатой форме задавать значения, указывая индекс. Например, следующая запись обеспечивает присвоение единичных значений всем элементам переменной `W` типа `word`, за исключением младших разрядов каждого байта, которым присваиваются нулевые значения:

```
W:=(0=>'0',8=>'0',16=>'0',24 =>'0',others=>'1');
```

Рассмотренные правила справедливы при любом *типе элементов* (*element-type*), но литерал типа `STD_LOGIC` легче всего записать в виде «строки». *Строкой* (*string*) в языке VHDL является последовательность символов ISO, заключенная в двойные кавычки, типа `"Hi there"`. Строка – это, конечно, массив символов; поэтому массиву типа `STD_LOGIC` заданной длины можно присвоить значение, выраженное строкой той же длины, если только символы в строке принадлежат набору из девяти символов, которыми, по определению, исчерпываются возможные значения элементов типа `STD_LOGIC`: `'0'`, `'1'`, `'U'` и т.д. Таким образом, предыдущие два примера можно переписать в виде:

```
B: ="11111111";
W: ="11111110111111101111111011111110";
```

Библиотека БГУИР

Можно также указывать подмножество непосредственно следующих один за другим элементов массива или, как говорят, *вырезку из массива* (*array slice*), задавая начальный и конечный индексы подмножества; например: `M(6 to 9)`, `B(3 downto 0)`, `W(15 downto 8)`, `R(0,7 downto 0)`, `R(1 to 2)`, `S(stop to go)`. Заметьте, что направление изменения индекса в вырезке должно быть таким же, как у исходного массива.

Наконец, массивы или элементы массивов можно объединять с помощью *оператора конкатенации* & (*concatenation operator*), который соединяет массивы и элементы в том порядке, в каком они записаны слева направо. Например, запись `'0'&'1'&"1Z"` эквивалентна строке `"011Z"`, а выражение `B(6 downto 0) & B(7)` представляет собой циклический сдвиг 8-разрядного массива `B` на 1 разряд влево.

Самым важным типом массивов в типичной программе на языке VHDL является определяемый пользователем в соответствии со стандартом IEEE 1164 логический тип `std_logic_vector`, которым задается упорядоченный набор элементов типа `std_logic`. Определение этого типа имеет вид:

```
type STD_LOGIC_VECTOR is array (natural range <>) of STD_LOGIC;
```

Это пример *типа массива без ограничений* (*unconstrained array type*): диапазон возможных значений индекса массива не задан, за исключением того, что он должен быть подмножеством определенного типа, в данном случае – типа `natural`. Эта особенность языка VHDL позволяет записывать архитектуры, функции и другие элементы программ в более общем виде, до некоторой степени независимо от размеров массивов и диапазонов возможных значений индексов. Действительный диапазон значений индекса определяется в тот момент, когда сигналу или переменной ставится в соответствие этот тип. В следующем разделе мы увидим примеры этого.

4.7.4. Функции и процедуры

Подобно функции в любом языке программирования высокого уровня, *функция* (*function*) в языке VHDL получает ряд *аргументов* (*arguments*) и возвращает *результат* (*result*). При определении функции на языке VHDL и при ее вызове каждый из аргументов и результат имеют предустановленный тип.

Синтаксис *определения функции* (*function definition*) приведен в табл. 4.35. За присвоением функции определенного имени следует список *формальных параметров* (*formal parameters*), который используется в теле функции; число параметров может быть любым, начиная с нуля. При вызове функции формальные параметры в обращении к ней замещаются *действительными параметрами* (*actual parameters*). В соответствии со строгим следованием типам в языке VHDL действительные параметры должны быть того же типа или подтипа, что и формальные параметры. Когда функция вызывается из архитектуры, на место ее вызова возвращается значение, тип которого указывается посредством *return-type*.

Библиотека БГУИР

Табл. 4.35. Синтаксис определения функции в языке VHDL

```
function function-name (
        signal-names : signal-type;
        signal-names : signal-type;
        ...
        signal-names : signal-type
) return return-type is
    type declarations
    constant declarations
    variable declarations
    function definitions
    procedure definitions
begin
    sequential-statement
    ...
    sequential-statement
end function-name;
```

Как видно из таблицы, внутри функции можно определить ее собственные локальные типы, константы, переменные и вложенные функции и процедуры. Между ключевыми словами `begin` и `end` располагается ряд «последовательных операторов», которые исполняются при вызове функции. Последовательные операторы и их синтаксис будут предметом более подробного разбора позднее, но вам должны быть понятны приводимые здесь примеры с учетом вашего предыдущего опыта программирования.

Архитектуру «вентиля запрета» на языке VHDL, приведенную в табл. 4.26, можно видоизменить, используя функцию, как показано в табл. 4.36. В определении функции момент возврата к месту вызова указывается ключевым словом *`return`*, за которым следует выражение, значение которого и будет возвращено. Тип результата вычисления этого выражения должен быть согласован со значением *return-type* в объявлении функции.

В логическом пакете стандарта IEEE 1164 определено много функций, оперирующих типами `std_logic` и `std_logic_vector`. Помимо того, что предусматривается ряд определяемых пользователем типов, пакет содержит также основные логические операции над сигналами или переменными этих типов, такие как `and` и `or`. Язык VHDL обладает тем достоинством, что в нем возможна *перегрузка операторов* (*operator overloading*). Это позволяет пользователю выбирать активизируемую функцию посредством применения символа встроенной операции (`and`, `or`, + и т. д.) к согласованному набору типов операндов. Возможно несколько определений одного и того же символа операции; каждый раз, когда встречается данный оператор, компилятор автоматически выбирает определение, согласованное по типу операндов.

В качестве примера табл. 4.37 содержит код, взятый из пакета IEEE, которым определяется операция "`and`" для операндов типа `std_logic`. Может казаться, что этот отрывок сложен, но нами уже введены все основные элементы языка, употребляемые здесь (за исключением слова "`resolved`", которое будет рассмотрено в разделе 5.6.4 в связи с логическими схемами с тремя состояниями).

Библиотека БГУИР

Табл. 4.36. Программа для «функции запрета» на языке VHDL

```
architecture Inhibit_archf of Inhibit is

function ButNot (A, B: bit) return bit is
begin
  if B = '0' then return A;
  else return '0';
  end if;
end ButNot;

begin
  Z <= ButNot(X,Y);
end Inhibit_archf;
```

Табл. 4.37. Определения в стандарте IEEE 1164, относящиеся к операции "and" над величинами типа STD_LOGIC

```
SUBTYPE UX01 IS resolved std_ulogic RANGE 'U' TO '1';
                                                  -- ('U','X','0','1')
TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic) OF std_ulogic;

-- truth table for "and" function
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' )   -- | - |
);
FUNCTION "and" ( L : std_ulogic; R : std_ulogic ) RETURN UX01 IS
BEGIN
   RETURN (and_table(L, R));
END "and";
```

Переменные данной функции могут быть величинами типа std_ulogic или его подтипа std_logic. Другой подтип UX01, по определению, должен быть использован в качестве типа возвращаемого функцией результата; даже если один из операндов в операции "and" имеет нелогическое значение ('Z', 'W' и т. д.), то результатом обращения к функции будет только одно из четырех возможных значений. Тип stdlogic_table задает двумерный массив 9×9, индексами которого является пара величин типа std_ulogic. Элементы таблицы and_table расположены так, что при значениях любого из индексов '0' или 'L' (слабый '0') элемент равен '0'. Значение

Библиотека БГУИР

'1' извлекается только тогда, когда оба индекса равны '1' или 'H' (слабая '1'). В противном случае результат операции равен 'U' или 'X'.

Двойные кавычки у имени функции в самом определении функции указывают на перегрузку оператора. «Исполняемая» часть функции состоит всего лишь из одного оператора, который возвращает элемент таблицы, выбранный по двум переменным L и R функции “and”.

Из-за требований языка VHDL строго следовать типам, часто бывает необходимо преобразовать сигнал одного типа в сигнал другого типа; пакет IEEE 1164 содержит несколько функций преобразования: например, переход от типа BIT к типу STD_LOGIC и наоборот. Величину типа STD_LOGIC_VECTOR обычно нужно преобразовать в соответствующее целое число. В стандарте IEEE 1164 нет такой функции преобразования, поскольку разным разработчикам могут понадобиться различные представления чисел, например, со знаком или без знака. Однако можно самим задать свое собственное преобразование так, как это сделано в табл. 4.38.

Табл. 4.38. Функция преобразования типа STD_LOGIC_VECTOR в тип INTEGER на языке VHDL

```
function CONV_INTEGER (X: STD_LOGIC_VECTOR) return INTEGER is
  variable RESULT: INTEGER;
begin
  RESULT := 0;
  for i in X'range loop
    RESULT := RESULT * 2;
    case X(i) is
      when '0' | 'L'  => null;
      when '1' | 'H'  => RESULT := RESULT + 1;
      when others     => null;
    end case;
  end loop;
  return RESULT;
end CONV_INTEGER;
```

В функции CONV_INTEGER применен простой итеративный алгоритм, эквивалентный приведенной в параграфе 2.3 формуле представления числа в виде поочередных вложений. Мы отложим описание используемых здесь операторов FOR, CASE и WHEN до раздела 4.7.8, сосредоточив внимание на основной идее. *Оператор* null (null *statement*) совсем прост: он означает «ничего не делать». Пределы выполнения цикла FOR определяются параметром “X'range”, где одиночная кавычка после имени сигнала означает «*атрибут*» (“*attribute*”), а range – встроенный идентификатор атрибута, который применяется только по отношению к массивам и означает «перебрать все значения индекса данного массива слева направо».

Можно осуществить преобразование и в обратном направлении, то есть перейти от целого числа к величине типа STD_LOGIC_VECTOR, как показано в табл. 4.39. В этом случае необходимо задать не только преобразуемое целое (ARG),

Библиотека БГУИР

но также и желаемое число разрядов (SIZE) в результате преобразования. Обратите внимание, что в данной функции объявлена локальная переменная "result" типа STD_LOGIC_VECTOR с интервалом, в пределах которого изменяется индекс, зависящим от значения SIZE. По этой причине параметр SIZE должен быть константой или какой-то другой величиной, которая известна к моменту компиляции функции CONV_STD_LOGIC_VECTOR. В этой функции реализуется алгоритм последовательного деления, также описанный в параграфе 2.3.

Табл. 4.39. Функция преобразования типа INTEGER в тип STD_LOGIC_VECTOR на языке VHDL

```
function CONV_STD_LOGIC_VECTOR(ARG: INTEGER; SIZE: INTEGER)
    return STD_LOGIC_VECTOR is
  variable result: STD_LOGIC_VECTOR (SIZE-1 downto 0);
  variable temp: integer;
begin
  temp := ARG;
  for i in 0 to SIZE-1 loop
    if (temp mod 2) = 1 then result(i) := '1';
    else result(i) := '0';
    end if;
    temp := temp / 2;
  end loop;
  return result;
end;
```

Процедура (*procedure*) в языке VHDL похожа на функцию за исключением того, что она не возвращает результат. Если обращение к функции может играть роль выражения, то вызов процедуры можно использовать в качестве оператора. В языке VHDL допускается задание аргументам процедур типа out или типа inout, что фактически и делает возможным «возвращение» процедурой результата. Мы не будем пользоваться процедурами в нашей книге и поэтому не станем рассматривать их подробнее.

4.7.5. Библиотеки и пакеты

VHDL-*библиотека* (*library*) – это место, где компилятор VHDL хранит информацию об отдельном варианте проекта, включая промежуточные файлы, используемые при анализе, моделировании и синтезе в рамках данной разработки. Место библиотеки в файловой системе компьютера зависит от реализации. Для очередного VHDL-проекта компилятор автоматически создает библиотеку под именем "work" и использует ее.

У законченного VHDL-проекта обычно бывает много файлов, каждый из которых содержит различные компоненты проекта, включая объекты и архитектуры. Анализируя отдельные файлы, компилятор помещает результаты в библиотеку "work", а также ищет в этой библиотеке необходимые определения, например, другие объекты. С учетом этого большой проект можно разбить на несколько файлов; компилятор найдет все, что нужно, по внешним указателям.

Библиотека БГУИР

Но не вся необходимая информация может находиться в библиотеке "work". Например, разработчик может опираться на определения и функциональные модули, общие для некоторого семейства различных проектов. У каждого проекта есть своя собственная библиотека "work" (обычно это подкаталог внутри всего каталога, относящегося к данному проекту), но в нем должны быть ссылки на общую библиотеку, содержащую совместно используемые определения. Даже в малых проектах может использоваться библиотека, которая содержит, например, стандартные определения IEEE. Разработчик может присвоить имя такой библиотеке с помощью *предложения* `library` (`library` *clause*) в начале соответствующего файла. Например, библиотеку IEEE можно задать фразой:

```
library ieee;
```

Предложение "`library work;`" помещается в начале каждого файла VHDL-проекта неявно.

Присвоение имен библиотекам проекта обеспечивает доступ к любым ранее проанализированным и запомненным объектам и архитектурам, но не к определениям типов и тому подобному. Эту функцию выполняют «пакеты» и «предложения `use`».

VHDL-*пакет* (*package*) – это файл, содержащий определения элементов, которые могут быть использованы другими программами. В пакет можно включить элементы такого рода, как сигнал, тип, константа, функция, процедура и объявления компонентов.

Помещенные в пакет сигналы являются «глобальными» и доступны любому VHDL-объекту, использующему этот пакет. Типы и константы, упомянутые в пакете, известны в любом файле, использующем этот пакет. Аналогично, из файлов, использующих данный пакет, можно вызвать перечисленные в нем функции и процедуры, а архитектуры, опирающиеся на этот пакет, могут «обрабатывать» включенные в него компоненты, описываемые в следующем разделе.

Проект может «использовать» тот или иной пакет, если в начало файла, относящегося к данному проекту, помещено *предложение* `use` (`use` *clause*). Например, чтобы воспользоваться всеми определениями пакета, содержащего стандарт IEEE 1164, нам следует написать:

```
use ieee.std_logic_1164.all;
```

Здесь "`ieee`" – это имя библиотеки, ранее введенное предложением `library`. В этой библиотеке файл с именем "`std_logic_1164`" содержит желаемые определения. Приставка "`all`" велит компилятору использовать все определения этого файла. Вместо "`all`" можно написать имя какого-то одного элемента, когда необходимо использовать только его определение, например:

```
use ieee.std_logic_1164.std_ulogic;
```

Эта фраза обеспечит доступ только к определению типа `std_ulogic`, приведенному в табл. 4.32, без учета всех других родственных типов и функций. Однако, записывая подряд несколько предложений "`use`", можно добавить использование и других определений.

Библиотека БГУИР

VHDL-СТАНДАРТЫ IEEE

Язык VHDL предоставляет замечательную возможность расширять типы данных и множество функций. Это важное свойство, поскольку встроенные типы BIT и BIT_VECTOR, в действительности, вовсе не адекватны требованиям моделирования реальных схем, когда обрабатываемые сигналы могут соответствовать третьему состоянию, а также быть неизвестными, безразличными и меняющейся интенсивности.

Поэтому вскоре после формализации языка в стандарте IEEE 1076 коммерческие поставщики начали вводить свои собственные встроенные типы данных, чтобы оперировать логическими величинами, отличающимися от 0 и 1. У разных поставщиков были, конечно, различные определения для этих расширенных типов, из-за чего началось возведение «Вавилонской башни».

Чтобы избежать этого, Институтом инженеров по электротехнике и электронике был разработан стандартный логический пакет 1164 (std_logic_1164), девятизначная логическая система которого удовлетворяет бóльшую часть потребностей проектировщиков. За этим пакетом позднее последовал стандарт 1076-3, описываемый в разделе 5.9.6, который содержит несколько пакетов со стандартными типами и операциями для векторов STD_LOGIC, интерпретируемых как целые числа со знаком и без знака. Эти пакеты включают std_logic_arith, std_logic_signed и std_logic_unsigned.

Следование стандартам IEEE гарантирует разработчикам высокую степень переносимости их проектов и возможность их взаимодействия. Последнее становится все более важным, так как переход на очень большие специализированные ИС делает необходимой кооперацию не только многих проектировщиков, но также и многих поставщиков, каждый из которых вносит свой вклад в создаваемую по кусочкам «систему в одном кристалле».

Библиотека БГУИР

Табл. 4.40. Синтаксис определения VHDL-пакета

```
package package-name is
   type declarations
   signal declarations
   constant declarations
   component declarations
   function declarations
   procedure declarations
end package-name;
package body package-name is
   type declarations
   constant declarations
   function definitions
   procedure definitions
end package-name;
```

Стандартными пакетами не исчерпываются все возможности. Любой разработчик может написать свой собственный пакет согласно синтаксису, приведенному в табл. 4.40. Все элементы, объявленные между "`package`" и первым оператором "`end`", видны из любого файла проекта, использующего этот пакет; элементы, следующие за ключевым словом "`package body`", являются локальными. Заметьте, в частности, что первая часть содержит «объявления функций», но не определения. В *объявлении функции* (*function declaration*) перечислены только имя функции, аргументы и тип вплоть до ключевого слова "`is`" в табл. 4.35, но не включая его. Полное определение функции приводится в теле пакета и пользователи функции его не видят.

4.7.6. Элементы структурного проектирования

Теперь мы, наконец, готовы взглянуть на внутренности VHDL-проекта, на «исполняемую» часть архитектуры. Вспомним, что согласно табл. 4.28 тело архитектуры представляет собой ряд *параллельных операторов* (*concurrent statements*). В языке VHDL каждый параллельный оператор исполняется одновременно с другими параллельными операторами в теле данной архитектуры.

Такой режим исполнения заметно отличается от последовательного выполнения операторов в программном обеспечении, написанном на одном из обычных языков программирования. Параллельные операторы необходимы для того, чтобы отобразить поведение схемы, у которой соединенные между собой элементы воздействуют друг на друга непрерывно, а не в отдельные моменты времени, следующие один за другим в определенном порядке. Таким образом, правило моделирования состоит в том, что в случае, когда последним оператором в теле VHDL-архитектуры изменяется сигнал, используемый в первом операторе данной архитектуры, программа должна вернуться назад к первому оператору и скорректировать его результаты, приведя их в соответствие с только что изменившимся сигналом. На самом деле моделирующая программа будет осуществлять множественные изменения и обновления результатов до тех пор, пока моделируемая схема не стабилизируется; подробнее мы рассмотрим этот вопрос в разделе 4.7.9.

В языке VHDL есть несколько параллельных операторов, а также механизм связывания в один узел набора последовательных операторов с тем, чтобы они исполнялись, как один параллельный оператор. Различное использование параллельных операторов привело к возникновению трех стилей проектирования и описания схем, слегка отличающихся один от другого; об этом и пойдет речь в данном разделе и двух следующих.

Самым главным параллельным оператором в языке VHDL является *оператор* `component` (`component` *statement*), синтаксис которого указан в табл. 4.41. Здесь *component-name* – имя определенного ранее объекта, который должен быть использован или *подвергнут обработке* (*instantiate*) в теле данной архитектуры. Для каждого оператора `component` с обращением к названному объекту создается одна копия этого объекта; каждая копия должна иметь уникальную метку *label*.

Библиотека БГУИР

Табл. 4.41. Синтаксис оператора component в языке VHDL

```
label: component-name port map(signal1, signal2, ..., signaln);

label: component-name port map(port1=>signal1, port2=>signal2, ..., portn=>signaln);
```

Ключевое слово *port map* вводит список, посредством которого портам названного объекта ставятся в соответствие сигналы данной архитектуры. Список может быть представлен одним из двух различных способов. Первый из них является позиционным: как и в обычных языках программирования, сигналы, упоминаемые в списке, связываются с портами объекта в том же самом порядке, в каком порты перечислены в определении объекта. Второй способ записи – явный: каждый порт объекта связывается с сигналом посредством оператора "=>", и эти соответствия могут следовать в любом порядке.

До того как компонент будет подвергнут обработке внутри архитектуры, он должен быть декларирован *объявлением компонента* (*component declaration*) в определении архитектуры (см. табл. 4.28). Как видно из табл. 4.42, объявление компонента является по существу таким же, что и часть объявления соответствующего объекта, где объявляются порты: приводятся имя, режим и тип каждого порта.

Табл. 4.42. Синтаксис объявления компонента в языке VHDL

```
component component-name
  port (signal-names : mode signal-type;
        signal-names : mode signal-type;
        ...
        signal-names : mode signal-type);
end component;
```

Используемые в архитектуре компоненты могут быть либо ранее определенными элементами данного проекта, либо библиотечными элементами. Табл. 4.43 представляет собой пример VHDL-объекта и его архитектуры, в которой используются компоненты «устройства для обнаружения простых чисел», структурно идентичные отдельным вентилям в схеме на рис. 4.30(c). В объявлении объекта названы входы схемы и ее выход. В части архитектуры, отведенной под объявления, присваиваются имена всем сигналам и компонентам, которые используются внутри данной архитектуры. Этот пример был разработан и скомпилирован в программной среде Xilinx Foundation 1.5 (см. Обзор литературы), где INV, AND2, AND3 и OR4 являются предопределенными компонентами.

Заметьте, что операторы component в табл. 4.43 исполняются *параллельно*. Даже в том случае, когда операторы расположены в другом порядке, результатом синтеза будет та же самая схема и моделирование ее работы будет приводить к одному и тому же.

VHDL-архитектуру, в которой используются компоненты, часто называют *структурным описанием* (*structural description*) или *структурной моделью*

Библиотека БГУИР

(*structural design*), поскольку ею задается реализующая данный объект точная конфигурация соединений, по которым сигналы передаются от одного элемента к другому. В этом отношении ясное структурное описание эквивалентно схеме устройства или списку соединений в нем.

Табл. 4.43. Структурная VHDL-программа для устройства, обнаруживающего простые числа

```
library IEEE;
use IEEE.std_logic_1164.all;

entity prime is
    port ( N: in STD_LOGIC_VECTOR (3 downto 0);
           F: out STD_LOGIC );
end prime;

architecture prime1_arch of prime is
signal N3_L, N2_L, N1_L: STD_LOGIC;
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
component INV port (I: in STD_LOGIC; O: out STD_LOGIC); end component;
component AND2 port (I0,I1: in STD_LOGIC; O: out STD_LOGIC); end component;
component AND3 port (I0,I1,I2: in STD_LOGIC; O: out STD_LOGIC); end component;
component OR4 port (I0,I1,I2,I3: in STD_LOGIC; O: out STD_LOGIC); end component;
begin
  U1: INV port map (N(3), N3_L);
  U2: INV port map (N(2), N2_L);
  U3: INV port map (N(1), N1_L);
  U4: AND2 port map (N3_L, N(0), N3L_N0);
  U5: AND3 port map (N3_L, N2_L, N(1), N3L_N2L_N1);
  U6: AND3 port map (N2_L, N(1), N(0), N2L_N1_N0);
  U7: AND3 port map (N(2), N1_L, N(0), N2_N1L_N0);
  U8: OR4 port map (N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0, F);
end prime1_arch;
```

В некоторых приложениях бывает необходимо создать несколько копий определенного блока внутри архитектуры. В разделе 5.10.2 мы увидим, например, что *n*-разрядный «сумматор со сквозным переносом» можно образовать каскадным включением *n* «полных сумматоров». В языке VHDL имеется *оператор* `generate` (`generate` *statement*), который позволяет создавать такие повторяющиеся блоки посредством своего рода цикла “for” без необходимости выписывать все копии по отдельности.

Синтаксис простого итеративного цикла `generate` показан в табл. 4.44. Идентификатор *identifier* объявляется явно как переменная, тип который совместим с диапазоном *range*. Параллельный оператор *concurrent statement* исполняется однократно для каждого возможного значения переменной *identifier* в пределах диапазона; переменную *identifier* можно использовать внутри параллельного оператора. В табл. 4.45 показано, как можно построить 8-разрядный инвертор.

Табл. 4.44. Синтаксис цикла `for-generate` на языке VHDL

```
label: for identifier in range generate
          concurrent-statement
       end generate;
```

Библиотека БГУИР

Табл. 4.45. VHDL-объект и его архитектура для 8-разрядного инвертора

```
library IEEE;
use IEEE.std_logic_1164.all;

entity inv8 is
    port ( X: in STD_LOGIC_VECTOR (1 to 8);
           Y: out STD_LOGIC_VECTOR (1 to 8) );
end inv8;

architecture inv8_arch of inv8 is
component INV port (I: in STD_LOGIC; O: out STD_LOGIC); end component;
begin
  g1: for b in 1 to 8 generate
        U1: INV port map (X(b), Y(b));
      end generate;
end inv8_arch;
```

Значение константы должно быть известно к моменту компиляции программы, написанной на языке VHDL. Во многих приложениях бывает полезно разработать и откомпилировать объект и его архитектуру, оставляя некоторые из его параметров не заданными, например, разрядность шины. Сделать это позволяет имеющийся в языке VHDL инструмент "generic".

С помощью *объявления общности* (*generic declaration*) в объявлении объекта можно определить одну или большее число *настраиваемых констант* (*generic constant*); это необходимо сделать до объявления портов согласно синтаксису, указанному в табл. 4.46. Каждую поименованную константу можно использовать в определении архитектуры данного объекта, а задание ее значения откладывается до того момента, когда этот объект будет подвергаться обработке оператором `component` из другой архитектуры. Значения присваиваются настраиваемым константам в этом операторе `component` с помощью предложения *`generic map`* таким же способом, какой употреблен в предложении `port map`. В табл. 4.47 приведен пример, в котором одновременно используются инструмент `generic` и оператор `generate` для создания «шинного инвертора» с задаваемой пользователем разрядностью. В программе, представленной в табл. 4.48, обрабатывается несколько копий такого инвертора, каждый со своим числом сигнальных линий.

Табл. 4.46. Синтаксис объявления общности в объявлении объекта

```
entity entity-name is
  generic (constant-names : constant-type;
           constant-names : constant-type;
           ...
           constant-names : constant-type);
  port (signal-names : mode signal-type;
        signal-names : mode signal-type;
        ...
        signal-names : mode signal-type);
end entity-name;
```

Библиотека БГУИР

Табл. 4.47. VHDL-объект и его архитектура для шинного инвертора с произвольной разрядностью

```
library IEEE;
use IEEE.std_logic_1164.all;

entity businv is
    generic (WIDTH: positive);
    port ( X: in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
           Y: out STD_LOGIC_VECTOR (WIDTH-1 downto 0) );
end businv;

architecture businv_arch of businv is
component INV port (I: in STD_LOGIC; O: out STD_LOGIC); end component;
begin
  g1: for b in WID-1 downto 0 generate
        U1: INV port map (X(b), Y(b));
      end generate;
end businv_arch;
```

Табл. 4.48. VHDL-объект и его архитектура, в которой используется шинный инвертор с произвольной разрядностью

```
library IEEE;
use IEEE.std_logic_1164.all;

entity businv_example is
    port ( IN8: in STD_LOGIC_VECTOR (7 downto 0);
           OUT8: out STD_LOGIC_VECTOR (7 downto 0);
           IN16: in STD_LOGIC_VECTOR (15 downto 0);
           OUT16: out STD_LOGIC_VECTOR (15 downto 0);
           IN32: in STD_LOGIC_VECTOR (31 downto 0);
           OUT32: out STD_LOGIC_VECTOR (31 downto 0) );
end businv_example;

architecture businv_ex_arch of businv_example is
component businv
    generic (WIDTH: positive);
    port ( X: in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
           Y: out STD_LOGIC_VECTOR (WIDTH-1 downto 0) );
end component;
begin
U1: businv generic map (WIDTH=>8) port map (IN8, OUT8);
U2: businv generic map (WIDTH=>16) port map (IN16, OUT16);
U3: businv generic map (WIDTH=>32) port map (IN32, OUT32);
end businv_ex_arch;
```

Библиотека БГУИР

4.7.7. Элементы потокового проектирования

Если бы операторы component были единственными параллельными операторами языка VHDL, то он лишь немногим отличался бы от простого иерархического языка описания соединений со строгим соблюдением типов. Несколько дополнительных параллельных операторов языка VHDL позволяют описывать схему в терминах потока данных и выполняемых схемой операций над этими данными. Такой подход носит название *потокового описания* (*dataflow description*) или потокового проектирования (*dataflow design*).

В потоковых проектах используются два дополнительных параллельных оператора; они приведены в табл. 4.49. Чаще всего используется первый из них; он называется *параллельным сигнальным оператором присваивания* (*concurrent signal-assignment statement*). Его можно прочесть так: «Сигнал с именем *signal-name* принимает значение выражения *expression*». Поскольку в языке VHDL необходимо строго соблюдать типы, тип выражения *expression* должен быть совместим с типом сигнала *signal-name*. В общем случае это означает, что типы должны быть либо тождественно одинаковыми, либо тип *expression* должен являться подтипом типа *signal-name*. В случае массивов, тип элементов и длина должны быть согласованными, однако множество значений и направление изменения индекса могут не совпадать.

Табл. 4.49. Синтаксис параллельных сигнальных операторов присваивания в языке VHDL

```
signal-name <= expression;

signal-name <= expression when boolean-expression else
               expression when boolean-expression else
               ...
               expression when boolean-expression else
               expression;
```

В табл. 4.50 представлена архитектура объекта для устройства, обнаруживающего простые числа (см. табл. 4.43), записанная в потоковой форме. При таком подходе вентили и соединения между ними в явном виде не указываются; вместо этого используются встроенные операторы языка VHDL and, or и not. (На самом деле для сигналов типа STD_LOGIC таких встроенных операторов нет, но они определяются и перегружаются пакетом IEEE 1164.) Заметьте, что у оператора not самый высокий приоритет, так что для получения нужного результата не требуется заключать в скобки подвыражение типа "not N(3)".

Можно также воспользоваться второй, *условной* формой *параллельного сигнального оператора присваивания* (*conditional signal-assignment statement*) с ключевыми словами *when* и *else*, как показано в табл. 4.49. В этом случае в выражении *boolean-expression* отдельные булевы термы объединяются посредством встроенных булевых операторов языка VHDL, таких как and, or и not.

Библиотека БГУИР

Под булевыми термами обычно понимаются булевы переменные или результаты сравнения, выполняемого с помощью *операторов отношений* (*relational operators*) =, /= (не равно), >, >=, < и <=.

Табл. 4.50. Потоковая VHDL-архитектура для устройства, обнаруживающего простые числа

```
architecture prime2_arch of prime is
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
begin
  N3L_N0     <= not N(3)                              and N(0);
  N3L_N2L_N1 <= not N(3) and not N(2) and     N(1)          ;
  N2L_N1_N0  <=              not N(2) and     N(1) and N(0);
  N2_N1L_N0  <=                  N(2) and not N(1) and N(0);
  F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
end prime2_arch;
```

Табл. 4.51 содержит пример использования условных параллельных операторов присваивания. Каждый бит переменной типа STD_LOGIC, например, N(3), сравнивается со знаковыми литералами '1' и '0' и результат возвращается в виде значения типа boolean. Результаты сравнения объединяются в булево выражение, помещенное в каждом операторе между ключевыми словами when и else. В общем случае требуются предложения else; совокупный набор условий в каждом из операторов должен покрывать все возможные комбинации входных сигналов.

Табл. 4.51. Архитектура устройства для обнаружения простых чисел, в которой использованы условные присваивания

```
architecture prime3_arch of prime is
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
begin
  N3L_N0     <= '1' when N(3)='0' and N(0)='1' else '0';
  N3L_N2L_N1 <= '1' when N(3)='0' and N(2)='0' and N(1)='1' else '0';
  N2L_N1_N0  <= '1' when N(2)='0' and N(1)='1' and N(0)='1' else '0';
  N2_N1L_N0  <= '1' when N(2)='1' and N(1)='0' and N(0)='1' else '0';
  F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
end prime3_arch;
```

Параллельный оператор присваивания другого рода – это *избирательное присваивание сигналу его значения* (*selected signal-assignment statement*), синтаксис которого указан в табл. 4.52. Этот оператор вычисляет заданное выражение *expression* и присваивает сигналу с именем *signal-name* значение сигнала *signal-value*, соответствующее той из альтернатив *choices*, значение которой равно *expression*. Альтернативой в каждом предложении when может быть одиночное возможное значение *expression* или список значений, разделенных вертикальной чертой (|). Альтернативы *choices* в данном операторе должны быть взаимно исключающими и в совокупности включать все возможные случаи. В последнем предложении when можно воспользоваться ключевым словом others в качестве указания на все значения *expression*, которые еще не были упомянуты.

Библиотека БГУИР

```
with expression select
   signal-name <= signal-value when choices,
                  signal-value when choices,
                  ...
                  signal-value when choices;
```

Табл. 4.52. Синтаксис избирательного сигнального оператора присваивания в языке VHDL

В архитектуре устройства для обнаружения простых чисел, приведенной в табл. 4.53, использован избирательный сигнальный оператор присваивания. Все *альтернативы*, для которых F равно '1' могли бы быть записаны в одном предложении when; в нашем примере они разнесены по нескольким предложениям только с учебной целью. Здесь избирательный сигнальный оператор присваивания как бы считывает запись множества включений функции F.

```
architecture prime4_arch of prime is
begin
  with N select
    F <= '1' when "0001",
         '1' when "0010",
         '1' when "0011" | "0101" | "0111",
         '1' when "1011" | "1101",
         '0' when others;
end prime4_arch;
```

Табл. 4.53. Архитектура устройства для обнаружения простых чисел, в которой используется избирательное присваивание сигналу его значения

Ту же самую архитектуру можно слегка видоизменить, чтобы воспользоваться более удобной числовой интерпретацией N в определении функции. Применяя приведенное ранее преобразование CONV_INTEGER, можно записать *альтернативу* в терминах целых чисел, которые, как это можно видеть из табл. 4.54, являются простыми, что и требовалось. О таком варианте представления структуры можно говорить как о «поведенческом» описании, поскольку желаемая функция отображена в нем таким образом, что поведение устройства оказывается совершенно очевидным.

ПОЛНЫЙ ПЕРЕБОР

При условном и избирательном присваивании сигналу его значения требуется перечисление всех возможных условий. В условном сигнальном присваивании заключительной фразой "else *expression*" покрываются опущенные условия. При избирательном сигнальном присваивании все остающиеся условия можно подобрать ключевым словом "others" в последнем предложении when.

Глядя на табл. 4.53, можно подумать, что вместо слова "others" мы могли бы записать девять остающихся 4-битовых комбинаций "0000", "0100" и т.д. Но это не так! Не забывайте, что STD_LOGIC – это девятизначная система, так что у 4-разрядной величины типа STD_LOGIC_VECTOR в действительности имеется 9^4 возможных значений. Поэтому "others" в данном примере на самом деле покрывает 6554 случая!

Библиотека БГУИР

Табл. 4.54. Описание устройства для обнаружения простых чисел, носящее поведенческий характер

```
architecture prime5_arch of prime is
begin
  with CONV_INTEGER(N) select
    F <= '1' when 1 | 2 | 3 | 5 | 7 | 11 | 13,
         '0' when others;
end prime5_arch;
```

4.7.8. Элементы поведенческого проектирования

Как видно из последнего примера, иногда параллельным оператором можно непосредственно описать требуемое поведение логической схемы. И это очень хорошо, потому что возможность *поведенческого описания* (*behavioral description*) и выполнение *поведенческого проекта* (*behavioral design*) является главным достоинством языков описания схем вообще и языка VHDL, в частности. Однако для большинства поведенческих описаний нужны некоторые дополнительные элементы языка, рассматриваемые в этом разделе.

Ключевым поведенческим элементом языка VHDL является «процесс». *Процесс* (*process*) – это совокупность «последовательных» операторов (они будут описаны чуть ниже), которые выполняются одновременно с другими параллельными операторами и с другими процессами. С помощью процесса можно задать сложное взаимодействие сигналов и событий таким способом, что при моделировании это взаимодействие реализуется практически за нулевое время в модели, а результатом синтеза становится комбинационная или последовательностная схема, которая выполняет моделируемую операцию непосредственно.

Оператор процесса (*process statement*) в языке VHDL можно использовать повсюду, где возможно употребление параллельного оператора. Оператор процесса вводится ключевым словом `process`; синтаксис этого оператора приведен в табл. 4.55. Оператор `process` пишется внутри некоторой объемлющей архитектуры, поэтому ему доступны все типы, сигналы, константы, функции и процедуры, объявленные в этой архитектуре, а также так или иначе видимые из этой архитектуры. Но можно также определять и локальные типы, переменные, константы, функции и процедуры внутри данного процесса.

Табл. 4.55. Синтаксис оператора `process` в языке VHDL

```
process (signal-name, signal-name, ..., signal-name)
  type declarations
  variable declarations
  constant declarations
  function definitions
  procedure definitions
begin
  sequential-statement
  ...
  sequential-statement
end process;
```

Библиотека БГУИР

Обратите внимание на то, что внутри процесса можно объявлять только «переменные», но не сигналы. *Переменная* (*variable*) в языке VHDL отслеживает состояние процесса только внутри него и вне процесса ее не видно. В зависимости от того, как используется переменная, ей в конце концов будет или не будет соответствовать определенный сигнал при физической реализации создаваемой схемы. Синтаксис определения переменной внутри процесса подобен синтаксису объявления сигнала в архитектуре, за исключением того, что используется ключевое слово `variable`:

`variable` *variable-names* : *variable-type*;

VHDL-процесс всегда либо *выполняется* (*running process*), либо *приостановлен* (*suspended process*). Перечнем сигналов в определении процесса, который называется *списком чувствительности* (*sensitivity list*), задаются условия, когда процесс выполняется. Первоначально процесс остановлен; когда изменяется значение любого из сигналов в его списке чувствительности, исполнение процесса возобновляется, начиная с его первого последовательного оператора, и оно продолжается, пока не будет достигнут конец. Если какой-либо сигнал из списка чувствительности изменяет свое значение в результате исполнения процесса, то процесс выполняется снова. Это продолжается до тех пор, пока запуск процесса не перестанет приводить к изменению значения любого из этих сигналов. При моделировании все это происходит за нулевое время в модели.

Если процесс записан надлежащим образом, то, будучи запущен, он исполняется один или несколько раз и останавливается. Однако существует возможность записать процесс неправильно, который никогда не остановится. Рассмотрим, например, процесс всего с одним последовательным оператором "`X <= not X`" и списком чувствительности "`(X)`". Поскольку на каждом проходе значение `X` изменяется, процесс будет запущен навсегда, хотя и будет занимать нулевое время в модели. Едва ли это полезно! На практике в моделирующих программах имеются средства защиты, которые обычно обнаруживают подобное нежелательное поведение и прерывают исполнение такого процесса после, скажем, тысячи проходов.

Список чувствительности является необязательным; при моделировании исполнение процесса, у которого нет списка чувствительности, начинается в нулевой момент времени. Одно из применений такого процесса – это генерирование входных колебаний при тестировании (см. табл. 4.65 ниже).

В языке VHDL имеются последовательные операторы нескольких видов. Первый из них – это *последовательный сигнальный оператор присваивания* (*sequential signal-assignment statement*); у него тот же самый синтаксис, что и у параллельного аналога (*signal-name* <= *expression*;), но последовательный оператор располагается в теле процесса, а не в теле архитектуры. Аналогичный оператор для переменных – это *оператор присваивания значения переменной* (*variable-assignment statement*), синтаксис которого имеет вид: "*variable-name* : = *expression*;". Заметьте, что в случае переменных используется другой оператор присваивания, а именно : = .

Библиотека БГУИР

Ради учебных целей, потоковая архитектура устройства для обнаружения простых чисел из табл. 4.50 воспроизведена в табл. 4.56 как процесс. Обратите внимание, что мы все еще продолжаем совершенствовать архитектуру того же самого объекта prime, который первоначально был объявлен в табл. 4.43. В этой новой архитектуре (prime6_arch) имеется только один параллельный оператор; этим оператором является процесс. Список чувствительности процесса содержит только массив N, представляющий собой первичные сигналы на входах устройства, реализующего желаемую комбинационную логическую функцию. Выходы вентилей И необходимо задать как переменные, а не как сигналы, поскольку внутри процесса определения сигналов не разрешены. В противном случае тело процесса было бы очень похоже на тело исходной архитектуры. На самом деле типичные программные средства синтеза, вероятнее всего, построили бы одну и ту же схему по любому из этих описаний.

Табл. 4.56. Потоковая VHDL-архитектура устройства обнаружения простых чисел, основанная на использовании процесса

```
architecture prime6_arch of prime is
begin
  process(N)
    variable N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
  begin
    N3L_N0     := not N(3)                           and N(0);
    N3L_N2L_N1 := not N(3) and not N(2) and     N(1)        ;
    N2L_N1_N0  :=              not N(2) and     N(1) and N(0);
    N2_N1L_N0  :=                  N(2) and not N(1) and N(0);
    F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
  end process;
end prime6_arch;
```

ПРИЧУДЛИВОЕ ПОВЕДЕНИЕ

Помните, что операторы исполняются внутри процесса *последовательно.* Предположим, что по какой-то причине мы записали последний оператор в табл. 4.56 (присвоение значения сигналу F) первым. Тогда мы наблюдали бы довольно причудливое поведение процесса.

При первом запуске процесса моделирующая программа пожаловалась бы, что значения переменных считываются до того, как этим переменным присвоены какие-либо значения. При последующих возобновлениях процесса сигналу F присваивалось бы значение, основанное на *предыдущих* значениях переменных, которые сохранялись, пока процесс был приостановлен. Затем переменным присваивались бы новые значения, которые запоминались бы до очередного запуска процесса. Таким образом, значение сигнала на выходе схемы всегда отставало бы от значений входных сигналов на один шаг.

Библиотека БГУИР

Другие последовательные операторы, помимо простого присваивания, дают возможность творчески подойти к описанию поведения схемы. По-видимому, самый знакомый из них – это *оператор* `if` (`if` *statement*), синтаксис которого приведен в табл. 4.57. В первой и простейшей форме этого оператора проверяется булево выражение *boolean-expression* и, если оно имеет значение `true`, то исполняется последовательный оператор *sequential-statement*. Во второй форме добавляется предложение "`else`" с другим последовательным оператором *sequential-statement*, который исполняется, если булево выражение имеет значение `false`.

Табл. 4.57. Синтаксис оператора `if` в языке VHDL

```
if boolean-expression then sequential-statement
end if;

if boolean-expression then sequential-statement
else sequential-statement
end if;

if boolean-expression then sequential-statement
elsif boolean-expression then sequential-statement
...
elsif boolean-expression then sequential-statement
end if;

if boolean-expression then sequential-statement
elsif boolean-expression then sequential-statement
...
elsif boolean-expression then sequential-statement
else sequential-statement
end if;
```

Для образования вложенных операторов `if-then-else` в языке VHDL используется специальное ключевое слово *elsif*, которое вводит «средние» предложения. Последовательный оператор *sequential-statement* предложения `elsif` исполняется в том случае, когда булево выражение *boolean-expression* в этом предложении истинно, а все предшествующие булевы выражения *boolean-expressions* оказываются ложными. Последовательный оператор *sequential-statement* заключительного необязательного предложения `else` исполняется только тогда, когда все предыдущие выражения *boolean-expressions* имеют значения `false`.

В табл. 4.58 представлен вариант архитектуры устройства для обнаружения простых чисел, в котором используется оператор `if`. Локальная переменная `NI` введена для того, чтобы отобразить преобразованное значение входного воздействия `N` в виде целого числа; это позволяет оперировать целыми числами при сравнениях в операторе `if`.

Булевы выражения в табл. 4.58 не перекрываются, то есть в любой момент времени значение `true` имеет только одно из них. На самом деле в данном при-

ложении не было необходимости использовать в полном объеме возможности вложенных операторов `if`. С помощью средств синтеза можно было бы построить схему, в которой вычисление логических выражений происходило последовательно, но схема при этом работала бы медленнее. Когда нужно выбирать среди нескольких альтернатив на основании значения только одного сигнала или выражения, обычно более читабельным и дающим лучший результат синтеза является *оператор* `case` (`case` *statement*).

Табл. 4.58. Архитектура устройства для обнаружения простых чисел, в которой использован оператор `if`

```
architecture prime7_arch of prime is
begin
  process(N)
    variable NI: INTEGER;
  begin
    NI := CONV_INTEGER(N);
    if NI=1 or NI=2 then F <= '1';
    elsif NI=3 or NI=5 or NI=7 or NI=11 or NI=13 then F <= '1';
    else F <= '0';
    end if;
  end process;
end prime7_arch;
```

Синтаксис оператора `case` представлен в табл. 4.59. В этом операторе вычисляется заданное выражение *expression*, по его значению выбирается одна из альтернатив *choices* и исполняются соответствующие последовательные операторы *sequential-statements*. Заметьте, что в каждом из наборов альтернатив *choices* можно записать один или большее число последовательных операторов. Сами альтернативы *choices* могут иметь форму одного значения или нескольких значений, разделенных вертикальной чертой (|). Альтернативы *choices* должны быть взаимно исключающими и содержать все возможные значения типа выражения *expression*; в последней альтернативе *choices* можно воспользоваться ключевым словом `others` для указания всех значений, которые еще не были упомянуты ранее.

Табл. 4.59. Синтаксис оператора `case` в языке VHDL

```
case expression is
  when choices => sequential-statements
  ...
  when choices => sequential-statements
end case;
```

В табл. 4.60 приведена еще одна архитектура устройства для обнаружения простых чисел, на этот раз записанная с использованием оператора `case`. Подобно тому, как это было в варианте с параллельным оператором `select`, оператор `case` позволяет в очень наглядной форме задать желаемое функциональное поведение.

Библиотека БГУИР

Табл. 4.60. Архитектура устройства для обнаружения простых чисел, в которой использован оператор `case`

```
architecture prime8_arch of prime is
begin
  process(N)
  begin
    case CONV_INTEGER(N) is
      when 1 => F <= '1';
      when 2 => F <= '1';
      when 3 | 5 | 7 | 11 | 13 => F <= '1';
      when others => F <= '0';
    end case;
  end process;
end prime8_arch;
```

Другой важный класс последовательных операторов образуют *операторы* `loop` (`loop` *statements*). Синтаксис простейшего из них указан в табл. 4.61; в этом примере возникает бесконечный цикл. Хотя в обычных языках, на которых пишется программное обеспечение, бесконечные циклы нежелательны, мы увидим в параграфе 7.12, как можно с большой пользой употребить такой цикл при моделировании работы схемы.

```
loop
  sequential-statement
  ...
  sequential-statement
end loop;
```

Табл.4.61. Синтаксис основного оператора `loop` в языке VHDL

Более знакомым вариантом цикла является уже рассматривавшийся выше *цикл* `for` (`for` *loop*), синтаксис которого указан в табл. 4.62. Заметьте, что переменная цикла *identifier* объявляется неявно ее упоминанием в цикле `for`, и она имеет тот же тип, что и диапазон *range*. Эту переменную можно использовать в последовательных операторах внутри цикла, и посредством ее перебираются все значения диапазона *range* слева направо по мере перехода от одного шага итерации к другому.

```
for identifier in range loop
  sequential-statement
  ...
  sequential-statement
end loop;
```

Табл. 4.62. Синтаксис цикла `for` в языке VHDL

Имеются еще два других полезных последовательных оператора, которые могут исполняться внутри цикла; это *операторы* `exit` (`exit` *statement*) и `next`

Библиотека БГУИР

(`next` *statement*). Оператор `exit`, когда он исполняется, передает управление оператору, непосредственно следующему за концом цикла. С другой стороны, оператор `next` вызывает пропуск всех остающихся в цикле операторов и переход к началу следующей итерации данного цикла.

Наше старое доброе устройство для обнаружения простых чисел вновь представлено в табл. 4.63, на этот раз – в виде архитектуры, в которой использован цикл `for`. Замечательно то, что в данном примере дается истинно поведенческое описание: здесь язык VHDL на самом деле используется для определения того, является входное воздействие N простым числом или нет. Мы увеличили также размерность массива N до 16 разрядов, чтобы подчеркнуть тот факт, что мы способны теперь создавать компактные модели схем, не перечисляя в явном виде сотни простых чисел.

Табл. 4.63. Архитектура устройства для обнаружения простых чисел, в которой использован оператор `for`

```
library IEEE;
use IEEE.std_logic_1164.all;

entity prime9 is
    port ( N: in STD_LOGIC_VECTOR (15 downto 0);
           F: out STD_LOGIC );
end prime9;

architecture prime9_arch of prime9 is
begin
  process(N)
  variable NI: INTEGER;
  variable prime: boolean;
  begin
    NI := CONV_INTEGER(N);
    prime := true;
    if NI=1 or NI=2 then null; -- take care of boundary cases
    else for i in 2 to 253 loop
           if NI mod i = 0 then
             prime := false; exit;
           end if;
        end loop;
    end if;
    if prime then F <= '1'; else F <= '0'; end if;
  end process;
end prime9_arch;
```

Оператор `loop` последнего вида – это *цикл* `while` (`while` *loop*), синтаксис которого приведен в табл. 4.64. В такой конструкции булево выражение *boolean-expression* вычисляется перед началом каждой итерации, и цикл выполняется только до тех пор, пока значение этого выражения остается равным `true`.

Библиотека БГУИР

ПЛОХОЙ ПРОЕКТ

Приведенный в табл. 4.63 вариант структуры – это хороший пример применения цикла `for` и плохой пример проектирования схемы. Хотя VHDL и является мощным языком программирования, использование всех его возможностей в полном объеме при описании устройства может оказаться неэффективным, а само проектируемое устройство – несинтезируемым.

Виновник этого в табл. 4.63 – оператор `mod`. Для этой операции требуется деление целых чисел, тогда как большинство программных средств, ориентированных на язык VHDL, не в состоянии синтезировать схемы деления за исключением особых случаев типа деления на степень 2 (реализуемого сдвигом).

Но даже если бы программные средства могли синтезировать делители, мы не захотели бы задавать структуру устройства для обнаружения простых чисел в таком виде. Приведенное в табл. 4.63 описание предполагает создание комбинационной схемы, поэтому программные средства должны были бы образовать 252 комбинационных делителя, по одному на каждое значение `i`, для «развертывания» цикла `for` и реализации схемы!

Процессы можно использовать для описания поведения как комбинационных, так и последовательностных схем. В главе 5, в разделах, относящихся к языку VHDL, будет дано много больше примеров описания комбинационных схем. Для описания последовательностных схем понадобятся некоторые дополнительные свойства этого языка; они будут рассмотрены в параграфе 7.12, а примеры проектирования последовательностных схем будут приведены в VHDL-разделах главы 8.

4.7.9. Отсчет времени и моделирование

Ни в одном из примеров, с которыми мы имели дело до сих пор, отсчет времени при работе схемы не моделировался: все происходило в модели за нулевое время. Однако в языке VHDL имеются замечательные средства отображать ход времени, и это является еще одной действительно важной стороной данного языка. В нашей книге мы не станем вдаваться в подробности по этому поводу, но несколько идей здесь все же приведем.

Язык VHDL позволяет с помощью ключевого слова *after* задавать временную задержку в любом сигнальном операторе присваивания, в том числе при последовательном, параллельном, условном и избирательном присваивании. Например, архитектуру вентиля запрета из табл. 4.26 можно было бы записать в следующем виде:

```
Z <= '1' after 4 ns when X='1' and Y='0' else '0' after 3 ns;
```

Этим моделируется вентиль запрета с задержкой, равной 4 нс при переходе выходного сигнала из 0 в 1, и с задержкой всего 3 нс при переходе из 1 в 0. В типичной среде проектирования специализированных ИС такими параметрами задержки наделены написанные на языке VHDL модели всех библиотечных компонентов

Библиотека БГУИР

нижнего уровня. Эти оценки позволяют моделирующей VHDL-программе приближенно предсказывать временнóе поведение больших по размеру схем, составленных из таких компонентов.

Другой способ включения отсчета времени предоставляет последовательный *оператор* `wait` (`wait` *statement*). Этим оператором можно воспользоваться, чтобы приостановить процесс на заданное время. В табл. 4.65 приведена в качестве примера программа, в которой оператор `wait` использован для того, чтобы сформировать в модели входное воздействие для тестирования вентиля запрета, состоящее в переборе четырех различных комбинаций входных сигналов с шагом 10 нс по оси времени.

Табл. 4.64. Синтаксис цикла `while` в языке VHDL

```
while boolean-expression loop
   sequential-statement
   ...
   sequential-statement
end loop;
```

Табл. 4.65. Использование оператора `wait` языка VHDL для генерирования входных воздействий при тестировании

```
entity InhibitTestBench is
end InhibitTestBench;

architecture InhibitTB_arch of InhibitTestBench is
component Inhibit port (X,Y: in BIT; Z: out BIT); end component;
signal XT, YT, ZT: BIT;
begin
  U1: Inhibit port map (XT, YT, ZT);
  process
  begin
    XT <= '0'; YT <= '0';
    wait for 10 ns;
    XT <= '0'; YT <= '1';
    wait for 10 ns;
    XT <= '1'; YT <= '0';
    wait for 10 ns;
    XT <= '1'; YT <= '1';
    wait; -- this suspends the process indefinitely
  end process;
end InhibitTB_arch;
```

Когда уже имеется VHDL-программа, корректная с точки зрения синтаксиса, для наблюдения за ее работой можно воспользоваться VHDL-средствами моделирования. Мы не будем подробно рассматривать эти вопросы, но составить общее представление о том, как работает моделирующая программа, полезно.

В момент, когда моделирующая программа начинает работать, *время в модели* (*simulation time*) равно нулю. В этот момент всем сигналам присваиваются их

Библиотека БГУИР

значения по умолчанию (от которых работа вашей программы не должна зависеть!). Инициализируются также сигналы и переменные, начальные значения которых объявлены явно (мы не говорим, как это делается). Затем моделирующая программа начинает исполнение всех процессов и параллельных операторов в данном проекте.

Конечно, моделирующая программа не может фактически обеспечить одновременное протекание процессов и исполнение параллельных операторов, но она притворяется, будто делает это, с помощью упорядоченного по времени «списка событий» и «матрицы чувствительности к сигналам». Заметьте, что каждый параллельный оператор эквивалентен одному процессу.

В нулевой момент времени в модели моделирующая программа готова начать исполнение всех процессов и выбирает один из них. Выполняются все последовательные операторы этого процесса, включая циклы, если они предусмотрены. Когда исполнение этого процесса заканчивается, выбирается еще один процесс и так далее, до тех пор пока не будут выполнены все процессы. Этим завершается *цикл моделирования* (*simulation cycle*).

Во время исполнения процесса могут возникать новые значения сигналов. Немедленного присвоения сигналам этих значений не происходит; вместо этого новые значения помещаются в *список событий* (*event list*), и намечается, что они станут эффективными в определенный момент времени. Если присвоение отнесено к явно заданному времени в модели (например, с задержкой, указанной в предложении `after`), то в списке событий предусматривается выполнение данного действия именно в этот момент времени. В противном случае считается, что присвоение должно произойти «немедленно»; однако в действительности реализация этого события откладывается до момента времени, равного текущему времени в модели плюс один «элементарный сдвиг по времени». Под *элементарным сдвигом по времени* (*delta delay*) понимается бесконечно короткий отрезок времени, такой что текущее время в модели плюс любое число элементарных сдвигов все еще остается равным тому же самому значению. Этот принцип позволяет, когда необходимо, многократно исполнять процесс в модели за нулевое время.

После того как цикл моделирования завершен, просматривается список событий в поисках одного или нескольких сигналов, которые изменяются в ближайший очередной момент времени. Таким моментом может быть более позднее время, отличающееся от текущего лишь на один элементарный сдвиг, либо очередной момент может определяться реальной задержкой в схеме, и тогда время в модели продвигается до этого момента. В любом случае осуществляется запланированное изменение сигнала. Некоторые из процессов могут быть чувствительны к изменяющимся сигналам согласно сведениям, хранящимся в матрице чувствительности к сигналам. В ней для каждого сигнала указано, у каких процессов этот сигнал имеется в списке чувствительности. (У процесса, эквивалентного параллельному оператору, в список чувствительности заносятся *все* его управляющие сигналы и данные.) Все процессы, чувствительные к только что изменившемуся сигналу, намечаются для исполнения в очередном цикле моделирования, к которому моделирующая программа и приступает.

Библиотека БГУИР

Двухфазная работа моделирующей программы на каждом цикле моделирования, сопровождающаяся просмотром списка событий и составлением очередного расписания присвоения сигналам их новых значений, продолжается неограниченно долго, до тех пор, пока список событий не будет исчерпан. На этом моделирование завершается.

Механизм списка событий позволяет имитировать исполнение параллельных процессов, несмотря на то, что работа моделирующей программы представляет собой один поток операций, выполняемых на единственном компьютере. А механизм элементарных сдвигов по времени обеспечивает правильность работы моделирующей программы, хотя может потребоваться многократное исполнение процесса или набора процессов в моменты времени, разделенные элементарными сдвигами, прежде чем изменяющиеся сигналы приобретут свои установившиеся значения. Этот механизм используется также для обнаружения выходящих из-под контроля процессов (типа "`X <= not X`"); если в результате выполнения тысячи циклов моделирования на тысяче элементарных сдвигов по времени не происходит продвижение времени в модели на сколько-нибудь «реальную» величину, то, вероятнее всего, что-то не так.

4.7.10. Синтез

Как мы упоминали в начале этого параграфа, язык VHDL первоначально предназначался для описания логических схем и моделирования и лишь позднее был приспособлен для синтеза. В этом языке много конструктивов и излишеств, которые не могут быть синтезированы. Однако представленные в этом параграфе сокращенные версии языка и стили написания программ в общем случае обеспечивают синтезирование большинством программных средств.

Но все же от того, как вы напишете программу, сильно зависит качество схемы, которую вы получите в результате синтеза. Приведем несколько примеров.

- Конструкции с «последовательным» управлением типа `if-elsif-elsif-else` могут приводить к цепочке последовательно включенных логических схем, проверяющих соответствующие условия. Если условия являются взаимно исключающими, то лучше использовать операторы `case` или `select`.
- Цикл в процессе обычно «разворачивается» в множество копий той или иной комбинационной логической схемы, чтобы обеспечить исполнение этого оператора. Если вы хотите, чтобы эта комбинационная логическая схема была только в одном экземпляре и использовалась последовательно шаг за шагом, то вы должны спроектировать последовательностную схему; о таких схемах речь пойдет в последующих главах.
- Если вы ошибетесь в условном операторе в процессе, не указав выход для какой-либо комбинации входных сигналов, то это приведет к созданию компилятором «защелки» для удержания старого значения сигнала, который в противном случае мог бы измениться. В общем случае образование таких защелок нежелательно.

Кроме того, в зависимости от реализации программных средств могут оказаться несинтезируемыми и другие элементы и структуры языка. Вам, безусловно, сле-

Библиотека БГУИР

дует обратиться к документации, чтобы узнать, что запрещено, что разрешено и что рекомендуется в вашем конкретном случае.

В предвидимом будущем проектировщикам, использующим программные средства синтеза, для получения хороших результатов придется обратить довольно пристальное внимание на стиль написания своих программ. А в настоящее время критерий «хорошего стиля программирования» в определенной степени зависит как от средств синтеза, так и от технологии микросхем, в которых нужно разместить результат синтеза. Хотя примеры, которые будут приведены в оставшейся части этой книги, будут синтаксически и семантически корректными, они дадут лишь поверхностное представление о методах написания программ на языках описания схем, которые применяются в больших проектах. Искусство и практика проектирования больших устройств с использованием языков описания схем все еще находятся в состоянии интенсивного развития.

5.2. Временные соотношения в схеме

«Время решает все» – это справедливо для инвестиций, в комедии, да и в цифровом проектировании. Как мы узнали в параграфе 3.6, в реальных схемах требуется время для возникновения на выходе реакции на входное воздействие, а многие современные схемы и системы настолько быстры, что существенной становится даже задержка сигнала, распространяющегося «со скоростью света» от выхода одного логического элемента до входа другого в пределах платы или корпуса микросхемы.

Большинство цифровых систем представляют собой последовательно включенные схемы, которые работают в пошаговом режиме, управляемые периодическим синхронизирующим сигналом, и частота этого сигнала ограничена максимальным временем, которое требуется для завершения всех операции на одном шаге. Таким образом, чтобы создавать быстродействующие схемы, которые работали бы правильно при всех условиях, разработчики цифровой аппаратуры должны четко представлять себе временные соотношения.

Последние несколько лет продемонстрировали существенные количественные и качественные достижения средств автоматизированного проектирования схем в отношении анализа временных диаграмм. Тем не менее, довольно часто наиболее сложной проблемой при разработке печатных плат и особенно специализированных ИС является достижение требуемых временных характеристик. В этом параграфе мы начинаем с основ, так что вы поймете, чтó делают программные средства, когда вы пользуетесь ими, а также разберетесь, как исправлять свои схемы, когда временные соотношения оказываются не совсем удачными.

Библиотека БГУИР

5.2.1. Временные диаграммы

Временнáя диаграмма (*timing diagram*) иллюстрирует характер изменения сигналов в цифровой схеме в зависимости от времени. Временные диаграммы составляют важную часть документации любой цифровой системы. Ими можно воспользоваться как для объяснения временных соотношений между сигналами в пределах системы, так и для того, чтобы сформулировать временные требования к внешним сигналам, которые поступают в систему.

На рис. 5.19(а) изображена блок-схема простого комбинационного устройства с двумя входами и двумя выходами. На рис. 5.19(b) показана задержка выходных сигналов относительно входного сигнала GO в предположении, что входной сигнал ENB зафиксирован на постоянном уровне. В каждой осциллограмме верхняя линия соответствует логической 1, а нижняя линия – логическому 0. Переходы сигналов изображены в виде наклонных линий, чтобы напоминать нам о том, что в реальных схемах изменения сигналов не происходят мгновенно. (Кроме того, наклонные линии лучше смотрятся, чем вертикальные.)

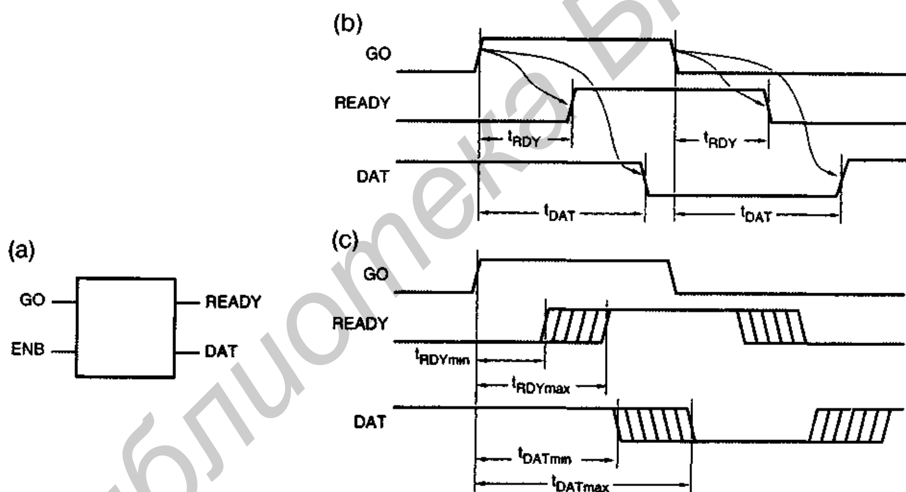


Рис. 5.19. Временные диаграммы для комбинационной схемы: (а) блок-схема устройства; (b) причинная обусловленность и задержка распространения; (с) минимальные и максимальные задержки.

Иногда, особенно в случае сложных временных диаграмм, стрелками отражают *причинную обусловленность связи* (*causality*), то есть информацию о том, изменение какого входного сигнала вызывает изменение того или иного выходного сигнала. В любом случае наиболее важной информацией, которую дают временные диаграммы, является величина *задержки* (*delay*) между переходами в сигналах.

Различные пути прохождения сигнала в схеме могут иметь различные задержки. На рис. 5.19(b), например, показано, что задержка между сигналами GO и READY меньше, чем задержка между сигналами GO и DAT. Точно так же задержка между

Библиотека БГУИР

входным сигналом ENB и выходными сигналами может быть различной, и это можно было бы наблюдать на другой временнóй диаграмме. Как мы увидим позже, задержка при прохождении сигнала по некоторому заданному пути может быть различной в зависимости от того, в каком направлении изменяется сигнал на выходе: от низкого уровня до высокого или от высокого уровня до низкого (это явление не отражено на рисунке).

В реальной схеме задержка обычно измеряется между средними точками переходов в сигналах, как показано на наших временных диаграммах. Одна единственная временнáя диаграмма может содержать много различной информации о задержках. Каждая задержка имеет свое обозначение; на нашем рисунке это t_{RDY} и t_{DAT}. В сложных временных диаграммах задержки обычно нумеруются для облегчения ссылки на них (например: $t_1, t_2, ..., t_{42}$). В любом случае временнáя диаграмма обычно сопровождается *таблицей временных параметров* (*timing table*), в которой указаны величина каждой задержки и условия, при которых задержка имеет такое значение.

Величина задержки редко определяется одним числом, так как задержки в реальных цифровых компонентах могут сильно зависеть от напряжения питания, температуры и различных производственных факторов. Поэтому в таблице временных параметров может быть указан диапазон значений, определяющий *минимальное, типичное* и *максимальное* значение для каждой задержки. Наличие некоторого диапазона возможных значений задержки иногда отражается на временнóй диаграмме, как показано на рис. 5.19(с), где переходы происходят не в строго фиксированные моменты времени.

Для некоторых сигналов нет необходимости изображать на временнóй диаграмме, как именно меняется значение сигнала в конкретный момент времени: от 1 до 0 или от 0 до 1; достаточно только показать, что переход имеет место. Этим свойством обладает любой сигнал, который несет информацию о бите «данных»: фактическое значение бита данных изменяется в зависимости от привходящих обстоятельств, но, независимо от его значения, бит передается, сохраняется или обрабатывается в определенный момент времени относительно «управляющих» сигналов в системе. Временнáя диаграмма на рис. 5.20(а) поясняет эту идею. Сигнал «данные» обычно имеет установившееся значение, равное 0 или 1, а переходы происходят только в определенные моменты времени. Понятие о не строго фиксированной величине задержки можно также применить к сигналам «данные», как показано на рисунке для сигнала DATAOUT.

В цифровых системах обработка группы сигналов данных в шине довольно часто производится идентичными схемами. В этом случае все сигналы в шине имеют одни и те же временные параметры и могут быть представлены одной временнóй диаграммой и соответствующей записью в таблице временных параметров. Если известно, что сигналы в шине в течение определенного времени имеют конкретные постоянные значения, то иногда это отображают на временнóй диаграмме двоичными, восьмеричными или шестнадцатеричными числами, как показано на рис. 5.20(b).

Библиотека БГУИР

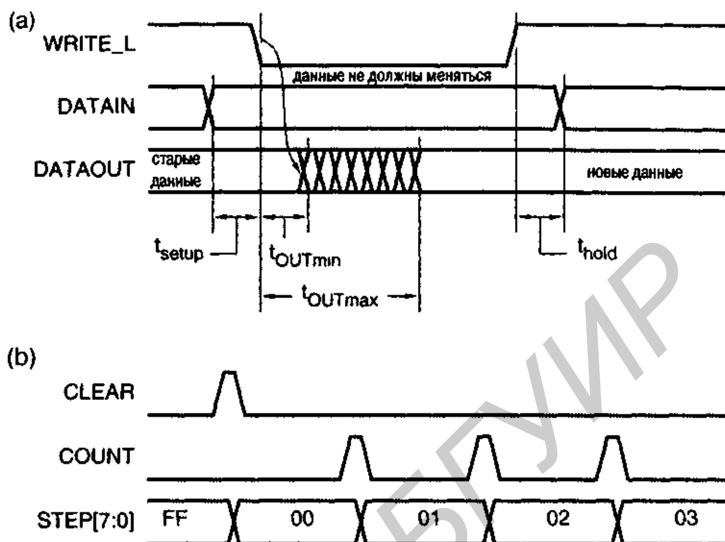


Рис. 5.20. Временные диаграммы сигналов «данные»: (а) фиксированные и не строго фиксированные моменты переходов; (b) последовательность значений сигналов на 8-разрядной шине

Библиотека БГУИР

5.2.2. Задержка распространения

В разделе 3.6.2 мы формально определили *задержку распространения* сигнала как время, затрачиваемое для того, чтобы изменение сигнала во входной цепи привело к изменению сигнала в выходной цепи. В комбинационной схеме с несколькими входами и выходами много различных путей прохождения сигнала, и каждый путь может иметь свое значение задержки распространения. Кроме того, задержка распространения при изменении выходного сигнала от низкого уровня до высокого (t_{pLH}) может отличаться от задержки, когда этот сигнал изменяется от высокого уровня до низкого (t_{pHL}).

Производитель комбинационных ИС обычно указывает все эти различные задержки распространения или, по крайней мере, задержки, которые могут представлять интерес в типичных приложениях. При объединении различных ИС в конструкцию бóльших размеров, для анализа временных соотношений в схеме в целом используются индивидуальные характеристики отдельных микросхем. Задержка распространения сигнала при прохождении через все логические элементы схемы равна сумме задержек, вносимых отдельными элементами.

5.2.3. Временные параметры

Временные параметры устройства можно охарактеризовать минимальным, типичным и максимальным значениями для каждой задержки распространения и для каждого направления изменения сигнала:

- *Максимальная задержка* (*maximum delay*). Этот параметр является одним из наиболее часто используемых опытными разработчиками, так как задержка распространения «никогда» не может превышать максимального значения. Однако понятие «никогда» варьируется от семейства к семейству и от изготовителя к изготовителю. Например, «максимальные» задержки распространения ТТЛ-схем 74LS и 74S бывают указаны для напряжения питания V_{CC} = 5 В при температуре T_A = 25°С и при почти отсутствующей емкостной нагрузке. Если напряжение питания или температура изменяются или если емкостная нагрузка больше 15 пФ, то задержка может быть больше. С другой стороны, «максимальная» задержка распространения для микросхем 74АС и 74АСТ относится ко всему рабочему диапазону напряжений питания и температур и к случаю большей емкостной нагрузки, равной 50 пФ.
- *Типичная задержка* (*typical delay*). Этот один из параметров, чаще всего используемых теми разработчиками, которые не предполагают находиться вблизи своего изделия, когда оно покинет дружественную среду лаборатории и будет отправлено заказчикам. «Типичная» задержка – это величина, которая характеризует устройство, изготовленное в хороший день и работающее при почти идеальных условиях.
- *Минимальная задержка* (*minimum delay*). Это наименьшее значение задержки распространения, которое когда-либо может быть получено. Работа большинства правильно построенных схем не зависит от этого параметра; то есть они будут нормально работать, даже если задержка нулевая. На это следует обращать внимание, потому что производители не указывают минимальную задержку для большинства логических семейств с умеренным быстродействием, включая ТТЛ-семейства 74LS и 74S. Однако у быстродействующих ИС, включая схемы ЭСЛ, а также КМОП-семейства 74АС и 74АСТ, минимальная задержка указывается и она отлична от нуля; это позволяет разработчику гарантированно выполнить временные требования, предъявляемые защелками и триггерами, которые рассматриваются в параграфе 7.2

Табл. 5.2 содержит типичные и максимальные значения задержки для некоторых КМОП- и ТТЛ-схем серии 74. В табл. 5.3 приведены те же самые параметры для большинства КМОП- и ТТЛ-схем средней степени интеграции, которые появятся в дальнейшем в этой главе.

НАСКОЛЬКО ТИПИЧНО ТИПИЧНОЕ?

Большинство ИС, возможно до 99%, действительно изготовлены в «хорошие» дни и имеют величину задержки, близкую к «типичному» значению. Однако если вы проектируете систему, которая работает только в том случае, когда все 100 используемых в ней ИС имеют «типичные» временные параметры, то, согласно теории вероятностей, система не будет работать с вероятностью $(1 - 0.99^{100})\cdot 100\% = 63\%$. Впрочем, взгляните на следующее рассуждение, вынесенное за пределы основного текста...

Библиотека БГУИР

Табл. 5.2. Задержка распространения в наносекундах некоторых 5-вольтовых КМОП- и ТТЛ-схем малой степени интеграции

Номер микросхемы	*74HCT*		*74AHCT*				*74LS*			
	Тип.	Макс.	Тип.		Макс.		Тип.		Макс.	
	t_{pLH}, t_{pHL}	t_{pLH}, t_{pHL}	t_{pLH}	t_{pHL}	t_{pLH}	t_{pHL}	t_{pLH}	t_{pHL}	t_{pLH}	t_{pHL}
'00, '10	11	35	5.5	5.5	9.0	9.0	9	10	15	15
'02	9	29	3.4	4.5	8.5	8.5	10	10	15	15
'04	11	35	5.5	5.5	8.5	8.5	9	10	15	15
'08, '11	11	35	5.5	5.5	9.0	9.0	8	10	15	20
'14	16	48	5.5	5.5	9.0	9.0	15	15	22	22
'20	11	35					9	10	15	15
'21	11	35					8	10	15	20
'27	9	29	5.6	5.6	9.0	9.0	10	10	15	15
'30	11	35					8	13	15	20
'32	9	30	5.3	5.3	8.5	8.5	14	14	22	22
'86 (2 уровня)	13	40	5.5	5.5	10	10	12	10	23	17
'86 (3 уровня)	13	40	5.5	5.5	10	10	20	13	30	22

СЛЕДСТВИЕ ЗАКОНА МЭРФИ

Закон Мэрфи гласит: «Если какая-то неприятность может случиться, то она произойдет». Следствие этого закона выглядит так: «Если вы хотите, чтобы случилась какая-то неприятность, то она не произойдет».

Имея в виду предыдущий пример, вы, возможно, полагаете, что при работе в лабораторных условиях с вероятностью 63% потенциальные проблемы, связанные с временными характеристиками, обнаруживаются. Однако проблемы не распределены равномерно, так как все ИС из данной партии склонны вести себя примерно одинаково. Следствие из закона Мэрфи говорит, что *все опытные образцы* изделия будут собраны из микросхем одной и той же «хорошей» партии. Поэтому все будет прекрасно работать в течение времени, достаточного только для того, чтобы система была принята в массовое производство и все начали радоваться и поздравлять себя с успехом.

Затем, без ведома производственного отдела, от поставщика прибывает «медленная» партия ИС какого-нибудь типа, устанавливаемых в каждой изготовляемой системе, так что *все перестает работать*. Инженеры-технологи суетятся, пытаясь разобраться в проблеме (не простой, потому что разработчик закопался и не потрудился представить описание схемы), а тем временем компания терпит крупные убытки, поскольку не в состоянии отправлять свою продукцию.

Библиотека БГУИР

Табл. 5.3. Задержка распространения в наносекундах некоторых КМОП- и ТТЛ-схем средней степени интеграции (any select – любой вход выбора, any data – любой вход данных, enable – вход разрешения, any input – любой вход, select – вход выбора, any – любой, output – выход)

			74HCT		*74AHCT / FCT*		*74LS*			
			Тип.	*Макс.*	*Тип.*	*Макс.*	*Тип.*		*Макс.*	
Номер микросхемы	***От***	***До***	t_{pLH}, t_{pHL}	t_{pLH}, t_{pHL}	t_{pLH}, t_{pHL}	t_{pLH}, t_{pHL}	t_{pLH}	t_{pHL}	t_{pLH}	t_{pHL}
'138	any select	output (2)	23	45	8.1 / 5	13 / 9	11	18	20	41
	any select	output (3)	23	45	8.1 / 5	13 / 9	21	20	27	39
	$\overline{G2A}, \overline{G2B}$	output	22	42	7.5 / 4	12 / 8	12	20	18	32
	G1	output	22	42	7.1 / 4	11.5 / 8	14	13	26	38
'139	any select	output (2)	14	43	6.5 / 5	10.5 / 9	13	22	20	33
	any select	output (3)	14	43	6.5 / 5	10.5 / 9	18	25	29	38
	enable	output	11	43	5.9 / 5	9.5 / 9	16	21	24	32
'151	any select	Y	17	51	- / 5	- / 9	27	18	43	30
	any select	$\overline{Y}$	18	54	- / 5	- / 9	14	20	23	32
	any data	Y	16	48	- / 4	- / 7	20	16	32	26
	any data	$\overline{Y}$	15	45	- / 4	- / 7	13	12	21	20
	enable	Y	12	36	- / 4	- / 7	26	20	42	32
	enable	$\overline{Y}$	15	45	- / 4	- / 7	15	18	24	30
'153	any select	output	14	43	- / 5	- / 9	19	25	29	38
	any data	output	12	43	- / 4	- / 7	10	17	15	26
	enable	output	11	34	- / 4	- / 7	16	21	24	32
'157	select	output	15	46	6.8 / 7	11.5 / 10.5	15	18	23	27
	any data	output	12	38	5.6 / 4	9.5 / 6	9	9	14	14
	enable	output	12	38	7.1 / 7	12.0 / 10.5	13	14	21	23
'182	any $\overline{Gi}, \overline{Pi}$	C1–3	13	41			4.5	4.5	7	7
	any $\overline{Gi}, \overline{Pi}$	$\overline{G}$	13	41			5	7	7.5	10.5
	any $\overline{Pi}$	$\overline{P}$	11	35			4.5	6.5	6.5	10
	C0	C1–3	17	50			6.5	7	10	10.5
'280	any input	EVEN	18	53	- / 6	- / 10	33	29	50	45
	any input	ODD	19	56	- / 6	- / 10	23	31	35	50
'283	C0	any Si	22	66			16	15	24	24
	any Ai, Bi	any Si	21	61			15	15	24	24
	C0	C4	19	58			11	11	17	22
	any Ai, Bi	C4	20	60			11	12	17	17
'381	CIN	any Fi					18	14	27	21
	any Ai, Bi	$\overline{G}$					20	21	30	33
	any Ai, Bi	$\overline{P}$					21	33	23	33
	any Ai, Bi	any Fi					20	15	30	23
	any select	any Fi					35	34	53	51
	any select	$\overline{G}, \overline{P}$					31	32	47	48
'682	any Pi	$\overline{PEQQ}$	26	69	- / 7	- / 11	13	15	25	25
	any Qi	$\overline{PEQQ}$	26	69	- / 7	- / 11	14	15	25	25
	any Pi	$\overline{PGTQ}$	26	69	- / 9	- / 14	20	15	30	30
	any Qi	$\overline{PGTQ}$	26	69	- / 9	- / 14	21	19	30	30

У схем малой степени интеграции задержка распространения от любого входа до выхода одна и та же. Заметьте, что в случае ТТЛ-схем задержки, как правило, различны при изменении сигнала от низкого уровня до высокого и от высокого уровня до низкого (t_{pLH} и t_{pHL}), а у КМОП-схем этого обычно нет. КМОП-схемы имеют более симметричные выходные характеристики, поэтому можно не обращать внимания на небольшое различие между этими двумя случаями.

Библиотека БГУИР

ОЦЕНКА МИНИМАЛЬНЫХ ЗАДЕРЖЕК

Если минимальная задержка ИС не указана, то предусмотрительный разработчик принимает ее равной нулю.

Некоторые схемы не будут работать, если задержка распространения фактически стремится к нулю, но затраты на изменения в схеме, обеспечивающие ее работоспособность при нулевой задержке, могут быть непомерно велики, тем более что такой случай, как предполагается, никогда не произойдет. Чтобы получить изделие, которое при «разумных» условиях всегда будет работать, разработчики часто принимают величину минимальной задержки ИС равной четверти или трети заявленного *типичного* значения задержки.

Задержка между моментом изменения входного сигнала и моментом изменения сигнала на выходе зависит от внутреннего пути сигнала, и в больших схемах этот путь может быть разным для различных входных комбинаций. Например, 2-входовой элемент ИСКЛЮЧАЮЩЕЕ ИЛИ в микросхеме 74LS86 состоит, как показано на рис. 5.71, из четырех вентилей И-НЕ и имеет два пути различной длины от любого входа до выхода. Если сигнал на одном из входов имеет низкий уровень, а на другом изменяется, то изменение проходит через два вентиля И-НЕ и мы получаем первый набор задержек, приведенный в табл. 5.2. Если сигнал на одном из входов имеет высокий уровень, а на другом изменяется, то изменение проходит внутри схемы через *три* вентиля И-НЕ и мы получаем второй набор задержек. Подобное поведение демонстрируют также микросхемы 74LS138 и 74LS139 (см. табл. 5.3). Однако у соответствующих КМОП-схем таких различий нет; точнее, различия достаточно малы и ими можно пренебречь.

5.2.4. Временно́й анализ

Для точного анализа временных соотношений в устройстве со многими МИС и СИС разработчику, вероятно, придется изучать ее поведение до мельчайших подробностей. Когда, например, инвертирующие ТТЛ-схемы (И-НЕ, ИЛИ-НЕ и т.д.) включаются последовательно, переход сигнала с низкого уровня на высокий на выходе одного из вентилей вызовет изменение сигнала от высокого уровня до низкого на выходе следующего вентиля, так что средняя задержка оказывается между величинами t_{pLH} и t_{pHL}. С другой стороны, при последовательном включении неинвертирующих вентилей (И, ИЛИ и т.д.) переключение вызывает изменение сигналов на всех выходах в одном и том же направлении, так что различие между значениями t_{pLH} и t_{pHL} увеличивается. Читателю предоставляется возможность провести подобного рода анализ в упражнениях 5.8–5.13.

Анализ оказывается более сложным, если задержка определяется устройствами средней степени интеграции, или в том случае, когда для сигнала имеется много путей от данного входа до данного выхода. Таким образом, в больших схемах, анализ задержки прохождения сигнала по всем возможным путям при переходе во всех направлениях может быть очень сложным.

Чтобы иметь возможность упростить анализ в «наихудшем случае», разработчики часто используют единственный параметр – *задержку в наихудшем случае*

Библиотека БГУИР

(*worst-case delay*), которая равна наибольшему из значений t_{pLH} и t_{pHL}. Тогда задержка в схеме для наихудшего случая вычисляется как сумма задержек, вносимых отдельными компонентами в наихудшем случае, независимо от направления переключения и других условий работы схемы. Это может дать чрезмерно завышенную оценку полной задержки, вносимой схемой, но сокращает время разработки и гарантирует работоспособность изделия.

5.2.5. Программные средства временнóго анализа

Еще проще провести временнóй анализ с помощью современных программных средств, входящих в состав автоматизированных систем логического проектирования. Встроенные библиотеки таких систем обычно содержат не только условные обозначения и функциональные модели различных логических элементов, но также модели их поведения во времени. В режиме моделирования можно задать последовательность входных сигналов и наблюдать, как и когда их действие проявляется в выходных сигналах. Обычно имеется возможность варьировать значения задержек, выбирая минимальные, типичные или максимальные значения или некоторую их комбинацию.

Даже используя моделирование, вы не гарантированы от ловушек. Обычно разработчик задает входные последовательности сигналов, для которых моделирующая программа должна сформировать выходные сигналы. Таким образом, необходимо иметь хорошее чутье при выборе тестирующих сигналов и условий моделирования, чтобы воспроизвести наихудший случай и наблюдать соответствующие задержки.

Некоторые программы временнóго анализа позволяют автоматически находить задержки по всем возможным путям сигнала в схеме и распечатывать упорядоченный список их значений, начиная с наибольшего. Однако эти результаты могут быть слишком пессимистическими, поскольку некоторые пути при нормальной работе схемы фактически не реализуются, и для соответствующей интерпретации результатов разработчику приходится все же напрягать свои умственные способности.

5.3. Комбинационные программируемые логические устройства

5.3.1. Программируемые логические матрицы

Исторически первыми программируемыми логическими устройствами (ПЛУ) были *программируемые логические матрицы* (*ПЛМ; programmable logic arrays, PLA*). ПЛМ представляют собой комбинационное двухуровневое устройство И–ИЛИ, которое можно запрограммировать для реализации любого логического выражения вида «сумма произведений» с учетом ограничений, накладываемых устройством. Такими ограничениями являются:

- число входов (*inputs*) *n*,
- число выходов (*outputs*) *m* и
- число термов-произведений (*product terms*) *p*.

Библиотека БГУИР

О таком устройстве можно говорить как о «ПЛМ размера $n \times m$ с p термами-произведениями». В общем случае p гораздо меньше числа минтермов с n переменными (2^n). Таким образом, ПЛМ не может реализовать произвольную логическую функцию с n переменными и m выходными значениями; их возможности ограничены функциями, которые могут быть представлены в виде суммы произведений с p или меньшим числом термов-произведений.

ПЛМ размера $n \times m$ с p термами-произведениями содержит p $2n$-входовых вентилей И и m p-входовых вентилей ИЛИ. На рис. 5.21 приведена небольшая ПЛМ с четырьмя входами, шестью вентилями И, тремя вентилями ИЛИ и тремя выходами. Сигналы поступают на входы буферов, на выходах каждого из которых появляются прямой и инверсный сигналы, используемые внутри матрицы. Возможные соединения в матрице обозначены символом ×; программирование устройства состоит в сохранении только тех соединений, которые необходимы. Выбираемые соединения выполнены в виде *плавких перемычек* (*fuses*), которые фактическими являются пережигаемыми соединениями или энергонезависимыми ячейками памяти в зависимости от технологии; мы рассмотрим это в разделах 5.3.4 и 5.3.5. Таким образом, сигналы на входах каждого вентиля И могут представлять собой любое подмножество первичных входных сигналов и их инверсий. Точно так же сигналы на входах каждого вентиля ИЛИ могут быть любым подмножеством выходных сигналов схем И.

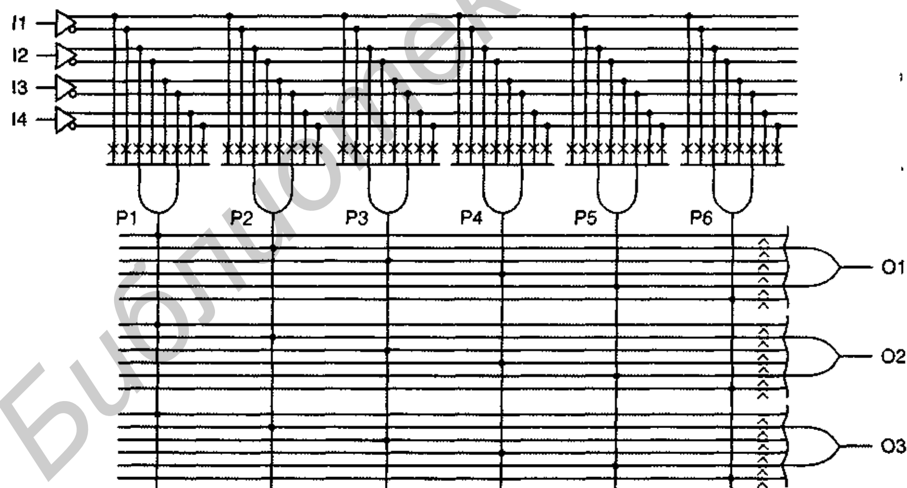


Рис. 5.21. ПЛМ размера 4×3 с шестью термами-произведениями

Возможен более компактный способ представления ПЛМ, указаннный на рис. 5.22. Такое изображение условных обозначений вентилей точнее соответствует их фактическому размещению внутри кристалла ПЛМ (см., например, рис. 5.28).

На рис. 5.22 изображена схема ПЛМ, способная реализовать любые три 4-входовые комбинационные логические функции, которые можно записать в виде сумм произведений, состоящих в общей сложности из шести или меньшего числа различных термов-произведений, например:

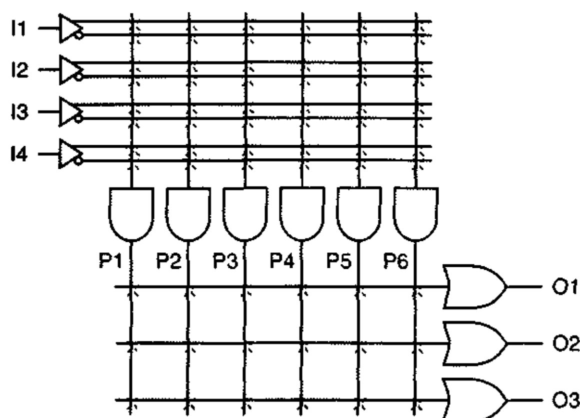


Рис. 5.22. Компактное представление ПЛМ размера 4×3 с шестью термами-произведениями

Всего в этих соотношениях имеется восемь термов-произведений, но первые два слагаемых в выражении для О3 те же самые, что и первые слагаемые в выражениях для О1 и О2. Этим логическим выражениям соответствует конфигурация запрограммированных соединений, приведенная на рис. 5.23.

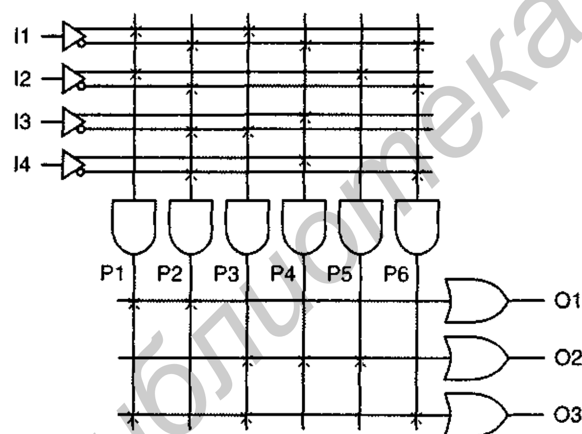


Рис. 5.23. Схема ПЛМ размера 4×3, запрограммированная для реализации трех логических выражений

$$O1 = I1 \cdot I2 + I1' \cdot I2' \cdot I3' \cdot I4'$$
$$O2 = I1 \cdot I3' + I1' \cdot I3 \cdot I4 + I2$$
$$O3 = I1 \cdot I2 + I1 \cdot I3' + I1' \cdot I2' \cdot I4'.$$

Иногда ПЛМ должна быть запрограммирована так, чтобы на выходе постоянно были 1 или 0. Как показано на рис. 5.24, сделать это не сложно. Терм-произведение P1 всегда равен 1, потому что на входы соответствующего вентиля И не подается ни один из входных сигналов и поэтому на его выходе всегда присутствует высокий уровень; этим термом-константой, равным 1, определяется сигнал на выходе О1. На входы вентиля ИЛИ, формирующего сигнал на выходе О2, не подан ни один терм-произведение, поэтому сигнал на этом выходе всегда равен 0. Возмо-

Библиотека БГУИР

жен другой метод получения на выходе константы, равной 0, как показано на рисунке в отношении выхода O3. На входы вентиля, формирующего терм-произведение P2 поданы все входные переменные и их инверсии; поэтому P2, а вместе с ним и O3, всегда равны 0 ($X \cdot X' = 0$).

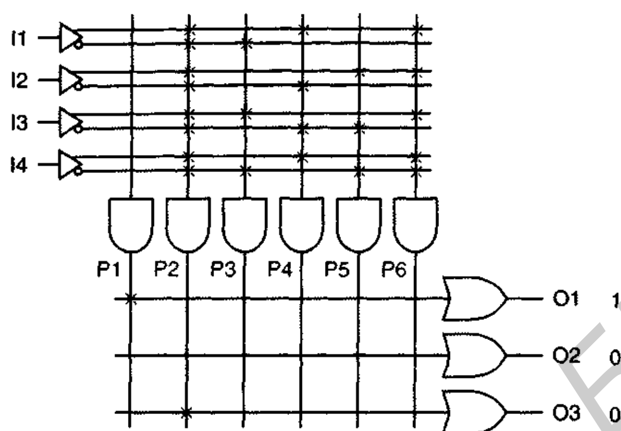


Рис. 5.24. ПЛМ размера 4×3, запрограммированная для получения на выходах констант, равных 0 и 1

В нашем примере ПЛМ имеет слишком мало входов, выходов и вентилей И (термов-произведений) чтобы быть очень полезной. В ПЛМ с *n* входами, в принципе, можно было бы получить до 2^n термов-произведений для реализации всех возможных минтермов с *n* переменными. Фактическое число термов-произведений в типичных коммерческих ПЛМ много меньше, примерно от 4 до 16 на выход, независимо от значения *n*.

Типичным примером ПЛМ является схема 82S100 фирмы Signetics, которая выпускалась в середине 70-х годов. У нее было 16 входов, 48 вентилей И и 8 выходов. Таким образом, в схеме было 2×16×48 = 1536 плавких перемычек в матрице вентилей И и 8×48 = 384 плавкие перемычки в матрице вентилей ИЛИ. Позже эти микросхемы были вытеснены устройствами типа PAL, CPLD и FPGA, однако внутри больших специализированных ИС для реализации сложной комбинационной логики часто создаются структуры типа ПЛМ.

МАЛОВЕРОЯТНЫЙ СБОЙ

Теоретически, когда *все* входные сигналы в схеме на рис. 5.24 изменяются *одновременно,* на выходе вентиля, формирующего терм-произведение P2, может появиться кратковременный паразитный импульс, то есть сигнал вида 0-1-0. В типичных приложениях эта ситуация очень маловероятна, и она невозможна в том случае, когда один из входов оказывается неиспользуемым и на него подан постоянный логический сигнал.

Библиотека БГУИР

5.3.2. Программируемые матричные логические устройства

Частным случаем ПЛМ, и на сегодня самым распространенным типом ПЛУ, являются *программируемые матричные логические устройства PAL* (*programmable array logic devices*). В отличие от ПЛМ, в которой программируемыми являются и матрицы вентилей И, и матрицы вентилей ИЛИ, схема PAL имеет *фиксированные* соединения в матрице вентилей ИЛИ.

В первых схемах PAL, появившихся в конце 70-х годов, применялась ТТЛ-совместимая *биполярная технология*. Основными новшествами в первых схемах PAL, помимо броского имени (англ. pal – товарищ, приятель), были использование фиксированной матрицы вентилей ИЛИ и двунаправленные выводы входов/выходов.

Хорошей иллюстрцией этих идей служит схема *PAL16L8*, показанная на рис. 5.25 и 5.26; эта схема – одна из наиболее широко применяемых в настоящее время комбинационных ПЛУ. В этой схеме программируемая матрица вентилей И имеет 64 строки и 32 столбца, пронумерованные на рисунке для целей программирования мелким шрифтом, и 64×32 = 2048 плавких перемычек. Каждый из 64 вентилей И в этой матрице имеет 32 входа, предназначенных для подключения 16 входных сигналов и их инверсий; отсюда число 16 в обозначении схемы «PAL16L8».

С каждым выходным контактом микросхемы PAL16L8 связаны восемь вентилей И. Семь из них формируют входные сигналы для фиксированных 7-входовых вентилей ИЛИ. Выход восьмого вентиля И, который мы называем *вентилем разрешения выходного сигнала* (*output-enable gate*), соединен с входом управления третьим состоянием выходного буфера; буфер открыт и пропускает сигнал на выход только тогда, когда на выходе вентиля разрешения выходного сигнала присутствует 1. Таким образом, микросхема PAL16L8 способна реализовать только такие логические функции, которые можно представить в виде суммы, состоящей из семи или меньшего числа термов-произведений. Каждый терм-произведение может быть функцией всех 16 входных сигналов или части из них, но возможны только семь таких термов-произведений.

ДРУЗЬЯ И ВРАГИ

PAL является зарегистрированной торговой маркой фирмы Advanced Micro Devices, Inc. Подобно другим торговым маркам, она должна использоваться только как прилагательное. Использовать аббревиатуру PAL в качестве существительного или без указания, что это торговая марка, вы можете только под свою ответственность (как я узнал об этом из письма юристов фирмы AMD в феврале 1989 года).

Имея в виду это предупреждение, я предлагаю пользоваться описательным именем, которое несет больше информации о внутренней структуре устройства: *элемент с фиксированной матрицей ИЛИ* (*fixed-OR element, FOE*; англ. foe – враг).

Библиотека БГУИР

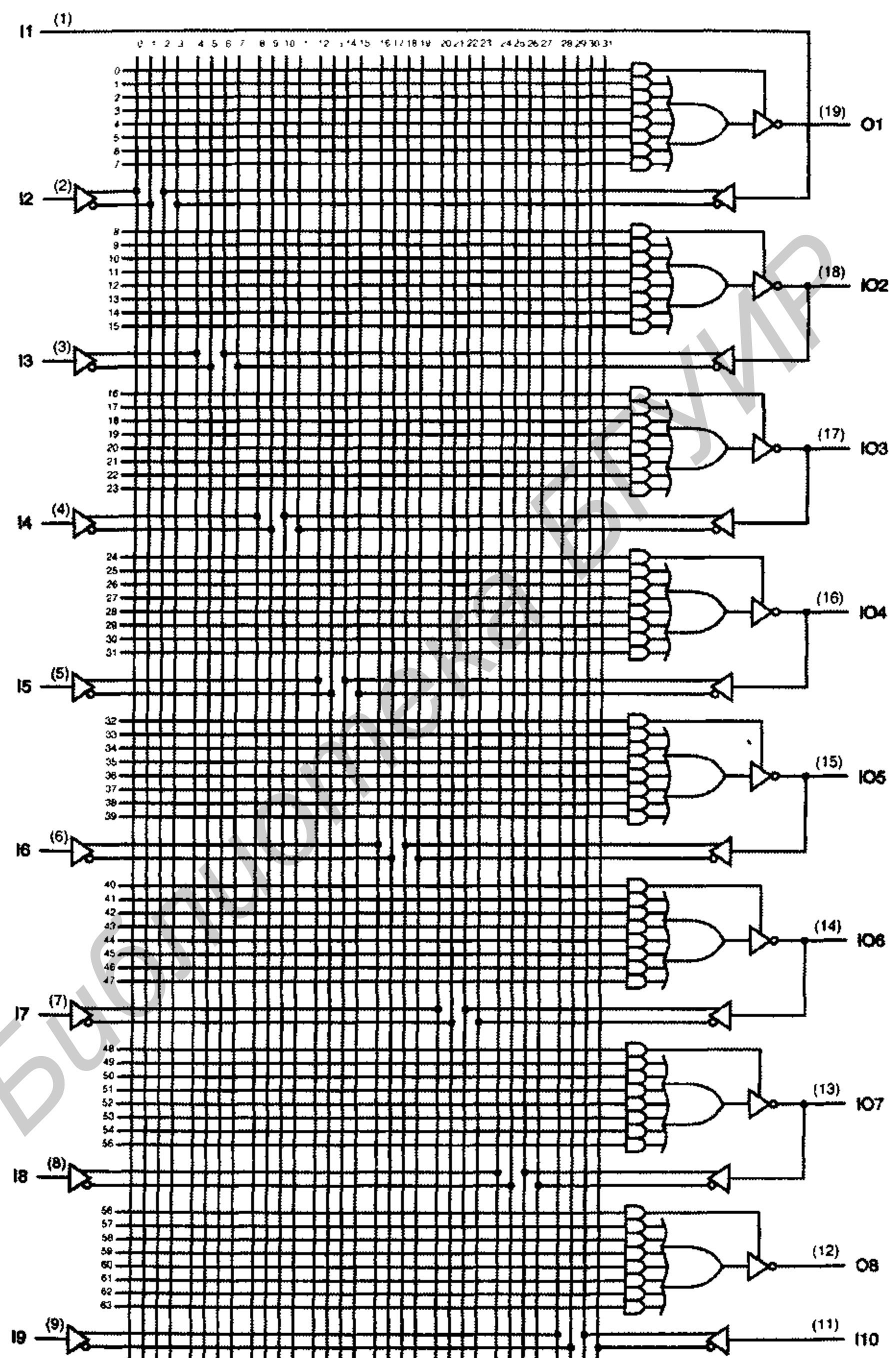


Рис. 5.25. Принципиальная схема ИС PAL16L8

Библиотека БГУИР

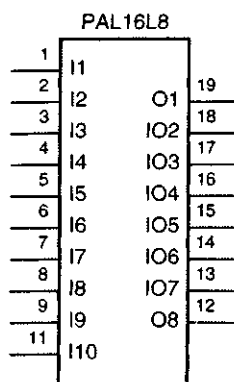


Рис. 5.26. Традиционное условное обозначение микросхемы PAL16L8

Хотя у микросхемы PAL16L8 может быть до 16 входов и до 8 выходов, она размещена в корпусе DIP всего лишь с 20 выводами, включая два вывода для подключения напряжения питания и земли (угловые выводы 10 и 20). Это достигается благодаря наличию шести двунаправленных выводов (13–18), которые можно использовать как входы или выходы, либо как то и другое. Таким образом, различия между PAL16L8 и структурой ПЛМ сводятся к следующему:

- PAL16L8 имеет фиксированную матрицу вентилей ИЛИ с семью вентилями И, постоянно соединенными с каждым из вентилей ИЛИ. Выходы вентилей И нельзя соединить с входами нескольких вентилей ИЛИ; если терм-произведение необходим двум вентилям ИЛИ, то его необходимо сформировать дважды.
- Каждый выход микросхемы PAL16L8 может находиться в третьем состоянии и управляется индивидуальным сигналом разрешения выхода с тремя состояниями, предназначенным для этого вентилем И (вентилем разрешения выходного сигнала). Следовательно, состояние выходов можно запрограммировать так, чтобы они всегда были активны, всегда были заблокированы или управлялись бы комбинацией входных сигналов, включенных в соответствующий терм-произведение.

ДОСТАТОЧНО ЛИ СЕМИ ТЕРМОВ-ПРОИЗВЕДЕНИЙ?

Для двухуровневой структуры И–ИЛИ наихудшей логической функцией является ИСКЛЮЧАЮЩЕЕ ИЛИ (контроль четности) с *n* переменными; в этом случае требуется $2^n - 1$ термов-произведений. Однако часто бывает так, что менее своенравную функцию можно реализовать с помощью микросхемы PAL16L8 даже в том случае, когда у нее число термов-произведений больше 7. Для этого ее нужно преобразовать к 4-уровневому виду И–ИЛИ–И–ИЛИ, и тогда данная функция может быть реализована за два прохода сквозь матрицу И–ИЛИ. К сожалению, в результате использования выходных сигналов ПЛУ в качестве термов, образующихся на первом проходе, при этом удваивается задержка, так как входной сигнал должен дважды пройти через ПЛУ, прежде чем он достигнет выхода.

Библиотека БГУИР

КОМБИНАЦИОННЫЙ, НЕ КОМБИНАТОРНЫЙ!

Шагом *назад* при популяризации микросхем PAL было введение слова «комбинаторный» для обозначения комбинационных схем. *Комбинационные* схемы не имеют памяти: в любой момент времени их выходные сигналы определяются текущей *комбинацией* входных сигналов. У образованного специалиста по компьютерам слово «комбинаторный» ассоциируется с биномиальными коэффициентами, сложностью решения задач и гением информатики Дональдом Кнутом.

- Между выходом каждого вентиля ИЛИ и внешним выводом микросхемы PAL16L8 включен инвертор.
- Шесть из выходных выводов ИС PAL16L8, названных *I/O-выводами* (*I/O pins*), можно использовать также в качестве входов. Благодаря этому возникает много возможностей использования каждого из I/O-выводов в зависимости от того, как запрограммировано устройство:
  - Если вентиль, управляющий I/O-выводом, вырабатывает постоянный сигнал, равный 0, то выходной буфер всегда находится в третьем состоянии и вывод используется строго в качестве входа.
  - Если входной сигнал на I/O-выводе не используется никакими схемами в матрице вентилей И, то вывод можно использовать строго как выход. В зависимости от того, как запрограммирован вентиль разрешения выходного сигнала, выходной буфер может быть активным всегда или только при некоторых входных условиях.
  - Если вентиль, управляющий I/O-выводом, вырабатывает постоянный сигнал, равный 1, то выходной буфер всегда активен, но данный вывод можно все же использовать также и как вход. Таким образом, выходами можно воспользоваться для образования на первом проходе «вспомогательных термов» в случае логических функций, которые не могут быть выполнены за один проход из-за ограничения по числу термов-произведений, доступных на одном выходе. В разделе 5.4.6 будет приведен соответствующий пример.
  - В другом случае, когда для данного I/O-вывода выходной сигнал постоянно разрешен, его можно использовать в качестве входного сигнала вентилей И, результатом действия которых определяется тот же самый выходной сигнал. Другими словами, в микросхеме PAL16L8 можно создать последовательностную схему с обратной связью. Этот случай мы рассмотрим в разделе 8.2.6.

Микросхема *PAL20L8* является другим комбинационным ПЛУ, подобным PAL16L8, за исключением того, что корпус этой микросхемы имеет на четыре вывода больше (эти выводы работают только на вход) и каждый из ее вентилей И снабжен еще восьмью входами, позволяющими использовать дополнительные входные сигналы. Выходы у этой микросхемы организованы так же, как у схемы PAL16L8.

Библиотека БГУИР

5.3.3 Универсальные матричные логические устройства

В параграфе 8.3 будут введены последовательностные ПЛУ – программируемые логические устройства, в которых некоторые или все выходы вентилей ИЛИ снабжены триггерами. Эти устройства можно запрограммировать для реализации ряда полезных функций, выполняемых последовательностными схемами.

Один тип последовательностных ПЛУ, впервые представленных фирмой Lattice Semiconductor и особенно популярных, назван *универсальными матричными логическими устройствами GAL* (*generic array logic device*). Единственное устройство GAL типа *GAL16V8* можно сконфигурировать (путем программирования и создания соответствующих соединений) так, чтобы имитировались схема вида И–ИЛИ, триггеры и выходные цепи, встречающиеся во всем многообразии комбинационных и последовательностных устройств PAL, включая уже рассмотренную нами микросхему PAL16L8. Более того, конфигурация GAL может быть электрически стерта и перепрограммирована.

На рис. 5.27 показана принципиальная схема ИС GAL16V8, сконфигурированной как исключительно комбинационное устройство, подобное PAL16L8. Эта конфигурация достигается программированием двух не показанных на рисунке соединений, «управляющих архитектурой». В изображенной конфигурации устройство носит название *GAL16V8C*.

Самое важное, что следует отметить при сравнении ИС GAL16V8C с ИС PAL16L8, состоит в том, что между каждым выходом вентиля ИЛИ и выходным буфером с тремя состояниями включен вентиль ИСКЛЮЧАЮЩЕЕ ИЛИ. Один из входов вентиля ИСКЛЮЧАЮЩЕЕ ИЛИ может быть «подтянут» к уровню логической 1, но плавкой перемычкой соединен с землей (0 В). Если эта перемычка сохранена, то вентиль ИСКЛЮЧАЮЩЕЕ ИЛИ просто пропускает без изменений сигнал, поступающий с выхода схемы ИЛИ; но если перемычку пережечь, то вентиль ИСКЛЮЧАЮЩЕЕ ИЛИ инвертирует сигнал, поступающий с выхода схемы ИЛИ. Говорят, что эта плавкая перемычка управляет *полярностью выходного сигнала* (*output polarity*) на соответствующем выходном контакте.

БЫСТРОДЕЙСТВИЕ КОМБИНАЦИОННЫХ ПЛУ

Быстродействие комбинационных ПЛУ обычно выражается одним числом t_{PD}, характеризующим задержку прохождения сигнала от любого входа до любого выхода при произвольном направлении переключения. Выпускаются ПЛУ с различным быстродействием; широко распространены микросхемы с задержкой 10 нс. В 1998 году самыми быстрыми были комбинационные ПЛУ на биполярных транзисторах PAL16L8 с задержкой 5 нс и 3.3-вольтовые ПЛУ на КМОП-транзисторах GAL22LV10 с задержкой 3.5 нс.

Библиотека БГУИР

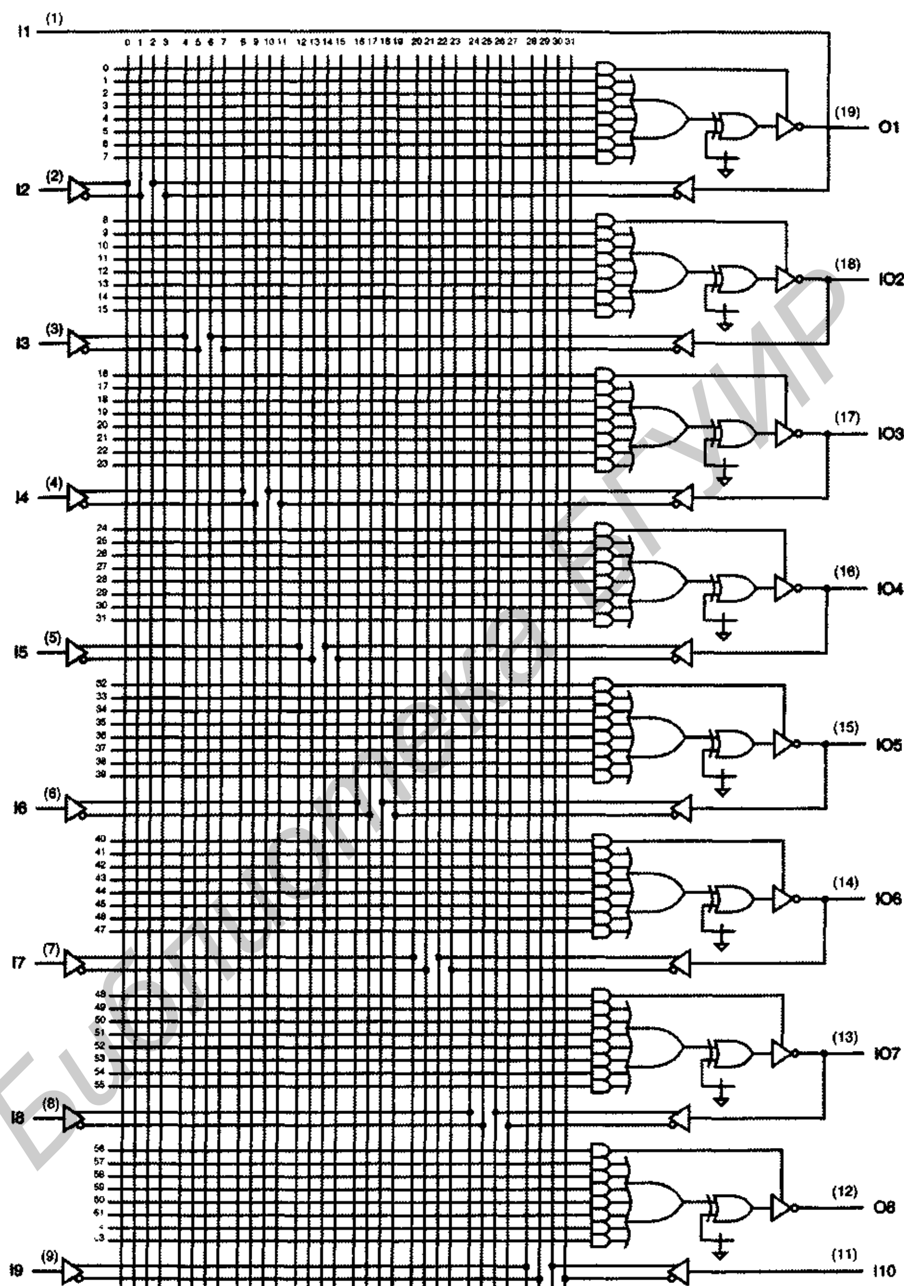


Рис. 5.27. Принципиальная схема ИС GAL16V8C

Возможность управления полярностью выходного сигнала является очень важным свойством современных ПЛУ, в том числе и микросхемы GAL16V8. Мы видели в разделе 4.6.2, что при минимизации заданной логической функции компиля-

Библиотека БГУИР

тор языка ABEL находит минимальные выражения вида «сумма произведений» как для самой функции, так и для ее инверсии. В случае, когда инверсия имеет меньшее число термов-произведений, этим можно воспользоваться, если перемычку, управляющую полярностью соответствующего выходного сигнала в схеме GAL16V8, разрушить. Транслятор автоматически выбирает лучший вариант и соответствующим образом решает, какие плавкие перемычки следует пережечь, если эта операция не отменена.

Несколько фирм выпускают микросхему *PALCE16V8*, которая является эквивалентом GAL16V8. Имеются также микросхемы *GAL20V8* или *PALCE20V8* в корпусе с 24 выводами, которые можно сконфигурировать так, чтобы они имитировали структуру схемы PAL20L8 или какой-либо другой схемы из числа последовательностных ПЛУ, рассматриваемых в разделе 8.3.2.

ОФИЦИАЛЬНОЕ ПРЕДУПРЕЖДЕНИЕ

GAL является торговой маркой фирмы Lattice Semiconductor, Hillsboro, OR 97124.

*5.3.4. Схемы биполярных ПЛУ

Для построения и физического программирования ПЛУ применяется несколько различных технологий. В первых коммерческих ПЛМ и устройствах PAL были использованы схемы на биполярных транзисторах. В качестве примера на рис. 5.28 показано, как можно построить приведенную в разделе 5.3.1 ПЛМ размера 4×3 на основе биполярной ТТЛ-подобной технологии. Каждое возможное соединение реализовано в виде последовательно включенных диода и металлического соединения, которое может присутствовать или отсутствовать. Если соединение имеется, то диод подключает соответствующий ему вход к диодной схеме И. Если соединение отсутствует, то соответствующий вход не оказывает никакого влияния на эту схему И.

Диодная схема И предназначена для того, чтобы на вертикальной «линии И» высокий уровень возникал только в том случае, когда на всех до единой горизонтальных «входных линиях», подключенных через диод к данной линии И, имеется высокий уровень. Если какая-то из входных линий имеет низкий уровень, то это приводит к низкому уровню на всех линиях И, с которыми соединена данная входная линия. Эта первая совокупность схемных элементов, реализующих функции И, которая называется *матрицей И* (*AND plane*).

За каждой линией И следует инвертирующий буфер, так что в целом реализуется функция И-НЕ. Выходы схем И-НЕ первого уровня объединяются другим набором программируемых диодных схем И, за которыми снова следуют инверторы. В результате мы имеем двухуровневую структуру И-НЕ–И-НЕ, которая является функциональным эквивалентом описанной ранее ПЛМ со структурой И–ИЛИ. Совокупность схемных элементов, реализующих функцию ИЛИ (или образующих второй уровень И-НЕ – в зависимости от того, как на это посмотреть), называется *матрицей ИЛИ* (*OR plane*). При изготовлении кристалла биполярного ПЛУ предус-

Библиотека БГУИР

матривается наличие в нем всех диодов, включенных последовательно с *крошечными пережигаемыми перемычками* (*fusible link*) (на рис. 5.28 они изображены небольшими волнистыми линиями). Применяя специальные трафареты, можно выбрать отдельные перемычки, приложить к ним высокое напряжение (10–30 В), и таким образом выбранную перемычку испарить.

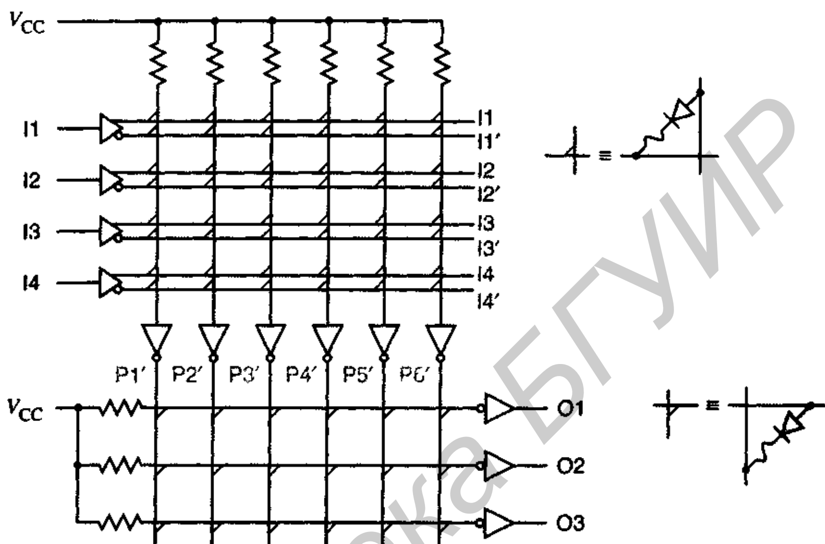


Рис. 5.28. ПЛМ размера 4×3 на основе ТТЛ-подобных схем с открытым коллектором и диодной логики

Первые биполярные ПЛУ были не очень надежны. Иногда зафиксированная конфигурация изменялась из-за не полностью испаренной перемычки, которая могла «снова вырасти», а иногда причиной периодических отказов были плавающие капельки внутри корпуса ИС. Однако в дальнейшем эти проблемы в значительной степени были устранены, и надежная технология с плавкими перемычками применяется в биполярных ПЛУ и сегодня.

*5.3.5 Схемы ПЛУ на основе КМОП-логики

Хотя биполярные ПЛУ все еще остаются доступными, они в значительной степени оказались вытесненными ПЛУ на основе КМОП-логики, которые обладают рядом преимуществ, в том числе меньшей потребляемой мощностью и возможностью перепрограммирования. На рис. 5.29 показан КМОП-вариант ПЛМ размера 4×3 из раздела 5.3.1.

В каждом пересечении входной линии с «линией слова» вместо диода помещен *n*-канальный транзистор с программируемым подключением. Если сигнал на входе имеет низкий уровень, то транзистор «закрыт», а если входной сигнал имеет высокий уровень, то транзистор «открыт», что приводит к появлению на линии И низкого уровня. В результате получаем схему И с инверсией на входе (то есть схему ИЛИ-НЕ). По своей структуре и выполняемой функции эта часть схемы

Библиотека БГУИР

подобна обычному КМОП-вентилю ИЛИ-НЕ с *k* входами, за исключением того, что обычное последовательное соединение *k* транзисторов с каналом *p*-типа, обеспечивавших высокий уровень на выходе, заменено резистором (на самом деле высокий уровень на линии И в ПЛМ обеспечивается одним постоянно открытым *p*-канальным транзистором).

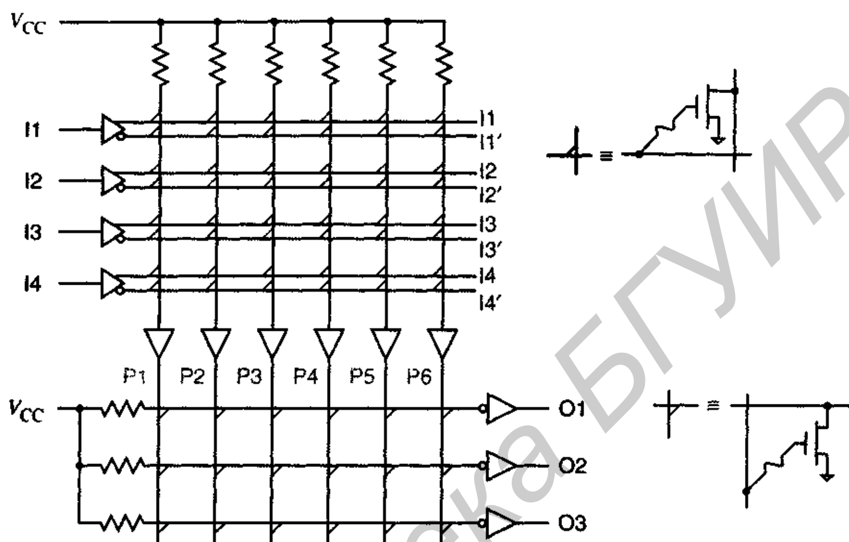


Рис. 5.29. ПЛМ размера 4×3 на основе КМОП-логики

Из рис. 5.29 следует, что – в отличие от схемы на рис. 5.28 – применение вентиля И с *инверсией* на входе нейтрализуется использованием входных линий с инверсными значениями значениями сигналов для каждого входа. Обратите внимание также на то, что соединение между матрицей И и матрицей ИЛИ является неинвертирующим, так что матрица И реализует истинную функцию И.

Выходы линий И первого уровня объединяются в матрице ИЛИ другим набором программируемых соединений, реализующих функцию ИЛИ-НЕ. На выходе каждой линии ИЛИ-НЕ включен инвертор, так что в результате получаем истинную функцию ИЛИ, а в целом ПЛМ реализует функцию И–ИЛИ, что и требовалось.

В ПЛУ, изготовляемых по КМОП-технологии, программируемые перемычки, показанные на рис. 5.29, первоначально бывают не расплавлены. В устройствах, не программируемых в процессе работы, типа заказных СБИС наличие или отсутствие отдельных перемычек определяется маской металлизации при изготовлении устройства. Однако в КМОП-схемах типа EPLD, рассматриваемых ниже, чаще всего, безусловно, применяется другая технология программирования.

В *стираемом программируемом логическом устройстве (erasable programmable logic device, EPLD)* можно запрограммировать любую желаемую конфигурацию связей, но можно также вернуть устройство в его первоначальное состояние, «стирая» эти связи электронным путем или облучая ультрафиолетовым светом. Нет, стирание не вызывает внезапного появления или исчезновения

связей! Правильнее сказать, что в устройствах EPLD применяется другая технология, называемая «МОП-структура с плавающим затвором».

Как показано на рис. 5.30, в схемах EPLD используются *МОП-транзисторы с плавающим затвором* (*floating-gate MOS transistor*). Такой транзистор имеет два затвора. «Плавающий» затвор ни к чему не подключен и окружен диэлектриком с очень малой проводимостью. В исходном состоянии в плавающем затворе нет никакого заряда, и он не влияет на работу схемы. В этом состоянии все транзисторы фактически «открыты»; то есть во всех точках пересечения линий в матрицах И и ИЛИ имеется логическая связь.

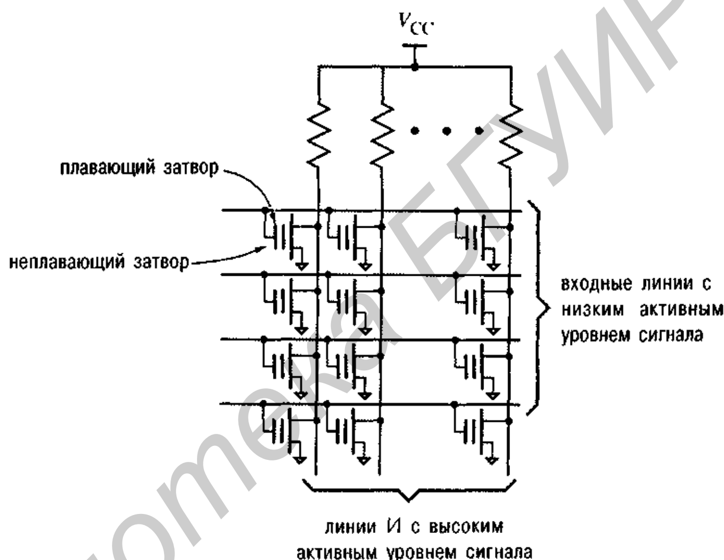


Рис. 5.30. Матрица И в перепрограммируемом логическом устройстве типа EPLD на основе МОП-транзисторов с плавающим затвором

Программирование схем EPLD осуществляется с помощью программатора: к неплавающему затвору в каждом месте, где связь не требуется, прикладывается высокое напряжение. Это вызывает временный пробой в диэлектрике, что позволяет отрицательному заряду накопиться в плавающем затворе. Когда высокое напряжение снимается, в плавающем затворе остается отрицательный заряд. При выполнении последующих операций отрицательный заряд препятствует «открыванию» транзистора, когда на неплавающий затвор поступает сигнал высокого уровня; транзистор оказывается фактически отключенным от схемы.

Производители схем EPLD утверждают, что надлежащим образом запрограммированная ячейка сохранит 70% своего заряда, по крайней мере, в течение 10 лет, даже в том случае, если микросхема будет хранится при температуре 125°С. Поэтому в большинстве приложений можно считать, что результат программирования сохраняется постоянно. Однако запрограммированную в схеме EPLD конфигурацию можно стереть.

Хотя некоторые первые микросхемы EPLD помещались в корпус с прозрачной крышкой и для стирания использовали свет, наиболее популярные современные

устройства являются *электрически стираемыми программируемыми логическими устройствами* (*electrically erasable PLD*). Плавающие затворы в электрически стираемом ПЛУ окружены очень тонким слоем диэлектрика, и заряд можно удалить из них путем подачи на неплавающий затвор напряжения, полярность которого противоположна полярности, необходимой для накопления заряда в плавающем затворе. Таким образом, тот же самый программатор, который обычно используется для программирования ПЛУ, можно применить также для стирания схемы EPLD перед программированием ее заново.

В бóльших по размерам *«сложных» ПЛУ ("complex" PLD, CPLD)* также применяется метод программирования с плавающим затвором. Даже в еще больших устройствах, часто называемых *перепрограммируемыми вентильными матрицами* (*field-programmable gate arrays, FPGA*), для управления каждым соединением используют ячейки оперативного запоминающего устройства (ОЗУ). Ячейки ОЗУ являются энергозависимыми: они не сохраняют свое состояние при выключении питания. Поэтому при подаче на FPGA напряжения питания в его ОЗУ необходимо записать конфигурацию, хранящуюся отдельно во внешней энергонезависимой памяти. Такой памятью обычно служит программируемое постоянное запоминающее устройство (ППЗУ), подключенное непосредственно к схеме FPGA, или память микропроцессорной подсистемы, загружающей схему FPGA при инициализации системы в целом.

*5.3.6. Программирование и тестирование микросхем

Для разрушения плавких перемычек, накопления заряда в плавающих затворах транзисторов или для осуществления каких-то других действий, необходимых для программирования ПЛУ, применяется специальное оборудование. Эта аппаратура, имеющаяся в настоящее время почти во всех лабораториях, разрабатывающих и создающих цифровые устройства, называется *программаторами ПЛУ* (*PLD programmer*) или *программаторами ПЗУ* (*PROM programmer*). [Программаторы можно применять как для записи в программируемые постоянные запоминающие устройства (ППЗУ), так и для программирования ПЛУ.] Типичный программатор ПЛУ имеет разъем или разъемы, с помощью которых подключаются программируемые устройства, и канал «загрузки» желаемой конфигурации соединений в программатор. Обычно загрузка осуществляется из подключенного к программатору персонального компьютера.

Как правило, при программировании микросхема ПЛУ переводится программатором в специальный режим работы. Например, применительно к описанным в этой главе ПЛУ программирование соединений осуществляется по отношению к восьми плавким перемычкам одновременно, и происходит это следующим образом:

1. Для перевода микросхемы в режим программирования на один из специально предназначенных для этого выводов подается высокое напряжение (порядка 14 В).
2. Путем подачи двоичного «адреса» на определенные входы микросхемы, выбирается группа из восьми плавких перемычек. (Например, в микросхеме 82S100 имеется 1920 перемычек, и поэтому ей требуется 8 входов, чтобы выбрать одну из 240 групп, по 8 перемычек в каждой.)

Библиотека БГУИР

ИЗМЕНЕНИЕ «ЖЕЛЕЗА» НА ЛЕТУ

В типичном случае нужная конфигурация соединений заносится в ОЗУ, входящее с состав FPGA, из ПЗУ, но существуют такие приложения, где конфигурация соединений фактически считывается с дискеты. Вы только что получили дискету с новой версией программы? Считайте, что вы только что получили также новый вариант аппаратуры!

Эта концепция приводит нас к захватывающей идее, уже использованной в некоторых приложениях, а именно – к созданию «перестраиваемых аппаратных средств», когда аппаратная часть перестраивается «на лету» с целью оптимизировать ее параметры применительно к решаемой в данный момент конкретной задаче.

3. На *выходы* микросхемы подается 8-разрядное число, задающее желаемый результат программирования для каждой из выбранных перемычек (выходы в режиме программирования используются как входы).
4. На некоторое время (порядка 100 микросекунд) увеличивается напряжение на другом специально предназначенном для этого выводе для программирования выбранных восьми плавких перемычек.
5. Напряжение на втором специальном выводе уменьшается (до 0 В), чтобы программатор мог выполнить считывание и проверить правильность программирования выбранных восьми плавких перемычек.
6. Шаги с 1 по 5 повторяются для каждой группы из восьми плавких перемычек.

Многие ПЛУ – в частности, самые большие схемы CPLD – обладают свойством *программируемости в системе* (*in-system programmability*). Это означает, что устройство может быть запрограммировано после того, как оно уже запаяно в систему. В этом случае конфигурация разрушаемых перемычек вводится последовательно с помощью четырех дополнительных сигналов и выводов, называемых *портом JTAG* (*JTAG port*; JTAG – Joint Test Automation Group), который определен стандартом IEEE 1149.1. Эти сигналы позволяют составить из различные устройств на данной печатной плате «цепочку последовательного опроса» («daisy chain») для выбора и программирования в процессе изготовления платы через единственный специальный разъем порта JTAG. При этом не требуется никакого специального высоковольтного источника питания; в каждом устройстве для получения высокого напряжения, необходимого при программировании, применяется внутренняя схема накачки заряда.

Как было отмечено выше, на 5-м шаге проверяется правильность программирования выбранных плавких перемычек. Если после первого программирования перемычек обнаружены ошибки, то операцию можно повторить; если ошибки обнаруживаются после нескольких попыток, то микросхема бракуется (часто с большим пристрастием и желанием нанести ей вред).

При проверке конфигурации запрограммированного устройства подтверждение того факта, что плавкие перемычки установлены должным образом, еще не

доказывает, что устройство будет выполнять логическую функцию, соответствующую установленной конфигурации перемычек. Это происходит потому, что устройство может иметь не связанные с программированием внутренние дефекты, такие как отсутствие соединений между плавкими перемычками и элементами решетки И–ИЛИ.

Единственный способ обнаружить все дефекты состоит в том, чтобы поставить устройство в нормальный режим работы, подать на входы набор нормальных логических сигналов и наблюдать сигналы на выходах. Соответствующие наборы входных и выходных сигналов, называемые тестовыми векторами, могут быть заданы разработчиком, как мы видели в разделе 4.6.7, или могут быть образованы автоматически в соответствии со специальной программой генерирования тестовых векторов. Независимо от того, как получены тестовые векторы, у большинства программаторов есть возможность подавать входные тестовые векторы на ПЛУ и сравнивать выходные сигналы с ожидаемыми.

Большинство ПЛУ имеют *защиту конфигурации соединений* (*security fuse*), которая, будучи установлена, блокирует возможность чтения конфигурации перемычек в устройстве. Производители могут запрограммировать схему так, чтобы никто не мог считать из ПЛУ конфигурацию перемычек с целью копирования устройства. Тем не менее, даже если установлена защита конфигурации соединений, тестовые векторы все же работают, так что проверка ПЛУ возможна.