Стандарт IEEE 1149.1-2001

(Пересмотр стандарта IEEE Std 1149.1-1990)

Стандартный порт тестового доступа IEEE и архитектура пограничного сканирования

Спонсор

Комитет по стандартам тестовых технологий

IEEE Computer Society

Утверждено 14 июня 2001 г.

Совет по стандартам IEEE-SA

Аннотация: Определены схемы, которые могут быть встроены в интегральную схему для облегчения тестирования, технического обслуживания и поддержки собранных печатных плат. Схема включает в себя стандартный интерфейс
, через который передаются инструкции и тестовые данные. Определен набор тестовых функций,
включая регистр граничного сканирования, такой, что компонент способен реагировать на минимальный набор
инструкций, предназначенных для облегчения тестирования собранных печатных плат. Кроме того, определен язык
, который позволяет строго описывать специфичные для компонента аспекты таких функций тестируемости
.

Ключевые слова: сканирование границ, архитектура сканирования границ, язык описания сканирования границ,
регистр сканирования границ, BSDL, печатные платы, схемотехника, интегральная схема, печатные платы,
TAP, тест, порт тестового доступа, VHDL, язык описания аппаратного обеспечения VHSIC

Институт инженеров электротехники и электроники, Inc.
Парк-авеню, 3, Нью-Йорк, Нью-Йорк 10016-5997, США

Печать: ISBN 0-7381-2944-5 Ш94949
PDF: ISBN 0-7381-2945-3 SS94949

Документы по стандартам IEEE разрабатываются в рамках обществ IEEE и Комитетов по координации стандартов Совета по стандартам Ассоциации стандартов IEEE (IEEE-SA). IEEE разрабатывает свои стандарты на основе согласованного процесса разработки, одобренного Американским национальным институтом стандартов, который объединяет добровольцев, представляющих различные точки зрения и интересы для создания конечного продукта. Добровольцы не обязательно являются членами Института и служат без какой-либо компенсации. Хотя IEEE управляет процессом и устанавливает правила для обеспечения справедливости в процессе разработки на основе консенсуса , IEEE не проводит независимую оценку, тестирование или проверку точности какой-либо информации, содержащейся в его стандартах.

Использование стандарта IEEE является полностью добровольным. Стандарт IEEE несет ответственности за любые травмы, имущества или других дам- возраст, любого характера, будь то специальные, косвенные, косвенные или компенсаторные, прямо или косвенно вытекающие из публикации, использования, или полагаться на этот, или любой другой стандарт IEEE документ.

IEEE не гарантирует и не представляет точность или содержание материалов, содержащихся в настоящем документе, и прямо отказывается от любых явных или подразумеваемых гарантий, включая любые подразумеваемые гарантии товарной пригодности или пригодности для определенной цели, или что использование материалов, содержащихся в настоящем документе, не содержит нарушений патентных прав. Документы по стандартам IEEE предоставляются “**КАК ЕСТЬ**”.

Существование стандарта IEEE не означает, что не существует других способов производить, тестировать, измерять, приобретать, продавать или предоставлять другие товары и услуги, относящиеся к сфере применения Стандарта IEEE. Кроме того, точка зрения, выраженная во время утверждения и выпуска стандарта, может быть изменена в результате развития современного уровня техники и комментариев, полученных от пользователей стандарта. Каждый стандарт IEEE пересматривается не реже одного раза в пять лет для пересмотра или подтверждения. Когда документу более пяти лет и он не был подтвержден, разумно сделать вывод, что его содержание, хотя и по-прежнему представляет определенную ценность, не полностью отражает современное состояние техники. Пользователям рекомендуется проверить, установлена ли у них последняя версия любого стандарта IEEE.

Публикуя и делая доступным этот документ, IEEE не предлагает и не оказывает профессиональных или иных услуг для какого-либо физического или юридического лица или от его имени. IEEE также не берет на себя обязательств по исполнению каких-либо обязательств какого-либо другого физического или юридического лица перед другим лицом. Любое лицо, использующее этот и любой другой документ стандартов IEEE, должно полагаться на рекомендации компетентного специалиста при определении разумной осторожности в любых данных обстоятельствах.

Толкования: Иногда могут возникать вопросы относительно значения отдельных частей стандартов, поскольку они относятся к конкретным применениям. Когда информация о необходимости интерпретаций будет доведена до сведения IEEE, Институт предпримет действия по подготовке соответствующих ответов. Поскольку стандарты IEEE представляют собой консенсус заинтересованных сторон, важно убедиться, что любая интерпретация также получила одобрение баланса интересов. По этой причине, IEEE и членов его СЕ- и казалось бы о стандартах координационных комитетов не в состоянии обеспечить мгновенный ответ в интерпретации запросов, за исключением тех случаев, когда дело уже поступило официальное рассмотрение.

Комментарии по пересмотру стандартов IEEE приветствуются от любой заинтересованной стороны, независимо от членской принадлежности к IEEE. Предложения по внесению изменений в документы должны быть в форме предлагаемого изменения текста вместе с соответствующими подтверждающими комментариями. Комментарии по стандартам и просьбы о толковании следует направлять по адресу:

Секретарь Совета по стандартам IEEE-SA
445 Hoes Lane
Почтовый ящик 1331
Пискатауэй, Нью-Джерси 08855-1331
США

Примечание: обращаем внимание на возможность того, что внедрение данного стандарта может потребоваться использование межпредметных мат- Тер защищена патентным правом. Публикуя настоящий стандарт, мы не занимаем никакой позиции относительно существования или действительности каких-либо патентных прав, связанных с ним. IEEE не несет ответственности за идентификацию патентов, для которых в соответствии со стандартом IEEE может потребоваться лицензия, или за проведение расследований юридической действительности или объема тех патентов, которые доведены до его сведения.

IEEE является единственной организацией, которая может разрешать использование сертификационных знаков, товарных знаков или других обозначений для обозначения соответствия материалам, изложенным в настоящем документе.

Введение

(Это введение не является частью стандарта IEEE Std 1149.1-2001 "Стандартный порт тестового доступа и архитектура граничного сканирования".)

Этот стандарт определяет порт тестового доступа и архитектуру граничного сканирования для цифровых интегральных схем и для цифровых частей смешанных аналоговых / цифровых интегральных схем. Средства, определенные стандартом, направлены на то, чтобы обеспечить решение проблемы тестирования собранных печатных плат и других продуктов, основанных на очень сложных цифровых интегральных схемах и технологиях поверхностного монтажа высокой плотности. Они также предоставляют средства доступа к функциям разработки для тестирования, встроенным в сами цифровые интегрированные системы, и управления ими. Такие функции могут, например, включать внутренние пути сканирования и функции самопроверки, а также а также другие функции, предназначенные для поддержки сервисных приложений в собранном продукте.

История разработки данного стандарта

Процесс разработки этого стандарта начался в 1985 году, когда в Европе была сформирована Объединенная европейская группа по проведению испытаний (JETAG). В 1986 году эта группа расширилась, включив в нее членов как из Европы , так и из Северной Америки, и, в результате, была переименована в Объединенную группу действий по тестированию (JTAG). В период с 1986 по 1988 год Технический подкомитет JTAG разработал и опубликовал серию предложений по стандартизированной форме сканирования границ. В 1988 году последнее из этих предложений — JTAG версии 2.0 — было предложено Комитету по стандартам IEEE по тестируемости шин (P1149) для включения в разрабатываемый в то время стандарт.
Комитет по стандартам тестирования шин согласился с таким подходом. Было решено, что предложение JTAG должно стать основой стандарта в семействе шин тестируемости, в результате чего был инициирован проект P1149.1. После принятия этих решений Технический подкомитет JTAG стал ядром Рабочей группы IEEE, разработавшей этот стандарт.

После первоначального утверждения этого стандарта в феврале 1990 года и его последующей публикации Рабочая группа немедленно приступила к разработке дополнения с целью исправления, уточнения и усовершенствования. Эти усилия, стимулируемые и направляемые взаимодействием между разработчиками и пользователями оригинального стандарта, завершились принятием стандарта IEEE Std 1149.1a-1993, который был утвержден в июне 1993 года.

Основными изменениями в этом стандарте, внесенными стандартом IEEE Std 1149.1a-1993, были

— Добавление двух дополнительных инструкций, *CLAMP* и *HIGHZ*, которые стандартизировали названия и спецификации функций, часто реализуемых как конструктивные особенности
— Добавление дополнительного средства для переключения компонента из режима, в котором он соответствует этому стандарту, в режим, в котором он поддерживает другой подход "разработка для тестирования"

Далее, начиная с предложения, сделанного Кеннетом П. Паркером и Стигом Оресьо в 1990 году, были предприняты усилия по разработке языка для описания компонентов, соответствующих этому стандарту. Эта работа завершилась утверждением стандарта IEEE Std 1149.1b-1994 в сентябре 1994 года.

Основным изменением, внесенным в этот стандарт стандартом IEEE Std 1149.1b-1994, стало добавление Приложения B, которое определяет язык описания граничного сканирования. Все остальные изменения были незначительными и носили исключительно пояснительный характер.

Изменения, внесенные этой редакцией

Эта редакция является в первую очередь служебным обновлением, предназначенным для закрепления знаний, полученных за первые 10 лет использования стандарта, в стандартном документе.

Внесены следующие основные изменения

— Чтобы снизить риск случайного перехода в тестовый режим, требование о том, чтобы двоичный код для *EXTEST* инструкции был {000...0}, было удалено, и использование этого двоичного кода для других инструкций, которые приводят к переходу в тестовый режим, устарело [см. 8.1.1 e), 8.8.1 h), B.8.11.3 d) и B.8.11.3 e)].

— Чтобы повысить гибкость, с которой инструкции могут быть реализованы и объединены, неявно объединенная *инструкция SAMPLE/PRELOAD* была переопределена как две отдельные инструкции: *SAMPLE* и *PRELOAD*. *Эти инструкции могут продолжать совместно использовать единый двоичный код*, что фактически приводит к объединенной инструкции *ВЫБОРКИ / ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ*, но в качестве альтернативы теперь они могут совместно использовать двоичные коды с другими инструкциями при условии, что никакие правила ни для одной из объединенных инструкций не нарушаются. [см. 8.1.1 g), 8.2.1 b), 8.6, 8.7, B.5.1.2, B.6.3 b), B.8.11.3 f), B.8.11.3 g), B.8.11.3 h), B.8.11.3 i), B.8.11.3 j) и B.8.13.3 a)].

— Для обеспечения более эффективной реализации ячеек регистра граничного сканирования, предоставляемых на выходах системной логики , источник данных, которые должны быть записаны в такие ячейки в ответ на команду *SAMPLE*, теперь может находиться на подключенном системном выводе [см. 8.6.1 d), 11.6.1 a), 11.6.1 b), B.8.14.4 l) и B.10.2.4.1 b)]. Кроме того, три новых типа ячеек, основанных на этой реализации (**BC_8** , **BC_9** , и **BC_10**) были добавлены в стандартный пакет VHDL (см. B.5.1.2, B.9 и таблицу B.3).

— Для обеспечения более гибких реализаций ячеек регистра граничной развертки в ограниченных случаях было разрешено совместное использование схем между регистром граничной развертки и другими элементами тестовой и/или системной логики [см. 11.2.1 j) и 11.2.1 k)].

— Для поддержки более полного описания драйверов выводов микросхемы со схемами bus keeper определено новое значение для <результат отключения> (**ХРАНИТЕЛЬ**, см. Б.5.1.2, Б.8.14.1 и Б.8.14.3.7).

— Для отслеживания широкого распространения BSDL этот язык был включен в качестве нормативной части стандарта и его использование в документации было санкционировано (см. 13.3.1 c) и приложение B).

Кроме того, был внесен ряд незначительных изменений для исправления и уточнения формулировок стандарта [особое примечание, см. 4.8.1 c), 11.7.1 b), B.8.14.4 k) и B.8.14.4 n)].

Участники

На момент выпуска стандарта IEEE Std 1149.1-2001 членами рабочей группы были:

Кристофер Дж. Кларк, *председатель*

Adam W. Лей, *редактор*

Джон Эндрюс
Билл Аронсон
Карл Ф. Барнхарт
Дилип К. Бхавсар
Терри Борроз
Джон Брейден
Билл Брюс
Тапан Дж. Чакраборти
Сон Чанг
Джим Коулман
Франс Де Йонг
Бюлент Дервисоглу
Тед Итон
Билл Эклоу
Дин Гердес
Грэди Л. Джайлз
Питер Хансен
Нил Джейкобсон
Наджми Джарвала
Лондон Джин
Вуудиан Ке
Артур Кху
Рамеш Кришнамурти
Том Лэнгфорд
Ларри Ли
Колин Маундер
Бенуа Надо-Дости
Джей Неджедло
Кеннет П. Паркер
Майкл Рикетти
Гордон Д. Робинсон
Роберт Дж. Рассел
Адам Шеппард
Стив Старк
Рон Вальтер
Ли Уэтсел
Томас В. Уильямс

Следующие члены группы по голосованию проголосовали за этот стандарт. Участники голосования могли голосовать за одобрение, неодобрение или воздержаться. :

Джон Эндрюс
Карл Барнхарт
Роджер Беннеттс
Терри Борроз
Джон Брейден
Кит Чоу
Сон С. Чанг
Крис Дж . Кларк
Франс де Йонг
Тед Итон
Уильям Эклоу
Дин Гердес
Грейди Джайлз
Питер Хансен
Ли Ф. Хорни
Мицуаки Исикава
Нил Джейкобсон
Джейк Каррфолт
Томас Л. Лэнгфорд
Адам В. Лей
Грегори А. Мастон
Колин Маундер
Патрик Макхью
Эрл Дж. Мейерс
Инхуа Мин
Джеймс А. Монзел
Бенуа Надо-Дости
Джей Неджедло
Брюс Э. Петерсон
Майк Рикетти
Гордон Робинсон
Джайдип Рой
Роберт Дж. Рассел
Скотт А. Ялкорт
Т. В. Уильямс

Когда Совет по стандартам IEEE-SA утвердил этот стандарт 14 июня 2001 года, в нем было следующее членство:

Дональд Н. Хейрман, *председатель*
Джеймс Т. Карло, *заместитель председателя*
Джудит Горман, *секретарь*

Сатиш К. Аггарвал
Марк Д. Боумен
Гэри Р. Энгманн
Гарольд Э. Эпштейн
Х. Лэндис Флойд
Джей Форстер*
Ховард М. Фрейзер
Рубен Д. Гарсон
Джеймс Х. Герни
Ричард Дж. Холлеман
Лоуэлл Г. Джонсон
Роберт Дж. Кеннелли
Джозеф Л. Кепфингер*
Питер Х. Липс
Л. Брюс Маккланг
Дейлип К. Мохла
Джеймс У. Мур
Роберт Ф. Манзнер
Рональд К. Петерсен
Джеральд Х. Петерсон
Джон Б. Поузи
Гэри С. Робинсон
Акио Тодзио
Дональд У. Зипс

* Почетный член

Также включены следующие представители Совета по стандартам IEEE-SA, не имеющие права голоса:

Алан Куксон, *представитель NIST*
Дональд Р. Волзка, *представитель TAB*

Дон Мессина
Редактор проектов стандартов IEEE

Содержание

Стандартный порт тестового доступа IEEE и архитектура пограничного сканирования

1. Общий обзор

1.1 Область применения

Этот стандарт определяет логику тестирования, которая может быть включена в интегральную схему для обеспечения стандартизированных подходов к

— тестирование межсоединений между интегральными схемами после их сборки на печатной плате или другой подложке;
— тестирование самой интегральной схемы; и
— наблюдение или изменение активности схемы во время нормальной работы компонента.

Логика тестирования состоит из регистра граничного сканирования и других стандартных блоков, доступ к которому осуществляется через порт тестового доступа (TAP).

1.2 Цель

1.2.1 Обзор работы стандарта IEEE Std 1149.1

Этот подраздел содержит общий обзор работы компонента, совместимого с данным стандартом, и служит основой для подробного обсуждения в последующих подразделах.

Схема, определенная настоящим стандартом, позволяет вводить инструкции по тестированию и связанные с ними тестовые данные в компонент и, впоследствии, позволяет считывать результаты выполнения таких инструкций. Вся информация (инструкции, тестовые данные и результаты тестирования) передается в последовательном формате.

Последовательностью операций будет управлять мастер шины, который может быть либо автоматическим тестовым оборудованием (ATE), либо компонентом, который взаимодействует с тестовой шиной более высокого уровня как часть полной архитектуры обслуживания системы . Управление достигается с помощью сигналов, подаваемых на входы выбора тестового режима (TMS) и тестовых тактовых импульсов (TCK) различных компонентов, подключенных к ведущей шине. Начиная с начального состояния, в котором тестовая схема, определенная настоящим стандартом, неактивна, типичная последовательность операций будет следующей.

Первыми шагами, как правило, были бы последовательные загрузки в компонент двоичного кода инструкции для конкретной операции, которая должна быть выполнена. Логика тестирования, определенная настоящим стандартом, спроектирована таким образом, что последовательное перемещение информации команды не является очевидным для тех схемных блоков, работой которых управляет команда. Инструкция, применяемая к этим блокам, изменяется только по завершении процесса переключения (загрузки инструкций).

Как только команда загружена, выбранная тестовая схема настроена на ответ. В некоторых случаях, однако, необходимо загрузить данные в выбранную тестовую схему, прежде чем можно будет получить значимый ответ . Такие данные загружаются в компонент последовательно способом, аналогичным процессу, использованному ранее для загрузки инструкции. Обратите внимание, что перемещение тестовых данных никак не влияет на инструкцию, присутствующую в тестовой схеме.

После выполнения инструкции по тестированию, основанной, при необходимости, на предоставленных данных, результаты теста могут быть проверены путем передачи данных из компонента в мастер шины или через него.

Обратите внимание, что в случаях, когда необходимо повторить одну и ту же тестовую операцию, но с другими данными, новые тестовые данные могут быть перенесены в компонент, в то время как результаты тестирования будут перенесены наружу. Нет необходимости в перезагрузке инструкции.

Работа тестовой схемы может продолжаться путем загрузки и выполнения нескольких дополнительных инструкций способом, подобным описанному, и завершаться возвратом тестовой схемы и, при необходимости, встроенной системной схемы в исходное состояние.

1.2.2 Использование стандарта IEEE Std 1149.1 для тестирования собранного изделия

В этом подпункте описывается использование схемы граничного сканирования, определенной настоящим стандартом, в процессе тестирования собранного изделия, такого как печатная плата.

Тестовую задачу для любого продукта, построенного из набора компонентов, можно разбить на три задачи:

a) Для подтверждения того, что каждый компонент выполняет свою требуемую функцию;
b) Для подтверждения того, что компоненты соединены между собой правильным образом; и
c) Для подтверждения того, что компоненты в продукте взаимодействуют правильно и что продукт выполняет свою предназначенную функцию.

Этот подход может быть применен к плате, построенной из интегральных схем, к системе, построенной из печатных плат, или к сложной интегральной схеме, построенной из набора более простых функциональных модулей. Чтобы упростить обсуждение, в дальнейшем это описание будет сосредоточено на случае собранной печатной платы, изготовленной из набора цифровых интегральных схем.

На уровне платы цель a) и цель b) обычно достигаются с помощью методов внутрисхемного тестирования; для цели c) требуется функциональный тест. Однако методы внутрисхемного тестирования имеют существенные ограничения, если рассматривать их по сравнению с развивающейся технологией межсоединений для поверхностного монтажа, например, сложность установления надежного контакта с миниатюрными элементами печатной платы с помощью крепления на гвоздях. Как же тогда могут быть достигнуты вышеуказанные три цели тестирования, если тестовый доступ будет ограничен обычными подключениями к сети плюс относительно небольшое количество тестовых подключений специального назначения?

Учитывая цель a), ясно, что поставщик интегральной схемы, используемой при проектировании на уровне платы, будет иметь установленную методологию тестирования этого компонента. Компоненты могут быть протестированы на собственной системе ATE или с использованием процедуры самопроверки, встроенной в конструкцию. Информация о принятой методологии тестирования, как правило, недоступна покупателю компонентов. Даже там, где известно о существовании режимов самотестирования, они могут не быть задокументированы и, следовательно, недоступны пользователю компонента. Альтернативными источниками тестовых данных для инженера-испытателя платы могут быть библиотеки тестов компонентов, поставляемые с

внутрисхемные тестовые системы или тестовые программы, разработанные пользователями компонентов для входного контроля поставляемых устройств.

Где бы ни были получены тестовые данные для компонента, следующим шагом является их использование после того, как компонент был собран на печатной плате. Если доступ ограничен обычными соединениями собранной схемы, эта задача может оказаться далеко не простой. Это особенно верно, если окружающие компоненты сложны или если разработчик платы привязал соединения некоторых компонентов к фиксированным логическим уровням или оставил выводы компонентов неподключенными. Как правило, невозможно протестировать компонент тем же способом, которым он тестировался изолированно, если только не возможно провести внутрисхемное тестирование.

Для обеспечения возможности использования встроенных средств тестирования или применения ранее существовавших шаблонов тестирования, необходима платформа, которая может использоваться для передачи тестовых данных к границам отдельных компонентов или от них, чтобы их можно было тестировать так, как если бы они были автономными. Эта платформа также обеспечит доступ к встроенным средствам тестирования компонентов и управление ими. В средствах тестирования компонентов. Сканирование границ в сочетании с шиной тестового доступа обеспечивает такую платформу.

Целью настоящего стандарта является определение архитектуры граничного сканирования, которая может быть принята в качестве стандартной характеристики интегральных схем, что позволяет создавать необходимую тестовую среду на собранных печатных платах и других продуктах.

1.2.3 Что такое пограничное сканирование?

Технология граничного сканирования включает включение каскада сдвигового регистра (содержащегося в ячейке регистра граничного сканирования) рядом с каждым выводом компонента, так что сигналами на границах компонентов можно управлять и наблюдать, используя принципы тестирования сканирования.

Рисунок 1-1 иллюстрирует пример реализации ячейки регистра с граничным сканированием, которая может использоваться для входного или выходного подключения к интегральной схеме. В зависимости от управляющих сигналов, подаваемых на мультиплексоры, данные могут быть либо загружены в регистр сканирования из порта ввода сигнала (например, входной контакт), либо выведены из регистра через порт вывода сигнала ячейки (например, в ядро конструкции компонента).
Как будет подробно обсуждаться в разделе 11, второй триггер (управляемый входным тактовым сигналом B) предусмотрен для обеспечения того, чтобы сигналы, выводимые из ячейки в последнем случае, удерживались, пока новые данные перемещаются в ячейку с использованием входного тактового сигнала A. Этот триггер не требуется во всех случаях, но приведен на рисунке 1-1 для упрощения обсуждения.

Рисунок 1-1—Ячейка регистра пограничного сканирования

Ячейки регистра граничной развертки для выводов компонента соединены между собой, образуя цепочку регистров сдвига по границе конструкции, и этот путь снабжен последовательными входными и выходными соединениями и

соответствующие тактовые и управляющие сигналы. В продукте, собранном из нескольких интегральных схем, регистры граничного сканирования для отдельных компонентов могут быть соединены последовательно, чтобы сформировать единый канал для всей конструкции, как показано на рисунке 1-2. В качестве альтернативы, конструкция платы может содержать несколько независимых контуров сканирования границ.

Рисунок 1-2—Схема платы с возможностью сканирования границ

Если все компоненты, используемые для построения схемы, имеют регистр граничного сканирования, результирующий последовательный путь через всю конструкцию может быть использован двумя способами:

а) Чтобы обеспечить возможность тестирования взаимосвязей между различными компонентами, тестовые данные могут быть перенесены во все ячейки регистра граничного сканирования, связанные с выходными выводами компонентов, и загружены параллельно через межсоединения компонентов в эти ячейки, связанные с входными выводами; и

б) Чтобы позволить тестировать компоненты на плате, регистр граничного сканирования может использоваться как средство изоляции системной логики на кристалле от воздействий, поступающих от окружающих компонентов, во время выполнения внутренней самодиагностики. В качестве альтернативы, если регистр граничного сканирования сконструирован надлежащим образом, он может разрешить ограниченное статическое тестирование системной логики на кристалле с низкой скоростью, поскольку он позволяет доставлять тестовые данные компоненту и проверять результаты тестирования.

Эти тесты позволяют достичь первых двух целей, о которых говорилось ранее, за счет использования регистра пограничного сканирования . По сути, тесты, применяемые с использованием register, могут обнаруживать многие неисправности, которые в настоящее время устраняют внутрисхемные тестировщики, но без необходимости широкого доступа к "гвоздям". Третья цель — функциональное тестирование работы всего продукта — остается неизменной и может быть достигнута либо с помощью функциональной (через pins) системы ATE, либо, например, с помощью самотестирования на системном уровне.

Обратите также внимание, что за счет параллельной загрузки ячеек как на входах, так и на выходах компонента и смещения результатов регистр граничного сканирования обеспечивает средство “выборки” данных, проходящих через компонент, не влияя на поведение компонента или собранной платы. Этот режим работы полезен для отладки конструкции и диагностики неисправностей, поскольку позволяет проверять соединения, которые обычно не доступны тестовой системе.

1.2.4 Использование стандарта IEEE Std 1149.1 для достижения других целей тестирования

Помимо применения при тестировании печатных плат и других изделий, содержащих несколько компонентов, логика тестирования, определенная настоящим стандартом, может использоваться для обеспечения доступа к широкому спектру конструкторских решений

функции для тестирования, встроенные в сами компоненты. Такие функции могут включать внутренние пути сканирования, функции самотестирования [например, с использованием встроенных элементов logic block observer (BILBO)] или другие вспомогательные функции.

Подобные функции, предназначенные для тестирования, могут быть доступны и управляться с помощью канала передачи данных между последовательными выводами тестовых данных ОТВОДА, определенными настоящим стандартом. Инструкции, которые вызывают внутреннюю реконфигурацию системной логики компонента таким образом, чтобы была включена тестовая операция, могут быть переданы в компонент через
ОТВОД.

1.3 Схема документа

Схемы, подобные тем, которые определены в настоящем стандарте, легче понять, если их спецификации сопровождаются общими описательными материалами, которые раскрывают детали различных частей конструкции в перспективе и предоставляют примеры реализации. Таким образом, раздел 1 содержит обзор применения настоящего стандарта к тестированию цифровых частей электронного изделия, состоящего из множества интегральных схем.

Последующие разделы документа содержат спецификации для конкретных функций этого стандарта. В этих разделах содержатся два класса материалов:

1.3.1 Технические характеристики

Подразделы, озаглавленные "Спецификации", содержат правила, рекомендации и разрешения, которые определяют этот стандарт:

a) *Правила* определяют обязательные аспекты настоящего стандарта. Подпункты, являющиеся правилами, содержат слово *должны*.
b) *Рекомендации* указывают на предпочтительную практику для конструкций, которые стремятся соответствовать этому стандарту. Подпункты , являющиеся рекомендациями, содержат слово *следует*.
c) *Разрешения* показывают, как дополнительные функции могут быть введены в дизайн, который стремится соответствовать этому стандарту. Эти функции расширят область применения тестовой схемы, определенной настоящим стандартом. Подразделы, являющиеся разрешениями, содержат слово *может*.

1.3.2 Описания

Материал, не содержащийся в подразделах, озаглавленных "Спецификации", является описательным материалом, который иллюстрирует необходимость в определяемых функциях или их применении. Этот материал включает схемы, иллюстрирующие возможную реализацию спецификаций настоящего стандарта. В приложении A к настоящему стандарту содержится альтернативный пример реализации. В описательном материале также обсуждаются проектные решения, принятые в ходе разработки данного стандарта.

ВНИМАНИЕ: Описательный материал, содержащийся в настоящем стандарте, носит исключительно иллюстративный характер и не определяет предпочтительную реализацию. В настоящем стандарте приведены примеры, иллюстрирующие возможные реализации схем. Там, где могут возникать расхождения между примерами и спецификациями, спецификации всегда имеют приоритет. Читателям следует проявлять осторожность при использовании этих примеров, чтобы гарантировать полное соответствие в их конкретных приложениях. В частности, подчеркивается, что примеры разработаны для эффективного донесения значения этого стандарта. Как таковые, они логически корректны; однако, как всегда, конкретная реализация может работать некорректно в отношении синхронизации и других параметрических характеристик. Одним из примеров этой проблемы является то, что примерная реализация контроллера ОТВОДА, изображенная на рисунке 6-5, обоснованно предполагает значительно большую задержку в триггерных источниках A и A*, чем в источнике TCK инвертора*. Можно спроектировать схему, в которой это допущение нарушается, вызывая возникновение критической гонки, которая сделает недействительным поведение контроллера отвода. Поэтому рекомендуется, чтобы внедрение было полностью проверено для обеспечения соответствия требуемым эксплуатационным условиям.

1.4 Условные обозначения

В настоящем стандарте используются следующие условные обозначения:

a) Правила, рекомендации и разрешения в каждом подразделе Спецификаций содержатся в едином списке, проиндексированном в алфавитном порядке. Ссылки на каждое правило, рекомендацию или разрешение приведены в форме:

```
                    15.1.1 в) 2)
                         |    |    |
 Номер подпункта         |    |    |
                      Указатель    |
              Опция (если есть)
```

b) Названия инструкций и состояний, определенные в настоящем стандарте, в тексте настоящего стандарта выделены *курсивом* .

c) Названия состояний и сигналов, которые относятся к регистрам тестовых данных, определенным настоящим стандартом, содержат символы DR, в то время как те, которые относятся к регистру команд, содержат символы IR.

г) Названия сигналов, которые активны в их низком состоянии, имеют звездочку в качестве последнего символа, например, TRST*.

e) Используется соглашение о положительной логике; т. е. Сигнал логики 1 передается как более положительное из двух напряжений, используемых для логических сигналов.

2. Список литературы

Настоящий стандарт должен использоваться в сочетании со следующими стандартами. Когда следующие стандарты заменяются утвержденной редакцией, применяется редакция.

EIA/JEP106, Публикация JEDEC 106, стандартный идентификационный код производителя [1]

IEEE 100, *Авторитетный словарь терминов стандартов IEEE*, Седьмое издание [2]

Стандарт IEEE Std 1076-1993, Справочное руководство по языку стандарта IEEE VHDL

Стандарт IEEE Std 1149.4-1999, стандарт IEEE для тестовой шины смешанных сигналов

3. Определения и сокращения

Для целей настоящего стандарта применяются следующие термины и определения. На термины, не определенные в этом*разделе, следует ссылаться на IEEE 100, Авторитетный*словарь терминов стандартов IEEE, Седьмое издание.

3.1 Определения

3.1.1 активный: Когда он связан с логическим уровнем (например, активный-низкий), этот термин определяет логический уровень, на который должен быть установлен сигнал, вызывающий выполнение определенного действия. При упоминании выходного драйвера (например, активного привода) этот термин описывает состояние, в котором драйвер способен определять напряжение сети, к которой он подключен.

[1] Копии можно получить в EIA/JEDEC, бульвар Уилсона, 2500, Арлингтон, Вирджиния 22201-3834, США (www.jedec.org).

[2] Публикации IEEE доступны в Институте инженеров электротехники и электроники, 445 Hoes Lane, почтовый ящик 1331, Пискатауэй, NJ 08855-1331, США (http://standards.ieee.org /).

3.1.2 двунаправленный вывод: компонентный вывод, который может либо управлять, либо принимать сигналы от внешних соединений.

3.1.3 опрос вслепую: Доступ к объекту (например, регистратору идентификации устройства) без предварительного знания логики тестирования конкретного компонента, к которому осуществляется доступ.

3.1.4 встроенный логический блок наблюдателя (Бильбо): сдвиг-зарегистрируйтесь на основе структуры используется при некоторых формах собственн-испытание схем.

3.1.5 захват: Загрузка значения в регистр данных или регистр команд в результате перехода в состояние контроллера *Capture-DR* или *Capture-IR* соответственно.

3.1.6 Тестирование микросхемы на плате: Испытание компонента после его сборки на печатной плате или другой подложке.

3.1.7 Тактовый сигнал: сигнал, в котором переходы между низким и высоким логическими уровнями (или наоборот) используются для указания, когда устройство с сохраненным состоянием, такое как триггер или защелка, может выполнить операцию.

3.1.8 понижающаяся граница: Переход от высокого логического уровня к низкому. В позитивной логике - переход от логики 1 к логике 0. События, которые должны происходить на восходящем (нисходящем) фронте сигнала, должны завершаться в течение фиксированной (не зависящей от частоты) задержки, указанной поставщиком компонентов.

3.1.9 Высокое: более высокое из двух напряжений, используемых для передачи одного бита информации. Для позитивной логики - логика 1.

3.1.10 неактивный: Когда речь идет о выходном драйвере (например, неактивном накопителе), этот термин описывает состояние, в котором драйвер не способен определять напряжение сети, к которой он подключен.

3.1.11 Входной вывод: компонентный вывод, который принимает сигналы от внешнего соединения.

3.1.12 инструкция: Двоичное слово данных, последовательно перемещаемое в логику тестирования для определения его последующей операции.

3.1.13 Младший значащий бит (LSB): цифра в двоичном числе, представляющая наименьшее числовое значение. Для сдвиговых регистров - бит, расположенный ближе всего к последовательному выходу, или первый бит, подлежащий смещению. Младший значащий бит двоичного слова или сдвигового регистра имеет номер 0.

3.1.14 проектирование сканирования с учетом уровня (LSSD): вариант метода проектирования сканирования, в результате которого получаются непротиворечивые, тестируемые цифровые электронные схемы.

3.1.15 Низкое: наименьшее из двух напряжений, используемых для передачи одного бита информации. Для положительной логики - логический 0.

3.1.16 Старший значащий бит (MSB): цифра в двоичном числе, представляющая наибольшее числовое значение. Для сдвиговых регистров - бит, наиболее удаленный от последовательного вывода, или последний бит, который должен быть смещен. Логические значения, выраженные в двоичной форме, показаны с указанием старшего бита слева.

3.1.17 отсутствие синхронизации: сигнал, при котором переходы между низким и высоким логическими уровнями сами по себе не вызывают срабатывание устройств с сохраненным состоянием. Логический уровень важен только во время перехода по тактовому сигналу.

3.1.18 Выходной вывод: компонентный вывод, который передает сигналы на внешние соединения.

3.1.19 вывод: Точка, в которой осуществляется соединение между интегральной схемой и подложкой, на которой она установлена (например, печатной платой). Для упакованных компонентов это будет штифт для упаковки; для компонентов, монтируемых непосредственно на подложку, это будет связующая прокладка.

3.1.20 основной источник: В случае, если несколько поставщиков предлагают компоненты, совместимые с pin-кодом, основным источником является поставщик, представивший тип компонента.

3.1.21 Частный: Конструктивная особенность, предназначенная исключительно для использования производителем компонентов.

3.1.22 Общедоступный: Конструктивная особенность, задокументированная в паспорте компонентов, которая может быть использована покупателями компонента.

3.1.23 сброс: Установление начального логического условия, которое может быть либо логическим 0, либо логическим 1, как определяется контекстом.

3.1.24 Восходящее преимущество: переход от низкого уровня логики к высокому. В позитивной логике - переход от логики 0 к логике 1. События, которые должны происходить на восходящем (нисходящем) фронте сигнала, должны завершаться в течение фиксированной (не зависящей от частоты) задержки, указанной поставщиком компонентов.

3.1.25 схема сканирования: Метод проектирования, который вводит пути сдвигового регистра в цифровые электронные схемы и тем самым улучшает их тестируемость.

3.1.26 путь сканирования: Путь сдвигового регистра через схему, разработанную с использованием технологии сканирования.

3.1.27 второй источник: В случае, если несколько поставщиков предлагают компоненты, совместимые с pin-кодом, поставщиками второго источника являются поставщики компонента, отличные от основного источника.

3.1.28 Выбранный регистр тестовых данных: Регистр тестовых данных выбирается, когда требуется выполнить инструкцию, подаваемую в логику тестирования.

3.1.29 анализ сигнатур: Метод сжатия последовательности логических значений, выводимых из тестируемой схемы, в небольшое количество битов данных (сигнатур), которые при сравнении с сохраненными данными укажут на наличие или отсутствие неисправностей в схеме.

3.1.30 Автономное тестирование: Испытание компонента, выполняемое перед его сборкой на плате или другом основании, например, с использованием автоматического испытательного оборудования (ATE).

3.1.31 неисправность при зависании: Сбой в логической схеме, из-за которого сигнальное соединение фиксируется на 0 или 1, независимо от работы схемы, которая им управляет.

3.1.32 система: Относящийся к нетестовой функции схемы.

3.1.33 системная логика: Любой элемент логики, предназначенный для реализации нетестовой функции компонента или на интересующий момент времени сконфигурированный для реализации некоторого аспекта нетестовой функции.

3.1.34 Системный вывод: Вывод компонента, который питает системную логику на кристалле.

3.1.35 тестовая логика: Любой элемент логики, который является выделенной частью архитектуры тестовой логики или на момент интереса сконфигурирован как часть архитектуры тестовой логики.

3.1.36 обновление: Передача логического значения из каскада сдвигового регистра ячейки регистра данных или ячейки регистра инструкций
в каскад параллельного вывода ячейки с блокировкой в результате падения фронта входного сигнала тестового*тактового сигнала в состоянии контроллера* Update-DR *или* Update-IR**,** соответственно.

3.1.37 Вывод с тремя состояниями: Вывод компонента, на котором привод может быть как активным, так и неактивным (например, при высоком сопротивлении).

3.2 Сокращения

ATE	автоматическое испытательное оборудование
БИЛЬБО	встроенный логический блок observer
LSB	младший значащий бит
LSSD	конструкция сканирования с чувствительностью к уровню
MSB	Старший значащий бит
КОСНИТЕСЬ	проверьте порт доступа (см. Раздел 4)
TCK	тестовый тактовый вход (см. 4.2)
TDI	ввод тестовых данных (см. 4.4)
TDO	выходные данные тестирования (см. 4.5)
TMS	выбор тестового режима (см. 4.3)
TRST*	тестовый сброс (см. 4.6)
TTL	транзисторная логика

4. Проверьте порт доступа (TAP)

TAP - это порт общего назначения, который может предоставлять доступ ко многим функциям поддержки тестирования, встроенным в компонент, включая логику тестирования, определенную этим стандартом. Он состоит как минимум из трех входных соединений и одного выходного соединения, требуемых логикой тестирования, определенной настоящим стандартом. Необязательное четвертое входное соединение обеспечивает асинхронную инициализацию тестовой логики, определенной этим стандартом.

4.1 Соединения, образующие порт тестового доступа (TAP)

4.1.1 Технические характеристики

Правила

а) Отвод должен включать следующие соединения (определенные в 4.2, 4.3, 4.4 и 4.5): TCK, TMS, TDI и TDO.

б) Если контроллер TAP не сбрасывается при включении питания в результате функций, встроенных в логику тестирования, должен быть предусмотрен вход TRST *, как определено в 4.6 (см. также 6.3).

с) Все входы и выходы TAP должны быть специально подключены к компоненту (т. е. Используемые контакты не должны использоваться для каких-либо других целей).

4.1.2 Описание

Для обеспечения доступа ко всему набору обязательных функций этого стандарта требуются специальные подключения TAP.

4.2 Тестовый тактовый вход (TCK)

Тестовый тактовый вход (TCK) обеспечивает синхронизацию для тестовой логики, определенной этим стандартом.

4.2.1 Технические характеристики

Правила

а) Устройства с сохраненным состоянием, содержащиеся в тестовой логике, должны сохранять свое состояние неопределенно долго, когда сигнал, подаваемый на TCK, прекращается на 0.

Рекомендации

б) Поскольку входы TCK для многих компонентов могут управляться с помощью одного драйвера, следует позаботиться о том, чтобы нагрузка, создаваемая TCK, была как можно меньше.

Разрешения

с) Устройства с сохраненным состоянием, содержащиеся в логике тестирования, могут сохранять свое состояние неопределенно долго, когда сигнал, подаваемый на TCK, прекращается на 1.

4.2.2 Описание

Специальный вход TCK включен для того, чтобы можно было использовать последовательный канал передачи тестовых данных между компонентами независимо от системных часов, зависящих от конкретного компонента, частота которых может значительно отличаться от одного компонента к другому. Это также позволяет изменять тестовые данные одновременно с нормальной работой системы компонента. Последнее средство требуется для поддержки использования регистров TAP и тестовых данных в конструкции для оперативного мониторинга системы. Наличие независимой синхронизации гарантирует, что тестовые данные могут быть перемещены в микросхему или с микросхемы без изменения состояния встроенной системной логики. Независимая синхронизация также важна, если регистры граничного сканирования должны использоваться для тестирования межсоединений платы при любых обстоятельствах, включая случаи, когда системные тактовые сигналы выводятся в одном компоненте для использования в других.

Хотя TCK во многих случаях будет управляться автономными часами с номинальным рабочим циклом 50%, могут возникнуть ситуации, когда часы необходимо остановить на некоторое время. Одним из примеров является случай, когда ATE необходимо извлечь тестовые данные из резервной памяти (например, с диска), поскольку некоторые тестовые системы не могут поддерживать работу часов во время такой операции. Этот стандарт требует, чтобы TCK можно было останавливать на 0 бесконечно долго, не вызывая никаких изменений состояния тестовой логики. Пока сигнал TCK остановлен на 0, устройства с сохраненным состоянием должны сохранять свое состояние, чтобы тестовая логика могла продолжить свою работу при перезапуске синхронизации. Необязательно, компонент также может позволять останавливать TCK на 1 на неопределенный период.

Многие части тестовой логики выполняют операции в ответ на возрастающий или падающий фронт TCK, на что указывает использование фразы “на возрастающем (падающем) фронте TCK”. Эти операции должны быть завершены в течение фиксированной (не зависящей от частоты) задержки после возникновения соответствующего изменения в TCK, и эта задержка должна определяться поставщиком компонентов. Следовательно, фразу “на восходящем (нисходящем) фронте TCK” следует интерпретировать как “в пределах указанной задержки после восходящего (нисходящего) фронта TCK”.

ПРИМЕЧАНИЕ—Во многих приложениях сигнал TCK, подаваемый на компоненты, соответствующие этому стандарту, будет иметь рабочий цикл, близкий к 50% (т. е. Периоды, которые часы проводят на 0 и 1, будут равны) на заданной частоте. Ожидается, что все задержки распространения будут такими, чтобы при этих обстоятельствах обеспечивалась корректная работа (рабочий цикл 50% при данной частоте TCK), особенно когда данные передаются между TDO одного чипа и TDI другого.

4.3 Выбор режима тестирования (TMS)

Сигнал, полученный в TMS, декодируется контроллером TAP для управления тестовыми операциями.

4.3.1 Технические характеристики

Правила

а) Сигнал, представленный в TMS, должен быть отобран логикой тестирования на переднем фронте TCK.

б) Конструкция схемы, питаемой от TMS, должна быть такой, чтобы невключенный входной сигнал выдавал логический отклик, идентичный применению логики 1.

Рекомендации

с) Поскольку входами TMS для многих компонентов можно управлять с помощью одного драйвера, следует позаботиться о том, чтобы нагрузка, создаваемая TMS, была как можно меньше.

4.3.2 Описание

Правило 4.3.1 b) включено таким образом, что контроллер отвода принудительно переводится в состояние *Тестовой логики-сброса* контроллера в случае неиспользованного вывода TMS. Это гарантирует, что нормальная работа всей конструкции может продолжаться без-помех от логики тестирования (см. 6.3). Для конструкций, совместимых с TTL, это правило может быть соблюдено путем включения подтягивающего резистора во входную схему TMS компонента.

Значения сигнала, представленные в TMS, отбираются тестовой логикой на переднем фронте TCK. Ожидается, что мастер шины (ATE, контроллер шины и т.д.) Изменит сигнал, подаваемый на входы TMS подключенных компонентов на падающем фронте TCK. Формы сигналов, приведенные в других разделах настоящего стандарта, отражают это ожидание.

4.4 Ввод тестовых данных (TDI)

Команды и данные последовательного тестирования принимаются логикой тестирования в TDI.

4.4.1 Технические характеристики

Правила

а) Сигнал, представленный на TDI, должен быть введен в логику тестирования на переднем фронте TCK.

б) Конструкция схемы, питаемой от TDI, должна быть такой, чтобы невключенный вход выдавал логический отклик, идентичный применению логики 1.

в) Когда данные перемещаются из TDI в TDO, тестовые данные, полученные в TDI, должны отображаться без инверсии в TDO после ряда восходящих и нисходящих фронтов TCK, определяемых длиной выбранной инструкции или регистра тестовых данных.

4.4.2 Описание

Контакты данных (TDI и TDO) обеспечивают последовательную передачу тестовых данных по цепи. Требование о передаче данных из TDI в TDO без инверсии включено для упрощения работы компонентов, совместимых с этим стандартом, соединенных на печатной плате.

Значения, представленные в TDI, синхронизируются в выбранном регистре (инструкции или тестовые данные) по возрастающему фронту TCK. Ожидается, что мастер шины (ATE, контроллер шины и т.д.) Изменит сигнал, подаваемый на вход TDI первого компонента по последовательному каналу на уровне платы на падающем фронте TCK. Формы сигналов, показанные в других разделах настоящего стандарта, отражают это ожидание.

Правило 4.4.1 b) включено для того, чтобы ошибки обрыва цепи в последовательном тестовом канале передачи данных на уровне платы приводили к смещению определенного логического значения в тестовую логику. Обратите внимание, что когда это постоянное значение сдвигается в регистр команд , будет выбран регистр обхода (как будет обсуждаться далее в разделе 8.4). Для TTL-совместимых конструкций это правило может быть соблюдено путем включения подтягивающего резистора во входную схему компонента TDI.

4.5 Вывод тестовых данных (TDO)

TDO - это последовательный вывод для тестовых инструкций и данных из тестовой логики, определенной в этом стандарте.

4.5.1 Технические характеристики

Правила

а) Изменения состояния сигнала, передаваемого через TDO, должны происходить только на падающем фронте TCK.
б) Драйвер TDO должен быть переведен в состояние неактивного привода, за исключением случаев, когда выполняется сканирование данных (см. 6.2).

4.5.2 Описание

Для обеспечения бесперебойной работы изменения на входах ОТВОДОВ (TMS и TDI) синхронизируются с логикой тестирования, определенной этим стандартом, на восходящем фронте TCK, в то время как изменения на выходе ОТВОДОВ (TDO) происходят на нисходящем фронте TCK. Аналогично, для тестовой логики, способной управлять или принимать сигналы от системных выводов (например, регистра пограничного сканирования), сигналы, выводимые компонентом из тестовой логики, изменяют состояние на понижающемся фронте TCK, в то время как сигналы, поступающие в тестовую логику, синхронизируются на повышающемся фронте (как будет обсуждаться в разделе 9.3).

Содержимое выбранного регистра (инструкция или данные) смещается за пределы TDO по падающему краю TCK.
На иллюстрациях, приведенных в этом документе, обычно используются схемы с краевым управлением. Для реализации, управляемой с помощью edge, обратите внимание, что изменения выходных данных TDO должны быть отложены до спадающего фронта TCK, что может быть достигнуто путем включения триггера, синхронизированного с падающим фронтом TCK, в выходной буфер TDO.
Там, где регистры построены из защелок master и slave, управляемых неперекрывающимися тактовыми сигналами, повторное включение, требуемое правилом 4.5.1 a), является неотъемлемой особенностью конструкции.

Способность TDO переключаться между активным и неактивным приводом необходима для обеспечения параллельного, а не последовательного подключения каналов тестовых данных на уровне платы в случаях, когда это требуется. Например, в технологиях TTL или CMOS это требование может быть выполнено за счет использования выходного буфера с тремя состояниями.

4.6 Тестовый ввод сброса (TRST*)

Дополнительный вход TRST* обеспечивает асинхронную инициализацию контроллера TAP (см. 6.3).

4.6.1 Технические характеристики

Правила

a) Если в TAP включен TRST*, контроллер TAP должен быть асинхронно сброшен в состояние *тестовой логики-Reset* контроллера, когда к TRST* применяется логический 0 (см. 6.3).

ПРИМЕЧАНИЕ — В результате этого события вся остальная тестовая логика в компоненте асинхронно сбрасывается в состояние, требуемое в *состоянии контроллера "Тестовая логика-сброс"*.

б) Если в ОТВОД включен TRST*, конструкция схемы, питаемой с этого входа, должна быть такой, чтобы невключенный вход выдавал логический отклик, идентичный применению логики 1.

c) TRST* не должен использоваться для инициализации какой-либо системной логики в компоненте.

Рекомендации

d) Чтобы обеспечить детерминированную работу тестовой логики, значение TMS должно удерживаться на уровне 1, пока сигнал, подаваемый при TRST*, изменяется с 0 на 1.

4.6.2 Описание

Инициализация контроллера TAP, в свою очередь, вызывает асинхронную инициализацию другой тестовой логики, включенной в проект, как описано в последующих разделах этого стандарта.

Правило 4.6.1 b) включено для гарантии того, что в случае нерегулируемого входа TRST* тестовая логическая операция может выполняться под управлением сигналов, подаваемых на входы TMS и TCK. Для конструкций, совместимых с TTL, это правило может быть соблюдено путем включения подтягивающего резистора во входную схему TRST* компонента.

Правило 4.6.1 c) гарантирует, что тестовая логика может быть сброшена независимо от встроенной системной логики. Это позволяет отключить логику тестирования путем жесткого подключения TRST * к логике 0.

Рекомендация 4.6.1 d) включена для обеспечения предсказуемого реагирования логики тестирования при изменении сигнала, применяемого к TRST*, с 0 на 1. Если нарастающие фронты возникают одновременно при TRST * и TCK, когда к TMS применяется логический 0, произойдет гонка, и контроллер TAP может либо остаться в состоянии контроллера с *логическим сбросом*, либо перейти в состояние контроллера с *тестированием / бездействием*.

4.7 Подключение компонентов, совместимых с настоящим стандартом

4.7.1 Технические характеристики

Разрешения

a) Входные и выходные соединения ОТВОДОВ могут быть соединены между собой на уровне платы способом, соответствующим собранному изделию.

4.7.2 Описание

На рис. 4-1, 4-2 и 4-3 показаны три альтернативных соединения компонентов, соответствующих этому стандарту, на уровне платы.

В каждом примере тестовая шина может управляться либо системой ATE, либо компонентом, который предоставляет интерфейс к тестовой шине на следующем уровне сборки продукта (например, на интерфейсе платы/объединительной платы). В этом стандарте устройство, управляющее тестовой шиной на уровне платы, называется ведущим устройством шины.

Обратите внимание, что минимальная конфигурация (показана на рисунке 4-1) содержит

— Два широковещательных сигнала (TMS и TCK) поступают от ведущего устройства шины тестирования ко всем подчиненным параллельно; и
— Последовательный канал, образованный последовательным соединением выводов последовательных тестовых данных (TDI и TDO).

Рисунок 4-1—Последовательное подключение с использованием одного сигнала TMS

Рисунок 4-2—Соединение в двух параллельных последовательных цепях

Гибридное последовательное/параллельное соединение, показанное на рис. 4-2, использует пару согласованных сигналов TMS (TMS1 и TMS2), чтобы гарантировать, что только один последовательный канал сканирует данные в данный момент времени. В этой конфигурации используется функция трех состояний выходного вывода TDO, которая гарантирует, что только компоненты, которые сканируют данные, имеют TDO в активном состоянии накопителя.

На рисунке 4-3 показаны четыре компонента, соединенные для обеспечения четырех отдельных последовательных каналов во всей конструкции платы. Эти пути имеют отдельные сигналы TDI и TDO, но могут управляться с помощью общих сигналов TCK и TMS.

При выборе конфигурации для соединения компонентов на уровне платы, соответствующей этому стандарту, необходимо учитывать возможности тестового оборудования и генераторов тестовых шаблонов. Полностью ожидается, что любое тестовое оборудование и/ или генераторы тестовых шаблонов, предназначенные для поддержки методологии тестирования, основанной на архитектуре граничного сканирования, определенной этим стандартом, смогут протестировать конфигурацию на уровне платы, показанную на рисунке 4-1, поскольку вырожденная форма этой конфигурации представляет собой единственный соответствующий компонент. С другой стороны, некоторое тестовое оборудование и/или генераторы тестовых шаблонов могут быть не в состоянии протестировать конфигурации на уровне платы, показанные на рис. 4-2 и 4-3.

Рисунок 4-3—Множество независимых каналов с общими сигналами TMS и TCK

4.8 Подчинение настоящего стандарта стратегии тестирования более высокого уровня

Хотя логика тестирования, указанная в этом стандарте, была разработана с возможностью расширения для удовлетворения конкретных потребностей отдельных разработчиков или компаний (например, за счет гибкости регистра команд), могут возникнуть случаи, когда будет желательно временно прекратить соответствие компонента этому стандарту и включить дополнительные функции тестирования. Пример (проиллюстрированный в Приложении A) включает в себя высокочувствительную инфраструктуру проектирования сканирования (LSSD), необходимую для использования во время тестирования “автономных" компонентов, которая не может работать одновременно с функциональностью тестирования, определенной настоящим стандартом (которая требуется для поддержки тестирования плат, на которых будут собраны компоненты, реализующие два метода тестирования).

Этот подпункт определяет, каким образом соответствие настоящему стандарту может быть “включено” или “выключено”. Правила требуют, чтобы изменение функциональности тестирования осуществлялось под контролем сигналов, подаваемых на один или более выводов компонента. Соответствие должно осуществляться с помощью единого логического шаблона, применяемого к этим выводам, а не с помощью последовательности таких шаблонов.

4.8.1 Технические характеристики

Правила

a) Если должен быть спроектирован компонент, содержащий оба
   1) Протестируйте функциональность, соответствующую этому стандарту; и
   2) Другие функциональные возможности тестирования, которые не должны контролироваться с помощью тестовой схемы и средств контроля, определенных в настоящем стандарте,

соответствие настоящему стандарту должно включаться / отключаться одним или несколькими логическими шаблонами устойчивого состояния (называемыми шаблонами “соответствие-включение”), применяемыми к фиксированному набору компонентных входов, которые будут называться “входами, обеспечивающими соответствие”. ……".

ПРИМЕЧАНИЕ —Установившийся комбинационный логический шаблон может быть выбран из набора таких шаблонов “разрешения соответствия”, все из которых имеют эквивалентный эффект [см. Разрешение 4.8.1 h)].

б) Любой из шаблонов, разрешающих соответствие требованиям, при применении к входным данным, разрешающим соответствие требованиям, без учета предыдущих шаблонов на этих входных данных, должен приводить компонент к полному соответствию настоящему стандарту.

в) Как только соответствие настоящему стандарту установлено путем применения шаблона, разрешающего соответствие, на входах, разрешающих соответствие, соответствие настоящему стандарту должно поддерживаться непрерывно до тех пор, пока логический шаблон, применяемый на входах, разрешающих соответствие, не перестанет быть шаблоном, разрешающим соответствие.

Примечания

1—Это правило подразумевает, что переход между шаблонами, обеспечивающими соответствие требованиям, не должен оказывать неблагоприятного воздействия на выполнение. ance. Ограничение количества шаблонов, обеспечивающих соответствие требованиям, - один из способов предотвратить возникновение проблем.

2—Правила, изложенные в других подразделах настоящего стандарта, применяются только в том случае, если включено соответствие. Следовательно, там, где предусмотрены входные данные, соответствующие
разрешению соответствия требованиям, каждому правилу следует предпослать фразу “Когда
включено соответствие этому стандарту ...”, например, Правило 4.1.1 c) следует читать как оговаривающее, что контакты ОТВОДА являются специальными соединениями и
не могут использоваться для каких-либо других целей, пока включено соответствие этому стандарту. Когда соответствие отключено,
ответвительные соединения могут использоваться повторно, например, для обеспечения контроля альтернативного тестового режима работы компонента.
3—в случае включения соответствии с настоящим стандартом путем изменения логики рисунком, нанесенным в соответствии включить
входы компонента нужно не влияет на компонент эквивалентен питания компонента.

г) Входные сигналы, обеспечивающие соответствие требованиям, должны быть выделенными для компонента и не должны использоваться для каких-либо других целей.

Рекомендации

е) Количество входных данных, обеспечивающих соответствие требованиям, предоставляемых компонентом, должно быть сведено к минимуму.

Разрешения

f) Компонент может иметь ноль, один или несколько входных данных, разрешающих соответствие требованиям.

ж) Если компонент с входными данными, разрешающими соответствие требованиям, имеет строку TRST *, включенную в его реализацию TAP , конструкция компонента может потребовать, чтобы вход TRST * был понижен в момент применения шаблона, разрешающего соответствие требованиям, для достижения сброса соответствующей тестовой логики одновременно с работой этой тестовой логики.

h) Компонент может иметь несколько шаблонов обеспечения соответствия, все из которых имеют эквивалентный эффект.

4.8.2 Описание

Если предусмотрены входные сигналы, разрешающие соответствие, должен существовать по крайней мере один логический шаблон, который при применении на входных сигналах, разрешающих соответствие, приведет к тому, что компонент станет полностью совместимым с этим стандартом.

5. Архитектура логики тестирования

Этот пункт определяет структуру верхнего уровня тестовой логики, доступ к которой осуществляется через TAP. Подробный дизайн требования к различным блокам, содержащимся в проекте тестовой логики, содержатся в последующих разделах настоящего стандарта.

5.1 Разработка логики тестирования

5.1.1 Технические характеристики

Правила

а) В архитектуре тестовой логики должны содержаться следующие элементы:
   1) ОТВОД (см. пункт 4);
   2) Контроллер ОТВОДА (см. пункт 6);
   3) Регистр инструкций (см. Раздел 7); и
   4) Группа регистров тестовых данных (см. Раздел 9).

б) Регистры командных и тестовых данных должны представлять собой отдельные каналы на основе сдвиговых регистров, которые подключены параллельно и имеют общий последовательный ввод данных и общий последовательный вывод данных, подключенный к сигналам TDI TAP и TDO соответственно.

в) Выбор между альтернативными путями регистрации инструкций и тестовых данных между TDI и TDO должен производиться под контролем контроллера TAP, как определено в 6.2.

5.1.2 Описание

Концептуальный вид архитектуры тестовой логики верхнего уровня, определенной этим стандартом, показан на рисунке 5-1. Этот рисунок и другие, включенные в описательный материал, содержащийся в настоящем стандарте, являются примерами, предназначенными только для иллюстрации возможного варианта осуществления этого стандарта. *Эти цифры не указывают на предпочтительную реализацию.*

Основные особенности дизайна заключаются в следующем:

a) Контроллер TAP. Он принимает TCK и интерпретирует сигналы на TMS. Контроллер TAP генерирует тактовые или управляющие сигналы или и то, и другое вместе, как требуется для регистров команд и тестовых данных, а также для других частей архитектуры. Спецификация контроллера ОТВОДА содержится в разделе 6.

b) Регистр инструкций. Это позволяет перенести инструкцию в дизайн. Инструкция используется для выбора теста, который будет выполнен, или регистра тестовых данных, к которому будет получен доступ, или и того, и другого. Спецификация регистра команд содержится в разделе 7.

c) Группа регистров тестовых данных. Группа регистров тестовых данных должна включать регистр обхода и регистр пограничного сканирования. Он также может включать дополнительный регистр идентификации устройства и дополнительные дополнительные регистры тестовых данных. Дополнительная информация о структуре группы регистров тестовых данных содержится в разделе 9.

Обратите внимание, что в зависимости от стиля реализации тестовой логики, определенного этим стандартом, в выходном каскаде, показанном на рисунке 5-1, может потребоваться схема для повторного включения сигнала, проходящего через него, для появления на падающем фронте TCK.

5.2 Реализация логики тестирования

5.2.1 Технические характеристики

Правила

а) Контроллер TAP, регистр команд и связанные с ним схемы, необходимые для управления регистрами команд и тестовых данных, должны представлять собой специальную тестовую логику (т.е. Эти блоки тестовой логики не должны выполнять никаких системных функций).

б) Если требуется тестовый доступ к регистру тестовых данных без создания каких-либо помех в работе встроенной системной логики, схема, используемая для построения этого регистра тестовых данных, должна быть специальной для тестирования логики.

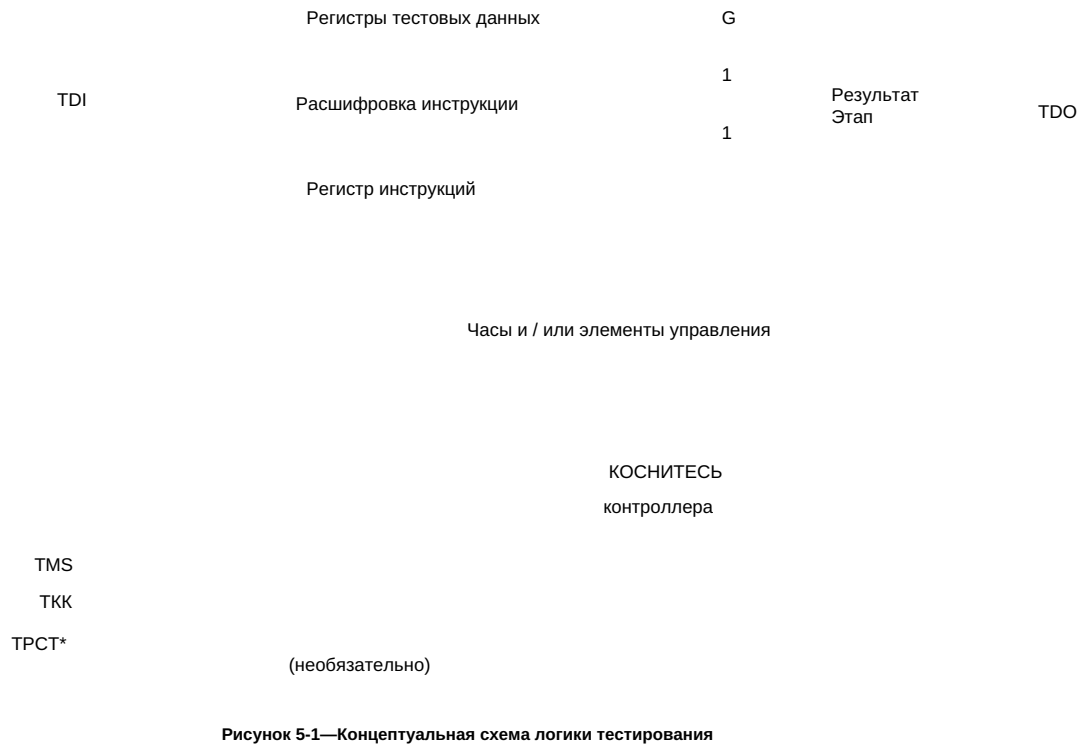


Рисунок 5-1—Концептуальная схема логики тестирования

5.2.2 Описание

Хотя примеры реализаций, содержащиеся в этом стандарте, показывают, что различные регистры тестовых данных являются отдельными физическими объектами, схемы могут совместно использоваться регистрами тестовых данных при условии соблюдения правил, содержащихся в этом стандарте. Например, это позволило бы регистру идентификации устройства и регистру граничного сканирования совместно использовать этапы сдвигового регистра, и в этом случае требования настоящего стандарта удовлетворялись бы за счет работы общей схемы в двух различных режимах - режиме регистра идентификации устройства и режиме регистра граничного сканирования.

6. Контроллер КРАНА

Контроллер TAP представляет собой синхронный конечный автомат, который реагирует на изменения в сигналах TMS и TCK на выводах TAP и управляет последовательностью операций схемы, определенной настоящим стандартом.

6.1 Диаграмма состояния контроллера TAP

6.1.1 Технические характеристики

Правила

a) Диаграмма состояния контроллера отвода должна быть такой, как показано на рисунке 6-1.

ПРИМЕЧАНИЕ — Значение, показанное рядом с каждым переходом состояния на этом рисунке, представляет сигнал, присутствующий в TMS во время нарастающего фронта в TCK.

Рисунок 6-1—диаграмма состояния контроллера TAP

б) Все переходы состояний контроллера TAP должны происходить на основе значения TMS в момент возникновения переднего края TCK.

в) Действия тестовой логики (регистр команд, регистры тестовых данных и т.д.) Должны выполняться либо на восходящем, либо на нисходящем фронте TCK в каждом состоянии контроллера, как показано на рисунке 6-2.

Рисунок 6-2—Хронометраж действий в состоянии контроллера

6.1.2 Описание

Поведение контроллера TAP и другой тестовой логики в каждом из состояний контроллера кратко описано следующим образом. Правила, регулирующие поведение тестовой логики, определенной настоящим стандартом, в каждом состоянии контроллера, содержатся в последующих разделах настоящего стандарта.

Тест-Логика-сброс

Тестовая логика отключена, чтобы нормальная работа встроенной системной логики (т. е. в ответ на стимулы, полученные только через системные контакты) могла продолжаться беспрепятственно. Это достигается путем инициализации регистра инструкции, содержащего команду *IDCODE*, или, если дополнительный регистр идентификации устройства не предусмотрен , команды *ОБХОДА* (см. 7.2). Независимо от исходного состояния контроллера, он перейдет в режим *Test-Logic-Reset*, когда TMS удерживается на высоком уровне по крайней мере в течение пяти возрастающих фронтов TCK. Контроллер остается в этом состоянии, пока TMS высока.

Если контроллер должен выйти из состояния *Test-Logic-Reset* контроллера в результате ошибочного низкого сигнала на линии TMS во время нарастающего фронта на TCK (например, сбой из-за внешних помех), он вернется в состояние*Test-Logic-Reset* после трех нарастающих фронтов TCK с линией TMS на предполагаемом высоком логическом уровне. Работа тестовой логики такова, что в результате такой ошибки не возникает никаких помех в работе системной логики на кристалле..... ation. При выходе из состояния контроллера *Test-Logic-Reset*, контроллер переходит в состояние контроллера *Run-Test/Idle*, в котором никаких действий выполняться не будет, поскольку текущая команда была настроена на выбор операции идентификации устройства или обхода регистра (см. 7.2). Логика тестирования также неактивна в состояниях контроллера *Select-DR-Scan* и *Select-IR-Scan*.

Обратите внимание, что контроллер TAP также будет переведен в состояние *Test-Logic-Reset* контроллера путем применения низкого логического уровня при TRST*, если таковой предусмотрен, или при включении питания (см. 6.3).

Запуск-тестирование/бездействие

Состояние контроллера между операциями сканирования. После ввода контроллер будет оставаться в состоянии *Run-Test / Idle* до тех пор, пока TMS поддерживается на низком уровне. Когда TMS высока и при TCK применяется повышенный фронт, контроллер переходит в состояние *Выбора DR-сканирования*.

В состоянии *Run-Test/Idle* контроллера активность в выбранной тестовой логике происходит только при наличии определенных инструкций . Например, команда *RUNBIST* вызывает выполнение самопроверки системной логики на кристалле в этом состоянии (см. 8.10). Самопроверки, выбранные инструкциями, отличными от *RUNBIST*, также могут быть предназначены для выполнения, пока контроллер находится в этом состоянии.

Для инструкций, которые не вызывают выполнение функций в состоянии *Run-Test/Idle* контроллера, все регистры тестовых данных, выбранные текущей командой, должны сохранять свое предыдущее состояние (т.е. Бездействие).

Инструкция не меняется, пока контроллер крана находится в этом состоянии.

Выбрать-DR-сканирование

Это временное состояние контроллера, в котором все регистры тестовых данных, выбранные текущей командой, сохраняют свое предыдущее состояние.

Если TMS удерживается на низком уровне и к TCK применяется повышающий фронт, когда контроллер находится в этом состоянии, контроллер переходит в состояние *Capture-DR* и инициируется последовательность сканирования для выбранного регистра тестовых данных. Если TMS удерживается на высоком уровне и к TCK применяется повышенный фронт, контроллер переходит в состояние *Выбора ИК-сканирования*.

Инструкция не меняется, пока контроллер крана находится в этом состоянии.

Выбор-ИК-сканирование

Это временное состояние контроллера, в котором все регистры тестовых данных, выбранные текущей командой, сохраняют свое предыдущее состояние.

Если TMS удерживается на низком уровне и к TCK применяется повышающий фронт, когда контроллер находится в этом состоянии, то контроллер переходит в состояние *Capture-IR* и инициируется последовательность сканирования регистра команд. Если TMS удерживается на высоком уровне и к TCK применяется повышающий фронт, контроллер возвращается в состояние *сброса логики тестирования*.

Инструкция не меняется, пока контроллер крана находится в этом состоянии.

Захват-DR

В этом состоянии данные контроллера могут быть параллельно загружены в регистры тестовых данных, выбранные текущей командой на переднем фронте TCK. Если регистр тестовых данных, выбранный текущей командой, не имеет параллельного ввода, или если запись не требуется для выбранного теста, регистр сохраняет свое предыдущее состояние без изменений.

Инструкция не меняется, пока контроллер крана находится в этом состоянии.

Когда контроллер TAP находится в этом состоянии и к TCK применяется повышающий фронт, контроллер переходит либо в состояние *Exit1-DR*, если TMS удерживается на 1, либо в состояние *Shift-DR*, если TMS удерживается на 0.

Смена-DR

В этом состоянии контроллера регистр тестовых данных, подключенный между TDI и TDO в результате текущей команды, сдвигает данные на один этап к своему последовательному выходу на каждом восходящем фронте TCK. Регистры тестовых данных, которые выбраны текущей командой, но не помещены в последовательный путь, сохраняют свое предыдущее состояние без изменений.

Инструкция не меняется, пока контроллер крана находится в этом состоянии.

Когда контроллер отвода находится в этом состоянии и к TCK применяется повышающий фронт, контроллер либо переходит в состояние *Exit1-DR*, если TMS удерживается на 1, либо остается в состоянии *Shift-DR*, если TMS удерживается на 0.

Выход1-DR

Это временное состояние контроллера. Если TMS поддерживается на высоком уровне, повышающий фронт, применяемый к TCK в этом состоянии, заставляет контроллер перейти в состояние *Update-DR*, которое завершает процесс сканирования. Если TMS удерживается на низком уровне, а к TCK применяется повышающий фронт, контроллер переходит в состояние *Пауза-DR*.

Все регистры тестовых данных, выбранные текущей инструкцией, сохраняют свое предыдущее состояние без изменений.

Инструкция не меняется, пока контроллер крана находится в этом состоянии.

Пауза-DR

Это состояние контроллера позволяет временно приостановить сдвиг регистра тестовых данных в последовательном канале между TDI и TDO. Все регистры тестовых данных, выбранные текущей инструкцией, сохраняют свое предыдущее состояние без изменений.

Контроллер остается в этом состоянии, пока TMS находится на низком уровне. Когда TMS становится высокой и к TCK применяется повышающий фронт, контроллер переходит в состояние *Exit2-DR*.

Инструкция не меняется, пока контроллер крана находится в этом состоянии.

Exit2-DR

Это временное состояние контроллера. Если TMS удерживается на высоком уровне и к TCK применяется повышающий фронт, находясь в этом состоянии, процесс сканирования завершается и контроллер TAP переходит в состояние контроллера *Update-DR*. Если TMS удерживается на низком уровне, а к TCK применяется повышающий фронт, контроллер переходит в состояние *Shift-DR*.

Все регистры тестовых данных, выбранные текущей инструкцией, сохраняют свое предыдущее состояние без изменений.

Инструкция не меняется, пока контроллер крана находится в этом состоянии.

Обновление-DR

Некоторые регистры тестовых данных могут быть снабжены заблокированным параллельным выходом для предотвращения изменений на параллельном выходе, когда данные сдвигаются в соответствующем пути регистра сдвига в ответ на определенные инструкции (например, *EXTEST*, *INTEST* и *RUNBIST*). Данные фиксируются на параллельном выходе этих регистров тестовых данных из пути сдвига регистра на падающем фронте TCK в состоянии контроллера *Update-DR*. Данные, хранящиеся на фиксированном paral-аллельном выходе, не должны изменяться иначе, чем в этом состоянии контроллера, если только не требуется операция во время выполнения самопроверки (например, во время *состояния запуска-тестирования / бездействия* контроллера в ответ на общедоступную инструкцию, специфичную для конкретной конструкции).

Все ступени сдвигового регистра в регистрах тестовых данных, выбранные текущей инструкцией, сохраняют свое предыдущее состояние без изменений.

Инструкция не меняется, пока контроллер крана находится в этом состоянии.

Когда контроллер TAP находится в этом состоянии и к TCK применяется повышенный фронт, контроллер переходит либо в состояние *Выбора DR-сканирования*, если TMS удерживается на 1, либо в состояние *запуска / бездействия*, если TMS удерживается на 0.

Захват-ИК

В этом состоянии контроллера сдвиговый регистр, содержащийся в регистре команд, загружает набор фиксированных логических значений ues на восходящем фронте TCK. Кроме того, в каскады сдвигового регистра могут быть загружены данные для конкретной конструкции, для которых не требуется устанавливать фиксированные значения (см. Раздел 7).

Регистры тестовых данных, выбранные текущей инструкцией, сохраняют свое предыдущее состояние. Инструкция не изменяется, пока контроллер крана находится в этом состоянии.

Когда контроллер TAP находится в этом состоянии и к TCK применяется повышающий фронт, контроллер переходит либо в состояние *Exit1-IR*, если TMS удерживается на 1, либо в состояние *Shift-IR*, если TMS удерживается на 0.

Сдвиг-ИК

В этом состоянии контроллера регистр сдвига, содержащийся в регистре команд, подключен между TDI и TDO и сдвигает данные на один этап в направлении их последовательного вывода на каждом восходящем фронте TCK.

Регистры тестовых данных, выбранные текущей инструкцией, сохраняют свое предыдущее состояние. Инструкция не изменяется, пока контроллер крана находится в этом состоянии.

Когда контроллер TAP находится в этом состоянии и к TCK применяется повышающий фронт, контроллер либо переходит в состояние *Exit1-IR*, если TMS удерживается на 1, либо остается в состоянии *Shift-IR*, если TMS удерживается на 0.

Выход1-IR

Это временное состояние контроллера. Если TMS поддерживается на высоком уровне, повышающий фронт, применяемый к TCK в этом состоянии, заставляет контроллер перейти в состояние *Update-IR*, которое завершает процесс сканирования. Если TMS удерживается на низком уровне и к TCK применяется повышающий фронт, контроллер переходит в состояние Pause-IR.

Регистры тестовых данных, выбранные текущей командой, сохраняют свое предыдущее состояние. Команда не изменяется, пока контроллер TAP находится в этом состоянии, и регистр команд сохраняет свое состояние.

Пауза-ИК

Это состояние контроллера позволяет временно приостановить сдвиг регистра команд.

Регистры тестовых данных, выбранные текущей инструкцией, сохраняют свое предыдущее состояние. Команда не изменяется, пока контроллер отвода находится в этом состоянии, и регистр команд сохраняет свое состояние.

Контроллер остается в этом состоянии, пока TMS находится на низком уровне. Когда TMS становится высокой и к TCK применяется повышающий фронт, контроллер переходит в состояние *Exit2-IR*.

Выход2-IR

Это временное состояние контроллера. Если TMS удерживается на высоком уровне и к TCK применяется повышенный фронт, находясь в этом состоянии, процесс сканирования завершается, и контроллер TAP переходит в состояние *Обновление-IR* контроллера . Если TMS удерживается на низком уровне, а к TCK применяется повышающий фронт, контроллер переходит в состояние *Shift-IR*.

Регистры тестовых данных, выбранные текущей инструкцией, сохраняют свое предыдущее состояние. Команда не изменяется, пока контроллер отвода находится в этом состоянии, и регистр команд сохраняет свое состояние.

Обновление-IR

Команда, сдвинутая в регистр команд, фиксируется на параллельном выходе из регистра сдвига пути на падающем фронте TCK в этом состоянии контроллера. Как только новая команда зафиксирована, она становится текущей инструкцией.

Регистры тестовых данных, выбранные текущей инструкцией, сохраняют свое предыдущее состояние.

Когда контроллер TAP находится в этом состоянии и к TCK применяется повышенный фронт, контроллер переходит в состояние *Выбора DR-сканирования*, если TMS удерживается на 1, или в состояние *запуска / бездействия*, если TMS удерживается на 0.

Состояния контроллера *Pause-DR* и *Pause-IR* включены для того, чтобы можно было временно приостановить передачу данных через регистр тестовых данных или команд. Например, это может быть необходимо для того, чтобы позволить системе ATE перезагрузить свою память pin-кодов с диска во время применения длинной тестовой последовательности. Тест с граничным сканированием последовательности, вероятно, будут составлять порядка 10^7 тестовые шаблоны для сложных конструкций плат.

Состояния контроллера TAP включают три основных действия, необходимых для тестирования: приложение-стимул (*Обновление-DR*), выполнение (*Run-Test/Idle*) и получение ответа (*Capture-DR*). Однако не все эти действия требуются для каждого типа теста. В таблице 6-1 перечислены действия, необходимые для ключевых типов тестов, поддерживаемых этим стандартом.

Для тестирования сканирования стимул становится доступным для использования в конце переключения или, если включена блокировка параллельного вывода, путем обновления параллельного вывода в состоянии контроллера *Update-DR*. Результаты теста фиксируются в регистре тестовых данных во время состояния контроллера *Capture-DR*.

Таблица 6-1—Использование состояний контроллера для различных типов тестов

Тип теста	Действие, требуемое в этом состоянии контроллера		
	Обновление-DR	***Запуск-тестирование/бездействие***	***Захват-DR***
Внешний тест пограничного сканирования (например, *EXTEST*)	ДА	НЕТ	ДА
Проверка внутреннего сканирования с помощью пограничного сканирования (например, *INTEST*)	Может быть	Может быть	ДА
Предварительная нагрузка при сканировании границ	ДА	Нет	Может быть
ОБРАЗЕЦ для сканирования границ	Возможно	НЕТ	ДА
Встроенные тесты с внутренним управлением (например, *RUNBIST*)	НЕТ	ДА	Нет
Проверка внутренним сканированием (т. Е. Определенная пользователем общедоступная инструкция)	Может быть	Нет Да	

Для схем с самотестированием с внутренним управлением начальные значения регистров доступны в конце переключения: нет фиксатора параллельного выхода для обновления. Регистры должны работать под контролем внутренней логики тестирования во время *выполнения-тестирования / простоя*. Поскольку результат уже содержится в регистре тестовых данных, никаких действий во время состояния контроллера *Capture-DR*не требуется.

Для теста внутреннего сканирования целевой регистр состоит из последовательной конкатенации элементов памяти, которые поддерживают нормальную системную работу компонента. Построение такого регистра выходит за рамки настоящего стандарта, и в данной реализации могут присутствовать, а могут и не присутствовать параллельные выходные защелки.

6.2 Управление контроллером КРАНА

6.2.1 Технические характеристики

Правила

а) Контроллер отвода должен изменять состояние только в ответ на следующие события:
   1) Растущее преимущество TCK;
   2) Переход к логическому 0 на входе TRST* (если предусмотрено); или
   3) Включение питания.

б) Контроллер TAP должен генерировать сигналы для управления работой регистров тестовых данных, командных регистров и связанных с ними схем, как определено в настоящем стандарте (рис. 6-3 и 6-4).

ПРИМЕЧАНИЕ —На этих рисунках сделано предположение, что сигналы, подаваемые на TMS и TDI, изменяют состояние на падающем фронте TCK. Время, в которое эти сигналы меняют состояние, настоящим стандартом не определено, но оно должно быть таким, чтобы соблюдались требования TMS и TDI к настройке и удержанию. Далее предполагается, что конструкция включает в себя дополнительное устройство идентификационный регистр. Следовательно, на рисунках показана команда *IDCODE*, устанавливаемая на выходе регистра команд в состоянии *контроллера Test-Logic-Reset*. Если регистр идентификации устройства не включен в конструкцию, выход ввода регистра команд будет установлен на команду *ОБХОДА* в состоянии контроллера *Тест-логический сброс*.

с) Выходной буфер TDO и схема, которая выбирает выходной сигнал регистра, подаваемый в TDO, должны управляться так, как показано в таблице 6-2.

г) Изменения в TDO, определенные в таблице 6-2, должны происходить на падающем крае TCK после перехода в состояние.

Рисунок

Таблица 6-2—Проверка работы логики в каждом состоянии контроллера

Состояние контроллера	Выбранная регистрация для управления TDO	Драйвер TDO
Тест-Логика-Сброс	Не определено	Неактивен
Выполнить-тест/ бездействие	Не определено	Неактивен
Выбрать-DR-Сканирование	Не определено	Неактивен
Выбрать-ИК-сканирование	Не определено	Неактивен
Инфракрасный захват	Не определено	Неактивен
Shift-IR	Инструкция	Активный
Выход1-IR	Не определено	Неактивен
Пауза-IR	Не определено	Неактивен
Exit2-IR	Не определено	Неактивен
Обновление-IR	Не определено	Неактивен
Захват-DR	Не определено	Неактивен
Смена-DR	Тестовые данные	Активен
Exit1-DR	Не определено	Неактивен
Пауза-DR	Не определено	Неактивен
Exit2-DR	Не определено	Неактивен
Обновление-DR	Не определено	Неактивен

ПРИМЕЧАНИЕ—Некоторые компоненты, разработанные до публикации настоящего стандарта, могут соответствовать требованиям во всех отношениях, за исключением того, что они имеют TDO, активный в состояниях контроллера *Capture-IR*, *Pause-IR*, *Exit1-IR*, *Exit2-IR*, *Capture-DR*, *Pause-DR*, *Exit1-DR* и *Exit2-DR*, в дополнение к состояниям контроллеров Shift-IR *и* DR *Shift*. Функциональность этих компонентов неотличима от функциональности компонентов, полностью соответствующих этому стандарту, за исключением случаев, когда выход TDO такого компонента подключен к выходу TDO другого компонента (например, как показано на рисунке 4-2).

6.2.2 Описание

Пример схемы, отвечающей требованиям пункта 6.2.1, показан на рис. 6-5 и 6-6. Эта схема генерирует диапазон тактовых и управляющих сигналов, необходимых не только для управления выбором между альтернативными путями команд и регистрами тестовых данных и активностью TDO (как определено в таблице 6-2), но также для управления примерами реализаций других элементов тестовой логики, которые содержатся в этом стандарте.

Назначение состояний контроллера в примере реализации приведено в таблице 6-3.

Логические уравнения для логики следующего состояния на рис. 6-5 и 6-6 следующие:

```
ND: = DC * + DB + T * CB * + D * CB * A*
NC: = CB * + CA + TB *
NB: = T * BA * + T * C * + T* D * B + T * D * A * + TCB * + TDCA
NA : = T * C * A + TB * + TA * + TDC
```

где

```
T = значение, присутствующее в TMS
```

ПРИМЕЧАНИЕ — Схема на рисунке 6-5 генерирует различные управляющие сигналы, используемые примерами схем, проиллюстрированных в других местах в этом стандарте. Обратите внимание, что сигнал Select будет использоваться для управления мультиплексором, показанным на рис. 5-1, а сигнал Enable будет использоваться для управления выходом TDO в трех состояниях. Обратите также внимание, что, хотя сигналы ShiftDR и ClockDR могут транслироваться во все регистры тестовых данных, распределением сигнала управления UpdateDR будет управлять согласно инструкции, хранящейся в регистре команд, таким образом, что сигнал подается только в регистр тестовых данных, который выбран в качестве последовательного пути между TDI и TDO.

Рисунок 6-5 —Реализация контроллера отвода—регистры состояния и логика вывода

электронная логика

**ТАТ
ы
Т
х**

—не

**о
ти
а**

ролик осуществления

**П прод
А**

6—А Т

е 6-

Figur

наполнитель

тро

А

достаточный объем

например
h

ион t

крыса

7—Опе

re 6-
U

ig
F

На рисунке 6-7 показана работа этой реализации контроллера с помощью команд и циклов сканирования регистра тестовых данных
.

Таблица 6-3—Назначения состояний для примера контроллера КРАНА

Состояние контроллера	DCBA (шестнадцатеричный)
Выход2-DR	0
Выход1-DR	1
Смена-DR	2
Пауза-DR	3
Выбор-ИК-сканирование	4
Обновление-DR	5
Захват-DR	6
Выбрать-DR-сканирование	7
Выход2-IR	8
Выход1-IR	9
Сдвиг-ИК	A
Пауза-ИК	B
Запуск-тестирование/бездействие	C
Обновление-IR	D
Захват-ИК	E
Тест-Логика-сброс	

6.3 Инициализация контроллера КРАНА

6.3.1 Технические характеристики

Правила

а) Контроллер отвода должен быть принудительно включен в *Тест-логический сброс* состояние контроллера при включении питания определяется либо использованием сигнала TRST*, либо схемой, встроенной в логику тестирования.

ПРИМЕЧАНИЕ—Если контроллер ОТВОДА должен быть сброшен при включении питания с помощью TRST *, конструкция собранной системы должна гарантировать, что логическое значение 0 применяется к TRST * при подаче питания. Аналогично, если контроллер TAP должен быть сброшен с помощью TRST * после включения соответствия этому стандарту, как описано в разделе 4.8, конструкция собранной системы должна гарантировать, что логика 0 применяется к TRST *, когда включено соответствие.

б) Контроллер TAP не должен инициализироваться операцией какого-либо системного ввода, такой как системный сброс.

в) Если предусмотрен специальный вывод сброса (TRST*), позволяющий инициализировать контроллер TAP, инициализация должна происходить асинхронно, когда вход TRST * изменяется на низкий логический уровень.

г) Если контроллер TAP инициализируется при включении питания с помощью схемы, встроенной в логику тестирования, результат должен быть эквивалентен тому, который был бы достигнут путем применения логического значения 0 к входу TRST* .

6.3.2 Описание

В конструкции платы, содержащей проводные соединения или шины, должно быть предусмотрено, чтобы при включении питания любой период разногласий между водителями в шине сохранялся в пределах, которые гарантируют отсутствие повреждений компонентов платы.

Когда схема граничного сканирования вставляется между встроенной системной логикой и выводами пакета, становится важно убедиться, что вскоре после включения питания эта схема переходит в состояние, в котором шины и проводные соединения управляются системной схемой, т.е. состояние контроллера *тестовой логики-сброса*.

ПРИМЕЧАНИЕ—Раздел 11 содержит правила, гарантирующие, что схема граничного сканирования на системных выводах не мешает нормальной работе системы, когда выбрано состояние контроллера *Test-Logic-Reset.*

В то время как контроллер TAP синхронно перейдет в состояние *Тестовой логики-сброса* контроллера после пяти нарастающих фронтов при TCK (при условии, что TMS поддерживается высокой), в наихудшем случае время, необходимое для достижения этого состояния, может превысить то, при котором может произойти повреждение. Кроме того, нельзя гарантировать, что часы будут работать в то время, в которое питание подается на плату. Следовательно, включено требование “сброса при включении питания”.

Это требование может быть выполнено различными способами, например, путем включения сброса включения питания в интегральную схему или за счет асимметричной конструкции защелок или регистров, используемых для построения контроллера отвода. Это также может быть достигнуто путем включения специального вывода TRST * для контроллера TAP. Однако системный сброс также нельзя использовать для инициализации контроллера TAP, поскольку это поставило бы под угрозу возможность тестирования системных межсоединений на уровне платы с использованием схемы граничного сканирования. В некоторых системах это также может случиться, - BLE, чтобы использовать независимости и испытание системы сбрасывается до взятия проб и проверки данных после сбоя системы. Это потребовало бы, чтобы тестовая логика была сброшена перед повторной инициализацией встроенной системной логики.

Если в компоненте предусмотрена функция сброса при включении питания, ее можно использовать для инициализации как системы, так и логики тестирования, например, как показано на рис. 6-8.

Рисунок 6-8—Использование сброса при включении питания для системы и логики тестирования

7. Регистр команд

Регистр команд позволяет включить команду в проект. Инструкция используется для выбора
теста, который будет выполнен, или регистра тестовых данных, к которому будет получен доступ, или и того, и другого. Как будет обсуждаться в разделе 8, настоящим стандартом определен ряд обязательных и необязательных инструкций. Далее конструкция конкретной обучения может быть добавлен, чтобы позволить функциональность тест на логику, встроенный в компонент, чтобы быть продлен.

Необязательно, регистр команд позволяет проверять информацию, относящуюся к конкретной конструкции, сгенерированную внутри
компонента.

Этот раздел содержит требования к дизайну регистра команд.

7.1 Проектирование и построение регистра инструкций

Регистр команд представляет собой конструкцию на основе сдвигового регистра, которая имеет дополнительный параллельный вход для ячеек регистра, отличных от двух ближайших к последовательному выходу. Инструкция сдвигается в регистре фиксируется на ком-
завершение переключения передач процесса.

7.1.1 Технические характеристики

Правила

а) Регистр команд должен включать по меньшей мере две ячейки на основе сдвигового регистра, способные хранить данные команд.

б) Команда, перемещенная в регистр команд, должна быть зафиксирована таким образом, чтобы изменения в действии команды происходили только в состояниях контроллера *Обновление-IR* и *Логический сброс-тестирования* (см. 7.2).

с) Не должно быть инверсии данных между последовательным входом и последовательным выходом регистра команд.

г) Две ячейки регистра команд с наименьшим значением (т. е. Те, которые расположены ближе всего к последовательному выходу) должны загружать фиксированный двоичный шаблон “01” (1 в ячейке младшего значащего бита) в состоянии *контроллера* с ИК-захватом (см. 7.2).

Рекомендации

е) Где параллельные входы регистре клетки не являются необходимыми для загрузки конструкция конкретной информация, то эти клетки должны быть рассчитаны на нагрузки фиксированных логических значений (0 или 1) в *захват-ИК* контроллер государства.

Разрешения

е) В ячейки регистра команд (отличные от двух наименее значимых ячеек) могут быть предоставлены параллельные входные данные для обеспечения возможности сбора информации, относящейся к конкретной конструкции, в состоянии контроллера *Capture-IR*.

7.1.2 Описание

Параллельный вывод из регистра команд заблокирован, чтобы гарантировать, что логика тестирования защищена от
шаблонов переходных данных, которые будут возникать на этапах работы с регистром сдвига по мере ввода новых данных команды.
Заблокированный параллельный выход управляется таким образом, что он может изменять состояние только в состояниях контроллера *Update-IR* и *Test-Logic-Reset*
. Сроки и характер этих изменений подробно обсуждаются в разделе 7.2.

Минимальный размер (две ячейки регистра команд) необходим для соответствия правилам, изложенным в других разделах настоящего стандарта:

а) Регистр команд должен позволять выбирать регистр обхода.

б) Регистр команд должен обеспечивать доступ к регистру граничного сканирования по крайней мере в трех конфигурациях (*EXTEST*, *PRELOAD* и *SAMPLE* - см. 8.2).

Допустимо [см. Разрешение 8.1.1 g)], чтобы инструкции совместно использовали двоичные коды при условии, что ни одно из правил, определяющих объединенные инструкции, не нарушено. В более ранних выпусках этого стандарта SAMPLE и ПРЕДВАРИТЕЛЬНАЯ ЗАГРУЗКА были определены как объединенная инструкция —*SAMPLE /PRELOAD* — таким образом, что четыре инструкции, предусмотренные этим стандартом — *BYPASS*, *EXTEST*, *PRELOAD* и *SAMPLE* — могли быть реализованы с использованием трех двоичных кодов, оставляя достижимой четвертую с минимальным 2-разрядным регистром команд, доступным для реализации дополнительной инструкции.

Кроме того, должна поддерживаться изоляция от неисправностей канала передачи данных последовательного тестирования на уровне платы. Это достигается за счет загрузки постоянного двоичного шаблона “01” в младшие значащие биты регистра команд в начале цикла сканирования команд.

Включение дополнительных входных данных, зависящих от конструкции, в регистр команд позволяет проверять сигналы ключевых данных внутри устройства в начале тестирования, при этом будущие тестовые действия потенциально зависят от собранной информации, зависящей от конструкции. Если параллельные входные данные в ячейки регистра команд не используются для специфичной для проекта информации, рекомендуется, чтобы эти ячейки были спроектированы так, чтобы загружать фиксированное логическое значение (0 или 1) во время состояния контроллера *Capture-IR*.

7.2 Работа с регистром инструкций

7.2.1 Технические характеристики

Правила

а) Поведение регистра команд в каждом состоянии контроллера отвода должно быть таким, как определено в таблице 7-1.

б) Все действия, являющиеся результатом команды, должны завершаться, когда другая команда передается на параллельный вывод регистра команд (т.е. В состояниях *контроллера обновления-IR* или логического сброса-тестирования).

с) Все операции ступеней сдвигового регистра должны выполняться на переднем фронте TCK после перехода в состояние контроллера.

Таблица 7-1—Работа регистра команд в каждом состоянии контроллера

Состояние контроллера	Каскад сдвигового регистра	Параллельный вывод
Тест-Логика-Сброс	Не определено	Устанавливается для передачи инструкции по *IDCODE* (или *ОБХОДУ*)
Захват-ИК	Загрузите 01 в LSBS и, по желанию , данные для конкретной конструкции или фиксированные значения в MSBS	Сохранять последнее состояние
Сдвиг-IR	Переход к последовательному выходу	Сохранять последнее состояние
Выход1-ИК *Выход2-ИК* *Пауза-ИК*	Сохранить последнее состояние	Сохранить последнее состояние
Обновить-IR	Сохранить последнее состояние	Загрузка из сдвигового регистра
Все остальные состояния	Не определено	Сохранить последнее состояние

г) Данные, присутствующие на параллельном выходе регистра команд, должны быть зафиксированы с каскада регистрации сдвига на падающем фронте TCK в состоянии контроллера *Update-IR*.

е) После перехода в состояние *тестовой логики-сброса* контроллера в результате синхронизированной работы TAP контроллера, команда *IDCODE* (или, если нет регистра идентификации устройства, команда *BYPASS*) должна быть зафиксирована на выходе регистра команд на падающем фронте TCK.

е) Если предусмотрен вход TRST*, то при подаче на вход низкого сигнала команда с блокировкой должна асинхронно измениться на *идентификационный КОД* (или, если регистр идентификации устройства не предусмотрен, на *БАЙПАС*).

7.2.2 Описание

На рисунке 7-1 показана реализация ячейки регистра команд, которая удовлетворяет этим требованиям и работает в ответ на сигналы, генерируемые примером конструкции контроллера TAP, содержащимся в разделе 6.2:

а) Параллельный выходной сигнал (помеченный бит инструкции) обновляется в конце цикла сканирования инструкций во время состояния *контроллера Update-IR*. Это должно происходить на падающем фронте TCK, потому что изменение в запертой инструкции может привести к изменению на выводах системного вывода из-за работы регистра граничного сканирования. Такие изменения должны происходить на понижающейся границе TCK, как определено в пункте 11. Обратите внимание, что на рисунке 7-1 триггер, запускаемый по фронту, предусмотрен рядом с каскадом сдвигового регистра для удовлетворения этого требования. Допустимы альтернативные реализации, например, когда используется защелка, управляемая уровнем, или элемент памяти следует (а не предшествует) логике декодирования команд.

ПРИМЕЧАНИЕ — Триггер параллельного вывода на этом рисунке снабжен входом сброса. Чтобы соответствовать Правилам 7.2.1 е) и 7.2.1 f) , некоторые или все ячейки регистра команд потребуют использования ввода set, а не reset.

Рисунок 7-1 —Ячейка регистра команд

б) Тактовый ввод в регистр по последовательному каналу применяется только во время состояний *Capture-IR* и *Shift-IR* контроллера.

в) Параллельный выход сбрасывается в состояние *Тестовой логики-Reset* контроллера в результате логического 0, принятого на входе Reset* ячейки. Обращаясь к рисункам 6-5 и 6-6, обратите внимание, что низкий сигнал сброса* будет сгенерирован на падающем фронте TCK после входа в *состояние* контроллера "Тестовая логика-сброс" под управлением TMS и TCK (значение TRST* удерживается на 1). Параллельный вывод регистра команд изменится на падающем фронте TCK, как это имеет место в состоянии контроллера *Update-IR*. Напротив, когда к TRST* применяется логическое значение 0, следовательно, утверждается Сброс * (низкий уровень) (см. Рисунок 6-5), и изменение на параллельном выходе происходит немедленно, независимо от состояния TMS или TCK. Обратите внимание, что некоторые ячейки необходимо будет спроектировать таким образом, чтобы параллельный вывод был установлен высоким во время этого состояния контроллера, чтобы значение команды *IDCODE* (или *BYPASS*) загружалось на выходы всего регистра, как того требует правило 7.2.1 е).

г) Применение значения 0 при TRST * приводит к тому, что параллельный вывод асинхронно устанавливается на низкий уровень. Опять же, некоторые ячейки, возможно, потребуется спроектировать так, чтобы TRST * устанавливал значение high таким образом, чтобы значение команды *IDCODE* (или *ОБХОДА*) принудительно передавалось на выходы регистра.

Обратите внимание, что параллельные входные данные для двух наименее значимых этапов (этапы 0 и 1 регистра команд) должны быть привязаны к фиксированным логическим уровням (1 для младшего по значению бита, 0 для следующего по значению бита).

Правило 7.2.1 b) гарантирует, что работа регистров тестовых данных и т.д. Определяется только текущей инструкцией и что нет никакой возможности, что действия, вытекающие из какой-либо инструкции (например, выполнение внутренней самопроверки), могут продолжаться после удаления инструкции. Тестируемая схема может находиться в неизвестном состоянии, если новая команда загружена до завершения выполнения предыдущей.

8. Инструкции

Регистр команд позволяет последовательно вводить команды в логику тестирования во время цикла сканирования регистра команд . Этот раздел определяет минимальный набор инструкций, которые должны быть предоставлены, и операции, которые выполняются в ответ на эти инструкции. Также определены дополнительные инструкции и результирующая операция логики тестирования, а также требования к расширениям набора инструкций, определенным в этом стандарте.

8.1 Реакция тестовой логики на инструкции

8.1.1 Технические характеристики

Правила

а) Каждая команда должна полностью определять набор регистров тестовых данных, которые могут работать и (где требуется) взаимодействовать с системной логикой на кристалле, пока команда является текущей.

b) Регистрами тестовых данных, которые не выбраны текущей командой, следует управлять таким образом, чтобы они не мешали работе встроенной системной логики или выбранных регистров тестовых данных.

в) Каждая команда должна приводить к включению единого последовательного канала регистрации тестовых данных для переключения данных между TDI и TDO в состоянии контроллера *Shift-DR* (как определено в таблице 6-2).

d) Двоичные коды команд, которые иным образом не требуются для обеспечения управления логикой тестирования, должны быть эквивалентны команде *ОБХОДА* (см. 8.4).

Рекомендации

е) Следует избегать использования двоичного кода {000...0} для инструкций, которые нарушают нормальную (т.е. нетестовую) работу компонента.

ПРИМЕЧАНИЕ—Более ранние версии этого стандарта требовали, чтобы двоичный код для *EXTEST* команды был равен {000...0} (т.е. логический 0 загружается в каждую ячейку регистра команд). Хотя использование этого двоичного кода больше не является обязательным и не запрещено, следует отметить, что такое использование может нанести ущерб внедрению систем с высокой надежностью, поскольку очевидное зависание на нулевом выводе TDI компонента может привести к неожиданному выбору *EXTEST* и последующему отключению компонента от нормальной эксплуатации.

Разрешения

е) Режим работы регистра тестовых данных может определяться комбинацией текущей инструкции и дополнительной управляющей информации, содержащейся в регистрах тестовых данных.

ж) Две или более инструкций могут совместно использовать единый двоичный код при условии соблюдения всех правил для отдельных инструкций.

8.1.2 Описание

Инструкции, загруженные в регистр команд, декодируются для выполнения двух ключевых функций.

Во-первых, каждая команда определяет набор регистров тестовых данных, которые могут работать, пока команда является текущей. Другими регистрами тестовых данных следует управлять таким образом, чтобы они не могли вмешиваться в работу встроенной системной логики или в работу выбранных регистров тестовых данных. Несколько регистров могут быть переведены в тестовые режимы одновременно (пример см. в разделе 9.2).

Во-вторых, инструкция определяет путь к регистру последовательных тестовых данных, который используется для переключения данных между TDI и TDO во время сканирования регистра данных. Обратите внимание, что конкретная команда может привести к подключению одного регистра тестовых данных между TDI и TDO или к последовательному подключению нескольких регистров тестовых данных между TDI и TDO (пример см. 9.2).

Правило 8.1.1 d) гарантирует, что каждый набор из единиц и нулей, которые могут быть введены в регистр команд, выдает определенный ответ и, в частности, что регистр тестовых данных подключен между TDI и TDO для каждого возможного двоичного кода команды.

Разрешение 8.1.1 g) позволяет объединять инструкции для работы в рамках единого двоичного кода, когда их соответствующие модели поведения не являются взаимоисключающими. Ярким примером такого объединения было бы совместное использование единственного двоичного кода между *SAMPLE* и *PRELOAD*. Результирующее объединенное поведение, которое можно было бы назвать *ВЫБОРКОЙ / ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКОЙ*, было бы полностью эквивалентно тому, которое предписывалось в более ранних выпусках этого стандарта.

8.2 Публичные инструкции

8.2.1 Технические характеристики

Правила

а) Общедоступные инструкции должны быть доступны для использования покупателями компонента.

б) Следующие общедоступные инструкции должны быть предоставлены для всех компонентов, заявляющих о соответствии настоящему стандарту: *БАЙПАС*, *ПРОБА*, *ПРЕДВАРИТЕЛЬНАЯ НАГРУЗКА* и *ПРЕДЕЛ* (см. 8.4, 8.6, 8.7 и 8.8 соответственно).

с) Если дополнительный идентификационный регистр устройства включен в компонент, должна быть предоставлена инструкция по *идентификационному КОДУ*.

г) Если дополнительное устройство идентификационный регистр включен в программируемых компонентов, которые никак не позволяют программировать через испытание логикой, определенной настоящим стандартом, то *USERCODE* обучения должны быть предусмотрены.

е) Двоичные коды для команды *ОБХОДА* должны соответствовать определению в 8.4.

Рекомендации

f) Рекомендуется, чтобы продукты поддерживали либо инструкцию *INTEST*, либо инструкцию *RUNBIST*, либо и то, и другое (см. 8.9 и 8.10).

Разрешения

g) Конструкция может содержать общедоступные инструкции в дополнение к тем, которые определены в настоящем стандарте, чтобы предоставить устройству доступ покупателя к функциям, специфичным для конструкции.

з) Если двоичные коды для общедоступных инструкций не определены настоящим стандартом, они могут быть назначены так, как требуется для конкретной конструкции.

8.2.2 Описание

Общедоступные инструкции предоставляют покупателю компонентов доступ к функциям тестирования, которые помогают в тестовых задачах, например, тестирование компонента в режиме ожидания с помощью его самопроверки и тестирование межсоединения платы с помощью регистра граничного сканирования. Покупатель ожидает, что результаты таких тестов не будут зависеть от варианта компонента, установленного на конкретной плате, источника компонента и т.д. Исключением, конечно, являются случаи, когда результаты теста предназначены для различения варианта и т.д., Как это было бы в случае, если бы *использовалась инструкция*IDCODE (см. 8.13).

Двоичный код команды - это последовательность битов данных, последовательно сдвигаемых в регистр команд из TDI во время состояния контроллера *Shift-IR*.

8.3 Личные инструкции

8.3.1 Технические характеристики

Разрешения

а) Общедоступные инструкции могут быть дополнены частными инструкциями, предназначенными исключительно для использования производителем компонентов.
б) Выполнение частных инструкций не обязательно документировать.
в) Если в компоненте используются частные инструкции, поставщик должен четко идентифицировать любые инструкции двоичные коды, выбор которых может привести к опасной работе компонента.

8.3.2 Описание

Частные инструкции позволяют производителю компонентов использовать логику TAP и test для получения доступа к тестовым функциям, встроенным в конструкцию, для проверки конструкции, производственных испытаний или диагностики неисправностей. Производитель компонентов может потребовать, чтобы тесты, выполняемые с использованием этих функций, давали результаты, отличающиеся между вариантами компонента, например, что затруднило бы документирование и использование покупателями компонентов
.

Обратите внимание, что некоторые частные инструкции могут привести к тому, что компонент будет работать опасным образом. Например, если частная команда приводит к тому, что входные данные компонента становятся выходными для тестовых данных и т.д., то выбор команды, когда компонент окружен другими компонентами на собранной плате, может привести к повреждению. Поэтому поставщик должен четко идентифицировать любые двоичные коды инструкций, которые могут привести к опасной эксплуатации, если их использует покупатель компонентов.

8.4 *Инструкция* по **ОБХОДУ**

Байпасный регистр содержит один каскад сдвигового регистра и используется для обеспечения последовательного канала минимальной длины между выводами TDI и TDO компонента, когда тестовая эксплуатация этого компонента не требуется. Это позволяет более быстро передавать тестовые данные к другим компонентам на плате и от них, которые необходимы для выполнения тестовых операций.

8.4.1 Технические характеристики

Правила

a) Каждый компонент должен предоставлять инструкцию по *ОБХОДУ*.
b) Двоичным кодом для команды *ОБХОДА* должно быть {111...1} (т.е. логическая единица 1, введенная в каждую ячейку регистра команд).

в) Команда *ОБХОДА* должна выбрать регистр обхода, который будет подключен для последовательного доступа между TDI и TDO в состоянии контроллера *Shift-DR*.

г) Когда выбрана команда *ОБХОДА*, все регистры тестовых данных, которые могут работать в системном режиме или в режимах тестирования, должны выполнять свои системные функции.

е) Когда выбрана команда *ОБХОДА*, работа тестовой логики не должна влиять на работу системной логики на кристалле.

Разрешения

е) Команда *ОБХОДА* может иметь двоичные коды в дополнение к тому, который определен в правиле 8.4.1 b).

8.4.2 Описание

Команду *ОБХОДА* можно ввести, удерживая TDI на постоянном высоком значении и завершая цикл "инструкция-сканирование". Следовательно, требования к основной тестовой системе снижаются в случаях, когда требуется доступ, скажем, только к чипу 57 на 100-чипной плате. В этом случае общий шаблон инструкций, который должен быть перенесен в дизайн, состоит из фона в 1 секунду с небольшим полем конкретных данных инструкций.

Обратите также внимание, что поскольку вход TDI сконструирован таким образом, что, когда он не отключен, он ведет себя так, как если бы подавался высокий сигнал, ошибка разомкнутой цепи в канале тестовых данных на уровне последовательной платы приведет к выбору байпасного регистра после цикла сканирования команд. Таким образом, не может возникнуть нежелательного вмешательства в работу встроенной системной логики.

Если в компоненте не предусмотрен регистр идентификации устройства, команда *ОБХОДА* принудительно вводится в защелки на параллельных выходах регистра команд во время состояния контроллера *Test-Logic-Reset*. Это гарантирует, что установлен полный последовательный путь либо через байпас, либо через регистры идентификации устройства.

При использовании следует учитывать, что если в одних компонентах работает *БАЙПАС*, а в других - инструкции тестового режима (например, *EXTEST*), то нормальная системная логика работы тех компонентов, которые используют *БАЙПАС*, может конфликтовать с тестовой работой других. Поэтому необходим тщательный анализ взаимодействий.

8.5 Инструкции по регистрации сканирования границ

Как обсуждалось в разделе 1, регистр граничного сканирования состоит из ячеек, соединенных между встроенной системной логикой и системными входными и выходными выводами компонента. Этот подраздел включен для того, чтобы предоставить обзор структуры и работы регистра сканирования границ, который поможет читателю понять спецификации обязательных и необязательных инструкций, использующих регистр сканирования границ
.

Требования к конструкции для инструкций регистра сканирования границ содержатся в разделах 8.6 - 8.11. Требования к конструкции ячеек регистра граничного сканирования содержатся в пункте 11.

8.5.1 Обзор работы регистратора сканирования границ

Регистр граничного сканирования представляет собой структуру, основанную на регистре сдвига, которая включает в себя множество различных конструкций ячеек, подобранных в соответствии с требованиями конкретного компонента. Различные конструкции ячеек используются в зависимости от типа соответствующего системного вывода (входной, выходной, трехфазный, двунаправленный) и в соответствии с набором поддерживаемых инструкций пограничного сканирования.

Упрощенный вид регистра сканирования границ показан на рис. 8-1.

Рисунок 8-1 — Упрощенный вид регистратора сканирования границ

Пример реализации ячейки, которую можно было бы использовать в каждом из местоположений, показанных на рис. 8-1, приведен на рис. 8-2.

Соединения, обозначенные как PI, PO, SI и TAK далее на рисунке 8-2, подключены к соседним ячейкам, встроенной в микросхему системной логике и системным выводам, как показано на рисунке 8-1. Как и все ячейки, показанные в этом стандарте, та, что показана на рисунке 8-2, предназначена для реагирования на сигналы ClockDR, ShiftDR и UpdateDR, генерируемые примерной реализацией контроллера TAP, показанной на рисунках 6-5 и 6-6. Входом режима следует управлять в соответствии с типом контакта, подключенного к ячейке (вход, выход и т.д.), и выбранной конкретной инструкцией.

Использование этой конструкции ячейки с соответствующими сигналами, подаваемыми на вход режима каждой ячейки, приведет к созданию компонента, поддерживающего команды *ВЫБОРКИ*, *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ*, *EXTEST* и *INTEST*. Как будет обсуждаться в пункте 11, возможны другие конструкции ячеек, соответствующие требованиям настоящего стандарта для различных наборов инструкций. Например:

a) R2 может быть либо триггером (как показано), либо защелкой.
b) R2 необязателен для ячеек, которые передают данные с системного вывода на встроенную системную логику, например, ячеек на системных входных выводах. Нижний входной сигнал на M2 в таких случаях подавался бы непосредственно с выхода R1.
c) Если бы команда INTEST не поддерживалась, R2 и M2 могли бы отсутствовать в ячейках, которые передают данные из системного вывода в системную логику на кристалле. Тогда вход с надписью PI был бы подключен напрямую к выходу с надписью PO.

8.5.2 Спецификации инструкций регистратора пограничного сканирования

Спецификации инструкций пограничного сканирования, приведенные в следующих подпунктах этого пункта, определяют

а) Является ли инструкция обязательной или необязательной;
б) Какие регистры тестовых данных могут быть подключены по последовательному каналу между TDI и TDO;
в) Ограничения (если таковые имеются) на выбор двоичных кодов для каждой команды (т.е. Шаблоны единиц и 0, которые при перемещении в регистр команд вызывают выбор команды); и
г) Поток данных между системными выводами компонента, ячейками регистра граничного сканирования и встроенной в микросхему системной логикой.

Рисунок 8-2 —Пример конструкции ячейки регистра с граничным сканированием

Спецификации подкреплены описательным текстом, который включает версию рисунка 8-3, на котором показаны один вход и один выход для компонента. Сплошными жирными линиями на более поздних копиях этого рисунка показаны обязательные потоки данных для каждой инструкции.

8.6 *ОБРАЗЕЦ* инструкции

Обязательный *ОБРАЗЕЦ* инструкции позволяет сделать снимок нормальной работы компонента и изучить его.

8.6.1 Технические характеристики

Правила

а) Каждый компонент должен содержать *ОБРАЗЕЦ* инструкции.

б) В *ОБРАЗЦЕ* инструкции должен быть выбран *только* регистр граничного сканирования, который должен быть подключен для последовательного доступа между TDI и TDO в состоянии контроллера *Shift-DR* (т.е. Никакой другой регистр тестовых данных не может быть подключен последовательно к регистру граничного сканирования).

в) Когда выбрана команда *SAMPLE*, работа тестовой логики не должна влиять на работу встроенной системной логики или на поток сигналов между системными выводами и встроенной системной логикой.

г) Когда выбрана команда *SAMPLE*, состояния всех сигналов, поступающих от системной логики на кристалле или через системные выводы (входные или выходные), должны загружаться в регистр граничного сканирования на переднем фронте TCK в состоянии контроллера *Capture-DR*.

ПРИМЕЧАНИЕ — Цель этого правила - указать, когда должно произойти действие загрузки. Подробные спецификации выбора загружаемых значений сигналов приведены в Правилах 11.5.1 f) и 11.6.1 h), соответственно, для системных логических входов и системных логических выходов.

Рекомендации

е) Там, где каждый из *SAMPLE* и *PRELOAD* реализует функциональность другого, они должны использовать общее двоичное значение (значения).

ионы

ruct
st

n в

ска

ри-

унда

те бо
ра

иллюзии
к

сед
u
это

3—Й круг

е 8-

Figur

,,,

Разрешения

е) Когда выбрана команда *SAMPLE*, параллельные выходные регистры/защелки, включенные в ячейки регистра пограничного сканирования, могут загружать данные, хранящиеся в соответствующем каскаде сдвигового регистра на падающем фронте TCK в состоянии контроллера *Update-DR*.

ж) Двоичное значение (значения) для *ОБРАЗЦА* инструкции может быть выбрано разработчиком компонентов.

8.6.2 Описание

ПРИМЕР инструкции позволяет сделать снимок состояний входных и выходных сигналов компонента, не вмешиваясь в нормальную работу собранной платы. Снимок делается на переднем фронте TCK в состоянии контроллера *Capture-DR*, и затем данные могут быть просмотрены путем переключения через выходные данные TDO компонента. Примерами применения возможности *SAMPLE* являются

а) Чтобы предоставить аналог к экскурсии-процесс зондирования выполняются на собранном совете при функциональной диагностики, но без необходимости физического контакта; и

б) Проверка взаимодействия компонентов во время нормального функционирования.

Поток данных для *ОБРАЗЦА* инструкции показан на рисунке 8-4. Как можно видеть, *SAMPLE* можно использовать , не создавая помех нормальной работе встроенной системной логики. Данные, полученные на системных входных выводах, подаются без модификации в системную логику на кристалле, данные из системной логики на кристалле передаются без модификации через выходные выводы системы и т.д. Для примера конструкции ячейки регистра с граничным сканированием, приведенной на рис. 8-2, это достигается удерживанием значения режима ввода на 0, когда *выбрана* команда для ВЫБОРКИ.

Примечания

1—На выходных контактах выборки сигнала могут быть либо выходными данными компонента, либо выходными данными встроенной системной логики.

2. Компонент может быть сконструирован таким образом, чтобы *ОБРАЗЕЦ* и *ПРЕДВАРИТЕЛЬНАЯ НАГРУЗКА* инструкции объединяются путем присвоения им одного и того же двоичного кода. В то время как *SAMPLE* фиксирует данные в регистре граничного сканирования и позволяет извлекать их через TDO для проверки, конкретное использование данных, перенесенных в TDI, не является обязательным. Напротив, *ПРЕДВАРИТЕЛЬНАЯ ЗАГРУЗКА* сдвигает данные в регистр граничного сканирования через TDI таким образом, что они могут быть загружены в параллельные выходные регистры / защелки регистра перед выбором команды (такой как *EXTEST*), которая передает данные, хранящиеся в этих регистрах / защелках, на выходные выводы компонента. Данные, записываемые в регистр сканирования границ перед сдвигом, не определены. Взаимная исключительность этих вариантов поведения позволяет объединять инструкции там, где это необходимо [см. Разрешения 8.1.1 g), 8.6.1 f) и 8.7.1 f)]. Далее, где инструкции по *ВЫБОРКЕ* и *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКЕ* объединяются таким образом, перемещая контроллер ОТВОДА через последовательность состояний *Capture-DR* → *Выход1-DR* → *Обновление-DR* пока*выбрана объединенная команда* ВЫБОРКИ/ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ , состояние сигналов, поступающих в системную логику на кристалле и выходящих из нее во время выборки, может быть загружено на заблокированный параллельный выход сдвигового регистра граничной развертки.

8.7 Инструкция по *ПРЕДВАРИТЕЛЬНОМУ НАТЯГУ*

Обязательная команда *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ* позволяет загружать значения данных на заблокированные параллельные выходы сдвигового регистра граничной развертки перед выбором других инструкций проверки граничной развертки.

8.7.1 Технические характеристики

Правила

а) Каждый компонент должен содержать инструкцию по *ПРЕДВАРИТЕЛЬНОМУ НАТЯГУ*.

б) Команда *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ* должна выбирать *только* регистр граничной развертки, который должен быть подключен для последовательного доступа между TDI и TDO в состоянии контроллера *Shift-DR* (т.е. Никакой другой регистр тестовых данных не может быть подключен последовательно к регистру граничной развертки).

в) Когда выбрана команда *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ*, работа тестовой логики не должна влиять на работу встроенной системной логики или на поток сигналов между системными выводами и встроенной системной логикой.

г) Когда выбрана команда *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ*, параллельные выходные регистры / защелки, включенные в ячейки регистра граничного сканирования, должны загружать данные, хранящиеся в соответствующем каскаде сдвигового регистра на падающем фронте TCK в состоянии контроллера *Update-DR*.

Рекомендации

е) Там, где каждый из *SAMPLE* и *PRELOAD* реализует функциональность другого, они должны использовать общее двоичное значение (значения).

Разрешения

f) Когда выбрана команда *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ*, состояния всех сигналов, проходящих через системные контакты (входные или выходные), могут быть загружены в регистр граничного сканирования на переднем фронте TCK в состоянии контроллера *Capture-DR*.

g) Двоичное значение (значения) для команды *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ* может быть выбрано разработчиком компонентов.

8.7.2 Описание

Команда *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ* позволяет сканировать регистр граничного сканирования, не создавая помех нормальной работе встроенной системной логики. Таким образом, это позволяет разместить начальный набор данных на заблокированных параллельных выходах ячеек регистра сканирования границ (например, как предусмотрено в ячейках, подключенных к системным выходным выводам) перед выбором другой тестовой операции сканирования границ. Например, перед выбором команды *EXTEST* данные могут быть загружены на заблокированные параллельные выходы с помощью *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ*. Как только команда *EXTEST* была передана на параллельный вывод регистра команд, предварительно загруженные данные передаются через выходные контакты системы. Это гарантирует, что известные данные, согласованные на уровне платы, передаются немедленно при вводе команды *EXTEST*; без *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ* неопределенные данные передавались бы до завершения первой последовательности сканирования.

Поток данных для команды *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ* показан на рисунке 8-5. Данные, полученные на системных входных выводах, подаются без модификации в встроенную системную логику, данные из встроенной системной логики передаются без модификации через системные выходные выводы и т.д. Для примера конструкции ячейки регистра с граничным сканированием, приведенной на рис. 8-2, это достигается удерживанием входного значения режима на 0 при выборе команды *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ*.

ПРИМЕЧАНИЕ—Компонент может быть спроектирован таким образом, что инструкции *SAMPLE* и *PRELOAD* объединяются путем присвоения им обоим одного и того же двоичного кода. В то время как *SAMPLE* фиксирует данные в регистре граничного сканирования и позволяет переносить их через TDO для проверки, конкретное использование данных, перенесенных в TDI, не является обязательным. Напротив, *ПРЕДВАРИТЕЛЬНАЯ ЗАГРУЗКА* сдвигает данные в регистр граничного сканирования через TDI таким образом, что они могут быть загружены в параллельные выходные регистры / защелки регистра перед выбором команды (такой как *EXTEST*), которая передает данные, хранящиеся в этих регистрах / защелках, на выходные выводы компонента. Данные, записываемые в регистр сканирования границ перед сдвигом, не определены.
Взаимная исключительность этих вариантов поведения позволяет объединять инструкции там, где это необходимо [см. Разрешения 8.1.1 g), 8.6.1 f) и 8.7.1 f)]. Далее, где инструкции по *ВЫБОРКЕ* и *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКЕ* объединяются таким образом, перемещая контроллер отвода по последовательности состояний *Capture-DR* → *Выход1-DR* → *Обновление-DR* пока выбрана объединенная команда *ВЫБОРКИ/ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ*, состояние сигналов, поступающих в системную логику на кристалле и выходящих из нее во время выборки, может быть загружено на заблокированный параллельный выход сдвигового регистра граничной развертки.

8.8 *РАСШИРЕННАЯ* инструкция

Обязательная инструкция *EXTEST* позволяет тестировать внекристальную схему и межсоединения на уровне платы. Данные обычно загружаются на заблокированные параллельные выходы каскадов сдвигового регистра пограничного сканирования с помощью использования команды *ПРЕДВАРИТЕЛЬНОЙ* ЗАГРУЗКИ перед выбором команды *EXTEST*.

n
io
ct
u
tr

ins

ПРЕДВАРИТЕЛЬНАЯ НАГРУЗКА
e

р-й
фо
ш

фло
та

Да

5—

re 8-
u
ig
F

ПРИМЕЧАНИЕ—После использования команды *EXTEST* системная логика на кристалле может находиться в неопределенном состоянии, которое будет сохраняться до тех пор, пока не будет применен системный сброс. Следовательно, встроенную системную логику, возможно, потребуется сбросить при возврате к нормальной (т. е. нетестовой) работе.

8.8.1 Технические характеристики

Правила

а) Каждый компонент должен содержать *РАСШИРЕННУЮ* инструкцию.

б) Команда *EXTEST* должна выбирать только регистр граничной развертки, который должен быть подключен для последовательного доступа между TDI и TDO в состоянии контроллера *Shift-DR* (т.е. Никакой другой регистр тестовых данных не может быть подключен последовательно к регистру граничной развертки).

с) Пока выбрана команда *EXTEST*, системная логика на кристалле должна управляться таким образом, чтобы она не могла быть повреждена в результате сигналов, принятых на системный вход или выводы системных тактовых импульсов.

ПРИМЕЧАНИЕ — Этого можно достичь, переведя встроенную системную логику в состояние сброса или “удержания”, пока выбрана *ПОСЛЕДНЯЯ* инструкция .

г) Когда выбрана команда *EXTEST*, состояние всех сигналов, поступающих с системных выходных выводов, должно полностью определяться данными, хранящимися в регистре граничного сканирования, и изменяться только на падающем фронте TCK в состоянии контроллера *Update-DR*.

е) Когда выбрана команда *EXTEST*, состояние всех сигналов, принятых на выводах системного ввода, должно загружаться в регистр пограничного сканирования на переднем фронте TCK в состоянии *контроллера* Capture-DR.

Рекомендации

f) Данные, загружаемые в ячейки регистра граничного сканирования, расположенные на выводах системного вывода (2-state, 3-state или двунаправленный) в состоянии контроллера *Capture-DR*, когда выбрана команда *EXTEST*, должны быть независимыми от работы системной логики на кристалле.

ж) Для каждой ячейки регистра граничного сканирования должно быть определено значение, которое при выборе *команды EXTEST* позволит перегружать все выходные сигналы компонента одновременно в течение неопределенного периода времени без риска повреждения компонента.

ПРИМЕЧАНИЕ —Этого легко достичь, если все выходы можно перевести в неактивное состояние привода с помощью инструкции *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ* .

Разрешения

h) Двоичное значение (значения) для команды *EXTEST* может быть выбрано разработчиком компонентов.

8.8.2 Описание

Команда *EXTEST* позволяет протестировать схемы, внешние по отношению к пакету компонентов, — обычно межсоединение платы . Ячейки регистра граничного сканирования на выходных выводах используются для применения тестовых стимулов, в то время как ячейки регистра на входных выводах фиксируют результаты теста. Эта инструкция также позволяет тестировать блоки компонентов, которые сами по себе не включают регистры граничного сканирования. Поток данных через ячейки регистра пограничного сканирования в этой конфигурации показан на рисунке 8-6. Например, на входных выводах данные сначала записываются в канал сдвигового регистра, а затем выводятся из компонента для проверки; на выходных выводах данные, перенесенные в компонент, применяются к внешнему межсоединению.

Как правило, первый тестовый стимул, который будет применен с использованием команды *EXTEST*, будет перенесен в регистр граничного сканирования с использованием команды *PRELOAD*. Таким образом, когда изменение *EXTEST* инструкции происходит в состоянии контроллера *Update-IR*, известные данные будут немедленно переданы из компонента на его внешние подключения. Где необходимо провести в общей сложности *N* тестов с использованием *РАСШИРЕННОЙ* инструкции, стимулов

для тестов от 2 до *N* будут сдвинуты внутрь, в то время как результаты тестов от 1 до *N*-1 будут смещены наружу. Обратите внимание, что в то время как результаты финального теста — test *N* — удаляются, должен быть перемещен определенный набор данных, который оставит плату в согласованном состоянии в конце процесса перемещения. Этого можно достичь, снова переведя стимулы для теста *N* (или любого другого теста) в регистр граничного сканирования.

Команда *EXTEST* также позволяет установить выходы компонентов в состояние, которое сводит к минимуму риск повреждения при перегрузке во время внутрисхемного тестирования [см. Рекомендацию 8.8.1 g)]. Такое тестирование может использоваться в тех случаях, когда не все компоненты на собранной плате можно протестировать с помощью граничного сканирования.

Обратите внимание, что ячейки регистра граничной развертки, расположенные на входных выводах, могут быть дополнительно сконструированы таким образом, чтобы сигналы могли подаваться в системную логику на кристалле, когда выбрана команда *EXTEST*. Это позволяет устанавливать определенные пользователем значения на входах системной логики, предотвращая неправильную работу в ответ на шумовые сигналы, поступающие от межсоединения на уровне платы. Управляемые значения могут быть постоянными в течение времени выбора *EXTEST* (например, путем включения блокирующего элемента на входе системной логики) или могут загружаться последовательно через регистр граничного сканирования, как показано на рисунке 8-6.

Рекомендация 8.8.1 f), при соблюдении которой данные, перемещенные из компонента в ответ на команду *EXTEST*, не изменяются из-за наличия сбоев в системной логике на кристалле. Это упрощает диагностику, поскольку любые ошибки в выходном битовом потоке могут быть вызваны только неисправностями во внекристальных схемах, во межсоединениях на уровне платы или в регистрах граничного сканирования, используемых для применения теста.

Пока выбрана команда *EXTEST*, встроенная системная логика может принимать входные сигналы, которые значительно отличаются от ожидаемых при нормальной (нетестовой) работе. Правило 8.8.1 c) возлагает ответственность за правильное разрешение этой ситуации на разработчика компонента. Если встроенная системная логика может допускать любую перестановку принимаемых входных сигналов, никаких специальных конструктивных изменений для соответствия этому правилу не требуется. (Примером здесь может служить случай, когда системная логика на кристалле полностью комбинационная.) Однако для некоторых компонентов могут существовать входные последовательности, которые могут привести системную логику на кристалле в состояние, при котором это может привести к повреждению. В этих случаях разработчик несет ответственность за предотвращение того, чтобы встроенная системная логика обрабатывала “незаконные” входные данные, пока выбрана команда *EXTEST*. Как уже отмечалось, это может быть достигнуто путем перевода встроенной системной логики в состояние сброса или “удержания”.

Альтернативно, данные, хранящиеся в регистре граничного сканирования, могут быть представлены системной логике на кристалле, пока выбирается самая *РАСШИРЕННАЯ* команда. Обратите внимание, что в этом случае Правило 11.3.1 e) запрещает наложение любых ограничений на логические значения, которые могут быть приведены к системной логике на кристалле.

Обратите внимание, что в то время как более ранние редакции этого стандарта требовали, чтобы двоичный код для *EXTEST* был равен {000 ...0}, использование этого двоичного кода для *EXTEST* и всех других инструкций тестового режима устарело (см. Рекомендацию 8.1.1 e)] и соответствующее примечание).

8.9 Инструкция *INTEST*

Дополнительная инструкция *INTEST* - это одна из двух инструкций, определенных настоящим стандартом, которые позволяют тестировать встроенную системную логику во время сборки компонента на плате. Используя инструкцию *INTEST*, тестовые стимулы сдвигаются по одному за раз и применяются к встроенной системной логике. Результаты теста записываются в регистр граничного сканирования и проверяются путем последующего сдвига. Данные обычно загружаются на зафиксированные параллельные выходы каскадов сдвигового регистра граничного сканирования с использованием команды *ПРЕДВАРИТЕЛЬНОЙ* ЗАГРУЗКИ перед выбором команды *INTEST*.

При предоставлении инструкции по *INTEST* применяются следующие правила.

ПРИМЕЧАНИЕ—После использования команды *INTEST* системная логика на кристалле может находиться в неопределенном состоянии, которое будет сохраняться до тех пор, пока не будет применен системный сброс. Следовательно, встроенную системную логику, возможно, потребуется сбросить при возврате к нормальной (т. е. нетестовой) работе.

выбрана инструкция

EXTEST

пока

течь
та

т да
с
е

6—Т

ре 8-

Рисунок

8.9.1 Технические характеристики

Правила

а) Команда *INTEST* должна выбирать только регистр граничного сканирования для подключения к последовательному доступу между TDI и TDO в состоянии контроллера *Shift-DR* (т.е. Никакой другой регистр тестовых данных не может быть подключен последовательно к регистру граничного сканирования).

б) Встроенная системная логика должна быть способна к одноэтапной работе, пока *выбрана команда*INTEST .

в) Когда выбрана команда *INTEST*, все системные выходные данные компонента должны быть определены следующим образом:

1) Все сигналы, исходящие от компонента, должны определяться данными, хранящимися в регистре пограничного сканирования , и должны изменяться только на падающем фронте TCK в *Update-DR* состояние контроллера или при выборе команды *INTEST*; или

2) Все выходные сигналы компонента (включая те, которые являются нетестовыми сигналами в двух состояниях) должны переводиться в неактивное состояние привода (например, высокоимпедансное) при выборе команды *INTEST*.

г) Когда выбрана команда *INTEST*, состояние всех сигналов без синхронизации, подаваемых в системную логику из регистра граничного сканирования, должно полностью определяться данными, хранящимися в регистре.

е) Когда выбрана команда *INTEST*, состояние всех сигналов, выводимых из системной логики в регистр пограничного сканирования, должно быть загружено в регистр на переднем фронте TCK в состоянии *контроллера Capture-DR*.

Рекомендации

е) Для ячеек регистра граничного сканирования, расположенных на системных входных контактах (тактовых или не тактовых) или на двунаправленных контактах, сконфигурированных как входные, данные, загружаемые в состояние контроллера *Capture-DR* при выборе команды *INTEST* , должны быть независимыми от работы внекристальных схем или межсоединений на уровне платы.

Разрешения

g) Двоичное значение (значения) для команды *INTEST* может быть выбрано разработчиком компонентов.

8.9.2 Описание

Команда *INTEST* позволяет проводить статическое (низкоскоростное) тестирование системной логики на кристалле, при этом каждый тестовый шаблон и отклик перемещаются через регистр граничного сканирования. Команда *INTEST* требует, чтобы системная логика на кристалле могла работать в одноступенчатом режиме, когда схема продвигается на один шаг вперед в своей работе каждый раз, когда завершается сдвиг регистра граничной развертки.

Поток данных через ячейки регистра пограничного сканирования при выборе команды показан жирными контурами на рисунке 8-7. Самый верхний выделенный жирным шрифтом путь через ячейку на выводе вывода - это путь, выбранный результатами тестирования системной логики на кристалле; самый нижний путь - это путь, выбранный данными, которые будут храниться на выводе, пока применяется тест. Обратите внимание, что для каждого теста зафиксированный параллельный вывод ячейки регистра граничного сканирования на выводе системного вывода обновляется на основе данных, сдвинутых в *до* того, как состояние сдвигового регистра перезаписывается ответом теста.

Пока выбрана команда *INTEST*, регистр граничного сканирования выполняет роль системы ATE, используемой для тестирования автономных компонентов. Ячейки на выводах ввода системы, отличных от тактовых, используются для применения тестового стимула, в то время как ячейки на выводах вывода системы фиксируют реакцию. Стимулы и реакции вводятся в схему и выводятся из нее путем сдвига регистра граничного сканирования. Обратите внимание, что для этого требуется, чтобы ячейки регистра пограничного сканирования, расположенные на системных входных контактах, могли передавать сигналы в системную логику на кристалле.

ЕД
КТ
Ле

ЮВ
ы
я
Н

Тио
с

тру
ы
Н
я

КИШЕЧНИК

я
ч

ш

а

st dat
e

рисунок 8-7—Т

Как правило, системная логика на кристалле принимает последовательность событий синхронизации между применением стимула и получением ответа таким образом, что достигается одноэтапная операция. Спецификация ячеек регистра граничного сканирования для входных выводов системных тактовых импульсов позволяет получать тактовые импульсы для встроенной системной логики несколькими способами при выборе команды *INTEST*. В качестве примеров предлагаются следующие:

а) Сигналы, принимаемые на выводах системных тактовых импульсов, могут подаваться непосредственно на встроенную системную логику, как при нормальной работе компонента. Если выбран этот параметр, конструкция компонента должна гарантировать, что выполняется ровно один шаг работы встроенной системной логики, пока, по крайней мере, применяется указанное минимальное количество циклов TCK во время состояния *Run-Test/Idle* контроллера . Компонент должен быть спроектирован таким образом, чтобы выполнялся только один этап работы, независимо от того, применяется ли более указанного минимального количества циклов TCK, пока контроллер TAP находится в состоянии *Run-Test /Idle* контроллера. Это может, например, потребовать, чтобы тактовые сигналы, поступающие на компонент, были стробированы перед применением к встроенной системной логике. Таким образом, работа встроенной системной логики может быть заблокирована, пока тестовые данные перемещаются через регистр граничного сканирования. Рисунок 8-8 иллюстрирует, как следует управлять системными часами, применяемыми к компоненту, во время тестирования встроенной системной логики с использованием команды *INTEST*.

В то время как рисунок 8-8 иллюстрирует ситуацию, в которой системный тактовый сигнал представляет собой один положительный импульс, правило 8.9.1 b) может быть обобщено для применения к компонентам, которые используют несколько тактовых циклов для каждого шага работы или которые имеют несколько входных выводов тактового сигнала, на которые принимаются многофазные тактовые сигналы. Обратите внимание, что хотя на рисунке 8-8 показан вход в состояние *Run-Test/Idle* контроллера из состояния *Update-DR* контроллера, тактовые импульсы также применялись бы к системной логике на кристалле, если бы состояние *Run-Test/Idle* контроллера было введено из состояния *Update-IR* контроллера.

Рисунок 8-8—Управление применяемыми системными часами во время *INTEST*

б) Встроенная системная логика может быть снабжена тактовыми сигналами, полученными от TCK в состоянии *Run-Test/Idle* контроллера. Во всех остальных состояниях контроллера часы не должны менять состояние. На рисунке 8-9 показан производный тактовый сигнал, в котором, например, встроенная системная логика реагирует на возрастающие фронты тактовой частоты.

Рисунок 8-9—Использование TCK в качестве синхронизации для встроенной системной логики во время *INTEST*

в) В компонент может быть встроена схема, которая при переходе в состояние *Run-Test/Idle* контроллера позволяет встроенной системной логике завершить один этап работы. Например, если бы компонентом был микропроцессор, было бы разрешено завершить один цикл обработки, например, путем внутренней генерации импульса по сигналу удержания. В этом случае часы, примененные к системному тактовому контакту (выводам) во время теста, могут работать автономно.

г) Тактовые сигналы могут быть сдвинуты по каналу граничного сканирования тем же способом, которым подаются не тактовые сигналы для встроенной системной логики. Обратите внимание, что для этого потребуется сдвигать регистр граничной развертки для каждого отдельного состояния тактового сигнала (например, дважды для однофазного тактового сигнала).

ПРИМЕЧАНИЕ — Эта операция может быть опасной для определенных схем.

Пока выбрана команда *INTEST*, состояние всех системных выходных выводов определяется логикой тестирования. Есть два варианта. Во-первых, состояние вывода может определяться данными, хранящимися в регистре граничного сканирования, смещенными на заблокированные параллельные выходы регистра во время каждого прохождения последовательности сканирования для регистра. Во-вторых, каждый системный выходной вывод может быть переведен в неактивное состояние привода (например, высокоимпедансный). Это гарантирует, что окружающие компоненты на собранной плате получают известные уровни сигнала во время выполнения встроенного тестирования системной логики. Как правило, согласованный набор значений данных будет перенесен в соответствующие этапы регистра сканирования границ с использованием команды *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ* перед выбором команды *INTEST*. Затем этот шаблон данных перезагружается каждый раз, когда новый тестовый шаблон *INTEST* перемещается в регистр граничного сканирования.

Рекомендация 8.9.1 f), при соблюдении которой данные, передаваемые из компонента в ответ на команду *INTEST*, не изменяются из-за наличия сбоев во внекристаллитной системной логике, межсоединениях на уровне платы и т.д.... Это упрощает диагностику, поскольку любые ошибки в выходном битовом потоке могут быть вызваны только неисправностями в встроенной системной логике или в регистре граничного сканирования.

8.10 *Инструкция* по РУНБИСТУ

Необязательная команда *RUNBIST* вызывает выполнение автономной самодиагностики компонента. Использование инструкции позволяет пользователю компонента определять работоспособность компонента без необходимости загружать сложные шаблоны данных и без необходимости выполнения одноступенчатой операции (как требуется для инструкции *INTEST*). Пока выбрана команда *RUNBIST*, состояние всех системных выходных выводов определяется логикой тестирования. Есть два варианта. Во-первых, состояние вывода может определяться данными, хранящимися в регистре граничного сканирования, смещенными на заблокированные параллельные выходы регистра во время каждого прохождения через последовательность сканирования для регистра. Во-вторых, каждый системный выходной вывод может быть переведен в неактивное состояние привода (например, высокоимпедансный).

При предоставлении инструкции по *RUNBIST* применяются следующие правила.

ПРИМЕЧАНИЕ—После использования инструкции *RUNBIST* системная логика на кристалле может находиться в неопределенном состоянии, которое будет сохраняться до тех пор, пока не будет применен системный сброс. Следовательно, может потребоваться сброс встроенной системной логики при возврате к нормальной (т.е. нетестовой) работе.

8.10.1 Технические характеристики

Правила

а) Когда выбрана команда *RUNBIST*, регистр тестовых данных, в который будут загружены результаты самостоятельного тестирования, должен быть подключен для последовательного доступа между TDI и TDO в состоянии *контроллера Shift-DR*.

b) Режим (режимы) самопроверки, доступ к которым осуществляется через команду *RUNBIST*, должен выполняться только в состоянии *Run-Test/Idle* контроллера.

в) Если требуется инициализация регистра тестовых данных перед выполнением самопроверки, это должно происходить в начале самопроверки без какого-либо требования о переносе данных в компонент (т.е. Не должно требоваться вводить начальные значения в какой-либо регистр тестовых данных).

ПРИМЕЧАНИЕ—Согласно правилу 8.10.1 k) 1), регистр граничного сканирования может (необязательно) потребоваться инициализировать для определения состояния сигналов, поступающих от системных выходных контактов. Однако это значение не следует использовать в качестве начального для операции самопроверки, поскольку оно может зависеть от платы.

г) Для теста, выполняемого в ответ на команду *RUNBIST*, должна быть указана продолжительность (например, количество возрастающих фронтов TCK или системные часы).

е) Результат самодиагностики, выполняемой в ответ на команду *RUNBIST*, должен быть загружен в регистр тестовых данных, подключенный между TDI и TDO, не позднее, чем возрастающий фронт TCK в состоянии контроллера *Capture-DR*.

е) По истечении указанного минимального срока результат теста, наблюдаемый при загрузке и сдвиге регистра данных теста, выбранного командой *RUNBIST*, должен быть постоянным независимо от того, когда введено состояние контроллера *Capture-DR*.

ж) Использование инструкции *RUNBIST* должно давать одинаковый результат во всех версиях компонента.

з) Данные, извлеченные из компонента после завершения выполнения самодиагностики, доступ к которой осуществляется с помощью команды *RUNBIST*, не должны зависеть от работы внекристаллитных схем или подключений на уровне платы.

i) Все этапы регистра тестовых данных, выбранные командой *RUNBIST*, должны быть установлены в определенные логические состояния (0 или 1) не позднее, чем возрастающий фронт TCK в состоянии контроллера *Capture-DR*.

к) Конструкция компонента должна обеспечивать, чтобы на результаты самопроверки, выполняемой в ответ на команду *RUNBIST*, не влияли сигналы, принимаемые на входные контакты системы без тактовой синхронизации.

к) Когда выбрана команда *RUNBIST*, все системные выходные данные компонента должны быть определены следующим образом:

1) Все сигналы, исходящие от компонента, должны определяться данными, хранящимися в регистре пограничного сканирования , и должны изменяться только на падающем фронте TCK в состоянии контроллера *Update-DR* или при выборе команды *RUNBIST*; или

2) Все выходные сигналы компонента (включая те, которые не являются тестовыми сигналами в двух состояниях) должны переводиться в неактивное состояние привода (например, высокоимпедансное) при выборе инструкции *RUNBIST* .

l) Состояния регистров параллельного вывода или защелок в ячейках регистра граничного сканирования, расположенных на системных выходных выводах (двух-, трех- или двунаправленных), не должны изменяться, пока *выбрана команда* RUNBIST , если только соответствующий вывод не был переведен в неактивное состояние привода (например, высокоимпедансный), какопределено в Правиле 8.10.1 k)2).

Рекомендации

м) Там, где это возможно, компоненты, совместимые с этим стандартом, должны поддерживать инструкцию *RUNBIST*.

Разрешения

п) Двоичное значение (значения) для команды *RUNBIST* может быть выбрано разработчиком компонентов.

о) Если компонент включает в себя несколько функций самопроверки, эти функции могут выполняться либо одновременно, либо в последовательности, определенной производителем компонента в ответ на инструкцию *RUNBIST*. В последнем случае вся последовательность должна выполняться внутри самого компонента, не требуя изменения содержимого регистра команд.

р) Могут быть предоставлены дополнительные общедоступные инструкции для предоставления пользователю доступа к отдельным функциям самопроверки внутри компонента.

в) Регистр тестовых данных, подключенный между TDI и TDO при выборе команды *RUNBIST*, может быть регистром граничного сканирования.

р) Пока выбрана команда *RUNBIST*, регистр сканирования границ может действовать как генератор шаблонов или уплотнитель сигнатур в состоянии *Run-Test/Idle* контроллера при условии соблюдения правила 8.10.1 l).

8.10.2 Описание

Инструкция *RUNBIST* предоставляет покупателю компонента средство запуска доступной пользователю функции самостоятельного тестирования внутри компонента в результате выполнения одной инструкции. Это позволяет всем компонентам на плате, которые предлагают инструкцию *RUNBIST*, выполнять свои самотестирования одновременно, обеспечивая быструю проверку работоспособности собранной платы. Обратите внимание, однако, что производитель компонента может включать дополнительные частные или общедоступные инструкции для предоставления доступа к отдельным функциям самопроверки по одной за раз или к функциям самопроверки, которые не вызываются инструкцией *RUNBIST*.

Последовательность шагов, требуемых для завершения выполнения *RUNBIST*, может быть определена как

a) (Необязательно) инициализация регистра сканирования границ (например, с помощью *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ*). Это требуется, если состояние вывода во время BIST должно определяться данными на заблокированных параллельных выходах регистра.
b) Инициировать BIST: отсканировать команду *RUNBIST* в регистр команд.
c) Выполнить BIST: заставить контроллер TAP оставаться в состоянии *Run-Test / Idle* контроллера на время, необходимое для завершения выполнения BIST.
г) Оцените результаты BIST: переведите контроллер TAP в состояние контроллера *Shift-DR* и отсканируйте результаты теста (например, подпись) из регистра, подключенного к TDI и TDO с помощью инструкции *RUNBIST*.

Во время выполнения теста логика тестирования определяет выходные данные компонента. Что касается инструкции по *INTEST*, доступны два варианта:

— Состояние PIN-кода может определяться данными, хранящимися в регистре граничного сканирования.
— Каждый вывод системного вывода может быть переведен в неактивное состояние привода (например, высокоомный).

При выборе первого варианта значения данных, передаваемые через выходные контакты системы, фиксируются в момент выбора команды *RUNBIST* на основе данных, хранящихся в регистре граничного сканирования в это время. (Эти данные, возможно, были предварительно загружены с помощью инструкции *PRELOAD*.) Регистр граничного сканирования управляется таким образом, что данные, хранящиеся на заблокированных параллельных выходах ячеек, которые питают выходные выводы системы, не изменяются при выборе команды *RUNBIST*. Ссылаясь на рисунок 8-2, это может быть достигнуто, например, за счет удерживания сигнала UpdateDR на 0, пока выбрана команда *RUNBIST*. Сигнал режима будет удерживаться на значении 1.

Ячейки регистра граничного сканирования также могут использоваться для хранения запрограммированных значений сигналов на входах встроенной системной логики во время выполнения самопроверки (опять же, как показано на рис. 8-7). В качестве альтернативы, ячейки регистра с граничным сканированием, расположенные на входах системной логики, отличных от тактовых, могут быть сконструированы так, чтобы действовать как источник данных самопроверки для встроенной системной логики. Аналогично, ячейки регистра граничного сканирования, расположенные на выходах системной логики, могут действовать как уплотнители результатов самопроверки.

Спецификация ячеек регистра граничной развертки для входных выводов системных тактовых импульсов позволяет получать тактовые импульсы для системной логики на кристалле одним из двух способов при выборе команды *RUNBIST*:

a) Сигналы, принимаемые на выводах системных тактовых импульсов, могут подаваться непосредственно на встроенную системную логику, как при нормальной работе компонента. Там, где это сделано, конструкция компонента должна гарантировать, что самотестирование выполняется *только* в состоянии *Run-Test/Idle* контроллера. Однако часы могут быть активны в других состояниях контроллера.
б) Встроенная системная логика может быть снабжена тактовыми сигналами, полученными от TCK в состоянии *Run-Test/Idle* контроллера. Во всех остальных состояниях контроллера часы не должны менять состояние.

Правила, касающиеся продолжительности самотестирования, выполняемого в ответ на инструкцию *RUNBIST* [Правила 8.10.1 d) и 8.10.1 f)], гарантируют, что может быть применено достаточное количество тактовых импульсов для завершения самотестирования, выполняемого одновременно в разных компонентах на собранной плате. Таким образом, в продуктах, содержащих компоненты с самопроверкой длины 1000, 5000, 10 000, 50 000 и растет часы кромок на ТЦК в

готовая плата должна оставаться в состоянии *Run-Test / Idle* контроллера не менее чем на 50 000 нарастающих фронтов тактовой частоты, чтобы гарантировать удовлетворительное завершение всех тестов. Тесты, завершенные до применения 50 000 фронтов синхронизации, будут сохранять свои результаты до тех пор, пока к ним не будет получен доступ.

Правило 8.10.1 g) включено для обеспечения независимости тестирования собранной платы от версий установленных на ней компонентов. Это важное соображение при работе в условиях технического обслуживания или ремонта , когда версии компонентов, используемых на плате, могут быть неизвестны. Правило может быть выполнено путем формирования исключающего-ИЛИ результата выполнения инструкции *RUNBIST* с фиксированным (зависящим от версии) шаблоном. Результатом работы этой функции станет результат, загруженный в регистр граничного сканирования или другой регистр тестовых данных, подключенный между TDI и TDO.

Правило 8.10.1 h) гарантирует, что данные, перемещенные из компонента в ответ на команду *RUNBIST*, не изменяются из-за наличия сбоев во внекристальной системной логике, межсоединениях на уровне платы и т.д. Это упрощает диагностику, поскольку любые ошибки в выходном битовом потоке могут быть вызваны только сбоями в системной логике на кристалле или в регистре тестовых данных, подключенном по пути между TDI и TDO.

8.11 Инструкция по *ЗАЖИМУ*

Дополнительная команда *CLAMP* позволяет определять состояние сигналов, поступающих от выводов компонентов, из регистра граничной развертки, в то время как регистр обхода выбирается в качестве последовательного канала между TDI и TDO. Сигналы, поступающие от выводов компонентов, не изменятся, пока *выбрана команда*CLAMP .

При использовании инструкции по *ЗАЖИМУ* применяются следующие правила.

ПРИМЕЧАНИЕ—После использования команды *CLAMP* системная логика на кристалле может находиться в неопределенном состоянии, которое будет сохраняться до тех пор, пока не будет применен системный сброс. Следовательно, встроенную системную логику, возможно, потребуется сбросить при возврате к нормальной (т. е. нетестовой) работе.

8.11.1 Технические характеристики

Правила

а) Команда CLAMP должна выбрать байпасный регистр, который будет подключен для последовательного доступа между TDI и TDO в состоянии контроллера Shift-DR.

ПРИМЕЧАНИЕ —Регистр обхода будет вести себя полностью так, как определено в пункте 10, пока выбрана команда *CLAMP*. Следовательно, сначала он загрузит логический 0 во время состояния контроллера *Capture-DR* и переместит данные во время состояния контроллера *Shift-DR*.

б) Когда выбрана команда *CLAMP*, состояние всех сигналов, поступающих с системных выходных выводов, должно полностью определяться данными, хранящимися в регистре граничного сканирования. (Например, эти данные могут быть сдвинуты в регистр граничного сканирования путем предварительного использования команды *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ*.)

в) Состояния регистров параллельного вывода или защелок в ячейках регистра граничной развертки, расположенных на системных выходных выводах (2-state, 3-state или двунаправленный), не должны изменяться, пока *выбрана команда*CLAMP .

г) Когда выбрана команда *CLAMP*, системная логика на кристалле должна управляться таким образом, чтобы она не могла быть повреждена в результате сигналов, принятых на системный вход или выводы системных тактовых импульсов.

ПРИМЕЧАНИЕ — Этого можно достичь, переведя встроенную системную логику в состояние сброса или “удержания”, пока выбрана команда *CLAMP* .

Разрешения

е) Двоичное значение (значения) для команды *CLAMP* может быть выбрано разработчиком компонентов.

8.11.2 Описание

Во время тестирования конкретной микросхемы или кластера микросхем на загруженной печатной плате может потребоваться установить статические “защитные” значения для сигналов, которые управляют работой логики, не участвующей в тестировании, например, перевести ее в состояние, в котором она не может реагировать на сигналы, полученные от тестируемой логики. Такие случаи, несомненно, будут возникать во время перехода от внутрисхемного тестирования к тестированию, которое в значительной степени основано на граничном сканировании. В таких случаях значения “охранного” сигнала будут поддерживаться во время применения теста.

Для этой цели можно было бы использовать инструкцию *EXTEST*. Эта инструкция будет загружаться последовательно в микросхемы, которые управляют сигналами, для которых требуются ”защитные" значения. Требуемые значения сигналов будут загружаться как часть полного последовательного потока данных, перемещаемого в канал на уровне платы, как в начале теста, так и при каждом вводе нового тестового шаблона. Ограничение этого подхода заключается в том, что длина шаблона данных, подлежащего сдвигу для каждого теста, увеличивается за счет включения регистров сканирования границ в микросхемы, участвующие в процессе “защиты”. В результате снижается частота применения теста.

Дополнительная инструкция *CLAMP* позволяет применять “защитные” значения с использованием регистров граничного сканирования соответствующих микросхем, но не сохраняет эти регистры в последовательном контуре во время тестового применения. В случае, в котором для создания “защиты” использовалась инструкция *CLAMP*, будет использоваться следующий процесс:

ПРИМЕЧАНИЕ—В следующем описании предполагается, что каждый компонент реализует необязательную инструкцию *CLAMP*.

a) Перед тестированием во все микросхемы будет загружена команда *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ*, которая будет обеспечивать “защитные” сигналы во время предстоящего теста. Назовем эту группу микросхем *G*. Если данные тестовой настройки требуются в микросхемах, а не в *G* (т.е. в тех микросхемах, которые будут активно участвовать в предстоящем тестировании), то в это время в эти микросхемы также может быть загружена команда *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ*

b) Переместите шаблон “охранения” во все соответствующие ячейки регистра сканирования границ микросхем в *G*. Также загружаются любые тестовые установочные данные, необходимые для тестирования микросхем.

в) С этого момента и до завершения теста каждый раз, когда необходимо сканировать инструкции в устройствах, расположенных на плате, вводите инструкцию *CLAMP* в микросхемы через *G*. До тех пор, пока команда *CLAMP* поддерживается в качестве активной команды в микросхемах *G*, значения выходных сигналов этих микросхем будут определяться “защитными” данными в их регистрах граничного сканирования. Кроме того, в результате использования инструкции *CLAMP* во всех микросхемах из *G* на протяжении теста выбираются обходные регистры; таким образом, они вносят очень небольшой вклад в общее время тестирования.

8.12 Инструкции по регистрации идентификации устройства

Использование дополнительного регистра идентификации устройства позволяет последовательно считывать код с компонента, который показывает

a) Идентификационный номер производителя;

b) Номер детали; и

в) Номер версии детали.

В этом стандарте определены две инструкции, использующие регистр идентификации устройства: *IDCODE* и *USERCODE*. Эти инструкции определены в 8.13 и 8.14. Использование инструкции *IDCODE* предоставит информацию о базовом компоненте, в то время как использование инструкции *USERCODE* предоставит информацию о конкретном программировании встроенного программируемого компонента (например, логического устройства с программируемым предохранителем).

8.13 Инструкция по *ИДЕНТИФИКАЦИОННОМУ КОДУ*

8.13.1 Технические характеристики

Правила

а) Если в конструкцию включен регистр идентификации устройства, компонент должен предоставлять инструкцию по *идентификационному КОДУ*.

б) Команда *IDCODE* должна выбирать *только* регистр идентификации устройства, который должен быть подключен для последовательного доступа между TDI и TDO в состоянии контроллера *Shift-DR* (т.е. Никакой другой регистр тестовых данных не может быть подключен последовательно к регистру идентификации устройства).

в) Когда выбрана команда *IDCODE*, идентификационный код поставщика должен быть загружен в регистр идентификации устройства на переднем крае TCK после перехода в состояние *контроллера* Capture-DR.

г) Когда выбрана команда *IDCODE*, все регистры тестовых данных, которые могут работать в системном режиме или в режимах тестирования, должны выполнять свои системные функции.

е) Когда выбрана команда *IDCODE*, работа тестовой логики не должна влиять на работу системной логики на кристалле.

Разрешения

f) Двоичное значение (значения) для инструкции *IDCODE* может быть выбрано разработчиком компонентов.

8.13.2 Описание

Когда регистр идентификации устройства включен в конструкцию компонента, команда *IDCODE* принудительно вводится в параллельные выходные защелки регистра команд во время состояния контроллера *Test-Logic-Reset*. Это позволяет выбирать регистр идентификации устройства путем манипулирования широковещательными сигналами TMS и TCK, а также а также с помощью обычной операции сканирования регистра команд.

Важность этого средства выбора доступа к регистру идентификации устройства заключается в том, что оно позволяет проводить слепой опрос компонентов, собранных на печатной плате, и т.д. Таким образом, в обстоятельствах, когда набор компонентов может варьироваться (например, из-за различного программирования программируемых деталей), можно определить, какие компоненты присутствуют в продукте.

8.14 Инструкция по *ПОЛЬЗОВАТЕЛЬСКОМУ КОДУ*

8.14.1 Технические характеристики

Правила

а) Если в конструкцию включен регистр идентификации устройства и компонент может быть запрограммирован пользователем таким образом, что программирование не может быть определено иным образом с помощью общедоступных инструкций (см. 8.2), компонент должен предоставлять команду *пользовательского КОДА*.

б) Команда *пользовательского КОДА* должна выбирать регистр идентификации устройства, который должен быть подключен для последовательного доступа между TDI и TDO в состоянии контроллера *Shift-DR* (т.е. Никакой другой регистр тестовых данных не может быть подключен последовательно к регистру идентификации устройства).

в) Когда выбрана команда *USERCODE*, 32-разрядный программируемый пользователем идентификационный код должен быть загружен в регистр идентификации устройства на переднем фронте TCK после перехода в состояние контроллера *Capture-DR*.

г) Когда выбрана команда *USERCODE*, все регистры тестовых данных, которые могут работать в системном или тестовом режимах, должны выполнять свои системные функции.

e) Когда выбрана команда *USERCODE*, работа тестовой логики не должна влиять на работу системной логики на кристалле.

Разрешения

f) Двоичное значение (значения) для команды *USERCODE* может быть выбрано разработчиком компонентов.

8.14.2 Описание

Инструкция *КОД ПОЛЬЗОВАТЕЛЯ* позволяет загружать и извлекать для проверки программируемый пользователем идентификационный код. Эта инструкция требуется только для программируемых компонентов, программирование в которых не может быть определено с помощью тестовой логики. Инструкция позволяет определить запрограммированную функцию компонента.

8.15 Инструкция по *HIGHZ*

Использование дополнительной команды *HIGHZ* переводит компонент в состояние, в котором *все* его системные логические выходы находятся в неактивном состоянии привода (например, с высоким сопротивлением). В этом состоянии внутрисхемная тестовая система может подавать сигналы на соединения, обычно управляемые выходом компонента, без риска повреждения компонента.

При предоставлении инструкции по *HIGHZ* применяются следующие правила.

ПРИМЕЧАНИЕ—После использования инструкции *HIGHZ* системная логика на кристалле может находиться в неопределенном состоянии, которое будет сохраняться до тех пор, пока не будет применен системный сброс. Следовательно, встроенную системную логику, возможно, потребуется сбросить при возврате к нормальной (т. е. нетестовой) работе.

8.15.1 Технические характеристики

Правила

а) Команда *HIGHZ* должна выбрать байпасный регистр, который будет подключен для последовательного доступа между TDI и TDO в состоянии контроллера *Shift-DR*.

ПРИМЕЧАНИЕ —Регистр обхода будет вести себя полностью так, как определено в пункте 10, пока выбрана команда *HIGHZ*. Следовательно, он загрузит логический 0 во время состояния контроллера *Capture-DR* и переместит данные во время состояния контроллера *Shift-DR*.

б) Когда выбрана команда *HIGHZ*, все логические выходы системы (включая выходы с 2 и 3 состояниями и двунаправленные контакты) компонента должны быть немедленно переведены в состояние неактивного привода.

Примечания

1—Если таковые имеются, цепи подтягивания, опускания и удержания не требуется отключать.

2—В соответствии с правилами, приведенными в разделе 11.2.1, не должно происходить никаких последовательных изменений в состояниях регистров параллельного вывода или защелок в ячейках регистра граничного сканирования. Например, при выходе из команды *HIGHZ* и выборе команды *EXTEST* данные, хранящиеся в регистре граничного сканирования до выбора команды *HIGHZ*, должны быть применены к выводам системного вывода.

в) Когда выбрана команда *HIGHZ*, системная логика на кристалле должна управляться таким образом, чтобы она не могла быть повреждена в результате сигналов, принятых на системный вход или выводы системных тактовых импульсов.

ПРИМЕЧАНИЕ —Этого можно достичь, переведя встроенную системную логику в состояние сброса или “удержания”, пока выбрана команда *HIGHZ*
.

Разрешения

d) Двоичное значение (значения) для команды *HIGHZ* может быть выбрано разработчиком компонентов.

8.15.2 Описание

На платах, где не все компоненты совместимы с этим стандартом, по-прежнему будет существовать потребность в использовании методов внутрисхемного тестирования, при которых тестовые сигналы от системы ATE подаются во внутренние соединения собранной платы. Чтобы это можно было сделать без риска повреждения компонентов, которые обычно управляют этими подключениями, компоненты должны быть сконструированы таким образом, чтобы их выходные контакты системной логики могли быть переведены в состояние неактивного привода во время проведения внутрисхемного тестирования. В компоненте, совместимом с этим стандартом, предоставление инструкции *HIGHZ* позволяет ввести такое состояние с помощью крана. (На компонентах, которые не соответствуют этому стандарту, это обычно достигается с помощью специального тестового контрольного вывода.)

Дальнейшее использование инструкции *HIGHZ* заключается в том, чтобы разрешить подключение источника тестовых данных к одному или нескольким сигналам, внутренним для загруженной платы, вместо обычных драйверов. Пример применения показан на рис. 8-10.

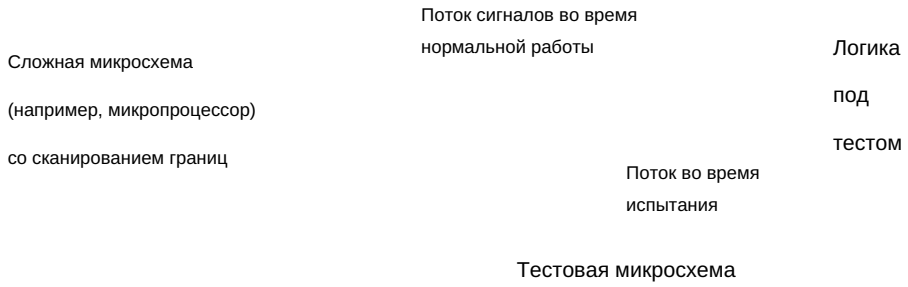


Рисунок 8-10 —Использование *инструкции* по **HIGHZ**

Во время нормальной работы выходы тестируемого чипа находились бы в неактивном состоянии привода (например, с высоким сопротивлением), в то время как выходы процессора были бы активны. Во время тестирования *HIGHZ* команда вводится в микросхему процессора, в результате чего его выходы переходят в неактивное состояние накопителя. Затем тестовый чип может быть включен для подключения к тестируемой логике (которая может, например, представлять собой массив микросхем памяти).

Обратите внимание, что там, где системным требованием является выходной вывод с двумя состояниями и активно управляются оба логических состояния, необходимо будет предоставить буфер с тремя состояниями исключительно для того, чтобы разрешить переход в неактивное состояние при выборе команды *HIGHZ* . Разрешающий ввод в этот буфер будет подаваться непосредственно из декодера команд, как показано на рис. 8-11. (На рисунке 8-11 сигнал от декодера команд будет представлять собой логику 1, отличную от той, когда выбрана команда *HIGHZ*.) В этом сигнальном тракте не требуется ячейка регистра пограничного сканирования.

Для вывода с двумя состояниями, где активно управляется только одно состояние, выходной сигнал должен быть переведен в неактивно управляемое состояние при выборе команды *HIGHZ*. Например, выходной понижающий транзистор для выхода с открытым коллектором был бы принудительно выключен.

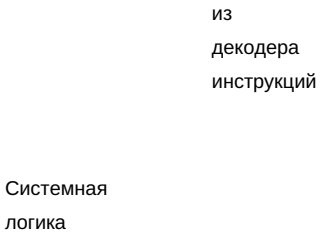


Рисунок 8-11—Положение *HIGHZ* на выводе с двумя состояниями

9. Регистры тестовых данных

Архитектура тестовой логики содержит минимум два регистра тестовых данных — регистры обхода и граничного сканирования
. Кроме того, определена конструкция третьего, необязательного регистра тестовых данных —
регистра идентификации устройства.

Архитектура расширяется за пределы минимальных требований, указанных в этом стандарте, чтобы обеспечить доступ к
любым функциям поддержки тестирования, встроенным в проект. Эти функции могут включать сканирование, самопроверку регистров,
или доступ к ключевым регистрам в проекте (например, через сканируемые теневые регистры). Дополнительные регистры тестовых данных
необязательно должны предназначаться для публичного доступа и использования.

Каждый именованный регистр тестовых данных имеет фиксированную длину, и к нему можно получить доступ с помощью одной или нескольких инструкций.
Регистры могут, при необходимости, совместно использовать схему и могут быть объединены для формирования дополнительных регистров при условии,
что каждой отдельной комбинации присваивается новое имя (что позволяет ей соответствовать требованию фиксированной длины).

Этот раздел определяет общие требования к дизайну для всех регистров тестовых данных, включенных в логику тестирования
архитектура, определенная настоящим стандартом. Конкретные конструктивные требования к регистрам обхода, граничного сканирования и
идентификации устройства содержатся в разделах 10, 11 и 12 соответственно.

9.1 Предоставление регистров тестовых данных

9.1.1 Технические характеристики

Правила

a) Группа регистров тестовых данных должна включать, как минимум, обходной регистр и регистр граничного сканирования,
разработанный в соответствии с требованиями, содержащимися в этом пункте, а также в пункте 10 и
пункте 11 соответственно.
б) Если регистр идентификации устройства включен в группу сканируемых регистров тестовых данных, он должен
быть разработан в соответствии с требованиями, содержащимися в этом пункте и в пункте 12.
c) Все регистры данных испытаний должны быть сконструированы в соответствии с требованиями, содержащимися в этом разделе.

Разрешения

d) В группе регистров тестовых данных могут быть предоставлены регистры тестовых данных для конкретной разработки, чтобы предоставить
доступ к специфичным для проекта функциям тестируемости.
e) Регистры тестовых данных для конкретной конструкции могут (но не обязательно) быть общедоступными.

9.1.2 Описание

На рисунке 9-1 показаны регистры обхода, граничного сканирования и дополнительные регистры тестовых данных, реализованные в виде набора элементов на основе сдвиговых регистров, соединенных параллельно между общим последовательным входом и общим последовательным выходом.
Выбор регистра, который формирует последовательный путь в данный момент времени, управляется из регистра команд.
На рисунке 9-1 показано, что это достигается с помощью мультиплексора; однако
возможны и другие реализации.

Рисунок 9-1 —Реализация группы регистров тестовых данных

Регистры, показанные на рис. 9-1, кратко описаны следующим образом:

Обходной регистр

Это обеспечивает одноразрядное последовательное соединение через схему, когда ни один из других регистров тестовых данных не выбран. Этот регистр может, например, использоваться для передачи тестовых данных через конкретное устройство к другим компонентам продукта, не влияя на нормальную работу конкретного компонента. Спецификация обходного регистра содержится в разделе 10.

Регистр сканирования границ

Это позволяет тестировать межсоединения платы, обнаруживая типичные производственные дефекты, такие как разрывы, короткие замыкания и т.д. Это также обеспечивает доступ ко входам и выходам компонентов при тестировании их системной логики или выборке сигналов, проходящих через системные входы и выходы. Спецификация регистра граничного сканирования содержится в разделе 11.

Регистр идентификации устройства

Это дополнительный регистр тестовых данных, который позволяет
определить производителя, номер детали и вариант компонента. Если этот реестр включен, он должен соответствовать спецификации, содержащейся в пункте 12.

Регистрируются тестовые данные для конкретной конструкции

Этих регистров могут быть предоставлены, чтобы разрешить доступ к дизайн-специфический тест, поддерживают функции в Инте-
тертый цепи, таких как самопроверка, сканирование дорожек и т. д. Они не обязательно должны предназначаться для публичного использования и доступа, но могут быть сделаны таковыми по желанию разработчика компонентов.

9.2 Проектирование и построение регистров тестовых данных

9.2.1 Технические характеристики

Правила

а) Каждому регистру данных испытаний должно быть присвоено уникальное имя.
б) Конструкция каждого регистра тестовых данных должна быть такой, чтобы при перемещении данных через него данные, применяемые к TDI, отображались без инверсии в TDO после соответствующего количества переходов TCK, когда контроллер TAP находится в состоянии *Shift-DR*.
в) Длина каждого регистра тестовых данных должна быть фиксированной, независимо от инструкции, с помощью которой осуществляется доступ к нему.
г) Для программируемых компонентов длина каждого регистра тестовых данных не должна зависеть от способа программирования компонента.

Разрешения

е) Регистр тестовых данных может быть сконструирован из сегментов или схем, также используемых в одном или нескольких других регистрах, при условии, что результирующая конструкция полностью соответствует правилам настоящего стандарта.

ПРИМЕЧАНИЕ —Результирующей комбинации должно быть присвоено имя, отличное от названия регистров, из которых она сконструирована.

е) Схемы (включая пути следования регистров сдвига) в различных регистрах тестовых данных, включенных в конструкцию, могут совместно использоваться регистрами тестовых данных при условии соблюдения правил, содержащихся в настоящем стандарте.
g) Если это специально не запрещено настоящим стандартом, схемы, содержащиеся в регистрах тестовых данных, могут использоваться для выполнения системных функций, когда тестовая операция не требуется.

9.2.2 Описание

Хотя примеры реализаций, содержащиеся в этом стандарте, показывают, что различные регистры тестовых данных являются отдельными физическими объектами, схемы могут совместно использоваться регистрами тестовых данных при условии соблюдения правил, содержащихся в этом стандарте. Например, это позволило бы регистру идентификации устройства и регистру граничного сканирования совместно использовать этапы сдвигового регистра, и в этом случае требования настоящего стандарта удовлетворялись бы за счет работы общей схемы в двух различных режимах - режиме регистра идентификации устройства и режиме регистра граничного сканирования. За исключением специально оговоренных случаев, регистры тестовых данных также могут выполнять системные функции и, таким образом, быть частью встроенной системной логики, когда они не требуются для выполнения тестовых функций.

Правило 9.2.1 c) требует, чтобы длина любого регистра тестовых данных была фиксированной; т. е. Регистр тестовых данных с именем FRED всегда должен содержать, скажем, 20 этапов, независимо от того, как и когда к нему осуществляется доступ. Обратите внимание, что разрешены виртуальные регистры (т. е. Именованный регистр тестовых данных может быть создан на основе схемы, совместно используемой с другими регистрами тестовых данных, или

с системной логикой). Следовательно, могут существовать требования к различным сегментам единого физического регистра
, к которым должен быть доступен доступ для различных тестов. В этих случаях правило 9.2.1 c) может быть выполнено путем присвоения уникального имени каждой различимой конфигурации регистра (см. Рисунок 9-2 и таблицу 9-1). Правило 9.2.1 d) требует, чтобы длина
регистра тестовых данных также была фиксированной независимо от способа программирования компонента. Эти ограничения
необходимы, чтобы избежать ненужного усложнения программного обеспечения, используемого для создания и применения тестов.

Рисунок 9-2—Построение регистров тестовых данных на основе совместно используемой схемы

Таблица 9-1—Наименования регистров тестовых данных, имеющих общую схему

Регистрационное наименование тестовых данных	Этапы, формирующие реестр
WHOLE_REG	5, 4, 3, 2, 1, 0
FRONT_REG	2, 1, 0
BACK_REG	5, 4, 3

9.3 Работа с регистратором тестовых данных

9.3.1 Технические характеристики

Правила

а) Каждая команда должна идентифицировать регистр тестовых данных, который будет последовательно подключен между TDI и TDO.

б) Регистр тестовых данных, подключенный между TDI и TDO, должен сдвигать данные на одну ступень в сторону TDO после каждого возрастающего фронта TCK в состоянии контроллера *Shift-DR*.

в) В состоянии контроллера *Test-Logic-Reset* все регистры тестовых данных должны быть настроены таким образом, чтобы они либо выполняли свою системную функцию (если таковая существует), либо не мешали работе встроенной системной логики.

г) Если требуется регистр тестовых данных для загрузки данных с параллельного ввода в ответ на текущую команду, эти данные должны загружаться на переднем фронте TCK после перехода в состояние *контроля с* захватом-DR
.

е) Регистры тестовых данных, предназначенные для передачи данных за пределы микросхемы, должны быть сконструированы таким образом, чтобы выходы компонентов изменялись только

1) На падающем фронте TCK после перехода в состояние контроллера *Update-DR*, *Update-IR*, *Run-Test/Idle* или *Test-Logic-Reset* в результате подачи сигналов на TCK и TMS; или

2) Сразу после перехода в состояние *Тестовой логики-сброса* контроллера в результате применения логического 0 при TRST*.

ПРИМЕЧАНИЕ — Для этого может потребоваться, чтобы регистр был снабжен заблокированным параллельным выходом.

е) Если для работы регистра тестовых данных требуется команда *RUNBIST*, требуемая операция должна выполняться в состоянии *Run-Test/Idle* контроллера.

ж) Если в данном состоянии контроллера не требуется никакой операции с выбранным регистром тестовых данных в ответ на текущую команду, регистр должен сохранять свое последнее состояние неизменным.

з) Регистры тестовых данных, которые не выбраны текущей командой, должны быть установлены таким образом, чтобы они либо соответствовали своей системной функции (если таковая существует), либо не мешали работе встроенной системной логики.

Разрешение

я) В дополнение к регистру тестовых данных, включенному для переключения между TDI и TDO, команда может выбирать дополнительные регистры тестовых данных.

ПРИМЕЧАНИЕ —Эти регистры должны сохранять свое последнее состояние в состоянии контроллера *Shift-DR*, но в остальном соответствовать правилам, изложенным выше.

9.3.2 Описание

Эти требования гарантируют правильную работу регистров тестовых данных совместно с контроллером ОТВОДОВ.

Обратите внимание, что, хотя команда может выбирать тестовую операцию для более чем одного регистра тестовых данных, между TDI и TDO может быть только один регистр тестовых данных. Все остальные выбранные регистры тестовых данных сохраняют свое состояние в состоянии контроллера *Shift-DR*. Одна из возможностей использования этой возможности заключается в следующем.

Рассмотрим случай, когда необходимо последовательно получить доступ к нескольким регистрам тестовых данных, чтобы установить начальные условия для теста или изучить результаты теста. В качестве примера, в таблице 9-2 показана последовательность событий (начиная с состояния контроллера *Тестовая логика-Сброс*), которая потребовалась бы, если бы для выполнения инструкции требовался доступ к двум специфичным для проекта регистрам тестовых данных и регистру граничного сканирования. На рисунке 9-3 показана конструкция группы регистров тестовых данных для этого примера.

Таблица 9-2—Последовательный доступ к регистрам тестовых данных

Шаг	Экшен
0	Тестовая логика неактивна в состоянии контроллера *Test-Logic-Reset* .
1	Введите инструкцию, которая выбирает регистр A для соединения между TDI и TDO.
2	Отсканируйте требуемые начальные значения в регистр A.
3	Введите инструкцию, которая выбирает регистр B для соединения между TDI и TDO, а также сохраняет регистр A в тестовом режиме работы.
4	Отсканируйте требуемые начальные значения в регистр B. Регистр A сохраняет свое состояние.
5	Введите команду тестирования, которая выбирает тестовую работу регистров A и B и подключает регистр пограничного сканирования между TDI и TDO.
6	Отсканируйте требуемые значения для входов и выходов компонентов в регистре граничного сканирования . Регистры A и B сохраняют свои состояния.
7	Выполните инструкцию, войдя в состояние *Run-Test/Idle* контроллера.
8	Введите инструкцию, которая выбирает регистр B для подключения между TDI и TDO, а также сохраняет регистр A в тестовом режиме работы.
9	Отсканируйте результаты теста из регистра B. Регистр A сохраняет свое состояние.
10	Введите инструкцию, которая выбирает регистр A для соединения между TDI и TDO.
11	Отсканируйте результаты теста из регистра A.

Рисунок 9-3 —Пример конструкции, содержащей два дополнительных регистра тестовых данных

На шаге 4 требуется последовательный доступ к регистрации B, чтобы установить ее начальное состояние, необходимое для выполнения теста. Однако регистр A был установлен в требуемое начальное состояние на шаге 2, и поэтому необходимо спроектировать логику тестирования таким образом, чтобы регистр A мог сохранять свое состояние между шагами 2 и 7 (когда выполняется самотестирование), пока осуществляется доступ к регистру B и регистру граничного сканирования. Аналогично, логика тестирования должна быть спроектирована таким образом, чтобы регистр B сохранял начальное условие, установленное во время шага 4 до шага 7, и таким образом, чтобы обратная последовательность событий могла произойти после завершения выполнения самотестирования.

Обратите внимание, что структура тестовой логики может быть такой, что доступ к регистрам тестовых данных должен осуществляться в фиксированном порядке для достижения желаемого результата. Например, логика тестирования может не позволить регистру B сохранять свое состояние во время сканирования регистра A.

10. Обходной регистр

Байпасный регистр обеспечивает последовательный путь минимальной длины для передачи тестовых данных между TDI и TDO. Этот путь может быть выбран, когда во время выполнения тестовой операции на уровне платы не требуется обращаться к другому регистру тестовых данных . Использование байпаса зарегистрироваться в компоненте скорости доступа для проверки данных регистров в другие компонент на уровне совета директоров тестовые данные путем.

10.1 Конструкция и эксплуатация обходного регистра

10.1.1 Технические характеристики

Правила

а) Байпасный регистр должен состоять из одной ступени сдвигового регистра.

б) Когда обходной регистр выбран для включения в последовательный канал между TDI и TDO с помощью текущей команды, каскад сдвигового регистра должен быть установлен на логический ноль на переднем фронте TCK после перехода в состояние контроллера *Capture-DR*.

в) Схема, используемая для реализации каскада сдвигового регистра в байпасном регистре, не должна использоваться для выполнения какой-либо системной функции (т. е. Она должна быть выделенной частью тестовой логики).

г) Работа обходного регистра не должна оказывать влияния на работу встроенной в микросхему системы логики.

10.1.2 Описание

Регистр обхода может быть реализован, как показано на рисунке 10-1.

Рисунок 10-1 —Реализация обходного регистра

Предоставление этого регистра позволяет обходить сегменты регистра данных последовательного тестирования на уровне платы, которые не требуются для конкретного теста. Сокращается время тестового доступа к интересующим сегментам.

В качестве примера рассмотрим печатную плату, содержащую 100 интегральных схем, каждая из которых имеет по 100 битов в регистре граничной развертки. Контур сканирования границ на собранной плате включал бы 10 000 ступеней сдвигового регистра , если бы все сегменты были соединены последовательно одновременно. Это привело бы к длительному тестированию, например, при доступе только к одной из интегральных схем на пути.

Возможность обходить сегменты пути сдвигового регистра под управлением соответствующего регистра команд позволяет значительно сократить общий путь в таких обстоятельствах. Продолжая пример, 99 из компонентов могут быть настроены на сдвиг только через их байпасный регистр, при этом тестируемая интегральная схема имеет в схеме полный регистр граничного сканирования. Это дало бы общую длину последовательного маршрута в 199 этапов — значительное сокращение по сравнению с 10 000.

Правило 10.1.1 b) включено для того, чтобы наличие или отсутствие регистра идентификации устройства в логике тестирования можно было определить путем изучения данных последовательного вывода. Регистр обхода (который выбирается при отсутствии регистра идентификации устройства) загружает логический 0 в начале цикла сканирования, тогда как регистр идентификации устройства загружает постоянную логическую 1 в свой LSB. Когда команда *IDCODE* загружается в регистр команд, последующий цикл сканирования регистра данных позволит проверить первый бит данных, смещенный из каждого компонента -логический элемент 1, показывающий, что присутствует регистр идентификации устройства. Это позволяет выполнять слепой опрос регистров идентификации устройства путем установки инструкции *IDCODE*, как описано в 12.1.

Требование команды *ОБХОДА* заключается в том, что при ее выборе системная логика на кристалле должна продолжать свою нормальную работу без помех. Правило 10.1.1 c) включено для того, чтобы это требование могло быть выполнено. Обратите внимание, однако, при условии соблюдения правила 10.1.1 d), каскад сдвиговых регистров может быть общим ресурсом, используемым несколькими регистрами, определенными настоящим стандартом, а также любым регистром тестовых данных, зависящим от конкретной конструкции.

11. Регистр сканирования границ

Регистр граничного сканирования позволяет тестировать схемы, внешние по отношению к компоненту, например, плату межсоединение или внешние компоненты, которые не соответствуют данному стандарту. Регистр также позволяет отбирать и исследовать системные сигналы, поступающие в системную логику и исходящие из нее, не вызывая

вмешательство в нормальную (нетестовую) работу встроенной системной логики. При необходимости могут поддерживаться дополнительные тестовые функции — например, тестирование встроенной системной логики.

11.1 Введение

Этот раздел определяет конструкцию регистра сканирования границ в компоненте и работу регистра в ответ на различные инструкции, определенные настоящим стандартом.

Среди регистров, требуемых этим стандартом, регистр граничного сканирования является наиболее сложным. Его сложность заключается не в его функции переключения, ни в его архитектурном размещении параллельно с другими необходимыми регистрами тестовых данных, оба из которых соответствуют правилам, изложенным в пункте 9. Сложность заключается вместо этого в способе подключения регистра к встроенной системной логике и в его работе в ответ на инструкции, определенные в разделе 8. Требования к конструкции как для подключения, так и для функциональной работы варьируются от ячейки к ячейке и определяются как типом сигнала (вводимого в встроенную системную логику или выводимого из нее), так и набором инструкций, которые должны поддерживаться.

Технические характеристики конструкции представлены в трех группах:

а) *Конструкция и функционирование регистра (11.2 и 11.3).* Структура регистра сканирования границ определена. Также представлены спецификации для работы сдвигового регистра, лежащего в основе регистра граничной развертки, и для параллельных выходных защелок или триггеров, которые требуются для некоторых каскадов сдвигового регистра.

б) *Обеспечение и эксплуатация ячеек (с 11.4 по 11.8).* Представлены правила, которые определяют, где должны быть предусмотрены ячейки регистра граничного сканирования и как они должны быть соединены между системными выводами компонента и системными входами и выходами встроенной системной логики.
Представлены дополнительные правила, которые определяют, как сигналы должны направляться через ячейки регистра пограничного сканирования в ответ на выбор различных инструкций, определенных этим стандартом.

в) *Объединение ячеек (11.9).* Наконец, указаны допустимые способы объединения ячеек регистра сканирования границ, требуемые правилами предоставления ячеек. Применение этих правил снизит стоимость внедрения регистра сканирования границ в определенных особых случаях.

Чтобы упростить изложение правил, в этом пункте используются терминология и подход, описанные в следующих подразделах.

11.1.1 Подход

Чтобы упростить представление правил в этом разделе, регистр сканирования границ будет описан так, как если бы он добавлялся к уже готовому проекту, который не соответствует этому стандарту. Такую конструкцию можно представить себе так, как показано на рис. 11-1.

Обратите внимание на следующие особенности на рисунке 11-1:

а) *Системная логика.* Это схема, которая реализует нетестовую цифровую функцию компонента.

б) *Системные контакты.* Термин “системный вывод” используется в настоящем стандарте для обозначения любого системного (т.е. нетестового) терминала, к которому может быть выполнено внешнее подключение. Для упакованных компонентов внешние соединения выполняются с контактами упаковки, как правило, с помощью процесса пайки. Однако в некоторых случаях микросхема интегральной схемы собирается непосредственно на подложке без предварительной упаковки.
В таких случаях термин “системный вывод” следует интерпретировать как точку, к которой осуществляется внешнее соединение, т.е. Соединительную площадку.

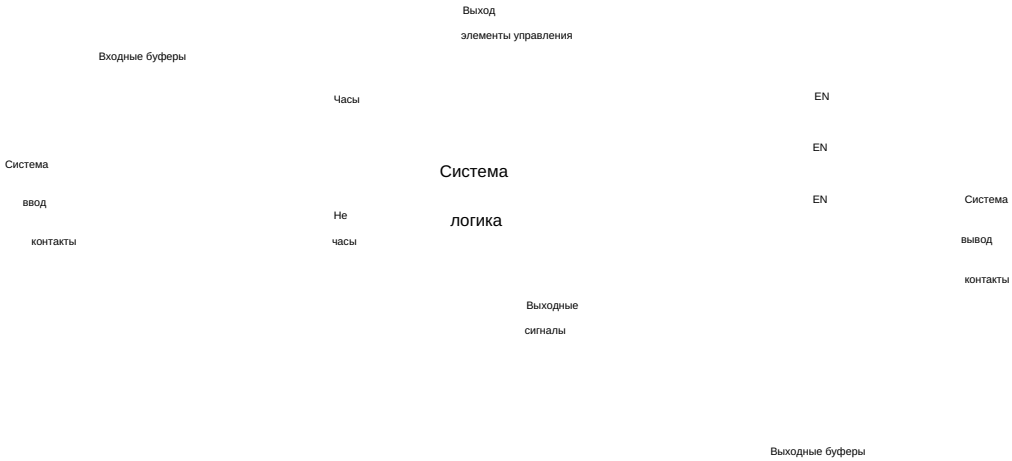


Рисунок 11-1 —Компонент без сканирования границ

c) *Буферы ввода/вывода*. Буферы подключены между системными выводами и системной логикой. Обратите внимание, что выходные буферы могут иметь управление, а также входные данные от системной логики. Сигналы, полученные на управляющих входах, определяют способ работы выходного буфера. Для примера, на рисунке 11-1 показано несколько выходных буферов с тремя состояниями, в которых входной сигнал enable (EN) используется для определения того, управляется ли выходной сигнал. Возможны другие типы выходного буфера, в котором управляющие сигналы могут использоваться для определения различных характеристик сигнала, подаваемого вне микросхемы.

d) *Входы и выходы системной логики*.

Следует различать два типа входных данных для встроенной системной логики следующим образом:

1) *Тактовые входы*. Переходы на этих входах от низкого к высокому логическому уровню (или наоборот) используются для указания, когда устройство с сохраненным состоянием, такое как триггер или защелка, может выполнить операцию. В конструкции с пограничным запуском граничные значения (переходы логического уровня), полученные на тактовых входах, используются для запуска работы всей или части системной логики на кристалле, и стационарные логические значения, полученные по этим сигналам, не имеют значения. В конструкции с чувствительностью к уровню тактовые входы используются для того, чтобы позволить устройствам хранения данных в системной логике на кристалле загружать значения данных. Обратите внимание, что значения, загружаемые в устройства с сохраненным состоянием, не определяются значениями тактовых входов.

2) *Не тактовые входы*. В эту группу входят все остальные входы встроенной системной логики. Как правило, сигналы, подаваемые на эти входы, используются для подачи данных или выбора операции, которая должна быть выполнена.

Выходы встроенной системной логики управляют выходными буферами (или, как будет обсуждаться позже, входами в блоки смешанных аналоговых/цифровых схем, внешние по отношению к системной логике). Необходимо различать два типа сигналов, которые могут подаваться в выходной буфер, следующим образом:

3) *Выводите управляющие сигналы*. В компоненте без граничного сканирования, таком, как показано на рисунке 11-1, эти сигналы будут напрямую управлять “разрешающими” входами выходных буферов и, следовательно, определять, активно ли они влияют на состояние соответствующих подключенных системных выводов.

4) *Сигналы передачи данных*. В компоненте без граничного сканирования эти сигналы будут направлять входные данные в выходные буферы (или, как будет обсуждаться позже, входные данные в блоки смешанных аналоговых / цифровых схем, внешние по отношению к системной логике).

ПРИМЕЧАНИЕ—Один выходной сигнал встроенной системной логики может подавать выходной управляющий сигнал в один выходной буфер и сигнал данных в другой.

11.1.2 Пути передачи сигналов к встроенной системной логике

Каждый путь прохождения сигнала в системную логику на кристалле считается деревом разветвлений с одной или несколькими ветвями. Сигналы поступают в дерево разветвлений по магистрали (например, из входного буфера) и выходят через ответвления (например, на входы встроенной системной логики). Например, на рисунке 11-2 показаны пути прохождения сигналов между одним системным входным выводом и несколькими входами встроенной системной логики. (Во многих случаях дерево разветвлений будет рассматриваться как содержащееся в системной логике. В этих случаях считается, что между выводом системного ввода и системной логикой присутствует только одно двухточечное соединение, то есть соединение только с одним разветвлением
.)

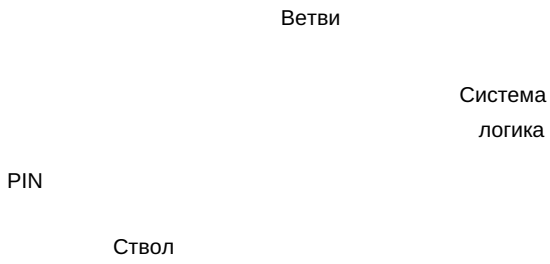


Рисунок 11-2—Входное соединение

11.1.3 Ячейка регистра пограничного сканирования

ПРИМЕЧАНИЕ —Правила, содержащиеся в этом разделе, описывают регистр пограничного сканирования как логическое устройство, а не конкретную физическую реализацию. Кроме того, для ясности представления, примеры конструкций регистровых ячеек с граничным сканированием, представленные в этом разделе, показывают, что схема отличается от схемы, используемой для создания различных других функций, определенных в этом стандарте. Однако имейте в виду, что правила этого стандарта разрешают использовать части схемы, используемые для построения ячеек регистра пограничного сканирования, в частности, каскады регистров сдвига, для реализации других функций, определенных этим стандартом, таких как регистры обхода и идентификации устройства. Там, где схема используется совместно между регистром пограничного сканирования и другими функциями, определенными настоящим стандартом, ячейки регистра пограничного сканирования могут оказаться более сложными, чем описанные здесь.

Считается, что каждая ячейка регистра граничного сканирования имеет некоторое количество терминалов передачи данных (по крайней мере, два) и количество тактовых и управляющих входов, соответствующих стилю реализации. Внутри каждой ячейки находится один каскад сдвигового регистра, который часто снабжен параллельным входом и параллельным выходом (которые могут быть заблокированы). Этот каскад сдвигового регистра использует два подключения для передачи данных ячейки в качестве последовательного входа и последовательного выхода. Посредством этих соединений ячейка связана с ячейками до и после нее в регистре граничного сканирования.

Ячейки, имеющие три терминала передачи данных, позволяют наблюдать сигналы, поступающие в системную логику на кристалле или покидающие ее, но не контролировать их. Для такой ячейки третий терминал данных функционирует как параллельный вход для каскада сдвигового регистра с параллельным входом и последовательным выходом. Когда регистр граничного сканирования выбирается в качестве последовательного пути между TDI и TDO с помощью команды, данные, присутствующие на этом терминале, загружаются в каскад сдвигового регистра на переднем фронте TCK в состоянии контроллера *Capture-DR* (т.е. Как требуется правилами пункта 9). Ячейки этого типа могут быть

так называемые ячейки “только для наблюдения”. Они будут подключены к сигнальному тракту, входящему в логику встроенной системы или выходящему из нее, как показано на рисунке 11-3.

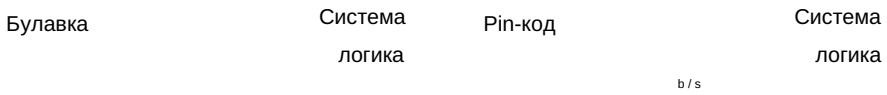


Рисунок 11-3—Подключение ячейки регистра пограничного сканирования, доступной только для наблюдения

Ячейки с четырьмя или более терминалами передачи данных вставляются в сигнальные тракты, входящие или выходящие из встроенной в микросхему системы логики, как показано для случая входа на рис. 11-4. Такие ячейки можно назвать ячейками “контроля и наблюдения”.
Каскад сдвигового регистра в ячейке управления и наблюдения может загружать значение сигнального тракта, в который они вставлены, и, следовательно, разрешать наблюдение за этим сигналом. Кроме того, при необходимости значение, хранящееся в каскаде сдвигового регистра , может быть передано на провод вместо обычного (нетестируемого) источника. В некоторых случаях постоянное значение сигнала также может подаваться на провод вместо обычного (нетестируемого) источника.

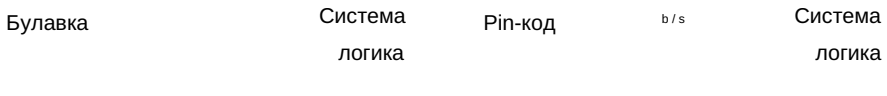


Рисунок 11-4 —Вставка ячейки регистра сканирования границ для контроля и наблюдения

Ячейки, имеющие только два терминала передачи данных, являются избыточными в том смысле, что их можно было бы исключить из конструкции без ущерба для соответствия компонента этому стандарту. Такие ячейки могут быть простыми ступенями сдвигового регистра. Как правило, они будут существовать в конструкции компонента в результате программирования или настройки ячеек регистра граничного сканирования (см. 11.8).

Правила, представленные далее в этом разделе, определяют способ вставки или добавления ячеек, требуемый при каждом типе подключения к системной логике или из нее.

Концептуальная модель ячейки регистра сканирования границ с контролем и наблюдением показана на рисунке 11-5. Такие ячейки содержат каскад сдвигового регистра с параллельным входом и параллельным выходом. Сигналы поступают в ячейку через один вывод

ячейку (называемую параллельным входом) и выходит через другую (называемую параллельным выходом). Логика (обычно один или несколько мультиплексоров) в ячейке управления и наблюдения определяет маршрутизацию сигнала с параллельного входа ячейки на параллельный выход ячейки. Сигнал может быть направлен через “параллельные” клеммы каскада сдвигового регистра с параллельным вводом, параллельным выводом или, при нормальной (нетестовой) работе компонента, логика маршрутизации устанавливается таким образом, что сигнал подается непосредственно с параллельного входа ячейки на параллельный выход без-какого-либо изменения значения сигнала.

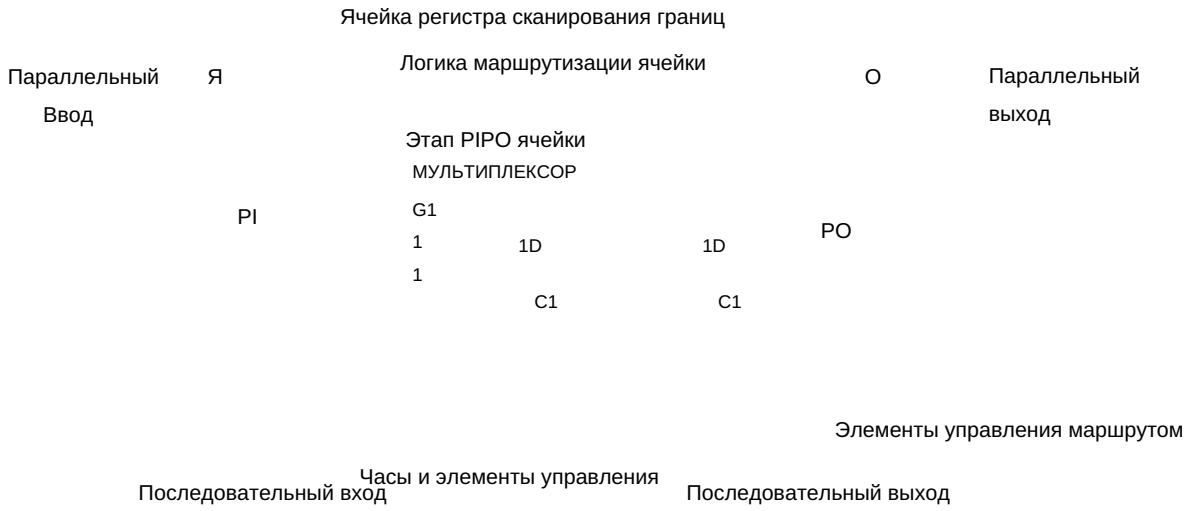


Рисунок 11-5—Концептуальный вид ячейки регистра сканирования границ для контроля и наблюдения

11.2 Конструкция регистра

11.2.1 Технические характеристики

Правила

a) Регистр граничного сканирования должен состоять только из ячеек регистра граничного сканирования, как определено в этом пункте.

b) Должно быть предусмотрено достаточное количество ячеек регистра граничного сканирования, чтобы полностью соответствовать требованиям для каждой системы подключение к системной логике на кристалле или из нее, как определено ниже в этом разделе.

в) Каждая ячейка регистра граничной развертки должна содержать один каскад регистра сдвига и должна иметь последовательный входной терминал и последовательный выходной терминал, посредством которых ячейка связана с ячейками до и после нее в регистре граничной развертки или, в случае ячеек на каждом конце регистра, с остальной частью логики тестирования, определенной настоящим стандартом.

г) Ячейка регистра граничного сканирования должна иметь два или более терминала передачи данных [включая последовательные терминалы, требуемые Правилом 11.2.1 c)].

ПРИМЕЧАНИЕ — Каждая ячейка также будет иметь несколько тактовых и управляющих входов, количество которых будет определяться стилем реализации.

е) Ячейки регистра граничного сканирования, имеющие три терминала данных, должны быть сконструированы таким образом, чтобы один терминал данных был подключен с помощью логики маршрутизации к параллельному входу каскада сдвиговых регистров в ячейке.

ПРИМЕЧАНИЕ — Такие ячейки называются ячейками “только для наблюдения”.

e) Для ячеек регистра с граничным сканированием, которые имеют четыре или более терминалов данных, терминалы данных, отличные от тех, которые используются для последовательного ввода и вывода, должны быть параллельными входами и параллельными выходами ячейки и соединяться логикой маршрутизации
   1) Друг другу; и
   2) К параллельному входу и параллельному выходу каскада сдвигового регистра.

Примечания

1—Такие ячейки называются ячейками “контроля и наблюдения”.

2—Часто, но не всегда, каскады со сдвиговым регистром требуют запертых параллельных выходов.

g) Для данного компонента порядок ячеек в регистре сканирования границ должен быть фиксированным. В частности, порядок не должен меняться в результате какой-либо операции встроенной системной логики.
h) Для данного компонента длина регистра сканирования границ должна быть фиксированной. В частности, длина не должна изменяться в результате какой-либо операции встроенной системной логики.
i) В случае, если для компонента не требуются ячейки регистра граничной развертки, должен быть предусмотрен регистр, состоящий из одной ступени сдвигового регистра.

ПРИМЕЧАНИЕ —Такая ситуация возникает, когда компонент содержит только тестовую логику, определенную или разрешенную этим стандартом. Такой компонент можно было бы описать как предназначенный для тестирования; он не будет влиять на системную функцию собранной платы.

j) Нормальная работа каких-либо системных функций не должна мешать работе ячеек регистра пограничного сканирования или прерываться ею (см. 11.3).

ПРИМЕЧАНИЕ — При условии соответствия правилу 11.2.1 k), схема может быть разделена между регистром граничного сканирования и другой частью тестовой логики. Например, ступени сдвигового регистра также могут использоваться другим регистром тестовых данных.

к) Если ячейка регистра граничной развертки имеет заблокированный параллельный выход, если система или другая тестовая функция изменяет значение заблокированного параллельного выхода (т. е. Заблокированный параллельный выход может быть общим), то значение, которое существовало бы, если бы заблокированный параллельный выход был выделен, должно быть восстановлено после ввода в *EXTEST*, *INTEST*, *CLAMP* или другую команду, требующую использования регистра граничной развертки (см. Правила 11.3.1 b) и 11.3.1 c) и описание в 11.3.2).

ПРИМЕЧАНИЕ —При условии соответствия Правилам 11.2.1 j) и 11.2.1 k), схема может быть разделена между регистром пограничного сканирования и обычной системной логикой.

Разрешения

l) Правило 11.2.1 i) может быть выполнено путем выбора обходного регистра всякий раз, когда этот стандарт требует выбора регистра граничного сканирования в качестве пути между TDI и TDO.
м) Ячейки регистра граничной развертки могут быть соединены в любом порядке.

ПРИМЕЧАНИЕ — См. Правило 11.2.1 g).

11.2.2 Описание

Регистр сканирования границ является обязательной характеристикой настоящего стандарта и должен быть включен в каждый компонент, который заявляет о соответствии настоящему стандарту. Он состоит из количества ячеек, равного количеству ступеней сдвигового регистра , содержащихся в регистре. Эти ячейки расположены вокруг встроенной системной логики компонента, как указано далее в этом разделе. Они соединены, образуя единый канал на основе сдвигового регистра, который соединяется между TDI и TDO в состоянии контроллера *Shift-DR*, когда выбрана соответствующая команда. (Что касается инструкций, определенных в этом стандарте, регистр граничного сканирования определен как последовательный путь между TDI и TDO в состоянии контроллера *Shift-DR* для *ВЫБОРКИ*, *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ*, *EXTEST*, *INTEST* и, необязательно, инструкций *RUNBIST*.)

Рисунок 11-6 иллюстрирует конструкцию части регистра сдвига регистра граничной развертки. Обратите внимание, что не все каскады снабжены запертыми параллельными выходами.

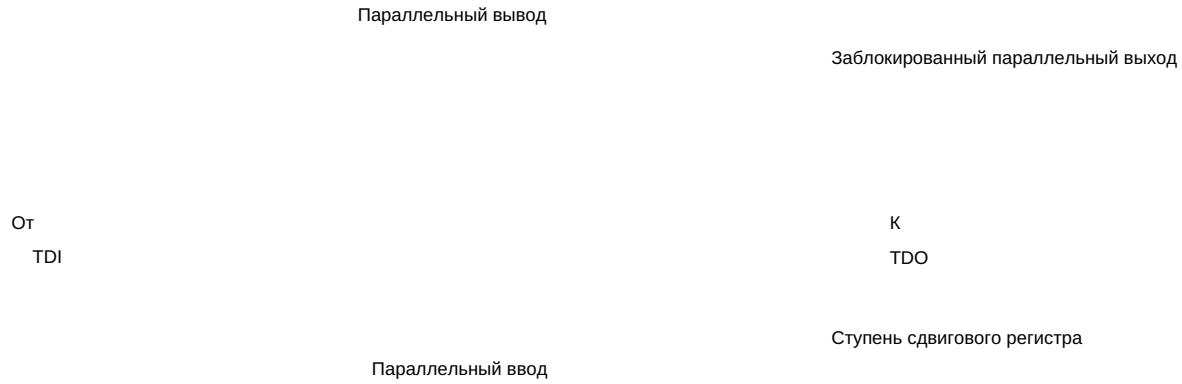


Рисунок 11-6—Схема сдвигового регистра пограничной развертки

Для данного компонента минимально допустимая длина регистра граничного сканирования зависит от количества и типа системных (нетестовых) подключений к встроенной системной логике и из нее. (Как правило, это будут подключения вне чипа.) Правила, определяющие минимальный набор ячеек, который должен быть предоставлен, представлены далее в этом разделе. Однако обратите внимание, что каждый компонент, претендующий на соответствие этому стандарту, должен иметь регистр граничной развертки, который содержит по крайней мере одну ступень сдвигового регистра.

11.3 Операция регистрации

11.3.1 Технические характеристики

Правила

а) Когда регистр граничной развертки выбран в качестве последовательного пути между TDI и TDO в состоянии контроллера *Shift-DR*, данные, введенные в TDI, должны отображаться без инверсии в TDO после применения количества парных восходящих и нисходящих фронтов в TCK, эквивалентном длине регистра граничной развертки.

б) Когда выбраны команды *EXTEST*, *INTEST* или *PRELOAD*, заблокированные параллельные выходы сдвигового регистра пограничного сканирования должны изменять состояние только на падающем фронте TCK в состоянии контроллера *Update-DR*, и в это время каждый должен быть установлен в состояние соответствующей ступени сдвигового регистра.

в) Когда выбраны команды *BYPASS*, *CLAMP*, *HIGHZ*, *IDCODE*, *RUNBIST* или *USERCODE*, все заблокированные параллельные выходы должны сохранять свое состояние по крайней мере до
   1) Состояние контроллера *Test-Logic-Reset* вводится в результате применения логического значения 0 при TRST*; или
   2) Первый падающий фронт TCK возникает в состоянии контроллера *Test-Logic-Reset*, когда это состояние вводится в результате подачи сигналов в TCK и TMS.

ПРИМЕЧАНИЕ—Разрешение 11.3.1 h) позволяет сбросить регистр пограничного сканирования после входа в *состояние ler для сброса* тестовой логики-управления-, и поэтому в случае, когда такие детали реализации неизвестны, следует предположить, что состояния заблокированных параллельных выходов регистра пограничного сканирования неизвестны после ввода этого состояния.

г) Не должно налагаться никаких ограничений на количество системных выходных контактов, которые могут изменять состояние в одном состоянии контроллера *Update-DR*, *Update-IR* или *Test-Logic-Reset* в результате работы тестовой логики; также не должны налагаться ограничения на шаблоны данных, которые могут управляться (например, максимальное ограничение на количество системных логических выходов, которые могут управлять логикой 1).

ПРИМЕЧАНИЕ —Разработчикам следует учитывать, что в тестовом режиме проверка границ может манипулировать несколькими шинами способами, которые никогда не могли бы произойти при нормальной работе системы. Это может привести к тому, что потребление энергии в переходном или установившемся режиме превысит ожидаемое для нормальной работы системы. Будьте особенно внимательны к этому при макетировании, подключении к розетке или монтаже компонентов, совместимых с этим стандартом, поскольку в этих средах распределение мощности может быть неоптимальным.

е) Не должно устанавливаться никаких ограничений на комбинации логических значений, которые могут быть перенесены в регистр граничного сканирования.

ПРИМЕЧАНИЕ—Во время тестовых операций комбинации сигналов могут подаваться либо на встроенную системную логику, либо вне чипа, которые не будут возникать при нормальной работе компонента. Это, в частности, тот случай, когда несколько ячеек регистра с граничным сканированием используются для управления сигналами, которые обычно поступают из одного источника [см. Правила 11.5.1 c), 11.5.1 d) и 11.6.1 e)].

Разрешения

е) Когда выбрана команда *SAMPLE*, заблокированные параллельные выходы регистра сдвига пограничного сканирования могут изменять состояние только на падающем фронте TCK в состоянии контроллера *Update-DR*, в это время каждый из них должен быть установлен в состояние соответствующего каскада регистра сдвига.

ж) Задержка между падающим фронтом TCK и последующими изменениями на выводах системного выхода может быть намеренно смещена между выходами системы, например, из-за необходимости избегать одновременного переключения на нескольких или всех системных выходах.

ПРИМЕЧАНИЕ —В случае примеров реализаций, показанных в этом стандарте, этот перекос может быть добавлен путем введения небольших задержек в тактовом тракте UpdateDR и в сигналах управления *Mode-n* в каждую ячейку регистра граничного сканирования. Такой перекос может быть необходим как для ячеек регистра граничного сканирования, которые подают сигналы данных, так и для тех, которые подают выходные управляющие сигналы. В последнем случае добавленный перекос предотвратит чрезмерный текущий спрос из-за одновременных изменений с “отключить” на “активный” или наоборот.

з) Если ячейки регистра граничной развертки снабжены каскадами сдвигового регистра с запертыми параллельными выходами , эти выходы могут быть сброшены в любое логическое состояние (0 или 1)

1) Когда состояние контроллера *Test-Logic-Reset* вводится в результате применения логического 0 при TRST*; или

2) На первом падающем фронте TCK в состоянии контроллера *Test-Logic-Reset*, когда это состояние вводится в результате подачи сигналов в TCK и TMS.

11.3.2 Описание

Чтобы соответствовать требованиям инструкций по *ОБРАЗЦУ* и *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКЕ*, должна быть предусмотрена возможность перемещения данных через регистр граничного сканирования без вмешательства в нормальную системную работу компонента.
Этого можно достичь, сделав этапы сдвигового регистра, используемые регистром граничного сканирования, выделенной частью тестовой логики; то есть они не выполняют никакой системной функции. В качестве альтернативы, ступени сдвигового регистра могут быть общим ресурсом, используемым несколькими регистрами, определенными настоящим стандартом, и любым регистром данных тестирования, зависящим от конкретной конструкции.
……

Напротив, для общедоступных инструкций, определенных этим стандартом, запертый параллельный вывод, требуемый на некоторых этапах регистрации граничного сканирования, управляется таким образом, что он сохраняет свое последнее состояние всякий раз, когда регистр пограничного сканирования не выбран текущей командой. Эта последняя функция позволяет сдвигать набор значений данных в регистр и помещать их на различные заблокированные параллельные выходы с помощью команды *ПРЕДВАРИТЕЛЬНОЙ ЗАГРУЗКИ*. Затем эти значения будут сохраняться до тех пор, пока они не будут изменены в конце последующего процесса переключения (т.е. В состоянии контроллера *Update-DR*). Таким образом, когда вводится команда, которая выбирает данные, хранящиеся в регистре пограничного сканирования, для извлечения из компонента (например, команда *EXTEST*), ранее загруженные данные немедленно доступны для использования. (Смотрите обсуждение в разделе 8.7.)

11.4 Общие правила, касающиеся предоставления сотовых

11.4.1 Спецификация

Правила

а) На каждом системном входе или выходе системной логики на кристалле должна быть предусмотрена одна или более ячеек регистра граничного сканирования, как описано в 11.5 и 11.6.

б) Для компонентов, которые содержат аналоговые схемы, внешние по отношению к встроенной системной логике (т.е. аналоговые схемы, имеющие внекристаллитные соединения, как показано на рисунке 11-7), соединения между встроенными аналоговыми схемами и встроенной системной логикой должны рассматриваться как внекристаллитные соединения.

Рисунок 11-7—Компонент, содержащий аналоговую схему

в) Ячейки регистра сканирования границ не должны предоставляться в
   1) Контакты ОТВОДА (TCK, TDI, TDO, TMS и TRST*);
   2) Контакты, обеспечивающие соответствие данному стандарту (см. 4.8); или
   3) Нецифровые контакты (например, силовые и аналоговые контакты).

ПРИМЕЧАНИЕ—Дифференциальные преобразователи и приемники, которые работают за счет определения направления протекания тока, считаются “аналоговыми” схемами. Следовательно, в таких случаях это правило требует размещения единственной ячейки регистра граничного сканирования между каждым дифференциальным драйвером или приемником и встроенной системной логикой, как показано на рис. 11-9. Смотрите 11.4.2 для получения информации о применении этих правил к другим типам сопряженных входов и выходов, например, к дифференциальным сигналам, которые работают с использованием обычных логических напряжений.

г) Соединение ячеек регистра граничного сканирования типа "Контроль и наблюдение" должно быть таким, чтобы, если каждая ячейка была заменена соединением с коротким замыканием между ее параллельным входом и параллельным выходом, нормальная (нетестовая) логическая работа компонента не изменялась.

ПРИМЕЧАНИЕ — Однако могут быть изменения в производительности.

е) Не должно быть никакой логики между любой ячейкой регистра пограничного сканирования и системным выводом, к которому эта ячейка подключена.

ПРИМЕЧАНИЕ — “Прозрачные” устройства, такие как буферы и буферы ввода-вывода, не считаются “логическими” и могут существовать вне регистра граничного сканирования. Преобразователи также могут существовать вне регистра граничного сканирования при условии соблюдения Правил 11.5.1 i), 11.5.1 j), 11.6.1 l), 11.6.1 m), 11.6.1 n) и 11.6.1 o). Устройства, выполняющие логическую операцию (например,

вентили, триггеры и защелки) считаются логическими устройствами и не должны размещаться за пределами регистра граничного сканирования.

Разрешения

e) Если некоторые входы или выходы не подключены к выводам пакета в конкретной конфигурации пакета компонента, для неподключенных сигналов могут быть предусмотрены ячейки регистра граничной развертки.

11.4.2 Описание

Рисунок 11-8 иллюстрирует размещение ячеек регистра граничной развертки для основных типов входных, выходных и двунаправленных выводов.

Рисунок 11-8—Расположение ячеек регистра пограничного сканирования

Ячейки регистра граничной развертки расположены таким образом, что состоянием каждого вывода цифровой системы (включая выводы синхронизации) можно управлять и/ или наблюдать с помощью регистра граничной развертки. Эти ячейки также могут позволять контролировать и наблюдать состояние логических входов и выходов системы соответственно.

Расширить конструкцию регистра граничной развертки, чтобы охватить случаи, в которых блоки аналоговых схем расположены снаружи от системной логики на кристалле, между логикой и выводами, несложно. В таких компонентах сигналы, которые формируют интерфейс между чисто цифровой схемой и блоком (блоками) смешанной аналого-цифровой схемы, считаются эквивалентными системным выводам. Следовательно, ячейки регистра граничного сканирования предусмотрены для соединений, которые поступают к смешанному аналого-цифровому блоку(блокам) или из него (см. Правило 11.4.1 b) и рисунок 11-7).

В спецификации регистра граничной развертки также рассматриваются случаи, в которых логические сигналы передаются между компонентами с помощью нецифровых или неэлектронных средств. Примерами могут служить оптические межсоединения или емкостная связь. В этих случаях драйверы и приемники считаются аналоговыми схемными блоками и размещаются вне ячеек регистра граничного сканирования для соответствующих логических сигналов.

В качестве альтернативы, читатели этого стандарта могут пожелать обратиться к архитектуре и возможностям, определенным стандартом IEEE Std 1149.4-1999 для тестовой шины смешанных сигналов. Стандарт IEEE Std 1149.4-1999 описывает схемы для передачи и измерения как непрерывных (dc), так и непостоянных (ac) сигналов на аналоговые выводы компонентов и внутреннюю логику и обратно через стандартизированный тестовый интерфейс, который является надмножеством и полностью совместим

с помощью порта тестового доступа, определенного этим стандартом. Следует также отметить, что стандарт IEEE Std 1149.4-1999 также содержит особые положения для тестирования выводов компонентов дифференциальной сигнализации.

Случай, в котором пара системных выводов используется для передачи единого логического сигнала в компонент или из него (например, на дифференциальном входе или выходе; см. Рис. 11-9), немного сложнее и заслуживает дальнейшего обсуждения.

Рисунок 11-9 —Компонент с дифференциальными входами и выходами

Как правило, “парный” ввод-вывод будет предоставляться для повышения производительности соединений между компонентами, например, для обеспечения надежной связи между компонентами в условиях шума или для уменьшения перекоса в высокопроизводительных системах. Характеристики, которые отличают парный ввод-вывод от обычных цифровых сигналов, следующие:

а) Сигналы, проходящие через пару выводов, обычно поступают из одного буфера и всегда принимаются одним буфером.
б) Сигналы всегда должны соединяться парами, если необходимо поддерживать “улучшенное” поведение (например, подавление шума) межчипового соединения.

Два типа парного ввода-вывода являются обычным явлением:

— Те, которые работают путем изменения или определения направления тока, протекающего по контуру, образованному двумя соединениями
— Например, те, которые во многих отношениях кажутся “обычными” цифровыми сигналами, потому что в них используются уровни напряжения, совместимые с TTL

Для сигналов ввода-вывода, связанных с потоком, дифференциальный входной или выходной буфер следует рассматривать как аналоговую схему. Следовательно, между каждым буфером и встроенной системной логикой должна быть размещена одна ячейка регистра граничного сканирования (рисунок 11-9).

Для парных обычных сигналов расположение ячеек регистра граничной развертки на выходных выводах будет зависеть от точных характеристик канала связи. Предлагается следующее:

— Там, где возможно независимое использование выходных сигналов (с потерей улучшенного характеристики подключения, но сохраняющие способность передавать логический сигнал), для каждого драйвера должны быть предусмотрены две ячейки, по одной на каждый выходной вывод. Это обеспечивает возможность тестирования межсоединений на уровне платы, где один из выходных сигналов от пары используется для управления “обычным” входным выводом на другом компоненте.
— В тех случаях, когда сигналы, составляющие пару, не могут использоваться независимо, между системной логикой и выходным буфером должна быть предусмотрена единственная ячейка регистрации пограничного сканирования

ister.

Для парного “обычного” ввода между буфером
и системной логикой на кристалле должна быть размещена единственная ячейка регистра пограничного сканирования, позволяющая отслеживать логический сигнал, передаваемый по паре.

11.5 Обеспечение и эксплуатация ячеек на входах системной логики

Этот подраздел содержит правила предоставления и эксплуатации ячеек регистра граничного сканирования на входах в системную логику на кристалле. Эти входы могут управляться буферами на выводах системного ввода или на выводах двунаправленной системы , когда они работают как системные входы. Альтернативно, такие входы могут управляться смешанными аналоговыми/цифровыми схемными блоками, расположенными снаружи от встроенной системной логики.

Как обсуждалось в 11.1.1, входные данные системной логики могут быть как тактовыми, так и не тактовыми. Многие из правил, представленных в 11.5.1, являются общими для обоих типов входных данных. Однако некоторые правила применяются только к одному типу ввода. Там, где это имеет место, правило, рекомендация или разрешение помечаются как “*Только для тактовых (не тактовых) входов:*.”

ПРИМЕЧАНИЕ—В этом подпункте рассматривается предоставление ячеек регистра граничного сканирования на входах встроенной системной логики в случаях, в которых эти входы управляются системными входными контактами во время нормальной (нетестовой) работы. Случай, когда вход в системную логику на кристалле управляется системным двунаправленным выводом во время обычной (нетестовой) работы, обсуждается в 11.7.

11.5.1 Технические характеристики

Правила

a) Каждый сигнал, принятый на вход системной логики (например, из входного буфера), должен быть способен наблюдаться по меньшей мере одной ячейкой регистра граничного сканирования (рисунок 11-10).

ПРИМЕЧАНИЕ— Ячейки могут быть доступны только для наблюдения или для управления и наблюдения. Где более одной контрольно-соблюдать клетки про-удостоверение личности соблюсти один сигнал, полученный в системе дискретного входа, правило 11.5.1 e) применяется. Обратите внимание, что такие дополнительные ячейки являются избыточными в том смысле, что их можно было бы исключить из конструкции без ущерба для соответствия настоящему стандарту (см. 11.8).

Рисунок 11-10—Предоставление ячейки регистра пограничного сканирования на входе системы

b) Каждая ячейка регистра пограничного сканирования, которая может наблюдать сигнал, принятый на входе системной логики, должна наблюдать точно один такой сигнал.

в) *Только для не тактовых входов:* Если поддерживается команда *INTEST*, каждый не тактовый вход *I* в системную логику на кристалле должен управляться точно из одной ячейки регистра граничного сканирования, и эта ячейка должна быть одной из тех, которые отслеживают системный входной сигнал, который управляет *I* во время нормальной (не тестовой) работы.

Примечания

1—Для тактовых входов предоставление ячейки управления и наблюдения необязательно [см. Правило 11.5.1 d) и Разрешение 11.5.1 l)].

2.Допустимо, чтобы набор ячеек регистра контроля и наблюдения с пограничным сканированием был распределен в сети разветвления например, из одного системного входного буфера в набор входов системной логики на кристалле, обычно управляемой этим буфером (см. Рисунок 11-11). В таких случаях применяется правило 11.3.1 e).

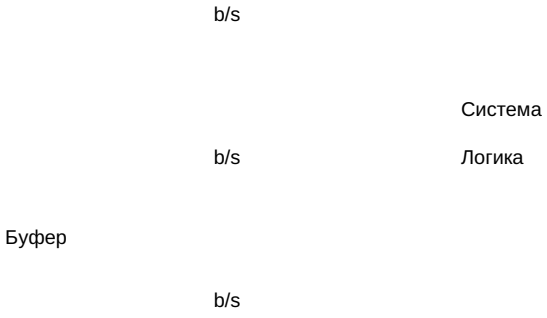


Рисунок 11-11—Предоставление нескольких ячеек регистра сканирования границ на одном входе

г) *Только для тактовых входов:* Если поддерживается команда *INTEST* и предусмотрена одна или несколько ячеек управления и наблюдения , которые отслеживают входные сигналы системных тактовых импульсов компонента, каждый вход *I* системных тактовых импульсов в системную логику на кристалле должен управляться точно из одной ячейки управления и наблюдения, и эта ячейка должна быть одной из тех, которые отслеживают входные сигналы системных тактовых импульсов, которые управляют *I* во время нормальной (не тестовой) работы.

Примечания

1—Это правило активируется при использовании опции, указанной в Разрешении 11.5.1 (l).

2.Допустимо, чтобы набор ячеек регистра контроля и наблюдения с пограничным сканированием был распределен в сети разветвления например, из одного системного входного буфера в набор входов системной логики на кристалле, обычно управляемой этим буфером (см. Рисунок 11-11). В таких случаях применяется правило 11.3.1 e).

е) Параллельный выход ячейки управления и наблюдения, которая отслеживает сигнал, принятый на системном логическом входе, должен приводить в действие только одно из следующих действий:

1) Один или несколько системных логических входов;
2) Выходной сигнал, подаваемый на один вывод системной логики или на блок внешней аналоговой/цифровой схемы ; или
3) Сигнал, поступающий на выход управления одним или несколькими выходными буферами.

ПРИМЕЧАНИЕ — Следствием этого правила является то, что там, где сигнал, например, от системного входного контакта обычно разветвляется для управления более чем одним из вышеуказанных вариантов, требуется более одной ячейки управления и наблюдения. Обратитесь к 11.6 для получения правил, относящихся к сигналам выходных данных и элементам управления выводом.

е) Каждая ячейка, которая отслеживает сигнал, принятый на входе системной логики, должна быть спроектирована для маршрутизации сигналов, как показано в таблице 11-1.

ж) *Только для тактовых входов:* Когда выбрана команда *INTEST* или *RUNBIST*, сигнал, подаваемый на встроенную системную логику, должен быть одним из следующих:

Таблица 11-1—Маршрутизация сигналов в ячейках на входах системной логики

Инструкция	Сигнал, загруженный в сдвиговый регистр Ступень каждой ячейки на переднем крае TCK в состоянии контроллера *Capture-DR* является	Сигнал, поступающий с параллельного выхода ячеек управления и наблюдения, когда выбрана команда, является
CLAMP	Не относится к делу	Не определено [a]
EXTEST	Сигнал, поступающий на вход системной логики от внешнего источника	Не определено [a]
HIGHZ	Не относится к делу	Не определено [a]
INTEST	Не определено	*Только для входов, отличных от тактовых:* Параллельный выход каскада сдвигового регистра *Только для тактовых входов:* Смотрите Правило 11.5.1 g)
ПРЕДВАРИТЕЛЬНАЯ НАГРУЗКА	Не определено	Сигнал, полученный на подключенном системном выводе
РАНБИСТ	Не определено (Не имеет значения, если только регистр граничного сканирования не выбран в качестве последовательного пути между TDI и TDO)	*Только для ввода без clock:* Не определено [b] *Только для тактовых входов:* Определяется правилом 11.5.1 g)
ПРИМЕР	Сигнал, поступающий на вход системной логики от внешнего источника	Сигнал, полученный на подключенном системном выводе
ОБХОД, *ИДЕНТИФИКАЦИОННЫЙ КОД*, *КОД ПОЛЬЗОВАТЕЛЯ*	Не относится к делу	Сигнал, полученный на подключенный системный PIN-код

[a] См. Правило 11.5.1 h) и разрешения 11.5.1 n) и 11.5.1 o).
[b] Смотрите Правила 11.5.1 h) и 11.5.1 k) и Разрешения 11.5.1 p) и 11.5.1 q).

1) Сигнал, полученный на подключенном системном выводе;
2) Сигнал TCK, управляемый таким образом, что встроенная системная логика изменяет состояние только в состоянии *Run-Test/Idle* контроллера; или
3) Только для *INTEST*, параллельного вывода каскада сдвигового регистра.

Примечания

1—Если выбран вариант 11.5.1 g) 1), разработчик компонента должен предположить, что сигнал, подаваемый на тактовый входной вывод, будет автономным и не поддающимся внешнему контролю. Следовательно, чтобы соответствовать правилам 8.9.1 b) и 8.10.1 b), в компонент должна быть встроена схема, обеспечивающая, чтобы только соответствующие тактовые переходы вызывали работу системной логики на кристалле (например, в случае команды *INTEST* сигнал “hold” может подаваться импульсно внутри для обеспечения одиночного перехода в состояние *Run-Test/Idle* контроллера).

2—Если компонент имеет более одного тактового входного вывода, разработчик компонента должен обеспечить правильную работу инструкций *INTEST* и *RUNBIST*, если таковые предусмотрены, для всех частотных и фазовых соотношений между тактовыми входными сигналами, которые допустимы для корректной нетестовой работы компонента.

з) Компонент не должен быть поврежден в результате сигналов, подаваемых на системную логику на кристалле, когда выбраны команды *CLAMP*, *EXTEST*, *HIGHZ* или *RUNBIST*.

ПРИМЕЧАНИЕ —Это может быть достигнуто путем отключения работы встроенной системной логики или путем проектирования ячейки таким образом, чтобы при выборе этих команд выдавался постоянный логический сигнал.

i) Конструкция ячейки регистра граничной развертки, которая отслеживает сигнал, принятый на входе системной логики, должна быть такой, что если логическое значение *V* присутствует на выводе системного ввода в момент загрузки данных из сигнала в каскад сдвигового регистра ячейки регистра граничной развертки (в состоянии контроллера *Capture-DR*), значение, смещенное из ячейки через TDO во время немедленного последующего сдвига регистра граничной развертки, должно быть *V*.

ПРИМЕЧАНИЕ — Например, если логическое значение 0 применяется к системному входному выводу, логическое значение 0 должно соблюдаться при TDO после загрузки ячейки, которая отслеживает этот вывод. Смотрите Рисунок 11-12.

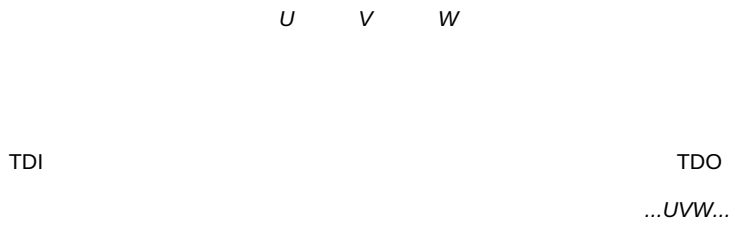
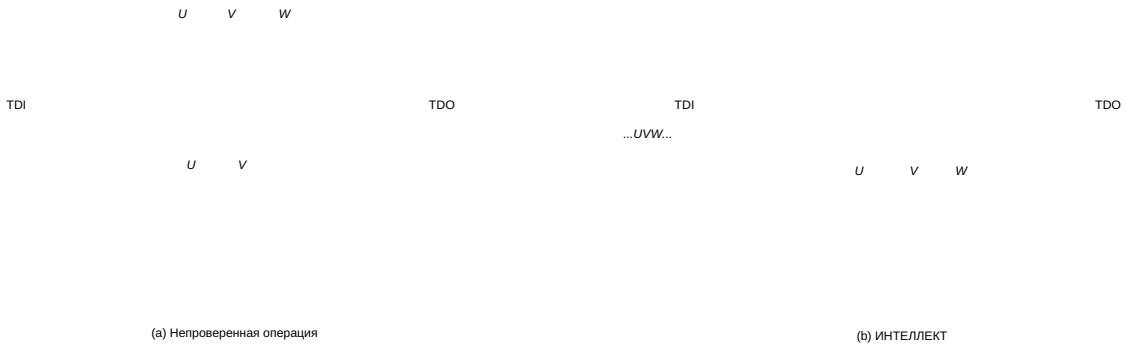


Рисунок 11-12—Неинверсия данных между pin и TDO

к) Для каждой ячейки управления и наблюдения, которая отслеживает сигнал, принятый системной логикой от системного входного вывода *P*, значение данных, *D* переданное в ячейку через TDI и впоследствии переданное в системную логику на кристалле при выборе команды *INTEST*, должно приводить к тому же результату, что и применение значения *D* в *P* при нормальной (нетестовой) работе компонента (рисунок 11-13).



(a) Непроверенная операция

(b) ИНТЕЛЛЕКТ

Рисунок 11-13—Неинверсия данных между TDI и системной логикой

к) Когда предоставляется и выбирается команда *RUNBIST*, конструкция ячеек регистра граничного сканирования, которые отслеживают сигналы, принимаемые системной логикой на кристалле, должна быть такой, чтобы предотвращать помехи при выполнении самотестирования в результате данных, полученных от внешних источников (например, выводов системного ввода).

Разрешения

л) *Только для тактовых входов:* Если поддерживается команда *INTEST*, именно одна из предоставленных ячеек регистра граничного сканирования, которая отслеживает сигнал, принятый на тактовом входе встроенной системной логики, также может управлять этим входом.

ПРИМЕЧАНИЕ — См. Правило 11.5.1 d).

м) Каскад сдвигового регистра ячейки управления и наблюдения, который управляет входом для системной логики на кристалле, может быть снабжен заблокированным параллельным выходом.

Примечания

1—Если заблокированный параллельный вывод опущен, значения переходных данных будут передаваться в системную логику на кристалле во время сдвига регистра граничного сканирования при выборе команды *INTEST*. Там, где это вызвало бы нежелательную работу встроенной системной логики, параллельный вывод каскада сдвиговых регистров может быть заблокирован таким образом, что данные, передаваемые в системную логику, изменяются только по завершении переключения (в состоянии контроллера *Update-DR*).

2—Если предусмотрены запертые параллельные выходы, применяются правило 11.3.1 c) и Разрешение 11.3.1 h).

п) Ячейки управления и наблюдения, которые управляют входами системной логики на кристалле, могут быть сконструированы таким образом, что при выборе команды *CLAMP*, *EXTEST* или *RUNBIST* сигнал, подаваемый на системную логику, является параллельным выходом каскада сдвиговых регистров.

о) Ячейки управления и наблюдения, которые управляют входами системной логики на кристалле, могут быть сконструированы таким образом, что при выборе команды *CLAMP*, *EXTEST* или *HIGHZ* в системную логику подается постоянное значение сигнала.

р) Ячейки могут быть сконструированы так, чтобы действовать как генераторы тестовых шаблонов для встроенной системной логики, когда выбрана команда *RUNBIST* (или альтернативная команда самопроверки).

в) Ячейки могут управляться таким образом, что во время выполнения команды *RUNBIST* или альтернативной команды самопроверки данные могут проходить между системными входными выводами и встроенной системной логикой без модификации.

ПРИМЕЧАНИЕ — Однако результаты выполнения команды *RUNBIST* не зависят от данных, полученных на выводе системного ввода, не относящемся к clock [см. Правило 8.10.1 j)].

г) Входные данные системной логики на кристалле могут наблюдаться одной или несколькими ячейками регистра сканирования границ только для наблюдения в дополнение к ячейке, требуемой Правилом 11.5.1 a).

ПРИМЕЧАНИЕ —Такие дополнительные ячейки являются избыточными в том смысле, что их можно было бы исключить из конструкции без ущерба для соответствия настоящему стандарту (см. 11.8).

11.5.2 Описание

В примерах реализаций для ячеек регистра пограничного сканирования, содержащихся в этом разделе, маршрутизация данных через каждую ячейку управляется одним или несколькими сигналами управления режимом (обозначенными как *Mode* или *Mode-N*). Различные сигналы управления режимом используются для ячеек на входных и выходных контактах компонента, и эти сигналы получены из команды, присутствующей на параллельном выходе регистра команд.

Примеры конструкций ячеек регистра граничной развертки, расположенных на системных входных выводах, приведены на рисунках с 11-14 по 11-18. Значение соответствующего <имя ячейки> из стандартного пакета VHDL BSDL, которое соответствует каждому рисунку, приведено для удобства (см. B.8.14 и пункт B.9), смещенное от основного заголовка в квадратных скобках []. Однако следует отметить, что рисунки не представляют ни предпочтительной, ни обязательной реализации названного типа ячейки BSDL.

Обратите внимание, что правило 11.4.1 e) разрешает принимать входные данные в ячейку регистра пограничного сканирования, доступную только для наблюдения, из любого сигнала, который прозрачно поступает с системного входа по неинвертирующему пути, например, из точки в дереве распределения сигналов.

В таблице 11-2 показано значение сигнала режима для ячеек, показанных на рис. 11-14 и рис. 11-15, для каждой из инструкций регистра граничного сканирования, определенных в разделе 8.

Таблица 11-2—Режим генерации сигнала для примеров ячеек на рис. 11-14 и 11-15

Инструкция	Режим
EXTEST	0
ПРЕДВАРИТЕЛЬНАЯ НАГРУЗКА	0
ПРИМЕР	0
ИНТЕРЕС	1
РАНБИСТ	X
ЗАЖИМ	X

ПРИМЕЧАНИЕ—Когда *САМЫЙ БОЛЬШОЙ* команда выбрана, ячейка, показанная на рис. 11-14, передает данные, полученные на системном входном выводе, в системную логику на кристалле. Во многих случаях система на чипе логики будут терпимы к таким сигналам, которые Усу-союзник будет не представитель тех, полученные в ходе нормального (nontest) операции. Однако в некоторых случаях может быть необходимо предотвратить передачу принятых данных (например, путем добавления логического элемента на выходе ячейки к системной логике на кристалле , как показано на рисунке 11-16). В качестве альтернативы может использоваться ячейка, показанная на рис.11-18, которая позволяет пользователю управлять данными, представленными системной логике на кристалле, пока выбрана команда *EXTEST*.

Рисунок 11-14 — Ячейка ввода с параллельным выходным регистром [BC_2]
(смотрите Таблицу 11-2 для получения сигнала режима)

Схемы на рис. 11-14 и рис. 11-15 позволяют управлять системной логикой на кристалле из ячейки регистра ограниченной развертки, когда команда *INTEST* выбрана в соответствии с Правилом 11.5.1 f) и, для тактовых входов, правилом 11.5.1 g) 3). Схема на рисунке 11-17 не может подавать сигналы в системную логику и может использоваться на входе синхронизации в соответствии с Правилом 11.5.1 (g) 1). Последняя конструкция может быть использована в обстоятельствах, когда задержка, вносимая в тракт сигнала мультиплексором, приведет к превышению проектного показателя. (Примером может служить высокопроизводительный тактовый вывод.)

Рисунок 11-15 — Ячейка ввода без параллельного выходного регистра [BC_3]
(смотрите Таблицу 11-2 для получения сигнала режима)

Рисунок 11-16—Ячейка, которая принудительно вводит системную логику равной 1 во время *EXTEST* [BC_4]

Схема на рис. 11-16 реализует Разрешение 11.5.1 o).

Схема на рис. 11-14 включает параллельный выходной регистр, который обновляется со ступени сдвигового регистра в состоянии контроллера *Update-DR* [Разрешение 11.5.1 m)]. Этот регистр включен для предотвращения применения изменений на выходе каскада сдвиговых регистров во время переключения к системной логике на кристалле при выборе команды *INTEST* (что может вызвать нежелательную операцию). Обратите внимание, что параллельный выход может альтернативно удерживаться в защелке с регулировкой уровня, включаемой логикой 1 на входе UpdateDR из примера контроллера отвода (рис. 6-5 и 6-6).

Рисунок 11-17 — Ячейка ввода, доступная только для наблюдения [BC_4]

Схема, показанная на рис. 11-18, может использоваться для ячеек регистра граничной развертки, расположенных как на системном входе, так и на выводах системы с двумя состояниями, хотя сигнал режима, подаваемый на ячейку, может отличаться в каждом конкретном случае.
Когда ячейка используется на системном входном выводе, сигналом режима следует управлять, как показано в таблице 11-3.

Рисунок 11-18 — Ячейка ввода, поддерживающая все инструкции [BC_1]
(смотрите Таблицу 11-3 для получения сигнала режима)

Обратите внимание, что единственное различие между правилами, которые применяются к входам без синхронизации и системным входам синхронизации, заключается в том, что предоставление ячеек контроля и наблюдения не является обязательным на входах синхронизации, когда *поддерживается команда* INTEST
. Таким образом, нет необходимости подключать схему к сигнальному тракту между тактовым
входным выводом и встроенной системной логикой.

Таблица 11-3—Режим генерации сигнала для примера ячейки на рисунке 11-18

Инструкция	Режим
EXTEST	1
ПРЕДВАРИТЕЛЬНАЯ НАГРУЗКА	0
ПРИМЕР	0
ИНТЕРЕС	1
РАНБИСТ	1
ЗАЖИМ	1

11.6 Обеспечение и эксплуатация ячеек на выходах системной логики

Правила, изложенные в этом подпункте, применяются к выходам встроенной системной логики. Эти выходы могут управлять данными входов буферов на системных выходных контактах или на двунаправленных системных контактах, когда они работают как системные выходы. Они также могут управлять регуляторами выходной активности буферов, расположенных на таких выводах. Альтернативно, встроенные системные логические выходы могут управлять встроенными блоками схемы смешанных сигналов, которые не являются частью встроенной системной логики.

Многие из правил, представленных в 11.6.1, применимы как к сигналам управления, так и к сигналам данных, выводимым из логики встроенной системы . Однако некоторые правила применимы только к встроенным системным логическим выходам, которые управляют вводами данных буферов на выводах системного вывода, в то время как другие применяются только к выходам, которые управляют входами управления в таких буферах. В случаях, в которых правило предназначено для ограниченного применения к тому или иному использованию выходного сигнала системной логики, правилу предшествует фраза “*Вводы управляющих данных только в буферы:*.”

Примечания

1—Входные сигналы для встроенных аналого-цифровых схемных блоков считаются эквивалентными входам данных для выходных буферов.

2—Если предусмотрена дополнительная команда *HIGHZ,* выбор этой команды переведет каждый выходной вывод в неактивное состояние накопителя, включая выводы, для которых нет системных требований для работы в трех состояниях. В тех случаях, когда возможность трех состояний будет предоставлена исключительно для обеспечения реализации команды *HIGHZ,* pin-код следует рассматривать как PIN-код с двумя состояниями для целей настоящего стандарта.

3—В этом подпункте рассматривается предоставление ячеек регистра граничного сканирования на выходах встроенной системной логики в случаях, когда эти выходы управляют выводами системы во время нормальной (нетестовой) работы. Случай, когда выходной сигнал от встроенной системной логики приводит в действие системный двунаправленный вывод во время обычной (нетестовой) работы, обсуждается в 11.7.

11.6.1 Технические характеристики

Правила

а) *Ввод данных только в буферы:* Для каждого вывода встроенной системной логики, которая управляет вводом данных в буфер на выводе системной логики, по крайней мере, в одной ячейке регистра сканирования границ с контролем и наблюдением должно выполняться только одно из следующих действий:

1) Сигнал, поступающий с выхода системной логики (рис. 11-19a); или

2) Сигнал на соответствующем выводе (рис. 11-19b).

б) *Управляющие входы только для буферов:* Для каждого выхода встроенной системной логики, который управляет управляющим входом буфера на выводе вывода системной логики, по крайней мере, одна ячейка регистрации пограничного сканирования с контролем и наблюдением должна регистрировать сигнал, поступающий с выхода системной логики (рис. 11-19 в).

..... (рис. 11-19 в).

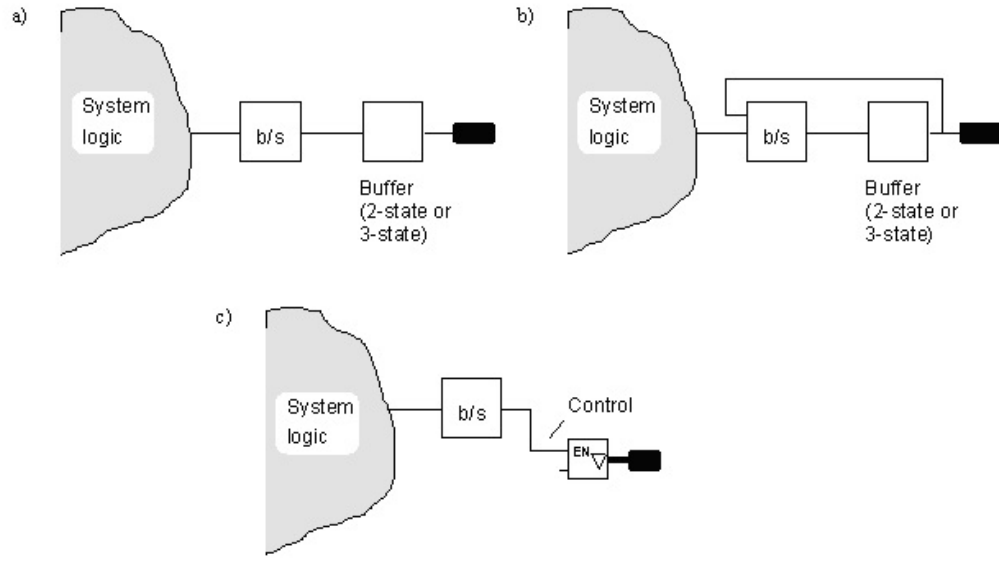


Рисунок 11-19—Расположение ячейки регистра граничной развертки на выводе цифровой системы

в) *Ввод данных только в буферы:* Для данного ввода данных в выходной буфер именно одна из ячеек регистра граничного сканирования, удовлетворяющая правилу 11.6.1 a), должна быть способна управлять этим вводом данных (см. Рисунок 11-20).

ПРИМЕЧАНИЕ—Правило 11.6.1 a) [11.6.1 b)] предусматривает, что по крайней мере одна ячейка будет отслеживать любые данные (управляющие), выводимые из встроенной системной логики. Таких ячеек может быть несколько. Если, например, выходной сигнал встроенной системной логики разветвляется на несколько системных выходных выводов, правила 11.6.1 c) и 11.6.1 e) требуют, чтобы в каждом ответвлении разветвления была размещена точно одна ячейка, способная управлять подключенным выводом. В конструкцию могут быть включены дополнительные “резервные” ячейки, предназначенные только для наблюдения [Разрешение 11.6.1 s)], которые могут отслеживать выходные данные встроенной системной логики, но которые не могут управлять каким-либо системным выводом (см. 11.8).

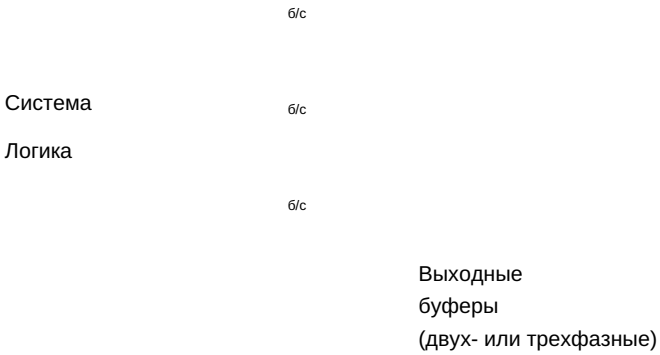


Рисунок 11-20—Предоставление ячеек регистра граничного сканирования на выходах системной логики

г) *Управляющие входы только для буферов:* Для данного управляющего входа в выходной буфер именно одна из ячеек регистра граничного сканирования, удовлетворяющих правилу 11.6.1 b), должна быть способна управлять этим управляющим входом.

ПРИМЕЧАНИЕ — Смотрите также Правила 11.6.1 e) и 11.6.1 f).

e) Параллельный выход ячейки регистра сканирования границ с контролем и наблюдением, которая наблюдает за системным выходом встроенной системной логики, должен управлять только одним из следующих действий:

1) Одиночный ввод данных в системный выходной буфер (двух- или трехфазный, или выходной сигнал для двунаправленного вывода); или

2) Управляющие входы для набора выходных буферов.

ПРИМЕЧАНИЕ — В последнем случае смотрите Правило 11.6.1 f). Обратите внимание, что там, где один выходной сигнал встроенной системной логики используется как в качестве управления выходными буферами на группе выводов, так и в качестве выходного сигнала данных на одном или нескольких других выводах, для трактов управляющих и неконтролируемых сигналов требуются отдельные ячейки регистра граничной развертки, как показано на рисунке 11-21.

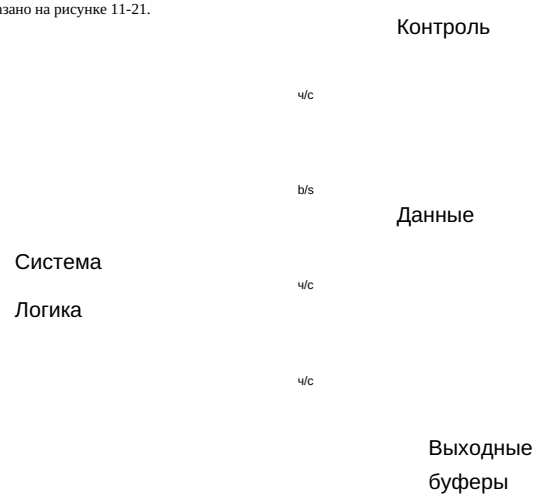


Рисунок 11-21—Предоставление ячеек, когда один выходной сигнал используется и в качестве управления, и в качестве данных

e) *Управляющие входы только для буферов:* Когда одна ячейка регистра граничного сканирования управляет управляющими входами для вывода буферов на нескольких системных выходных выводах, заданное значение данных (0 или 1), хранящееся в ячейке, должно вызывать одинаковую операцию на всех подключенных системных выходных выводах (например, 0 может быть определено для перевода всех подключенных выводов в неактивное состояние привода).

ПРИМЕЧАНИЕ — см. также Рекомендацию 11.6.1 q).

ж) *Управляющие входы только для буферов:* В случаях, когда единый управляющий сигнал подается на набор системных выходных выводов, который включает как трехфазные, так и двунаправленные выводы, при интерпретации Правила 11.6.1 f) включение трехфазного системного вывода (ов) должно рассматриваться как перевод двунаправленного системного вывода (ов) в режим вывода; отключение трехфазного системного вывода (ов) должно рассматриваться как перевод двунаправленного системного вывода (ов) в режим вывода.

h) Каждая ячейка, которая отслеживает логический вывод системы, должна быть спроектирована для маршрутизации сигналов, как показано в таблице 11-4.

i) *Управляющие входы только для буферов:* Для каждой соответствующей команды параметр в правом столбце таблицы 11-4 для управления значением, отключающим подключенные выходные буферы, должен быть выбран либо для всех ячеек, которые управляют управляющими входами выходных буферов, либо ни для одной из этих ячеек.

ПРИМЕЧАНИЕ —Например, если при выборе команды *INTEST* один выходной вывод компонента с тремя состояниями был переведен в высокоимпедансное состояние, все остальные выходные выводы с тремя состояниями также были бы переведены в высокоимпедансное состояние при выборе команды *INTEST*.

к) Каждая ячейка регистра сканирования границ с контролем и наблюдением, которая отслеживает выходные данные встроенной системной логики (данные или управление), должна быть снабжена заблокированным параллельным выходом.

к) Когда выбраны команды *EXTEST*, *PRELOAD* или *INTEST*, заблокированные параллельные выходы ячеек регистра сканирования границ с контролем и наблюдением, которые отслеживают выходы системной логики на кристалле , должны зафиксировать данные, хранящиеся в каскаде сдвигового регистра на падающем фронте TCK в состоянии *контроля с обновлением-DR.* .

Таблица 11-4—Маршрутизация сигналов в ячейках на выходах системной логики

Инструкция	Сигнал, загруженный в сдвиговый регистр Стадия каждой ячейки на переднем фронте TCK в состоянии *контроллера* *Capture-DR* является	Сигнал, поступающий с параллельного выхода каждой ячейки управления и наблюдения при выборе команды является
CLAMP	Не относится к делу	Заблокированный параллельный выход каскада сдвигового регистра
EXTEST	Не определено	Заблокированный параллельный выход каскада сдвигового регистра
HIGHZ	Не имеет значения	Значение, отключающее подключенные выходные буферы
ИНТЕРЕС	Выходной сигнал от встроенной в микросхему системной логики	Заблокированный параллельный вывод каскада сдвигового регистра или значение, отключающее подключенные выходные буферы[a]
ПРЕДВАРИТЕЛЬНАЯ ЗАГРУЗКА	Не определено	Выходной сигнал от встроенной в микросхему системной логики
РАНБИСТ	Не определено (Не имеет значения, если только в качестве последовательного пути между TDI и TDO не выбран регистр пограничного сканирования)	Заблокированный параллельный выходной сигнал каскада сдвигового регистра; или значение, которое отключает подключенные выходные буферы[a]
ВЫБОРКА	Выходной сигнал от встроенной системной логики или от подключенного выходного вывода	Выходной сигнал от встроенной в микросхему системной логики
ОБХОД, ИДЕНТИФИКАЦИОННЫЙ КОД, КОД ПОЛЬЗОВАТЕЛЯ	Не относится к делу	Выходной сигнал от встроенной в микросхему системной логики

[a] Эта опция доступна только для ячеек, которые управляют управляющими входами выходных буферов. Если эта опция выбрана для управляющего ввода в выходной буфер, данные, вводимые в этот буфер при выборе команды, могут рассматриваться как “Не определенные”.

l) Когда выбраны команды *CLAMP*, *HIGHZ* или *RUNBIST*, заблокированные параллельные выходы ячеек регистра сканирования границ с контролем и наблюдением, которые наблюдают за выходами системной логики на кристалле, должны сохранять свое состояние неизменным во всех состояниях контроллера.

м) ~~*Только ввод данных в буферы*~~:

1) Если *C* - ячейка регистра сканирования границ для контроля и наблюдения, которая передает данные в выходной буфер для вывода системы *P*; и
2) Если *O* - выходной сигнал встроенной системной логики, который отслеживается *C* и который передает данные в *P* во время нормальной (нетестовой) работы; и
3) Если *C* является *n*-й ячейкой регистра сканирования границ; и
4) Если *V* - логический сигнал, поступающий от *P* в обычном (нетестовом) режиме (т.е. Когда сигнал, поступающий от *P*, определяется выходом от *O*),

когда выбирается команда, которая требует, чтобы вывод *O* был записан в *C* (например, команда *SAMPLE*), логическое значение, наблюдаемое как *n*-й бит, выводимый из регистра граничного сканирования в TDO в операции сканирования сразу после записи, должно быть *V* (рисунок 11-22).

Примечания

1—Например, если системная логика будет передавать логический 0 через вывод системного вывода, логический 0 должен наблюдаться при TDO после загрузки ячейки, которая наблюдает этот вывод. Смотрите Рисунок 11-22.

2—Для выходов, на которых активно задается только одно логическое значение (например, выходы с открытым коллектором), получение одного значения данных (0 или 1) от встроенной системной логики приведет к тому, что выход будет неактивным. В этих случаях значение данных, наблюдаемое в TDO, будет значением, которое при подаче в выходной буфер из встроенной системной логики приведет к неактивному выводу.

Figure 11-22—Noninversion of data between the system logic and TDO

п) *Только ввод данных в буферы*:

1) Если *C* - ячейка регистра сканирования границ для контроля и наблюдения, которая передает данные в выходной буфер для вывода системы *P*; и
2) Если *C* является *n*-й ячейкой регистра сканирования границ; и
3) Если *V* - значение *n*-го бита последовательного потока данных, *S* вводимого в регистр граничного сканирования через TDI; и
4) Если длина *S* равна количеству ячеек в регистре граничного сканирования,

when an instruction is selected that causes the latched parallel output of *C* to determine the value of the data signal driven from *P* (e.g., *EXTEST*) and when the data stream *S* is shifted into the boundary-scan register and, immediately subsequent to the shifting operation, updated to the latched parallel outputs of the boundary-scan register, the value of the data signal output from *P* shall be *V* (Figure 11-23).

NOTE—For outputs where only one logic value is actively driven (e.g., open-collector outputs), receipt of one data value (0 or 1) from the on-chip system logic will cause the output to be inactive. In these cases, the data value input at TDI shall be the value that, when fed to the output buffer from the on-chip system logic, will cause the output to be inactive.

o) *Control inputs to buffers only*:

1) If *C* is the control-and-observe boundary-scan register cell that drives the control input of the output buffer for a system output pin *P*; and
2) If *O* is the output of the on-chip system logic that is observed by *C* and that controls the activity of *P* during normal (nontest) operation; and
3) If *C* is the *n*th cell in the boundary-scan register; and
4) If *V* (not *V*) is the value of the *n*th bit of *S* output from the boundary-scan register through TDO immediately after capture of *O* into *C* when *P* would be active (inactive) in normal (nontest) operation; and
5) If the length of *S* is equal to the number of cells in the boundary-scan register,

when an instruction is selected that causes the latched parallel output of *C* to determine the activity of *P* (e.g., *EXTEST*) and when a data pattern *S* containing *V* (not *V*) as the *n*th bit is shifted onto the latched parallel outputs of the boundary-scan register, *P* shall be active (inactive).

NOTE—For example, where a logic 0 output from the system logic normally would disable a driven output buffer, a logic 0 should be shifted out through TDO whenever the output would have been disabled. Further, a logic 0 shifted into

the cell through TDI should, if driven to the output buffer, cause the output buffer to be disabled. The effect of this rule is similar to that of Rules 11.6.1 m) and 11.6.1 n) (Figure 11-24).

Figure 11-23—Noninversion of data between TDI and a system output pin

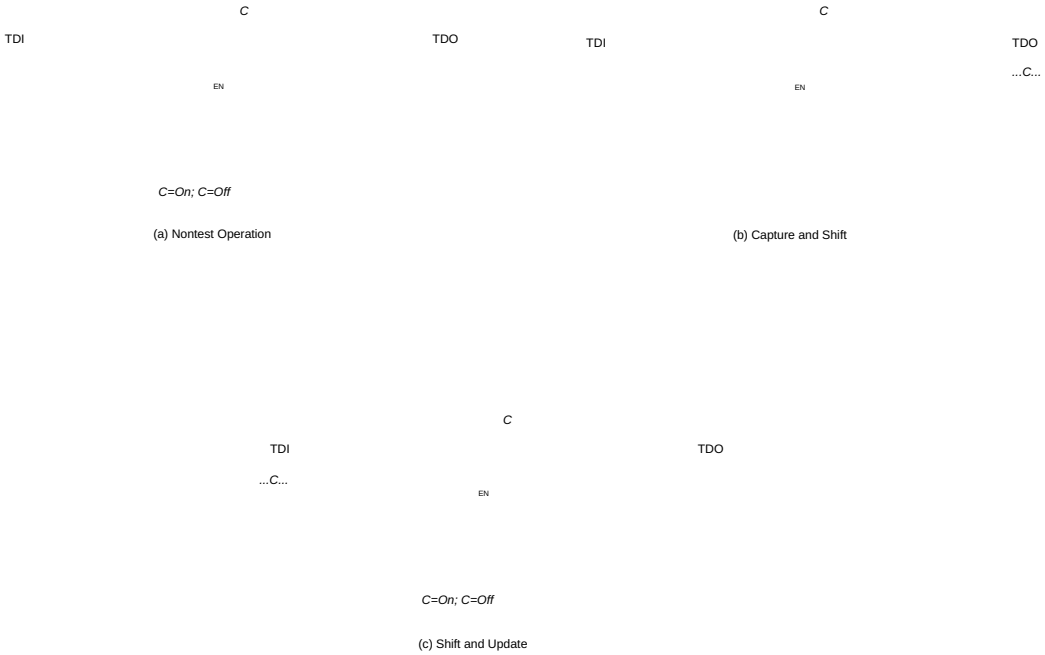


Figure 11-24—Noninversion of control signal values between the system logic and TDO

p) *Control inputs to buffers only*: If the latched parallel output of a boundary-scan register cell that drives a control input to an output buffer is reset in the *Test-Logic-Reset* controller state, it shall be reset to the state that will cause the connected output buffer(s) to be disabled.

NOTE—The timing of the reset is specified in Rule 11.3.1 c) and Permission 11.3.1 h).

Recommendations

q) *Control inputs to buffers only*: The control signal for each functionally distinct group of system output pins (e.g., an address bus or a data bus) should be driven from a distinct boundary-scan register cell dedicated to that purpose, even where the output from the on-chip system logic observed by that cell normally would drive a common control signal to more than one such group of pins.

NOTE—This reduces the complexity of the test generation task because during the test, buffers can be enabled independently as required.

Permissions

r) Boundary-scan register cells that observe outputs from the on-chip system logic may be designed to act as a part of a signature computing register for test results when the *RUNBIST* instruction (or an alternative self-test instruction) is selected.

s) Outputs from the on-chip system logic may be observed by one or more observe-only boundary-scan register cells in addition to the control-and-observe cells required by the preceding rules.

NOTE—Such additional cells are redundant in the sense that they could be omitted from the design without jeopardizing compliance to this standard (see 11.8).

11.6.2 Description

For 2-state output pins, where signals only can be at the high or low logic level at any given instant, one boundary-scan register cell is sufficient to allow the state of the pin to be controlled or observed. However, for 3-state pins the capability exists for data to be driven actively or inactively, such that four states are possible. Data from a minimum of two boundary-scan register cells therefore is required to allow the state (signal value plus active/inactive) of a 3-state pin to be controlled or observed.

Figure 11-25—Control of multiple 3-state outputs from one signal

Although it would appear that the additional cells might significantly increase the overhead needed to implement boundary scan, it is necessary to provide only one additional cell for each activity control signal generated in the circuit, although a judicious use of a few additional cells is recommended in Recommendation 11.6.1 q) (see, for example, Figure 11-25). Thus, where the activity of many 3-state output pins is controlled from a single source, for example, in a microprocessor address bus, only one additional cell is required to give the necessary control. Since, given the basic design of the circuit, it would be a design error if such 3-state pins were wired together, there is no need for the design of the boundary-scan register to account for this possibility.

The need for the additional cells at output control signals is illustrated in Figure 11-26. This shows a wired junction between 3-state outputs from a number of components. To test this junction, a series of tests shall be performed, each of which checks that one of the outputs can drive either a 0 or a 1 to the receiving devices. During each test, the other outputs have to be set to the opposite data value (1 or 0, respectively) with a high-impedance drive. Table 11-5 shows the pair of tests needed to check the operation of one of the outputs connected to the junction.

Figure 11-26—Testing board-level bus lines

To apply the test, it is necessary to be able to control both the data value at each output and whether the output is enabled. This can be done via the boundary-scan register independently of the on-chip system logic.

Table 11-5—Test for driver B

Stimulus applied to the bus from			Result seen at
Component A	**Component B**	**Component C**	**Component D**
1/off	0/on	1/off	0
0/off	1/on	0/off	1

For similar reasons, there shall be additional boundary-scan register cells associated with each 3-state bidirectional system pin that control the activity of the associated output buffer. As in the case of 3-state system pins, these cells may be shared across a bus or between any group of 3-state bidirectional system pins that obtain their output control signal from a single source.

Figure 11-27 highlights the following problems that might be encountered when applying tests to logic blocks external to a component by using the boundary-scan register of the component but that are avoided by implementing boundary-scan register cells as defined in this clause.

— The logic block being tested may contain asynchronous sequential logic that will be set into undesirable states if shifting patterns appear at its inputs.

Figure 11-27—Testing external logic via the boundary-scan path

— The signals applied from the boundary-scan register may feed into clock inputs on the logic block being tested, which again will produce undesirable effects if the logic is not shielded from shifting patterns.

Since in a generally applicable architecture it cannot be guaranteed that such features do not exist in the circuitry under test, the boundary-scan design shall be such that these problems are guaranteed to be avoided. A design compatible with this standard ensures this by requiring a parallel output register or latch in each boundary-scan register cell that can affect the state of an output driver at a system pin. The inclusion of this register or latch ensures that while the *EXTEST* instruction is selected, the data driven from a component to neighboring circuitry changes only on completion of the shifting process.

A further potential problem is highlighted by the primitive (noncompliant) boundary-scan register cell design shown in Figure 11-28. During testing of the on-chip system logic (for example, through the *INTEST* or *RUNBIST* instruction), the example cell would allow responses from the system logic to pass through the data-path multiplexer to the shift-register input of the cell. This allows the output response from the on-chip system logic to be loaded into the output boundary-scan register cells and shifted out for inspection. However, a problem arises from the fact that the cell also allows the test response from the on-chip system logic to be output from the host component and hence to be applied to neighboring components on a board assembly.

Figure 11-28—A primitive noncompliant output cell design with potential problems

The application of raw test-response data from one component can have a damaging effect on other components in the circuit if it is received at clock or asynchronous data or control inputs. For example, if built-in self-testing is being performed on the memory controller of Figure 11-29, there is a distinct possibility that one or more test-response patterns from the core logic of the memory controller will cause simultaneous activation of the outputs feeding the chip select (CS) inputs of the memory devices. This situation would not occur during normal operation of the complete design, either due to constraints between the logic values applied to the inputs of the memory controller or due to the design of the on-chip system logic. The design would in some way ensure that only one output from the controller was active at any time.

The duration of an on-chip test is dependent on the type of system logic test performed. For static tests applied using the *INTEST* instruction, these potentially damaging output patterns can remain in effect over the interval between successive occurrences of the *Update-DR* controller state. For instance, in a circuit having a scan path length of 500 bits and a TCK rate of 5 MHz, the approximate interval between closest consecutive *Update-DR* controller states is 100 ms. For large board designs, the period could be sufficiently long to cause damage to drivers in contention on a bus.

Figure 11-29—A circuit illustrating potential boundary-scan test problem

One solution is to cause the output buffers of the memory controller that feed the memory CS inputs to be placed in a high-impedance state during internal testing of the controller. However, floating inputs can fluctuate between high and low logic levels and are susceptible to induced voltages from adjacent board wiring interconnects. Applying a pull-up resistor on the 3-state buffers will solve the bus contention problem in external components with active-low 3-state enables, but those with active-high 3-state enables are still at risk.

The solution adopted in this standard is to ensure that boundary-scan register cells placed at 2-state output pins are designed such that user-defined logic values can be placed at the associated pins while system logic within the component is tested. Figure 11-30 shows a design that provides this facility and meets the rules defined in this clause. Table 11-6 shows how the mode signal for Figure 11-30 is derived for each of the boundary-scan register instructions defined in Clause 8.

Figure 11-30—An output cell that supports all instructions [BC_1]
(see Table 11-6 for mode signal generation)

Table 11-6—Mode signal generation for the example cells in Figure 11-30, Figure 11-34, Figure 11-36, and Figure 11-46

Instruction	Mode
EXTEST	1
PRELOAD	0
SAMPLE	0
INTEST	1
RUNBIST	1
CLAMP	1

Note that the path in Figure 11-30 between the data input from the system logic and the multiplexer that feeds data to the system pin will not be used during execution of either the *EXTEST* or the *INTEST* instruction. In some cases, it may therefore be necessary to use additional test operations at the board level to test the circuitry within a component fully.

Figure 11-31—An output cell that supports *SAMPLE*, *PRELOAD*, *EXTEST*, and *RUNBIST* [BC_2] (see Table 11-7 for mode signal generation)

The example cell design in Figure 11-31 could be used where the *INTEST* instruction is not supported by a component, since this design does not permit Rule 11.6.1 h) to be met with respect to the *INTEST* instruction. Note that while the cell meets Rule 11.6.1 h) with respect to the *SAMPLE* instruction without additional provision, if the cell drives off-chip via an output buffer, it can be ensured that the signal value captured using the *SAMPLE* instruction is the one intended to be driven off-chip, not the one actually on the off-chip connection at the time. The latter may be affected by faults on the off-chip connection or, for bus connections, by the combination of drivers active at the time. By ensuring that the signal that should have been driven from the chip is sampled at the driving end, while the signal actually driven is sampled at the receiving end, additional diagnostic information is obtained.

Table 11-7 shows how the mode signal for Figure 11-31 is derived for each of the boundary-scan register instructions supported by the cell design.

Table 11-7—Mode signal generation for the example cells in Figure 11-31, Figure 11-33, and Figure 11-39

Instruction	Mode
EXTEST	1
PRELOAD	0
SAMPLE	0
RUNBIST	1
CLAMP	1

On the other hand, it is highly useful that the signal value captured using the *EXTEST* instruction is the one at the corresponding system output pin, according to Rule 11.6.1 a) 2). Doing so allows a connected system network both to be driven and to be captured at the same pin, thus allowing such networks to be tested for shorts to others even where there are no other connected boundary-scan device pins. Output cells of this type are termed self-monitoring. A cell design that implements this option while still capturing the system logic output during *SAMPLE* and *INTEST* is shown in Figure 11-32. An alternative, simpler cell design that also implements this option but cannot capture the system logic output, and thus does not support *INTEST*, is shown in Figure 11-33.

Figure 11-32—A self-monitoring output cell that supports *INTEST* [BC_9] (see Table 11-8 for mode signal generation)

Where a component has 3-state system output pins, these pins may feed onto a wired junction at the board level. In order to test the interconnections forming the wired junction using the *EXTEST* instruction, it shall be possible to drive independently onto the junction from each of the possible driving pins. As was dis-

Table 11-8—Mode signal generation for the example cell in Figure 11-32

Instruction	Mode 1	Mode 2
EXTEST	1	1
PRELOAD	X	0
SAMPLE	0	0
INTEST	0	1
RUNBIST	X	1
CLAMP	X	1

cussed earlier in this clause, to achieve this it is necessary to be able to control the output control signals fed to the output drivers at 3-state or bidirectional system pins.

Figure 11-33—A self-monitoring output cell that does not support *INTEST* [BC_10] (see Table 11-7 for mode signal generation)

In addition, it is necessary to ensure that contention does not occur on board-level interconnections when the on-chip system logic is tested using the *INTEST* or *RUNBIST* instruction. This requirement can be met in either of two ways:

a) The state of a system pin can be fully defined by the user by shifting data into the boundary-scan register.

b) A system pin can be forced into the inactive drive state. This additional option is possible since the board-level circuit design shall necessarily be designed such that components driven from the 3-state bus do not erroneously respond to high-impedance conditions during normal system operation. Therefore, the inactive drive state can be safely driven during testing of the system logic within a component.

Figure 11-34—Boundary-scan register cells at a 3-state output—Example 1 [BC_1, control and data] (see Table 11-6 for mode signal generation)

The options listed for the *INTEST* and *RUNBIST* instructions in Rule 11.6.1 h) cover these two possibilities. Figure 11-34 and Figure 11-35 give example designs for a boundary-scan register cell that could be used at a 3-state system output pin. Figure 11-34 implements option a), while Figure 11-35 implements option b). In Figure 11-34, the mode signal should be controlled as shown in Table 11-6.

In Figure 11-35, the design of the circuitry around the shift-register stages is such that all paths can be tested if both the *EXTEST* and *INTEST* instructions are executed with appropriate data. The Mode_1 and Mode_2 signals should be controlled as shown in Table 11-9.

Table 11-9—Mode signal generation for the example cell in Figure 11-35

Instruction	Mode_1	Mode_2
EXTEST	1	1
PRELOAD	0	1
SAMPLE	0	1
INTEST	0	0
RUNBIST	0	0
CLAMP	1	1
HIGHZ	0	0

Figure 11-35—Boundary-scan register cells at a 3-state output—Example 2 [BC_2, control and data] (see Table 11-9 for mode signal generation)

11.7 Provision and operation of cells at bidirectional system logic pins

11.7.1 Specifications

Rules

a) Boundary-scan register cells shall be provided at bidirectional system pins such that
   1) Whenever the pin is an input pin, all rules are met for cells provided at system input pins and inputs to the on-chip system logic (see 11.5);
   2) Whenever the pin is an output pin, all rules are met for cells provided at outputs of the on-chip system logic that drive data inputs of system output buffers (see 11.6); and
   3) All rules are met for cells provided at outputs of the on-chip system logic that drive control inputs of buffers at system output pins (see 11.6).

NOTE—In cases where the direction of signal flow is determined by an output *O* of the on-chip system logic, a boundary-scan register cell will exist in the signal path between *O* and the system pin. When the *EXTEST*, *CLAMP*, *INTEST*, or *RUNBIST* instruction is selected, the direction of signal flow will be determined by the data held in the latched parallel output of the shift-register stage of the boundary-scan register cell.

b) Whenever two separate boundary-scan register cells are provided at a bidirectional system pin to meet the requirements of Rules 11.7.1 a) 1) and 11.7.1 a) 2),
   1) The cell that meets the requirements of Rule 11.7.1 a) 1) shall at all times meet all rules for cells provided at system input pins and inputs to the on-chip system logic (see 11.5); and

2) The cell that meets the requirements of Rule 11.7.1 a) 2) shall at all times meet all rules for cells provided at outputs of the on-chip system logic that drive data inputs of system output buffers (see 11.6).

NOTE—A structure of two boundary-scan register cells that would meet Rule 11.7.1 a) while failing to meet Rule 11.7.1 b) not only would require more logic than a structure which meets both Rules 11.7.1 a) and 11.7.1 b) but also would have less test usefulness.

11.7.2 Description

These requirements represent a merging of those for 2-state or 3-state output pins with those for system input pins.

Figure 11-36 and Figure 11-37 give examples of the provision of boundary-scan register cells at 3-state bidirectional pins.

a) Figure 11-36 allows the state of the pin to be fully controlled while the *INTEST* or *RUNBIST* instruction is selected. The mode signal shown in Figure 11-36 should be controlled as indicated in Table 11-6. The *Reset** signal is fed from the example TAP controller in Figure 6-5 to the parallel output register of the control cell in accordance with Permission 11.3.1 h).

Table 11-10—Mode signal generation for the example cells in Figure 11-37

Instruction	Mode_1	Mode_2	Mode_3
EXTEST	1	0	1
PRELOAD	0	0	1
SAMPLE	0	0	1
INTEST	0	1	0
RUNBIST	X	X	0
CLAMP	1	X	1
HIGHZ	X	X	0

b) In Figure 11-37, a single boundary-scan register cell is used to control and observe both output and input data. This cell meets the requirements of 11.6.1 when the pin is defined to be an output and the requirements of 11.5.1 when it is defined to be an input. The various control signals used by the cell should be controlled as shown in Table 11-10.

As discussed in connection with Figure 11-35, the design of the circuitry around the shift-register stages in Figure 11-37 permits all circuitry in the cell to be tested if the *EXTEST* and *INTEST* instructions are executed with appropriate data.

c) Figure 11-38 is similar to Figure 11-37. Note that while this design conforms fully to the rules set out in this standard, it is not recommended for use in new component designs. This is because the combined input and output cell does not capture as much data as possible about the external interconnection when the *EXTEST* instruction is selected. The example design in Figure 11-37 is superior in this respect.

**Figure 11-36—Boundary-scan register cells at a bidirectional pin—
Example 1 [BC_1, control] (see Table 11-6 for mode signal generation)**

Figure 11-37—Boundary-scan register cells at a bidirectional pin—Example 2 [BC_2, control; BC_7, data] (see Table 11-10 for mode signal generation)

Figure 11-38—Boundary-scan register cells at a bidirectional pin—not recommended for new designs [BC_2, control; BC_6, data] (see Table 11-11 for mode signal generation)

Table 11-11—Mode signal generation for the example cells in Figure 11-38

Instruction	Mode_1	Mode_2	Mode_3	Mode_4
EXTEST	1	0	0	1
PRELOAD	0	0	1	1
SAMPLE	0	0	1	1
INTEST	0	1	1	0
RUNBIST	X	X	X	0
CLAMP	1	X	0	1
HIGHZ	X	X	0	0

Figure 11-39 shows how boundary-scan register cells may be provided at an open-collector bidirectional pin. Note that, like the cell design in Figure 11-31, this cell design does not support the *INTEST* instruction.

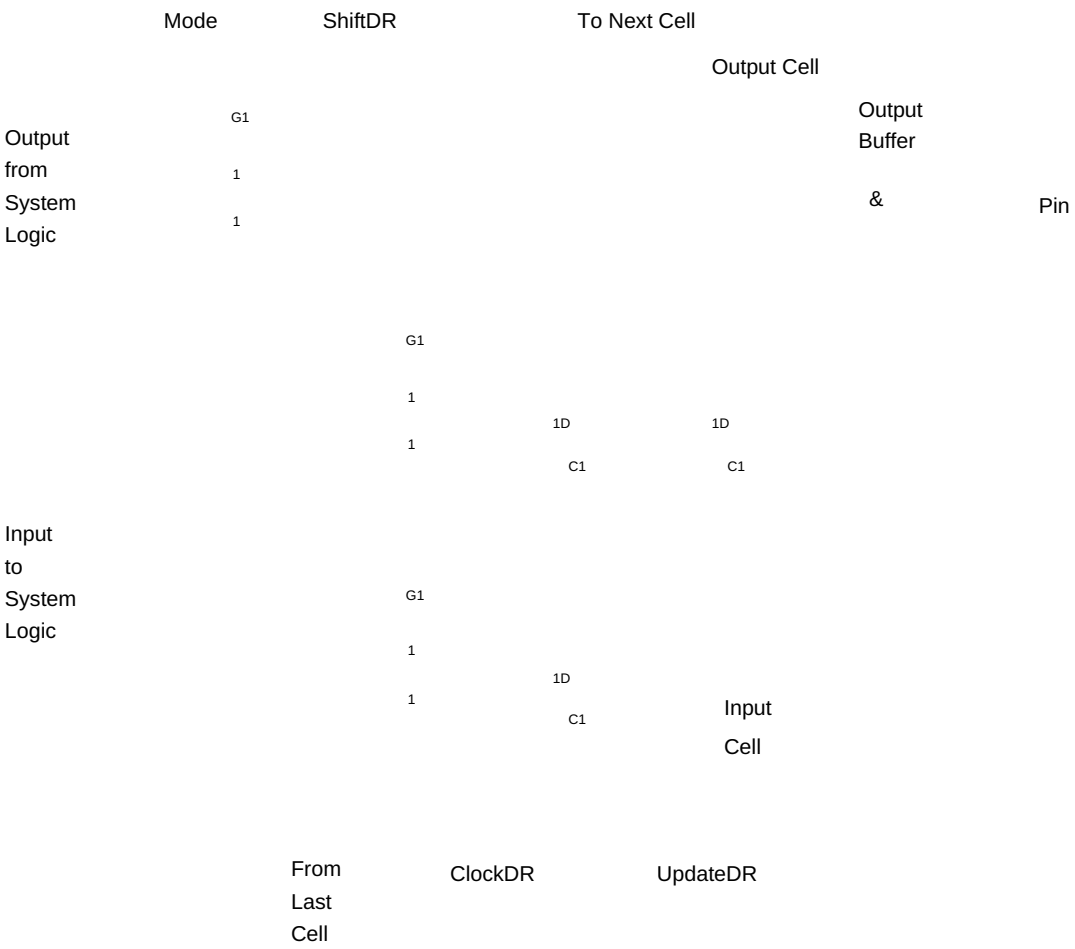


Figure 11-39—Boundary-scan register cells at an open-collector bidirectional pin [BC_4, input; BC_2, output] (see Table 11-7 for mode signal generation)

Note that in cases where a bidirectional pin is provided using an open-drain or open-collector output driver, the pin direction is defined as output for the one state that is actively driven and as input otherwise. Thus, a separate control cell is not required (i.e., a single bidirectional cell can provide for both data and control), as shown in Figure 11-40.

Figure 11-40—A boundary-scan register cell at an open-collector bidirectional pin where no control cell is required [BC_8] (see Table 11-12 for mode signal generation)

Table 11-12—Mode signal generation for the example cells in Figure 11-40 and Figure 11-41

Instruction	Mode
EXTEST	1
PRELOAD	0
SAMPLE	0
RUNBIST	X
CLAMP	1
HIGHZ	X

Figure 11-41 illustrates an alternative, reduced-complexity cell for use at a 3-state bidirectional pin. This cell is designed such that the signal present at the system pin is always captured. Because of this feature, this cell cannot be used where the *INTEST* instruction is to be provided. Where *INTEST* is to be supported, a cell of design similar to that shown in Figure 11-37 is required.

Figure 11-41—Boundary-scan register cells for use at a bidirectional pin where *INTEST* is not provided [BC_2, control; BC_8, data] (see Table 11-12 for mode signal generation)

11.8 Redundant cells

Redundant cells may exist in a component design for a number of reasons. For example:

a) They may be observe-only cells that observe a signal (input or output) that is observed by another boundary-scan register cell.

b) They may be parts of boundary-scan register cells designed for bidirectional system pins in cases where the pin has been programmed or otherwise customized to be permanently an input pin or an output pin. For example, a programmable component may be provided with three boundary-scan register cells at each system pin, sufficient to permit each system pin to be programmed as an input pin, 2-state or 3-state output pin, or bidirectional pin. After programming, certain of these cells may not be logically connected either to a given system pin or to a system logic input or output or both. Alternatively, a vendor of application-specific components may build a boundary-scan register into the basic component design (i.e., the design before the component is “committed”) that provides for a fully bidirectional signal at each possible system pin. When the basic component is “committed,” these cells will be constrained such that only the required functionality is connected.

11.8.1 Specifications

Rules

a) The contents of a redundant boundary-scan register cell shall not affect the behavior of any other part of the component.

Recommendations

b) The number of redundant boundary-scan register cells included in a component should be minimized.

c) Redundant cells should be designed such that the data shifted out through TDO after loading of the shift-register stage in the *Capture-DR* controller state is either a constant or the data just previously shifted into the cell.

11.8.2 Description

Some programmable components (e.g., programmable gate arrays or application-specific ICs) offer input/output circuits that can be programmed as input, output, 3-state, or bidirectional pins. To permit programming as a 3-state or bidirectional pin, two or more boundary-scan register cells would have to be included in each configurable cell to allow access to the data and control signals. However, when the cell is programmed as an input or 2-state output pin, only one cell will be required. In some implementations, the cells not associated with the programmed system function of a pin may be logically disconnected from the pin and from the system logic. Under such circumstances, the disconnected cells could no longer be used during testing and would become redundant. Rule 11.2.1 h) requires that the unused cells remain in the boundary-scan register so that the register has a fixed length regardless of how the component is programmed.

NOTE—In many programmable devices, programmed lack of logical connection(s) may occur only with regard to a boundary-scan register cell and the on-chip system logic. The cells provided for a particular programmable pin may remain logically connectable to that pin during testing, and the bidirectional control cell would then remain functional. The rules of this subclause do not prohibit this “excess” functionality at a pin. Indeed, interconnect test generation may actually be easier when all pins on a board-level net *appear* from *outside* the components to be provided with full bidirectional boundary-scan capability.

To minimize the number of redundant cells contained in the boundary-scan register of a component, the register should contain only cells that, in some programmed configuration of the component, can provide access to signals at the boundary of the on-chip system logic. For example, a cell that receives its parallel data input from the on-chip system logic and sends its parallel data output into the on-chip system logic should not be included in the boundary-scan register (e.g., as shown in Figure 11-42).

11.9 Special cases

11.9.1 Specifications

Permissions

a) In a case in which a system logic input pin is used s*olely* as a source of control or s*olely* as a source of data for a system output pin, a single cell may be provided that meets the rules of 11.5.1 (for the input pin) and 11.6.1 (for the output pin).

Figure 11-42—A cell that should not be included in the boundary-scan register

11.9.2 Description

Where the signal received at a system logic input pin is used solely to provide data or control for a system output pin, it is possible to use a single boundary-scan register cell to meet both sets of requirements. A common example of a situation where this might arise is one in which a system input pin is used solely to provide an output control signal for 3-state or bidirectional system pins. In such a case, either

a) Two separate boundary-scan register cells may be included, as shown in Figure 11-43; or
b) The functions of both cells may be combined into a single cell, as shown in Figure 11-44.

In the latter case, care should be taken in the design of the cell to ensure that it conforms to all the rules for the set of boundary-scan test instructions supported by the component.

Figure 11-43—Input pins used only to control output pins—Case A

Figure 11-44—Input pins used only to control output pins—Case B

Note that the situation illustrated in Figure 11-45 violates the rules of this standard. In this case, the signal received from the system input pin is used both as an output control and as an input to the on-chip system logic.

In a case in which the signal from a system input pin is used only as a control for the 3-state output buffer and the option has been taken to provide a single boundary-scan register cell as shown in Figure 11-18, the top cell in Figure 11-34 has to be modified if Recommendation 8.9.1 f) is to be met. Specifically, the cell has to reload its own state in the *Capture-DR* controller state when the *INTEST* instruction is selected to avoid taking on a value dependent on off-chip circuitry. Figure 11-46 shows how this could be achieved. For this design, the mode signal should be controlled as shown in Table 11-6.

Figure 11-45—Illegal use of a single cell for output control and data

12. The device identification register

This clause defines the design and operation of the optional device identification register. If provided, this register allows the manufacturer, part number, and version of a component to be determined through the TAP. One application of the device identification register is to distinguish the manufacturer(s) of components on a board when multiple sourcing is used. As more components emerge that conform to this standard, it may become desirable to allow for a system diagnostic controller unit to blindly interrogate a board design in order to determine the type of each component in each location. The need to do this becomes more apparent if one considers systems that are configurable by the addition of option boards or by programming certain components, etc. This information is also available for factory process monitoring and failure mode analysis of assembled boards.

Figure 11-46—Boundary-scan register cells at a 3-state pin where output control is from a system pin [BC_5, control; BC_1, data] (see Table 11-6 for mode signal generation)

NOTE—The design requirements contained in this clause apply only when the optional device identification register is included in a component.

12.1 Design and operation of the device identification register

12.1.1 Specifications

Rules

a) The device identification register shall be a shift-register-based path that has a parallel input but no parallel output.

b) The circuitry used to implement shift-register stages in the device identification register shall not be used to perform any system function (i.e., it shall be a dedicated part of the test logic).

c) On the rising edge of TCK in the *Capture-DR* controller state, the device identification register shall be set such that subsequent shifting causes an identification code to be presented in serial form at TDO.

d) The component shall contain a vendor-defined identification code, containing four fields (see Figure 12-1), which is accessed when the *IDCODE* instruction is entered.

e) For user-programmable components where programming cannot be completely determined by use of public instructions (see 8.2), the ability shall be provided to permit the user to program a supplementary 32-bit identification code that will be loaded into the device identification register in response to the *USERCODE* instruction.

f) The operation of the device identification register shall have no effect on the operation of the on-chip system logic.

Figure 12-1—Structure of the device identification register

12.1.2 Description

Figure 12-2 shows a design for a device identification register cell that satisfies these requirements.

Figure 12-2—Device identification register cell design

The identification code loaded into the device identification register in response to the *IDCODE* instruction allows the manufacturer, part number, and variant for the component to be read in a serial binary form. In situations where blind interrogation of a product is necessary, this information allows the structure of the board to be determined along with, by reference to stored data, the instruction set and other details for each component. (It is assumed that the components in a product will be selected from a limited set.)

Examination of the identification code also allows the structure of the boundary-scan register to be deduced, including the positioning of cells at input and output pins and the location of cells that control 3-state or bidirectional pins. This information is valuable in ensuring that contention between drivers at the board level is avoided (for example, as discussed in 11.6).

For programmable components, however, the configuration of pins as inputs, outputs, etc., may be determined by programming rather than by the basic design of the component. In such cases, therefore, a supplementary identification code is required to indicate how the component has been programmed. This supplementary code shall be user-programmable and accessed through the device identification register in response to the *USERCODE* instruction.

NOTE—The supplementary identification code is required only in cases where the component cannot be reprogrammed through the test logic defined by this standard. In cases where such reprogramming is possible, the ATE or master device controlling the operation of the component can ensure that it is programmed to the correct state at the start of the test sequence.

Since the bypass register (which is selected in the absence of a device identification register by the instruction loaded in the *Test-Logic-Reset* controller state) loads a logic 0 at the start of a scan cycle, whereas a device identification register will load a constant logic 1 into its LSB, examination of the first bit of data

shifted out of a component during a test data scan sequence immediately after exit from the *Test-Logic-Reset* controller state will show whether a device identification register is included in the design.

A requirement of the *IDCODE* and *USERCODE* instructions is that when they are used, the on-chip system logic shall continue its normal operation undisturbed. Rule 12.1.1 b) is included so that this requirement can be met. Note, however, provided that Rule 12.1.1 f) is met, the shift-register stages may be shared resources used by several of the registers defined by this standard and also by any design-specific test data register.

12.2 Manufacturer identity code

12.2.1 Specifications

Rules

a) The 11-bit manufacturer identity code shall be a compressed form of the code specified by EIA/JEP106 [3] generated as follows:
   1) Manufacturer identity code bits 7-1. The seven LSBs are derived from the last byte of the EIA/JEP106 code by discarding the parity bit.
   2) Manufacturer identity code bits 11-8. The four MSBs provide a binary count of the number of bytes in the EIA/JEP106 code that contain the continuation character (hex 7F). Where the number of continuation characters exceeds 15, these four bits contain the modulo-16 count of the number of continuation characters.

b) The manufacturer code 00001111111 shall not be used in components that are otherwise compatible with this standard.

Recommendations

c) Where the component is an application-specific integrated circuit (ASIC), the manufacturer identity code should be that of the manufacturer of the component rather than that of the designer.

12.2.2 Description

This scheme utilizes a listing of manufacturer identification codes specified by EIA/JEP106 as administered by Electronic Industries Association/Joint Electron Device Council (EIA/JEDEC).

The EIA/JEP106 code is formed from a variable number of eight-bit bytes. Each byte contains seven data bits and an odd parity bit (the MSB). Bytes other than the last contain continuation characters (hex 7F), while the last contains 127 different codes that, together with a knowledge of the number of preceding continuation code bytes, allow the manufacturer's identity to be determined.

The compressed form of the EIA/JEP106 code used within the device identification register limits the number of bits needed in the device identification register to contain the manufacturer identity code and allows the length of the code to be standardized. The length of the compressed code is fixed at 11 bits (see 12.1), which allows for 2032 different manufacturer codes. (Note that 16 codes are unused since these codes correspond to the hex 7F code in the seven LSBs—the EIA/JEP106 continuation character.)

One of the unused codes (00001111111) should be treated as illegal for components compatible with this standard. By shifting a dummy device identification code containing this manufacturer identity code from the bus master (ATE, board-level controller, etc.) into the board-level serial path set up by moving directly from the *Test-Logic-Reset* controller state into scanning of the test data registers, it is possible to detect the end of the identity code sequence.

[3] Information on references can be found in Clause 2.

When test data register scanning is entered in this way, the serial path at the board level includes

a) The device identification registers of components that provide them; and
b) The bypass registers of components that do not include a device identification register.

As discussed in 12.1, the fact that identification codes begin with a logic 1 whereas the bypass registers load a logic 0 allows the identification codes in the serial stream read out of the board to be detected. By feeding in the dummy identification code at the board's serial input and checking the serial output for the invalid manufacturer identity code 00001111111, it is possible to locate the end of the identification code sequence for a board containing an unknown number of components.

12.3 Part-number code

12.3.1 Specifications

Rules

a) The part-number code shall consist of 16 bits.
b) The manufacturer shall ensure that no two component types that are offered in the same package with the TAP pins in the same location have the same part-number code.

12.3.2 Description

The part-number code may be used to verify the type of the component inserted in a particular location on an assembled product. The use of a 16-bit value for this code gives an acceptably low chance that an incorrect component inserted in the location will return a correct part-number code.

Part-number codes could, for example, be generated from the textual part-number code by using a data compaction scheme.

12.4 Version code

12.4.1 Specifications

Rules

a) The version code shall consist of 4 bits.

Recommendations

b) The value of the version code for a component should be assigned to identify the variant of a component type.

13. Conformance and documentation requirements

13.1 Claiming conformance to this standard

The level of conformance to this standard can vary according to the range of test operations supported.

13.1.1 Specifications

Rules

a) Components that claim conformance to this standard shall comply with all relevant rules in the Specifications subclauses of this standard.

NOTE—Components that were designed before publication of this standard and conform fully with the requirements of this standard except in the control of the TDO output driver with regard to the controller states in which it is active (see the note that follows Table 6-2) also may claim conformance to this standard.

b) When it is claimed that a component conforms to this standard, the claim shall clearly identify the subset of the public instructions defined in this standard that is supported, as listed in Table 13-1 and defined in 8.2.

Table 13-1—Public instructions

Instruction	Status
BYPASS	Mandatory
CLAMP	Optional
EXTEST	Mandatory
HIGHZ	Optional
IDCODE	Optional
INTEST	Optional
PRELOAD	Mandatory
RUNBIST	Optional
SAMPLE	Mandatory
USERCODE	Optional

Recommendations

c) It is recommended that components support either the *INTEST* or the *RUNBIST* instruction or both.

Permissions

d) ASIC vendors may claim conformance to this standard by illustrating an interconnection of cells that, if built, would produce a component that meets the requirements of this standard.

13.1.2 Description

The minimum requirement for conformance to this standard is set to ensure that the user of an integrated circuit can perform two basic functions using the test logic: examine the operation of a prototype system and test assembled products for assembly-induced defects during manufacturing.

To enable efficient and comprehensive verification of internal component operation at the board and system level, it is strongly recommended that either the *INTEST* or *RUNBIST* instruction or both be supported.

13.2 Prime and second source components

13.2.1 Specifications

Rules

a) With the sole exception of the device identification code, the publicly accessible test logic for second source components shall operate in the same manner as that for the prime source component in response to all public instructions.

13.2.2 Description

It is essential that both the system and the test logic of prime and second source components operate in the same manner in the component purchaser's environment. This ensures that test programs created for a printed circuit board containing multiply sourced components produce consistent results regardless of the source of individual components.

The only exceptions to this requirement are the optional device identification register and any test logic that is accessed only in response to private instructions. In the former case, the identification code shall vary to identify the source of the particular component, its part number, and its revision (see Clause 12). In the latter case, test logic that is not publicly accessible is not intended for use other than by the component vendor; therefore, this test logic should not be operated by a board-level test program.

13.3 Documentation requirements

13.3.1 Specifications

Rules

a) For any component that claims conformance to this standard, the operation of all test logic accessed in response to public instructions shall be fully documented.

b) The following information, required by the component purchaser for use in test development and other activities, shall be supplied by the component manufacturer:

1) *Instruction register*. The following information pertaining to the instruction register is required:
   i) Its length.
   ii) The pattern of fixed values loaded into the register during the *Capture-IR* controller state.
   iii) The significance of each design-specific data bit presented at a parallel input, where provided.

2) *Instructions*. For each public instruction offered by a component, the following information is required:
   i) The binary code(s) for the instruction.
   ii) A list of test data registers placed in a test mode of operation by the instruction.
   iii) The name of the serial test data register path enabled to shift data by the instruction.
   iv) A definition of any data values that shall be written into test data registers before selection of the instruction and the order in which these values shall be loaded.
   v) The effect of the instruction. Any system pins whose drivers become inactive as a result of loading the instruction should be clearly identified.
   vi) A definition of the test data registers that will hold the result of applying a test and of how they are to be examined.
   vii) A description of the method of performing the test and of how data inputs and their corresponding data outputs are to be computed.

If private instructions are utilized in a component, the vendor shall clearly identify any instruction binary codes that, if selected, would cause hazardous operation of the component.

3) *Self-test operation*. For each instruction that causes operation of a self-test function, the following information is required in addition to that listed under Rule 13.3.1 b) 2):
   i) The minimum duration (e.g., a number of cycles of TCK) required to ensure completion of the test.
   ii) A definition of the test data registers whose states are altered during execution of the test.
   iii) A definition of the results of executing the self-test on a fault-free component.
   iv) An estimate of the percentage (e.g., to the nearest 5%) of the single stuck-at faults in the component's circuitry that will be detected by the self-test function *or* a description of the operation of the self-test function and the circuitry exercised.
4) *Test data registers*. For each test data register available for public use and access in a component, the following information is required:
   i) The name of the register used for reference in other parts of the data sheet.
   ii) The purpose of the register.
   iii) The length.
   iv) A full description of the operating modes of the register.
   v) The result of setting each bit at the parallel output of the register.
   vi) The significance of each bit loaded from the parallel input of the register.
5) *Boundary-scan register*. The following information is required in addition to that listed under Rule 13.3.1 b) 4):
   i) The correspondence between boundary-scan register bits and system pins, system direction controls, or system output enables.
   ii) Whether each pin is an input, a 2-state output, a 3-state output, or a bidirectional pin.
   iii) For each boundary-scan register cell at an input pin, whether the cell can apply tests to the on-chip system logic.
   iv) For each boundary-scan register cell associated with an output or direction control signal, a list of the pins controlled by the cell and the value that shall be loaded into the cell to place the driver at each pin in an inactive state or will be observed using the *SAMPLE*, *PRELOAD*, or *INTEST* instructions when the on-chip system logic causes the driver to be inactive.
   v) The method by which single-step operation is to be achieved while the *INTEST* instruction is selected if this instruction is supported.
   vi) The method of providing clocks to the on-chip system logic while the *RUNBIST* instruction is selected if this instruction is supported.
   vii) For each redundant cell, whether the cell returns either the value shifted in or a constant after loading of the cell in the *Capture-DR* controller state.
6) *Device-identification register*. Where a device identification register is included in a component, the following information is required in addition to that listed under Rule 13.3.1 b) 4):
   i) The value of the manufacturer’s identification code.
   ii) The value of the part-number code.
   iii) The value of the version code.
   iv) The method of programming the value of the supplementary identification code, where required.
7) *Performance*. The performance of the test logic should be fully defined, including the following information:
   i) The maximum acceptable TCK clock frequency.
   ii) A full set of timing parameters for the test logic.
   iii) The logic switching thresholds for TAP input and output pins.
   iv) The load presented by the TCK, TMS, TDI, and TRST* pins.
   v) The drive capability of the TDO output pin.
   vi) The extent to which the TDO driver may be overdriven when active (e.g., using an in-circuit test system).

vii) Whether TCK may be stopped in the logic 1 state.

8) *Compliance enable inputs*. If a component has compliance-enable inputs as defined in 4.8.1, the following documentation shall be provided:

i) A complete list of these inputs labeled as compliance-enable inputs.

ii) A complete list of those logic patterns that, when applied at the compliance-enable inputs, will enable compliance to this standard.

iii) A clear indication of any patterns that, if applied to the compliance-enable inputs, would cause hazardous operation of the component.

c) A BSDL description of the component shall be supplied by the component manufacturer (see Annex B).

13.3.2 Description

Figure 13-1 and Figure 13-2 show how set-up and hold timing parameters and propagation delays should be measured relative to the test clock TCK and a reference voltage V_{ref}. Note that such timing parameters are required for TMS, TDI, and TDO and also for system pins that can be driven from the test logic (e.g., the system data input set-up time for the boundary-scan register before the rising edge of TCK in the *Capture-DR* controller state).

Figure 13-1—Measuring set-up and hold timing

Figure 13-2—Measuring propagation delays

Annex A

(informative)

An example implementation using level-sensitive design techniques

To illustrate how a circuit might be constructed to meet the requirements of this standard, example designs are included in the standard. These examples form a consistent set and could be used as the basis for an implementation. However, it is important to emphasize that *the designs contained in this standard are neither mandatory nor recommended in preference to any other implementation.* Many other implementations are possible. For example, this annex illustrates one of many possible implementations of the test logic that could be based on Level-Sensitive Scan Design (LSSD) techniques. This implementation has two modes of operation:

a) A “chip-on-board” mode where the component responds to the signals received at the TAP inputs in the manner required by this standard; and
b) A “stand-alone” mode that allows the entire component (including both the on-chip system logic and the test logic) to be tested using LSSD techniques, for example, as a part of a postproduction test.

The stand-alone mode extends the functionality of the component beyond that required by this standard while maintaining compatibility with this standard for chip-on-board testing.

A.1 Top-level test logic design

The design for the test logic is shown in Figure A.1. The following design features should be noted:

a) All stored-state devices are constructed from level-sensitive latches. Shift-register stages contained in the test logic require two such latches controlled from a pair of nonoverlapping clocks. The clock generator circuit shown in Figure A.2 generates these clocks, as shown in Figure A.3 and Figure A.4. [4]
b) An internal scan path is provided that visits all shift-register latches in the design, including those in the test logic. The internal scan path is shown as a bold line in Figure A.1.
c) For chip-on-board operation, the LSSD clocks LSSD_A, LSSD_B, and LSSD_P are held at 0, while LSSD_C1 and LSSD_C2 are held at 1. This allows the test logic to operate in response to signals received at the TMS, TDI, and TCK inputs, as defined in this standard.
d) For stand-alone component testing using the internal scan path, TCK and the clocks for the on-chip system logic are operated in concert with the LSSD clocks (LSSD_A, LSSD_B, LSSD_P, LSSD_C1, and LSSD_C2). To ensure that LSSD testing of the test logic is correctly synchronized to that of the on-chip system logic, the signals LSSD_C1 and LSSD_C2 are used. These signals are controlled in concert with the remaining LSSD clocks that enable shifting along the scan path. In this mode, the TCK input to the clock generator is used as a control signal that can enable or disable the signals supplied to LSSD_C1 and LSSD_C2. For example, to permit a positive-going pulse on LSSD_C1 to propagate through to C1, a logic 1 must first be applied at TCK. Similarly, positive-going pulses at LSSD_C2 are allowed through to C2 if TCK is first set to 0. Figure A.4 shows the expected relationships between the various clock signals during LSSD stand-alone testing.

[4] The clocks LSSD_C1 and LSSD_C2 are dedicated test clocks that are used only during stand-alone LSSD testing of the component. They allow logic driven from single-phase clock inputs such as TCK to be controlled completely in accordance with level-sensitive design principles during component testing.

Figure A.1—Test logic schematic

Figure A.2—Generation of nonoverlapping clocks from TCK

e) Since the serial outputs of the instruction and test data registers change state on the falling edge of TCK due to the master-slave operation of the latches that form the shift-register stages, it is not necessary to retime the output fed to TDO.

Figure A.3—Operation of the clock generator

Figure A.4—Control of clocks for "stand-alone" component testing

Note that the capture of input signals to the test logic (e.g., those received at TMS, at TDI, and at system input pins) occurs on the falling edge of C1, which itself occurs at a fixed delay after receipt of a rising edge at TCK. Provided that the width of pulse C1 is independent of the frequency of TCK, the requirements of this standard will be met (see 4.2.2).

A.2 Latch designs

Figure A.5 gives NAND gate equivalent circuits for the latches used in the remainder of the schematics in this annex.

A.3 TAP controller implementation

Figure A.6 and Figure A.7 show the implementation of the TAP controller. Note that for this example, the output decoding logic is defined for each register in the following clauses of this annex. The assignments of logic states to controller states are as for the previous implementation (see Table 6-3).

Figure A.5—Schematics for level-sensitive latches

Figure A.6—TAP controller—A

Note the inclusion of the scan test path through the controller latches, which allows the controller to be fully tested as a part of a scan test on the complete integrated circuit.

A.4 Instruction register implementation

To allow for the scan path input to the instruction register, the design of the cell nearest to TDI will differ from that of other cells in the register. Figure A.8 shows a design for the cell nearest to TDI, while Figure A.9 shows a design for other cells.

The control signals for these cell designs are generated from the TAP controller outputs and the various clock signals as follows:

```
CaptureClockIR = C1·Y4·Y3·Y2·Y1*
ShiftClockIR = C1·Y4·Y3*·Y2·Y1*
SetClockIR = C2·Y4L1·Y3L1·Y2L1·Y1L1
L2ClockIR = C2 + LSSD_B
PClockIR = C2·Y4L1·Y3L1·Y2L1*·Y1L1 + LSSD_P
```

Figure A.7—TAP controller—B

Figure A.8—Instruction register cell nearest to TDI

Figure A.9—Other instruction register cells

A.5 Bypass register implementation

The bypass register can be implemented as shown in Figure A.10.

Figure A.10—Bypass register

Control signals for this implementation are generated as follows:

```
CaptureClockByp = C1·Y4*·Y3·Y2·Y1*·BYPASS
ShiftClockByp = C1·Y4*·Y3*·Y2·Y1*·BYPASS
L2ClockByp = C2 + LSSD_B
```

Note that the variable BYPASS is true when the *BYPASS* instruction is present at the instruction register outputs.

A.6 Boundary-scan register implementation

A set of boundary-scan register cell designs is included in the following figures. The control signals required by these cells are as follows:

```
CaptureClockBS = C1·Y4*·Y3·Y2·Y1*·BST
ShiftAClockBS = C1·Y4*·Y3*·Y2·Y1*·BST + LSSD_A
```

```
L2ClockBS = C2 + LSSD_B
PClockBS = C2·Y4L1*·Y3L1·Y2L1*·Y1L1·BST + LSSD_P
ResetClockBS = C2·Y4L1·Y3L1·Y2L1·Y1L1
```

The variable BST is true whenever a boundary-scan test instruction is present at the instruction register's outputs. The signals labeled DriveOut and LogicTest are controlled as follows:

a) DriveOut is true when the *EXTEST*, *INTEST*, or *RUNBIST* instruction is selected.
b) LogicTest is true when the *INTEST* or *RUNBIST* instruction is selected.

Note that the boundary-scan register nearest to TDI should have its first latch modified to accept either the signal received from TDI (when C1 · Y4* · Y3* · Y2 · Y1* · BST is true) or the signal at the scan input to the register (when LSSD_A is true).

Figure A.11—A level-sensitive input cell design

Figure A.12—A level-sensitive output cell design

The parallel output latches in the control cells in Figure A.13 and Figure A.14 are reset in the *Test-Logic-Reset* controller state as allowed by Permission 11.3.1 h) and Rule 11.6.1 p). A similar reset capability is provided in Figure A.11 and Figure A.12.

Figure A.13—Level-sensitive cells at a 3-state output

Figure A.14—Level-sensitive cells at a bidirectional pin

Annex B

(normative)

Boundary-scan description language

This annex defines a machine-readable language that allows rigorous description of testability features in components that comply with IEEE Std 1149.1. The language is called the Boundary-Scan Description Language (BSDL). It is a subset and standard practice of the VHSIC Hardware Description Language (VHDL) (IEEE Std 1076-1993).

B.1 General information

B.1.1 Document outline

Clauses B.2 to B.4 define the purpose of BSDL, its scope, and its relationship to VHDL. Clauses B.5 to B.7 describe general characteristics of the language. Clause B.8, the entity description, describes the overall structure of a BSDL description. B.8.2 to B.8.18 are detailed descriptions of each of the mandatory and optional sections of a BSDL description. They are documented in the order in which they should appear in such a description. Clause B.9 describes the Standard VHDL Package, and Clause B.10 describes a user-supplied VHDL package. Clause B.11 includes some special cases for purposes of illustration and indicates how such components can be specified in BSDL. Clause B.12 shows a typical BSDL description. Clause B.13 briefly documents the 1990 version of BSDL (see also B.1.3). Clause B.14 documents the 1994 version of BSDL (the first version approved by the IEEE Standards Board and published by the IEEE).

In B.8.2 to B.8.18, the detailed descriptions of each of the mandatory and optional sections of a BSDL description are organized in the following way:

— Short introduction
— Syntax
— Notes
— Example and additional text
— Semantic checks

The syntax and semantic subclauses are where the language is defined; however, sometimes material in short introductions, notes, and examples is key to a full understanding of the language. Therefore, such text is also part of this definition. The notes often provide the semantic interpretation of various lexical elements.

Commonly used syntactic elements are defined in B.6.3.

B.1.2 Conventions

a) Examples are printed in `Courier` font.
b) See Clause B.6 for conventions relating to syntax.
c) For clarity, all reserved words, predefined words, and punctuation are shown in **bold Helvetica** type within this document. VHDL reserved and predefined words will be shown in **lowercase** letters, and BSDL reserved words will be shown in **UPPERCASE** letters. (BSDL itself is case-insensitive; this convention is adopted for clarity.)

B.1.3 BSDL history

The development of BSDL started soon after the first promulgation of this standard in 1990. When, out of common self-interest, an industrywide group of companies implementing tools to support this standard realized that a single language for describing the boundary-scan implementations in components would be of benefit, many tools were built using early draft specifications of the BSDL language.

Developments made both to this standard and to the BSDL definition since the language was first proposed in 1990 have resulted in the obsolescence of some of the constructs from the first draft versions of the language. Some constructs were rendered unnecessary as a result of the standardization of the *CLAMP* and *HIGHZ* instructions in 1993, while others were found to duplicate information provided elsewhere in a BSDL description and thus were removed as redundant.

Because a significant number of BSDL descriptions have been written based on the 1990 draft version of the language and because these descriptions are likely to remain in circulation for some time, implementors of tools based on BSDL may wish to design them to read both the BSDL language defined in this annex and the earlier 1990 version. Information on how to do this is contained in Clause B.13.

The first IEEE promulgation of a definition for BSDL was the 1994 version. As this standard has been revised for 2001, the definition of BSDL has been modified to correspond. Implementors of tools based on BSDL shall design them to accommodate both the 2001 version of the BSDL language defined in this annex and the earlier 1994 version. Information on how to do this is contained in Clause B.14.

B.2 Purpose of BSDL

BSDL provides a machine-readable means of representing some parts of the documentary information specified in 13.3. The scope of the language is defined in Clause B.3.

The goal of the language is to facilitate communication between companies, individuals, and tools that need to exchange information on the design of test logic that complies with this standard. For example,

- A vendor of a component that supports this standard shall supply a BSDL description to purchasers.
- Automated test-generation tools may use a library of BSDL descriptions to allow the generation of a test for a particular loaded board.
- The test logic defined by this standard could be synthesized with the support of a BSDL description.

BSDL describes “finished” silicon, not “work-in-progress.” For example, when a bare die conforming to this standard is produced, a BSDL description can be provided for it. A die may be inserted into one or more types of component packages as well, with each variation described in BSDL (see B.8.7). BSDL for partially synthesized test logic is considered “work-in-progress,” is not necessarily compliant with this annex, and in most cases should not be transmitted beyond the synthesis environment.

B.3 Scope of BSDL

BSDL is not a general-purpose hardware description language—it is intended solely as a means of describing key aspects of the implementation of this standard within a particular component. A BSDL description is not itself a simulation model. Examples of features that are and are not described using BSDL are listed in Table B.1.

Table B.1—Scope of BSDL

Features described by BSDL	Features that cannot or need not be described
Length and structure of the boundary-scan register	TAP-controller state diagram
Availability of the optional TRST* pin	Bypass register
Physical locations of the TAP pins	Length of the device-identification register
Instruction binary codes	Provision of *BYPASS*, *SAMPLE*, *PRELOAD*, and *EXTEST* instructions
Device-identification code	Operation of user-defined instructions

Note that the language describes only features of the test logic that can vary from component to component, depending on the choice of the component designer. Features that are completely specified by this standard (without option) are not required to be described in BSDL but may be described if users so wish.

Further, BSDL does not have a general means for providing for the specification of logic levels, timing parameters, power requirements, and similar factors. These data do not affect the logical behavior of an implementation and most likely are already described in other parts of the specification of any given component.

B.4 Relationship of BSDL to VHDL

BSDL is a subset of VHDL (IEEE Std 1076-1993) and shall be fully conformant to the requirements of the VHDL standard.

However, the user must be prepared to modify a BSDL description in the face of implementation dependencies in VHDL-based tools. No way has been found to avoid this small amount of effort without introducing further undesirable complications. Specifically, the <standard use statement> (see B.8.4) and the <use statement> (see B.8.5) may require editing because of tool and file system dependencies. The syntax of the statements as defined is legal VHDL; however, an additional prefix (identifying a library in which the Standard VHDL Package will be found) will need to be added for some applications. A syntax lacking such a prefix has been chosen to force an error in such an application rather than risk unpredictable and confusing errors due to the inclusion of an inappropriate prefix.

NOTE—In the event of an error or omission in this annex regarding VHDL syntax, IEEE Std 1076-1993 shall take precedence.

Let it be noted that

a) BSDL does not employ all the syntactic elements of VHDL. Only those elements required to meet the scope of BSDL are used. The following VHDL statements shall be employed in BSDL:

attribute	**generic**	**subtype**
constant	**package**	**type**
end	**package body**	**use**
entity	**port**	

In some cases, only a subset of a particular VHDL language element syntax is used in BSDL. Descriptions of the lexical elements and statement syntax for each syntax element are contained in Clauses B.5 through B.8 and B.10. Historical information on such elements is found in Clause B.13.

b) For cases in which a feature could be described in several ways within VHDL, a restricted set of ways has been selected and defined explicitly as the standard practice for BSDL. This restriction simplifies the application of the VHDL subset for BSDL, particularly for tools that are required only to read or generate BSDL (i.e., tools that have no requirement to read or write the full VHDL language).

c) BSDL imposes additional requirements on the syntax and content of certain character strings, that is, sequences of characters between quotation marks (e.g., "**EXTEST**"). A VHDL parser will not check the information in these strings. In contrast, a BSDL parser shall check that the information in the strings is appropriate for the relevant parameters or attributes for which such strings might be values. There are several string syntaxes defined within BSDL.

B.5 Lexical elements of BSDL

The lexical elements of BSDL shall be a subset and standard practice of those of VHDL as defined in IEEE Std 1076-1993. The following subclauses enumerate the lexical elements needed to understand the BSDL language definition.

B.5.1 Character set

The language shall not be case-sensitive; that is, for example, the character **a** shall be considered identical to the character **A**. Therefore, the following names are identical:

```
FRED        Fred        fred
```

The following characters shall be permitted within the language:

a) Upper- and lowercase letters: **A** to **Z** and **a** to **z** (the language is not case-sensitive)
b) Digits: **0** to **9**
c) Special characters: **" & ' ( ) * , - . : ; < = >** _ (Note that this list is smaller than that of VHDL.) Other special characters shall be allowed as part of a comment (see B.5.3) or as part of the string element of a design warning (see B.8.18).
d) Logical separators: The space character and VHDL format effectors shall be used as logical separators. VHDL format effectors are the characters called horizontal tabulation, vertical tabulation, carriage return, line feed, and form feed.

Logical separators shall have two purposes. They shall be used to eliminate lexical ambiguity by separating lexical tokens such as reserved words and/or identifiers. For example, the reserved word **entity** must be separated from the component name identifier that immediately follows it rather than being run together with it. Logical separators also may be used in combinations to create visually appealing layouts.

B.5.1.1 Identifiers

Identifiers shall be user-supplied names and reserved words functioning as names. Identifiers shall start with a letter and may contain letters, digits, or the underscore character. For example, the following are valid identifiers:

```
BSDL
IEEE_Std_1149_1
```

There shall be no upper limit to the number of characters in an identifier.

The underscore character (_) shall not be allowed to be the last character in an identifier (by VHDL).

```
IEEE_STD_1149_          -- This is not a legal identifier.
```

Adjacent underscore characters (_ _) shall not be allowed (by VHDL).

B.5.1.2 BSDL reserved words

The identifiers listed in this subclause shall be BSDL reserved words and shall have a fixed significance in the language. These identifiers shall not be used for any purpose in a BSDL description other than as part of a comment. For example, a reserved word shall not be used as an explicitly declared identifier. **BC_0** to **BC_99** shall be variable names used in the Standard VHDL Package. Names **BC_0** through **BC_10** are used today, while **BC_11** through **BC_99** shall be reserved for future use. Similarly, all names that start with **STD_1149_** shall be reserved.

AT_PINS
BC_0 to BC_99
BIDIR
BIDIR_IN
BIDIR_OUT
BOTH
BOUNDARY
BOUNDARY_LENGTH
BOUNDARY_REGISTER
BSCAN_INST
BSDL_EXTENSION
BYPASS
CAP
CAP_DATA
CAPTURES
CELL_DATA
CELL_INFO
CELL_TYPE
CLAMP
CLOCK
CLOCK_INFO
CLOCK_LEVEL
COMPLIANCE_PATTERNS
COMPONENT_CONFORMANCE
CONTROL
CONTROLR
INSTRUCTION_OPCODE
INSTRUCTION_PRIVATE
INTERNAL
INTEST
INTEST_EXECUTION
KEEPER
LOW
OBSERVE_ONLY
OBSERVING
ONE
OUTPUT2
OUTPUT3
PHYSICAL_PIN_MAP
PI
PIN_MAP
PIN_MAP_STRING
PO
PORT_GROUPING
PRELOAD
PULL0
PULL1
REGISTER_ACCESS
RUNBIST
RUNBIST_EXECUTION
SAMPLE
STD_1149_*

DESIGN_WARNING	**TAP_SCAN_CLOCK**
DEVICE_ID	**TAP_SCAN_IN**
DIFFERENTIAL_CURRENT	**TAP_SCAN_MODE**
DIFFERENTIAL_VOLTAGE	**TAP_SCAN_OUT**
EXPECT_DATA	**TAP_SCAN_RESET**
EXTEST	**UPD**
HIGHZ	**USERCODE**
ID_BITS	**USERCODE_REGISTER**
ID_STRING	**WAIT_DURATION**
IDCODE	**WEAK0**
IDCODE_REGISTER	**WEAK1**
INPUT	**X**
INSTRUCTION_CAPTURE	**Z**
INSTRUCTION_LENGTH	**ZERO**

NOTE—In the list of reserved words above, the entry **STD_1149_*** is to be interpreted to mean all names that start with **STD_1149_**, for example, **STD_1149_1_2001**.

B.5.1.3 VHDL reserved and predefined words

The identifiers listed below shall be called VHDL (IEEE Std 1076-1993) reserved and predefined words and shall have a fixed significance in the language. These identifiers shall not be used for any purpose in a BSDL description other than as part of a comment. For example, a reserved word shall not be used as an explicitly declared identifier. Reserved words shown in the list below in lowercase letters are part of the BSDL subset of VHDL. Those in uppercase letters are not part of BSDL but shall not be used as identifiers. The VHDL reserved words are shown in this subclause for convenience, *but the latest edition of the VHDL standard shall be consulted as the final authority*.

ABS	**GUARDED**	**REGISTER**
ACCESS	**IF**	**REJECT**
AFTER	**IMPURE**	**REM**
ALIAS	**in**	**REPORT**
all	**INERTIAL**	**RETURN**
AND	**inout**	**ROL**
ARCHITECTURE	**is**	**ROR**
array	**LABEL**	**SELECT**
ASSERT	**LIBRARY**	**SEVERITY**
attribute	**linkage**	**SHARED**
BEGIN	**LITERAL**	**signal**
bit	**LOOP**	**SLA**
bit_vector	**MAP**	**SLL**
BLOCK	**MOD**	**SRA**

body	**NAND**	**SRL**
buffer	**NEW**	**string**
BUS	**NEXT**	**subtype**
CASE	**NOR**	**THEN**
COMPONENT	**NOT**	**to**
CONFIGURATION	**NULL**	**TRANSPORT**
constant	**of**	**true**
DISCONNECT	**ON**	**type**
downto	**OPEN**	**UNAFFECTED**
ELSE	**OR**	**UNITS**
ELSIF	**OTHERS**	**UNTIL**
end	**out**	**use**
entity	**package**	**VARIABLE**
EXIT	**port**	**WAIT**
FALSE	**positive**	**WHEN**
FILE	**POSTPONED**	**WHILE**
FOR	**PROCEDURE**	**WITH**
FUNCTION	**PROCESS**	**XNOR**
GENERATE	**PURE**	**XOR**
generic	**range**	
GROUP	**record**	

B.5.2 Strings

A string shall be defined as a sequence of zero or more characters enclosed between quotation marks. A quotation mark character shall not be allowed within a string in BSDL. For example,

```
"Mary had a little lamb"-- Allowed
"Fred said ""HELP"""-- Not allowed
```

Strings are used extensively in BSDL. Since many of the BSDL strings are potentially much longer than a single line, the concatenation operator **&** shall be used to break them into manageable pieces. For example,

```
"Mary had a little lamb. " &
"Its fleece was white as snow."
```

is a single string identical to

```
"Mary had a little lamb. Its fleece was white as snow."
```

BSDL shall not permit replacement of the quotation mark with any other character.

A string literal shall fit on one line since it is a lexical element. Therefore, the only legal VHDL format effector in a string literal shall be horizontal tabulation.

B.5.3 Comments

Text between a double dash (**--**) symbol and the end of a line shall be treated as a comment. The text shall be allowed to contain any special character allowed by VHDL, not just those given in item c) of B.5.1.
Comments syntactically terminate a line of a description. Comments may be interspersed with lexical elements. For example, the following represents a single VHDL string:

```
"This is all" &              -- An example of a string split by a comment
" a single string"
```

B.6 Notes on syntax definition

B.6.1 BNF conventions

The syntax of BSDL shall be presented in Backus-Naur Form (BNF) as follows:

a) Any item enclosed in chevrons (i.e., between the character “<” and the character “>”) shall be the name of a syntax item that will be defined in this annex.
b) Items enclosed between braces (i.e., between the character “{” and the character “}”) shall be either omitted or included one or more times.
c) Items enclosed between square brackets (i.e., between the character “[” and the character “]”) shall be either omitted or included only one time.
d) Items enclosed between the characters “ ý ” and “ í ” may appear in any order.
e) Except with regard to case, text shown in **bold Helvetica** type shall be included exactly as it is presented in this annex.
f) Alternative syntaxes shall be separated by a vertical bar (“|”).
g) The symbol “::=” shall be read as “shall be defined as.” Note that the nonboldface “::=” is only part of a BNF description; in the BSDL file, the boldface characters “ **:=** ” are used to indicate assignment.
h) White space (spaces, tabulation, carriage returns, etc.) is used in these BNF descriptions to provide enhanced readability and is not part of the syntax. However, white space needed for resolving lexical ambiguity (logical separation) is required as described in B.5.1.

B.6.2 Lexical atoms

a) A <VHDL identifier> shall be any valid identifier chosen as a name for an item (see B.5.1.1).
b) An <integer> shall be any valid unsigned VHDL integer made up of an unsigned numeric character sequence not containing an underscore (_) character and not using an exponent field.
c) A <real number> shall be any valid VHDL real number of the form <integer> **.** <integer> or <integer> **.** <integer> **E** <integer> all written contiguously without spaces or format effectors. Note that 1E3 is not real because it does not contain a decimal point. The number **20.0E6** is real, as is **20000000.0** .
d) A <pattern> shall be a contiguous sequence of one or more **0** , **1** , and **X** characters containing no spaces or format effectors. For example, **001X00** and **XX010X** are legal. However, **100 X00** is not legal because of the embedded space. A low state (see 3.1.15) shall be denoted by **0** , a high state (see 3.1.9) shall be denoted by **1** , and a don’t-care value shall be denoted by **X** .
e) A <32-bit pattern> shall be a <pattern> with exactly 32 characters in its character sequence.
f) A <left bracket> shall be the left bracket character (**[**).
g) A <right bracket> shall be the right bracket character (**]**).
h) A <string> shall be any valid string or concatenated string structure meeting the definition in B.5.2.

Lexical ambiguity exists in certain situations and shall be resolved by context. For example, a <pattern> that starts with an **X** shall be differentiated from a <VHDL identifier> by context derived from the syntax. Similarly, a <pattern> that does not contain an **X** shall be differentiated from an integer such as **100** (one hundred).

B.6.3 Commonly used syntactic elements

a) A <port ID> shall identify a component signal that may be used to interface to external signals. A port shall be dimensioned as either a **bit** or a **bit_vector** (see B.8.3). Subscripted names shall be allowed only when **bit_vector** -dimensioned port signals are used.

<port ID>::= <port name> | <subscripted port name>
<port name>::= <VHDL identifier>
<subscripted port name>::= <VHDL identifier> (<subscript>)
<subscript>::= <integer>

b) An <instruction name> shall be an instruction name defined in this standard or an instruction named by the manufacturer of a component.

<instruction name>::= **BYPASS** | **CLAMP** | **EXTEST** | **HIGHZ** |
IDCODE | **INTEST** | **PRELOAD** | **RUNBIST** |
SAMPLE | **USERCODE** | <VHDL identifier>

NOTE—In earlier editions of this standard, the BSDL reserved word **PRELOAD** did not exist and **SAMPLE** was used as an abbreviated name when the *SAMPLE* and *PRELOAD* instructions were merged [see Permission 8.1.1 g) and Rule B.8.11.3 f)].

B.7 Components of a BSDL description

A BSDL description shall be composed of the following parts, as shown in part in Figure B.1. These parts are

a) *The entity description.* An entity description shall be written for each component. It shall specify component-specific parameters of the test logic and might, for example, be written by the designer of a component.
b) *The Standard VHDL Package and Standard VHDL Package Body*. These parts shall contain two types of information:
   1) The Standard VHDL Package shall give a definition of the BSDL subset of VHDL, which is required to allow BSDL descriptions to be read by a VHDL-conformant parser.
   2) The Standard VHDL Package Body shall give definitions of commonly used types of boundary-scan register cells.

Typically, the Standard VHDL Package and Standard VHDL Package Body would reside with system-accompanying software that utilizes BSDL descriptions. Individual BSDL descriptions are not burdened with having to include all the elements contained in the Standard VHDL Package and Standard VHDL Package Body.

NOTE—The Standard VHDL Package and Standard VHDL Package Body shall be exactly as listed in Clause B.9. They shall not be modified and would not normally be shipped along with BSDL files that describe particular components. Normally, the Standard VHDL Package and Standard VHDL Package Body will be supplied as a part of a BSDL-compliant tool and will be maintained by the tool supplier.

The cell definitions provided in the Standard VHDL Package Body could have been given within the Standard VHDL Package in a full VHDL implementation. The advantage of a package body is that the

Figure B.1—Components of a BSDL description

information it contains can be updated without causing the need for recompilation of all entities that reference the package. If a package is modified, recompilation is necessary. The package with package body structure is a standard practice of BSDL.

c) *User-specified VHDL packages and VHDL package bodies*. Users may provide VHDL packages and package bodies that define boundary-scan register cell designs specific to any group of components. A vendor of application-specific ICs (ASICs), for example, could provide a user-specified VHDL package and package body to describe the particular boundary-scan register cell designs offered. Global BSDL extensions also could be provided in user-specified VHDL packages (see Clause B.10 and B.8.17).

B.8 The entity description

The entity description and supporting VHDL packages shall make up a BSDL model of the component and are, in effect, the electronic data sheet for its test logic. It shall contain statements through which parameters that may vary from one chip to another are defined, as discussed in Clause B.3.

B.8.1 Overall structure of the entity description

B.8.1.1 Syntax

The BSDL entity description shall have the following structure:

<BSDL description>::=

entity <component name> **is**

<generic parameter>	(see B.8.2)
<logical port description>	(see B.8.3)
<standard use statement>	(see B.8.4)
{<use statement>}	(see B.8.5)
<component conformance statement>	(see B.8.6)
<device package pin mappings>	(see B.8.7)
[<grouped port identification>]	(see B.8.8)
<scan port identification>	(see B.8.9)
[<compliance enable description>]	(see B.8.10)
<instruction register description>	(see B.8.11)
[<optional register description>]	(see B.8.12)
[<register access description>]	(see B.8.13)
<boundary-scan register description>	(see B.8.14)
[<runbist description>]	(see B.8.15)
[<intest description>]	(see B.8.16)
{<BSDL extensions>}	(see B.8.17)
[<design warning>]	(see B.8.18)

end <component name> **;**

<component name>::= <VHDL identifier>

NOTE—While VHDL permits some elements within an entity description to be in an arbitrary order, the fixed ordering above shall be required for BSDL. This ordering is defined to ease the development of tools that are not themselves required to be fully VHDL-compliant.

The <component name> shall identify a particular integrated circuit and should be distinct from the names of all other components that may be used together (i.e., that may be assembled onto a single loaded board).

B.8.1.2 Semantic checks

a) Any <component name> referenced in any attribute statement shall be the same as the component name declared in the entity description.

B.8.2 Generic parameter statement

The <generic parameter> statement shall facilitate selection between multiple component packaging options that are described within the BSDL description (see B.8.7). Each such option shall define a mapping between the physical package pins of the component and the <port ID> elements of the component (see B.6.3).

The component package option relevant to a particular instance of a component shall be identified each time a given BSDL description is referenced.

B.8.2.1 Syntax

<generic parameter>::=
generic (PHYSICAL_PIN_MAP: string); |
generic (PHYSICAL_PIN_MAP: string := <default device package type> **);**

<default device package type>::= **"** <VHDL identifier> **"**

In the first alternative <generic parameter> statement syntax, a <pin mapping name> (see B.8.7) that identifies a component package option shall be supplied with the BSDL description when it is referenced. In the second alternative syntax, a value is given for the <default device package type> that shall be used as the default in the case in which no <pin mapping name> is supplied with the BSDL description when it is referenced. The <default device package type> shall be the quoted name of a <pin mapping name> used to specify the pin mapping for the component (see B.8.7).

B.8.2.2 Examples

```
generic (PHYSICAL_PIN_MAP: string);
```

or

```
generic (PHYSICAL_PIN_MAP: string := "DW");
```

NOTE—It is recommended that the component package name from the data sheet of a given component be used as the relevant <pin mapping name>, for example, SSOP_56, PQFP_84, or PGA_18x18.

B.8.2.3 Semantic checks

a) It shall be an error if
   1) No <default device package type> is specified in the <generic parameter> statement of a BSDL description and no package <pin mapping name> (see B.8.7) is specified when a BSDL description is processed
   2) The <default device package type> specified in the <generic parameter> statement is not identical to any <pin mapping name> appearing in any <pin mapping> (see B.8.7) and no <pin mapping name> is specified when the BSDL description is processed
   3) A <pin mapping name> is specified when a given BSDL description is processed and that <pin mapping name> is not identical to any <pin mapping name> appearing in any <pin mapping> (see B.8.7) of that description

B.8.3 Logical port description statement

The BSDL port description shall be a specialization of the VHDL port list. It shall be used to assign meaningful symbolic names to the pins of a component. These symbolic names, which shall be used in subsequent statements in the description, allow the majority of such statements to be independent of a renumbering or other reorganization of the pins of the component.

It is strongly recommended that the existence of nondigital pins (such as power, ground, no-connects, or analog signals) should be recorded in the BSDL description of the component through use of the **linkage** <pin type> defined in the following syntax. BSDL applications may then be able to support substantially improved error detection during board testing; however, only a warning shall be issued in the complete absence of such pins within a description since it is technologically possible for a component not to have any such pins.

B.8.3.1 Syntax

<logical port description>::= **port** (<pin spec> { **;** <pin spec>} **);**

<pin spec>::= <identifier list>: <pin type> <port dimension>
<identifier list>::= <port name> {,<port name>} (see B.6.3)
<pin type>::= **in** | **out** | **buffer** | **inout** | **linkage**
<port dimension>::= **bit** | **bit_vector** (<range>)
<range>::= <integer_1> **to** <integer_2> | <integer_2> **downto** <integer_1>
<integer_1>::= <integer>
<integer_2>::= <integer>

The definitions of the possible values of <pin type> are given in Table B.2, and <port name> is defined in B.6.3.

Table B.2—Pin types

Value	Meaning
in	An input-only pin
out	An output-only pin that may be connected to a bus wire driven by multiple drivers (e.g., a 3-state or open-collector output)
buffer	A 2-state output-only pin where either state is always actively driven (e.g., cannot be connected to a bus wire)
inout	A bidirectional pin
linkage	Other pin types, such as power, ground, analog, and no-connect

NOTE—Where a 2-state output pin has a 3-state mode only to meet the requirements of the *HIGHZ* instruction, it shall be described as a buffer.

The <port dimension> shall define the number of signals that constitute a port. If the <port dimension> is **bit**, one <port name> shall correspond to that one signal. If the <port dimension> is **bit_vector**, one <port name> shall correspond to a collection of *n* signals that shall be individually referenced by subscripting the port name, i.e., by a <subscripted port name> (see B.6.3).

NOTE—The signal types **bit** and **bit_vector** are signal types known to VHDL and shall be the only signal types permitted in BSDL. Also, while VHDL allows multiple, possibly broken ranges to be specified for a **bit_vector**, BSDL syntax shall allow only a single unbroken range.

B.8.3.2 Examples

```
port( TDI, TMS, TCK: in bit;
         TDO: out bit; IN1, IN2: in bit;
         OUT1: out bit; OUT2: buffer bit;
         OUT3: out bit_vector (1 to 8);
         OUT4: out bit_vector (4 downto 1);
         BIDIR1, BIDIR2, BIDIR3: inout bit;
         GND, VCC: linkage bit);
```

B.8.3.3 Semantic checks

a) A **bit_vector** declared in a <logical port description> statement shall have the value of <integer_1> less than or equal to the value of <integer_2>.

b) Each <port name> appearing in an <identifier list> of a <logical port description> shall occur only once within the <logical port description> statement.

B.8.4 Standard use statement

B.8.4.1 Syntax and semantics

The <standard use statement> shall identify one Standard VHDL Package in which attributes, types, constants, and other elements are defined that shall be referenced elsewhere in the BSDL description.

B.8.4.1.1 Syntax

<standard use statement>::= **use** <standard VHDL package identifier> **.all;**

<standard VHDL package identifier>::=
STD_1149_1_1994 | **STD_1149_1_2001** | <other package identifier>

<other package identifier>::= <VHDL identifier>

The <standard VHDL package identifier> shall be the name of the Standard VHDL Package that contains the information to be included. The suffix **.all** shall indicate that all declarations within the VHDL package are to be used.

Values for <other package identifier> may be assigned with future revisions of this annex as the language evolves. For toolmakers desiring to support the earlier draft version of BSDL as well as the version specified in this annex, the value of <other package identifier> may be **STD_1149_1_1990** (see B.13.1).

NOTE—The construct of the <standard use statement> may not be syntactically complete for use by a given VHDL analyzer. This is because a library or default working area has not been specified. In such a case, a complete statement is "use work.STD_1149_1_2001.all;" in which the preface "work." tells a VHDL analyzer to find the Standard VHDL Package in the current work area. The field "work." could be replaced by an arbitrary library name such as "BSCAN." telling a VHDL analyzer where in its system of libraries to find the Standard VHDL Package. Since there is no standardization of library structures from one VHDL environment to another, some editing of BSDL files to specify the location of Standard VHDL Packages is generally unavoidable. The specification used in this subclause may cause an error in a VHDL analyzer, forcing the user to edit the BSDL file for the correct location of the Standard VHDL Package information.

The <standard VHDL package identifier> shall indicate

— An instruction to tools to read a standard package that contains BSDL syntax definitions
— The version of the BSDL language (shown by the year number embedded in the identifier) that was used when creating the BSDL file

As a general rule, future enhancements to the BSDL language will be designed as extensions to previous versions of the language. Therefore, the version information contained in the <standard VHDL package identifier> may be used by BSDL-specific tools to indicate which tool versions will be able to process the information in the input file. For example, if the input file records the version as "**_1994**", tools that can process any language variant "**_1994**" or later will be able to process the input file correctly.

NOTE—Versions of the BSDL language should not be confused with the version of the standard itself to which a given IC may conform. The <component conformance statement> identifies the version of the standard (see B.8.6).

Within a specific system processing BSDL descriptions, the Standard VHDL Package may be a file somewhere in the file system of the host computer. The **.all** suffix is meaningful to VHDL and is not part of a file name. While VHDL permits the use of a wider range of suffixes, **.all** shall be the only suffix permitted in BSDL.

The content of the Standard VHDL Package shall be the current definition of the BSDL language and shall not be modified by users.

The <standard use statement> shall appear in every BSDL description before any <use statement> (see B.8.5) so that tools that are fully VHDL-compliant can locate information relevant to all components that conform to this standard. (Tools that are limited to the BSDL language shall always use this information.)

B.8.4.1.2 Examples

```
use STD_1149_1_2001.all;                -- Today
```

or

```
use STD_1149_1_2012.all;                -- Sometime in the future
```

The contents of the **STD_1149_1_2001** Standard VHDL Package and its associated Standard VHDL Package Body shall be as listed in Clause B.9. (The contents of the **STD_1149_1_1990** Standard VHDL Package and its associated Standard VHDL Package Body shall be as listed in B.13.1. The contents of the **STD_1149_1_1994** Standard VHDL Package and its associated Standard VHDL Package Body shall be as listed in B.14.1.)

B.8.4.1.3 Semantic checks

a) The Standard VHDL Package content shall be used when processing a BSDL description.

The <standard use statement> also shall be used for version control (see B.8.4.2).

B.8.4.2 Version control

At the time this standard was approved, there were three versions of BSDL descriptions: the 1990 version (see Clause B.13), the 1994 version (approved by the IEEE Standards Board and published by the IEEE in 1994, see Clause B.14), and the 2001 version described here. The <standard use statement> shall indicate whether the BSDL description has been written using the provisional syntax published in the *Proceedings of the International Test Conference* in 1990 or according to this annex. This additional application of the <standard use statement> is intended to provide backwards compatibility to all BSDL descriptions already written and can be used in a similar manner for BSDL descriptions based on future revisions of this annex. It is intended that a parser handling a form of BSDL described in a version of this annex *approved by the IEEE Standards Board and published by the IEEE* shall handle all forms of BSDL described in previous versions of this annex *approved by the IEEE Standards Board and published by the IEEE*.

B.8.4.2.1 Examples

```
use STD_1149_1_2001.all;
```

In the 1990 version of BSDL, the following syntactic elements are not supported:

<component conformance statement>
<grouped port identification>
<compliance enable description>

<runbist description>
<intest description>
<BSDL extensions>

Also, in the 1990 version, cell types **BC_0** and **BC_7** that are defined in this 2001 version (as originally identified and defined by the 1994 version) need to be specified in a user-supplied VHDL package if they are to be referenced. Further, in the 1990 and 1994 versions, cell types **BC_8**, **BC_9**, and **BC_10,** which are identified and defined in this 2001 version, need to be specified in a user-supplied VHDL package if they are to be referenced.

In the 1990 and 1994 versions, the identifiers **KEEPER** and **PRELOAD** were not BSDL reserved words. **KEEPER** and **PRELOAD** become BSDL reserved words in this 2001 version.

Software that processes BSDL descriptions needs to have access to all Standard VHDL Packages related to BSDL. See Clauses B.9, B.13, and B.14 for the three that are currently defined.

B.8.4.2.2 Semantic checks

a) A BSDL parser shall be expected to reject and/or issue warning/error messages when it encounters unrecognized text.

NOTE—Specifically constructed foreign attributes may be added via BSDL extensions (see B.8.17).

B.8.5 Use statement

The optional <use statement> shall identify a VHDL package in which specific attributes or constants are defined so that they can be referenced elsewhere in a BSDL description.

B.8.5.1 Syntax

<use statement>::= **use** <user VHDL package identifier> **.all;**

<user VHDL package identifier>::= <VHDL identifier>

The <user VHDL package identifier> in the <use statement> shall be the name of the VHDL package that contains the information to be included. The suffix **.all** shall indicate that all declarations within the VHDL package are to be used. The **.all** suffix is meaningful to VHDL and is not part of a file name. While VHDL permits the use of a wider range of suffixes, **.all** shall be the only suffix permitted in BSDL. See B.10.1 for the content of a user-supplied package.

B.8.5.2 Examples

```
use Private_Package.all;                    -- Identifies the proprietary VHDL
                                            -- package of a user
```

NOTE—The construct of the <use statement> may not be syntactically complete for use by a given VHDL analyzer. This is because a library or default working area has not been specified. In such a case, a complete statement is “use work.Private.all;” in which the preface “work.” tells a VHDL analyzer to find the user-supplied VHDL package in the current work area. The field “work.” could be replaced by an arbitrary library name such as “BSCAN.” telling a VHDL analyzer where in its system of libraries to find the user-supplied VHDL package. Since there is no standardization of library structures from one VHDL environment to another, some editing of BSDL files to specify the location of a user-supplied VHDL package is generally unavoidable. This specification may cause an error in a VHDL analyzer, forcing the user to edit the BSDL file for the correct location of the user-supplied VHDL package information.

In full VHDL, two or more use statements may be given which reference VHDL packages that contain different definitions of the same item, such as a Boundary-Scan Register Cell. Later, the ambiguity of which

package is being referenced can be removed by qualifying each reference in a specified manner. This facility shall not be supported in BSDL.

B.8.5.3 Semantic checks

a) Information in a VHDL package named in a <use statement> always shall be used when processing a BSDL description (see Clause B.10).
b) Identifiers declared in VHDL packages shall not be defined
   1) More than once in a single VHDL package
   2) In more than one VHDL package referenced by a <use statement> or <standard use statement> in a single BSDL description

B.8.6 Component conformance statement

The <component conformance statement> shall identify the edition of this standard to which the testability circuitry of a physical component conforms.

It is possible for a component designed in 1990 to be described by the version of BSDL defined by this annex, but this cannot imply that the component conforms to the rules of IEEE Std 1149.1-2001. For example, IEEE Std 1149.1-1990 allowed (by omission of rules) two drivers controlled by a single control cell to be disabled by opposing values loaded into that cell. This is explicitly forbidden in IEEE Std 1149.1-2001 (as it was originally in IEEE Std 1149.1a-1993). The <component conformance statement> allows tools to account for changes in the rules that may occur in past and future editions of this standard.

B.8.6.1 Syntax

<component conformance statement>::= **attribute COMPONENT_CONFORMANCE of**
<component name> **: entity is** <conformance string> **;**

<conformance string>::= **"** <conformance identification> **"**
<conformance identification>::=
STD_1149_1_1990 |**STD_1149_1_1993** |**STD_1149_1_2001**

The symbol **STD_1149_1_1990** shall refer to IEEE Std 1149.1-1990. **STD_1149_1_1993** shall refer to to IEEE Std 1149.1a-1993. **STD_1149_1_2001** shall refer to IEEE Std 1149.1-2001. Subsequent editions of this annex may add new values to the <conformance identification> element.

B.8.6.2 Examples

```
attribute COMPONENT_CONFORMANCE of My_Old_IC:entity is
        "STD_1149_1_1990";              -- 1990 component described
                                        -- in this annex
```

B.8.6.3 Semantic checks

No semantic check is necessary for this statement. However, some semantic checks described in this annex shall be influenced by the value that appears in the <conformance identification> element [see, for example, semantic check B.8.14.4 p)].

B.8.7 Device package pin mappings

The mapping of logical signals onto the physical pins of a particular component package shall be defined through the use of an attribute statement and an associated BSDL string.

B.8.7.1 Syntax

<device package pin mappings>::= <pin map statement> <pin mappings>

<pin map statement>::= **attribute PIN_MAP of** <component name> **: entity is PHYSICAL_PIN_MAP;**

<pin mappings>::= <pin mapping> {<pin mapping>}
<pin mapping>::= **constant** <pin mapping name> **: PIN_MAP_STRING:=** <map string> **;**
<pin mapping name>::= <VHDL identifier>
<map string>::= **"** <port map> { **,** <port map>} **"**
<port map>::= <port name> **:** <pin list> (see B.6.3)
<pin list>::= <pin ID> | **(** <pin ID> {, <pin ID>} **)**
<pin ID>::= <VHDL identifier> | <integer>

NOTE—No subscripting of <port name> shall be allowed. Only the base name of a port shall appear if the port was described to be a **bit_vector** .

B.8.7.2 Examples

```
attribute PIN_MAP of ttl74bct8374: entity is PHYSICAL_PIN_MAP;
constant DW:PIN_MAP_STRING:=
         "CLK:1, Q:(2,3,4,5,7,8,9,10), " &
         "D:(23,22,21,20,19,17,16,15), " &
         "GND:6, VCC:18, OC_NEG:24, " &
         "TDO:11, TMS:12, TCK:13, TDI:14";
constant FK:PIN_MAP_STRING:=
         "CLK:9, Q:(10,11,12,13,16,17,18,19), " &
         "D:(6,5,4,3,2,1,26,25), " &
         "GND1:15, VCC1:8, GND2:14, VCC2:22, OC_NEG:7, " &
         "TDO:20, TMS:21, TCK:23, TDI:24";
constant DIE_BOND:PIN_MAP_STRING:=
         "CLK:Pad01, " &
         "Q:(Pad02,Pad03,Pad04,Pad05,Pad06,Pad07,Pad08,Pad09), " &
         "D:(Pad10,Pad11,Pad12,Pad13,Pad14,Pad15,Pad16,Pad17), " &
         "GND1:Pad18, VCC1:Pad19, GND2:Pad20, VCC2:Pad21, " &
         "OC_NEG:Pad22, " &
         "TDO:Pad23, TMS:Pad24, TCK:Pad25, TDI:Pad26";
```

Attribute **PIN_MAP** shall be a string that is set to the value of the parameter **PHYSICAL_PIN_MAP** , which is defined by the generic statement. VHDL constants shall then be declared, one for each packaging variation. In the example description, three packaging variations exist: `DW` , `FK` , and `DIE_BOND` .

The constants shall identify component packages that are typified for BSDL purposes by the mapping between logical port names and the physical pins of a component. A BSDL parser shall look for the constant with a name matching the value of **PIN_MAP** . The standard practice for BSDL mandates that the type of the constant shall be **PIN_MAP_STRING** .

The following is an example of a <map string>:

```
"CLK:1,Q:(2,3,4,5,7,8,9,10), D:(23,22,21,20,19,17,16,15), " &
"GND:6, VCC:18, OC_NEG:24, TDO:11, TMS:12, TCK:13, TDI:14"
```

Notice that this is the concatenation of two smaller strings. A BSDL parser reads the contents of the string. It matches signal names, such as `CLK`, with the names in the port definition.

For a given <port map>, the <pin list> shall identify the physical pin (or set of physical pins) associated with the port called <port name>. A <pin ID> may be a number or an alphanumeric identifier because some component packages, such as Pin-Grid Arrays (PGAs), use coordinate identifiers, such as A07 or H13. Note, however, that names such as 7A and 13H are illegal since they are not valid VHDL identifiers.

If signals such as that having <port name> `Q` in the example are identified as **bit_vector** in the <logical port description>, there must be a one-to-one mapping between the members of that **bit_vector** and the members of the <pin list> associated with `Q`.

The ordering of items in the <pin list> shall be significant, and the ordering shall provide correlation between a given <subscripted port name> and its associated <pin ID>.

For example, if `Q` were defined to have members `"1 to 8"`, the physical pin mapped onto port `Q(1)` in the example (for the `DW` component package) would be pin `2`, and `Q(2)` would be pin `3`, etc. If `Q` were defined to have members `"8 downto 1"`, `Q(1)` would be pin `10` and `Q(2)` would be pin `9`, etc.

Of the three mappings shown in the example, the first two are for packaging variations of a finished IC and the third shows a die-bond mapping for a finished bare die that might be used in a multichip module.

B.8.7.3 Semantic checks

a) Within a given <pin mapping>, each <pin ID> shall appear only once.
b) All nonlinkage ports in the <logical port description> of a given BSDL description shall be referenced in each <pin mapping> of that description, and vice versa.
c) For a given <port map>, the number of members in the <pin list> shall be the same as the number of bits of the **bit_vector** of the port or shall be 1 if the relative <port dimension> is **bit**.
d) The <port map> for a <port name> defined as **bit** in the <logical port description> shall be defined using a single <pin ID> without a subscript. The <port map> for a <port name> defined as a **bit_vector** in the <logical port description> shall be defined using one or more <pin ID> elements within parentheses.
e) Each <pin mapping name> shall be unique within a <pin mappings>.
f) Any <port name> element in a <port mapping> element of a given BSDL description shall appear in an <identifier list> element of the <logical port description> statement of the description.

Semantic check B.8.7.3 b) requires separate BSDL descriptions for components that do not have pin connections for some nonlinkage signals. [Concomitant differences in the boundary-scan register description (see B.8.14) will also result from these differences.]

There may be different numbers of linkage ports among packaging variants for a component. In the given example, the `FK` package had two more power/ground pins than the `DW` package. All ports `GND`, `GND2`, `VCC`, and `VCC2` must have appeared in the port definition. It is strongly recommended that all physical linkage pins should be included in a <pin mapping>.

B.8.8 Grouped port identification

The optional <grouped port identification> shall be used to identify system I/O signals that have the special characteristic of using more than one pin to carry a bit of data and for which the allowed states of the pins are restricted. Typically, these are differential pairs of signals operating in either the voltage domain or the current domain. The syntax shown below allows for future expansion if it is later determined that the description of signal groups containing more than two signals is important.

In the case of differential pairs where one signal is always the complement of the other (a state restriction), there is a “Plus” pin and a “Minus” pin. This standard states that a differential pair may need to be treated as an analog circuit with a single boundary-scan register cell providing or receiving data (see Figure 11-7 and Figure 11-9). However, due to the prevalence of differential signaling and the fact that digital data is indeed being transmitted, it is desirable to accomplish boundary-scan interconnect testing on each pin of a differential signal pair. The grouped port identification shall be used to describe such a situation to test generation software.

Rules 11.5.1 i) and 11.6.1 n) mandate that the data provided by or captured by a boundary-scan register cell shall have the same polarity as the data bit transmitted by the associated I/O pin. Clearly, the “Minus” signal cannot do this since it always transmits the complement of the “Plus” signal. The “Plus” signal can be identified to software so that it can be directly associated with the boundary-scan register. Then the “Minus” signal is linked to the “Plus” signal to indicate that a pairing exists and that the “Minus” signal is *not* associated with a cell in the boundary-scan register.

Ports shall have a <grouped port identification> when the state restrictions apply during boundary-scan operation, e.g., during *EXTEST*. If the ports are restricted during normal system operation but *not* during boundary-scan operation, the <grouped port identification> will lead software to produce incomplete results, e.g., an inadequate board interconnect test.

B.8.8.1 Syntax

<grouped port identification>::=
attribute PORT_GROUPING of <component name> **: entity is** <group table string> **;**

<group table string>::= **"** <group table> **"**
<group table>::= <twin group entry> { **,** <twin group entry>}
<twin group entry>::= <twin group type> **(** <twin group list> **)**
<twin group type>::= **DIFFERENTIAL_VOLTAGE** | **DIFFERENTIAL_CURRENT**
<twin group list>::= <twin group> { **,** <twin group>}
<twin group>::= **(** <representative port>, <associated port> **)**
<representative port>::= <port ID> (see B.6.3)
<associated port>::= <port ID>

The <representative port> pin shall be treated as the positive (“Plus”) pin of a differential pair. The <associated port> shall be the negative (“Minus”) pin of a differential pair.

Software must determine how to handle **DIFFERENTIAL_CURRENT** signals that are directly accessible to tester resources. The **DIFFERENTIAL_VOLTAGE** signals typically would be physically similar to other logic signals being tested. These two keywords help to inform software as to the physical nature of the way data is transmitted.

B.8.8.2 Examples

This example includes a <boundary-scan register description> (see B.8.14) for purposes of illustrating certain semantic relationships.

```
entity diff is
generic(Physical_Pin_Map:string:= "Pack");

port (CLK:in bit;
        D_Pos:in bit_vector(1 to 4);
        D_Neg:in bit_vector(1 to 4);
        Q_Pos:out bit_vector(1 to 4);
```

```
          Q_Neg:out bit_vector(1 to 4);
          GND, VCC:linkage bit; OC_NEG:in bit;
          TDO:out bit; TMS, TDI, TCK:in bit);

-- Get IEEE Std 1149.1-2001 attributes/definitions

use STD_1149_1_2001.all;

attribute COMPONENT_CONFORMANCE of diff : entity is
                  "STD_1149_1_2001";

attribute PIN_MAP of diff : entity is PHYSICAL_PIN_MAP;

constant PACK:PIN_MAP_STRING:="CLK:1, " &
                  "Q_Pos:(2,3,4,5), " &
                  "Q_Neg:(7,8,9,10), " &
                  "D_Pos:(23,22,21,20), " &
                  "D_Neg:(19,17,16,15), " &
                  "GND:6, VCC:18, OC_NEG:24, " &
                  "TDO:11, TMS:12, TCK:13, TDI:14";

attribute PORT_GROUPING of diff : entity is
"Differential_Voltage ((Q_Pos(1),Q_Neg(1)),"& -- Voltage signals
                                  "(Q_Pos(2), Q_Neg(2)),"&
                                  "(Q_Pos(3), Q_Neg(3)),"&
                                  "(Q_Pos(4), Q_Neg(4))),"&
"Differential_Current ((D_Pos(1),D_Neg(1)),"& -- Current signals
                                  "(D_Pos(2), D_Neg(2)),"&
                                  "(D_Pos(3), D_Neg(3)),"&
                                  "(D_Pos(4), D_Neg(4)))";
```

(... some BSDL deleted for brevity ...)

```
attribute BOUNDARY_REGISTER of diff : entity is
-- num cell         port          function safe [ccell disval rslt]
     "9 (BC_1, CLK,                 input,        X)," &
     "8 (BC_1, OC_NEG,              input,        X)," & -- Merged input/control
     "8 (BC_1, *,                   control, 1)," & -- Merged input/control
     "7 (BC_1, D_Pos(1), input,                   X)," &
     "6 (BC_1, D_Pos(2), input,                   X)," &
     "5 (BC_1, D_Pos(3), input,                   X)," &
     "4 (BC_1, D_Pos(4), input,                   X)," &
     "3 (BC_1, Q_Pos(1), output3, X,                      8, 1, Z),"&-- Also bussable
     "2 (BC_1, Q_Pos(2), output3, X,                      8, 1, Z),"&
     "1 (BC_1, Q_Pos(3), output3, X,                      8, 1, Z),"&
     "0 (BC_1, Q_Pos(4), output3, X,                      8, 1, Z)";

end diff;
```

Software processing this BSDL description example is able to identify four pairs of voltage-differential pins and four pairs of current-differential pins. Each differential signal shall be separated into positive and negative pins, with the positive pins associated with cells in the boundary-scan register.

B.8.8.3 Semantic checks

a) Any <port ID> used in a <grouped port identification> shall have been declared in the <logical port description> statement and name a signal of a nonlinkage type.

b) If a <port ID> is a <subscripted port name>, the <subscript> (see B.6.3) shall lie within the range specified for the **bit_vector** of the relevant port (see B.8.3).

c) Any <port ID> appearing as a <representative port> shall also appear as a <port ID> in a <cell spec> in the subsequent <boundary-scan register description> (see B.8.14).

d) Any <port ID> appearing as an <associated port> *shall not* appear as a <port ID> in a <cell spec> in the subsequent <boundary-scan register description> [this is an exception to semantic check B.8.14.4 s)].

e) The two ports listed in a <twin group> shall have the same pin type (see Table B.2).

f) Any <port ID> appearing in a <twin group> shall be a <subscripted port name> or a <port name> defined in a <logical port description> statement with a corresponding <port dimension> equal to **bit** .

B.8.9 Scan port identification

The scan port identification statements shall define the TAP of the component.

B.8.9.1 Syntax

<scan port identification>::=
ý <TCK stmt> <TDI stmt> <TMS stmt> <TDO stmt> [<TRST stmt>] í

<TCK stmt>::= **attribute TAP_SCAN_CLOCK of** <port ID> **: signal is (** <clock record> **);**
<TDI stmt>::= **attribute TAP_SCAN_IN of** <port ID> **: signal is true;**
<TMS stmt>::= **attribute TAP_SCAN_MODE of** <port ID> **: signal is true;**
<TDO stmt>::= **attribute TAP_SCAN_OUT of** <port ID> **: signal is true;**
<TRST stmt>::= **attribute TAP_SCAN_RESET of** <port ID> **: signal is true;**

<clock record>::= <real number> **,** <halt state value>
<halt state value>::= **LOW** | **BOTH**

The statements shall identify specific logical signals of the component as being signals of the TAP.

A <clock record> shall be a pair consisting of

— A real number that gives the maximum operating frequency for TCK in hertz. In the example of B.8.9.2, 20.0 MHz is specified as the maximum operating frequency.

— A VHDL type that shall have one of two values— **LOW** and **BOTH** . This shall specify the state(s) in which the TCK signal may be stopped without causing loss of data held in the test logic. **BOTH** shall indicate that the clock can be stopped in either state. Components that allow TCK to be stopped only in the high state do not conform to this standard.

B.8.9.2 Examples

```
attribute TAP_SCAN_IN of TDI : signal is true;
attribute TAP_SCAN_OUT of TDO : signal is true;
attribute TAP_SCAN_MODE of TMS : signal is true;
attribute TAP_SCAN_RESET of TRST : signal is true;
attribute TAP_SCAN_CLOCK of TCK : signal is (20.0e6, BOTH);
```

Signal names `TDI`, `TDO`, `TMS`, `TRST`, and `TCK` are those that were used in the <logical port description> in the above example (see B.8.8.2). The names used in the above examples are those defined in this standard; however, arbitrary names could have been used.

NOTE—The meaning of each signal in relation to this standard shall be deduced from the attribute name, not from the value of a <port ID> element in the <scan port identification>.

B.8.9.3 Semantic checks

a) With respect to the <port ID> elements in the <scan port identification>,
   1) The unique <port name> element in a <port ID> occurring in a <TDO stmt> shall appear as a value in an <identifier list> of a <pin spec> (see B.8.3) in the <logical port description>. Furthermore, the value of <pin type> on that <pin spec> shall be **out**.
   2) The unique <port name> element in a <port ID> occurring in a <TCK stmt>, a <TDI stmt>, a <TMS stmt>, or a <TRST stmt> shall appear as a value in an <identifier list> of a <pin spec> (see B.8.3) in the <logical port description>. Furthermore, the value of <pin type> on that <pin spec> shall be **in**.
   3) Any <port ID> appearing in the <scan port identification> shall be a <subscripted port name> or a <port name> defined in a <logical port description> statement with a corresponding <port dimension> equal to **bit**.
b) No <port ID> in the <scan port identification> shall later appear in the <boundary-scan register description> [this is an exception to semantic check B.8.14.4 s)].
c) A given <port ID> shall occur at most once in the <scan port identification>.
d) If a <port ID> is a <subscripted port name>, the <subscript> (see B.6.3) shall lie within the range specified for the **bit_vector** of the relevant port.
e) No value of a <port ID> element in the <scan port identification> shall have appeared as a <representative port> or an <associated port> in a <twin group> (see B.8.8).

B.8.10 Compliance-enable description

This portion of a BSDL description shall appear in the description of a component or bare chip if the optional compliance-enable feature described by this standard (see 4.8) has been implemented in that component or bare chip. Otherwise, improper operation of the part may occur during an automatically generated test.

B.8.10.1 Syntax

<compliance enable description>::= **attribute COMPLIANCE_PATTERNS of**
<component name> **: entity is** <compliance pattern string> **;**

<compliance pattern string>::= **" (** <compliance port list> **) (** <pattern list> **)"**
<compliance port list>::= <port ID> { **,** <port ID>} (see B.6.3)
<pattern list>::= <pattern> { **,** <pattern> }

NOTE—A number of <pattern> elements may be specified, reflecting the fact that there may be many combinations of bits that enable compliance. For convenience, a <pattern> may contain X bits to reduce the size of a <pattern list>. As an example, the <pattern list> 1111X, 0XXX0 specifies 10 unique bit patterns that enable compliance.

A semantic relationship shall positionally associate <port ID> elements in a <compliance port list> with the bits in a <pattern>. For example, the first appearing <port ID> is associated with the first (leftmost) bit in <pattern>.

The following example is given for a component that implements an LSSD test mode as well as standard boundary scan, as described in Annex A [see A.1 c)]. When the LSSD test mode is used, the test logic

defined by this standard becomes configured as a part of a scan path such that LSSD techniques can be used to verify the operation of the complete component. Thus, the LSSD control pins are compliance-enable pins.

B.8.10.2 Examples

```
attribute COMPLIANCE_PATTERNS of Annex_A_IC: entity is
        "(LSSD_A, LSSD_B, LSSD_P, LSSD_C1, LSSD_C2) (00011)";
```

Software that generates tests using test facilities defined by this standard must assure that any compliance-enable conditions are first set up before exercising the TAP of the affected IC. Any compliance-enable pattern (provided as a <pattern> within the <pattern list>) must be held constant for the duration of all boundary-scan testing.

B.8.10.3 Semantic checks

a) The number of <port ID> elements in the <compliance port list> shall be equal to the number of bits in each <pattern> within the <pattern list>.

b) No <port ID> value shall occur more than once in the <compliance port list>.

c) No <port ID> value in the <compliance port list> shall also appear as a <port ID> value in the <scan port identification> (see B.8.9).

d) Each <port ID> value in the <compliance port list> shall appear as a value in an <identifier list> of a <pin spec> (see B.8.3) in the <logical port description>. Furthermore, the value of <pin type> in that <pin spec> shall be **in**.

e) No <port ID> value in the <compliance port list> shall later appear in the <boundary-scan register description> [this is an exception to semantic check B.8.14.4 s)].

f) If a <port ID> value in the <compliance enable description> is a <subscripted port name>, the <subscript> (see B.6.3) shall lie within the range specified for the **bit_vector** of the relevant port.

g) No <port ID> value in the <compliance port list> shall appear as a <port ID> in the <grouped port identification> (see B.8.8).

B.8.11 Instruction register description

The next segment of the BSDL description concerns the component-dependent characteristics of the instruction register. The details that shall be specified to characterize the implementation of the instruction register in a particular component are

— *Length*. The instruction register will be at least two bits long. Its length is not otherwise limited.

— *Instructions*. The register is required to support certain instructions. A designer may add any or all of the optional instructions defined by this standard and/or any number of design-specific instructions. Also, this standard provides for private instructions, which shall be marked as such to warn applications not to use them in order to prevent unsafe or undocumented behavior.

— *Instruction binary codes (opcodes)*. The *BYPASS* instruction is decoded from a bit pattern fixed by this standard [see Rule 8.4.1 b)]. Bit patterns for other instructions shall be specified by the test logic designer. Each instruction may be decoded from several bit patterns.

— *Instruction capture*. On passing through the *Capture-IR* controller state, the instruction register will load data from its parallel input. In some register stages, certain fixed values are required, but in other register stages, design-dependent values may be loaded.

BSDL shall provide a means of describing these characteristics and take advantage of opportunities for semantic checks, thus verifying that the component is in compliance with this standard (that is, it has implemented the required instruction binary codes properly).

The characteristics of the instruction register that shall be specified using BSDL are its length, the opcodes, the pattern captured in the *Capture-IR* controller state, and whether any given instruction is public or private.

B.8.11.1 Syntax

<instruction register description>::= <instruction length stmt>
<instruction opcode stmt>
<instruction capture stmt>
[<instruction private stmt>]

<instruction length stmt>::= **attribute INSTRUCTION_LENGTH of** <component name>
: entity is <integer> **;**
<instruction opcode stmt>::= **attribute INSTRUCTION_OPCODE of** <component name>
: entity is <opcode table string> **;**
<instruction capture stmt>::= **attribute INSTRUCTION_CAPTURE of** <component name>
: entity is <pattern list string> **;**
<instruction private stmt>::= **attribute INSTRUCTION_PRIVATE of** <component name>
: entity is <instruction list string> **;**

<opcode table string>::= **"** <opcode description> {, <opcode description>} **"**
<opcode description>::=<instruction name> (<pattern list>)
<pattern list>::= <pattern> { **,**<pattern>}
<pattern list string>::= **"** <pattern list> **"**
<instruction list string>::= **"** <instruction list> **"**
<instruction list>::= <instruction name> { **,**<instruction name>} (see B.6.3)

B.8.11.2 Examples

```
attribute INSTRUCTION_LENGTH of My_IC:-- Must be first
                                                entity is 4;
attribute INSTRUCTION_OPCODE of My_IC:-- Must be second
                                                entity is
         "EXTEST (0011), " &
         "EXTEST (1011), " &
         "BYPASS (1111), " &
         "SAMPLE (0001, 1000), " &
         "PRELOAD(1001, 1000)," &
         "HIGHZ       (0101), " &
         "SECRET (1010) ";
attribute INSTRUCTION_CAPTURE of My_IC:-- Must be third
                                                entity is "0001";
attribute INSTRUCTION_PRIVATE of My_IC:-- Optional
                                                entity is "Secret";
```

NOTES

1—In the above example, *BYPASS* was shown to be decoded from “1111”. Because this is the mandatory pattern specified by this standard, its expression is redundant and not required. In addition to any explicitly assigned patterns, all unassigned patterns will also be decoded as *BYPASS*.

2—For devices designed to conform to editions of this standard before IEEE Std 1149.1-2001, *EXTEST* had a mandatory pattern of all zeros (for example, “0000”), and so, where the value of <conformance identification> in a given BSDL description is **STD_1149_1_1990** or **STD_1149_1_1993**, the expression of this mandatory decode is redundant and not required. For devices designed to conform to this edition of the standard, *EXTEST* has no mandatory bit pattern and

so, where the value of <conformance identification> in a given BSDL description is **STD_1149_1_2001**, **EXTEST** and its bit pattern must occur in the <instruction opcode stmt>.

3—Also, notice that **EXTEST** is given on two lines with two decodes. This shows that multiple lines may be used for each instruction if needed (for instance, when this attribute is written by a computer program).

4—As would be required in the case that the <conformance_identification> were **STD_1149_1_2001**, the above example illustrates the specification of a pattern for **PRELOAD**. It also illustrates options for *SAMPLE* and *PRELOAD* instructions both where the same pattern is used for both and where unique patterns are used for each. It should be further noted that where a pattern for *SAMPLE* is the same as a pattern for PRELOAD, all rules for both instructions must be met. This combination is equivalent to the *SAMPLE/PRELOAD* instruction in previous editions of this standard.

The purpose of each attribute shall be as follows:

— **INSTRUCTION_LENGTH**. The <instruction length stmt> shall define the length of the instruction register and, hence, the number of bits that each opcode pattern shall contain in subsequent statements of the <instruction register description>.

— **INSTRUCTION_OPCODE**. The <instruction opcode stmt> shall be a BSDL string containing instruction identifiers and their associated bit patterns. The rightmost bit in the pattern shall be that closest to TDO (i.e., that shifted in first). Each <opcode description> shall be such a pair. This standard mandates the existence of *BYPASS*, *SAMPLE*, *PRELOAD*, and *EXTEST* instructions, with a mandatory bit pattern for *BYPASS*. Decoding of unused bit patterns shall default to the *BYPASS* instruction.

— **INSTRUCTION_CAPTURE**. The <instruction capture stmt> shall specify the bit pattern that is loaded into the instruction register when the TAP controller passes through the *Capture-IR* state. This bit pattern shall be shifted out whenever a new instruction is shifted in. This standard mandates that the two least significant bits must be “01”. The remainder of this bit pattern is design-specific.

— **INSTRUCTION_PRIVATE**. The optional <instruction private stmt> shall identify instructions that are private and potentially unsafe for use by other than the manufacturer of the component. By definition, the effects of these instructions are undefined to the general public; their use should be avoided. Note that failure to follow warnings about private instructions can result in damage to the component, circuit board, or system.

B.8.11.3 Semantic checks

a) The integer value of the attribute **INSTRUCTION_LENGTH** shall be greater than or equal to 2. This value shall be interpreted as the length of the instruction register.

b) All opcodes defined within a <pattern list> shall have length equal to that of the instruction register.

c) The opcode made up of all 1s shall decode to *BYPASS* or not be defined explicitly for any instruction.

d) Where the value of <conformance identification> is **STD_1149_1_2001**, an opcode for *EXTEST* shall be defined, and the <opcode description> in which it is defined shall have **EXTEST** as the value of its <instruction name> element.

NOTE—Where a device conforms to this 2001 edition of this standard, as indicated by a <conformance identification> value of **STD_1149_1_2001**, the all 0s opcode is not mandated for *EXTEST*. Therefore, an opcode bit pattern for *EXTEST* is required to be explicitly defined. Further, if the all 0s opcode is not otherwise assigned, it decodes to *BYPASS*.

e) Where the value of <conformance identification> is **STD_1149_1_1990** or **STD_1149_1_1993**, the opcode made up of all 0s shall decode to *EXTEST* or shall not be defined explicitly for any instruction.

NOTE—Where a device conforms to earlier editions of this standard, as indicated by a <conformance identification> value of **STD_1149_1_1990** or **STD_1149_1_1993**, the all 0s opcode is mandated for *EXTEST* and so shall be defined as such or be not explicitly defined (in which case it is presumed by implication).

f) Where the value of <conformance identification> is **STD_1149_1_2001**, an opcode for *SAMPLE* shall be defined, and the <opcode description> in which it is defined shall have **SAMPLE** as the value of its <instruction name> element.

g) Where the value of <conformance identification> is **STD_1149_1_2001**, an opcode for *PRELOAD* shall be defined, and the <opcode description> in which it is defined shall have **PRELOAD** as the value of its <instruction name> element.

h) Where the value of <conformance identification> is **STD_1149_1_1990** or **STD_1149_1_1993**, an opcode for *SAMPLE/PRELOAD* shall be defined as follows:

1) At least one <opcode description> shall have **SAMPLE** as the value of its <instruction name> element.

2) Any <opcode description> that has **PRELOAD** as the value of its <instruction name> element shall contain only such values for <pattern> as have been defined for any <opcode description> that fulfills the requirement for B.8.11.3 h) 1).

NOTE—Where a device conforms to earlier editions of this standard, as indicated by a <conformance identification> value of **STD_1149_1_1990** or **STD_1149_1_1993**, the functions of *SAMPLE* and *PRELOAD* were required to be implemented in a merged fashion, *SAMPLE/PRELOAD*. Therefore, every opcode which is defined for *SAMPLE* is implicitly defined for *PRELOAD*. Further, thus, where an opcode is defined for *PRELOAD*, it must have a binary value that matches one defined for *SAMPLE*.

i) Where Permission 8.1.1 g) is met, the length of the selected registers shall be identical for each of the instructions that share the same opcode.

j) Where Permission 8.1.1 g) is not met, opcodes containing **X** bits shall not be ambiguous (i.e., decodable as two or more different instructions). That is, if there are two <opcode description> elements and two <pattern> elements such that an **X** appears in each of the two <opcode description> elements, the two <pattern> elements shall differ in some character position in which *neither* pattern contains the character **X**.

k) The <pattern> value in the <instruction capture stmt> shall have a length equal to that of the instruction register. The two least significant bits of this <pattern> shall be **01**.

l) Only user-defined instructions shall be defined as private.

m) Any <instruction name> appearing in an <instruction list> shall appear only once and also shall appear in the <opcode table>.

B.8.12 Optional register description

This section shall identify whether the optional device identification register is provided by specifying the bit patterns returned in response to selection of the device identification register instructions—*IDCODE* and, if implemented, *USERCODE*.

B.8.12.1 Syntax

<optional register description>::= ý <idcode statement> [<usercode statement>] í

<idcode statement>::= **attribute IDCODE_REGISTER of** <component name>
: entity is <32-bit pattern list string> **;**

<usercode statement>::= attribute USERCODE_REGISTER of <component name>
: entity is <32-bit pattern list string> **;**

<32-bit pattern list string>::= **"** <32-bit pattern list> **"**
<32-bit pattern list>::= <32-bit pattern> {, <32-bit pattern>} (see B.6.2)

B.8.12.2 Examples

```
attribute IDCODE_REGISTER of My_IC: entity is
        "0011" &                            -- Version
        "11110000111110000" &               -- Part number
        "00001010100" &                     -- Identity of the manufacturer
        "1";                                -- Required by IEEE Std 1149.1-1990

attribute USERCODE_REGISTER of My_IC: entity is
   "10XX" & "0011" & "1100" & "1111" & -- Start 1st 32-bit pattern
   "0000" & "0000" & "0000" & "1111,"& -- End 1st 32-bit pattern
   "111X" & "0011" & "1001" & "1000" & -- Start 2nd 32-bit pattern
   "0000" & "0100" & "1001" & "1000";                    -- End 2nd 32-bit pattern
```

In these examples, concatenation is used to delimit fields within the codes.

An “ **X** ” shall be used to mask subfields within a code that are not important for testing purposes; “ **X** ” specifies a “don't care.” It is also possible that a single component type may be sourced from different manufacturers with different version fields. The <32-bit pattern list> shall allow all desired sourcing to be fully specified without using “ **X** ” bits.

B.8.12.3 Semantic checks

a) If a device identification register is specified by the inclusion of an <idcode statement>, the least significant bit in any <32-bit pattern> element within the <idcode statement> shall be **1** .

b) An <idcode statement> shall appear in a BSDL description if and only if **IDCODE** appears as the value of an <instruction name> element in an <opcode description> of the <instruction opcode stmt>.

c) A <usercode statement> shall appear in a BSDL description if and only if both **IDCODE** and **USERCODE** appear as the value of <instruction name> elements in an <opcode description> of the <instruction opcode stmt>.

d) An illegal bit pattern in the manufacturer code shall cause an error [see Rule 12.2.1 b)].

B.8.13 Register access description

All instructions shall place a test data register between TDI and TDO. User-defined instructions may access test data registers mandated by this standard or design-specific registers. This standard allows a designer to place additional test data registers, referenced by user-defined instructions, in the component.

It is important for test development software to know of the existence and length of all test data registers and the names of their associated instructions.

B.8.13.1 Syntax

<register access description>::=
attribute REGISTER_ACCESS of <component name> **: entity is**
<register string> **;**

<register string>::= **"** <register association> { **,** <register association>} **"**
<register association>::= <register> **(** <instruction capture list> **)**
<instruction capture list>::= <instruction capture> {, <instruction capture>}
<instruction capture>::= <instruction name> [**CAPTURES** <pattern>](see B.6.3)
<register>::= **BOUNDARY** | **BYPASS** | **DEVICE_ID** |

<VHDL identifier> <left bracket> <register length> <right bracket>

<register length>::= <integer>

B.8.13.2 Examples

```
attribute REGISTER_ACCESS of ttl74bct8374: entity is
        "BOUNDARY (READBN, READBT, CELLTST), " &
        "BYPASS (TOPHIP, SETBYP, RUNT, TRIBYP), " &
        "BCR[2] (SCANCN, SCANCT CAPTURES 0X)";
```

In this example, `READBN`, `READBT`, `CELLTST`, `TOPHIP`, `SETBYP`, `RUNT`, `TRIBYP`, `SCANCN`, and `SCANCT` must have been defined in the <instruction opcode stmt> (see B.8.11). The first three instructions select the boundary-scan register, while the next four instructions select the bypass register. Note that the last two instructions (`SCANCN` and `SCANCT`) select a two-bit register called `BCR[2]`. This is a design-specific test data register. The `SCANCT` instruction also shows a capture value `0X` that will be loaded into `BCR[2]` when passing through the *Capture-DR* controller state. No capture value is specified for the `SCANCN` instruction.

By identifying the association between instructions and test data registers, software that can read BSDL descriptions can determine the length of the test data scan sequence for a given instruction. The lengths of the mandatory boundary-scan register, bypass register, and instruction register, as well as the optional identification register, are known from other statements as well as from their relationship to the instructions for the component. A semantic check shall be made to ensure that each instruction has an associated test data register as required by this standard. Exceptions to this are private instructions (see B.8.11). They are not to be accessed, nor do their target registers need to be identified. Because these instructions should not be used by component users, their associated test data registers need not be defined in the BSDL description. (However, they may be defined if the user wishes.)

Note that this standard allows user instructions to reference several registers at once in a concatenated mode [see Permission 9.2.1 e)]. In BSDL, the logical register resulting from such a concatenation shall be treated as if it were a separate register with a distinct name and length.

B.8.13.3 Semantic checks

a) The association of registers with the *BYPASS*, *CLAMP*, *EXTEST*, *HIGHZ*, *IDCODE*, *INTEST*, *PRELOAD*, *SAMPLE*, and *USERCODE* instructions is mandated by this standard. Descriptions of these assignments are redundant and not needed in BSDL, but if they are given,
   1) They shall be checked against the mandatory assignment specified in this standard and an error issued if they are not correct; and
   2) They shall not have a “**CAPTURES** <pattern>” element in their description. (This capture data is specified either in this standard or elsewhere in BSDL.)

b) Any public instruction (i.e., an instruction whose name appears in the <instruction opcode stmt> but does not appear in the <instruction private stmt>, see B.8.11) not listed in semantic check B.8.13.3 a) shall have an associated test data register defined.

c) The length of each publicly accessible design-specific test data register shall be specified and shall be greater than 0; if the <register> appears in more than one <register association>, the first appearance shall define the length, and subsequent appearances shall either not define it again or define it identically.

d) All instructions, the names of which appear as the value of an <instruction name> element in any <instruction capture> element of the <register string>, shall appear as an <instruction name> element in an <opcode description> element in the <instruction opcode stmt> (see B.8.11).

e) Any <instruction name> element shall appear in only one <instruction capture list> within the <register string>.

f) The <pattern> value in an <instruction capture> element shall contain the same number of bits as the <register> in the same <register association>.

B.8.14 Boundary-scan register description

The <boundary-scan register description> shall contain a list of boundary-scan register cells numbered 0 to LENGTH-1 (where the total number of cells in the register is LENGTH). The cells may be listed in any order, but all shall be defined. Cell 0 shall be that closest to TDO. The boundary-scan register cells can vary in design and purpose. Clause 11 shows many example cell designs, but many others are possible under the rules of this standard.

The characteristics of each cell design used in a component shall be specified before the cells can be referenced in the <boundary-scan register description>. For the example cell designs included in this standard, cell descriptions shall be contained in the Standard VHDL Package **STD_1149_1_2001** (see Clause B.9). Cells defined in this VHDL package shall be referenced through the simple set of names listed in Table B.3.

Table B.3—List of cells defined in the Standard VHDL Package and relevant figure numbers

Name	Figures [a]	Comments
BC_0	Special cell	Degenerate form [b]
BC_1	11-18, 11-30, 11-34c, 11-34d, 11-36c, 11-46d	Design usable for many functions
BC_2	11-14, 11-31, 11-35c, 11-35d, 11-37c, 11-38c, 11-39(output), 11-41c	*INTEST* unsupported on Output2
BC_3	11-15	Input or Internal only
BC_4	11-16, 11-17, 11-39(input)	Input, Observe-Only, Clock or Internal only
BC_5	11-46c	Combined Input/Control
BC_6	11-38d	Bidirectional, deprecated [c]
BC_7	11-37d	Bidirectional
BC_8	11-40, 11-41d	Simpler bidirectional, lacking *INTEST* support; observes the signal at the corresponding pin even while operating in output mode
BC_9	11-32	Output that observes the signal at the corresponding pin for *EXTEST* and observes the signal driven from the system logic for *INTEST* and *SAMPLE*
BC_10	11-33	Simpler output, lacking *INTEST* support; always monitors the signal at the corresponding pin instead of the signal driven from the system logic

[a] The suffix “c” is used to denote a control cell shown in a cited figure. The suffix “d” denotes a data cell.

[b] **BC_0** is a cell that captures the value specified by the rules of this standard and that captures a don't-care value whenever this standard allows options. It can be used whenever there is uncertainty about the exact behavior of a compliant cell.

[c] **BC_6** meets the requirements of this standard, but it lacks desirable features and is not recommended for use in new designs. The design for **BC_7** (see Figure 11-37d) is preferred.

The method of describing cells other than those depicted in this standard is described in B.10.2. When such cell designs are to be used, their descriptions shall be given in a user-supplied VHDL package and VHDL package body.

Several rules shall be observed when combining cells to create a boundary-scan register conformant to this standard. Adherence to some rules shall be checked during processing of the BSDL description of a component. For example, some cell designs may be used only on a component input. Some will not support the *INTEST* instruction—this is allowable if **INTEST** does not appear in the <instruction opcode stmt> (see B.8.11). Some cells require the aid of another to control 3-state enables or the direction of signal flow.

A very general cell design from this standard (see Figure 11-18) is shown in Figure B.2. Figure B.3 shows a symbolic representation of the same cell design.

Figure B.2—Cell design corresponding to Figure 11-18

The design in Figure B.2 consists of a parallel input (PI), a parallel output (PO), a multiplexer controlled by a mode signal, and two flip-flops. The mode signal is a function of the current instruction. A serial input (SI) and a serial output (SO) form the shift path through the cell. The mode signal is a logic 0 or 1 that tells a cell what test function to perform (see Table 11-3). Note that the symbolic representation does not include

— The multiplexer controlled by signal ShiftDR;
— The mode signal and multiplexer; and
— The clock signals, ClockDR, and UpdateDR.

These parts of the cell design do not need to be considered in BSDL because the control and operation of boundary-scan register cells are fully defined by this standard. Thus, Figure B.3 is a full representation of the cell design shown in Figure B.2 for the purposes of BSDL. The parallel input and output are shown in the figure, and they are connected to various places, depending on the application. The two flip-flops are labeled CAP and UPD to represent their uses. The CAP flip-flop captures data from the system data input of the cell in the *Capture-DR* controller state and lies on the shift register path. The UPD flip-flop loads data from the CAP flip-flop in the *Update-DR* controller state. The shift path is shown because many such cells will be linked together in a shift chain that makes up the boundary-scan register. The shift path links only the CAP flip-flops.

Figure B.3—A symbolic representation of a boundary-scan register cell

One cell design, shown in Figure 11-17 in this standard, is an observe-only cell and has a symbol with no UPD flip-flop (Figure B.4). This cell can be used at a system input pin and has the advantage of lower propagation delay in some implementations. However, it does not support the optional *INTEST* instruction if provided at a nonclock input.

Figure B.4—A symbolic representation of a boundary-scan register cell without an update stage

The symbols in Figure B.5 and Figure B.6 show a bubble on top of the UPD flip-flop (indicating that the flip-flop may be preset to 1) or a bubble on the bottom (indicating that the flip-flop may be cleared to 0) when the *Test-Logic-Reset* controller state is entered. No signal connection is made to these bubbles.

NOTE—The resetting or clearing of such cells shall be consistent with Rule 11.6.1 p), which states that the value forced into the cell shall be the one that would disable the associated drivers.

Figure B.5—A symbol for a boundary-scan register cell with preset

Figure B.6—A symbol for a boundary-scan register cell with clear

B.8.14.1 Syntax

<boundary-scan register description>::= <boundary length stmt> <boundary register stmt>

<boundary length stmt>::=
attribute BOUNDARY_LENGTH of <component name> **: entity is** <integer> **;**

<boundary register stmt>::= **attribute BOUNDARY_REGISTER of** <component name>
: entity is <cell table string> **;**

<cell table string>::= **"** <cell table> **"**
<cell table>::= <cell entry> { **,** <cell entry> }
<cell entry>::= <cell number> **(** <cell info> **)**
<cell number>::= <integer>
<cell info>::= <cell spec> [**,** <disable spec>]

<cell spec>::= <cell name> , <port ID or null> , <function> , <safe bit>
<cell name>::= <VHDL identifier>
<port ID or null>::= <port ID> | * (see B.6.3)
<function>::= **INPUT** | **OUTPUT2** | **OUTPUT3** | **CONTROL** | **CONTROLR** | **INTERNAL** | **CLOCK** | **BIDIR** | **OBSERVE_ONLY**
<safe bit>::= **0** | **1** | **X**
<disable spec>::= <ccell> , <disable value> , <disable result>
<ccell>::= <integer>
<disable value>::= **0** | **1**
<disable result>::= **Z** | **WEAK0** | **WEAK1** | **PULL0** | **PULL1** | **KEEPER**

B.8.14.2 Examples

```
attribute BOUNDARY_LENGTH of ttl74bct8374: entity is 18;
```

The <boundary length stmt> shall define the number (LENGTH) of cells in the boundary-scan register. This number shall match the number of <cell entry> elements in the <boundary register stmt>, which describes the structure of the boundary-scan register. Some cells may require two lines of description (see B.11.1.3).

```
attribute BOUNDARY_REGISTER of ttl74bct8374: entity is
--
--num cell         port/*      function safe [ccell disval rslt]
         --
"17     (BC_1, CLK,            input,        X)," &
"16     (BC_1, OC_NEG, input,               X)," &
"16     (BC_1, *,              control, 1)," &
. . .
. . .
"3      (BC_1, Q(5),           output3, X,          16,        1,       Z)," &
"2      (BC_1, Q(6),           output3, X,          16,        1,       Z)," &
"1      (BC_1, Q(7),           output3, X,          16,        1,       Z)," &
"0      (BC_1, Q(8),           output3, X,          16,        1,       Z)";
```

B.8.14.3 Cell entry elements

The <boundary register stmt> shall contain a string (<cell table string>) within which there is a list of elements (<cell entry>), each with two fields. The <cell entry> elements may be listed in any order, but all shall be listed.

— The <cell number> element shall be in the range from 0 to LENGTH-1.
— The <cell info> shall contain a list of either four or seven elements contained within parentheses. (In the above example, the elements are labeled— `cell` , `port/*` , `function` , `safe` , `ccell` , `disval` , and `rslt` —as indicated by the commented header.)

All <cell entry> elements shall include values for the first four elements of the second field. Only <cell entry> elements for cells that drive system outputs that can be set to an inactive drive state (e.g., open-collector or 3-state outputs) shall have the remaining three elements of the second field, which specify how the output may be disabled. If the <function> element is **OUTPUT3** or **BIDIR** , the last three elements shall be defined. If the <function> element is **BIDIR** , the action of placing the relevant driver in the inactive drive state shall be taken as equivalent to setting the cell to operate as a receiver. If the <function> element is **OUTPUT2** , the last three elements either shall or shall not be defined, depending on whether the described driver is an asymmetrical driver, e.g., open-collector (VHDL <port type> equal to out) or capable of actively driving both states (VHDL <port type> equal to **buffer**), respectively.

The <cell spec> and <disable spec> elements shall be as defined in the following subclauses.

B.8.14.3.1 The <cell name> element

This shall identify the cell design used. It shall match a cell described in the Standard VHDL Package or in a user-supplied VHDL package.

B.8.14.3.2 The <port id or null> element

This element shall identify the system input or output connected to a given cell. Any name supplied for this element shall match one specified in the <logical port description>. A cell serving as an output control or internal cell shall have an asterisk (*) supplied for this element. Either a <port name> element with the corresponding <port dimension> previously described as bit or a <subscripted port name> shall be supplied as the value of a <port ID or null> element.

B.8.14.3.3 The <function> element

This element shall define the primary function of the relevant cell. Table B.4 lists the possible values of the <function> element.

Table B.4—Function element values and meanings

Value	Meaning	Example figure in this standard [a]
INPUT	Control-and-observe cell	11-18
	Observe-only cell not supporting *INTEST*	11-16
CLOCK	Cell at a clock input	11-17
OUTPUT2	A cell that drives a 2-state (either symmetric or asymmetric) output	11-30
OUTPUT3	A cell that drives data to a 3-state output	11-34d
CONTROL	A cell that controls a 3-state enable or direction control	11-34c
CONTROLR	A control cell that is forced to its disable state in the *Test-Logic-Reset* controller state	11-36c
INTERNAL	Cell not associated with a device signal pin that captures constants “1”, “0”, or “X”	–
BIDIR	A reversible cell for a bidirectional pin	11-37d
OBSERVE_ONLY	A solitary observe-only cell on an input	11-17
	Additional cell observing any device signal pin	

[a] The suffix “c” is used to denote a control cell shown in a cited figure. The suffix “d” denotes a data cell.

Notice that many of the cell designs of this standard are somewhat general, meaning that they can be used in more than one context. For example, the <function> element in a description of the cell depicted in Figure 11-30 can have as its value **INPUT**, **OUTPUT2**, **OUTPUT3**, **CONTROL**, or **INTERNAL**. The value of the <function> element has important implications in describing a given cell (see B.10.2).

An **INPUT** function shall indicate that the cell has observe capability (control-and-observe capability if *INTEST* is implemented in the device) and is connected to a system pin. This pin shall have a <pin type> value of **in** or **inout** only.

A **CLOCK** function shall indicate an observe-only cell that is connected to a system input clock pin that allows support of *INTEST* [see Rule 11.5.1 g) 1)].

An **OUTPUT2** function shall indicate that the cell (which must have control-and-observe capability) provides data for a 2-state (symmetric or asymmetric) driver and is connected to a system pin. This pin shall have a <pin type> value of **out**, **buffer**, or **inout**.

An **OUTPUT3** function shall indicate that the cell (which must have control-and-observe capability) provides data for a 3-state driver and is connected to a system pin. This pin shall have a <pin type> value of **out** or **inout**.

A **CONTROL** function shall indicate that the cell (which must have control-and-observe capability) provides output enable control and/or direction control to one or more output drivers or bidirectional pins. A **CONTROL** cell shall not be referenced to a system pin in the <cell spec> element; see B.11.1.3 for detail on system input pins that are used to control system output drivers.

A **CONTROLR** function shall be identical to a **CONTROL** function with the exception that it also has the capability to be reset or cleared when the TAP passes through the *Test-Logic-Reset* state (see Figure 11-36).

A **BIDIR** function shall indicate a control-and-observe cell that is connected to a system pin with <pin type> **inout**.

An **INTERNAL** function shall indicate that the cell is either a placeholder cell that has gone unused because of the programming of user-programmable logic, or a cell that sits at the interface between digital and analog portions of the core circuitry of a component [see Rule 11.4.1 b)]. An **INTERNAL** cell shall not be connected to a system pin.

An **OBSERVE_ONLY** function shall indicate that the cell (which has observe-only capability) is either a cell like **INPUT** without *INTEST* support capability or an additional cell that can monitor any kind of nonlinkage system pin not exempted by semantic checks B.8.8.3 d) (grouped ports), B.8.9.3 b) (scan port identification), or B.8.10.3 e) (compliance-enable description).

Clause 11 effectively classifies system pins as INPUT, CLOCK, TWO-STATE OUTPUT, THREE-STATE OUTPUT, and BIDIRECTIONAL. For a system pin classification of

a) INPUT pin, a cell shall have a <function> of **INPUT** or **OBSERVE_ONLY**. Additional cells with <function> **INPUT** or **OBSERVE_ONLY** may be connected to the INPUT pin. If INTEST is a supported instruction, there shall be at least one cell with <function> of **INPUT** connected to the INPUT pin.
b) CLOCK pin, there are two cases:
   1) At least one cell shall have a <function> of **CLOCK**. Additional cells with <function> **OBSERVE_ONLY** also may be connected to the pin. This case is used where external clocking must be supplied to support *INTEST* or *RUNBIST*.
   2) At least one cell shall have a <function> of **INPUT**. Additional cells with <function> **INPUT** (see Figure 11-11) or **OBSERVE_ONLY** also may be connected to the CLOCK pin. This case is used where clocking must be supplied by shifting to support *INTEST* [see Rule 11.5.1 g) 3)].
c) TWO-STATE OUTPUT pin, one cell shall have a <function> of **OUTPUT2**. Additional cells with <function> **OBSERVE_ONLY** also may be connected to the pin.
d) THREE-STATE OUTPUT pin, one cell shall have a <function> of **OUTPUT3**. Additional cells with <function> **OBSERVE_ONLY** also may be connected to the pin.

e) BIDIRECTIONAL pin, there are two cases:

1) A single cell with <function> **BIDIR** shall be attached to the pin. Additional cells with <function> **OBSERVE_ONLY** also may be connected to the pin; or

2) A two-cell structure shall be used to create bidirectionality; one of these cells shall have a <function> of either **OUTPUT2** or **OUTPUT3**, and the other cell shall have a <function> of either **INPUT** or **OBSERVE_ONLY**. Additional cells with <function> **OBSERVE_ONLY** also may be connected to the pin.

Clause 11 classifies system logic (different from system pin) inputs and outputs as INPUT, CLOCK, OUTPUT DATA, and OUTPUT CONTROL. For a system logic classification of

a) INPUT or CLOCK, there are two cases:

1) Cell provisions shall be the same as noted for system INPUT or CLOCK pins, or

2) For system logic receiving data from analog circuitry, cells with <function> **INTERNAL** shall be connected, but they shall not be referenced to a system pin in the <cell spec> element.

b) OUTPUT DATA or OUTPUT_CONTROL, there are two cases:

1) Cell provisions shall be the same as noted for system pins above; additional cells with <function> **INTERNAL** also may be connected, but they shall not be referenced to a system pin in the <cell spec> element, or

2) For system logic providing data to analog circuitry, cells with <function> **INTERNAL** shall be connected, but they shall not be referenced to a system pin in the <cell spec> element.

Clause 11 also specifies that cells may exist that are connected neither to system pins nor to system logic due to the programming of programmable system logic (see 11.8). Such cells shall have <function> values of **INTERNAL**.

B.8.14.3.4 The <safe bit> element

This element shall supply a value that should be loaded into the CAP flip-flop (and UPD flip-flop if it exists) of a given cell when board-level test generation software might otherwise choose a value randomly.

The <safe bit> value is not intended to force software to use particular values for cells. Rather, it provides values for cells where software would otherwise choose a 0 or 1 at random. An "**X**" shall signify that the value does not matter and that test generation software may assign either a 1 or a 0 in a case where there is no value that the algorithm requires.

For control cells, the <safe bit> value shall be that which turns off the associated drivers. Other examples where the <safe bit> value might be defined as "**0**" or "**1**" (rather than "**X**") are

— The value that an output should have during *INTEST* that minimizes driver current, or
— A preferred value to present to on-chip logic at a component input during *EXTEST*.

B.8.14.3.5 The <ccell> element

This element shall identify for the relevant <port ID> the <cell number> of the control cell that can disable the output.

B.8.14.3.6 The <disable value> element

This element shall give the value that must be scanned into the control cell identified by the previous <ccell> element to disable the port named by the relevant <port ID>.

B.8.14.3.7 The <disable result> element

The value of <port ID> within a given <cell entry> element is the name of a signal. If that signal can be disabled, the value of the <disable result> element within the same <cell entry> shall specify the condition of the driver of that signal when it is disabled. The permissible values shall be

— A high-impedance state (**Z**); or
— A weak “0” external pull down (**WEAK0**); or
— A weak “1” external pull up (**WEAK1**); or
— A weak “0” internal pull down (**PULL0**); or
— A weak “1” internal pull up (**PULL1**); or
— A “kept” state memory of the last strongly driven logic state (**KEEPER**)

The values **WEAK1** and **WEAK0** would be used for asymmetrical drivers, such as TTL open-collector or ECL open-emitter outputs, when a pull-down or a pull-up is external to the component. The values **PULL0** and **PULL1** would be used for asymmetrical drivers, such as TTL open-collector or ECL open-emitter outputs, when a pull-down or a pull-up is internal to the component. The value **KEEPER** would be used for drivers that maintain a weakly driven memory of the state last seen on the board network to which such a driver is connected.

NOTES

1—It must be emphasized that bus keepers generally do not retain a reliable logic state useful as part of a logic implementation. Indeed, any glitch or system noise on a “kept” bus may upset the state of any connected keepers. Drivers with bus keepers can be thought of as types that disable to a high-impedance state that always stays out of the forbidden voltage zone between defined low and high logic values. Just as a high-impedance state conveys no information, neither does a kept state. Implementers of board or system application software probably will choose the same treatment of the <disable result> values **KEEPER** and **Z** .

2—The keeper feature is an important parametric option in the design of a device’s drivers. An IC vendor using such drivers would want to verify their action on an IC tester. By giving the ability for BSDL to denote the existence of such drivers, IC test software can automatically set up tests (at the logical level) for these features that are similar to hysteresis measurements. Of course, the analog parameters of a keeper, like other analog information, are not described in BSDL.

By processing the <boundary-scan register description>, it is possible for software to check that every non-linkage, non-TAP controller, non-compliance enable, non-grouped port name in the port statement has been named by a <port ID> of the <boundary-scan register description>. Missing <port ID> values (other than linkage, TAP controller ports, compliance-enable ports, and grouped ports) identify digital system signals lacking corresponding cells in the boundary-scan register, which indicates a noncompliant device or an error in entering the BSDL description.

B.8.14.4 Boundary-scan register semantic checks

a) The value of the <integer> (LENGTH) in the <boundary length stmt> shall be greater than zero.
b) Every <cell entry> element of the <cell table> shall include a <cell number> element with a value in the range from 0 to LENGTH - 1.
c) Every <integer> with a value between 0 and LENGTH - 1 shall appear as a <cell number> in some <cell entry> of the <boundary register stmt>.
d) Only a pair of merged cells (see B.11.1.3) shall correspond to two <cell entry> elements containing identical <cell number> elements in the <cell table>. Moreover,
    1) The only possible mergers shall be of cells with <function> equal to **INPUT** and cells with <function> equal to **OUTPUT2** , **OUTPUT3** , **CONTROL** , or **CONTROLR**.
    2) The value of the <cell name> element in both <cell entry> elements shall be equal.
    3) The <safe bit> values for these two cells shall be identical unless one value is **X**.
    4) The <data source> values of the <capture descriptor> values (see B.10.2) for these two cells shall be identical for all supported instructions.

e) Every <cell name> appearing in the <cell table> shall be the name of a cell described in either the Standard VHDL Package or a user-supplied VHDL package.

f) Any value of a <port ID> element in a <cell spec> that is not a <subscripted port name> shall be a <port name> in an <identifier list> element of a <pin spec> in the <logical port description> such that the value of the <port dimension> element in that <pin spec> is **bit**.

g) Any <port name> of a <port ID> appearing in a <cell spec> shall appear also in the <logical port description>; if a given <port ID> in a <cell spec> is a <subscripted port name>, its <subscript> shall be in the range of the <range> element in the <pin spec> element of the <logical port description> in which the <VHDL identifier> of that <subscripted port name> appears as a value of a <port name> element.

h) A <port ID or null> shall have the value asterisk (*) when, in the same <cell spec> element, the <function> element has the value **CONTROL**, **CONTROLR**, or **INTERNAL**.

i) Any <cell info> element containing a <function> element equal to **INPUT**, **CONTROL**, **CONTROLR**, **INTERNAL**, **OBSERVE_ONLY**, or **CLOCK** shall not also contain a <disable spec> element.

j) Any <cell entry> element containing a <function> element equal to **OUTPUT3** or **BIDIR** also shall contain a <disable spec> element.

k) Any <cell entry> element including a <function> element equal to **OUTPUT2** and also containing a <disable spec> element shall satisfy the following conditions:

1) The value of <cell number> shall equal the value of <ccell>. (This implies that an **OUTPUT2** cell may control itself.)

2) The value of the <disable value> shall be equivalent to the (weak) logical value of the <disable result>. That is, when the <disable value> is **0**, the <disable result> be **WEAK0** or **PULL0**, and when the <disable value> is **1**, the <disable result> shall be **WEAK1** or **PULL1**.

l) Any <cell entry> element containing a <function> element equal to **BIDIR** and a <ccell> element value equal to the value of the <cell number> element (implying a bidirectional cell that controls itself) shall have the value of the <disable value> equivalent to the (weak) logical value of the <disable result> element. That is, when the <disable value> is **0**, the <disable result> shall be **WEAK0** or **PULL0**, and when the <disable value> is **1**, the <disable result> shall be **WEAK1** or **PULL1**.

m) For any <cell entry> including a <function> element equal to **CONTROL** or **CONTROLR**, the value of the <cell number> element of that <cell entry> shall appear as the value of the <ccell> element of the <disable spec> element of other <cell entry> elements.

n) For any <cell entry> including a <function> element equal to **CONTROL** or **CONTROLR**, the value of the <safe bit> element of that <cell entry> shall be equal to the value of the <disable value> element of the <disable spec> element of the other <cell entry> elements that satisfy semantic check B.8.14.4 m) (i.e., the controlled cells).

o) The <ccell> element of a <disable spec> element shall have only the values permitted under the conditions of semantic checks B.8.14.4 k), B.8.14.4 l), and B.8.14.4 m).

p) When the value of <conformance identification> (see B.8.6) is **STD_1149_1_1993** or **STD_1149_1_2001**, if two distinct <disable spec> elements in the <boundary register stmt> have <ccell> elements with a common value, the values of the <disable value> elements of these two <disable spec> elements also shall be equal.

q) For any <port ID> element appearing in a <cell entry> element of the <boundary register stmt>, a <VHDL identifier> with the same value as the given <port ID> element shall appear in an <identifier list> of a <pin spec> in the <logical port description>.

r) If a <VHDL identifier> appears in an <identifier list> of a <pin spec> in the <logical port description>, and if the <pin type> appearing in that <pin spec> has the value **linkage**, the given <VHDL identifier> shall never appear as

1) The <port ID> in any <cell entry> of the <boundary register stmt>, or

2) The initial <VHDL identifier> of a <subscripted port name> serving as a <port ID> in any <cell entry> of the <boundary register stmt>

s) Excepting those elements explicitly mentioned in the following list, all <port ID> elements with a <VHDL identifier> appearing in an <identifier list> of a <pin spec> also including a <pin type> not

equal to **linkage** [in the <logical port description> (see B.8.3) of the BSDL description] shall appear as <port ID> elements in the <boundary register stmt>. Specifically exempted from this check are any <port ID> elements satisfying any of the following conditions:

1) Semantic check B.8.8.3 d) (grouped ports)
2) Semantic check B.8.9.3 b) (scan port identification)
3) Semantic check B.8.10.3 e) (compliance-enable description)

NOTE—This semantic check means that all nonexempted system pins shall be associated with cell(s) in the boundary-scan register. Further, this semantic check means that all exempted system pins shall not be associated with cell(s) in the boundary-scan register.

Semantic checks B.8.14.4 q), B.8.14.4 r), and B.8.14.4 s) state which <port ID> elements in the <logical port description> shall appear in the <boundary register stmt>, and vice versa. The next semantic checks state the properties that shall exist for <function> elements within <cell entry> elements.

t) It is allowed that additional <cell entry> elements with the <function> **OBSERVE_ONLY** may have <port ID> elements in common with any other <cell entry> obeying semantic check B.8.14.4 u).

NOTE—Additional **OBSERVE_ONLY** cells shall always monitor the state of a system I/O pin.

u) For any <port ID> element appearing in a <cell entry> element of the <boundary register stmt>, when the <pin type> in the <pin spec> of that <port ID> is
   1) **in**, the <function> of the <cell entry> shall be **INPUT**, **CLOCK**, or **OBSERVE_ONLY** only.
   2) **out**, the <function> of the <cell entry> shall be **OUTPUT2** or **OUTPUT3** only; furthermore, no other <cell entry> containing the same <port ID> shall have <function> **OUTPUT2** or **OUTPUT3**; also, when the <function> is **OUTPUT2**, the <cell entry> shall have a <disable spec> according to semantic check B.8.14.4 k).
   3) **buffer**, the <function> of the <cell entry> shall be **OUTPUT2** only, and the <cell entry> shall not contain a <disable spec>; furthermore, no other <cell entry> containing the same <port ID> shall have <function> **OUTPUT2**.
   4) **inout**, the <function> of the <cell entry> shall be **BIDIR**, **OUTPUT2**, **OUTPUT3**, **INPUT**, or **OBSERVE_ONLY** only; if the <function> value is **BIDIR**, no other <cell entry> containing the same <port ID> shall have the <function> **BIDIR**, **OUTPUT2**, or **OUTPUT3**; if the <function> of the <cell entry> is **OUTPUT2** or **OUTPUT3**, no other <cell entry> containing the same <port ID> shall exist with the <function> value of **OUTPUT2** or **OUTPUT3**, and at least one other <cell entry> containing the same <port ID> but a different <cell number> shall exist with the <function> value of **INPUT** or **OBSERVE_ONLY**, and vice versa.

v) The <function> in a <cell entry> shall be an existing <cell context> (see B.10.2) within the <capture descriptor> of the cell named by <cell name>.

w) If **INTEST** occurs as the value of an <instruction name> element in an <opcode description> element of the <instruction opcode stmt>, for each <port ID> element satisfying semantic check B.8.14.4 s), a <cell entry> shall exist that references that <port ID> *and* that possesses *INTEST support capability*.

NOTES

1—For this semantic check, a given <cell entry> does not possess *INTEST support capability* unless the <capture descriptor list> (see B.10.2) of the cell design named by the <cell name> element meets one of the following conditions:

—for a <function> element value of **INPUT**, **CLOCK**, **OUTPUT2**, **OUTPUT3**, **CONTROL**, or **CONTROLR**, a <capture descriptor> element contains a <cell context> element value that matches the <function> element value *and* has a <capture instruction> element value of **INTEST**; or,

—for a <function> element value of **BIDIR**, one <capture descriptor> element contains a <cell context> element value of **BIDIR_IN** *and* another <capture descriptor> element contains a <cell context> element value of **BIDIR_OUT**, *and* both such <capture descriptor> elements have a <capture instruction> element value of **INTEST**.

2—For this semantic check, a given <cell entry> does not possess *INTEST support capability* if its <function> element value is **OBSERVE_ONLY**. An **OBSERVE_ONLY** cell cannot by itself provide *INTEST support capability*.

See Clause B.11 for more information given by example for describing the boundary-scan register.

B.8.15 RUNBIST description

The goal of this portion of a BSDL description shall be to provide support for the *RUNBIST* instruction as specified within this standard. The intent is to describe only those aspects of the *RUNBIST* instruction that this standard specifies. In some cases, this may not completely define the Built-In Self-Test (BIST) operational environment. In such cases, additional information must be supplied externally.

Note that the following features of a BIST implementation are not supported explicitly by BSDL:

— Timing-related information (beyond active clock and number of clock cycles)
— Frequency and phase relationship(s) of clock(s)

B.8.15.1 Syntax

<runbist description>::=
attribute RUNBIST_EXECUTION of <component name>
: entity is " <runbist spec> **";**

<runbist spec>::= <wait spec> **,** <pin spec> **,** <signature spec>
<wait spec>::= **WAIT_DURATION** **(** <duration spec> **)**
<duration spec>::= <clock cycles list> | <time> [**,** <clock cycles list>]
<clock cycles list>::= <clock cycles> { **,** <clock cycles> }
<time>::= <real number>
<clock cycles>::= <port ID> <integer> (see B.6.3)
<pin spec>::= **OBSERVING** <condition> **AT_PINS**
<condition>::= **HIGHZ** | **BOUNDARY**
<signature spec>::= **EXPECT_DATA** <det pattern>
<det pattern>::= <bit> { <bit> }
<bit>::= **0** | **1**

A <det pattern> shall be a contiguous sequence of one or more **0** and **1** characters containing no spaces or format effectors. For example, **001100** and **110101** are legal. However **100 X00** is not legal because of the embedded space and the **X** character.

Time shall be specified in seconds (via the value of the <time> element). Where both time and clock cycles are specified, they shall be interpreted as the maximum of the time specified or the time required to apply the required number of clock cycles. Where more than one clock is specified, the duration shall be the time required for all of the clock inputs to receive the specified number of clock cycles.

B.8.15.2 Examples

Example 1

```
attribute RUNBIST_EXECUTION of BIST_IC1: entity is
        "Wait_Duration (1.0e-3)," &
        "Observing HIGHZ At_Pins," &
        "Expect_Data 010101";
```

In this example, the value of <time> in the <wait spec> is specified at 1 ms, which is the minimum duration the device needs to stay in the *Run-Test/Idle* controller state. Also, note that the output pins are forced to

high impedance, which implies that there is no need to initialize the update latches of the boundary-scan register.

Example 2

```
attribute RUNBIST_EXECUTION of BIST_IC2: entity is
        "Wait_Duration (TCK 23000)," &
        "Observing HIGHZ At_Pins," &
        "Expect_Data 010101";
```

In this example, the device needs to wait in the *Run-Test/Idle* controller state for the duration sufficient for the application of 23 000 clock cycles at TCK.

Example 3

```
attribute RUNBIST_EXECUTION of BIST_IC3: entity is
        "Wait_Duration (1.0e-3, TCK 23000)," &
        "Observing HIGHZ At_Pins," &
        "Expect_Data 010101";
```

In this example, <wait spec> is to be interpreted as 1 ms or the time required for TCK to receive 23 000 cycles, whichever is greater.

Example 4

```
attribute RUNBIST_EXECUTION of BIST_IC4: entity is
        "Wait_Duration (CLK 100000, SYSCK 24000)," &
        "Observing BOUNDARY At_Pins," &
        "Expect_Data 010101";
```

In this example, <wait spec> is to be interpreted as the time required for CLK and SYSCK to receive 100 000 and 24 000 clock cycles, respectively. Also note that the boundary-scan register is visible at the pins, indicating that it needs to be initialized before the execution of *RUNBIST*.

B.8.15.3 Semantic checks

a) The number of bits in the value of the <det pattern> element of the <signature spec> element shall be equal to the length of the register whose name appears in the <register> element of that <register association> element of the <register access description> in which **RUNBIST** appears as the value of an <instruction name> element.
   1) If the value of the associated <register> element is **BOUNDARY**, the register length shall be specified by the value of the <integer> element of the <boundary length stmt>.
   2) If the value of the associated <register> element is *not* **BOUNDARY**, the register length shall be specified by the explicitly defined value of the <integer> element in that same <register> element.

b) Any value of <port ID> in the <wait spec> statement shall
   1) Appear as the value of <port ID> in the <TCK stmt> of the BSDL description (see B.8.9); or
   2) Appear as the value of <port ID> in a <cell spec> of the <boundary register stmt> in which the <function> element has the value **CLOCK** (see B.8.14).

c) If the <runbist description> statement occurs in a BSDL description, **RUNBIST** shall be the value of some <instruction name> element in the <opcode table> of the <instruction opcode stmt>.

d) Values of <time> and <clock cycles> shall be greater than 0.

e) A given <port ID> shall not appear more than once in the <runbist description> element.

NOTE—The existence of **RUNBIST** in the **INSTRUCTION_OPCODE** table shall not require <runbist description> to be specified in a BSDL description.

B.8.16 INTEST description

The goal of this portion of a BSDL description shall be to describe

— How test patterns are to be applied to the component when the *INTEST* instruction is selected (i.e., the source of clock pulses for the component and the time for which the test logic must remain in the *Run-Test/Idle* controller state to permit execution of each applied test); and
— The external behavior of the component while the *INTEST* instruction is selected.

Note that the test patterns themselves are not specified and are assumed to be provided by an alternative method not specified in this annex. For *INTEST*, the duration shall be not the duration for the entire test (as is the case of *RUNBIST*) but the time required for the application of a single vector. With the application of each vector via the boundary-scan register, this standard permits the device to execute a single step of the operation that may require several clock cycles to complete. Otherwise, the interpretation of <pin spec> shall be identical to that in **RUNBIST_EXECUTION**. The syntax of the <wait spec> and <pin spec> elements shall be as given in B.8.15.

B.8.16.1 Syntax

<intest description>::=
attribute INTEST_EXECUTION of <component name>
: entity is " <intest spec> **";**

<intest spec>::= <wait spec> **,** <pin spec> (see B.8.15)

B.8.16.2 Examples

Example 1

```
attribute INTEST_EXECUTION of IC1: entity is
         "Wait_Duration (1.0e-3)," &
         "Observing HIGHZ At_Pins";
```

In this example, the value of <time> in the <wait spec> is specified at 1 ms, which is the minimum duration the device needs to stay in the *Run-Test/Idle* controller state. Also, note that the output pins are forced to high impedance.

Example 2

```
attribute INTEST_EXECUTION of IC2: entity is
         "Wait_Duration (TCK 250)," &
         "Observing HIGHZ At_Pins";
```

In this example, the device needs to wait in the *Run-Test/Idle* controller state for a duration sufficient for the application of 250 clock cycles of TCK to permit the device to complete one single step of operation.

Example 3

```
attribute INTEST_EXECUTION of IC3: entity is
         "Wait_Duration (CLK 100, SYSCK 200)," &
         "Observing BOUNDARY At_Pins";
```

In this example, <wait spec> is to be interpreted as the time required for CLK and SYSCK to receive 100 and 200 clock cycles, respectively. Also note that the state of the pins is controlled by the data held in the boundary-scan register.

B.8.16.3 Semantic checks

a) Any value of <port ID> in the <wait spec> statement shall
   1) Appear as the value of <port ID> in the <TCK stmt> of the BSDL description (see B.8.9); or,
   2) Appear as the value of <port ID> in a <cell spec> of the <boundary register stmt> in which the <function> element has the value **CLOCK** (see B.8.14).

b) If the <intest description> statement occurs in a BSDL description, **INTEST** shall be the value of some <instruction name> element in the <opcode table> of the <instruction opcode stmt>.

c) Values of <time> and <clock cycles> shall be greater than 0.

d) A given value of <port ID> shall not appear more than once in the <intest description> element.

NOTE—The existence of **INTEST** in the **INSTRUCTION_OPCODE** table shall not require <intest description> to be specified in a BSDL description.

B.8.17 User extensions to BSDL

Optional BSDL extensions shall provide a way to expand BSDL for proprietary needs without losing compatibility with the general definition of BSDL. The Standard VHDL Package **STD_1149_1_2001** defines a VHDL subtype **BSDL_EXTENSION** (as originally defined in Standard VHDL Package **STD_1149_1_1994**). It allows the user to define foreign attributes as being “BSDL extensions.” These generally may be ignored by a BSDL parser. BSDL extensions shall appear in an entity description as the last portion before the (optional) **DESIGN_WARNING** (see B.8.18). In this manner, they may reference any data items defined previously.

B.8.17.1 Syntax

<BSDL extensions>::= <BSDL extension> { <BSDL extension> }

<BSDL extension>::= <extension declaration> | <extension definition>

<extension declaration>::= **attribute** <extension name> **: BSDL_EXTENSION;**

<extension definition>::= **attribute** <extension name> **of** <component name>
: entity is <extension parameter string> **;**

<extension name>::= <entity defined name> | <VHDL package defined name>
<entity defined name>::= <VHDL identifier>
<VHDL package defined name>::= <VHDL identifier>
<extension parameter string>::= <string>

An <extension definition> shall appear after its corresponding <extension declaration>. The <extension declaration> may appear in the description itself or in a user-supplied VHDL package. If several BSDL extensions exist in the description, they may be intermixed in any manner as long as the declaration of an attribute precedes the definition of that attribute. The ability to define BSDL extensions in user-supplied VHDL packages allows global definition of extensions.

B.8.17.2 Examples

Example 1

```
          Package Global_extension is -- An example BSDL extension package
                                                  -- Does not define boundary cells,
                                                  --      just extensions

          use STD_1149_1_2001.all;

     -- Deferred constant declarations go here, if any (see Clause B.10)

          attribute First_extension                : BSDL_EXTENSION; -- Declare BSDL
          attribute Second_extension : BSDL_EXTENSION; --                              extensions here
          attribute Third_extension                : BSDL_EXTENSION;

          end Global_extension;

          package body Global_extension is

     -- Deferred constant definitions go here, if any (see Clause B.10)

          end Global_extension;
```

In this example, a user-supplied VHDL package containing a BSDL extension is given; this package will be referenced by the entity of the next example.

Example 2

```
          entity example is
                   generic (PHYSICAL_PIN_MAP : string := "DW_PACKAGE");

          port (CLK:in bit; Q:out bit_vector(1 to 8);
                   D:in bit_vector(1 to 8); GND, VCC:linkage bit;
                   OC_NEG:in bit; TDO:out bit; TMS, TDI, TCK:in bit);

          use STD_1149_1_2001.all;
          use Global_extension.all; -- Get declarations of
                                              --      global extensions
```

. . . Many deleted lines . . .

```
          -- Local declarations

          attribute Local_extension1: BSDL_extension; --Declare local BSDL
          attribute Local_extension2: BSDL_extension; --                          extensions here

          -- Now, define some proprietary extensions that were declared
          --      in package - Global_Extension

          attribute First_extension of example:entity is                            -- Define attr.
                   " String of data " &                                  -- (global extension)
                   " in proprietary form. ";
          attribute Second_extension of example:entity is
                   " More data, etc. ";
```

```
        -- Local definition

        attribute Local_extension1 of example:entity is                    -- Define attr.
                " Finally defined ";                                -- (local extension)

        -- Optional design warning still located here --

end example;
```

In this example, an entity is shown that uses global extensions as well as local extensions defined in the entity. Note that not all declared extensions are defined (e.g., `Third_extension`).

B.8.17.3 Semantic checks

a) Any <VHDL identifier> appearing as a value of the <extension name> element in an <extension definition> shall appear also as the value of the <extension name> element of an <extension declaration> that occurs earlier in the BSDL description or in a given VHDL package. In the case in which the <extension declaration> occurs in a VHDL package, appearance of the corresponding <extension definition> in the BSDL description shall be considered as an “appearance after” the given <extension declaration>.

b) Each <extension name> value in an <extension declaration> shall be unique even if the <extension name> values are defined in separate places, i.e., in separate user-supplied VHDL packages or one in the BSDL entity and one in a user-supplied VHDL package.

If a value of an <extension name> element appearing in an <extension declaration> never appears as the value of an <extension name> element in any <extension definition> within a given BSDL description, no error shall be generated by an application parsing that description.

NOTE—When inventing names for <extension name> elements, take care to assure uniqueness of the names with respect to names created in other organizations that are also inventing extensions by avoiding common names that might be thought of by others. A company name could be appended to the name to maximize uniqueness.

B.8.18 Design warning

A component designer may know of situations in which the system usage of a component can be subverted via the boundary-scan feature and cause circuit problems. As a simple example, a component may have dynamic system logic that requires clocking to maintain its state. Thus, clocking must be maintained when bringing the component out of system mode and into test mode for *INTEST*. The DESIGN_WARNING attribute shall be assigned a string message to alert future consumers of the potential for problems.

B.8.18.1 Syntax

<design warning>::=
attribute DESIGN_WARNING of <component name> **: entity is** <string> **;**

B.8.18.2 Examples

```
        attribute DESIGN_WARNING of My_IC:entity is
                "Dynamic device, " &
                "maintain clocking for INTEST.";
```

This warning is for application-specific display purposes only. It shall be a textual message of arbitrary length with no specified syntax and is not intended for software analysis.

B.8.18.3 Semantic checks

No semantic checks are necessary.

B.9 The Standard VHDL Package STD_1149_1_2001

The following shall be the complete content of Standard VHDL Package STD_1149_1_2001. Note that both the VHDL package and the VHDL package body are shown. This information shall define the basis of BSDL and typically would be write-protected by a system administrator. An explanation of the cell definitions (e.g., **BC_1** , **BC_2** , etc.) in the package body is given in B.10.2. BSDL descriptions that use <standard VHDL package identifier> **STD_1149_1_2001** shall be processed using this Standard VHDL Package.

NOTE—Where figures are cited, the suffix “c” is used to denote a control cell. The suffix “d” denotes a data cell.

```
package STD_1149_1_2001 is

-- Give component conformance declaration

attribute COMPONENT_CONFORMANCE : string;
-- Give pin mapping declarations

attribute PIN_MAP : string;
subtype PIN_MAP_STRING is string;

-- Give TAP control declarations

type CLOCK_LEVEL is (LOW, BOTH);
type CLOCK_INFO           is record
   FREQ : real;
   LEVEL: CLOCK_LEVEL;
end record;

attribute        TAP_SCAN_IN          : boolean;
attribute        TAP_SCAN_OUT         : boolean;
attribute        TAP_SCAN_CLOCK: CLOCK_INFO;
attribute        TAP_SCAN_MODE : boolean;
attribute        TAP_SCAN_RESET: boolean;

-- Give instruction register declarations

attribute        INSTRUCTION_LENGTH : integer;
attribute        INSTRUCTION_OPCODE : string;
attribute        INSTRUCTION_CAPTURE : string;
attribute        INSTRUCTION_PRIVATE : string;

-- Give ID and USER code declarations

type ID_BITS is ('0', '1', 'x', 'X');
type ID_STRING is array (31 downto 0) of ID_BITS;
attribute IDCODE_REGISTER                :    ID_STRING;
attribute USERCODE_REGISTER:                  ID_STRING;

-- Give register declarations
```

```
attribute REGISTER_ACCESS : string;

-- Give boundary cell declarations

type BSCAN_INST is (EXTEST, SAMPLE, INTEST);
type CELL_TYPE is (INPUT, INTERNAL, CLOCK, OBSERVE_ONLY,
                          CONTROL, CONTROLR, OUTPUT2,
                          OUTPUT3, BIDIR_IN, BIDIR_OUT);
type CAP_DATA is (PI, PO, UPD, CAP, X, ZERO, ONE);
type CELL_DATA is record
   CT : CELL_TYPE;
   I    : BSCAN_INST;
   CD : CAP_DATA;
end record;
type CELL_INFO is array (positive range <>) of CELL_DATA;

-- Boundary cell deferred constants (see package body)

constant BC_0          : CELL_INFO;
constant BC_1          : CELL_INFO;
constant BC_2          : CELL_INFO;
constant BC_3          : CELL_INFO;
constant BC_4          : CELL_INFO;
constant BC_5          : CELL_INFO;
constant BC_6          : CELL_INFO;
constant BC_7          : CELL_INFO;
constant BC_8          : CELL_INFO;
constant BC_9          : CELL_INFO;
constant BC_10           : CELL_INFO;

-- Boundary register declarations

attribute BOUNDARY_LENGTH : integer;
attribute BOUNDARY_REGISTER : string;

-- Miscellaneous

attribute PORT_GROUPING : string;
attribute RUNBIST_EXECUTION : string;
attribute INTEST_EXECUTION : string;
subtype BSDL_EXTENSION is string;
attribute COMPLIANCE_PATTERNS : string;
attribute DESIGN_WARNING : string;

end STD_1149_1_2001;              -- End of 1149.1-2001 Package


package body STD_1149_1_2001 is                     -- Standard boundary cells

-- Generic cell capturing minimum allowed data

constant BC_0 : CELL_INFO :=
  ((INPUT,         EXTEST,       PI),           (OUTPUT2,        EXTEST,       X),
    (INPUT,        SAMPLE,       PI),           (OUTPUT2,        SAMPLE,       PI),
```

```
   (INPUT,         INTEST,       X),            (OUTPUT2,        INTEST,       PI),
   (OUTPUT3, EXTEST,             X),            (INTERNAL, EXTEST,             X),
   (OUTPUT3, SAMPLE,             PI),           (INTERNAL, SAMPLE,             X),
   (OUTPUT3, INTEST,             PI),           (INTERNAL, INTEST,             X),
   (CONTROL, EXTEST,             X),            (CONTROLR, EXTEST,             X),
   (CONTROL, SAMPLE,             PI),           (CONTROLR, SAMPLE,             PI),
   (CONTROL, INTEST,             PI),           (CONTROLR, INTEST,             PI),
   (BIDIR_IN,EXTEST,             PI),           (BIDIR_OUT, EXTEST, X ),
   (BIDIR_IN,SAMPLE,             PI),           (BIDIR_OUT, SAMPLE, PI),
   (BIDIR_IN,INTEST,             X ),           (BIDIR_OUT, INTEST, PI),
   (OBSERVE_ONLY, SAMPLE, PI),                  (OBSERVE_ONLY, EXTEST, PI) );

-- Description for f11-18, f11-30, f11-34c, f11-34d, f11-36c, f11-46d

constant BC_1 : CELL_INFO :=
  ((INPUT,         EXTEST,       PI),     (OUTPUT2,        EXTEST,       PI),
    (INPUT,         SAMPLE,       PI),     (OUTPUT2,        SAMPLE,       PI),
    (INPUT,         INTEST,       PI),     (OUTPUT2,        INTEST,       PI),
    (OUTPUT3, EXTEST,             PI),     (INTERNAL, EXTEST,             PI),
    (OUTPUT3, SAMPLE,             PI),     (INTERNAL, SAMPLE,             PI),
    (OUTPUT3, INTEST,             PI),     (INTERNAL, INTEST,             PI),
    (CONTROL, EXTEST,             PI),     (CONTROLR, EXTEST,             PI),
    (CONTROL, SAMPLE,             PI),     (CONTROLR, SAMPLE,             PI),
    (CONTROL, INTEST,             PI),     (CONTROLR, INTEST,             PI) );

-- Description for f11-14, f11-31, f11-35c, f11-35d, f11-37c,
--      f11-38c, f11-39(output) and f11-41c

constant BC_2 : CELL_INFO :=
  ((INPUT,         EXTEST,       PI),     (OUTPUT2, EXTEST,              UPD),
    (INPUT,         SAMPLE,       PI),     (OUTPUT2, SAMPLE,              PI),
    (INPUT,         INTEST,       UPD),      -- Intest on output2 not supported
    (OUTPUT3, EXTEST,             UPD), (INTERNAL, EXTEST,               PI),
    (OUTPUT3, SAMPLE,             PI),     (INTERNAL, SAMPLE,             PI),
    (OUTPUT3, INTEST,             PI),     (INTERNAL, INTEST,             UPD),
    (CONTROL, EXTEST,             UPD), (CONTROLR, EXTEST,               UPD),
    (CONTROL, SAMPLE,             PI),     (CONTROLR, SAMPLE,             PI),
    (CONTROL, INTEST,             PI),     (CONTROLR, INTEST,             PI) );

-- Description for f11-15

constant BC_3 : CELL_INFO :=
  ((INPUT, EXTEST,            PI),        (INTERNAL, EXTEST,             PI),
    (INPUT, SAMPLE,           PI),        (INTERNAL, SAMPLE,             PI),
    (INPUT, INTEST,           PI),        (INTERNAL, INTEST,             PI) );

-- Description for f11-16, f11-17, f11-39(input)

constant BC_4 : CELL_INFO :=
  ((INPUT, EXTEST,            PI),              -- Intest on input not supported
    (INPUT, SAMPLE,           PI),
    (OBSERVE_ONLY, EXTEST, PI),
    (OBSERVE_ONLY, SAMPLE, PI),                 -- Intest on observe_only not supported
    (CLOCK, EXTEST,           PI),       (INTERNAL, EXTEST,             PI),
```

```
   (CLOCK, SAMPLE,            PI),       (INTERNAL, SAMPLE,             PI),
   (CLOCK, INTEST,            PI),       (INTERNAL, INTEST,             PI) );

-- Description for f11-46c, a combined input/control

constant BC_5 : CELL_INFO :=
  ((INPUT, EXTEST,            PI),       (CONTROL, EXTEST,            PI),
    (INPUT, SAMPLE,           PI),       (CONTROL, SAMPLE,            PI),
    (INPUT, INTEST,           UPD),      (CONTROL, INTEST,            UPD) );

-- Description for f11-38d, a reversible cell
-- !! Not recommended; replaced by BC_7 below !!

constant BC_6 : CELL_INFO :=
  ((BIDIR_IN, EXTEST,             PI),     (BIDIR_OUT, EXTEST,             UPD),
    (BIDIR_IN, SAMPLE,            PI),     (BIDIR_OUT, SAMPLE,             PI),
    (BIDIR_IN, INTEST,            UPD), (BIDIR_OUT, INTEST,                PI) );

-- Description for f11-37d, self monitor reversible
-- !! Recommended over cell BC_6 !!

constant BC_7 : CELL_INFO :=
  ((BIDIR_IN, EXTEST,             PI),     (BIDIR_OUT, EXTEST,             PO),
    (BIDIR_IN, SAMPLE,            PI),     (BIDIR_OUT, SAMPLE,             PI),
    (BIDIR_IN, INTEST,            UPD), (BIDIR_OUT, INTEST,                PI) );

-- Description for 11-40, f11-41d

constant BC_8 : CELL_INFO :=
  -- Intest on bidir not supported
  ((BIDIR_IN, EXTEST,             PI),     (BIDIR_OUT, EXTEST,             PO),
    (BIDIR_IN, SAMPLE,            PI),     (BIDIR_OUT, SAMPLE,             PO) );

-- Description for f11-32

constant BC_9 : CELL_INFO :=
  -- Self-monitoring output that supports Intest
  ((OUTPUT2, EXTEST,            PO),     (OUTPUT3, EXTEST,            PO),
    (OUTPUT2, SAMPLE,           PI),     (OUTPUT3, SAMPLE,            PI),
    (OUTPUT2, INTEST,           PI),     (OUTPUT3, INTEST,            PI) );

-- Description for f11-33

constant BC_10 : CELL_INFO :=
  -- Self-monitoring output that does not support Intest
  ((OUTPUT2, EXTEST,            PO),     (OUTPUT3, EXTEST,            PO),
    (OUTPUT2, SAMPLE,           PO),     (OUTPUT3, SAMPLE,            PO) );

end STD_1149_1_2001;              -- End of IEEE Std 1149.1-2001 Package Body
```

B.10 User-supplied VHDL packages

B.10.1 User-supplied VHDL package structure

A user-supplied VHDL package shall be used to express the behavior of user-designed boundary-scan register cells. It has a VHDL package section and a VHDL package body section. The VHDL package section is abbreviated compared to the Standard VHDL Package, since the definitions of BSDL are supplied in the Standard VHDL Package specified by the <standard use statement>. The names of the cells that are defined in the VHDL package body shall be given (they are called deferred constants).

NOTE—When writing a user-supplied VHDL package, it is possible to create an error if a later (e.g., 2001) construct such as a **BSDL_EXTENSION** is referenced that is not defined in an earlier-defined (e.g., 1990) Standard VHDL Package specified in the <standard use statement>. To avoid such an error, a 2001 user-supplied package should reference **STD_1149_1_2001** .

B.10.1.1 Syntax

<user package>::= **package** <user package name> **is**
<standard use statement>
{ <extension declaration> } (see B.8.17)
{ <deferred constant> }
end <user package name> **;**

<user package body>::= **package body** <user package name> **is**
<standard use statement>
{ <cell description constant> } (see B.10.2)
end <user package name> **;**

<user package name>::= <VHDL identifier>
<deferred constant>::= **constant** <cell name> **: CELL_INFO;**
<cell name>::= <VHDL identifier>

For <cell description constant>, see B.10.2.

B.10.1.2 Examples

For an example, see B.10.3.

B.10.1.3 Semantic checks

a) The <user package name> value shall be unique.
b) All <cell name> values shall be unique.
c) The <user package name> value in the <user package> shall match the <user package name> value in the <user package body>.

B.10.2 Cell description constants

The syntax of cell description constants shall be as given in B.10.2.1.

B.10.2.1 Syntax

<cell description constant>::=
constant <cell name> **: CELL_INFO := (** <capture descriptor list> **) ;**

<cell name>::= <VHDL identifier>
<capture descriptor list>::= <capture descriptor> { , <capture descriptor> }
<capture descriptor>::= **(** <cell context> **,** <capture instruction> **,** <data source> **)**
<cell context>::= **INPUT** | **OUTPUT2** | **OUTPUT3** | **INTERNAL** | **CONTROL** |
CONTROLR | **CLOCK** | **BIDIR_IN** | **BIDIR_OUT** | **OBSERVE_ONLY**
<capture instruction>::= **EXTEST** | **SAMPLE** | **INTEST**
<data source>::= **PI** | **PO** | **CAP** | **UPD** | **ZERO** | **ONE** | **X**

NOTE—Although the standard-defined instruction *PRELOAD* does operate the boundary-scan register, it is not listed as a <capture instruction> element since all cells capture unspecified data (“**X**”) when *PRELOAD* is in effect (see 8.7).

A cell description constant shall be a specific VHDL constant record made up of a variable number of data triples containing VHDL enumerated types. For example, a capture descriptor looks like the following:

B.10.2.2 Examples

Example 1

```
(INPUT, EXTEST, PI)
```

This can be read as, “for this cell used as an **INPUT** cell, while *EXTEST* is in effect, the capture flip-flop loads the Parallel Input (**PI**) data at the *Capture-DR* controller state.”

Example 2

```
(BIDIR_IN, INTEST, UPD)
```

This can be read as, “for this cell used as a bidirectional cell acting as an input (**BIDIR_IN**), while *INTEST* is in effect, the capture flip-flop loads the value of the Update flip-flop (or latch) data (**UPD**) at the *CAPTURE-DR* controller state.”

Example 3

```
(OUTPUT2, SAMPLE, PI)
```

This can be read as, “for this cell used as a (2-state) output cell (**OUTPUT2**), while *SAMPLE* is in effect, the capture flip-flop loads the Parallel Input (**PI**) data at the *Capture-DR* controller state.”

Example 4

```
(OUTPUT3, EXTEST, ZERO)
```

This can be read as, “for this cell used as a (3-state) output cell (**OUTPUT3**), while *EXTEST* is in effect, the capture flip-flop loads a logic 0 (**ZERO**) at the *Capture-DR* controller state.”

B.10.2.3 Cell context values

Table B.5 lists <cell context> values.

With the exception of <cell context> values of **BIDIR_IN** and **BIDIR_OUT**, all the <cell context> values in Table B.5 shall map onto the like-named <function> values in Table B.4 and supporting text (see B.8.14.3.3). The <cell context> values of **BIDIR_IN** and **BIDIR_OUT** both shall map onto the <function> value **BIDIR**. The behavior of a **BIDIR** cell shall be dependent on whether it is currently set to be driving

Table B.5—Cell context element values and meanings

Cell context value	Meaning
INPUT	Control-and-observe input cell
CLOCK	Observe-only cell for clock pins (supports *INTEST* instruction)
OUTPUT2	2-state output cell
OUTPUT3	3-state output cell
CONTROL	Output enable or direction control cell
CONTROLR	**CONTROL** with preset/clear at *Test-Logic-Reset*
BIDIR_IN	Single-cell bidirectional pin acting as input
BIDIR_OUT	Single-cell bidirectional pin acting as output
INTERNAL	Internal cell, not associated with a system pin
OBSERVE_ONLY	Observe-only cell, associated with a system pin

data out (**BIDIR_OUT**) or receiving data in (**BIDIR_IN**), as determined by the data value contained in the controlling cell identified by the <ccell> value.

B.10.2.4 Data source values

Table B.6 gives the values for <data source>. Figure B.7 gives a general model of the data source possibilities.

Figure B.7—A general model of a boundary-scan register cell showing CAP inputs

It is important to know the context of a cell to know how to interpret the data source. For example, the cell design in this standard at Figure 11-18 (called **BC_1**) can be used as an input cell (**INPUT**), an output cell for a 2-state pin (**OUTPUT2**), a 3-state pin (**OUTPUT3**), an internal cell (**INTERNAL**), or a control cell

Table B.6—Data source element values and meanings

Data source value	Meaning
PI	Parallel input
PO	Parallel output (the output pad if a driver is present)
CAP	Capture flip-flop data
UPD	Update flip-flop (or latch) data
ZERO	Constant “0”
ONE	Constant “1”
X	Unknown data

(**CONTROL**). This context determines how software shall interpret **PI** and **PO** . If the cell is used as an input (or is a bidirectional cell acting as an input), **PI** shall be interpreted as a system pin whose data is being captured. If the cell is used as an output (or a bidirectional cell acting as an output), **PI** shall be interpreted as the output from the system logic; during *EXTEST*, the cell would capture **X** unless a full simulation of the system logic were used to predict the system logic output. If the cell is used as an output, **PO** shall be the system pin; during *EXTEST*, the cell would capture board levels outside the component. When the cell is used as an input, **PO** will capture **X** .

B.10.2.4.1 Semantic checks

In the following tables, an “L” shall indicate legality. An “M” shall indicate legality in the case of merged cells only. An “A” shall indicate legality when the cell is an additional cell not mandated by this standard. A <capture descriptor> shall be a (<cell context> <capture instruction> <data source>) element. An example of an illegal <capture descriptor> is (**INPUT** , **EXTEST** , **UPD**).

a) For a <capture descriptor> element with a <cell context> element equal to **INPUT** , **CLOCK** , or **BIDIR_IN** , legal <data source> values for given <capture instruction> elements shall be as given in Table B.7.

Table B.7—Legal capture sources for <cell context> of INPUT, CLOCK, and BIDIR_IN

<capture instruction>	PI	PO	UPD	CAP	X	ZERO	ONE
EXTEST	L	–	–	–	–	–	–
SAMPLE	L	–	–	–	–	–	–
INTEST	L	L	L	L	L	L	L

b) For a <capture descriptor> element with a <cell context> element equal to **OUTPUT2** , **OUTPUT3** , or **BIDIR_OUT** , legal <data source> values for given <capture instruction> elements shall be as given in Table B.8.

c) For a <capture descriptor> element with a <cell context> element equal to **CONTROL** or **CONTROLR** , legal <data source> values for given <capture instruction> elements shall be as given in Table B.9.

Table B.8—Legal capture sources for <cell context> of OUTPUT2, OUTPUT3, and BIDIR_OUT

<capture instruction>	PI	PO	UPD	CAP	X	ZERO	ONE
EXTEST	L	L	L	L	L	L	L
SAMPLE	L	L[a]	–	–	–	–	–
INTEST	L	–	–	–	–	–	–

[a] This combination becomes legal with this 2001 version of BSDL. Beginning with edition IEEE Std 1149.1-2001, this standard now allows an output of the system logic to be sampled at the data output (pin) of the associated output buffer as well as at the data input of the associated output buffer [see Rules 11.6.1 a) and 11.6.1 h)].

Table B.9—Legal capture sources for <cell context> of CONTROL and CONTROLR

<capture instruction>	PI	PO	UPD	CAP	X	ZERO	ONE
EXTEST	L	L	L	L	L	L	L
SAMPLE	L	–	–	–	–	–	–
INTEST	L	–	M	–	–	–	–

d) For a <capture descriptor> element with a <cell context> element equal to **INTERNAL**, legal <data source> values for given <capture instruction> elements shall be as given in Table B.10.

Table B.10—Legal capture sources for <cell context> of INTERNAL

<capture instruction>	PI	PO	UPD	CAP	X	ZERO	ONE
EXTEST	L	L	L	L	L	L	L
SAMPLE	L	L	L	L	L	L	L
INTEST	L	L	L	L	L	L	L

NOTE—For a <cell context> of **INTERNAL**, a <capture descriptor> value of **PI** is essentially identical to **X** since internal cells do not capture anything other than constant 0s (**ZERO**), 1s (**ONE**), or the values previously shifted in (**CAP**, **UPD**, or **PO**).

e) For a <capture descriptor> element with a <cell context> element equal to **OBSERVE_ONLY**, legal <data source> values for given <capture instruction> elements shall be as given in Table B.11.

Table B.11—Legal capture sources for <cell context> of OBSERVE_ONLY

<capture instruction>	PI	PO	UPD	CAP	X	ZERO	ONE
EXTEST	L	–	–	–	–	–	–
SAMPLE	L	–	–	–	–	–	–
INTEST	A	–	–	–	–	–	–

f) No combination of a <cell context> value and a <capture instruction> value shall appear more than once in a single <capture descriptor list>.

g) The <cell name> value of a <cell description constant> in a <user package body> shall match the <cell name> value of a <deferred constant> in the <user package>, where the <user package body> and <user package> specify the same <user package name>.

NOTE—In the 1990 version of BSDL, the *RUNBIST* instruction was included as one of the <capture instruction> elements, but it does not appear in this 2001 version (nor in the earlier 1994 version). This reflects the fact that *RUNBIST* may or may not reference the boundary-scan register and that the **RUNBIST_EXECUTION** attribute has been added to describe *RUNBIST* capture behavior.

B.10.3 Example of a user-supplied VHDL package

The following is an example of a user-supplied VHDL package that describes two new cells. These cells are able to capture constants (“0” and “1”) during certain situations. For example, used as outputs during *EXTEST*, they capture constant data rather than the system logic values (usually interpreted as “X”). By using these cells in the output cell positions of a boundary-scan register, it is possible to implement an informal ID code. They will capture a pattern of “1” and “0” bits.

```
-- User-defined package describing two new cells

package USER_PACKAGE is

          use STD_1149_1_2001.all;                  -- Get definition of "Cell_Info"

          -- Boundary Cell deferred constants (defined in package body)

          constant USER_0           : CELL_INFO;
          constant USER_1           : CELL_INFO;

          end USER_PACKAGE;                         -- End of user package


package body USER_PACKAGE is                        -- User boundary cells

          use STD_1149_1_2001.all;

          constant USER_0 : CELL_INFO :=
          ((OUTPUT2, EXTEST,             ZERO),
            (OUTPUT2, SAMPLE,            PI),
            (OUTPUT3, EXTEST,            ZERO), (INTERNAL, EXTEST,               ZERO),
            (OUTPUT3, SAMPLE,            PI),       (INTERNAL, SAMPLE,           PI),
            (OUTPUT3, INTEST,            PI),       (INTERNAL, INTEST,           PI),
            (CONTROL, EXTEST,            ZERO), (CONTROLR, EXTEST,               ZERO),
            (CONTROL, SAMPLE,            PI),       (CONTROLR, SAMPLE,           PI),
            (CONTROL, INTEST,            PI),       (CONTROLR, INTEST,           PI) );

          constant USER_1 : CELL_INFO :=
          ((OUTPUT2, EXTEST,             ONE),
            (OUTPUT2, SAMPLE,            PI),
            (OUTPUT3, EXTEST,            ONE),      (INTERNAL, EXTEST,           ONE),
            (OUTPUT3, SAMPLE,            PI),       (INTERNAL, SAMPLE,           PI),
            (OUTPUT3, INTEST,            PI),       (INTERNAL, INTEST,           PI),
            (CONTROL, EXTEST,            ONE),      (CONTROLR, EXTEST,           ONE),
```

```
          (CONTROL, SAMPLE,            PI),       (CONTROLR, SAMPLE,            PI),
          (CONTROL, INTEST,            PI),       (CONTROLR, INTEST,            PI) );

end USER_PACKAGE;                                 -- End of user package body
```

B.11 Example applications of BSDL

B.11.1 Boundary-scan register description

The following examples illustrate a number of “special case” boundary-scan register structures.

B.11.1.1 Multiple cells per pin

Component pins can be serviced by more than one cell. Each cell can perform a different function. Note that this function is with respect to the boundary-scan register cell, not the component pin. For example, on a bidirectional pin (see Figure 11-36), one cell can serve as an input receiver while the other serves as an output driver. Additional **OBSERVE_ONLY** cells may be connected to any I/O pin.

The component shown in Figure B.8 will be used to illustrate a boundary-scan register with several **OBSERVE_ONLY** cells.

Cell 2 is an additional **OBSERVE_ONLY** cell associated with bidirectional pin 9 of the component.

Cell 7 is an additional **OBSERVE_ONLY** cell associated with output pin 8 of the component. Cells 7, 8, and 9 may have been a programmable two-cell bidirectional implementation that has been reprogrammed to a 2-state output.

Cell 15 is an additional **OBSERVE_ONLY** cell associated with input pin 6 of the component. Cells 13 and 14 are also associated with pin 6, but they are described as **INPUT** cells and are connected to the system logic.

Cell 17 is an additional **OBSERVE_ONLY** cell associated with input pin 5 of the component. Cell 16 is the normal **INPUT** cell for pin 5.

If the component in Figure B.8 supports *INTEST*, cell 18 shall be of type **CLOCK**. If the component does not support *INTEST*, cell 18 could be either be an **INPUT** cell or an **OBSERVE_ONLY** cell.

B.11.1.2 Internal cells

Internal cells can be used to capture “constants” or system-logic-dependent values (0s and 1s) within a design. One proposed use of this possibility is the capturing of a hard-coded value (perhaps in the first few bits of the boundary-scan register) as an informal identification code. Another application is to monitor power connections to ensure that they are receiving the correct input supply and to capture a data bit based on the measured results. If the power connections are good, the data loaded will be 1, for example, while a faulty power input would cause a 0 to be captured. Internal cells, with either control-and-observe capability or observe-only capability, may sit at the boundary between digital and analog sections of the core circuitry. Finally, there may be “extra,” unused cells in a programmable component (see Clause 11).

Note that this standard does not allow system logic to surround internal cells, as shown in Figure 11-42.

The component shown in Figure B.8 will be used to illustrate a boundary-scan register with several **INTERNAL** cells.

Figure B.8—A component that illustrates several OBSERVE_ONLY and INTERNAL cells

Cells 0 and 1 are **INTERNAL** cells between the digital system logic and the analog system functions. Note that cells 0 and 1 are not associated with pin 10.

Cells 6, 9, and 12 are cells that are observing signals from the system logic, are not associated with system pins, and are described as **INTERNAL** cells.

Cell 19 is an extra cell in the boundary-scan register. It is not observing the system logic or a system pin and is described as an **INTERNAL** cell.

The definition of the boundary-scan register for Figure B.8 is as follows:

```
attribute BOUNDARY_LENGTH of Figure_B8: entity is 20;
attribute BOUNDARY_REGISTER of Figure_B8: entity is
--
--num cell        port           function         safe [ccell disval rslt]
--
" 0     (BC_1, *,                internal,            0),                        "&
" 1     (BC_1, *,                internal,            1),                        "&
" 2     (BC_0, BIDIR_1,          observe_only, X),                               "&
```

```
" 3     (BC_1, BIDIR_1,           input,               X),                          "&
" 4     (BC_1, BIDIR_1,           output3,             0,    5,       0,       Z), "&
" 5     (BC_1, *,                 control,             0),                          "&
" 6     (BC_0, *,                 internal,            X),                          "&
" 7     (BC_0, OUTPUT_2, observe_only, X),                                          "&
" 8     (BC_1, OUTPUT_2, output2,                      1),                          "&
" 9     (BC_0, *,                 internal,            X),                          "&
"10     (BC_1, OUTPUT_1, output3,                      0,    11,      0,       Z), "&
"11     (BC_1, *,                 control,             0),                          "&
"12     (BC_0, *,                 internal,            X),                          "&
"13     (BC_1, INPUT_1,           input,               X),                          "&
"14     (BC_1, INPUT_1,           input,               X),                          "&
"15     (BC_0, INPUT_1,           observe_only, X),                                 "&
"16     (BC_1, INPUT_2,           input,               X),                          "&
"17     (BC_0, INPUT_2,           observe_only, X),                                 "&
"18     (BC_0, INPUT_3,           observe_only, X),                                 "&
"19     (BC_0, *,                 internal,            X)                           ";
```

B.11.1.3 Merged cells

The component shown in Figure B.9 will be used to illustrate a boundary-scan register description in which special cases are handled. These special cases arise because this standard allows boundary-scan register cells to be merged when the system logic between them is null (see, for example, Figure 11-43 and Figure 11-44). Cells may be merged if the “logic” between them is simply a data path, such as a wire or a buffer. When the merging is done, the resulting cell must obey a combination of the rules of the merged cells.

The definition of the boundary-scan register for Figure B.9 is as follows:

```
attribute BOUNDARY_LENGTH of Figure_B9: entity is 10;
attribute BOUNDARY_REGISTER of Figure_B9: entity is
--
--num cell port                function         safe [ccell disval rslt]
--
" 0     (BC_1, *,                 control,             0),                          "&
" 1     (BC_1, OUT2,              output2,             1,    1,       1,       Weak1),"&
" 2     (BC_7, BIDIR1,            bidir,               X,    3,       0,       Z), "&
" 3     (BC_2, *,                 control,             0),                          "&
" 4     (BC_1, *,                 control,             0),                          "&
" 5     (BC_1, BIDIR3,            input,               X),                          "&
" 5     (BC_1, BIDIR2,            output3,             X,    7,       1,       Z), "&
" 6     (BC_1, BIDIR2,            input,               X),                          "&
" 6     (BC_1, BIDIR3,            output3,             X,    4,       0,       Z), "&
" 7     (BC_1, *,                 control,             1),                          "&
" 8     (BC_1, IN2,               input,               X),                          "&
" 9     (BC_1, IN1,               input,               X),                          "&
" 9     (BC_1, OUT1,              output3,             X,    0,       0,       Z)   ";
```

Cells 0, 4, and 7 are control cells located between the system logic and the enable for signals `OUT1`, `BIDIR2`, and `BIDIR3`. Notice that values are assigned to the “safe” subfields for these cells to cause the associated drivers to disable.

Cell 3 is the control for the reversible cell (see Figure 11-37c) used on the bidirectional signal `BIDIR1`. Notice that its “safe” subfield is given the value that causes `BIDIR1` to be an input.

Figure B.9—A component that illustrates several merged cells

Cell 1 is a 2-state output data cell. Note that it has the three extra fields, indicating that it controls its own open-collector, asymmetrical driver. By placing a “1” in cell 1, the driver at `OUT2` can be set to the inactive drive state, in which it will output the “ **WEAK1** ” state.

Cell 2 is the reversible cell of Figure 11-37d. This cell serves either as an input (if the control cell has turned off the output driver, meaning cell 3 produces a “0”) or as the data source for the driver (if the output is enabled).

Note that the structures for `BIDIR2` and `BIDIR3` (see Figure 11-36) would allow observation of the driver, thus allowing a simple consistency check.

Cell 5 (and similarly cells 6 and 9) has merged behavior—it serves as the input receiver for `BIDIR3` and as the data source for `BIDIR2` . As a result, the cell has two lines of description in the boundary-scan register definition. The first gives its behavior as an input cell, while the second describes its characteristics as an output cell. Note that cell **BC_1** used in this capacity must support both **INPUT** and **OUTPUT3** functions. This is reflected in the definition of **BC_1**, where both functions exist for all instructions.

The example illustrated by Figure B.9 is extreme and includes several unusual cases. Most component implementations will be quite simple and routine, as illustrated by the example component description in Clause B.12.

B.12 A typical application of BSDL

The following example is for the Texas Instruments SN74BCT8374 Octal D Flip-Flop (see Figure B.10) using version STD_1149_1_2001.

Figure B.10—Texas Instruments SN74BCT8374

```
entity ttl74bct8374 is

   generic (PHYSICAL_PIN_MAP : string := "DW");

   port (CLK:in bit; Q:out bit_vector(1 to 8); D:in bit_vector(1 to 8);
   GND, VCC:linkage bit; OC_NEG:in bit; TDO:out bit;
   TMS, TDI, TCK:in bit);

   --Get IEEE Std 1149.1-2001 attributes and definitions
   use STD_1149_1_2001.all;
```

```
attribute COMPONENT_CONFORMANCE of ttl74bct8374: entity is
     "STD_1149_1_2001";

attribute PIN_MAP of ttl74bct8374: entity is PHYSICAL_PIN_MAP;

constant DW:           PIN_MAP_STRING:="CLK:1, Q:(2,3,4,5,7,8,9,10), " &
            "D:(23,22,21,20,19,17,16,15)," &
            "GND:6, VCC:18, OC_NEG:24, TDO:11, TMS:12, TCK:13, TDI:14";

constant FK:           PIN_MAP_STRING:="CLK:9, Q:(10,11,12,13,16,17,18,19)," &
            "D:(6,5,4,3,2,27,26,25)," &
            "GND:14, VCC:28, OC_NEG:7, TDO:20, TMS:21, TCK:23, TDI:24";

   attribute TAP_SCAN_IN                of TDI : signal is true;
   attribute TAP_SCAN_MODE              of TMS : signal is true;
   attribute TAP_SCAN_OUT               of TDO : signal is true;
   attribute TAP_SCAN_CLOCK of TCK : signal is (20.0e6, BOTH);

   attribute INSTRUCTION_LENGTH of ttl74bct8374 : entity is 8;

   attribute INSTRUCTION_OPCODE of ttl74bct8374 : entity is
        "BYPASS (11111111, 10001000, 00000101, 10000100, 00000001)," &
        "EXTEST (00000000, 10000000)," &
        "SAMPLE (00000010, 10000010)," &
        "PRELOAD(00000010, 10000010),"&
        "INTEST (00000011, 10000011)," &
        "HIGHZ      (00000110, 10000110)," &
        "CLAMP      (00000111, 10000111)," &
        "RUNT       (00001001, 10001001)," &                    -- Boundary run test
        "READBN (00001010, 10001010)," &                        -- Boundary read normal
        "READBT (00001011, 10001011)," &                        -- Boundary read test
        "CELLTST(00001100, 10001100)," &                        -- Boundary self-test normal
        "TOPHIP (00001101, 10001101)," &                        -- Boundary toggle-out test
        "SCANCN (00001110, 10001110)," &                        -- BCR scan normal
        "SCANCT (00001111, 10001111)";                          -- BCR scan test

   attribute INSTRUCTION_CAPTURE of ttl74bct8374 : entity is
        "10000001";

   attribute REGISTER_ACCESS of ttl74bct8374 : entity is
        "BOUNDARY (READBN, READBT, CELLTST)," &
        "BYPASS (TOPHIP, RUNT)," &
        "BCR[2] (SCANCN, SCANCT)";                  -- 2-bit boundary control register

   attribute BOUNDARY_LENGTH of ttl74bct8374 : entity is 18;

   attribute BOUNDARY_REGISTER of ttl74bct8374 : entity is
      -- num cell         port           function         safe [ccell disval rslt]
         "17 (BC_1, CLK,                 input,                X)," &
         "16 (BC_1, OC_NEG,              input,                X), " &-- Merged input/
         "16 (BC_1, *,                   control,              1), " &--                 control
         "15 (BC_1, D(1),                input,                X)," &
         "14 (BC_1, D(2),                input,                X)," &
         "13 (BC_1, D(3),                input,                X)," &
         "12 (BC_1, D(4),                input,                X)," &
```

```
            "11 (BC_1, D(5),                input,              X)," &
            "10 (BC_1, D(6),                input,              X)," &
            " 9 (BC_1, D(7),                input,              X)," &
            " 8 (BC_1, D(8),                input,              X)," &
                                    -- cell 16 @ 1 -> Hi-Z
            " 7 (BC_1, Q(1),                output3,            X,    16,      1,       Z)," &
            " 6 (BC_1, Q(2),                output3,            X,    16,      1,       Z)," &
            " 5 (BC_1, Q(3),                output3,            X,    16,      1,       Z)," &
            " 4 (BC_1, Q(4),                output3,            X,    16,      1,       Z)," &
            " 3 (BC_1, Q(5),                output3,            X,    16,      1,       Z)," &
            " 2 (BC_1, Q(6),                output3,            X,    16,      1,       Z)," &
            " 1 (BC_1, Q(7),                output3,            X,    16,      1,       Z)," &
            " 0 (BC_1, Q(8),                output3,            X,    16,      1,       Z)";

end ttl74bct8374;
```

B.13 The 1990 version of BSDL

The 1990 version of BSDL is a de facto industry standard. The information presented in this clause is provided as a courtesy to tool implementers who wish to support BSDL descriptions written before this 2001 version was defined. This form of BSDL is a subset of the 2001 version except where noted below. New BSDL descriptions should not use the 1990 form.

In the 1990 version of BSDL, the following syntactic elements are not supported:

<component conformance statement>
<grouped port identification>
<compliance enable description>
<runbist description>
<intest description>
<BSDL extensions>

In the 1990 version, **KEEPER** and **PRELOAD** were not recognized as reserved words. Also, in the 1990 version, cell types **BC_0** , **BC_7** , **BC_8** , **BC_9** , and **BC_10** need to be specified in a user-supplied VHDL package.

B.13.1 The 1990 Standard VHDL Package STD_1149_1_1990

The following shall be the complete content of Standard VHDL Package STD_1149_1_1990. Note that both the VHDL package and the VHDL package body are shown. This information defines the basis of BSDL and typically would be write-protected by a system administrator. An explanation of the cell definitions (e.g., **BC_1** , **BC_2** , etc.) in the package body is given in B.10.2. BSDL files that use <standard VHDL package identifier> **STD_1149_1_1990** shall be processed using this Standard VHDL Package.

NOTES

1—The figure references are to the 1990 edition of IEEE Std 1149.1.

2 —Where figures are cited, the suffix “c” is used to denote a control cell. The suffix “d” denotes a data cell.

```
package STD_1149_1_1990 is

-- Give pin mapping declarations

attribute PIN_MAP : string;
```

```
subtype PIN_MAP_STRING is string;

-- Give TAP control declarations

type CLOCK_LEVEL is (LOW, BOTH);
type CLOCK_INFO           is record
   FREQ : real;
   LEVEL: CLOCK_LEVEL;
end record;

attribute        TAP_SCAN_IN          : boolean;
attribute        TAP_SCAN_OUT         : boolean;
attribute        TAP_SCAN_CLOCK: CLOCK_INFO;
attribute        TAP_SCAN_MODE : boolean;
attribute        TAP_SCAN_RESET: boolean;

-- Give instruction register declarations

attribute        INSTRUCTION_LENGTH : integer;
attribute        INSTRUCTION_OPCODE : string;
attribute        INSTRUCTION_CAPTURE : string;
attribute        INSTRUCTION_DISABLE : string;
attribute        INSTRUCTION_GUARD : string;
attribute        INSTRUCTION_PRIVATE : string;
attribute        INSTRUCTION_USAGE : string;
attribute        INSTRUCTION_SEQUENCE : string;

-- Give ID and USER code declarations

type ID_BITS is ('0', '1', 'x', 'X');
type ID_STRING is array (31 downto 0) of ID_BITS;
attribute IDCODE_REGISTER                :    ID_STRING;
attribute USERCODE_REGISTER:                  ID_STRING;

-- Give register declarations

attribute REGISTER_ACCESS : string;

-- Give boundary cell declarations

type BSCAN_INST is (EXTEST, SAMPLE, INTEST, RUNBIST);
type CELL_TYPE is (INPUT, INTERNAL, CLOCK,
                           CONTROL, CONTROLR, OUTPUT2,
                           OUTPUT3, BIDIR_IN, BIDIR_OUT);
type CAP_DATA is (PI, PO, UPD, CAP, X, ZERO, ONE);
type CELL_DATA is record
   CT : CELL_TYPE;
   I    : BSCAN_INST;
   CD : CAP_DATA;
end record;
type CELL_INFO is array (positive range <>) of CELL_DATA;

-- Boundary cell deferred constants (see package body)
```

```
constant BC_1          : CELL_INFO;
constant BC_2          : CELL_INFO;
constant BC_3          : CELL_INFO;
constant BC_4          : CELL_INFO;
constant BC_5          : CELL_INFO;
constant BC_6          : CELL_INFO;

-- Boundary register declarations

attribute BOUNDARY_CELLS : string;
attribute BOUNDARY_LENGTH : integer;
attribute BOUNDARY_REGISTER : string;

-- Miscellaneous

attribute DESIGN_WARNING : string;

end STD_1149_1_1990;              -- End of IEEE Std 1149.1-1990 Package


package body STD_1149_1_1990 is                     -- Standard boundary cells

-- Description for f10-12, f10-16, f10-18c, f10-18d, f10-21c

constant BC_1 : CELL_INFO :=
  ((INPUT,         EXTEST,       PI),     (OUTPUT2,        EXTEST,       PI),
    (INPUT,         SAMPLE,       PI),     (OUTPUT2,        SAMPLE,       PI),
    (INPUT,         INTEST,       PI),     (OUTPUT2,        INTEST,       PI),
    (INPUT,         RUNBIST, PI),          (OUTPUT2,        RUNBIST, PI),
    (OUTPUT3, EXTEST,             PI),     (INTERNAL, EXTEST,             PI),
    (OUTPUT3, SAMPLE,             PI),     (INTERNAL, SAMPLE,             PI),
    (OUTPUT3, INTEST,             PI),     (INTERNAL, INTEST,             PI),
    (OUTPUT3, RUNBIST, PI),                (INTERNAL, RUNBIST, PI),
    (CONTROL, EXTEST,             PI),     (CONTROLR, EXTEST,             PI),
    (CONTROL, SAMPLE,             PI),     (CONTROLR, SAMPLE,             PI),
    (CONTROL, INTEST,             PI),     (CONTROLR, INTEST,             PI),
    (CONTROL, RUNBIST, PI),                (CONTROLR, RUNBIST, PI) );

-- Description for f10-8, f10-17, f10-19c, f10-19d, f10-22c

constant BC_2 : CELL_INFO :=
  ((INPUT,         EXTEST,       PI),     (OUTPUT2, EXTEST,              UPD),
    (INPUT,         SAMPLE,       PI),     (OUTPUT2, SAMPLE,              PI),
    (INPUT,         INTEST,       UPD),      -- Intest on output2 not supported
    (INPUT,         RUNBIST, UPD), (OUTPUT2, RUNBIST,                     UPD),
    (OUTPUT3, EXTEST,             UPD), (INTERNAL, EXTEST,               PI),
    (OUTPUT3, SAMPLE,             PI),     (INTERNAL, SAMPLE,             PI),
    (OUTPUT3, INTEST,             PI),     (INTERNAL, INTEST,             UPD),
    (OUTPUT3, RUNBIST, PI),                (INTERNAL, RUNBIST, UPD),
    (CONTROL, EXTEST,             UPD), (CONTROLR, EXTEST,               UPD),
    (CONTROL, SAMPLE,             PI),     (CONTROLR, SAMPLE,             PI),
    (CONTROL, INTEST,             PI),     (CONTROLR, INTEST,             PI),
    (CONTROL, RUNBIST, PI),                (CONTROLR, RUNBIST, PI) );
```

```
-- Description for f10-9

constant BC_3 : CELL_INFO :=
  ((INPUT, EXTEST,            PI),        (INTERNAL, EXTEST,             PI),
    (INPUT, SAMPLE,           PI),        (INTERNAL, SAMPLE,             PI),
    (INPUT, INTEST,           PI),        (INTERNAL, INTEST,             PI),
    (INPUT, RUNBIST, PI),                 (INTERNAL, RUNBIST, PI) );

-- Description for f10-10, f10-11

constant BC_4 : CELL_INFO :=
  ((INPUT, EXTEST,            PI),     -- Intest on input not supported
    (INPUT, SAMPLE,           PI),     -- Runbist on input not supported
    (CLOCK, EXTEST,           PI),       (INTERNAL, EXTEST,             PI),
    (CLOCK, SAMPLE,           PI),       (INTERNAL, SAMPLE,             PI),
    (CLOCK, INTEST,           PI),       (INTERNAL, INTEST,             PI),
    (CLOCK, RUNBIST, PI),                (INTERNAL, RUNBIST, PI) );

-- Description for f10-20c, a combined input/control

constant BC_5 : CELL_INFO :=
  ((INPUT, EXTEST,            PI),       (CONTROL, EXTEST,            PI),
    (INPUT, SAMPLE,           PI),       (CONTROL, SAMPLE,            PI),
    (INPUT, INTEST,           UPD),      (CONTROL, INTEST,            UPD),
    (INPUT, RUNBIST, PI),                (CONTROL, RUNBIST, PI) );

-- Description for f10-22d, a reversible cell

constant BC_6 : CELL_INFO :=
  ((BIDIR_IN, EXTEST,             PI),     (BIDIR_OUT, EXTEST,             UPD),
    (BIDIR_IN, SAMPLE,            PI),     (BIDIR_OUT, SAMPLE,             PI),
    (BIDIR_IN, INTEST,            UPD), (BIDIR_OUT, INTEST,                PI),
    (BIDIR_IN, RUNBIST, UPD), (BIDIR_OUT, RUNBIST, PI) );

end STD_1149_1_1990;              -- End of 1990 Package Body
```

B.13.2 A typical application of BSDL, 1990 version

The following example is for the Texas Instruments SN74BCT8374 Octal D Flip-Flop using version STD_1149_1_1990.

```
entity ttl74bct8374 is
   generic (PHYSICAL_PIN_MAP : string := "DW");

   port (CLK:in bit; Q:out bit_vector(1 to 8); D:in bit_vector(1 to 8);
      GND, VCC:linkage bit; OC_NEG:in bit; TDO:out bit;
      TMS, TDI, TCK:in bit);

   --Get IEEE Std 1149.1-1990 attributes and definitions
   use STD_1149_1_1990.all;

   attribute PIN_MAP of ttl74bct8374 : entity is PHYSICAL_PIN_MAP;

   constant DW:           PIN_MAP_STRING:="CLK:1, Q:(2,3,4,5,7,8,9,10), " &
               "D:(23,22,21,20,19,17,16,15)," &
```

```
              "GND:6, VCC:18, OC_NEG:24, TDO:11, TMS:12, TCK:13, TDI:14";

   constant FK:           PIN_MAP_STRING:="CLK:9, Q:(10,11,12,13,16,17,18,19)," &
              "D:(6,5,4,3,2,27,26,25)," &
              "GND:14, VCC:28, OC_NEG:7, TDO:20, TMS:21, TCK:23, TDI:24";

      attribute TAP_SCAN_IN                 of TDI : signal is true;
      attribute TAP_SCAN_MODE               of TMS : signal is true;
      attribute TAP_SCAN_OUT                of TDO : signal is true;
      attribute TAP_SCAN_CLOCK of TCK : signal is (20.0e6, BOTH);

      attribute INSTRUCTION_LENGTH of ttl74bct8374 : entity is 8;

      attribute INSTRUCTION_OPCODE of ttl74bct8374 : entity is
           "BYPASS (11111111, 10001000, 00000101, 10000100, 00000001)," &
           "EXTEST (00000000, 10000000)," &
           "SAMPLE (00000010, 10000010)," &
           "INTEST (00000011, 10000011)," &
           "TRIBYP (00000110, 10000110)," &                          -- Boundary Hi-Z
           "SETBYP (00000111, 10000111)," &                          -- Boundary 1/0
           "RUNT       (00001001, 10001001)," &                      -- Boundary run test
           "READBN (00001010, 10001010)," &                          -- Boundary read normal
           "READBT (00001011, 10001011)," &                          -- Boundary read test
           "CELLTST(00001100, 10001100)," &                          -- Boundary self-test normal
           "TOPHIP (00001101, 10001101)," &                          -- Boundary toggle-out test
           "SCANCN (00001110, 10001110)," &                          -- BCR Scan normal
           "SCANCT (00001111, 10001111)";                            -- BCR Scan test

     attribute INSTRUCTION_CAPTURE of ttl74bct8374 : entity is "10000001";
      attribute INSTRUCTION_DISABLE of ttl74bct8374 : entity is "TRIBYP";
                                                                    -- Now obsolete, see note below.
      attribute INSTRUCTION_GUARD                  of ttl74bct8374 : entity is "SETBYP";
                                                                    -- Now obsolete, see note below.
--NOTE-The INSTRUCTION_SEQUENCE and INSTRUCTION_USAGE attributes would
--appear here, but they are now obsolete.

      attribute REGISTER_ACCESS of ttl74bct8374 : entity is
           "BOUNDARY (READBN, READBT, CELLTST)," &
           "BYPASS (TOPHIP, SETBYP, RUNT, TRIBYP)," &
           "BCR[2] (SCANCN, SCANCT)";                  -- 2-bit Boundary Control Register

      attribute BOUNDARY_CELLS of ttl74bct8374 : entity is "BC_1";
                                                                    -- Now obsolete, see note below.
      attribute BOUNDARY_LENGTH of ttl74bct8374 : entity is 18;

--NOTE-The "attribute INSTRUCTION_DISABLE of ttl74bct8374:entity is
--"TRIBYP";" statement is the same as the HIGHZ instruction defined in
--this standard and the "attribute INSTRUCTION_GUARD of ttl74bct8374:
--entity is "SETBYP";" statement is the same as CLAMP instruction. The
--"attribute BOUNDARY_CELLS of ttl74bct8374:entity is "BC_1";" statement
--has been removed from later versions of BSDL, since the cells being
--used can be identified while processing the BOUNDARY_REGISTER
--attribute.
```

```
attribute BOUNDARY_REGISTER of ttl74bct8374 : entity is
   -- num cell         port           function         safe [ccell disval rslt]
      "17 (BC_1, CLK,                 input,               X), " &
      "16 (BC_1, OC_NEG,              input,               X), " &-- Merged input/
      "16 (BC_1, *,                   control,             1), " &--                 control
      "15 (BC_1, D(1),                input,               X), " &
      "14 (BC_1, D(2),                input,               X), " &
      "13 (BC_1, D(3),                input,               X), " &
      "12 (BC_1, D(4),                input,               X), " &
      "11 (BC_1, D(5),                input,               X), " &
      "10 (BC_1, D(6),                input,               X), " &
      " 9 (BC_1, D(7),                input,               X), " &
      " 8 (BC_1, D(8),                input,               X), " &
                              -- cell 16 @ 1 -> Hi-Z
      " 7 (BC_1, Q(1),                output3,             X,    16,      1,       Z)," &
      " 6 (BC_1, Q(2),                output3,             X,    16,      1,       Z)," &
      " 5 (BC_1, Q(3),                output3,             X,    16,      1,       Z)," &
      " 4 (BC_1, Q(4),                output3,             X,    16,      1,       Z)," &
      " 3 (BC_1, Q(5),                output3,             X,    16,      1,       Z)," &
      " 2 (BC_1, Q(6),                output3,             X,    16,      1,       Z)," &
      " 1 (BC_1, Q(7),                output3,             X,    16,      1,       Z)," &
      " 0 (BC_1, Q(8),                output3,             X,    16,      1,       Z)";

end ttl74bct8374;
```

B.13.3 Obsolete syntax

Several attributes were enumerated in the 1990 version of BSDL that were all string-valued. They shall be obsolete in this 2001 version (and the earlier 1994 version) of BSDL. The **INSTRUCTION_GUARD** and **INSTRUCTION_DISABLE** attributes were made obsolete by the standardization of the *CLAMP* and *HIGHZ* instructions. The **BOUNDARY_CELLS** attribute was found to be unnecessary. With regard to the **INSTRUCTION_SEQUENCE** and **INSTRUCTION_USAGE** attributes, agreement was never reached on either their scope or the applications they were intended to handle. These two attributes have been dropped.

All these obsolete attributes had string values, and so a tool intended to ignore statements defining the attributes can be designed to ignore string-valued attributes with the obsolete names. Their syntax shall be provided in B.13.3.1 to assist users in the development of tools that can read both the version of BSDL documented in this annex and the earlier draft version of the language.

B.13.3.1 Syntax

attribute <obsoleted attribute name> **of** <component name> **: entity is** <obsoleted string> **;**

<obsoleted attribute name>::= **INSTRUCTION_GUARD** | **INSTRUCTION_DISABLE** | **INSTRUCTION_SEQUENCE** | **INSTRUCTION_USAGE** | **BOUNDARY_CELLS**

<obsoleted string>::= <string>

B.13.4 Miscellaneous points on the 1990 version

In parsing a 1990 version of BSDL, a violation of the condition described in semantic check B.8.14.4 p) shall result in the issuance of a warning message by parsing software. This acknowledges that the 1990 edition of this standard allowed a control cell to fan out to more than one driver, and each driver could be enabled with an independent choice of control value. Semantic check B.8.14.4 p) reflects the strengthening

of this standard to require specifically that all drivers controlled by a single control cell shall be disabled by the same value.

The VHDL package body shown in B.13.1 has a fourth value of <capture instruction>, with a value of RUNBIST shown in the <capture descriptor> elements (see B.10.2). This value has been removed in this 2001 version of BSDL (and the earlier 1994 version), since this version now provides for RUNBIST description (see B.8.15).

In the 1990 version of BSDL, the device ID register was named **IDCODE** in the **REGISTER_ACCESS** attribute. In this annex, this name has been changed to **DEVICE_ID** to match the definition used elsewhere in this standard.

In the 1990 version of BSDL, the following values of given BSDL syntactical elements did not exist:

— <cell context> value **OBSERVE_ONLY**
— <disable result> value **KEEPER**
— <function> value **OBSERVE_ONLY**
— <instruction name> value **PRELOAD**

B.14 The 1994 version of BSDL

The 1994 version of BSDL was defined by IEEE Std 1149.1b-1994. The information presented in this clause is provided to help tool implementers support BSDL descriptions written before this 2001 version was defined. This form of BSDL is a subset of the 2001 version except where noted below. New BSDL descriptions should not use the 1994 form.

In the 1994 version, **KEEPER** and **PRELOAD** were not recognized as a reserved words. Also, in the 1994 version, cell types **BC_8**, **BC_9**, and **BC_10** need to be specified in a user-supplied VHDL package.

B.14.1 The Standard VHDL Package STD_1149_1_1994

The following shall be the complete content of Standard VHDL Package STD_1149_1_1994. Note that both the VHDL package and the VHDL package body are shown. This information shall define the basis of BSDL and typically would be write-protected by a system administrator. An explanation of the cell definitions (e.g., **BC_1**, **BC_2**, etc.) in the package body is given in B.10.2. BSDL descriptions that use <standard VHDL package identifier> **STD_1149_1_1994** shall be processed using this Standard VHDL Package.

NOTE—The figure references are to the 1993 edition of IEEE Std 1149.1.

```
package STD_1149_1_1994 is

-- Give component conformance declaration

attribute COMPONENT_CONFORMANCE : string;

-- Give pin mapping declarations

attribute PIN_MAP : string;
subtype PIN_MAP_STRING is string;

-- Give TAP control declarations
```

```
type CLOCK_LEVEL is (LOW, BOTH);
type CLOCK_INFO           is record
   FREQ : real;
   LEVEL: CLOCK_LEVEL;
end record;

attribute        TAP_SCAN_IN           : boolean;
attribute        TAP_SCAN_OUT          : boolean;
attribute        TAP_SCAN_CLOCK: CLOCK_INFO;
attribute        TAP_SCAN_MODE : boolean;
attribute        TAP_SCAN_RESET: boolean;

-- Give instruction register declarations

attribute        INSTRUCTION_LENGTH : integer;
attribute        INSTRUCTION_OPCODE : string;
attribute        INSTRUCTION_CAPTURE : string;
attribute        INSTRUCTION_PRIVATE : string;

-- Give ID and USER code declarations

type ID_BITS is ('0', '1', 'x', 'X');
type ID_STRING is array (31 downto 0) of ID_BITS;
attribute IDCODE_REGISTER                :    ID_STRING;
attribute USERCODE_REGISTER:                  ID_STRING;

-- Give register declarations

attribute REGISTER_ACCESS : string;

-- Give boundary cell declarations

type BSCAN_INST is (EXTEST, SAMPLE, INTEST);
type CELL_TYPE is (INPUT, INTERNAL, CLOCK, OBSERVE_ONLY,
                           CONTROL, CONTROLR, OUTPUT2,
                           OUTPUT3, BIDIR_IN, BIDIR_OUT);
type CAP_DATA is (PI, PO, UPD, CAP, X, ZERO, ONE);
type CELL_DATA is record
   CT : CELL_TYPE;
   I    : BSCAN_INST;
   CD : CAP_DATA;
end record;
type CELL_INFO is array (positive range <>) of CELL_DATA;

-- Boundary cell deferred constants (see package body)

constant BC_0          : CELL_INFO;
constant BC_1          : CELL_INFO;
constant BC_2          : CELL_INFO;
constant BC_3          : CELL_INFO;
constant BC_4          : CELL_INFO;
constant BC_5          : CELL_INFO;
constant BC_6          : CELL_INFO;
constant BC_7          : CELL_INFO;
```

```
-- Boundary register declarations

attribute BOUNDARY_LENGTH : integer;
attribute BOUNDARY_REGISTER : string;

-- Miscellaneous

attribute PORT_GROUPING : string;
attribute RUNBIST_EXECUTION : string;
attribute INTEST_EXECUTION : string;
subtype BSDL_EXTENSION is string;
attribute COMPLIANCE_PATTERNS : string;
attribute DESIGN_WARNING : string;

end STD_1149_1_1994;              -- End of 1149.1-1994 Package

package body STD_1149_1_1994 is                     -- Standard boundary cells

-- Generic cell capturing minimum allowed data

constant BC_0 : CELL_INFO :=

  ((INPUT,         EXTEST,       PI),           (OUTPUT2,        EXTEST,       X),
    (INPUT,        SAMPLE,       PI),           (OUTPUT2,        SAMPLE,       PI),
    (INPUT,        INTEST,       X),            (OUTPUT2,        INTEST,       PI),
    (OUTPUT3, EXTEST,            X),            (INTERNAL, EXTEST,             X),
    (OUTPUT3, SAMPLE,            PI),           (INTERNAL, SAMPLE,             X),
    (OUTPUT3, INTEST,            PI),           (INTERNAL, INTEST,             X),
    (CONTROL, EXTEST,            X),            (CONTROLR, EXTEST,             X),
    (CONTROL, SAMPLE,            PI),           (CONTROLR, SAMPLE,             PI),
    (CONTROL, INTEST,            PI),           (CONTROLR, INTEST,             PI),
    (BIDIR_IN,EXTEST,            PI),           (BIDIR_OUT, EXTEST, X ),
    (BIDIR_IN,SAMPLE,            PI),           (BIDIR_OUT, SAMPLE, PI),
    (BIDIR_IN,INTEST,            X ),           (BIDIR_OUT, INTEST, PI),
    (OBSERVE_ONLY, SAMPLE, PI),                 (OBSERVE_ONLY, EXTEST, PI) );

-- Description for f10-18, f10-29, f10-31c, f10-31d, f10-33c, f10-41d

constant BC_1 : CELL_INFO :=
  ((INPUT,         EXTEST,       PI),     (OUTPUT2,        EXTEST,       PI),
    (INPUT,        SAMPLE,       PI),     (OUTPUT2,        SAMPLE,       PI),
    (INPUT,        INTEST,       PI),     (OUTPUT2,        INTEST,       PI),
    (OUTPUT3, EXTEST,            PI),     (INTERNAL, EXTEST,             PI),
    (OUTPUT3, SAMPLE,            PI),     (INTERNAL, SAMPLE,             PI),
    (OUTPUT3, INTEST,            PI),     (INTERNAL, INTEST,             PI),
    (CONTROL, EXTEST,            PI),     (CONTROLR, EXTEST,             PI),
    (CONTROL, SAMPLE,            PI),     (CONTROLR, SAMPLE,             PI),
    (CONTROL, INTEST,            PI),     (CONTROLR, INTEST,             PI) );

-- Description for f10-14, f10-30, f10-32c, f10-32d, f10-35c

constant BC_2 : CELL_INFO :=
  ((INPUT,         EXTEST,       PI),     (OUTPUT2, EXTEST,              UPD),
    (INPUT,        SAMPLE,       PI),     (OUTPUT2, SAMPLE,              PI),
```

```
   (INPUT,         INTEST,       UPD),      -- Intest on output2 not supported
   (OUTPUT3, EXTEST,             UPD), (INTERNAL, EXTEST,                PI),
   (OUTPUT3, SAMPLE,             PI),     (INTERNAL, SAMPLE,             PI),
   (OUTPUT3, INTEST,             PI),     (INTERNAL, INTEST,             UPD),
   (CONTROL, EXTEST,             UPD), (CONTROLR, EXTEST,                UPD),
   (CONTROL, SAMPLE,             PI),     (CONTROLR, SAMPLE,             PI),
   (CONTROL, INTEST,             PI),     (CONTROLR, INTEST,             PI) );

-- Description for f10-15

constant BC_3 : CELL_INFO :=
  ((INPUT, EXTEST,            PI),        (INTERNAL, EXTEST,             PI),
    (INPUT, SAMPLE,           PI),        (INTERNAL, SAMPLE,             PI),
    (INPUT, INTEST,           PI),        (INTERNAL, INTEST,             PI) );

-- Description for f10-16, f10-17

constant BC_4 : CELL_INFO :=
  ((INPUT, EXTEST,            PI),              -- Intest on input not supported
    (INPUT, SAMPLE,           PI),
    (OBSERVE_ONLY, EXTEST, PI),
    (OBSERVE_ONLY, SAMPLE, PI),                 -- Intest on observe_only not supported
    (CLOCK, EXTEST,           PI),       (INTERNAL, EXTEST,             PI),
    (CLOCK, SAMPLE,           PI),       (INTERNAL, SAMPLE,             PI),
    (CLOCK, INTEST,           PI),       (INTERNAL, INTEST,             PI) );

-- Description for f10-41c, a combined input/control

constant BC_5 : CELL_INFO :=
  ((INPUT, EXTEST,            PI),       (CONTROL, EXTEST,            PI),
    (INPUT, SAMPLE,           PI),       (CONTROL, SAMPLE,            PI),
    (INPUT, INTEST,           UPD),      (CONTROL, INTEST,            UPD) );

-- Description for f10-35d, a reversible cell
-- !! Not recommended; replaced by BC_7 below !!

constant BC_6 : CELL_INFO :=

  ((BIDIR_IN, EXTEST,             PI),     (BIDIR_OUT, EXTEST,             UPD),
    (BIDIR_IN, SAMPLE,            PI),     (BIDIR_OUT, SAMPLE,             PI),
    (BIDIR_IN, INTEST,            UPD), (BIDIR_OUT, INTEST,                PI) );

-- Description for f10-34d, self monitor reversible
-- !! Recommended over cell BC_6 !!

constant BC_7 : CELL_INFO :=
  ((BIDIR_IN, EXTEST,             PI),     (BIDIR_OUT, EXTEST,             PO),
    (BIDIR_IN, SAMPLE,            PI),     (BIDIR_OUT, SAMPLE,             PI),
    (BIDIR_IN, INTEST,            UPD), (BIDIR_OUT, INTEST,                PI) );

end STD_1149_1_1994;              -- End of IEEE Std 1149.1-1994 Package Body
```

B.14.2 Miscellaneous points on 1994 version

In the 1994 version of BSDL, the following values of given BSDL syntactical elements did not exist:

— <disable result> value **KEEPER**

— <instruction name> value **PRELOAD**